HIGHEST

МАЙКЛ ЛУКАС

второе издание

FreeBSD

ПОДРОБНОЕ РУКОВОДСТВО

СИМВОЛ®

Absolute FreeBSD

The Complete Guide to FreeBSD

Second Edition

Michael Lucas

HIGH TECH

FreeBSD

Подробное руководство

Второе издание

Майкл Лукас

Санкт-Петербург — Москва
2009

Серия «High tech»

Майкл Лукас

FreeBSD. Подробное руководство, 2-е издание

Перевод А. Киселева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*П. Щеголев*
Научный редактор	*Ф. Торчинский*
Редакторы	*Ю. Бочина, Т. Темкина*
Корректор	*Л. Минина*
Верстка	*Д. Орлова*

Лукас М.
FreeBSD. Подробное руководство, 2-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 864 с., ил.
ISBN-10: 5-93286-126-6
ISBN-13: 978-5-93286-126-4

«FreeBSD. Подробное руководство» – всеобъемлющее руководство по FreeBSD, мощной, гибкой и бесплатной операционной системе семейства UNIX, выбранной многими предприятиями в качестве серверной платформы. Прочитав книгу, вы сможете использовать FreeBSD для предоставления сетевых сервисов, научитесь управлять системами FreeBSD, поддерживать их и накладывать «заплатки». Руководство охватывает установку системы, работу в сети, вопросы безопасности, производительность системы, тонкую настройку ядра, файловые системы, SMP, проведение обновлений, устранение неполадок и управление программным обеспечением.

Руководство написано одним из активных участников проекта FreeBSD и адресовано администраторам UNIX, у которых назрела потребность в сборке и конфигурировании выделенных серверов FreeBSD. Книга будет также интересна пользователям, планирующим применять FreeBSD на своем рабочем компьютере или комбинировать настольные/серверные системы. Настоящее издание существенно обновлено и дополнено с учетом появления новых версий системы и аппаратных средств.

ISBN-10: 5-93286-126-6
ISBN-13: 978-5-93286-126-4
ISBN-13: 978- 1-59327-151-0 (англ)

© Издательство Символ-Плюс, 2009
Authorized translation of the English edition © 2008 No Starch Press, Inc. This translation is published and sold by permission of No Starch Press, Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 30.09.2008. Формат 70x100$^{1}/_{16}$. Печать офсетная.
Объем 54 печ. л. Тираж 2500 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Посвящается Лиз.
Надеюсь, что мы прекрасно подходим друг другу.

Оглавление

Отзывы на первое издание книги «FreeBSD. Подробное руководство»

«Даже те, кто давно используют FreeBSD, могут быть удивлены мощью и широтой возможностей, которые может предложить FreeBSD как серверная платформа, а книга *Absolute BSD* являет собой превосходный справочник по их использованию».

– UnixReview.com

«...содержит прекрасно подобранные примеры и справочные сведения, которые помогут максимально полно использовать возможности этой операционной системы».

– LinuxUser & Developer Magazine

«...замечательный источник сведений как для тех, кто мало знаком с BSD, так и для тех, кто работает с этой операционной системой на протяжении многих лет. Майкл Лукас пишет простым и понятным языком».

– Freshmeat

«Прекрасная книга. Она написана не о том, как реализовать какие-либо решения на базе BSD, а об управлении системами».

– ;LOGIN:

«...содержит огромный объем информации».

– Daemon News

Отзывы на книгу Майкла Лукаса «Absolute OpenBSD»

«Книга Майкла Лукаса *Absolute OpenBSD* представляет собой полное введение в мир операционной системы OpenBSD. Оно обладает достаточной полнотой и глубиной, чтобы любому, кто мало знаком с OpenBSD, дать надежную опору для выполнения реальной работы и знания для дальнейших исследований... Потенциально скучная тема администрирования операционных систем оказывается понятной и интересной благодаря простому языку и шутливому тону автора».

– Крис Палмер (Chris Palmer), президент группы пользователей OpenBSD, Сан-Франциско

«...хорошо написанная книга, которая попала точно в цель. Те, кто мало знаком с OpenBSD, оценят комплексный подход, который проведет их от концепции к функциональному использованию. Опытные пользователи извлекут пользу из обсуждения таких специфичных для OpenBSD тем, как возможности обеспечения безопасности и администрирование пакетного фильтра (pf)».

– Slashdot

«Я рекомендую книгу *Absolute OpenBSD* всем программистам и администраторам, работающим с операционной системой OpenBSD или рассматривающим такую возможность».

– UnixReview.com

Отзывы на книгу Майкла Лукаса «PGP & GPG»

«Это еще одна замечательная книга Майкла Лукаса. Я получал истинное удовольствие от чтения других его книг благодаря их содержимому и стилю изложения. Книга *PGP & GPG* продолжила эту прекрасную традицию. Если вам необходимо узнать, как правильно использовать PGP и GPG на практике, или вы просто хотите убедиться в том, что вы применяете эти технологии должным образом, прочитайте эту книгу».

– TaoSecurity

«Самая лучшая по своей простоте и доступности книга, посвященная вопросам обеспечения безопасности использования электронной почты. Если вы не знакомы с криптографией или никогда не использовали электронную почту, вам следует прочитать эту книгу».

– Len Sassaman, основатель CodeCon

«Замечательное руководство, легко читается, а легкий юмор делает чтение приятным».

– InfoWorld

«Отличная книга, которая в простой и интересной манере демонстрирует конечному пользователю почти все, что он должен знать для эффективного использования PGP и OpenPGP».

– Slashdot

Отзывы на книгу Майкла Лукаса «Cisco Routers for the Desperate»

«…эта книга не просто справочник – это руководство по выживанию, своего рода ремни безопасности... Должен отметить, что книга, очевидно, была написана для таких, как я – кто не питает особого интереса к теме управления маршрутизаторами, но чья деятельность зависит от их надежной работы».

– asp.netPRO

«Если бы книга *Cisco Routers for the Desperate* оказалась на моей книжной полке на несколько лет раньше! Это позволило бы мне сэкономить массу времени при конфигурировании маршрутизаторов Cisco... Я настоятельно рекомендую эту книгу как ИТ-специалистам, которые приступают к работе с маршрутизаторами Cisco, так и всем тем, кто время от времени должен иметь с ними дело, но у кого нет возможности или технических знаний, чтобы углубиться в чтение более специализированных книг по этой теме».

– Blogcritics Magazine

Предисловие

Мне выдалась приятная возможность написать предисловие к новой книге Майкла Лукаса (Michael Lucas) «FreeBSD. Подробное руководство». Вот уже в течение пяти лет автор создает замечательные пособия по программному обеспечению BSD, которые могут использоваться не только как справочник, но и как учебник. Это очень важно, так как хотя в настоящее время и существует достаточное количество справочников по FreeBSD, но эта книга представляет собой практическое руководство, которое для многих читателей окажется неоценимым подспорьем.

Майкл уже много лет принимает активное участие в жизни сообщества FreeBSD. Эта книга является обобщением разнообразных подходов к использованию этой операционной системы на практике, описывая, какие ставились задачи, что удавалось реализовать, а что – нет. Помимо этого Майкл рассказывает об огромном числе разработчиков программного обеспечения (от любителей до профессиональных программистов и профессоров университетов), участвующих в создании FreeBSD, а также о развитии этого сообщества и программного обеспечения, создаваемого им. Я полагаю, и вы могли бы принять в этом участие.

FreeBSD – это мощная сетевая операционная система с современными техническими характеристиками, что обусловливает не только широкое ее распространение, но и делает ее простым и практичным инструментом для предоставления различных услуг. От веб-серверов Yahoo! и Verio до продуктов NetApp, от устройств Cisco для борьбы со спамом и маршрутизаторов Juniper до корневых серверов имен – в Интернете сложно «кинуть камень» и не попасть во FreeBSD. Однако FreeBSD – это не продукт какой-то одной компании, это результат деятельности крупного свободного сообщества: в проект FreeBSD входят разработчики, пользователи и бесчисленное множество сторонников. Вы можете, как многие, использовать программное обеспечение FreeBSD, не вступая в контакт с сообществом, но, вступив в него, вы сможете существенно обогатить свой опыт.

Являетесь ли вы начинающим пользователем или уже опытным специалистом, ресурсы, доступные на веб-сайте *http://www.freebsd.org*, бесчисленные списки рассылки, региональные группы пользователей и конференции станут для вас неоценимыми источниками информации. Появился вопрос? Просто отправьте письмо на адрес

questions@FreeBSD.org, и один или более из сотен добровольцев непременно ответят вам. Хотите подробнее узнать о новых возможностях будущих версий FreeBSD? Обратитесь к ежеквартальным отчетам проекта, читайте списки рассылки для разработчиков или присоединитесь к какой-либо из многочисленных региональных конференций по BSD – к моменту написания этих строк объявлено, например, о новой конференции BSDConTR в Стамбуле, Турция.

Эти ресурсы – результат деятельности проекта FreeBSD и его сообщества, большого числа отдельных людей и компаний, а также некоммерческой организации FreeBSD Foundation, координирующей финансирование, осуществляющей юридическую поддержку и поддержку разработки и деятельности сообщества. Книга Майкла предоставляет начинающим пользователям возможность приобщиться к опыту сообщества и стать его активными членами.

FreeBSD – это программное обеспечение, распространяемое с открытыми исходными текстами, доступное для бесплатного использования и распространения. Помогая в поддержке или даже разработке, вы можете внести свою лепту в этот проект и способствовать дальнейшему росту сообщества.

Неважно, начинаете ли вы только знакомиться с новой системой или уже являетесь опытным специалистом, я уверен, что вы захотите всегда иметь у себя под рукой книгу «FreeBSD. Подробное руководство».

Роберт Н. М. Уотсон (Robert N. M. Watson),
член основной группы разработчиков FreeBSD,
президент организации FreeBSD Foundation

Кембридж, Великобритания
сентябрь 2007 года

Благодарности

Хочу выразить признательность всем членам сообщества FreeBSD за их нелегкий труд, преданность и дружелюбие. Сообщество спасало меня много раз, и я рад возможности ответить благодарностью. В нем есть несколько человек, заслуживших особую признательность.

Дуг Бартон (Doug Barton), Кери Дэвис (Ceri Davies), Алекс Дюпре (Alex Dupre), Макс Лейер (Max Laier), Александр Лейдингер (Alexander Leidinger), Ремко Лоддер (Remko Lodder), Бенно Рис (Benno Rice), Том Родс (Tom Rhodes), Глеб Смирнов (Gleb Smirnoff) и Роберт Уотсон (Robert Watson) – все они представили ценные замечания по этой книге. Кто-то из них читал отдельные главы, обладая богатым опытом по выбранным темам, другие полностью прочитали всю рукопись, независимо от уровня своих познаний. Уилко Булт (Wilko Bult) не только сделал обзор этой книги, он добровольно согласился на это еще в 2001 году, выполнив обзор всего первого издания. Он определенно заслуживает награды за постоянство! Джон Болдуин (John Baldwin) прекрасно справился с окончательным техническим рецензированием книги, выловив большое число самых разнообразных, малозаметных и явных ошибок. Все ошибки в этой книге допущены мной несмотря на все усилия этих людей.

Я хотел бы поблагодарить Дэвида Бойда (David Boyd), Дэвида О'Брайена (David O'Brien) и Уилко Булта (Wilko Bulte) за предоставленное разнообразное аппаратное обеспечение, которое помогло мне написать эту книгу. Особое спасибо Мэтту Оландеру (Matt Olander) из компании iXSystems, который передал мне сервер на базе процессора amd64, в котором я действительно, *действительно* нуждался. Говоря об аппаратных средствах, когда работа над этой книгой уже подходила к концу, я задавался вопросом – как мне получить ошибку в ядре, чтобы написать о ней в последней главе. Благодаря Скотту Лонгу (Scott Long) мне удалось получить такую ошибку, и я смог написать главу 21.

Как всегда парни из No Starch Press прекрасно потрудились над тем, чтобы эта книга попала к вам. Вы все заслуживаете хорошего отпуска за долготерпение, проявленное ко мне – сообщите, Биллу, что я так сказал. Точно так же длительного отдыха заслуживают сотрудники школы китайских боевых искусств. К сожалению, теперь, когда эта книга закончена, я смогу проводить некоторое время на циновках, так что им не удастся отдохнуть от меня. Извините, ребята.

И как всегда я признателен моей жене Лиз за то, что, пока я заканчивал работу над этой книгой, она не поддалась искушению стукнуть меня по голове и похоронить меня вместе с моим ноутбуком за гаражом. Она проявила необычайное терпение, ожидая, когда я закончу и смогу наконец вынести мусор, который накопился с марта прошлого года...

Майкл Лукас (Michael Lucas)

Сент-Клер Шорз, Мичиган
сентябрь 2007 года

Введение

Добро пожаловать во «FreeBSD. Подробное руководство»! Эта книга предназначена для системных администраторов, у которых назрела необходимость в сборке, настройке и управлении серверов, работающих под управлением FreeBSD. Кроме того, книга будет интересна пользователям, собирающимся запускать эту систему на своих настольных компьютерах, на серверах и на бездисковых рабочих станциях. Прочитав книгу, вы сможете использовать FreeBSD для предоставления сетевых сервисов. Вы научитесь управлять системами FreeBSD, поддерживать их и накладывать «заплатки». Вы получите базовое представление о работе в сети, безопасности системы и управлении программным обеспечением. В книге обсуждается FreeBSD версии 7. На момент выхода данной книги эта версия рекомендована для широкого использования. Большая часть книги также применима для предшествующих и последующих версий.

Что такое FreeBSD?

FreeBSD – это операционная система, подобная UNIX, которая свободно доступна в Интернете. Она широко применяется в компаниях-провайдерах услуг Интернета, во встроенных устройствах и в любом другом месте, где важна надежность. Однажды FreeBSD чудесным образом появилась в Интернете полностью сформированная, порожденная небывалым интеллектом ее создателя. Это шутка, правда выглядит намного внушительнее. Операционная система FreeBSD – это результат непрерывного, в течение более тридцати лет, процесса разработки, исследований и доводки. История началась с проекта BSD, в 1979 году.

BSD – бабушка FreeBSD

Много лет назад компании AT&T потребовалось собственное, специализированное программное обеспечение для ведения бизнеса. Однако она не имела права вторгаться в компьютерную индустрию и поэтому не могла продавать свое программное обеспечение. В результате AT&T предоставила различные куски программного обеспечения и его исходный код университетам по очень низкой цене. Университеты смогли сэкономить средства, используя это программное обеспечение вместо предлагаемого по коммерческим ценам, а студенты университетов

получили доступ к отличной технологии. Они могли читать исходный код и изучать его работу. Взамен AT&T получила бесплатную площадку для экспериментов и поколение специалистов по вычислительной технике, выросших на оборудовании AT&T. Все были довольны. Система UNIX была самым известным программным обеспечением, распространяемым по этому плану лицензирования.

Исходная система UNIX обладала массой недостатков в сравнении с современными операционными системами. Однако тысячи студентов имели доступ к исходному коду, а сотням преподавателей требовались интересные проекты для их студентов. Когда программы вели себя не так, как ожидалось, или в самой операционной системе обнаруживались какие-либо ошибки, то все, кто работал в системе день за днем, имели возможность и стимул исправить эти недостатки. Благодаря их усилиям система UNIX была вскоре улучшена и в ней появились многие возможности, которые сейчас мы воспринимаем как нечто само собой разумеющееся. Студенты добавили возможность управлять запуском программ (*управление заданиями*, *job control*). Файловая система UNIX S51K доводила до слез системных администраторов, поэтому они заменили ее файловой системой Fast File System, возможности которой перекочевали во все современные файловые системы. За многие годы было написано большое число полезных программ, постепенно заменивших целые блоки операционной системы UNIX.

Группа по исследованию компьютерных систем (Computer Systems Research Group, CSRG) Калифорнийского университета, участвовавшая в этих усовершенствованиях, стала центральным хранилищем улучшений кода UNIX. Группа CSRG собирала изменения, оценивала их, упаковывала и бесплатно передавала сборки всем обладателям действительной лицензии AT&T UNIX. Кроме того, CSRG сотрудничала с управлением перспективных исследовательских проектов (Defense Advanced Research Projects Agency, DARPA) с целью реализации в UNIX разнообразных функциональных возможностей, таких как стек протоколов TCP/IP. Итоговая коллекция программного обеспечения получила название Berkeley Software Distribution, или BSD.

Пользователи BSD брали программное обеспечение, улучшали его и затем возвращали свои наработки обратно в BSD. Сегодня мы считаем такой способ стандартным для разработки программного обеспечения с открытыми исходными текстами, но в 1979 году он стал революционным. Разработка была долгой. Посмотрев на информацию об авторских правах старой системы BSD, можно увидеть следующее:

```
Copyright 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
      The Regents of the University of California. All rights reserved.
```

Да, 15 лет работы – это в разработке программного обеспечения целая жизнь. Как много частей операционной системы не только продолжают использоваться, но и активно разрабатываются по прошествии 15 лет? По существу, в оригинальную систему BSD было внесено

столько изменений, что за эти годы произошла почти полная замена оригинального кода UNIX кодом, созданным сотрудниками CSRG и их помощниками. От продукта AT&T осталось совсем мало.

В конце концов финансирование CSRG прекратилось и стало очевидным, что дальнейшее развитие проекта BSD подходит к концу. В 1992 году после некоторых споров в недрах Калифорнийского университета код BSD был открыт широкой публике. Такая передача прав получила название лицензии BSD.

Лицензия BSD

Код BSD стал доступен всем желающим на, пожалуй, самых либеральных условиях в истории разработки программного обеспечения. Сегодня основные положения лицензии выглядят так:

- Не заявляйте о том, что вы написали этот код.
- Не обвиняйте нас за ошибки в коде.
- Не используйте наше имя для продвижения своего продукта.

Это означает, что вы можете делать с исходными текстами BSD все, что угодно. (Первоначальная лицензия BSD требовала упоминать об использовании программного кода, выпущенного на основе лицензии BSD, но позднее это требование было снято.) В ней нет даже требования делиться своими изменениями с авторами первоначального кода! Любой желающий может свободно включать код BSD в патентованные продукты, бесплатные продукты и продукты с открытым исходным кодом. Код BSD можно распечатать на перфокартах и покрыть ими лужайку. Хотите выпустить 10 000 компакт-дисков с операционной системой BSD и подарить их своим друзьям? Пожалуйста. Иногда при обсуждении лицензии BSD упоминается не «copyright» (авторское право), а «copycenter» (копировальный центр) – «возьмите эту лицензию в копировальный центр и напечатайте для себя несколько копий». Неудивительно, что некоторые компании, например Sun Microsystems, так за нее ухватились – она бесплатна, надежна и большое число дипломированных специалистов имеет опыт работы с ней. Специально для использования преимуществ BSD UNIX была даже создана компания BSDi.

Состязание AT&T/CSRG/BSDi

Даже в эпоху расцвета CSRG работа над UNIX в AT&T не прекращалась. AT&T принимала распространяемые куски BSD UNIX, интегрировала со своей системой UNIX, а затем снова передавала получившийся результат университетам, которые вносили эти улучшения. Такой подход прекрасно себя зарекомендовал, пока AT&T не распалась и образовавшимся компаниям не получили разрешение конкурировать на рынке программного обеспечения. У AT&T была одна значительная собственность: высококлассная операционная система, которая была отлажена тысячами специалистов со всего мира. Эта операционная система

обладала множеством полезных функциональных возможностей, таких как небольшие, но достаточно мощные команды, современная файловая система, возможность управления заданиями и стек протоколов TCP/IP. AT&T образовала свой филиал, Unix Systems Labaratories (USL), который стал успешно продавать UNIX предприятиям за большие деньги, поддерживая при этом отношения с университетами, которые и предоставили ей такую усовершенствованную операционную систему.

Публичный «выпуск» программного кода BSD, проведенный Беркли в 1992 году, был с большим неудовольствием встречен компанией USL (UNIX System Laboratories). Почти сразу же ее представители вызвали несколько фирм по производству программного обеспечения (в частности, BSDi) и университет в суд. В ответ Калифорнийский университет утверждал, что группа CSRG собрала систему BSD благодаря вкладу тысяч сторонних разработчиков, не имеющих отношения к AT&T, и поэтому то, чем пыталась распоряжаться AT&T, является интеллектуальной собственностью группы CSRG.

Этот судебный процесс побудил многих получить копию BSD, чтобы самим посмотреть, из-за чего разгорелась шумиха, в то время как другие компании стали создавать на его основе свои программные продукты. Одним из таких продуктов стала система 386BSD, которая со временем образовала ядро FreeBSD 1.0.

В 1994 году после двухлетних разбирательств адвокаты Калифорнийского университета доказали, что значительная часть кода в AT&T UNIX действительно могла быть взята почти наверняка только из BSD, а не наоборот. Хуже того, компания AT&T фактически нарушила условия лицензии BSD, нарушив авторские права группы CSRG на файлы, которые были включены в состав UNIX. (Компания могла нарушить самую либеральную в мире лицензию только преднамеренно!) Источником разногласий были всего полдюжины файлов. Чтобы положить конец этим спорам, компания USL подарила часть этих файлов BSD, запатентовав остальные.

После того как все утихло, была выпущена новая версия BSD UNIX под названием BSD 4.4-Lite. Последующее обновление, система BSD 4.4-Lite2, является бабушкой текущего кода FreeBSD, а также прародительницей многих других операционных систем семейства BSD.

Рождение FreeBSD

Одним из ранних достижений BSD стала система 386BSD – версия, которая могла работать на недорогих процессорах серии 386.[1] В рамках проекта 386BSD система была благополучно перенесена на процессоры Intel 386, но на этом дальнейшее развитие проекта остановилось. По-

[1] В то время компьютер стоимостью несколько тысяч долларов считался очень недорогим. Современная молодежь не ценит то, что ей досталось.

сле периода застоя группа пользователей 386BSD решила самостоятельно продолжить развитие и создать FreeBSD, чтобы иметь возможность продолжать совершенствовать операционную систему. (Одновременно еще несколько групп начали работу над своими проектами, основанными на 386BSD, из которых выжила только система NetBSD.)

Системы 386BSD и FreeBSD растут из выпуска BSD 1992 года, ставшего причиной судебных споров с AT&T. В результате этих разбирательств все пользователи оригинальной версии BSD были вынуждены получать разрешение на производство любых продуктов, основанных на BSD 4.4-Lite2. Система BSD 4.4-Lite2 не была законченной операционной системой, в частности, из-за тех нескольких файлов, оставшихся в интеллектуальной собственности AT&T, которые имели чрезвычайную важность для функционирования системы. (В конце концов, если бы эти файлы не были так важны, AT&T не стала бы и беспокоиться!) Группа разработки FreeBSD приложила отчаянные усилия, чтобы заменить недостающие файлы, и вскоре была выпущена версия FreeBSD 2.0. С тех пор развитие системы продолжается.

Сегодня FreeBSD применяется по всему миру наиболее значительными и заметными компаниями, деятельность которых связана с Интернетом. Компания Yahoo! почти полностью работает на системах FreeBSD. IBM, Nokia, Juniper, NetApp и многие другие производители аппаратных средств применяют ее во встроенных системах, хотя об этом даже трудно догадаться. По существу, если компании необходима серьезная полоса пропускания в Интернете, она наверняка запускает FreeBSD или другую систему из семейства BSD. Машины FreeBSD окружают вас; вы просто не видите их потому, что аварии на них случаются редко. Основой надежности системы является группа разработчиков и сообщество пользователей, которые суть одно и то же.

Разработка FreeBSD

Существует старая поговорка, согласно которой управление программистами подобно выгулу группы котов. Несмотря на тот факт, что разработчики FreeBSD разбросаны по всему миру и говорят на разных языках, они отлично работают вместе в составе единой команды. Они больше напоминают семью львов, чем группу кошек. И, в отличие от некоторых других проектов, вся разработка FreeBSD ведется открыто. Систему разрабатывают три группы: создатели (committers), помощники (contributors) и пользователи (users).

Создатели

Сегодня в состав команды FreeBSD входит почти 500 разработчиков, или *создателей*. Они имеют доступ на чтение/запись к основному репозитарию исходного кода FreeBSD и могут разрабатывать, отлаживать и улучшать любую часть кода по своему усмотрению. (Термин *committer*

произошел от английского *commit* – передавать или вносить изменения в исходный код.) Любые изменения исходного кода могут вывести систему из строя, сделать ее неработоспособной, поэтому на плечи создателей тяжким грузом ложится бремя ответственности за свои действия. Создатели отвечают за сохранение FreeBSD в работоспособном состоянии, они следят, чтобы добавление новых возможностей, по крайней мере, не привело к нарушениям и оценивают «заплаты», получаемые от других создателей. Большинство разработчиков являются добровольцами – лишь незначительная их часть получают оплату за свой усердный труд, большинство же других зарабатывают себе на жизнь, занимаясь другой работой. Например, компания Intel нанимает создателя, чтобы убедиться, что сетевые карты Intel правильно поддерживаются в FreeBSD. Система FreeBSD имеет большой авторитет среди компаний, имеющих отношение к Интернету, поэтому Intel беспокоится о надежной работе своих сетевых карт в этой операционной системе.

Чтобы присоединиться к плеяде разработчиков FreeBSD, достаточно подписаться на почтовую рассылку *FreeBSD-hackers@FreeBSD.org*, содержащую большую часть обсуждений технических вопросов. Технические вопросы разбиты на более специализированные почтовые рассылки – например, разработка сетевых служб обсуждается в *FreeBSD-net@FreeBSD.org*.

Раз в несколько лет команда создателей выбирает несколько человек, которые будут играть роль основной команды, или *Ядра*. Ядро играет важную роль, которая порой недооценивается и неправильно понимается. Ядро теоретически отвечает за управление всем кодом FreeBSD, но на практике занимается решением споров и конфликтов, возникающих между создателями. Кроме того, ядро принимает новых членов в ряды создателей и делегирует ответственность за крупные части FreeBSD отдельным лицам и группам разработчиков. Например, ядро делегирует управление портами и системой пакетов специально созданной команде управления портами. Ядро не определяет архитектурные направления развития FreeBSD и не определяет процедурные вопросы – такого рода решения принимаются общим голосованием создателей. Но как бы то ни было, ядро выдвигает свои предложения, добивается определенных решений и играет вдохновляющую роль.

Помимо этого, ядро принимает на себя всю тяжесть управления. Основные функции в управлении – контроль, стимулирование и разрешение проблем между людьми. Контроль обеспечивают миллионы пользователей, сразу заявляющих, когда что-то не работает или работает не так, как ожидается; у создателей FreeBSD нет проблем со стимуляцией. Самая неприятная часть в управлении – улаживание споров между двумя людьми, и такой работы у Ядра хватает. Слова: «я вхожу в состав Ядра» – недостаточная награда за необходимость улаживать конфликт между двумя талантливыми разработчиками, которые играют друг у друга на нервах.

Помощники

В дополнение к команде создателей сообщество FreeBSD включает в себя тысячи помощников. *Помощники* не связаны с репозитарием операционной системы; они просто представляют заплатки на рассмотрение создателей. Создатели оценивают представленный код и решают, принять его или отклонить. Помощник, который постоянно представляет приемлемый код, зачастую получает от создателей предложение самому стать создателем.

Например, в течение нескольких лет я вносил свой вклад в развитие FreeBSD, когда чувствовал такую необходимость. Всякий раз, когда я чувствую, что зря прожил жизнь, я могу зайти на веб-сайт FreeBSD и увидеть, что моя работа была принята создателями и применяется тысячами пользователей. После того как я передал издателю рукопись первого издания этой книги, я потратил появившееся свободное время на подготовку заплаток для FreeBSD FAQ. В результате участники Проекта Документирования FreeBSD обратились ко мне с предложением стать создателем. В награду я получил адрес электронной почты и возможность выслушивать оскорбления тысяч людей, что еще раз демонстрирует, что хорошее дело не остается безнаказанным.

Если бы я ничего не предлагал, я оставался бы пользователем. И ведь в этом нет ничего плохого!

Пользователи

Пользователи – это те, кто использует систему FreeBSD. Достоверно оценить количество пользователей FreeBSD невозможно, хотя такие организации, как BSDstats Project (*http://www.bsdstats.org*), предпринимают такие попытки. В конце концов, вы можете бесплатно скачать всю систему FreeBSD и при этом никогда не регистрироваться, не обновлять систему и не подписываться на почтовую рассылку. По приблизительным оценкам компаний, таких как Netcraft, от 5 до 15 процентов машин в Интернете работают под управлением BSD-систем. Если из этого числа исключить все компьютеры с операционной системой Windows, расположенные на офисных столах, этот процент вырастет многократно.

Поскольку FreeBSD, безусловно, – наиболее популярная система BSD с открытым кодом, это количество выглядит значительным. А поскольку один сервер FreeBSD может обслуживать сотни и тысячи доменов Интернета, то количество сайтов, задействующих FreeBSD, несоизмеримо с количеством серверов. Это означает, что по всему миру ежедневно сотни тысяч, если не миллионы системных администраторов работают с операционной системой FreeBSD.

Другие системы BSD

FreeBSD – наиболее популярная система BSD, но не единственная. Система BSD 4.4-Lite2 породила различные проекты, у каждого из которых есть свое назначение. Эти проекты в свою очередь дали начало другим проектам, некоторые из которых продолжают развиваться и по сей день.

NetBSD

Система NetBSD во многом подобна FreeBSD, а их команды делят между собой разработчиков и сам программный код. Основное назначение проекта NetBSD – предоставить безопасную и надежную операционную систему, которую можно перенести на любую аппаратную платформу с минимальными усилиями. Так, NetBSD работает на VAX, устройствах PocketPC и высокопроизводительных серверах SPARC и Alpha. Мне удалось запустить NetBSD даже на моем карманном компьютере HP Jornada.[1]

OpenBSD

OpenBSD ответвилась от NetBSD в 1996 году с целью стать самой безопасной BSD. OpenBSD стала первой системой, в которой была реализована поддержка криптоаппаратуры, и ее разработчики с полным основанием гордятся тем, что их первоначальная система удаленно не взламывалась свыше четырех лет. Команда OpenBSD передала миру некоторые ценные разработки, наиболее существенной из которых является пакет OpenSSH, используемый практически всеми операционными системами и производителями аппаратного обеспечения.

Mac OS X

Mac OS X? Все верно. Большие фрагменты FreeBSD были включены в Mac OS X. Если вы ищете стабильную операционную систему с дружелюбным интерфейсом и мощной основой, система Mac OS X вам, бесспорно, подойдет. Хотя FreeBSD предоставляет замечательную настольную среду для профессионалов-компьютерщиков, я не установил бы ее для своей бабушки. Не раздумывая, я установил бы ей Mac OS X и был бы уверен, что поступаю правильно. Впрочем, Mac OS X включает в себя функции, которые вовсе не обязательны для сервера Интернета. Эта система работает только на платформе Apple, поэтому я не рекомендую ее для недорогих серверов общего назначения.

Развитие программного кода идет в двух направлениях. FreeBSD включает в себя код, изначально разрабатывавшийся для Mac OS X. И хотя

[1] Если вам потребуется когда-нибудь доказать, что вы специалист экстракласса, то запуск UNIX на наладонном компьютере в этом вам безусловно поможет.

вам недоступен исходный код пользовательского интерфейса Mac OS X, вы можете изучить базовый код BSD, использованный для этой операционной системы, и ядро Mach. Компания Apple выпустила их под кодовым названием *Darwin*.

Потомки FreeBSD

Несколько проектов взяли за основу систему FreeBSD и на ее основе создали свои продукты. Проект FreeNAS превращает систему на платформе x86 в сетевой файловый сервер с очень простым меню. FreeSBIE – загружаемый компакт-диск, который позволяет запустить FreeBSD, не устанавливая ее. Еще один загружаемый компакт-диск – проект m0n0wall; он превращает систему в межсетевой экран с прекрасным веб-интерфейсом. Проект PC-BSD добавил дружественный интерфейс к FreeBSD, превратив ее в систему, которой могут пользоваться даже бабушки. Время от времени появляются и другие проекты, хотя не все они добиваются успеха. Я уверен, что когда эта книга увидит свет, у нас появятся один или два более или менее солидных члена этой группы.

Другие UNIX

Существует несколько других операционных систем, которые ведут свою родословную от UNIX или подражают ей. Безусловно, список таких систем значителен, однако мы коснемся лишь основных.

Solaris/OpenSolaris

Наиболее известная система UNIX – это Solaris компании Sun Microsystems и ее новый потомок OpenSolaris. Solaris работает на высокопроизводительной аппаратной платформе, которая поддерживает десятки процессоров и *кучу* (*gobs*) дисков. (Да, «gobs» – это технический термин, означающий *очень большое количество дисков, большее, чем вам может когда-нибудь понадобиться, причем я хорошо понимаю, что вам нужно гораздо больше дисков, чем я думаю*.) В операционной системе Solaris, особенно в ранних ее версиях, очень четко прослеживались корни BSD. Solaris применяется многими приложениями уровня предприятия, такими как Oracle. В основном Solaris запускается на аппаратной платформе SPARC, которая выпускается компанией Sun. Поскольку Sun контролирует как аппаратное, так и программное обеспечение, ее системы поддерживают много интересных функций, например возможность «горячей» замены модулей памяти и материнских плат. Однако OpenSolaris больше нацелен на широко распространенные аппаратные платформы.

AIX

Другой конкурент UNIX – система AIX компании IBM. AIX знаменита журналируемой файловой системой, которая регистрирует все дис-

Почему UNIX-подобные?

Следует заметить, что FreeBSD, Linux и аналогичные системы называют UNIX-подобными, а не UNIX. Название UNIX является торговой маркой Open Group. Чтобы операционная система могла получить право называться UNIX, производитель должен доказать, что она соответствует текущей версии Single Unix Specification (единая спецификация UNIX). Система FreeBSD несомненно отвечает требованиям стандартов, но непрерывное тестирование и необходимость повторного прохождения классификации требуют денег, которых нет у проекта FreeBSD. Кроме того, процедура сертификации требует, чтобы кто-то подписал документ, заявляя, что он или она принимает на себя ответственность за соответствие FreeBSD единой спецификации UNIX, и что он или она обязуются устранить любые несоответствия, которые будут обнаружены в будущем. Модель развития FreeBSD предусматривает даже больше – найденные ошибки и отклонения исправляются, но нет никого, кто мог бы подписать документ, гарантирующий 100-процентное соответствие стандартам.

ковые транзакции по мере их выполнения. Она позволяет без особого труда восстановить систему после аварий, обеспечивая высокую надежность. Кроме того, в течение многих лет она была стандартной UNIX-системой Голубого Гиганта. AIX вобрала в себя значительную часть кода BSD.

Linux

Linux – это клон UNIX, написанный с нуля. Система Linux во многом подобна FreeBSD, хотя FreeBSD обладает намного большим наследием и лучше подходит для коммерческого использования, чем Linux. Linux включает в себя требование, гласящее, что все изменения, выполненные для коммерческих продуктов, должны вноситься обратно в Linux. BSD не имеет такого ограничения. Поклонники Linux утверждают: «В эксплуатации FreeBSD более уязвима, чем Linux». Разработчики Linux верят в идею совместного использования кода, тогда как разработчики BSD предлагают свой код в подарок любому желающему. Все зависит от того, что важнее для вас.

У многих пользователей UNIX возникает ощущение конфликта между лагерями BSD и Linux. Однако если копнуть немного глубже, можно обнаружить, что большинство разработчиков этих операционных систем открыто и дружелюбно взаимодействуют друг с другом. Лишь узкий круг пользователей и малая часть разработчиков становятся источником трений, как группы футбольных хулиганов или поклонники различных серий фильма «Star Trek».

IRIX, HP/UX и другие

Есть и другие системы UNIX, например IRIX компании Silicon Graphics (солидная система UNIX для графических приложений) и HP/UX компании Hewlett-Packard, популярная на больших предприятиях. Если выполнить поиск в Интернете, можно также обнаружить меньших конкурентов, таких как Tru64 UNIX и UnixWare – убийственную разработку SCO Group. Можно также встретить старые ископаемые, такие как A/UX компании Apple и Xenix компании Microsoft. (Да, Microsoft была лицензированным поставщиком UNIX, но это было очень давно, когда по земле бродили динозавры и мой отец охотился на мамонтов и участвовал в ритуальных плясках своего племени.) Многие приложения высокого класса разрабатывались для работы под управлением какой-то одной версии UNIX. Все современные системы UNIX учли опыт более старых версий, и современные UNIX и UNIX-подобные системы во многом схожи.

Сильные стороны FreeBSD

Как же после всего этого можно охарактеризовать FreeBSD?

Переносимость

Цель проекта FreeBSD – предоставить стабильную, безопасную и свободно распространяемую операционную систему, способную работать на популярных аппаратных платформах. Сегодня это системы, совместимые с Intel x86 (486, различные версии процессора Pentium, AMD и другие, а также 64-битовая архитектура AMD – amd64, скопированная компанией Intel как EMT64). На более старых платформах новые версии FreeBSD уже не работают, но большинство устаревших систем либо уже пришли в негодность, либо на них не предполагается менять операционную систему.

Новым дополнением к FreeBSD является платформа ARM, которая используется для встраиваемых устройств. Кроме того, FreeBSD поддерживает процессоры SPARC компании Sun и Itanium компании Intel (IA64), а также процессор PowerPC компании Motorola. Существуют и другие платформы, такие как архитектуры x86 и amd64, но они появились слишком поздно и потому не получают должного внимания.

Мощь

Поскольку FreeBSD адекватно работает на аппаратной платформе 386, она достаточно хорошо работает на современных компьютерах. Довольно приятно работать с системой, не требующей процессора Pentium III и полгигабайта оперативной памяти для поддержки пользовательского интерфейса. В результате вся вычислительная мощь может быть задействована для решения насущных задач, а не тех, до которых вам нет дела. Если пользователь выберет приятный графический интерфейс

с необычными прибамбасами, FreeBSD его поддержит, но не станет требовать, чтобы он работал только с ним. Кроме того, FreeBSD поддерживает современные многопроцессорные аппаратные платформы.

Упрощенное управление программным обеспечением

Благодаря коллекции «портов» система FreeBSD облегчает управление программным обеспечением. Традиционно в системе UNIX настройка программного обеспечения требовала значительной квалификации. Коллекция «портов» существенно упрощает эту задачу за счет автоматизации и документирования установки, удаления и конфигурирования тысяч программных пакетов.

Оптимизированный процесс обновления

В отличие от других операционных систем, в которых процедура обновления мучительна и опасна, простой процесс обновления FreeBSD строит операционную систему, оптимизированную для работы на конкретной программно-аппаратной платформе. FreeBSD учитывает каждую ее особенность, а не сводит поддержку к наименьшему общему знаменателю. Заменяя аппаратные средства, можно пересобрать систему с учетом их особенностей. Именно так поступают Sun и Apple, поскольку они создают как аппаратные средства, так и операционные системы. В то же время FreeBSD не привязывает вас к конкретной аппаратной платформе.

Улучшенная файловая система

Файловая система (*filesystem*) определяет, как информация хранится на физическом диске – то есть как файл с именем *My Resume* преобразуется в последовательность нулей и единиц на металлической пластине жесткого диска. В состав FreeBSD входят очень сложные файловые системы. Они могут поддерживать файлы размером до петабайта (тысяча тысяч гигабайт). Они обладают высокой устойчивостью к сбоям, а запись и чтение файлов происходят чрезвычайно быстро. Файловая система BSD настолько усовершенствована, что принята многими производителями коммерческих версий UNIX.

Кому следует использовать FreeBSD

FreeBSD может применяться как очень мощная настольная система и среда разработки, но ее история свидетельствует о преимущественной ориентации на Сеть, почтовые и файловые службы, а также сервисы поддержки. По существу, основная сильная сторона FreeBSD – это серверы Интернета. Система представляет собой замечательный выбор для любого интернет-сервиса. Если такие крупные фирмы, как Yahoo! полагаются на FreeBSD для предоставления своих услуг, то она прекрасно подойдет и для вас.

Тому, кто задумывается о запуске FreeBSD (или любой другой системы UNIX) на настольной машине, необходимо понимать, как работает компьютер. FreeBSD – это не лучший выбор, если вы ищете простую систему с интерфейсом «укажи-и-щелкни». Если ваша цель такова, выберите компьютер Макинтош и вы сможете получить доступ к мощи UNIX, когда она вам потребуется, и не вспоминать о ней в оставшееся время. Если вам требуется изучить FreeBSD, то запустите ее на своем настольном компьютере, а как – будет рассказано позже.

Кому следует использовать другие разновидности BSD

Ближайшими конкурентами FreeBSD являются NetBSD и OpenBSD. Однако, в отличие от конкурентов в коммерческом мире, здесь конкуренция большей частью дружелюбна. FreeBSD, NetBSD и OpenBSD свободно разделяют код и разработчиков; некоторые разработчики даже поддерживают одну и ту же подсистему в нескольких операционных системах.

Если необходима поддержка устаревшего оборудования, лучшим выбором будет NetBSD. В течение нескольких лет я использовал NetBSD на древней рабочей станция SGI в качестве сервера доменных имен (Domain Name System, DNS) и файлового сервера. И она прекрасно справлялась со своей работой, пока не рабочая станция не задымилась и не испустила дух.

В OpenBSD было реализовано значительное число функций обеспечения безопасности. Многие инструментальные средства в конечном счете были интегрированы и во FreeBSD, но это заняло месяцы и годы. Если вам необходим высокий уровень безопасности, но не требуется поддержка сложных многопроцессорных систем, обратите свое внимание на OpenBSD.

Если же вы просто экспериментируете, тогда вам подойдет любая из BSD-систем!

Кому следует использовать патентованную операционную систему

Патентованные[1] операционные системы, подобные Solaris, Windows, AIX и другие системы такого рода по-прежнему довольно популярны,

[1] Это удачный перевод слова «proprietary», которое менее удачно переводят как «проприетарные». В список таких систем автор несправедливо поместил Solaris – несмотря на то, что в нем используются запатентованные технологии Sun Microsystems, практически весь код этой системы открыт на условиях лицензии CDDL, схожей с лицензией BSD, а сама система распространяется бесплатно (как и OpenSolaris). – *Прим. научн. ред.*

хотя операционные системы с открытыми исходными текстами расширяют свое присутствие на рынке. Предприятия с высоким уровнем автоматизации достаточно сильно привязаны к этим операционным системам. Ситуация меняется медленно, и вы наверняка столкнетесь с коммерческими версиями операционных систем в этих средах. Однако применение FreeBSD для предоставления основных услуг, таких, как диспетчерский контроль или файловый сервер, рассчитанный на отдел, может сделать вашу жизнь намного легче за гораздо меньшие деньги. Yahoo! и NetApp целиком построили свой бизнес на использовании FreeBSD, а не на коммерческих аналогах.

Конечно, если ваше программное обеспечение работает только на патентованной системе, то ваш выбор уже определен. Тем не менее всегда спрашивайте производителя программ, доступна ли версия для FreeBSD; вас может ждать приятный сюрприз.

Как читать эту книгу

Толщина многих компьютерных книг достаточна для того, чтобы ими можно было оглушить быка, – если вы сможете поднять их достаточно высоко. Кроме того, они либо всеобъемлющи, либо настолько насыщены деталями, что их трудно читать. Нужен ли вам снимок экрана, который говорит только «нажмите OK» или «примите лицензионное соглашение»? Когда последний раз вы читали энциклопедию?

Книга «FreeBSD. Подробное руководство» немного другая. Она предназначена для однократного прочтения от начала и до конца. При желании главы можно пропускать, однако каждая глава построена на материале предыдущей. Кроме того, книга не очень большая, поэтому усвоить ее содержимое довольно легко. Прочитав ее один раз, вы сможете использовать ее в качестве справочника.

Если вы часто покупаете компьютерные книги, можете сами добавить обычные рекомендации типа «для наилучшего усваивания читайте за раз одну главу» и т. д. Я не собираюсь нянчиться с вами, – если вы приобрели эту книгу, то, возможно, у вас найдется пара извилин или кто-то, кто вам поможет. (Если последнее, то, надеюсь, ваш начальник достаточно умен, чтобы отобрать у вас эту книгу прежде, чем вы узнаете достаточно и станете опасны.)

Что вы должны знать?

Эта книга адресована начинающим администраторам UNIX. Два десятилетия назад средний администратор UNIX обладал опытом разработки программного кода ядра и имел ученую степень по вычислительной технике. Даже десять лет тому назад он уже был квалифицированным пользователем UNIX, обладающим навыками программирования и дипломом бакалавра. Сегодня UNIX-подобные операционные системы

доступны любому желающему, а компьютеры стали дешевле продуктов питания и даже 12-летние дети могут запускать UNIX, читать исходный код и получать знания, наводя ужас на нас, старых администраторов. Поэтому я не жду от вас, что вы обладаете серьезными знаниями об операционной системе UNIX.

Чтобы использовать весь потенциал этой книги, вы должны быть знакомы с базовыми командами UNIX. Это команды для перехода в другой каталог, вывода списка файлов в нем, а также входа в систему с именем пользователя и паролем. Тем, кто не знаком с базовыми командами и с командной оболочкой UNIX, я рекомендую начать с книги Эви Немет (Evi Nemeth) с соавторами «UNIX System Administration Handbook» (Prentice Hall PTR, 2006). Чтобы облегчить жизнь начинающим системным администраторам, я включил в текст книги точные команды для получения конкретных результатов. Если вы лучше усваиваете материал на примерах, то в этой книге вы найдете все, что вам нужно.

Кроме того, надо немного разбираться в аппаратных средствах компьютера – не очень глубоко, уверяю вас, но в некоторой степени. Например, полезно знать, чем отличаются кабели IDE, SCSI или SATA. Конечно, надо знать именно свои аппаратные средства, однако если вы заинтересовались этой книгой, то думаю, что достаточными знаниями вы обладаете.

Начинающим системным администраторам

Если вы еще не знакомы с UNIX, лучший способ знакомства заключается в том, чтобы «жить жизнью вашей собственной собаки». Нет, я вовсе не предлагаю, чтобы вы обедали вместе с Рексом. Но если бы вы управляли компанией по производству корма для собак, вы наверняка захотели бы выпускать такой корм, который ваша собака ела бы с удовольствием. Если ваша собака воротит нос от вашего последнего рецепта, значит, вы что-то не так делаете. Суть в том, что если вы производите продукт, вы сами должны использовать его. То же относится и к UNIX-подобным операционным системам, включая FreeBSD.

FreeBSD как настольная среда

Если вы всерьез намерены изучить FreeBSD, я предлагаю удалить имеющуюся операционную систему с вашего основного компьютера и установить на него FreeBSD. Теперь я знаю, что собачья еда не так плоха на вкус. Однако изучение операционной системы сродни изучению иностранного языка. Самым быстрым методом обучения является метод полного погружения. Именно этот подход я избрал и сегодня я могу заставить UNIX-систему делать все, что угодно. Фактически эта книга была полностью написана на моем ноутбуке, работающем под управлением FreeBSD, с помощью текстового редактора с открытым кодом XEmacs и бизнес-пакета OpenOffice. Кроме того, во FreeBSD я смотрю

видеофильмы, записываю и прослушиваю музыку в формате MP3, проверяю баланс моего счета в банке, обрабатываю электронную почту и путешествую по Интернету. Когда я пишу эти строки, по экрану, прямо поверх окон, бегает с дюжину маленьких анимационных демонов BSD[1] и я иногда развлекаюсь, щелкая по ним мышкой. Если это не считать дурацкой экранной программкой, то не знаю, что и считать.[2]

Многие системные администраторы UNIX в наши дни приходят из мира Windows. Они усердно трудятся в своем маленьком мирке, когда мимо пролетает босс и говорит: «Вы в состоянии обслуживать еще одну систему? Рад слышать это! Кстати, это – система UNIX», а затем исчезает в административных небесах. Если только новый администратор решает поберечь свои руки, руки «белого воротничка», и не начинать захватывающую карьеру специалиста по вскрытию китов, он начинает ковыряться в системе. Он узнает, что команда `ls` подобна `dir`, а `cd` одинаково выполняется на обеих платформах. Команды можно заучивать наизусть, читать их описания и исследовать на практике. Однако таким образом нельзя понять, как думает машина UNIX. Она не приноровится к вам – вы должны приспособить ее под себя. И Windows, и Mac OS X требуют подобного приспособления, но они скрывают это за блестящим фасадом. Помня об этом, мы потратим некоторое время на обсуждение того, как следует думать о системе UNIX.

Как думать о UNIX

Современные UNIX-системы обладают превосходным графическим интерфейсом, что называется, прямо из коробки, но это лишь украшение для глаз. Реальная работа ведется в командной строке, и неважно, какое количество инструментов скрывается за ней. Командная строка – это одна из самых сильных сторон UNIX и благодаря командной строке система обладает беспрецедентной гибкостью.

В основу UNIX положена философия: *множество маленьких инструментов, каждый из которых выполняет единственную операцию, но делает это хорошо*. В каталоге локальных программ в моем ноутбуке (*/usr/local/bin*) находятся 662 программы. Я установил каждую из них прямо или косвенно. Большинство из них – это маленькие простые программы, которые решают единственную задачу, за редким исключением, таким как офисный пакет. Этот массив маленьких инструментов обеспечивает операционной системе UNIX необычайную гибкость и приспособляемость. Многие коммерческие программные пакеты стремятся вобрать в себя сразу целую массу функций; они реализуют

[1] Изображение забавного чертенка или демона является логотипом FreeBSD. – *Прим. перев.*

[2] В первом издании этой книги я забыл упомянуть, как создать подобную программку, что вызвало больше электронных писем, чем любая другая тема во всей книге. Это ошибка, которую я не повторю!

самые разные функциональные возможности, но основную задачу решают весьма посредственно. Помните, когда-то необходимо было быть программистом, чтобы использовать UNIX, не говоря о том, чтобы ею управлять. Программисты не против разработки своих собственных инструментов. А концепция каналов в UNIX способствует этому.

Каналы коммуникации

Пользователи, привыкшие к средам с графическим интерфейсом, таким как Windows и Mac OS X, наверняка незнакомы с тем, как UNIX обслуживает ввод и вывод. Они щелкают мышью и видят либо сообщение «ОК», либо ошибку, либо вообще ничего, либо (увы, слишком часто) красивый голубой экран, заполненный терминами из области высоких технологий, объясняющими, где произошла авария в системе. В UNIX все немного по-другому.

Программы UNIX имеют три канала коммуникации: стандартный ввод, стандартный вывод и стандартная ошибка. Разобравшись с работой этих каналов, вы сделаете хороший шаг на пути к пониманию всей системы.

Стандартный ввод (*standard input*) – это источник информации. Когда вы набираете команды за консолью, стандартным вводом является клавиатура. Если программа прослушивает сеть, стандартным вводом считается сеть. Многие программы могут переназначать стандартный ввод, чтобы принимать данные из сети, из файла, с клавиатуры и из любого другого источника.

Стандартный вывод (*standard output*) – это место, где отображается вывод программы. Зачастую это консоль (экран). Сетевые программы обычно возвращают вывод в сеть. Программы могут отправлять свой вывод в файл, передавать другим программам, отправлять в сеть или еще в какое-то место, доступное для компьютера

Наконец, *стандартный поток ошибок* (*standard error*) – это место, куда направляются сообщения об ошибках. Часто консольные программы возвращают ошибки на консоль; другие протоколируют ошибки в файле. Если программа неправильно настроена, это может привести к потере всех сообщений об ошибках.

Эти три канала можно переназначать произвольным образом, что, вероятно, труднее всего усваивается начинающими пользователями и администраторами UNIX. Например, сообщения об ошибках можно перенаправить с терминала в файл. Если вы не хотите постоянно вводить большой объем информации в командной строке, можете поместить эту информацию в файл (ее можно будет использовать повторно) и отправить содержимое файла на стандартный ввод команды. Или еще лучше: запустите команду, чтобы сгенерировать эту информацию и поместить ее в файл или просто организовать конвейер и отправить информацию напрямую второй команде.

Маленькие программы, каналы и командная строка

Крайние проявления каналов ввода/вывода могут ошеломить нового пользователя. Когда при прохождении учебных курсов по администрированию UNIX я впервые увидел специалиста, набирающего в командной строке нечто вроде нижеследующего, я захотел сменить профессию.

```
$ tail -f /var/log/messages | grep -v popper | grep -v named &
```

Строки малопонятного текста начали появляться на экране. Хуже того, мой инструктор продолжал набирать, когда вывод закрыл собой весь экран! Длинная строка команд определенно может запугать выходцев из среды с «мышиным» интерфейсом. Что означают все эти странные слова, не говоря уже о символах?

Изучение работы с командной строкой сродни изучению языка. Учась языку, мы начинаем с простых слов. По мере расширения нашего словаря мы узнаем, как связывать слова. Мы начинаем понимать, что порядок размещения слов имеет определенный смысл и что при размещении слов в другом порядке можно этот смысл потерять. В трехлетнем возрасте вы говорили не очень хорошо – дайте себе слабину и вы опять вернетесь к тому состоянию.

Маленькие и простые программы вкупе с каналами коммуникации обеспечивают беспримерную гибкость. Хотелось ли вам когда-нибудь использовать функцию из одной программы в другой? Применяя множество маленьких программ и назначая ввод и вывод по своему усмотрению, можно задавать любое требуемое поведение системы. Вы едва ли будете чувствовать себя уверенно, если не сможете пропустить вывод команды через конвейер `| sort -rnk 6 | less`.[1]

Все сущее есть файлы

Невозможно проработать длительное время с системой UNIX и ни разу не услышать, что все сущее в ней представляет собой файлы. Программы, учетные записи и настройки системы – все это хранится в файлах. Система UNIX не имеет аналога реестра Windows – если создать резервные копии всех файлов, будет создана копия всей системы.

Более того, аппаратные устройства в системе также представлены в виде файлов! Например, устройство CD-ROM представлено файлом */dev/acd0*. Сетевая карта – файлом */dev/net*. Даже виртуальные устройства, такие как устройство для сбора и анализа сетевых пакетов или разделы на жестком диске, тоже являются файлами.

---

[1] Эта группа команд принимает вывод предшествующей последней команды, сортирует по шестому столбцу в обратном порядке и выводит на экран по одной странице за раз. Если команда выводит тысячи строк текста и вам требуется узнать, какие строки содержат самые высокие значения в шестом столбце, данная группа команд поможет вам в этом. Или, если у вас масса свободного времени, можно отправить вывод в электронную таблицу и сделать то же самое с помощью малопонятных команд за более длительное время.

Помните об этом, когда возникает необходимость решить какую-либо проблему. Все сущее есть файлы или находится в файлах, где-нибудь в недрах системы. Все, что вам нужно будет сделать, – это найти этот файл!

Примечания ко второму изданию

Когда я написал свою первую книгу, все системы семейства BSD имели очень много общего. Системный администратор, знакомый с одной из BSD-систем, мог спокойно сесть за другую и настроить ее за пару часов. Некоторые инструменты могли находиться в непривычном месте, последовательность начальной загрузки системы могла несколько отличаться, некоторые функциональные особенности могли быть другими, но в общем и целом все системы несли в себе черты BSD 4.4. Так было пять лет тому назад, но за это время все системы BSD шли своим путем. В них еще остались некоторые общие черты, но различия стали настолько существенными, что я больше не могу говорить, что большая часть этой книги в равной степени применима ко всем трем BSD-системам. По этой причине второе издание книги получило название «FreeBSD. Подробное руководство», а не «BSD. Подробное руководство».

Безусловно, здесь вы найдете массу отличий от первого издания книги. Диапазон различий, от трудно уловимых до явных, между FreeBSD 4 и FreeBSD 7 настолько велик, что любой может быть сбит с толку, если не проявит должного внимания. В систему были интегрированы многие инструменты администрирования Sendmail, поэтому я буду рассказывать не о Postfix, а о Sendmail. (Мне по-прежнему нравится Postfix, но эта книга рассказывает о FreeBSD.) В 2000 году было трудно представить себе компьютер без накопителя на гибких магнитных дисках, теперь же некоторые компьютеры продаются вообще без устройств на сменных носителях. Это делает работу с бездисковыми станциями особенно важной, поскольку для некоторых аппаратных платформ это единственный способ запустить операционную систему на машине! Наконец, за последние пять лет FreeBSD далеко продвинулась в своем развитии, да и мои знания расширились гораздо значительнее, чем я мог себе представить. Надеюсь, что все это вместе взятое сделает второе издание книги значительно лучше ее предшественницы.

Структура книги

Второе издание «FreeBSD. Подробное руководство» включает следующие главы:

Глава 1 «Как получить помощь»

Охватывает дополнительные информационные ресурсы, которые проект FreeBSD и его сторонники предоставляют пользователям. Ни одна книга не может охватить весь материал. Умение приме-

нять многие доступные ресурсы FreeBSB поможет вам заполнить любые пробелы в информации, которую вы здесь найдете.

Глава 2 «Установка FreeBSD»

Содержит обзор установки FreeBSD на выделенной машине и советы по оптимальному конфигурированию.

Глава 3 «Пуск! Процесс загрузки»

Эта глава расскажет вам о том, как протекает процесс загрузки FreeBSD, а также о том, как запускать, останавливать и перезагружать систему в различных конфигурациях.

Глава 4 «Прочтите это раньше, чем что-нибудь испортите!»

Описывает способы резервирования данных как на уровне всей системы, так и на уровне отдельных файлов. Здесь же рассказывается о том, как вносить изменения, которые легко отменить.

Глава 5 «Эксперименты с ядром»

Описывает конфигурирование ядра FreeBSD. В отличие от некоторых других операционных систем ядро FreeBSD надо настраивать, чтобы оно лучше отвечало вашим целям. Такая возможность обеспечивает потрясающую гибкость и позволяет извлекать максимальную пользу из аппаратных средств.

Глава 6 «Работа в сети»

Содержит обсуждение сетевых возможностей и их работу во FreeBSD.

Глава 7 «Организация защиты системы»

Научит вас, как защитить систему от взломщиков и злоумышленников.

Глава 8 «Диски и файловые системы»

Рассматривает некоторые детали работы FreeBSD с жесткими дисками, файловые системы FreeBSD, поддержку других файловых систем и некоторые сетевые файловые системы.

Глава 9 «Расширенные средства защиты»

Описывает наиболее интересные функции защиты, присутствующие во FreeBSD.

Глава 10 «Каталог /etc»

Описывает основные конфигурационные файлы FreeBSD и их применение.

Глава 11 «Делаем систему полезной»

Описывает систему «портов» и пакетов, которая применяется для управления дополнительным программным обеспечением.

Глава 12 «Расширенное управление программным обеспечением»

Описывает некоторые из наиболее интересных особенностей управления программным обеспечением в системах FreeBSD.

Глава 13 «Обновление FreeBSD»

Обучает вас, как задействовать процесс обновления. Эта процедура FreeBSD, по сравнению с тем, что предлагают любые другие операционные системы, – одна из самых замечательных и логичных.

Глава 14 «Ориентирование в Интернете: DNS»

Описывает сервер доменных имен (DNS) и научит вас, как его установить и настроить.

Глава 15 «Управление малыми системными сервисами»

Здесь будут обсуждаться некоторые из небольших программ, которые потребуются вам для обеспечения нормальной работы FreeBSD.

Глава 16 «Спам, черви и вирусы (плюс электронная почта)»

Описывает, как во FreeBSD настроить систему электронной почты, которая обеспечит надежную доставку почты и будет защищена от спама и вирусов.

Глава 17 «Веб и FTP-сервисы»

Объясняет, как настроить и обезопасить две самые популярные службы Интернета.

Глава 18 «Советы по настройке геометрии дисков»

В этой главе рассматриваются некоторые интересные возможности FreeBSD в поддержке функции зеркалирования дисков, экспорт дисковых устройств в сети, а также общие вопросы защиты и манипулирования данными.

Глава 19 «Производительность системы и ее мониторинг»

Описывает некоторые инструменты FreeBSD для тестирования производительности и устранения неполадок, а также демонстрирует примеры интерпретации результатов. Здесь также будет рассматриваться процедура входа в систему, реализованная с использованием SNMP.

Глава 20 «Передовой край FreeBSD»

В этой главе будут даны наиболее интересные советы по работе с FreeBSD, например запуск системы на устройствах без дисков или с маленькими дисками, а также создание отказоустойчивых и резервных настроек.

Глава 21 «Аварии и паника системы (и системных администраторов)»

Обучает, как поступать в редких случаях неполадок с FreeBSD, как выявлять и устранять проблемы и готовить отчет о проблеме.

Приложение «Полезные секреты sysctl»

Описаны наиболее интересные и полезные параметры настройки ядра.

Итак, довольно вводного материала! Вперед!

1

Как получить помощь

В книге такого объема, как эта, невозможно охватить все темы. В конце концов, система UNIX имеет сорокалетнюю историю, BSD появилась более четверти века назад, а FreeBSD известна уже более 10 лет. Даже если запомнить весь материал книги, этого будет недостаточно для всех возможных ситуаций, особенно когда FreeBSD начинает вести себя как типичный подросток и нуждается в хорошем воспитании. Проект FreeBSD поддерживает широкий круг информационных ресурсов, включая многочисленные почтовые рассылки FreeBSD, веб-сайт FreeBSD, не говоря уже об официальном руководстве, Справочнике (Handbook) и веб-сайтах пользователей, где можно найти огромное количество документации. Море информации может подавлять, и у вас может появиться желание просто написать письмо с просьбой о помощи. Однако прежде чем публиковать свой вопрос в почтовой рассылке, убедитесь, что необходимой информации нет в других источниках.

Почему не почтой с самого начала?

Почтовые рассылки FreeBSD – это замечательные ресурсы для получения технической поддержки. Многие специалисты, отслеживающие почтовую рассылку, обладают большим багажом знаний и могут быстро ответить на вопросы. Однако помните: размещая свой вопрос в почтовой рассылке FreeBSD, вы требуете, чтобы его просмотрели десятки тысяч пользователей по всему миру. Это означает, что один или несколько человек потратят свое время на ответ, вместо того чтобы посмотреть любимое телевизионное шоу, насладиться обедом в кругу семьи или отоспаться. Неприятности возникают, когда на один и тот же вопрос эксперты отвечают 10, 50 и даже 100 и более раз. Они становятся раздражительными. Некоторые из них могут рассердиться не на шутку.

Плохо то, что эти люди затрачивают массу времени на подготовку ответов на эти вопросы – ответов, которые доступны в другом месте. Если очевидно, что запрашиваемых сведений нет в различных информационных источниках Проекта FreeBSD, спрашивающий наверняка получит вежливый полезный ответ. Однако, отвечая на вопрос, который уже задавался несколько сотен раз, эксперт может не выдержать и показать свой буйный нрав. Делайте сами свои домашние задания – есть шанс, что ответ будет найден раньше, чем придет совет по почте.

Позиция FreeBSD

«Домашние задания? Что это значит, домашние задания? Назад в школу? Что вы хотите – огненных жертвоприношений и преклоненных колен?» Да, обратно в школу. Мир информационных технологий – это пожизненное, непрекращающееся самообразование. Вам придется либо согласиться с этим, либо покинуть этот мир. Заметим, однако, что цифровая передача огненных жертвоприношений затруднительна, да и неуместны они в современном мире.

Коммерческие операционные системы скрывают свою начинку. Все, к чему есть доступ, – это опции, представленные производителем. Даже при желании невозможно понять, как работает тот или иной компонент. Когда происходит сбой, есть один выход – звонить производителю и униженно просить помощи. Хуже того, люди, оказывающие такую помощь за деньги, знают ненамного больше, чем пользователи, задающие вопросы.

Если ранее вы никогда не работали с производителями открытого программного обеспечения, механизм поддержки FreeBSD может удивить вас. Вы не найдете номер телефона, куда можно было бы позвонить, и нет производителя, с которым можно было бы поругаться. И вы не сможете связаться с менеджером, потому что вы и есть менеджер! Поздравляю с повышением!

Варианты поддержки

Следует сказать при этом, что вы не брошены на произвол судьбы. Сообщество FreeBSD насчитывает огромное число разработчиков, помощников и пользователей, которые обладают глубокими познаниями, и они будут рады поработать с вами. Система FreeBSD предоставляет все, что только может потребоваться: полный доступ к исходным текстам системы, инструменты, способные превратить эти исходные тексты в программы, и отладчики, которые используются разработчиками. От вас ничего не будет скрыто, вы сможете увидеть все достоинства и недостатки. Вы сможете ознакомиться с историей развития FreeBSD, начиная с самого начала, включая каждое изменение и причины, его вызвавшие. Вы можете не уметь пользоваться этими инструментами, но это не проблема проекта. Многие члены сообщества готовы помогать вам развивать ваши навыки, благодаря чему вы научи-

тесь пользоваться этими инструментами. Вам будет предоставлена неограниченная помощь в выполнении ваших обязанностей.

Общее правило таково: люди помогают себе подобным. Пользователь FreeBSD должен перейти от употребления готовой пищи к изучению рецептов ее приготовления. Пользователя, желающего подробно изучить свою систему, ждут с распростертыми объятиями. Если же он просто хочет знать, на какую кнопку нажать, ему следует обратиться к документации: у специалистов сообщества FreeBSD просто нет стимула помогать тем, кто не помогает себе сам или не способен следовать инструкциям.

Если вы хотите использовать систему, но не имеете ни времени, ни желания учиться, тогда заключите контракт на получение коммерческой поддержки. Это не означает, что вы получите возможность контактировать с владельцем FreeBSD, но, по крайней мере, у вас появится кто-то, на кого можно будет выплеснуть свое недовольство. На веб-сайте FreeBSD вы найдете перечень компаний, которые предоставляют платную техническую поддержку.

Важно помнить, что проект FreeBSD поддерживает лишь систему FreeBSD. Если у вас произошла неприятность с каким-либо другим программным обеспечением, списки рассылки FreeBSD – не самое лучшее место для размещения вашего вопроса. Вообще разработчики FreeBSD обладают серьезным опытом и в другом программном обеспечении, но это не означает, что у них появится желание объяснить вам, например, как настроить KDE.

Итак, первая часть домашнего задания – изучить, какие источники информации существуют помимо этой книги. Сюда входят интегрированное справочное руководство, веб-сайт FreeBSD, архивы почтовой рассылки и другие веб-сайты.

Страницы руководства

Страницы руководства (man pages; man – сокращение от «manual», руководство) – это первичная документация UNIX. Несмотря на то, что они считаются непонятными, трудными для изучения, и говорят даже, что их невозможно читать, они вполне дружелюбны к пользователям – к особым пользователям. Когда создавались эти страницы, средний системный администратор писал программы на C. В результате эта документация была написана программистами для программистов. Пользователь, способный думать как программист, сочтет эти страницы совершенными. Я пробовал думать как программист, но реального успеха достиг лишь после двух дней непрерывного изучения документации (помогли сильное нервное возбуждение и кофеин).

За последние несколько лет упал уровень квалификации, необходимый для системного администрирования. Теперь системному администратору не обязательно быть программистом. Аналогичная вещь произошла

со страницами руководства – читать их стало легче. Страницы руководства – это не учебник, они объясняют поведение программ, а не как достичь желаемого результата. По существу, они являются первой линией атаки при изучении работы системы. Пользователь, отправивший письмо в почтовую рассылку, не заглянув в эти страницы документации, наверняка получит краткий совет – *заглянуть именно в них*.

Разделы страниц руководства

Руководство по FreeBSD состоит из девяти разделов. Эти разделы выглядят так:

1. General commands (Основные команды)
2. System calls and error numbers (Системные вызовы и коды ошибок)
3. The C libraries (Библиотеки C)
4. Devices and device drivers (Устройства и драйверы устройств)
5. File formats (Форматы файлов)
6. Game instructions (Инструкции к играм)
7. Miscellaneous information (Всякая всячина)
8. System maintenance commands (Команды обслуживания системы)
9. Kernel system interfaces (Системные интерфейсы ядра)

Каждая страница руководства начинается с названия команды, которую она описывает, вслед за которым следует номер раздела в круглых скобках, например reboot(8). Когда такое упоминание команды встречается где-нибудь в документации, это означает, что необходимо прочитать эту страницу руководства из указанного раздела. Почти каждая тема имеет ссылку на страницу документации. Например, для просмотра страницы руководства по редактору vi надо ввести следующую команду:

```
$ man vi
```

В ответ система выдаст такую информацию:

```
VI(1)                                                          VI(1)

NAME (ИМЯ)
    ex, vi, view - text editors    (текстовые редакторы)

SYNOPSIS (СИНОПСИС)
    ex [-eFGRrSsv] [-c cmd] [-t tag] [-w size] [file ...]
    vi [-eFGlRrSv] [-c cmd] [-t tag] [-w size] [file ...]
    view [-eFGRrSv] [-c cmd] [-t tag] [-w size] [file ...]

LICENSE (ЛИЦЕНЗИЯ)
    The vi program is freely redistributable. You are welcome to copy,
    modify and share it with others under the conditions listed in the
    LICENSE file. If any company (not individual!) finds vi sufficiently
    useful that you would have purchased it, or if any company wishes to
    redistribute it, contributions to the authors would be appreciated.
```

```
        (Перевод: Программа vi распространяется свободно, без ограничений.
        Допускается ее копирование, модификация и передача другим лицам согласно
        условиям, приведенным в файле LICENSE. Если любая компания (не отдельное
        лицо!) сочтет программу vi в достаточной мере полезной, она может купить
        ее. Если любая компания пожелает распространять vi, авторы будут
        признательны за полученные деньги.)

DESCRIPTION
        Vi is a screen oriented text editor. Ex is a line-oriented text editor
        Ex and vi are different interfaces to the same program, and it is
        possible to switch back and forth during an edit session. View is the
        (Vi - экранный текстовый редактор. Ex - строчный. Ex и vi - это различные
        интерфейсы к одной и той же программе, между которыми можно
        переключаться на протяжении сеанса редактирования.)
:
```

Страница начинается с названия команды (vi) и номера раздела (1), затем следует название команды. Данная страница описывает три команды: ex, vi и view. Дав команду `man ex` или `man view`, вы получите ту же самую страницу.

Навигация по страницам руководства

Вызвав страницу руководства, можно нажатием клавиши пробела или Page Down переместиться на один экран вперед. Если вам не требуется шагать так далеко, воспользуйтесь клавишей Enter или клавишей «стрелка вниз» для перемещения вперед на одну строку. Клавиша B или Page Up переместит вас на один экран назад. Чтобы выполнить поиск в пределах страницы, нажмите клавишу / и вслед за символом слэша введите искомое слово. В результате вы переместитесь к первому вхождению слова, и оно будет подсвечено. Последующие нажатия клавиши N будут перемещать вас к следующим вхождениям этого слова.

Все это сказано в предположении, что вы используете программу постраничного просмотра по умолчанию – more(1). Если вы пользуетесь другой программой, используйте комбинации клавиш, присущие ей. Разумеется, если вы знаете UNIX достаточно, чтобы использовать другую программу постраничного просмотра, то вы можете пропустить этот раздел.

Поиск страниц руководства

Начинающие пользователи нередко говорят, что были бы счастливы ознакомиться со страницами руководства, если бы только могли найти нужные им. Поиск в страницах руководства может выполняться с помощью команд `apropos(1)` и `whatis(1)`. Команда `apropos(1)` отыщет все страницы руководства, названия или описания которых включают указанное вами слово. Команда `whatis(1)` выполнит такой же поиск, но отыщет только те страницы, где указанное слово встречается целиком. Например, для случая с командой `vi` можно попробовать выполнить такой поиск:

```
$ apropos vi
BUS_ADD_CHILD(9)        - add a device node to the tree with a given priority
BUS_PRINT_CHILD(9)       - print information about a device
BUS_READ_IVAR(9), BUS_WRITE_IVAR(9) - manipulate bus-specific device
instance
variables
DEVICE_ATTACH(9)         - attach a device
...
```

В общей сложности будет найдено 581 вхождение, что, вероятно, много больше, чем вам может потребоваться. Большинство из этих вхождений не имеют никакого отношения к vi(1), просто в названии или в описании этих страниц присутствует комбинация символов *vi*. Сочетание слов *device driver* имеют широкое распространение в страницах руководства, поэтому такой результат неудивителен. В подобных случаях больше пользы может принести команда `whatis(1)`.

```
$ whatis vi
ex(1), vi(1), view(1)     - text editors
etags(1), ctags(1)        - generate tag file for Emacs, vi
$
```

Здесь было получено всего два результата, и оба имеют явное отношение к vi(1). Иногда поиск с помощью команды `apropos(1)` дает более востребованные результаты, чем `whatis(1)`. Поэкспериментируйте с этими командами, и вы быстро поймете, какие из них когда будут вам полезны.

Номера разделов и страницы руководства

Иногда случается так, что для одного и того же названия команды имеется несколько страниц в руководстве. Например, в каждом разделе руководства имеется вводная страница, в которой приводится описание раздела. Чтобы указать, в каком разделе следует искать страницу руководства, указывайте номер раздела сразу вслед за командой `man`.

```
$ man 3 intro
```

Данная команда приведет вас на вводную страницу третьего раздела руководства. Я рекомендую прочитать все вводные страницы хотя бы ради того, чтобы вы осознали глубину и широту доступной информации.

Содержимое страниц руководства

Страницы руководства делятся на разделы. Хотя в странице руководства могут присутствовать практически любые заголовки, некоторые из них являются стандартными. См. mdoc(7) для получения неполного списка и другой информации о стандартах страниц руководства:

- `NAME` сообщает имя (или имена) программы или утилиты. Некоторые программы могут иметь несколько имен, например, текстовый редактор vi(1) имеет также имена ex(1) и view(1).

- SYNOPSIS приводит перечень возможных ключей командной строки и их аргументы или как выглядит вызов библиотечной функции. Если я уже знаком с программой, но просто не могу вспомнить допустимые параметры, этого раздела достаточно, чтобы освежить свою память.
- DESCRIPTION содержит краткое описание программы, библиотеки или функции. Содержимое этого раздела зависит от темы, которую охватывает страница руководства, – страницы о программах, файлах и базовых интерфейсах документируются по-разному.
- OPTIONS (ключи) параметры командной строки программы и их воздействие.
- BUGS (ошибки) описывает известные проблемы, связанные с программным кодом. Зачастую это позволяет избавиться от многих хлопот. Сколько раз вы бились, решая какую-нибудь задачу, только чтобы убедиться, что программа не работает так, как можно было ожидать по документации? Цель раздела BUGS – сэкономить ваше время и описать известные ошибки и случаи странного поведения программы.[1]
- SEE ALSO (см. также) традиционно является последним разделом. Надо помнить, что система UNIX подобна языку и компоненты системы связаны в одно целое. Подобно клейкой ленте, ссылки из раздела SEE ALSO покажут, как компоненты соединяются в одно целое.

Если в какой-то момент у вас не окажется доступа к страницам руководства, вы сможете найти их на многих веб-сайтах. Так же как и на основном веб-сайте FreeBSD.

FreeBSD.org

Веб-сайт FreeBSD (*http://www.freebsd.org*) содержит массу разнообразной информации по вопросам установки и администрирования FreeBSD. Наиболее важными частями являются Справочник (Handbook), сборник FAQ (Frequently Asked Question, часто задаваемые вопросы) и архивы почтовых рассылок, однако здесь же вы найдете огромное число статей на самые разные темы. В дополнение к документации о FreeBSD на веб-сайте также имеется большой объем информации о внутреннем руководстве проектом FreeBSD и о состоянии различных частей проекта.

Если вы обнаружите, что основной веб-сайт работает слишком медленно, попробуйте воспользоваться зеркалом сайта. На основном сайте имеется раскрывающийся список национальных сайтов-зеркал, кроме того можно попробовать ввести адрес в формате *http://www.*

[1] Такая информация называется «честной». Многим специалистам в области информационных технологий этот термин может показаться незнакомым, и в этом случае вам поможет простой словарь.

<код_страны>.freebsd.org. Практически во всех странах существуют свои сайты, дублирующие веб-сайт FreeBSD. Я нередко замечал, что сайты-зеркала работают гораздо быстрее основного веб-сайта.

Веб-документы

Документация FreeBSD разделена на статьи (articles) и книги (books). Различие между ними весьма условно. Как правило, книги больше статей и охватывают более широкие темы, а статьи короче и описывают один предмет. Две книги, наиболее интересные новым пользователям, – это Handbook (Справочник) и FAQ.

Справочник – это непрерывно изменяющийся путеводитель проекта FreeBSD. Он описывает методику выполнения основных задач в системе и является хорошим подспорьем в начале работы. В действительности я преднамеренно не включил некоторые темы в эту книгу, потому что они достаточно подробно описываются в Справочнике (Handbook).

FAQ (Frequently Asked Question, часто задаваемые вопросы) содержит ответы на часто задаваемые вопросы из почтовых рассылок FreeBSD. Часть информации из FAQ продублирована в Справочнике, но далеко не вся.

Остальные книги охватывают самые разнообразные темы, начиная от отладки ядра и заканчивая организацией проекта FreeBSD.

Из имеющихся примерно 50 статей некоторые были сохранены по историческим причинам (например, план выпуска FreeBSD 5), тогда как другие обсуждают детали определенных частей системы, например, последовательные порты или CVSup. Некоторые из них написаны очень давно и сохраняются специально для тех пользователей, которые все еще используют системы двадцатого века.

Эти документы носят весьма формальный характер, и для их чтения требуется определенная подготовка. Кроме того, они всегда немного отстают от реальной жизни. Как правило, сначала появляется новая функциональная возможность и лишь потом, спустя недели или даже месяцы, в Справочнике появляется соответствующая запись. Если веб-документация покажется вам устаревшей, лучшим источником сведений для вас будет архив почтовых рассылок.

Архивы почтовых рассылок

Если возникшая проблема не вписывает новую страницу в историю FreeBSD, значит, кто-то с ней уже сталкивался и, вероятно, рассказал об этом в почтовой рассылке. Как-никак, появление архивов восходит к 1994 году и они содержат около двух миллионов сообщений. Конечно, найти необходимую информацию среди такого количества сообщений – задача непростая. (Когда в свет вышло первое издание книги, архивы содержали всего один миллион сообщений – их количество удвоилось всего за несколько лет!)

Сайт FreeBSD обладает возможностью поиска по веб-страницам, но она бледнеет перед теми возможностями, что предлагает поисковая система Google. Компания Google имеет поисковый сайт, предназначенный специально для поиска по темам, имеющим отношение к BSD, – *http://www.google.com/bsd*. Попробуйте поискать свое сообщение об ошибке в Google, как в Интернете, так и в отдельных группах. Почтовые рассылки FreeBSD также проиндексированы в группах Google, и вы можете с помощью Google выполнить поиск на веб-сайте FreeBSD.org, включив в строку запроса текст: *site:freebsd.org*. Кроме того, замечательный механизм поиска по темам, имеющим отношение к BSD, предоставляет сайт *http://freebsd.rambler.ru*. Этот сайт работает под управлением FreeBSD и как минимум один из его специалистов является членом коллектива создателей FreeBSD.

Другие веб-сайты

Руками пользователей FreeBSD было создано множество веб-сайтов, где вы можете найти ответы на интересующие вас вопросы, получить помощь, найти обучающую информацию, познакомиться с продуктами и просто пообщаться. Ниже приводится список моих любимых сайтов:

Daemon News (*http://www.daemonnews.org*)

На этом сайте вы найдете ссылки на самые разные новости, имеющие отношение ко всем системам BSD, а не только FreeBSD.

FreeBSD Mall (*http://www.freebsdmall.com*)

Те, кто занимается поддержкой FreeBSD Mall, с самого начала предоставляют коммерческую поддержку FreeBSD. Они продают компакт-диски с FreeBSD, предоставляют возможность обучения и поддержки, а также продают различные товары с символикой FreeBSD, например футболки и игрушки. Сайт FreeBSD Mall принадлежит компании IX Systems.

O'Reilly Network BSD Developer Center (*http://www.onlamp.com/bsd*)

Здесь размещено множество статей по BSD и других материалов, которые будут интересны пользователям BSD. По моему непредвзятому мнению, самое интересное на данном сайте – это колонка «Big Scary Daemons» (Большие ужасные демоны), впрочем, все остальное тоже будет полезно.

Использование ресурсов FreeBSD, связанных с принятием решений

Теперь возьмем простую задачу и воспользуемся ресурсами FreeBSD для ее решения. Следующей вопрос я несколько раз встречал в почтовых рассылках FreeBSD, поэтому начнем с него:

«Я только что установил FreeBSD на компьютер с процессором 486, и моя сеть не работает. После выполнения команды ping на консоли появляется сообщение `ed0: timeout`. В чем тут дело?»

Для нахождения ответа применим несколько методов.

Поиск в Справочнике/FAQ

Беглый просмотр Справочника ничего не дал. Однако в FAQ, в разделе Troubleshooting (Устранение неполадок), есть статья:

```
I keep seeing messages like "ed1: timeout". What's wrong?
(Я постоянно вижу сообщения типа ed1: timeout. В чем тут дело?)
```

Достаточно близко. Прочитайте статью и попробуйте применить предложенное решение.

Поиск в страницах руководства

Далее будет объяснено, что цифры после имени устройства (device name) просто указывают на конкретное устройство. `ed0` – это устройство `ed` с номером `0`. Для каждого драйвера устройства имеется своя страница руководства, поэтому надо набрать `man ed` и на экране появится следующая информация:

```
ED(1)         FreeBSD General Commands Manual         ED(1)
        (Руководство по основным командам FreeBSD)

NAME (ИМЯ)
    ed, red -- text editor

SYNOPSIS (СИНОПСИС)
    ed [-] [-sx] [-p string] [file]
    red [-] [-sx] [-p string] [file]

DESCRIPTION
    The ed utility is a line-oriented text editor. It is used to create,
    (Перевод: ed - это строчный редактор. Он применяется для создания,)
...
```

Текстовый редактор? В чем дело? У меня с текстовым редактором все в порядке! Здесь явно что-то не так. Приглядитесь внимательнее к этой странице руководства – она принадлежит к первому разделу под названием «General Command» (основные команды). Вам необходимо отыскать другую страницу руководства, которая содержала бы слово *ed*. Так как такая комбинация символов имеет достаточно большое распространение, воспользуемся командой поиска `whatis(1)`.

```
$ whatis ed
ed(1), red(1)       - text editor
ed(4)               - NE-2000 and WD-80x3 Ethernet driver
                      (драйвер Ethernet для устройств NE-2000 и WD-80x3)
```

Ага! Теперь видно, что команда вызова редактора ed(1) – это команда общего назначения. А нам нужно посмотреть описание ed в четвертом

разделе. Наберите `man 4 ed`, и на экране появится страница руководства с описанием сетевого устройства. Она имеет достаточно большой объем, порядка 500 строк. Будучи ленивым от природы, я не буду читать всю страницу, а воспользуюсь поиском и отыщу те сведения, которые мне действительно необходимы. Взглянув еще раз на сообщение об ошибке, я подумал, что слово *timeout* может служить неплохим ключевым словом для поиска. Введите `/timeout` и нажмите клавишу Enter.

```
ed%d: device timeout Indicates that an expected transmitter interrupt
did not occur. Usually caused by an interrupt conflict with another card
on the ISA bus.
(ed%d: тайм-аут в работе устройства свидетельствует, что ожидаемого
прерывания от передатчика не произошло. Обычно это вызвано конфликтом
по прерываниям с другой картой на шине ISA.)
```

Опять удача! Здесь представлено краткое описание проблемы и вызвавшая ее причина (конфликт по прерываниям). Старая добрая проблема с IRQ, и если ваш компьютер действительно создан на базе процессора 486, теперь вы знаете об этой проблеме гораздо больше, чем вам хотелось бы.

Поиск в архивах почтовых рассылок

Поиск в архивах почтовых рассылок можно производить с помощью поискового механизма веб-сайта FreeBSD, но я предпочитаю пользоваться поисковой системой Google или Rambler. Поиск по строке *ed0: timeout site:FreeBSD.org* дает целую массу результатов. Некоторые из них датированы 1994 годом. Я только что выполнил такой поиск, и первая же ссылка привела меня к нужному ответу. Когда я работал над первым изданием книги, результат был тот же. Разве это не быстрее, чем составление письма в почтовую рассылку?

Использование ответа

Любой ответ, полученный на вопрос о «`ed0: timeout`», подразумевает, что спрашивающий знает, что такое IRQ и как настраивать аппаратуру. Эти знания соответствуют типичному уровню квалификации,

Спрашиваем снова... и снова... и снова...

Некоторые ответы на данный вопрос датированы 1994 годом. Да, этой проблеме уже более 12 лет! Помните, я упоминал тех, кому уже надоело отвечать на один и тот же вопрос снова и снова? Некоторые из таких вопросов задают по несколько раз в году. Поэтому обязательно проверьте все источники информации, где вы могли бы найти решение вашей проблемы. Если вы не нашли ничего похожего, тогда вполне вероятно, что ваше проблема действительно настолько уникальна, что можно возвестить о ней миру.

необходимому для решения основных задач. Если полученный ответ неясен, эту тему следует изучить и понять. Несмотря на то, что опытный разработчик или системный администратор, вероятно, не загорятся желанием объяснять, что такое IRQ, они охотно сообщат адрес веб-страницы, где рассматривается эта тема.

Письмо о помощи

Если вы наконец решили просить о помощи, оформляйте свою просьбу так, чтобы другие могли оказать вам действенную помощь. Обязательно включите в письмо всю информацию, имеющуюся в вашем распоряжении (это мы обсудим чуть ниже). Так как входящей информации очень много, вы можете кое-что (или многое) опустить. Если в описании проблемы будет отсутствовать необходимая информация, тогда случится одно из двух:

- Вопрос будет проигнорирован.
- Придет масса сообщений, требующих эту информацию собрать.

С другой стороны, если вы действительно хотите получить помощь в решении проблемы, включите в сообщение следующую информацию:

- Полное описание проблемы. Сообщение вроде «Как заставить модем работать?» вызовет массу вопросов. Что требуется от модема? Модем какого типа используется? Каковы симптомы? Как вы его используете?
- Вывод команды `uname -a`. В нем сообщается версия операционной системы и платформа.
- В случае, если система обновлялась из CVSup, укажите дату и время последнего обновления. (Это дата самого свежего файла в каталоге */usr/src*.)
- Текст сообщения об ошибке. Предоставляйте как можно более полную информацию и включайте в письмо любые сообщения, которые выводятся на консоль или в системный журнал, особенно в */var/log/messages*, а также сообщения из журнала программы, если таковой существует. Сообщения о проблемах с аппаратурой должны содержать копию файла */var/run/dmesg.boot*.

Лучше всего начать письмо так: «Мой модем не устанавливает соединение с провайдером. Модем BastardCorp v.90 model BOFH667. Версия ОС – 7.2 на двухъядерном процессоре Opteron. В файлах */var/log/messages* и */var/log/ppp.log* сообщений об ошибках нет». Таким образом можно избежать потока писем со встречными вопросами, и вы получите удовлетворяющий вас ответ гораздо быстрее.

Подготовка письма

Во-первых, будьте вежливы. Люди частенько такое пишут в электронных письмах, что никогда не решились бы сказать в лицо. Помните,

что рассылка поддерживается добровольцами, которые отвечают на письма лишь из добрых побуждений. Перед тем как нажать кнопку Send (Отправить), спросите себя: «А был бы я готов ради ответа на это письмо опоздать на свидание с ожидающими меня двойняшками?»[1] Иногда просто жесткая позиция, которая вырабатывается у тех, кто работает в корпоративной службе технической поддержки и сидит на телефоне, заставляет специалистов удалять ваши письма, так и не прочитав их. Их мир не приемлет грубых слов. Кричать, пока вам кто-то не поможет, – это ценный навык, если вы имеете дело с коммерческой службой технической поддержки, но этот навык только навредит вам, если вы пытаетесь получить поддержку в сообществе FreeBSD.

Письмо должно содержать только текст (plain text) и никакой разметки HTML. Многие разработчики FreeBSD читают письма с помощью текстовых программ электронной почты, таких как mutt или elm. Это очень мощные инструменты, предназначенные для обработки больших объемов электронной почты, но они не могут отображать сообщения в формате HTML. Чтобы понять, как это выглядит, установите программу */usr/ports/mail/mutt* и попробуйте прочитать какое-нибудь электронное письмо в формате HTML с ее помощью. Если вы пользуетесь почтовым клиентом с графическим интерфейсом, таким как Microsoft Outlook, тогда либо отправляйте свои письма исключительно в текстовом виде, либо убедитесь, что ваше письмо включает в себя и простой текст, и версию сообщения в формате HTML. Все почтовые клиенты *могут* это; остается лишь выяснить, где в недрах графического интерфейса спрятана нужная кнопка. Кроме того, ограничивайте длину строк в сообщении 72 символами. Наличие длинных строк в письме может привести к тому, что оно останется непрочитанным.

Слишком строго? Нисколько, вы почувствуете это, когда вы поймете, кому вы пишете. Большинство почтовых клиентов мало подходят для обработки тысяч сообщений, приходящих ежедневно из десятков почтовых рассылок, в каждой из которых ведутся одновременные обсуждения. Большинство популярных почтовых клиентов облегчают чтение почты, но делают это неэффективно – когда вы получаете такое количество почты, эффективность становится важнее простоты. Большинство тех, кто подписался на почтовые рассылки, находятся в похожей ситуации, поэтому простой текст для них гораздо удобнее.

Аналогичное замечание можно сделать по поводу вложений – вложения в большинстве случаев не нужны. Вам не следует использовать OpenPGP при отправке сообщений в общественный список рассылки, а прикрепленные визитки только лишний раз демонстрируют, что вы

[1] Некоторые разработчики уверяли меня, что несомненно предпочтут свидание с упомянутыми двойняшками всей этой вежливости. Большие мешки денег их тоже устроили бы, желательно набитые непомеченными, крупными купюрами.

не являетесь системным администратором. Не используйте длинные подписи в письмах. Стандартная подпись состоит из четырех строк. Именно – из четырех, каждая из которых имеет длину не более 72 символов. Длинные подписи, с художественным оформлением, следует отбросить.

Во-вторых, оставайтесь в рамках одной темы. Если вы испытываете трудности с X.org, следует зайти на сайт X.org. Если не работает менеджер окон, то спрашивать надо людей, которые его поддерживают. Просить у специалистов по FreeBSD помощи в конфигурировании Java Application Server – все равно, что жаловаться поставщикам аппаратного обеспечения на обед в экспресс-закусочной. Может быть, у них и есть лишний пакетик с кетчупом, но это не их проблема. С другой стороны, если необходимо, чтобы система FreeBSD не запускала sendmail при каждой загрузке, то эта проблема действительно имеет отношение к FreeBSD.

Отправка письма

Как только вы составите письмо, содержащее всю необходимую информацию и вопрос в вежливой форме, отправьте его по адресу *FreeBSD-questions@FreeBSD.org*. Да, есть другие почтовые рассылки FreeBSD, которые, возможно, посвящены как раз тому, с чем возникли трудности. Однако вопросы новичков почти всегда лучше всего подходят для *FreeBSD-questions*. В течение многих лет я отслеживал другие рассылки и должен заметить, что новички задают вопросы, на которые лучше ответят в *FreeBSD-questions*. Обычно задающих такие вопросы отсылают к *FreeBSD-questions*.

Незаметно мы вернулись к вопросу вежливости. Если сообщение с вопросом о том, какие аппаратные архитектуры поддерживаются системой FreeBSD, отправить в почтовую рассылку по архитектурным вопросам, это лишь вызовет раздражение у тех, кто готов обсуждать *проблемы* различных аппаратных архитектур. Ответ может быть получен, но друзей у вас не прибавится. Наоборот, специалисты, поддерживающие рассылку *FreeBSD-questions*, добровольно решили отвечать на вопросы пользователей вроде вас. Они готовы рассмотреть умные, хорошо и подробно описанные проблемы. Многие из них участвуют в разработке FreeBSD, а некоторые даже являются ключевыми игроками. Одни добровольцы – высококвалифицированные специалисты, другие – новички, уже имевшие дело с подобными проблемами.

Обработка ответов

Ответ на вопрос может быть очень коротким и содержать лишь строку URL или даже всего два слова: *man такой-то*. Если вы получили такой ответ, значит вам следует идти по указанному адресу. Не задавайте дополнительных вопросов, пока не ознакомитесь с информацией по указанному вам адресу. Если у вас появится вопрос по указанному со-

держимому, или вас смущает сама ссылка, рассматривайте это как другую проблему. Сузьте проблему, конкретизируйте ее и задайте вопрос об этом. Страницы руководства и справочники далеки от совершенства и некоторые их части могут казаться противоречащими друг другу, пока вы не начнете понимать их.

Следуйте указаниям. Если в ответе на вопрос запрашивается дополнительная информация, ее следует предоставить. Если неясно, как ее получить, – изучите этот вопрос. Если вы заработаете плохую репутацию, вам уже никто не поможет.

Электронная почта forever

Те из нас, кто общался в Интернете в 80-х годах прошлого века, помнят, что мы рассматривали его как что-то вроде личной игровой площадки. Мы могли говорить что угодно и кому угодно. В конечном итоге все было достаточно эфемерно. Никто не запоминал такие вещи, это было как радио личного диапазона, вы могли быть полным ослом и не испытывать никаких проблем.

Теперь все стало не так. Точнее, все стало совсем наоборот. Потенциальные работодатели, потенциальные возлюбленные, даже члены семьи могут отыскивать ваши сообщения в почтовых рассылках или на досках объявлений, пытаясь понять, что вы за личность. Я не раз отказывал людям в найме на работу только на основании их сообщений в почтовых рассылках. Я хочу работать с системными администраторами, умеющими отправлять на форумы поддержки вежливые и грамотные сообщения, а не инфантильно-напыщенные высказывания, не содержащие достаточно информации, чтобы на них можно было дать полезный ответ. И я думаю, что родственников со стороны жены у меня стало бы значительно меньше, если бы я где-нибудь столкнулся с сообщением от одного из них, в котором он выглядел бы круглым дураком. Почтовые рассылки FreeBSD хранятся в архиве достаточно долго, поэтому старайтесь подбирать слова и выражения, потому что их будут читать в течение многих десятилетий.

Теперь, когда вы знаете, как обращаться за помощью, если что-то пойдет не так, давайте приступим к установке FreeBSD.

2

Установка FreeBSD

Запустить FreeBSD на компьютере недостаточно, независимо от того, насколько удовлетворительным был первый запуск. Важно, чтобы установка прошла успешно. *Успешно* – означает, что система должна быть сконфигурирована в соответствии с ее назначением. К веб-серверу, почтовому серверу, настольной системе или серверу базы данных предъявляются различные эксплуатационные требования, а соответствие этим требованиям можно запланировать заранее. Правильный подбор аппаратного окружения делает установку FreeBSD более легкой. Недостаток такого подхода – вы получите меньше опыта в переустановке системы, потому что вам достаточно будет выполнить ее всего один раз. Если ваша единственная цель состоит в овладении процессом установки, можете пропустить эти скучные размышления и сразу перейти в середину главы.

Я предполагаю, что вы намереваетесь использовать FreeBSD в реальной жизни, для решения реальных задач, в реальном окружении. Этим реальным окружением может быть даже ваш ноутбук. Вы можете возразить, сказав, что ноутбук нельзя рассматривать как реальную систему, в ответ на это я предлагаю вам удалить с вашего ноутбука все данные, не создавая резервных копий, и повторить мне то же самое еще раз. Если вы используете свой компьютер исключительно для экспериментов и не заботитесь о сохранности ваших данных, я все равно рекомендую вырабатывать в себе хорошие привычки.

Оцените аппаратные средства, которые имеются в наличии и которые могут потребоваться. Затем решите, как лучше использовать эту аппаратуру, какие части потребуются для установки FreeBSD и как поделить жесткий диск на разделы. Только после этого можно будет загрузить компьютер и установить FreeBSD. Наконец, выполните настройки после установки, и ваша система готова к работе!

Аппаратное обеспечение FreeBSD

Операционная система поддерживает большое число аппаратных платформ, включая различные архитектуры и устройства для каждой из архитектур. Одна из целей проекта состоит в том, чтобы обеспечить поддержку как можно более широкому спектру аппаратного обеспечения. Список поддерживаемых аппаратных средств неуклонно расширялся на протяжении последних лет и теперь включает в себя много больше, чем просто «персональный компьютер». На сегодняшний день FreeBSD поддерживает следующие архитектуры:

amd64 64-битовое расширение 32-битового процессора i386, скопированный компанией Intel как EM64T, и иногда называемый x64. На этой архитектуре могут работать как 32-битовые, так и 64-битовые версии FreeBSD. (В Linux эта платформа называется x86-64.)

i386 Старый добрый Intel-совместимый персональный компьютер.

powerpc Процессор PowerPC используется в старых компьютерах фирмы Apple и во многих встраиваемых устройствах.

pc98 Архитектура, похожая на i386, получившая широкое распространение в Японии.

sparc64 Используется в высокопроизводительных серверах компании Sun Microsystems.

xbox Да! FreeBSD в состоянии работать даже на игровых приставках Xbox производства компании Microsoft.

Система FreeBSD поддерживает множество сетевых карт, контроллеров жестких дисков и других устройств в каждой из архитектур. Поскольку в большинстве из упомянутых архитектур используются похожие интерфейсы сопряжения с аппаратными устройствами, то выбор типа устройства становится не таким уж важным. Устройство с интерфейсом SCSI останется таковым в любой архитектуре, а сетевая карта Intel Ethernet не приобретет волшебных качеств только потому, что кто-то вставит ее в компьютер с архитектурой sparc64.

По большей части FreeBSD мало заботится о типе аппаратного окружения, главное, чтобы оно было работоспособным. Большинство читателей уже знакомо с архитектурой i386, поэтому основной упор будет сделан на эту архитектуру. Однако в последнее время быстро набирает популярность архитектура amd64, поэтому мы коснемся и ее, а также затронем архитектуру sparc64.

Операционная система FreeBSD была перенесена и на множество других платформ, например ARM и Intel Itanium. Эти версии системы либо недостаточно полны, либо используются узким кругом разработчиков. Несмотря на широкое распространение устройств с архитектурой ARM, работающих под управлением FreeBSD, тем не менее, вы не сможете пойти в магазин и купить такое устройство, чтобы поэкспериментировать с ним.

FreeBSD замечательно работает на устаревшем оборудовании, главное, чтобы это оборудование находилось в исправном состоянии. Если ваш старый Pentium постоянно «падает» из-за того, что микросхемы оперативной памяти выходят из строя, установка FreeBSD не прекратит эти «падения».

Пример аппаратного окружения

Эта книга была написана с использованием следующего аппаратного окружения:

- Ноутбук на базе двухъядерного процессора amd64, SATA Sager 9750
- Двухпроцессорный Opteron
- Система для архитектуры i386 на базе Pentium 800
- Плата Soekris net4801
- Sun Ultra 1
- Внешний дисковый массив SCSI

Поблагодарите этих людей

Большая часть этих аппаратных средств была подарена теми, кому пришлось по душе первое издание книги. Их имена приводятся в начале книги. Если эта книга окажется для вас полезной, я буду признателен, если вы купите кому-нибудь из них выпить, закусить или автомобиль Мазерати. Без их помощи я не смог бы получить неисправные устройства. Без неисправных устройств, использованных для тестирования надежности, я не смог бы изучить реальные ограничения FreeBSD, особенно после того, как мой босс очень доходчиво объяснил, что клиенты, которые платят деньги, не оценят мои исследовательские порывы.

Патентованное аппаратное обеспечение

Некоторые производители аппаратных средств полагают, что удержание в секрете интерфейсов их аппаратных средств будет препятствовать возможности копирования конкурентами и их проникновению на рынок. Порочность этой идеи уже ни у кого не вызывает сомнений, особенно в последние годы, когда поток универсальных аппаратных элементов буквально растоптал этих скрытничающих производителей. И все же некоторые производители, особенно те, что выпускают видео- и звуковые карты, продолжают придерживаться этой стратегии.

Разработка драйверов устройств с засекреченными спецификациями интерфейсов – дело весьма трудоемкое. Некоторые устройства могут неплохо поддерживаться и без полной документации, а если такие устройства получили очень широкое распространение, то есть смысл за-

трачивать усилия на разработку драйверов и при нехватке документации. В частности группа разработки драйверов звуковых карт для FreeBSD проделала огромную работу по анализу интерфейсов звуковых карт и реализовала универсальную инфраструктуру, которая прекрасно работает даже с плохо документированными звуковыми картами. Для других аппаратных средств, таких как комплект микросхем, используемый в шине PCI системы UltraSPARC III, очень сложно обеспечить поддержку без полной документации.

Если у разработчиков FreeBSD будут в наличии полные спецификации аппаратных устройств и заинтересованность в их использовании, они наверняка смогут обеспечить их поддержку. В противном случае эти устройства не будут работать с FreeBSD. В большинстве случаев неподдерживаемые патентованные устройства могут быть заменены менее дорогими и более открытыми аналогами.

Некоторые производители сами разрабатывают и предоставляют драйверы к своим устройствам. Например, компания Nvidia предлагает драйверы к своим видеоустройствам. Кроме того, во FreeBSD используются некоторые ухищрения, с помощью которых имеется возможность использовать драйверы сетевых карт, разработанных для операционной системы Windows, например для беспроводных сетевых карт, поддерживаемых проектом «Project Evil» (проект дьявола).[1] Однако самую лучшую поддержку предоставляют драйверы FreeBSD с открытыми исходными текстами.

Поддерживается ли мое аппаратное обеспечение?

Самый простой способ найти ответ на этот вопрос – заглянуть в примечания к выпуску той версии FreeBSD, которую вы планируете установить. Примечания к выпуску можно найти на сайте *http://www.freebsd.org*.

О чем не будет говориться в книге

Мы не рассказываем о картах с интерфейсом ISA – повсеместное распространение в течение последних 10 лет получили карты с интерфейсом PCI, и я очень сомневаюсь, что кто-то до сих пор использует карты ISA.[2]

---

[1] Да, этот проект действительно носит название «Project Evil» (проект дьявола). Его усилиями реализован программный интерфейс Windows в ядре FreeBSD, что оправдывает название проекта.

[2] Если вы используете эти карты или достаточно давно имеете с ними дело, то, скорее всего, вы даже не будете читать эту книгу или дойдете до всего своим умом. Замечу только, что последнее не недостаток.

В Справочнике (Handbook) FreeBSD имеется достаточный объем инструкций, как заставить работать карты ISA.

PowerPC и pc98 – это уже устаревшие архитектуры и сейчас находятся в упадке, поэтому мы также не будем обсуждать их. Подобно динозаврам, старые серверы очень сложно уничтожить, разве только в результате удара метеорита. Запуск FreeBSD на Xbox – это скорее забава, которая больше напоминает трюкачество, чем идею, представляющую какую-либо ценность.

Аппаратные требования

FreeBSD предъявляет самые минимальные требования к аппаратному окружению, и, тем не менее, наилучшие результаты будут достигнуты на системах с определенной конфигурацией. Вот несколько основных рекомендаций для систем i386, которые, впрочем, мало отличаются и для других платформ.

В главе 19 обсуждаются вопросы измерения производительности системы, что позволяет максимально использовать возможности аппаратуры.

Процессор

Для работы FreeBSD марка процессора несущественна. Неважно, будет ли это процессор Intel, AMD, IBM или Cyrix/Via. Тип процессора выясняется при загрузке, а далее FreeBSD учитывает все особенности микропроцессора, которые были распознаны. Когда-то серверы эффективно работали на машинах с процессорами i486 и даже переполняли каналы T1, предоставляющие доступ к Интернету. Однако все-таки лучше применять процессоры Pentium или более быстрые. Прогон некоторых примеров из этой книги занял бы несколько дней в системе с процессором i486, но я теперь не настолько терпелив. Те же самые операции на моем ноутбуке с двухъядерным процессором выполняются менее чем за час.

Память

Прежде всего, память (RAM) – это хорошо, и чем больше памяти, тем лучше. Добавление микросхем RAM увеличит быстродействие системы заметнее, чем модернизация любых других компонентов. Я рекомендую иметь по меньшей мере 64 Мбайта RAM, но если у вас имеется 256 Мбайт или больше, вы увидите, с какой легкостью станет работать FreeBSD. Если вы пытаетесь сократить объем памяти до минимума, то вы сможете запустить ядро на 16 Мбайтах, но с таким объемом памяти вы не сможете запустить процесс установки.

Жесткие диски

Жесткие диски могут стать узким местом производительности. Несмотря на то что диски IDE очень дешевы, их производительность ниже, чем у дисков SAS, SCSI или даже SATA. Система SAS или устарев-

шая SCSI может на полной скорости передавать данные между контроллером и каждым из установленных дисков, тогда как скорость обмена данными с контроллером IDE или SATA тем меньше, чем больше дисков подключено к контроллеру. Кроме того, контроллер SCSI может обслуживать до 15 дисков, а контроллер IDE – не больше двух. Контроллер SATA допускает подключение к одному каналу всего одного диска. Пятнадцать дисков, работающих на полной скорости, или два диска с «половинным» быстродействием – большая разница для пропускной способности!

Как бы то ни было, при наличии дисков IDE их лучше подсоединять к отдельным контроллерам. Сегодня во многих системах жесткий диск подсоединен к одному контроллеру IDE, а привод CD-ROM – к другому. Добавляя второй жесткий диск, подключите его ко второму контроллеру. Как-никак, обращение к CD-ROM происходит реже, чем к жесткому диску.

Базовая установка FreeBSD вполне уместится в 500 Мбайт дискового пространства, а сильно урезанная версия – в 32 Мбайта. Наличие в системе 5 Гбайт дискового пространства вполне вас удовлетворит, но я предполагаю, что у вас имеется хотя бы 10 Гбайт. Для сборки некоторых программных продуктов необходимо иметь значительный объем свободного дискового пространства, например, для сборки пакета OpenOffice требуется 10 Гбайт в разделе */usr*. Впрочем, любой достаточно новый жесткий диск наверняка будет иметь больший объем.

Подготовка к установке

Прежде чем приступать к установке, необходимо решить, для каких целей будет использоваться система. Будет ли это веб-сервер? Сервер баз данных? Сервер регистрации в сети? Требования для каждого из них мы рассмотрим в соответствующих разделах.

Создание разделов на жестком диске

Разделы – это логические диски. Система FreeBSD может работать с самыми разными разделами и даже допускает существование разных файловых систем или операционных систем на разных разделах. Если вы впервые устанавливаете FreeBSD, у вас наверняка не будет каких-то определенных требований к разделам жесткого диска, поэтому вы можете просто воспользоваться процедурой автоматического разбиения, предлагаемой инсталлятором. Если же у вас имеются какие-то определенные требования, я рекомендую записать их на листе бумаги прежде, чем вы приступите к установке.

Процесс создания разделов может вызвать головную боль. Если вы знакомы с какими-либо другими UNIX-подобными операционными системами, такими как Linux, у вас может появиться желание создать один большой корневой раздел и поместить в него все. Если Windows

или Linux позволяют держать все на одном большом диске, тогда зачем для FreeBSD необходимо создавать несколько разделов меньшего размера? В чем преимущества такого деления?

С физической точки зрения разные части диска имеют разную скорость чтения данных. Поместив в самый быстрый раздел диска данные, к которым приходится обращаться очень часто, вы тем самым оптимизируете производительность системы. Единственный способ достичь этого эффекта – использовать несколько разделов. С логической точки зрения FreeBSD работает с каждым из разделов по отдельности. Это означает, что для каждого раздела можно определить свои правила работы. Разделы, где хранятся данные пользователя, не должны содержать *setuid-программ* (программы, которые запускаются с привилегиями суперпользователя root), и для вас может быть желательно, чтобы в них вообще не было таких программ. Этого легко можно добиться, используя разделение диска.

При повреждении данных на диске высока вероятность, что эти повреждения будут ограничены единственным разделом. Вы сможете загрузить систему из неповрежденного раздела и попытаться восстановить данные в поврежденном разделе. При использовании единственного большого раздела любые повреждения будут затрагивать всю систему в целом, уменьшая шансы на ее восстановление.

Наличие разделов поможет минимизировать повреждения, вызванные недостаточным вниманием администратора системы. Так, неконтролируемые программы могут заполнить диск файлами журналов. Наличие большого раздела не означает, что проблема не будет проявлять себя длительное время – это означает лишь, что файлы журналов будут иметь большие объемы. В главе 19 обсуждаются способы сопровождения файлов журналов, потому что переполнение жесткого диска может даже препятствовать запуску системы, который необходим, чтобы ликвидировать появившиеся проблемы! Разбиение диска на разделы сводит действие таких проблем к минимуму.

Наконец, многие программы создания резервных копий, например dump(8), работают с целыми разделами. На промышленной системе вы наверняка захотите определить разные стратегии резервирования для данных различных типов. В системе FreeBSD стандартными являются следующие разделы: / (root, или корневой раздел), пространство свопинга, */var*, */tmp* и */usr*.

/ (root)

Корневой раздел содержит базовые конфигурационные файлы системы, ядро, а также основные утилиты UNIX. Все остальные разделы лежат в корневом разделе или связаны с ним. При наличии неповрежденного корневого раздела вы сможете загрузить систему в однопользовательский режим и выполнить восстановление остальной части системы. Системе необходим быстрый доступ к корневому разделу, поэтому

раздел root следует разместить в первую очередь. Поскольку корневой раздел содержит только основные утилиты и конфигурационные файлы, он не должен быть слишком большим – по умолчанию FreeBSD выделяет под корневой раздел 512 Мбайт, этого более чем достаточно.

Пространство свопинга

Следующий раздел жесткого диска – это *пространство свопинга* – дисковое пространство, используемое подсистемой виртуальной памяти. Когда физическая память заполнена, система перемещает в раздел свопинга информацию, которая не была задействована некоторое время. Если в системе все идет как надо, ей вообще не нужно пространство свопинга, но если оно потребуется, система должна иметь как можно более быстрый доступ к нему.

Итак, каков необходимый объем пространства свопинга? Это предмет долгих дискуссий системных администраторов. Короткий ответ – «зависит от системы». Старая мудрость гласит, что объем пространства свопинга должен, по крайней мере, вдвое превышать емкость физической памяти. Однако эта мудрость давно уже устарела и возможности современных систем лишили это эмпирическое правило силы. Когда процесс теряет контроль и начинает потреблять память (например, в бесконечном цикле), ядро уничтожит его, как только будет исчерпана виртуальная память. Если в вашей системе имеется 6 Гбайт RAM и 9 Гбайт пространства свопинга, процесс должен потребить 15 Гбайт памяти, прежде чем ядро уничтожит его! В системах на базе процессора i386 адресное пространство виртуальной память составляет примерно 3 Гбайта, и это пространство совместно используется ядром, разделяемыми библиотеками, стеком и т. д. Платформа i386 ограничивает процесс объемом памяти в 512 Мбайт, то есть ядро остановит разбушевавшийся процесс достаточно быстро. В 64-битовых системах, таких как amd64, виртуальное адресное пространство намного больше, поэтому вышедший из-под контроля процесс может потребить гигабайты памяти. Когда система начинает интенсивно перекидывать гигабайты данных между RAM и пространством свопинга, она перестает откликаться, становится медлительной и очень неудобной. В наше время желательно иметь достаточный объем пространства свопинга. Я рекомендую выделить под пространство свопинга объем, равный объему физической памяти или даже на несколько мегабайтов больше.

Основное назначение пространства свопинга в современных системах состоит в том, чтобы предоставить место для хранения дампа физической памяти на случай краха системы. Для обеспечения максимальной безопасности необходимо, чтобы пространство свопинга могло вмещать в себя полный объем физической памяти. Но это в самом тяжелом случае. Современные же системы FreeBSD, версии 7.0 или выше, по умолчанию создают дамп меньшего размера, сохраняя только память ядра. Такой минидамп существенно меньше, чем полный дамп физической памяти, – в системе с 8 Гбайт RAM средний размер мини-

дампа составляет порядка 256 Мбайт. Вам наверняка удастся избежать неприятностей, выделив под пространство свопинга 1 Гбайт, чего с лихвой хватит даже для сильно раздутого минидампа.

/tmp

Каталог */tmp* – это общесистемный временный каталог, доступный для всех пользователей в системе. Если вы не создадите отдельный раздел для */tmp*, он будет размещаться в корневом разделе. То есть пространство для хранения временных файлов будет подчиняться тем же условиям, что и весь корневой раздел. Скорее всего, это не то, что вам хотелось бы получить, если вы планируете разрешить доступ к корневому разделу только для чтения.

Требования, предъявляемые к каталогу */tmp,* – это скорее вопрос личных предпочтений. В конечном итоге вы всегда сможете использовать для хранения временных файлов часть пространства, выделенного для домашнего каталога, а кроме того, имеется каталог */var/tmp*, на случай, если появится необходимость работать с временными файлами большого размера. Я предпочитаю выделять для каталога */tmp* пространство объемом хотя бы 512 Мбайт. Программы-инсталляторы нередко стремятся извлекать файлы в каталог */tmp*, но работать с такими инсталляторами при переполнении каталога */tmp* хотя и возможно, но очень утомительно.

В системах, где не предполагается наличие каталога */tmp* большого объема (например, веб-серверы или серверы баз данных), пространство для */tmp* можно выделять в памяти. Эта возможность будет рассматриваться в главе 8. Если вы предполагаете выделять место для каталога */tmp* в памяти, то не создавайте этот раздел.

/var

Раздел */var* хранит быстро изменяющиеся данные, такие как файлы журналов, почтовые ящики, временные рабочие файлы, файлы обновлений таких инструментальных средств, как portsnap и FreeBSD-update и т. п. Если система выполняет функции веб-сервера, файлы журналов веб-сайта будут размещаться в этом разделе. Для этого раздела вполне может потребоваться 2 Гбайта или более. На маленьких почтовых или веб-серверах я использую треть всего дискового пространства под каталог */var*. Если система предназначена для выполнения функций почтового сервера или сервера баз данных, я увеличил бы эту долю до 70% и даже выше или просто выделил бы сначала достаточное пространство для других разделов, а все остальное отвел бы под раздел */var*. Если вы сильно ограничены в дисковом пространстве, можно выделить под этот раздел достаточно небольшой объем дискового пространства, например 30 Мбайт.

Желательно, чтобы по своему размеру раздел */var* намного превышал объем физической памяти. По умолчанию, в случае краха системы,

FreeBSD записывает дамп памяти в файл */var/crash*. Подробнее дампы памяти мы будем рассматривать в главе 21, а пока поверьте мне на слово – если раздел */var* будет иметь достаточно большой объем, чтобы вместить содержимое физической памяти, это существенно поможет вам даже в случае самых серьезных неполадок в системе.

/usr

Раздел */usr* хранит программы операционной системы, исходный код, компиляторы, библиотеки, дополнительное программное обеспечение и другие подобные файлы, которые обеспечивают выполнение фактических функций системы. В большинстве случаев изменения в этом разделе происходят только в процессе обновления системы. Здесь также находятся домашние каталоги пользователей, изменения в которых происходят достаточно часто. Если в вашей системе предполагается большое число пользователей, создайте отдельный раздел */home*. Вы, конечно, можете определить квоты на использование дискового пространства, но отдельный раздел позволит вам защитить важные файлы операционной системы.

При наличии современного жесткого диска я рекомендую выделять для раздела */usr* как минимум 6 Гбайт. Этого вполне хватит для работы операционной системы, хранения основных файлов с исходными текстами и сборки обновлений до следующей версии FreeBSD. В случае веб-сервера, где пользователи имеют возможность выгружать файлы в свои домашние каталоги, я предложил бы выделить под этот раздел большую часть жесткого диска.

Остальные разделы

Опытные системные администраторы всегда имеют свою предпочтительную схему разбиения жесткого диска. Кроме того, некоторые компании определяют свои, стандартные схемы разбиения. Различные производители операционной системы UNIX пытались протолкнуть свои стандарты деления диска на разделы. В различных версиях UNIX вы сможете увидеть такие разделы, как */opt* и */u1*.

Если у вас имеется своя предпочтительная схема разбиения, используйте ее. Вы можете указать FreeBSD, куда устанавливать дополнительное программное обеспечение, по своему выбору. Вы также можете разместить домашние каталоги пользователей, например, в каталоге */gerbil*, если это сделает вас счастливым. Лучший совет, какой я могу дать читателям, с которыми я никогда не встречусь и к чьим системам я не буду иметь никакого отношения: вам предстоит жить с вашими разделами, поэтому сначала думайте!

Несколько жестких дисков

Если у вас более одного жесткого диска, они сравнимы по качеству и не используются в RAID, вы можете найти им прекрасное примене-

ние: разместите на одном диске свои данные, а на другом – операционную систему. Один из разделов будет содержать информацию, специфичную для сервера. Сервер баз данных хранит свои данные в каталоге */var*, поэтому можно поместить раздел */var* на отдельный диск. Если это веб-сервер, можно поместить на второй диск раздел */usr*.

Если сервер выполняет какие-то специализированные функции, можно создать скрытый раздел специально для этих функций. Нет ничего неправильного в том, чтобы создать раздел */home*, */www* или */data* на втором диске и выделить целый диск для нужд системы.

Вообще отделение операционной системы от пользовательских данных повысит эффективность системы. Как и другие эмпирические правила, такая стратегия является спорной. Но ни один сисадмин не скажет, что это плохая или опасная идея.

При наличии нескольких дисков можно повысить эффективность пространства свопинга, расположив его на нескольких дисках. Подключите первый диск с разделом свопинга ко второму слоту контроллера, к которому подключен диск с корневым разделом, а остальные диски с разделами свопинга – к первым слотам других контроллеров дисков. Это позволит распределить операции чтения/записи между несколькими контроллерами и даст вам дополнительный выигрыш на уровне контроллеров. Однако не забывайте, что дамп памяти, создаваемый в случае краха системы, должен целиком умещаться в единственный раздел свопинга.

Наибольшего выигрыша от распределения пространства свопинга можно достичь в случае использования дисков SAS или SCSI. Если у вас используются приводы IDE или SATA, для достижения лучших результатов они должны располагаться на разных контроллерах IDE. Не забывайте, что контроллер IDE делит общую пропускную способность среди всех подключенных к нему жестких дисков. Если у вас имеется два жестких диска, подключенных к одному и тому же контроллеру IDE, и одновременно производится обращение к обоим дискам, каждый диск будет работать в среднем с половиной той скорости, как если бы он работал один на том же самом канале. Самое узкое место в использовании пространства свопинга – это скорость работы диска, и поэтому не стоит устраивать борьбу между дисками за обладание шиной IDE.

Другой способ использования нескольких дисков состоит в том, чтобы на их основе реализовать программный RAID. Это защитит вас от отказов жестких дисков, т. к. копии данных будут распределены по нескольким дискам. Возможности FreeBSD по созданию дисковых массивов RAID мы рассмотрим в главе 18. Чтобы реализовать программный массив RAID, «нарезка» всех дисков на разделы должна быть одинакова. Проще всего этого добиться, когда все диски имеют одинаковый объем, но это совершенно не обязательно.

Размер блока раздела

В этом разделе описываются параметры, которые могут снизить производительность системы. Если вы не знакомы с FreeBSD, прочитайте этот раздел *только* для ознакомления – не пытайтесь изменять описываемые здесь значения! Эти параметры предназначены для опытных администраторов UNIX, которые *точно знают*, что они делают.

Размер блока определяет минимальный размер строительных блоков файловой системы, которые используются для хранения файлов. Каждый блок может делиться на фрагменты. Во FreeBSD размер блока по умолчанию равен 16 Кбайт (16 384 байтов), а размер фрагмента – 2 Кбайта (2 048 байтов). Для размещения файлов используется комбинация фрагментов и блоков. Например, файл размером 15 Кбайт займет один блок, а файл размером 17 Кбайт – один блок и один фрагмент. Более подробно о блоках и фрагментах мы поговорим в главе 18.

Если вы четко представляете, что делаете, и хотите изменить размер блока, вы можете сделать это во время установки. Будьте внимательны, установлено, что наиболее оптимальное поведение FreeBSD обеспечивается, когда один блок содержит восемь фрагментов – вы можете выбрать любое другое соотношение, отличное от 1:8, но за это придется заплатить производительностью.

Выбор варианта установки

Вариант установки – это подмножество системы FreeBSD. В ходе установки вы можете выбрать один или более вариантов. Вы можете доустановить необходимые компоненты позднее, но проще и лучше предусмотреть установку всего необходимого с самого начала. Инсталлятор предлагает девять вариантов:

All Содержит абсолютно все, что входит в состав FreeBSD, включая X Window System. (В системе FreeBSD используется версия X.org.) Если установка производится на тестовую машину, выберите этот вариант.

Developer Включает в себя все, за исключением игр и X.

X-Developer Включает в себя все, за исключением игр.

Kern-Developer Включает программы FreeBSD и документацию, а также исходный код ядра.

X-Kern-Developer Включает в себя те же компоненты, что и вариант Kern-Feveloper, плюс X Window System.

User Включает в себя только программы операционной системы FreeBSD и документацию – никаких исходных текстов и X.

X-User Включает в себя те же компоненты, что и вариант User, плюс X.

Minimal Содержит только основные программы FreeBSD без каких-либо исходных текстов и без документации. Этот вариант подходит, если жесткий диск имеет очень маленький объем.

Custom Предоставляется возможность самостоятельного выбора устанавливаемых пакетов.

Если вы устанавливаете FreeBSD с целью ознакомления с ней, вам определенно следует выбрать вариант All. Для сервера Интернета лучшим выбором будет вариант User или даже X-User, если вы уже знакомы с X Window System. Опытные пользователи могут выбрать вариант Custom.

Игры?

Да, FreeBSD включает в себя простейшие игры. Это маленькие игровые программы, работающие в текстовой консоли, которые были типичны для систем лет 20 тому назад. Начинающие пользователи FreeBSD могут найти полезные советы на странице руководства fortune(6), но если вам интересны более современные игры, загляните в */usr/ports/games* и прочитайте главу 11.

> **X Window System**
>
> X Window System – это стандартный графический интерфейс для UNIX-подобных операционных систем. Если вы не предполагаете постоянно сидеть за текстовой консолью и выполнять в ней повседневную работу, вам наверняка потребуется X Window System. Если же вы не собираетесь путешествовать по Интернету или выполнять другие действия, требующие наличия графического интерфейса, X Window System вам скорее всего не понадобится. В крайнем случае вы всегда сможете доустановить X Window System позднее.

FTP-сайт FreeBSD

Так же, как основным источником информации о FreeBSD является веб-сайт FreeBSD, основным источником получения самой операционной системы является FTP-сервер FreeBSD. Вы можете купить компакт-диски с FreeBSD и тем самым сделать приличные инвестиции, но многие предпочитают получать все необходимое с помощью Интернета. Даже если у вас уже имеется компакт-диск, вам все равно придется взаимодействовать с FTP-сервером.

Основной FTP-сервер FreeBSD находится по адресу *ftp.freebsd.org*, но в мире существует множество зеркал, которые помогают снизить нагрузку на основной сервер и обеспечивают быстрый и надежный дос-

туп. Полный список зеркал можно найти на www.FreeBSD.org. Впрочем, зеркала можно легко выбрать и без списка. Имена зеркальных серверов соответствуют следующему шаблону:

```
ftp<number>.<country>.freebsd.org
```

Код страны (`country`) может отсутствовать, и тогда под местом расположения сервера обычно подразумевается континентальная часть США, например *ftp14.FreeBSD.org*, *ftp2.uk.FreeBSD.org*, *ftp5.ru.FreeBSD.org* и т. д.

Как правило, FTP-зеркала с меньшими номерами более загружены, чем серверы с большими номерами. Можно попробовать *ftp12.freebsd.org* или сервер с высоким номером, имя которого включает в себя код страны, и посмотреть, насколько быстрым будет соединение.

Содержимое FTP-сервера

Многие серверы, являющиеся зеркалами FreeBSD, содержат и другое программное обеспечение. На таких серверах FreeBSD находится в каталоге */pub/FreeBSD*. Вот как может выглядеть его содержимое:

```
CERT
ERRATA
ISO-IMAGES-amd64
ISO-IMAGES-i386
ISO-IMAGES-ia64
ISO-IMAGES-pc98
ISO-IMAGES-ppc
ISO-IMAGES-sparc64
README.TXT
distfiles
doc
ports
releases
snapshots
tools
torrents
```

Сколько тут всего! К счастью, на большую часть файлов можно не обращать внимания, но на некоторые каталоги следует обратить особое внимание:

CERT Этот каталог содержит рекомендации по обеспечению безопасности в FreeBSD, начиная с момента основания проекта. Подробнее об этих рекомендациях мы поговорим в главе 7.

ERRATA Этот каталог содержит информацию обо всех ошибках, обнаруженных в различных версиях системы. Об ошибках мы подробнее поговорим в главе 13.

ISO-IMAGES Все каталоги, имена которых начинаются с *ISO-IMAGES*, содержат образы компакт-дисков с дистрибутивами FreeBSD

для различных архитектур. Например, ISO-IMAGES-386 содержит образы дисков в формате ISO для установки FreeBSD на платформу i386. Вы можете записать образы на компакт-диски и выполнить установку с них. (Сделать это вам поможет документация к вашему устройству для записи компакт-дисков.)

README.TXT В этом файле находится описание различных каталогов на FTP-сайте и их содержимое. Вы можете заглянуть в этот файл, чтобы ознакомиться с изменениями, произошедшими с момента подготовки данной книги к печати.

distfiles В этом каталоге содержится достаточно много исходных текстов и двоичных файлов программ сторонних разработчиков, созданных для работы под управлением FreeBSD. Это, пожалуй, самый крупный каталог на FTP-сервере FreeBSD.org. Только не пытайтесь загрузить к себе все файлы из этого каталога, иначе ваши диски переполнятся.

doc В этом каталоге содержится самый свежий комплект документации к FreeBSD на разных языках. Если вы читаете эту книгу на английском языке, скорее всего вам следует заглянуть в подкаталог с именем *en* (English).[1] Здесь вы найдете все статьи и книги в заархивированном виде, доступные для загрузки.

ports В этом каталоге вы найдете полную инфраструктуру и комплект пакетов системы портов. Подробнее порты будут рассматриваться в главе 11.

releases В этом каталоге находятся самые свежие версии FreeBSD, выпущенные в различных ветках версий. Старые версии системы можно найти на сервере *ftp-archive.freebsd.org*. Подробнее о ветках версий мы поговорим в главе 13.

snapshots В этом каталоге находятся самые свежие версии FreeBSD-current и FreeBSD-stable. Здесь вы найдете самые последние тестовые и готовые к промышленной эксплуатации версии.

tools Здесь расположены разнообразные программы для операционной системы Windows, которые могут использоваться для подготовки системы к установке FreeBSD в качестве второй операционной системы на компьютере.

torrents Этот каталог будет полезен пользователям BitTorrent – он содержит torrent-файлы для загрузки самых свежих версий FreeBSD. (Если вы еще не знакомы с BitTorrent, вам определенно стоит с ним познакомиться.)

Теперь, когда вы знаете, что и где искать, можно начинать процесс установки.

---

[1] Документация на русском языке находится в подкаталогах *ru* и *ru_RU.KOI-8*. – *Прим. перев.*

> **Какую версию выбрать?**
>
> Операционная система FreeBSD имеет множество версий, и продолжают выходить новые версии. Подробнее номера версий и выпусков мы обсудим в главе 13. А пока я предлагаю посетить сайт *http://www.freebsd.org*. На главной странице найдите ссылку Production Release с номером версии. Используйте эту версию.

Процесс установки

Самое интересное[1] в новой операционной системе – это выяснить, как можно запустить ее на компьютере. Для большинства современных операционных систем сделать это довольно просто: достаточно вставить компакт-диск в привод CD-ROM и загрузиться с него. Однако FreeBSD может устанавливаться на очень древние компьютеры, которые не поддерживают возможность загрузки с компакт-диска. Но и это не проблема – можно выполнить загрузку с дискеты. Система FreeBSD может также устанавливаться на самые современные компьютеры, в которых нет ни накопителя на гибких магнитных дисках, ни CD-ROM. Что тогда?

Процесс установки любой операционной системы делится на три этапа: загрузка программы установки, доступ к носителю с дистрибутивом и копирование программного обеспечения на жесткий диск. Даже инсталлятор Windows сначала загружает «мини-Windows», чтобы установить саму систему Windows. FreeBSD также предлагает различные варианты прохождения каждого из этих этапов. Если компьютер загрузился, и у вас наготове пригодные для использования носители с дистрибутивом, установка программного обеспечения на диск проходит без затруднений.

Выбор загрузочного носителя

Если ваш компьютер способен загружаться с компакт-диска – это самый простой способ выполнить установку. Вы можете приобрести компакт-диски с FreeBSD у различных производителей или загрузить их образы с FTP-сайта. Убедитесь, что в BIOS компьютера, в настройках очередности загрузки компакт-диск стоит перед жестким диском, вставьте диск с дистрибутивом в лоток привода CD-ROM и перезагрузите компьютер. Если вам потребуется помощь в настройке параметров BIOS компьютера, обращайтесь к документации производителя.

---

[1] Древнее китайское проклятие, которое звучит как: «Чтоб вам жить в эпоху перемен», – определенно подходит и для этого случая.

Если ваш компьютер не может выполнять загрузку с компакт-диска, но может загружаться с дискет, загрузите образы дискет из Интернета и загрузитесь с них. Многие старые модели компьютеров имеют приводы CD-ROM, но не могут использовать их в качестве загрузочных устройств, но как только система загрузится, вы сможете использовать их для доступа к файлам дистрибутива, расположенным на компакт-диске.

Многие современные компьютеры не имеют ни накопителя на гибких магнитных дисках, ни CD-ROM. Это часто случается с минисерверами,

Нет устройств на сменных носителях?

Если на компьютере, куда вы собираетесь установить FreeBSD, отсутствуют накопитель на гибких магнитных дисках или CD-ROM, если по каким-либо причинам отсутствует возможность подключения CD-ROM или загрузки с устройства USB, и вы не знаете, как запустить инсталлятор PXE (слишком много «если», но под это описание подпадает достаточно много небольших серверов), не отчаивайтесь.

Самое простое, что можно сделать, – это извлечь жесткий диск и подключить его к системе, где имеется устройство на сменных носителях. В отличие от некоторых других операционных систем, FreeBSD позволяет выполнить установку на одной машине, а запускать ее на другой.

Если этот вариант вам не подходит, можно использовать еще одну уловку, которую я много раз использовал на практике. (Она может привести к тому, что аппаратные компоненты или вы сами поджаритесь, как на электрическом стуле, и, конечно же, лишит вас права на гарантийный ремонт. Автор не несет ответственность за барбекю из аппаратуры или системных администраторов!)

Отыщите старый компьютер, работающий под любой операционной системой, с приводом IDE CD-ROM. Поставьте рядом с ним ваш компьютер, куда будет установлена FreeBSD, выключите его и откройте корпус. Откройте корпус вашего компьютера. В старом компьютере отключите кабель IDE, идущий к приводу CD-ROM, от разъема на контроллере. Кабель питания оставьте подключенным к CD-ROM. Подключите кабель к контроллеру IDE вашего компьютера. Включите старый компьютер. При этом включится привод CD-ROM, несмотря на то, что он не подключен к контроллеру IDE старого компьютера. Теперь включите новый компьютер, и он опознает подключенный CD-ROM.

После установки верните все на место, и никто и никогда не узнает о способе установки, который вы нашли.

монтирующимися в стойки, где приходится экономить на пространстве. В этом случае вы можете установить устройство CD-ROM или воспользоваться установкой PXE (Preboot Execution Environment – предварительная загрузка среды исполнения), как описывается в главе 20. (Установка в режиме PXE требует выполнить начальную загрузку с работающей системы FreeBSD.)

Выбор носителя с дистрибутивом

Обычно установка системы производится с компакт-дисков или с FTP.

Компакт-диски прекрасно подходят для случая, когда у вас имеется множество компьютеров и все они снабжены приводами CD-ROM. Установка с компакт-дисков выполняется легко и быстро даже при отсутствующем подключении к сети. Компакт-диски и DVD можно приобрести у разных производителей. В частности, уже много лет FreeBSD поддерживается компанией iX Systems, которая недавно купила компанию FreeBSD Mall – производителя компакт-дисков с дистрибутивами FreeBSD. Диски DVD обладают большей емкостью, чем компакт-диски, и на них присутствуют файлы, которые могут быть загружены из Интернета, тем не менее и компакт-диски содержат все, что вам действительно необходимо. С этого момента я буду говорить только о компакт-дисках, но все это в равной степени относится и к DVD. Если вы не хотите покупать компакт-диски, можете загрузить их ISO-образы с FTP-сервера FreeBSD и записать их у себя.

Образы компакт дисков с дистрибутивом FreeBSD и сопутствующие материалы можно найти на десятках FTP-северов. Инсталлятор FreeBSD может получать программное обеспечение с этих серверов напрямую. Однако, чтобы использовать метод установки через FTP, вам необходимо иметь подключение к Интернету, а кроме того, скорость установки будет в значительной степени зависеть от скорости соединения с выбранным FTP-сервером. Кроме того, есть вероятность, что нашелся злоумышленник, который взломал FTP-сервер и выгрузил на него испорченную версию FreeBSD, впрочем, команда FreeBSD очень внимательно относится к таким случаям и моментально ликвидирует проблемы. Помимо этого группа, выпускающая FreeBSD, предоставляет контрольные суммы к каждой версии, которые вы можете использовать для проверки целостности файлов.

Подготовка загрузочных дискет

Вам потребуется подготовить несколько дискет (на момент написания этих строк их было четыре, но в будущем это число может увеличиться). Найдите в каталоге с выбранной архитектурой подкаталог с требуемой версией системы. Здесь вы найдете подкаталог *floppies*. Например, для архитектуры i386 и версии FreeBSD 7.0 путь к этому каталогу выглядит следующим образом: *ftp://ftp.freebsd.org/pub/freebsd/*

releases/i386/7.0-RELEASE/floppies. (Этот же каталог можно найти на компакт-диске с дистрибутивом FreeBSD.) Здесь имеется несколько файлов с расширением *.ftp*, один из которых называется boot.ftp, а также несколько файлов с именами в формате *kernX.ftp*, например *kern1.ftp* и *kern2.ftp*. Эти файлы представляют собой образы дискет. Загрузите их все.

Далее необходимо записать эти образы на дискеты. Здесь есть одна хитрость – вы не можете просто скопировать эти файлы на дискеты, используя, например, операцию перетаскивания мышью в Windows. Образ должен быть записан на дискету определенным образом.

Если у вас уже имеется рабочая UNIX-подобная система, тогда можно воспользоваться командой `dd(1)`. При этом вы должны знать, как называется устройство, соответствующее накопителю на гибких магнитных дисках, и скорее всего это будет */dev/fd0*, */dev/floppy* или */dev/rfd0*. Если устройство называется */dev/fd0*, как во всех системах BSD, введите команду:

```
# dd if=kern1.flp of=/dev/fd0
```

которая запишет на дискету образ *kern1.ftp*. Скопируйте каждый образ на отдельную дискету.

Если вы работаете с Microsoft Windows, для копирования образов вам потребуется специальная утилита. Компания Microsoft не предоставляет таких утилит, как это делает FreeBSD, но вы сможете найти ее в подкаталоге *tools* на основном сайте. Она называется *fdimage.exe*.

Это свободно распространяемая программа, предназначенная для копирования образов дискет. Используется она очень просто. Утилита принимает два аргумента: имя файла образа и имя диска, куда следует скопировать образ. Например, чтобы скопировать образ *boot.ftp* на дискету в приводе a:, откройте командную строку DOS и введите следующую команду:

```
c:> fdimage boot.flp a:
```

После того как запись на дискету завершится (что может потребовать некоторого времени), повторите процесс для всех остальных образов дискет.

Подготовка загрузочных компакт-дисков

Если вы приобрели официальный компакт-диск с дистрибутивом FreeBSD, то ваш установочный носитель готов к использованию. Если нет – вам необходимо загрузить ISO-образ с FTP-сайта и записать его. Первый этап заключается в том, чтобы отыскать каталог с образом. Зайдите на FTP-сайт и выберите каталог с ISO-образами для вашей архитектуры. В этом каталоге вы найдете каталог для каждой из существующих ныне версий FreeBSD. Например, ISO-образы с дистрибути-

вом FreeBSD 7.0 для архитектуры i386 можно найти по адресу: *ftp://ftp.freebsd.org/pub/freebsd/ISO-IMAGES-i386/7.0*. Здесь находится сразу несколько образов.

Имя ISO-образа составлено из номера версии, слова RELEASE, названия архитектуры и комментария; все эти части разделены дефисами. Например, для версии 7.0 имеются следующие образы дисков:

```
7.0-RELEASE-i386-bootonly.iso
7.0-RELEASE-i386-disc1.iso
7.0-RELEASE-i386-disc2.iso
```

Образ, помеченный как *disc1*, содержит полный дистрибутив FreeBSD, X Window System, несколько основных пакетов и *оперативную файловую систему* (*live filesystem*), которая может использоваться для восстановления системы после аварий.

Образ, помеченный как *disc2*, содержит наиболее популярные программы для FreeBSD, предварительно скомпилированные и готовые к использованию с этой версией системы.

Образ, помеченный как *bootonly*, загружает инсталлятор FreeBSD, который позволяет производить установку по FTP. Многие спрашивают: «Если я могу произвести установку через FTP, зачем мне все эти компакт-диски?» Стандартный ISO-образ FreeBSD содержит массу всего. Если вы устанавливаете дистрибутив не полностью, позднее вам может потребоваться доустановить что-нибудь, что имеется на компакт-диске. Не у всех имеется неограниченное высокоскоростное подключение к Интернету.[1]

После выбора образа его необходимо записать на компакт-диск. Способ записи существенно зависит от операционной системы, даже в мире UNIX-подобных систем запись компакт-дисков в разных системах выполняется по-разному. В операционной системе Windows можно воспользоваться такими программами, как Nero или Stomp. В системе FreeBSD, при использовании стандартной программы записи на IDE-привод компакт-дисков, запись выполняется, как показано ниже:

```
# burncd -f /dev/acd0 data imagename fixate
```

Ознакомьтесь с инструкциями к вашей операционной системе по записи образов компакт-дисков на физические носители. Этот файл должен записываться как образ, а не как обычный файл. Если что-то вы делаете неправильно, программное обеспечение, выполняющее запись на компакт-диски, сообщит, что файл не помещается на носителе. Файл образа не поместится на диск, если записывать его как обычный файл, но если его записывать как образ, таких ошибок происходить не будет.

---

[1] А те, у кого такое подключение имеется, должны воздерживаться от язвительных замечаний в адрес тех, кто такого подключения не имеет.

Установка по FTP

Если вы начали установку с компакт-диска, установочный носитель уже готов к работе – это тот самый диск, с которого была выполнена загрузка. Но в случае установки по FTP вам необходимо выбрать FTP-сервер и иметь представление о том, как ваш компьютер подключен к локальной сети.

Выбор FTP-сервера – это важный этап. Найдите список зеркал FTP-сайтов и с помощью утилиты ping попробуйте обратиться к ним. Вам нужно найти FTP-сервер с наименьшим временем отклика – это хороший признак высокой доступности сервера при вашем местоположении. Как только вы отберете пару достойных кандидатов, соединитесь с ними по FTP со своего настольного компьютера. Проверьте, насколько высока скорость передачи файлов. Выберите тот, что имеет более высокую скорость, и убедитесь, что на нем присутствует устанавливаемая версия FreeBSD. Запомните имя сервера, потому что оно потребуется вам в процессе установки.

Если в вашей сети для назначения IP-адресов и выбора других параметров настройки используется протокол динамического выбора конфигурации хоста (Dynamic Host Configuration Protocol, DHCP), то можно считать, что все готово к продолжению установки. В противном случае, если в вашей сети IP-адреса назначаются сетевым администратором, получите от него следующую информацию:

- IP-адрес будущей системы FreeBSD
- Сетевую маску будущей системы FreeBSD
- IP-адреса серверов имен в вашей сети
- IP-адрес шлюза по умолчанию
- Сведения о прокси-сервере (если это необходимо)

Без этой информации при отсутствии DHCP вы не сможете подключиться к сети и выполнить установку по FTP.

Непосредственная установка FreeBSD

Теперь, когда были приняты все необходимые решения, и вы приступили к установке FreeBSD, осталось лишь пройти через тернии неудобного инсталлятора. Вставьте загрузочный носитель в устройства и включите питание компьютера.[1] Вы увидите несколько экранов начальной загрузки и системную отладочную информацию, которые подробнее мы рассмотрим в главе 3.

---

[1] На самом деле питание все-таки придется включить в самом начале, иначе вам будет нелегко вставить компакт-диск в выключенный привод. – *Прим. научн. ред.*

Первое меню, которое вы увидите, предложит вам выбрать раскладку клавиатуры. Оно включает в себя полный список всех раскладок, поддерживаемых системой FreeBSD. Обратите внимание: раскладка клавиатуры не имеет никакого отношения к языку, на котором будут даваться инструкции в процессе установки, – это всего лишь раскладка клавиатуры.

Далее FreeBSD представит вам первый экран, как показано на рис. 2.1.

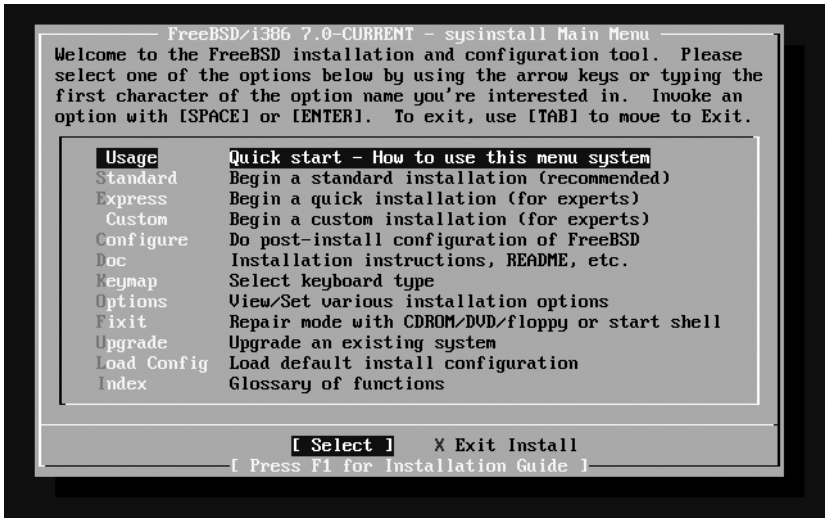

Рис. 2.1. *Основное меню sysinstall*

Это печально известный инсталлятор FreeBSD – sysinstall(8). В то время как другие операционные системы имеют симпатичные графические инсталляторы с меню, управляемыми мышью, и многоцветными секторными диаграммами, FreeBSD выглядит, как старая программа DOS. Работа по ее замене ведется, но сейчас, когда я пишу эти строки, мне кажется, что sysinstall еще некоторое время из FreeBSD никуда не денется.

Выбор вариантов меню производится клавишей пробела, а не Enter.

С помощью курсорных клавиш опуститесь вниз, до строки выбора типа установки Standard, и нажмите клавишу Enter. После этого вы увидите текст предупреждения программы fdisk с некоторыми простыми инструкциями (рис. 2.2).

Ознакомьтесь с инструкциями, чтобы убедиться, что они не изменились с момента выхода книги в свет, и нажмите клавишу Enter.

Если у вас несколько жестких дисков, FreeBSD позволит вам выбрать диск для установки. Выберите требуемый диск клавишей пробела (рис. 2.3).

В этом месте для некоторых дисков могут выводиться пугающие предупреждения о геометрии. При наличии современного оборудования это не проблема. Подробнее о геометрии дисков мы поговорим в главах 8 и 18. Если вам интересно, можете сначала заглянуть в эти главы, а потом продолжить установку. Чтобы перейти к следующему экрану, просто нажмите клавишу Enter (рис. 2.4).

Здесь вам нужно определить, какой объем дискового пространства вы ответете для FreeBSD. В случае сервера было бы желательно отвести весь диск. Нажмите клавишу A, чтобы выделить весь диск для FreeBSD, а затем клавишу Q, чтобы завершить работу с программой. Далее инсталлятор представит экран выбора расположения MBR (Master Boot Record – главная загрузочная запись), как показано на рис. 2.5.

Нажмите клавишу со стрелкой вниз, чтобы переместиться на строку с меткой Standard, а затем клавишу Tab, чтобы выделить кнопку с надписью OK. При таком выборе будет выполнена стандартная установка главной загрузочной записи (MBR), что приведет к удалению существующего менеджера загрузки, который мог использоваться на вашем

Рис. 2.2. *Инструкции программы fdisk*

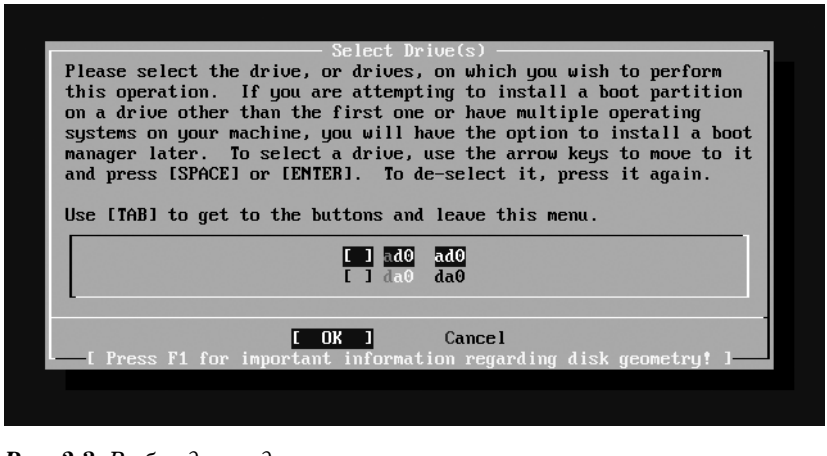

Рис. 2.3. *Выбор диска для установки*

компьютере для загрузки какой-либо другой операционной системы. (Речь идет о создании сервера Интернета, значит дисковое пространство не будет разделяться, скажем, с Windows Vista.) Для продолжения нажмите клавишу Enter.

Если у вас имеется несколько жестких дисков, инсталлятор вернет вас к экрану выбора жесткого диска. Выберите следующий диск или с помощью клавиши Tab выберите кнопку OK и перейдите к следующему этапу установки. После этого sysinstall выведет инструкции по использованию инструмента создания разделов (рис. 2.6).

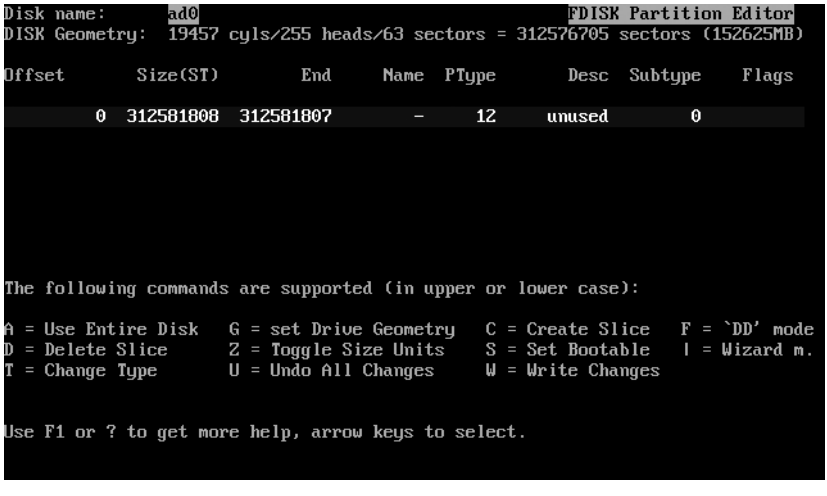

Рис. 2.4. *Меню программы fdisk*

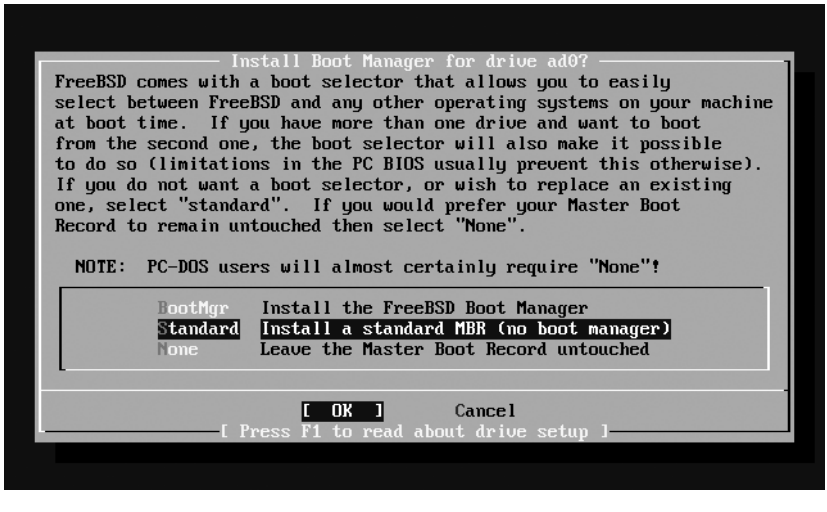

Рис. 2.5. *Установка MBR*

Ознакомьтесь с инструкциями, чтобы убедиться, что они не изменились с момента выхода книги в свет, и нажмите клавишу Enter.

Теперь вы должны находиться в меню создания разделов. Ранее в этой же главе мы уже говорили о разбиении диска, поэтому у вас уже должно быть готовое решение о том, какие разделы должны быть созданы. Здесь вы должны воплотить свои решения (рис. 2.7).

Чтобы принять рекомендации FreeBSD, предлагаемые по умолчанию, нажмите клавишу A. В противном случае, чтобы создать новый раздел, нажмите клавишу C. Введите требуемый размер раздела, используя символ *M* для обозначения мегабайтов или *G* – для гигабайтов. После этого инсталлятор спросит вас, будет ли это файловая система или пространство свопинга. Если вы скажете, что это будет файловая система, то вам будет предложено указать точку монтирования (*/*, */usr*, */var* и т. д.).

Когда создание разделов будет закончено, нажмите клавишу Q, чтобы выйти из редактора разделов.

Рис. 2.6. *Инструкции по созданию разделов*

Рис. 2.7. *Редактор разделов*

Далее вам будет предложено указать источник файлов для установки (рис. 2.8).

С помощью курсорных клавиш выделите желаемый носитель и нажмите клавишу Enter, чтобы выбрать его. FreeBSD либо проверит компакт-диск, чтобы убедиться в его пригодности для выполнения установки, либо попросит вас выбрать сервер FTP, либо предложит выполнить настройки для другого выбранного вами носителя. Я рекомендую выполнять установку или с компакт-диска, или по FTP.

В следующем меню вам будет предложено выбрать, что устанавливать (рис. 2.9). Хотя FreeBSD предлагает несколько сокращенных вариантов установки системы на случай ограниченности дискового пространства, тем не менее современные жесткие диски стали настолько велики, что их объема с лихвой хватит для установки полной версии FreeBSD. В случае современного компьютера я рекомендовал бы установить все, особенно если вы собираетесь изучать FreeBSD. Нажмите клавишу со стрелкой вниз, чтобы переместиться на строку с меткой All, а затем клавишу Enter, чтобы подтвердить выбор.

Далее sysinstall спросит, хотите ли вы установить коллекцию портов (Ports Collection). Ответьте утвердительно, даже если пока вы не знаете, что это такое. Выберите кнопку Yes.

После этого на экране опять появится меню выбора варианта установки. Выделите пункт Exit this menu (Выйти из этого меню) и нажмите клавишу Enter.

Затем инсталлятор даст вам последний шанс что-то изменить, прежде чем начнет копирование файлов. Как только вы ответите Yes, install (Да, установить), он отформатирует жесткий диск, привод компакт-

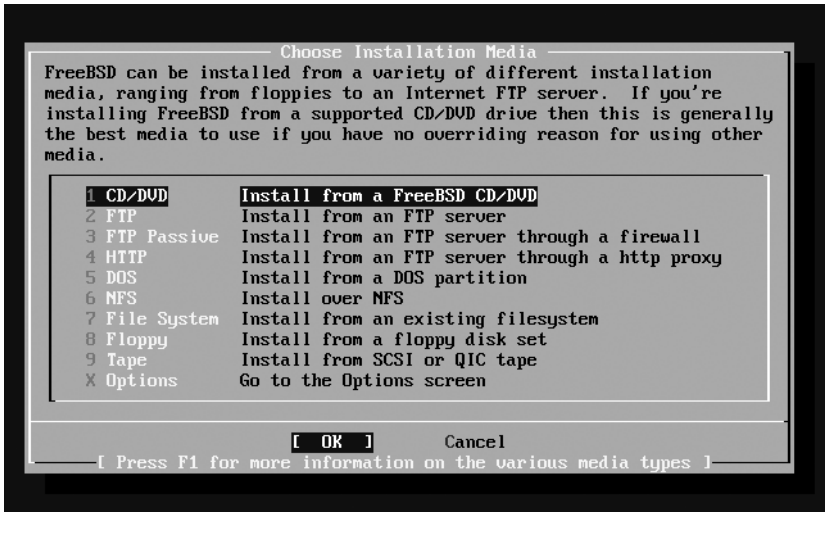

Рис. 2.8. *Выбор носителя дистрибутива*

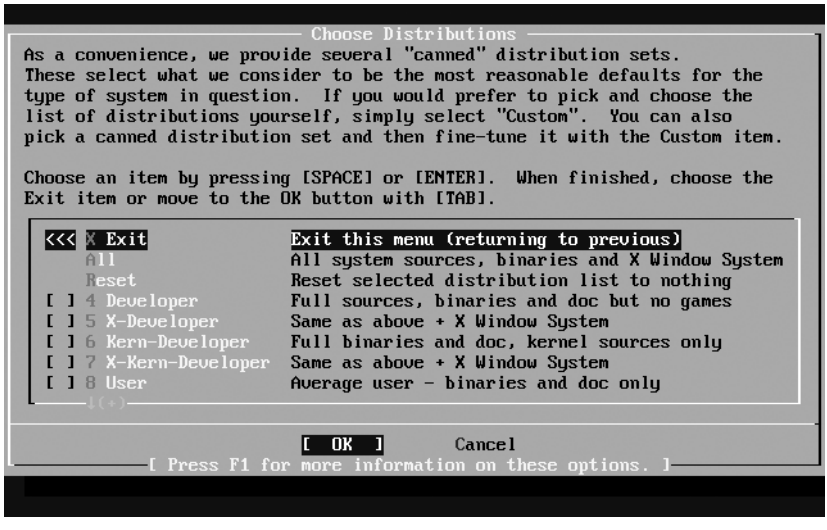

Рис. 2.9. *Выбор варианта установки*

дисков заморгает лампочкой и через несколько минут FreeBSD будет установлена.

Затем инсталлятор перейдет к группе вопросов, касающихся настройки основных системных служб.

Настройка сети

Инсталлятор предложит вам выполнить настройку сетевого интерфейса. Ответьте Yes.

Вам будет предложено на выбор сразу несколько сетевых интерфейсов для настройки (рис. 2.10). Да, FreeBSD может работать с протоколами TCP/IP через FireWire! Более того, она способна работать с протоколами TCP/IP даже через параллельный порт. Хотя эта возможность используется крайне редко, тем не менее она существует. Найдите пункт, который напоминал бы карту Ethernet, и выберите его. На рис. 2.10 можно увидеть карту *Intel EtherExpress Pro/100B PCI Fast Ethernet*, которая наиболее похожа на искомое. Выделите эту строку и нажмите клавишу Enter, чтобы приступить к настройке.

Вам будет предложено попытаться использовать настроить этот интерфейс для IPv6. Скорее всего, вам это не нужно. Затем вам будет предложено выполнить настройку DHCP. Так как речь идет о сервере, скорее всего это тоже вам не подходит. После этого вы переместитесь в экран Network Configuration (Настройка сети), как показано на рис. 2.11.

Здесь вам нужно указать имя хоста, имя домена, а также информацию о подключении к сети, которую вы должны были получить у сетевого администратора.

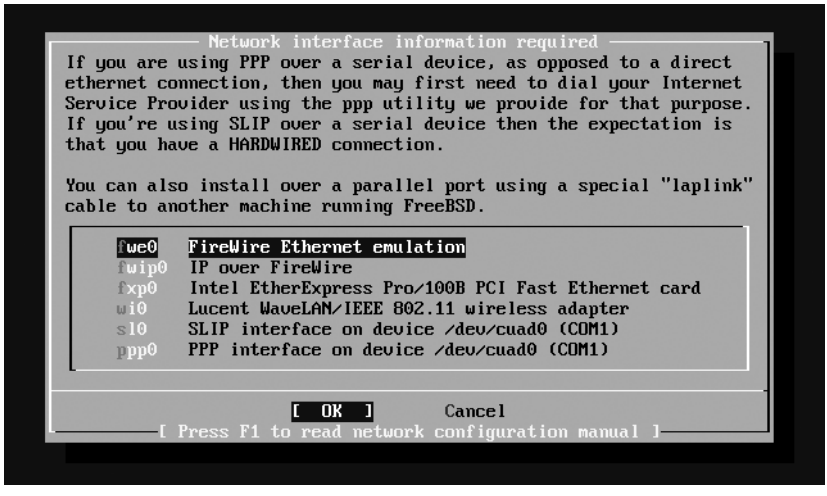

Рис. 2.10. *Меню выбора сетевого интерфейса*

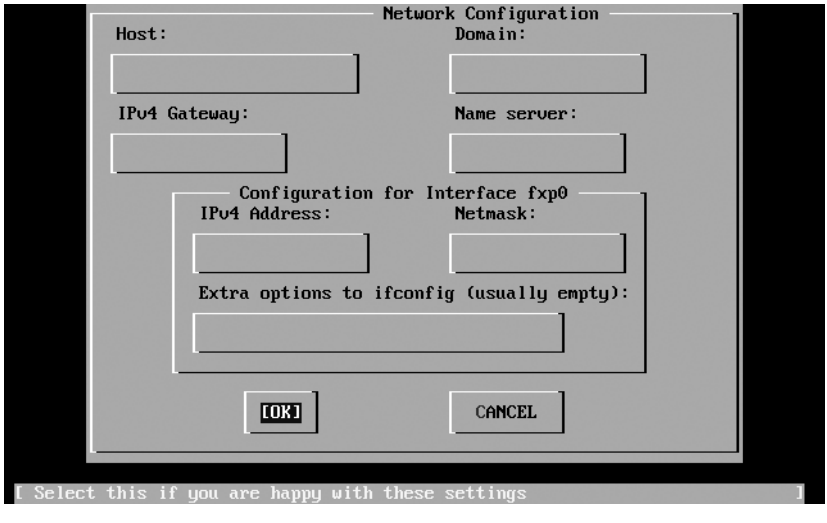

Рис. 2.11. *Настройка сети*

Даже если используется DHCP, вам все равно необходимо указать имя хоста и имя домена. В противном случае ваша система во время загрузки будет присваивать себе имя *Amnesiac* (забывчивый). (Вы можете использовать сервер DHCP для настройки имени хоста, но это слишком новая возможность и в большинстве случаев она не обеспечивается.)

Различные сетевые службы

Далее инсталлятор предложит вам ответить на ряд вопросов, связанных с функциями, которые будет выполнять система. Если только вы

не опытный системный администратор, вам едва ли потребуется запускать большую часть этих функций. Мы будем разрешать запуск некоторых из них в процессе повествования. Как только вы начнете понимать, о чем идет речь, вы сможете разрешить запуск требуемых служб при выполнении последующих установок.

Например, инсталлятор спросит, будет ли система выполнять функции сетевого шлюза, или следует ли выполнить настройку inetd. Выберите ответ No на оба вопроса. Когда будет предложено разрешить удаленный вход через SSH, ответьте Yes – это достаточно надежная и безопасная служба, необходимая практически любой системе. Не разрешайте запуск анонимного FTP-сервера, сервера NFS, клиента NFS или настройте syscons.

Часовой пояс (Time Zone)

Инсталлятор попросит вас указать часовой пояс. Затем вам будет предложено установить время системных часов в UTC: ответьте No и обойдите предложенные экраны с настройками, где вам будет предложено выбрать континент, страну и часовой пояс.

Режим совместимости с Linux (Linux Mode)

Теперь инсталлятор предложит вам разрешить режим совместимости с Linux. Я предлагаю ответить No на этот вопрос. Если вам потребуется режим совместимости с Linux, я расскажу, как его активировать в главе 12.

Мышь PS/2

Мышь, подключаемая к порту USB, начинает работать автоматически, но мышь, подключаемая к порту PS/2 или более старая, требует выполнения специальных настроек. Инсталлятор предложит вам настроить мышь PS/2. Если вы пользуетесь стандартной двух- или трехкнопочной мышью PS/2, ответьте Yes и выберите пункт меню Enable (Разрешить). После этого вы должны увидеть на экране указатель мыши, который будет реагировать на ее перемещения.

Инсталлятор спросит, работает ли ваша мышь. Если указатель перемещается в соответствии с движениями самой мыши, ответьте Yes. Честно признаться, за последние 10 лет у меня не было такой мыши PS/2, которая не заработала бы. Более древние мыши могут оказаться более сложными в настройке, но они встречаются все реже и реже.

Добавление пакетов

Далее инсталлятор спросит, желаете ли вы установить дополнительные пакеты с программным обеспечением. Если вы уже опытный системный администратор, то наверняка знаете, какое дополнительное программное обеспечение вам необходимо установить. Возможно, у вас

есть свои предпочтения относительно командной оболочки, которая может не устанавливаться по умолчанию.

Система FreeBSD подразделяет пакеты программного обеспечения на категории. Отыщите категорию, которая, на ваш взгляд, лучше всего подходит для требуемой программы, откройте список программного обеспечения в этой категории, который находится на вашем носителе. Найдите требуемую программу и нажмите клавишу пробела, чтобы выбрать ее. Например, чтобы установить популярную командную оболочку Bash, найдите категорию Shells, нажмите клавишу Enter, отыщите пункт Bash и нажмите клавишу пробела. Затем нажмите клавишу Enter, чтобы вернуться обратно в меню Package Selection (Выбор пакетов).

Когда будут выбраны для установки все необходимые пакеты, вернитесь в главное меню Package Selection. Нажмите клавишу Tab, чтобы выделить кнопку OK to Install, и нажмите клавишу Enter. В результате система установит выбранные пакеты.

Добавление пользователей

По возможности все операции следует проводить, входя в систему с правами обычного пользователя. Регистрация под именем root необходима лишь при проведении изменений в системе. Первое время это происходит часто, но затем все реже и реже. Однако для входа в систему в качестве обычного пользователя необходимо создать его учетную запись. Инсталлятор даст вам возможность создать учетные записи пользователей в процессе установки. Ответьте Yes на вопрос и вы увидите экран, как показано на рис. 2.12.

Прежде всего следует выбрать регистрационное имя-идентификатор (Login ID), или имя пользователя. Некоторые компании устанавливают

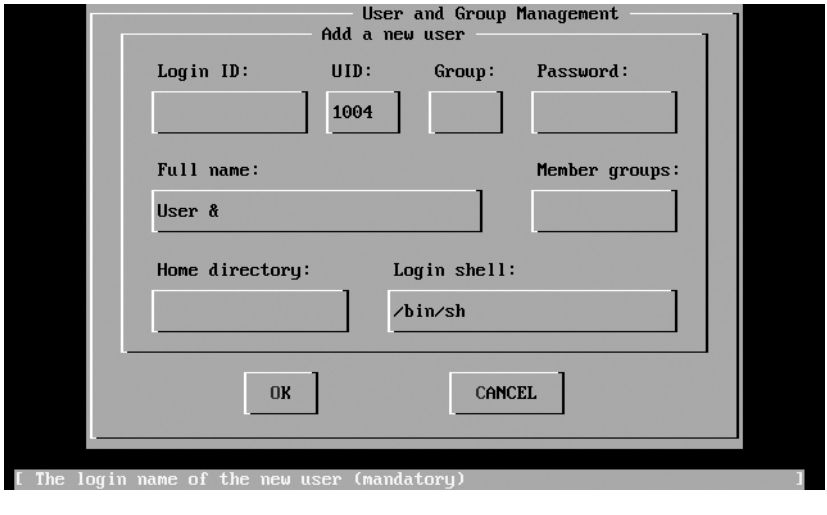

Рис. 2.12. *Добавление пользователя*

стандарт, которому должны соответствовать имена пользователей. Я предпочитаю, чтобы имя пользователя состояло из его инициалов и фамилии (инициалы помогают избежать неожиданных конфликтов).

Идентификатор пользователя (UID) назначается системой.

По умолчанию во FreeBSD пользователь попадает в «группу» (Group), название которой соответствует имени пользователя. Например, пользователь «mwlucas» автоматически попадает в группу «mwlucas». Опытный администратор может изменить это.

Full name (*Полное имя*) – это имя пользователя. Другие пользователи системы могут видеть это имя, поэтому не стоит выбирать его произвольно. У некоторых системных администраторов возникали трудности, когда они назначали пользователям сомнительные имена, например «Pain in the Tuckus[1]».

Member group (*Группы пользователей*) – простой список других групп системы, в которые будет входить пользователь. Если необходимо, чтобы пользователь мог применять пароль root и становиться пользователем root, этого пользователя следует добавить в группу «wheel». В этой группе должны присутствовать только администраторы.

Home directory (*Домашний каталог*) – здесь хранятся файлы пользователя. Обычно подходит значение по умолчанию.

Наконец, для нового пользователя надо выбрать командную оболочку (shell). Умудренные опытом администраторы и пожилые преподаватели зачастую предпочитают */bin/sh*. В примерах, приводимых в книге, подразумевается более дружелюбная командная оболочка */bin/tcsh*, входящая в современные дистрибутивы BSD. Если вы предпочитаете другую командную оболочку, используйте ее.

После этого нажмите OK, и пользователь будет создан.

Пароль root

Теперь инсталлятор предложит вам установить пароль пользователя root. Если такого пароля нет, любой сможет войти в систему как root, не набрав никакого пароля. Поскольку root обладает полной властью над аппаратным и программным обеспечением, это плохо.

Система FreeBSD предложит вам ввести пароль дважды. Запомните пароль пользователя root, так как восстановить его будет очень непросто. Подробнее о пароле пользователя root и о проблемах безопасности мы поговорим в главе 7.

Настройка после инсталляции

В заключение вам будет предложено выполнить заключительные настройки вашего сервера FreeBSD. Меню FreeBSD Configuration Menu

[1] Что-то вроде «шило в заднице». – *Прим. перев.*

(рис. 2.13) обеспечивает простой доступ к начальным настройкам компьютера.

В этом меню вы сможете разрешить или запретить все настройки, которые были выполнены на более ранних этапах установки, а также настроить некоторые дополнительные сетевые функции. Если в вашей сети имеется сервер точного времени (NTP), с помощью этого меню вы можете разрешить своей системе FreeBSD обращаться к нему. Позднее в этой книге будет показано, как активировать все эти службы, но если вы точно знаете, что делаете, можете выполнить необходимые настройки прямо сейчас.

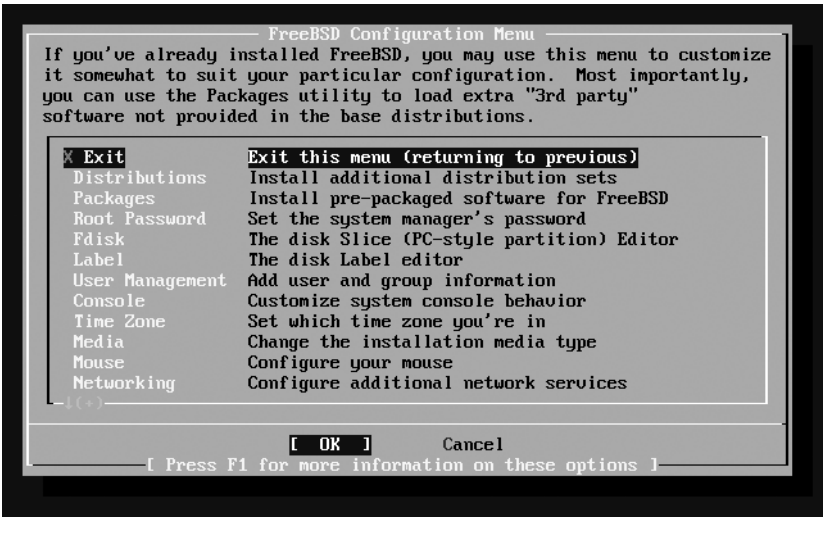

Рис. 2.13. *Настройка после инсталляции*

Перезапуск!

Как только будут выполнены все настройки после установки, вернитесь в главное меню инсталлятора и выберите пункт Exit (выйти). После этого будет выполнена перезагрузка и установленная система FreeBSD будет готова для выполнения примеров из этой книги.

Если позднее вам потребуется воспользоваться программой sysinstall(8) для настройки системы, вы сможете запустить ее в любое время. В конце книги вы узнаете, как те же самые настройки, которые выполняет sysinstall, можно сделать из командной строки быстрее и проще.

Теперь посмотрим, что в действительности происходит во время перезагрузки.

3

Пуск! Процесс загрузки

После включения питания FreeBSD загружается автоматически, но точное понимание, что именно происходит на каждой стадии, сделает вас лучшим администратором. Вмешательство в процесс загрузки требуется очень редко, но когда это будет необходимо, вы будете рады, что знаете, как это сделать. И как только вы почувствуете, что неплохо ориентируетесь в процессе загрузки, вы обнаружите, что в состоянии решить проблемы, которые раньше просто принимали и терпели.

Сначала мы выясним, как запускается загрузчик, затем рассмотрим некоторые изменения, которые можно сделать, и информацию, которую можно собрать из командной строки загрузчика, включая загрузку альтернативного ядра и запуск системы в однопользовательском режиме. Затем поговорим о последовательных консолях – стандартном инструменте управления системой. В процессе загрузки в многопользовательском режиме производится запуск всех служб системы, которые делают компьютер пригодным к практическому использованию, поэтому мы обратим внимание и на этот процесс. Дополнительно мы рассмотрим информацию, которая записывается системой в процессе загрузки, а также как выключать компьютер, чтобы не повредить данные.

Осторожно, рекурсия!

Некоторые темы, описываемые в этой главе, ссылаются на материал последующих глав. В свою очередь, эти последующие главы требуют, чтобы вы имели полное представление о том, что описывается в этой главе. Нельзя однозначно сказать, с какого места лучше начинать изучение. Если вы вообще не понимаете, о чем говорится в той или иной части этой главы, просто бегло пролистайте ее и продолжите чтение – позднее все встанет на свои места.

Сам процесс загрузки можно разделить на три основных этапа: загрузчик, запуск однопользовательского режима и запуск многопользовательского режима.

Включение питания и загрузчик

В любом i386 компьютере имеется базовая система ввода/вывода (Basic Input/Output System, BIOS), у которой достаточно ума, чтобы отыскать операционную систему на диске. (В других платформах имеется ПЗУ с микропрограммами, которые выполняют ту же самую функцию.) Если BIOS отыскивает операционную систему на диске, она передает управление компьютером этой операционной системе. Если операционная система не будет обнаружена, BIOS сообщит об этом и остановится. В большинстве BIOS операционные системы распознаются по очень примитивным признакам. *Загрузочные блоки* – это секторы диска, по которым BIOS компьютера распознает наличие операционной системы. В эти блоки FreeBSD записывает маленькую программу, которая запускает главную программу запуска FreeBSD, loader(8). Программа loader выводит на экран логотип FreeBSD и меню из семи пунктов:

1. Boot FreeBSD [default] (Загрузка FreeBSD [по умолчанию])
2. Boot FreeBSD with ACPI disabled (Загрузка FreeBSD с запретом ACPI)
3. Boot FreeBSD in safe mode (Загрузка FreeBSD в безопасном режиме)
4. Boot FreeBSD in single-user mode (Загрузка FreeBSD в однопользовательском режиме)
5. Boot FreeBSD with verbose logging (Загрузка FreeBSD с подробным протоколированием)
6. Escape to loader prompt (Перейти в командную строку загрузчика)
7. Reboot (Перезагрузка)

Если подождать 10 секунд, загрузчик автоматически перейдет к загрузке FreeBSD по умолчанию. Некоторые другие пункты меню востребованы только для отладки или устранения проблем. Не нужно стараться запоминать весь перечень пунктов меню, достаточно, чтобы вы свободно могли использовать самые основные, когда это потребуется.

Boot FreeBSD with ACPI disabled (Загрузка FreeBSD с запретом ACPI)

ACPI – Advanced Configuration and Power Interface (усовершенствованный интерфейс управления конфигурированием и энергопотреблением), стандарт конфигурирования аппаратного обеспечения, принятый Intel/Toshiba/Microsoft. Он заместил устаревшие стандарты Microsoft – APM (Advanced Power Management – усовершенствованные средства управления питанием[1]), PnPBIOS, таблицу MP,

[1] Урок на будущее: никогда не употребляйте в названиях слово *advanced* (усовершенствованный, передовой). Наступит день, когда такое название не будет соответствовать действительности.

таблицу $PIR и целую связку стандартов еще более туманных и непонятных. Подробнее ACPI будет рассматриваться в главе 5. Этот интерфейс настроен на современное аппаратное обеспечение, но в некоторых аппаратных средствах интерфейс ACPI реализован с ошибками. С другой стороны, самые современные многопроцессорные аппаратные платформы требуют поддержки ACPI в обязательном порядке.

Если вновь установленная система не загружается обычным образом, попробуйте загрузить ее с запретом ACPI. Если система проработала какое-то время, но вдруг неожиданно ушла на перезагрузку, в этом случае запрет ACPI скорее всего не поможет.

Boot FreeBSD in safe mode (Загрузка FreeBSD в безопасном режиме)

При загрузке FreeBSD в *безопасном режиме* активизируются лишь самые консервативные возможности операционной системы. Для жестких дисков ATA отключается поддержка DMA (Direct Memory Access – прямой доступ к памяти) и кэширования записи, тем самым ограничивается скорость их работы, но зато увеличивается устойчивость к разного рода проблемам, связанным с аппаратной частью. Запрещается использование интерфейса ACPI и не выполняется попытка активизировать слоты EISA. На платформе i386 запрещается многопроцессорная обработка. Клавиатуры с интерфейсом USB не работают в однопользовательском режиме. Безопасный режим удобно использовать для восстановления системы или ее отладки, а также для разрешения проблем, связанных с аппаратной частью.

Boot FreeBSD in single-user mode (Загрузка FreeBSD в однопользовательском режиме)

Однопользовательский режим – это минимальный режим запуска, который часто используется для запуска поврежденной системы. Это самый ранний момент, когда FreeBSD в состоянии предоставить командную строку в ваше распоряжение. Этот режим имеет настолько важное значение, что он будет обсуждаться в отдельном разделе этой главы.

Boot FreeBSD with verbose logging (Загрузка FreeBSD с подробным протоколированием)

В процессе загрузки FreeBSD получает уйму информации о компьютере. Большая часть этой информации будет не нужна вам в повседневной жизни, но она может оказать вам существенную помощь при отладке. При загрузке в режиме подробного протоколирования FreeBSD выводит все сведения о параметрах системы и присоединенных устройствах. (Впоследствии эта информация будет доступна в файле */var/run/dmesg.boot*, о котором подробнее будет рассказываться ниже в этой же главе.) Вы можете разок попробовать режим подробного протоколирования на всех ваших компьютерах, чтобы осознать сложность их устройства.

Escape to loader prompt (Перейти в командную строку загрузчика)

Загрузчик содержит в себе командный интерпретатор, с помощью которого вы сможете запускать команды настройки запуска системы под свои нужды. Подробнее об этом будет рассказываться в разделе «Командная строка загрузчика».

Reboot (Перезагрузка)

Повторить, но на этот раз с чувством! Наиболее важными из этих пунктов являются однопользовательский режим и командная строка загрузчика.

Однопользовательский режим

Минимальная начальная загрузка, так называемый *однопользовательский режим*, подразумевает загрузку ядра и нахождение устройств. При этом не происходит автоматического монтирования диска, подключения к сети, активизации защиты системы и запуска стандартных служб UNIX. Однопользовательский режим – это первый рубеж, на котором система способна дать вам доступ к командной строке (command prompt). Отсюда можно активизировать любые службы, которые не были запущены автоматически.

При запуске в однопользовательском режиме вы увидите привычный поток системных сообщений. Однако перед запуском каких-либо программ ядро предложит вам выбрать командную оболочку. Вы можете выбрать любую оболочку, интерпретатор которой находится в корневом разделе диска. Я обычно выбираю оболочку по умолчанию – */bin/sh*, но вы можете выбрать оболочку, исходя из своих предпочтений, например */bin/tcsh*.

Диски в однопользовательском режиме

В однопользовательском режиме монтируется только корневой раздел, причем в режиме только для чтения. Остальные диски не монтируются. (Подробнее диски и файловые системы будут обсуждаться в главе 8, а пока просто читайте дальше.)

Многие программы, которые могут вам потребоваться, находятся в разделах, отличных от корневого, поэтому все разделы должны быть доступны в режиме чтения/записи. Чтобы убедиться, что эти файловые системы находятся в исправном состоянии, запустите следующие команды:

```
# fsck -p
# mount -a
```

Программа fsck(8) «почистит» файловые системы, подтвердит их внутреннюю целостность и действительное наличие файлов раздела на диске. Доступ к файловой системе открывается командой mount(8).

Ключ `-a` говорит о необходимости выполнить монтирование всех файловых систем, перечисленных в файле */etc/fstab* (глава 8), но если какая-либо из этих файловых систем вызывает проблемы, можно смонтировать требуемые файловые системы по отдельности, указав в командной строке путь к точке монтирования (например, `mount /usr`). Если вы обладаете опытом настройки сетевых файловых систем (глава 8), вы увидите сообщения об ошибке монтирования этих файловых систем, поскольку в этот момент сеть еще не была запущена.

Если монтирование по имени каталога не получилось, попробуйте вместо него использовать имя устройства. Имя устройства корневого раздела, скорее всего, будет либо */dev/ad0s1a* (для дисков IDE), либо */dev/da0s1a* (для дисков SCSI). При этом вам также потребуется указать имя точки монтирования этого раздела. Например, чтобы смонтировать первый раздел диска IDE как корневую файловую систему, введите команду:

```
# mount /dev/ad0s1a /
```

Если на вашем сервере имеются сетевые файловые системы, а сеть еще не запущена, тогда смонтируйте все локальные файловые системы, указав тип файловой системы. Следующая команда монтирует все локальные файловые системы типа UFS, которая по умолчанию используется системой FreeBSD:

```
# mount -a -t ufs
```

Программы, доступные в однопользовательском режиме

Круг команд, доступных в однопользовательском режиме, зависит от того, какие разделы были смонтированы. Некоторые основные команды находятся в каталогах */bin* и */sbin*, расположенных в корневом разделе, и доступны, даже если корневой раздел смонтировать в режиме только для чтения. Другие команды находятся в каталоге */usr* и будут недоступны, пока вы не смонтируете этот раздел. (Загляните в каталоги */bin* и */sbin*, чтобы получить представление, чем вы сможете воспользоваться, когда что-то пойдет не так.)

Примечание

Если системные библиотеки находятся в шифруемом разделе (глава 12), ни одна из этих программ работать не будет. Для таких случаев FreeBSD предоставляет версии многих базовых утилит, связанных с библиотеками статически, в каталоге */rescue*.

Сеть в однопользовательском режиме

Если в однопользовательском режиме необходимо подключиться к сети, используйте сценарий на языке командной оболочки */etc/netstart*. Этот сценарий вызовет другие сценарии, необходимые для запуска сетевой подсистемы, настройки сетевых интерфейсов, а также механиз-

мов фильтрации пакетов и маршрутизации. Если вам нужно запустить не все службы, а только некоторые из них, вам необходимо ознакомиться с текстом этого сценария и выполнить соответствующие команды вручную.

Работа в однопользовательском режиме

В однопользовательском режиме доступ к системе ограничен только вашими познаниями FreeBSD и UNIX.

Например, если вы забыли пароль root, его можно переопределить, находясь в однопользовательском режиме:

```
# passwd
Changing local password for root
New Password:
Retype New Password:
#
```

Примечание

Обратите внимание: система не требует ввода старого пароля пользователя root. Загрузившись в однопользовательский режим, вы автоматически становитесь пользователем root, поэтому passwd(8) не просит ввести старый пароль.

Или, если вы допустили опечатку в файле */etc/fstab*, которая сбивает систему с толку и препятствует нормальной загрузке, вы можете смонтировать корневой раздел, используя имя устройства, отредактировать файл */etc/fstab* и тем самым ликвидировать проблему.

Или, если вы используете программу, которая вызывает у системы панику на этапе загрузки и вам необходимо предотвратить запуск этой программы во время загрузки, вы можете либо отредактировать файл */etc/rc.conf*, запретив запуск программы, либо переопределить права файла сценария, сделав его неисполняемым.

```
# chmod 444 /usr/local/etc/rc.d/program.sh
```

Подробнее о программах сторонних производителей (порты и пакеты) мы поговорим в главе 11.

Примечание

Во всех этих примерах описываются ошибки, допущенные человеком. Отказы аппаратуры встречаются довольно редко, еще реже – отказы самой системы FreeBSD. Если бы человеку не было свойственно ошибаться, наши компьютеры практически всегда работали бы без перебоев. По мере расширения ваших познаний о FreeBSD будут расширяться и ваши возможности в однопользовательском режиме.

Мы еще не раз будем возвращаться к однопользовательскому режиму, а сейчас давайте поговорим о командной строке загрузчика.

Командная строка загрузчика

Командная строка загрузчика – это ограниченная по своим возможностям вычислительная среда, которая позволяет вносить некоторые изменения в процесс загрузки и в переменные, которые должны быть настроены перед началом загрузки. После выхода в командную строку загрузчика (пункт 6 в меню загрузки) вы увидите такое приглашение к вводу:

```
OK
```

Это и есть командная строка загрузчика. Слово *OK* выглядит вполне дружественным и ободряющим, но следует заметить, что это одна из немногих дружественных вещей в среде загрузчика. Ее нельзя рассматривать как полноценную командную строку операционной системы, она служит лишь инструментом, который дает возможность настроить процесс загрузки системы и не предназначена для несведущих или слабонервных пользователей. Любые изменения, которые вы вносите в командной строке загрузчика, оказывают воздействие только на текущий процесс загрузки. Чтобы отменить изменения, необходимо выполнить перезагрузку. (Как запоминать изменения, произведенные в командной строке загрузчика, вы узнаете в следующем разделе.)

Чтобы увидеть перечень всех доступных команд, введите знак вопроса.

```
OK ?
Available commands:
    heap         show heap usage
    reboot       reboot the system
    bcachestat   get disk block cache stats
...
```

Первые три команды загрузчика, перечисленные выше, практически бесполезны и могут пригодиться разве что разработчику. Мы же сосредоточим свое внимание на командах, которые могут использоваться для администрирования системы.

Чтобы увидеть перечень дисков, доступных для загрузчика, используйте команду `lsdev`.

```
  OK lsdev
❶ cd devices:
  disk devices:
    disk0: ❷BIOS drive C:
      ❸disk0s1a: FFS
       disk0s1b: swap
       disk0s1d: FFS
       disk0s1e: FFS
       disk0s1f: FFS
    disk1: ❹BIOS drive D:
       disk1s1a: FFS
       disk1s1b: swap
pxe devices:
```

Загрузчик проверил наличие приводов компакт-дисков ❶ и не нашел ни одного. (Загрузчик может обнаружить приводы компакт-дисков только при загрузке с компакт-диска, поэтому на этот счет не стоит беспокоиться.) Он обнаружил два жестких диска, которые в BIOS известны под именами C ❷ и D ❹. Далее следуют описания разделов, обнаруженных на этих жестких дисках. Как будет говориться в главе 8, имя устройства с корневым разделом обычно оканчивается символом *a*. Это означает, что здесь под именем `disk0s1a` показан корневой раздел ❸. Если бы вам пришлось столкнуться с неполадками во время загрузки незнакомой системы, это знание оказалось бы весьма полезным.

Загрузчик использует в своей работе переменные, значения которых устанавливаются ядром или берутся из конфигурационных файлов. Увидеть переменные и их значения можно с помощью команды `show`.

```
OK show
LINES=24
acpi_load=YES
autoboot_delay=NO
...
```

Нажатие клавиши пробела приведет к появлению следующей страницы. Эти значения включают номера IRQ и адреса памяти для старых карт ISA, низкоуровневые параметры настройки ядра и информацию, извлеченную из BIOS. Частичный список переменных загрузчика мы увидим в разделе «Настройка загрузчика»; еще ряд значений будут встречаться на протяжении всей книги в соответствующих разделах.

Изменить эти значения для каждой конкретной загрузки можно с помощью команды `set`. Например, чтобы изменить значение `console` на `comconsole`, необходимо ввести следующую команду:

```
OK set console=comconsole
```

К тому моменту, как загрузчик предоставит в ваше распоряжение свою командную строку, ядро уже будет загружено в память. Ядро – это сердце FreeBSD и подробно будет рассматриваться в главе 5. Если ранее вам никогда не доводилось работать с ядром, просто приберегите эти лакомые кусочки до соответствующей главы. Увидеть ядро и модули ядра, загруженные в память, можно с помощью команды `lsmod`.

```
OK lsmod
  0x400000: ❶/boot/kernel/kernel (elf kernel, 0x6a978c)
    ❷modules: ❸elink.1 io.1 splash.1 agp.1 nfsserver.1 nfslock.1 nfs.1 nfs4.1
wlan.1 if_gif.1 if_faith.1 ether.1 sysvshm.1 sysvsem.1 sysvmsg.1 cd9660.1
isa.1 pseudofs.1 procfs.1 msdosfs.1 usb.1 cdce.0 random.1 ppbus.1 pci.1
pccard.1 null.1 mpt_raid.1 mpt.1 mpt_cam.1 mpt_core.1 miibus.1 mem.1 isp.1
sbp.1 fwe.1 firewire.1 exca.1 cardbus.1 ast.1 afd.1 acd.1 ataraid.1 atapci.1
ad.1 ata.1 ahc.1 ahd.1 ahd_pci.1 ahc_pci.1 ahc_pci.1 ahc_isa.1 ahc_eisa.1 scsi_low.1
❹cam.1
  0xaaa000: ❺/boot/kernel/snd_via8233.ko (elf module, 0x6228)
    modules: snd_via8233.1
```

```
0xab1000: ❻/boot/kernel/sound.ko (elf module, 0x23898)
  modules: sound.1
0xad5000: ❼/boot/kernel/atapicam.ko (elf module, 0x4bac)
  modules: atapicam.1
```

Хотя некоторые из этих сведений имеют ценность только для разработчиков, тем не менее много интересного они могут дать и системному администратору. Наиболее интересная информация – это путь к загруженному ядру ❶. Это всегда должен быть */boot/kernel/kernel*, если загрузчик не настроен на загрузку ядра из какого-либо другого места.

Вы также получаете список модулей, подключаемых при каждой загрузке файла ядра ❷. В данном примере приводится список модулей непосредственно из самого ядра, начиная от elink ❸ и заканчивая cam ❹. Загрузчик также загрузил файлы *snd_via8233* ❺, *sound* ❻ и *atapicam* ❼ с соответствующими им модулями.

Чтобы полностью стереть из памяти загруженное ядро вместе со всеми модулями, используйте команду `unload`.

```
OK unload
```

Никакого подтверждения успешного выполнения этой операции вы не получите, но последующая команда `lsmod` продемонстрирует, что загрузчик не имеет никакой информации о файлах ядра.

Для загрузки другого ядра используется команда `load`.

```
OK load boot/kernel.good/kernel
boot/kernel.good/kernel text=0x4a6324 data=0x84020+0x9908c
syms=[0x4+0x67220+0x4+0x7e178]
```

В ответ загрузчик сообщит имя файла и выведет кое-какую дополнительную техническую информацию.

Хотя я и коснулся здесь вопроса загрузки альтернативного ядра, тем не менее, прежде чем делать это, вы должны четко понимать, зачем это необходимо и как безопаснее это сделать. Перейдите к главе 5 и прочитайте обсуждение в разделе «Загрузка альтернативного ядра».

После того как вы загрузите систему необходимым вам способом, вам может потребоваться сохранить параметры загрузки для последующего использования. Система FreeBSD позволяет сделать это с помощью файла */boot/loader.conf*. Однако прежде чем вносить какие-либо изменения, вы должны понять, какие конфигурационные файлы используются системой по умолчанию.

Файлы по умолчанию

В системе FreeBSD файлы с настройками подразделяются на файлы по умолчанию и изменяемые файлы. Файлы по умолчанию содержат присваивания переменных. Эти файлы не предназначены для редактирования администратором; их роль – подмена файлами с теми же

именами. Файлы с настройками по умолчанию хранятся в каталоге, который так и называется – *defaults*.

Например, параметры настройки загрузчика хранятся в файле */boot/loader.conf*, а настройки загрузчика по умолчанию – в файле */boot/defaults/loader.conf*. Если вам потребуется узнать полный список переменных загрузчика, загляните в файл по умолчанию.

Процесс обновления полностью заменит файлы по умолчанию, при этом ваши локальные файлы с настройками останутся нетронутыми. Такое разделение позволяет оставлять локальные изменения нетронутыми и добавлять в систему новые значения. С выходом каждой новой версии в систему FreeBSD добавляются новые возможности, и их разработчики прикладывают немалые усилия к тому, чтобы изменения этих файлов были обратно совместимы. Это означает, что вы не должны проходить по обновленной конфигурации и вручную объединять все изменения. Надо лишь проверить новый файл по умолчанию на предмет появления новых возможностей конфигурирования.

Прекрасным примером таких файлов может служить файл с настройками загрузчика. Файл */boot/defaults/loader.conf* содержит десятки записей следующего вида:

```
verbose_loading="NO"      # Установите YES, чтобы получать подробный отчет
```

Переменная `verbose_loading` по умолчанию имеет значение `NO`. Чтобы изменить ее, не нужно редактировать файл */boot/defaults/loader.conf* – добавьте строку в файл */boot/loader.conf* и измените значение там. Значения из файла */boot/loader.conf* переопределяют значения параметров настройки по умолчанию, и локальные файлы с настройками могут содержать только ваши локальные изменения. Сисадмин легко сможет просмотреть, какие изменения были произведены и чем данная конфигурация системы отличается от конфигурации «из коробки».

Механизм настроек по умолчанию широко используется в системе FreeBSD, особенно в основной ее части.

Не копируйте настройки по умолчанию!

Одна из распространенных ошибок состоит в том, что некоторые администраторы копируют настройки по умолчанию в локальные файлы и затем производят в них изменения. Такое копирование вызывает определенные проблемы в некоторых частях системы. Неприятностей, возможно, удастся избежать в случае с парой каких-нибудь файлов, но рано или поздно проблемы вас догонят. Например, копирование файла */etc/defaults/rc.conf* в */etc/rc.conf* приведет к тому, что система не сможет загрузиться. Вам следует быть осторожными.

Настройка загрузчика

Чтобы сделать изменения в настройках загрузчика постоянными, следует сохранять их в файле */boot/loader.conf*. Параметры настройки из этого файла считываются самим загрузчиком при запуске системы. (Безусловно, если вам нравится при каждой загрузке системы ковыряться в командной строке загрузчика, то вам нет смысла беспокоиться об этом!)

Если заглянуть в настройки загрузчика по умолчанию, можно увидеть множество параметров, которые напоминают переменные, перечисленные в загрузчике. Например, здесь можно определить имя устройства консоли:

```
console="vidconsole"
```

По всей документации к FreeBSD разбросаны ссылки на *настройки времени загрузки* и *настройки загрузчика*. Все эти настройки находятся в файле *loader.conf*. В число этих настроек входят и значения systctl, которые после запуска системы доступны только для чтения. (Подробнее об этом будет рассказываться в главе 5. В приложении приводится список наиболее часто использующихся параметров sysctl ядра.) Ниже приводится пример присваивания значения `32` переменной ядра `kern.maxusers`.

```
kern.maxusers="32"
```

Некоторые из этих переменных не имеют конкретного значения в файле *loader.conf* – вместо значения используются пустые кавычки. Это означает, что загрузчик предлагает сделать выбор значения ядру, но если у вас появится необходимость переопределить значение, устанавливаемое ядром, у вас будет такая возможность.

```
kern.nbuf=""
```

Ядро знает, какое значение выбрать для `kern.nbuf`, но вы с помощью загрузчика, можете указать свое значение.

Тонкую настройку системы с использованием загрузчика мы будем рассматривать в соответствующем разделе – например, значения параметров настройки ядра будут обсуждаться в главе 5, где это обсуждение более уместно по смыслу. А здесь будут рассматриваться наиболее общие настройки, которые оказывают влияние на образ действий и внешний вид самого загрузчика, а также на весь процесс загрузки. С развитием FreeBSD разработчики вводили новые и изменяли функциональность старых параметров настройки, поэтому обязательно загляните в ваш файл */boot/defaults/loader.conf*, чтобы увидеть полный список параметров.

`boot_verbose="NO"`

Изменяет степень подробности протоколирования процесса загрузки, аналогичного поведения можно добиться, выбрав соответствую-

щий пункт меню загрузки. В стандартном режиме ядро выводит лишь самые основные сведения о каждом из идентифицированных им устройств. В режиме подробного протоколирования ядро будет выводить полную информацию о каждом устройстве, а также подробную информацию о параметрах настройки этих устройств в ядре. Это может быть полезно для отладки и разработки, но не для повседневного использования.

`autoboot_delay="10"`

Это число секунд между появлением меню и автоматическим запуском загрузки. Я часто уменьшаю это значение до 2–3 секунд, чтобы мои компьютеры загружались как можно быстрее.

`beastie_disable="NO"`

Этот параметр управляет внешним видом меню (первоначально меню украшает изображение «Beastie» – талисмана BSD, нарисованное символами ASCII). Если установить значение `"YES"`, талисман выводиться не будет.

`loader_logo="fbsdbw"`

С помощью этого параметра можно определить, какой логотип будет выводиться справа от меню загрузки. Значению по умолчанию `fbsdbw` соответствует надпись «FreeBSD», художественно оформленная символами ASCII. В качестве других значений можно использовать `beastiebw` (оригинальный логотип), `beastie` (цветной логотип) и `none` (меню выводится вообще без логотипа).

Последовательные консоли

Любая консоль – вещь сама по себе хорошая, но как быть в случае появления проблем, когда ваша система FreeBSD находится от вас на большом удалении – на противоположном конце страны или на другом континенте? Клавиатура и монитор тоже штука хорошая, но во многих информационных центрах для них просто нет места. И как перезагрузить машину, когда она не отвечает на запросы по сети? Последовательная консоль поможет решить все эти проблемы и не только их.

Последовательная консоль просто направляет ввод с клавиатуры и вывод видео в последовательный порт, а не на клавиатуру и монитор. Последовательные консоли присутствуют во всех типах сетевого оборудования, начиная от сетевых маршрутизаторов и коммутаторов Cisco и заканчивая сетевыми мультиплексорами KVM. Многие устройства обеспечения безопасности, такие как дверные замки с клавиатурой, также имеют последовательные консоли. Соединив последовательные порты двух компьютеров стандартным нуль-модемным кабелем компьютера, вы из одной системы получите доступ к загрузочным сообщениям второй системы. Это особенно удобно, если компьютеры находятся на большом удалении друг от друга. Чтобы получить доступ

к последовательной консоли, ваш компьютер должен иметь последовательный порт. С каждым годом растет выпуск систем, «неотягощенных наследством», в которых отсутствуют такие основные аппаратные элементы, как последовательные порты и порты PS/2 для подключения клавиатуры и мыши.

Последовательные консоли могут быть как аппаратными, так и программными.

Аппаратные последовательные консоли

Настоящие аппаратные средства UNIX (такие как Sparc64) имеют аппаратные последовательные консоли. В этих системах можно подключить последовательный кабель к порту последовательной консоли и получить свободный доступ к аппаратным настройкам, загрузочным сообщениям и параметрам запуска. Большинство аппаратных средств x86 не предоставляют такой возможности – необходимо сидеть за клавиатурой и следить за монитором, чтобы зайти в BIOS или нажать пробел для прерывания загрузки. Такой функциональностью обладает лишь малая часть системных плат x86 и amd64, но все большее и большее число производителей, таких как Dell или HP предлагают порты последовательных консолей как дополнительную особенность своих компьютеров – но это специальная функция, которую не сразу найдешь. (Поддержка последовательной консоли в HP RILOE позволяет даже управлять питанием компьютера с помощью последовательной консоли, что приятно.)

Если ваш компьютер не имеет последовательной консоли, ни одна операционная система не предоставит вам доступ к PC-совместимым сообщениям BIOS через последовательный порт. Такие сообщения появляются до запуска операционной системы и обращения к жестким дискам. Для преодоления этих трудностей есть аппаратные решения. Самое лучшее из тех, что я видел, это PC Weasel (*http://www.realweasel.com*). Это видеоплата с последовательным портом вместо видеопорта. Подсоединив последовательный кабель к Weasel, можно манипулировать BIOS удаленно, прерывать загрузку для перехода в однопользовательский режим и выполнять любые операции в системе, как будто вы находитесь за консолью.

Аппаратные консоли не требуют обеспечения поддержки со стороны операционной системы.

Программные последовательные консоли

Если вам нужен доступ не к ранним сообщениям BIOS, а только к сообщениям загрузчика, то вполне достаточно будет программной последовательной консоли, реализованной в системе FreeBSD. При загрузке система FreeBSD решает, куда выводить сообщения и откуда принимать ввод. По умолчанию это монитор и клавиатура. Однако с помощью несложных манипуляций можно настроить консоль на последовательном порте, который должен быть в системе. Вы не сможете по-

лучить доступ к BIOS, но эта последовательная консоль даст вам возможность влиять на процесс загрузки. FreeBSD позволяет настраивать последовательную консоль в двух различных местах. Для обычных рабочих систем лучше всего поместить настройки последовательной консоли в файл */boot/config*. Это обеспечит вам доступ к самым ранним стадиям процесса загрузки. У вас есть три варианта: использовать в качестве консоли стандартные клавиатуру/монитор/мышь, использовать в качестве консоли последовательный порт или использовать двойную консоль. Стандартный вариант настройки консоли используется по умолчанию, поэтому его не нужно настраивать отдельно. Чтобы принудить FreeBSD использовать последовательную консоль, введите `-h` в файл */boot/config*.

Двойные консоли позволяют использовать как стандартные, так и последовательные консоли, в зависимости от потребностей. Однако при этом вам нужно указать, какая из этих консолей будет являться первичной. Дело в том, что существуют низкоуровневые задачи, такие как загрузка с помощью альтернативного загрузчика или вызов отладчика, которые доступны только с первичной консоли, но во всем остальном по своим возможностям консоли совершенно идентичны. Введите `-D` в файл */boot/config*, чтобы разрешить использование консолей обоих типов и назначить первичной стандартную консоль. Введите `-Dh` в файл */boot/config*, чтобы разрешить использование консолей обоих типов и назначить первичной последовательную консоль. Я рекомендую использовать двойную консоль.

Управлять настройками консолей можно также из файла */boot/loader.conf*. Действие записей в этом файле проявляется немного позднее, на заключительной стадии загрузки и запуска ядра. Чтобы использовать только последовательную консоль, добавьте в */boot/loader.conf* следующую запись:

```
console="comconsole"
```

Чтобы переключиться обратно к видеоконсоли, удалите или закомментируйте эту строку. Кроме того, в */boot/loader.conf* можно задать такую строку:

```
console="vidconsole"
```

Вы можете указать необходимость использования обеих консолей, перечислив в значении переменной `comconsole` и `vidconsole`, поместив предпочтительную консоль первой. Ниже приводится вариант настройки с предпочтительной последовательной консолью:

```
console="comconsole vidconsole"
```

В серверной комнате вам может понадобиться переключаться от стандартной консоли к последовательной и обратно. Как правило, я управляю большим количеством машин FreeBSD через последовательную консоль.

Автоматическое обнаружение клавиатуры

В некоторых описаниях FreeBSD, которые можно найти в Интернете, вам могут встретиться сведения об использовании автоматического обнаружения клавиатуры при выборе консоли. Суть идеи состоит в том, что если к компьютеру не подключена клавиатура, это означает, что вы скорее всего используете последовательную консоль. Такой подход замечательно работает, если используются клавиатуры AT или PS/2, но автоматическое обнаружение клавиатуры, подключенной к порту USB, обычно терпит неудачу. Поэтому лучше выполнить настройку двойной консоли и не полагаться на автоматическое обнаружение клавиатуры.

Физическая настройка программной последовательной консоли

Независимо от типа имеющейся у вас последовательной консоли, вам необходимо правильно подключиться к ней. Для этого вам потребуется нуль-модемный кабель (его можно приобрести в любом компьютерном магазине или в интернет-магазине). Позолоченные кабели не стоят тех денег, которые за них просят, но не стоит покупать и самую дешевую модель из имеющихся; когда в чрезвычайных обстоятельствах вам понадобится последовательная консоль, помехи в линии передачи будут совершенно некстати.

Подключите один конец нуль-модемного кабеля к порту последовательной консоли на сервере FreeBSD – по умолчанию это первый последовательный порт (COM1 или sio0, в зависимости от того, к какой операционной системе вы привыкли). Это значение можно изменить с помощью перекомпиляции ядра, но обычно проще использовать значение по умолчанию.

Другой конец нуль-модемного кабеля подключите к открытому последовательному порту другой системы. Я рекомендую либо другую систему FreeBSD (либо иную систему UNIX), либо терминальный сервер. Можно задействовать систему Windows, если это все, что у вас имеется.

Если у вас есть две удаленные машины FreeBSD и вы хотите применять на них последовательные консоли, убедитесь, что каждый из компьютеров имеет два последовательных порта. Возьмите два нуль-модемных кабеля и соедините первый последовательный порт каждого компьютера со вторым последовательным портом другого компьютера. Таким способом вы сможете использовать любой компьютер в качестве консольного клиента другого компьютера. При наличии трех машин можно объединить их в замкнутую кольцевую цепочку. Комбинируя группы по две и три машины, вы сможете разместить последовательные

консоли на любом количестве систем FreeBSD. Я работал с 30–40 машинами FreeBSD в одной комнате, где установка мониторов была бы просто непрактична, а последовательные консоли приносили большую пользу. Однако, как только у вас появится стойка из двух серверов, совершенно нелишним будет вложить деньги в приобретение терминального сервера.[1] Довольно недорого их можно купить на eBay.

Другой вариант заключается в использовании переходников DB9-RJ45 – это позволит подключаться к вашим консолям с помощью кабеля CAT5. Если вы работаете в информационном центре, где доступ людей в машинный зал запрещен, кабели последовательных консолей могут выходить рядом с вашим рабочим столом или в другом доступном месте, на панели с разъемами Ethernet. Большинство современных информационных центров лучше оснащено сетями Ethernet, чем последовательными кабелями.

Использование последовательной консоли

Теперь, когда вы получили основные сведения, настало время настроить клиента для доступа к последовательной консоли. Ниже приводятся ключевые параметры настройки доступа к последовательной консоли, которые необходимо запомнить:

- Скорость 9600 бод
- 8 бит
- Без контроля четности
- 1 стоповый бит

Введите эти значения в любом эмуляторе терминала на клиентском компьютере, и последовательная консоль «просто начнет работать». Эмуляторы терминала можно найти, например, в Windows (HyperTerm – пожалуй, самый известный), Macintosh и в любых других операционных системах. Несколько лет тому назад я часто использовал карманный компьютер Palm с последовательным кабелем для доступа к последовательным консолям.

FreeBSD обращается к линиям последовательной передачи данных с помощью программы tip(1), позволяющей подключаться к удаленной системе, аналогично telnet. Для запуска tip наберите следующую команду, обладая правами пользователя root:

```
# tip portname
```

Имя порта (*portname*) – это сокращение для указания номера последовательного порта и скорости, на которой он работает. Файл */etc/remote* содержит список имен портов. Большинство записей в этом файле являются пережитком тех дней, когда протокол UUCP был основным

[1] Устройство типа KVM будет значительно дешевле терминального сервера. – *Прим. научн. ред.*

протоколом передачи данных, а последовательные терминалы были нормой, а не исключением из правил.[1] Однако в конце этого файла есть несколько таких записей:

```
# Finger friendly shortcuts
sio0|com1:dv=/dev/cuad0:br#9600:pa=none:
sio1|com2:dv=/dev/cuad1:br#9600:pa=none:
sio2|com3:dv=/dev/cuad2:br#9600:pa=none:
sio3|com4:dv=/dev/cuad3:br#9600:pa=none:
sio4|com5:dv=/dev/cuad4:br#9600:pa=none:
sio5|com6:dv=/dev/cuad5:br#9600:pa=none:
sio6|com7:dv=/dev/cuad6:br#9600:pa=none:
sio7|com8:dv=/dev/cuad7:br#9600:pa=none:
```

Названия `sio` соответствуют стандартным именам устройств в UNIX, тогда как имена `com` добавлены для удобства тех, кто вырос на аппаратном обеспечении x86. Предположим, что у вас есть две машины FreeBSD, подсоединенные друг к другу. С помощью нуль-модемного кабеля последовательный порт 1 одной машины соединен с последовательным портом 2 другой. Пусть вы хотите обратиться к локальному последовательному порту 2 для взаимодействия с последовательной консолью другой системы:

```
# tip sio1
connected
```

Что бы вы ни набрали, ничего другого вы не увидите.

Если вы зарегистрируетесь на другой системе и перезагрузите ее, то в окне tip вдруг появится следующее:

```
Shutting down daemon processes:.
Stopping cron.
Shutting down local daemons:.
Writing entropy file:.
Terminated
.
Waiting (max 60 seconds) for system process 'vnlru' to stop...done
Waiting (max 60 seconds) for system process 'bufdaemon' to stop...done
Waiting (max 60 seconds) for system process 'syncer' to stop...
Syncing disks, vnodes remaining...1 0 0 done
All buffers synced.
Uptime: 1m1s
Shutting down ACPI
Rebooting...
```

Далее при загрузке системы наступит долгая пауза. Если вы находитесь рядом с системой, то сможете увидеть пробегающие стандартные сообщения BIOS. Наконец, вы увидите примерно следующее:

---

[1] Они появились позже динозавров, но до того, как изобрели спам. Только представьте.

```
/boot/kernel/kernel text=0x4a6324 data=0x84020+0x9908c
syms=[0x4+0x67220+0x4+0x7e178]
/boot/kernel/snd_via8233.ko text=0x3a14 data=0x328
syms=[0x4+0xa10+0x4+0xac5]
loading required module 'sound'
/boot/kernel/sound.ko text=0x17974 data=0x37a8+0x10d8
syms=[0x4+0x3290+0x4+0x3d7d]
/boot/kernel/atapicam.ko text=0x2a30 data=0x1d8+0x4
syms=[0x4+0x7b0+0x4+0x7d6]
```

Это говорит о том, что загрузчик отыскал и загрузил файлы ядра, прежде чем вывести загрузочное меню. Поздравляю! Вы пользуетесь последовательной консолью. Нажмите клавишу пробела, чтобы прервать процесс загрузки. Неважно, как далеко система находится от вас, вы можете изменить загружаемое ядро, получить подробный отчет о загрузке, перейти в однопользовательский режим или вручную проверить файловые системы с помощью fsck, одним словом – сделать все, что угодно. Программная последовательная консоль может не обеспечивать доступ к BIOS, но к этому моменту BIOS уже загрузилась. Когда вы обладаете определенным опытом пользования последовательной консолью, вам совершенно неважно, где находится управляемая система – в противоположном углу комнаты или на другом краю мира, и вам скорее покажется тяжелой работой подняться со стула только для того, чтобы подойти к обычной консоли.

Если продолжить загрузку системы, то через некоторое время поток сообщений остановится и последовательная консоль прекратит свое функционирование. Это происходит потому, что последовательная консоль не предназначена для входа в систему. (Возможность входить в систему с помощью последовательной консоли иногда бывает полезной – подробнее об этом рассказывается в главе 20.)

Если удаленная система полностью заблокирована, вы можете подключить к ней свою последовательную консоль и «протянуть руки» к выключателю, управляющему электропитанием системы. Это может иметь нежелательные последствия для компьютера, но находиться в заблокированном состоянии тоже не очень хорошо. С помощью последовательной консоли можно выполнить загрузку в однопользовательском режиме и определить источник проблемы, покопавшись в файлах протоколов или где-либо еще, что может иметь отношение к неисправности. Подробнее об устранении неисправностей будет говориться в главе 21.

Выключение последовательной консоли

В качестве управляющего символа программа tip(1) использует тильду (~). Чтобы выключить последовательную консоль, можно в любой момент ввести последовательность выключения «тильда-точка»:

```
~.
```

Вы грациозно отключитесь.

Сообщения на запуске системы

Во время загрузки FreeBSD выводит сообщения с указанием информации о подключенных аппаратных устройствах, версии операционной системы и сведений о запуске различных программ и служб. Эти сообщения имеют большое значение при первой установке системы или во время поиска неисправностей. Сообщения этапа загрузки всегда начинаются с одного и того же – с перечисления авторских прав, принадлежащих проекту FreeBSD и членам правления Калифорнийского университета:

```
Copyright (c) 1992-2007 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
     The Regents of the University of California. All rights reserved.
FreeBSD 7.0-CURRENT-SNAP010 #0: Tue Dec 13 11:25:44 UTC 2005
     root@harlow.cse.buffalo.edu:/usr/obj/usr/src/sys/GENERIC
```

Помимо этого выводятся номер версии загружаемой системы FreeBSD, а также дата и время ее сборки. Здесь также сообщается, кто компилировал ядро, на каком компьютере была выполнена сборка и даже имя каталога, в котором производилась сборка ядра. Если у вас имеется большое число ядер, эта информация будет для вас бесценной, поскольку она поможет идентифицировать доступные функциональные особенности системы.

```
WARNING: WITNESS option enabled, expect reduced performance.

(ВНИМАНИЕ: активирован параметр WITNESS, ожидается снижение
производительности.)
```

В процессе начальной загрузки ядро выводит диагностические сообщения. Сообщение выше говорит о том, что в данном конкретном ядре я активировал отладочный программный код и код диагностики, что в результате может привести к снижению производительности. В данном случае меня не интересуют проблемы производительности. Причины этого вскоре станут понятны.

```
Timecounter "i8254" frequency 1193182 Hz quality 0
```

Это сообщение идентифицирует одно из устройств в аппаратном окружении. *Таймер* (*timecounter*), или *аппаратные часы* (*hardware clock*) – это специальное устройство; при том, что оно совершенно необходимо вашему компьютеру, оно практически недоступно конечному пользователю. Время от времени вам будут встречаться сообщения о подобных аппаратных устройствах, которые непосредственно недоступны пользователю, но имеют жизненно важное значение для системы. Например, среди сообщений можно увидеть информацию о центральном процессоре:

```
CPU:AMDAthlon(tm) 64 X2 ❶Dual Core Processor 4200+ (❷2200.10-MHz 686-class
CPU)
  Origin = "AuthenticAMD" Id = 0x20fb1 Stepping = 1
```

```
❸Features=0x178bfbff<FPU,VME,DE,PSE,TSC,MSR,PAE,MCE,CX8,APIC,SEP,MTRR,PGE,M
CA,CMV,PAT,PSE36,CLFLUSH,MMX,FXSR,SSE,SSE2,HTT>
  Features2=0x1<SSE3>
❹ AMD Features=0xe2500800<SYSCALL,NX,MMX+,FFXSR,LM,3DNow+,3DNow>
  AMD Features2=0x3<LAHF,CMP>
  Cores per package: 2
```

Возможно, вы не в курсе, что простой процессор может иметь такое число характеристик? А теперь о причинах, по которым меня не волнует снижение производительности, обусловленное параметром `WITNESS`, о котором упоминалось выше: в этом компьютере используется двухъядерный процессор ❶, каждое ядро обладает достаточно высокой скоростью ❷ и обладает множеством особенностей, важных для современных микропроцессоров ❸, а также некоторыми дополнительными особенностями, характерными для микропроцессоров компании AMD ❹. То есть у меня имеется достаточный запас мощности[1] и немалый объем памяти.

```
❶real memory = 1072693248 (1023 MB)
❷avail memory = 1040453632 (❸992 MB)
```

Истинный объем памяти (real memory) ❶ – это объем RAM (оперативной памяти), физически установленной в компьютере, а доступный объем памяти (avail memory) ❷ – это объем RAM, свободной после загрузки ядра. В моем компьютере для реальной работы осталось доступно 992 Мбайта RAM ❸, чего более чем достаточно для загрузки системы.

```
FreeBSD/SMP: Multiprocessor System Detected: 2 CPUs
```

Ядро также выводит информацию об обнаруженных аппаратных устройствах и о том, как оно будет обслуживать эти устройства. Например, в отрывке выше ядро сообщает, что были обнаружены оба ядра микропроцессора и оно готово управлять ими.

```
❶ioapic0 <Version 0.3> ❷irqs 0-23 on motherboard
❸ioapic1 <Version 0.3> irqs 24-47 on motherboard
```

Это довольно типичная запись, которую выводит драйвер устройства. Данное устройство известно как ioapic. Ядро определило, что оно имеет номер версии 0.3, и вывело дополнительную информацию о нем ❷. Кроме того, были обнаружены два устройства этого типа, которые получили порядковые номера 0 ❶ и 1 ❸. (Все драйверы устройств получают порядковые номера, начиная с нуля.) Вы можете получить дополнительную информацию об устройстве, обслуживаемом данным драйвером, прочитав страницу руководства к драйверу. Не для всех, но почти для всех драйверов устройств имеются страницы руководства.

```
npx0: [FAST]
npx0: <math processor> on motherboard
npx0: INT 16 interface
```

[1] Я мог бы сказать: «Завидуйте!», но к тому моменту, когда эта книга попадет на книжные полки, мой ноутбук безвозвратно устареет.

Не все драйверы устройства выводят свою информацию в одной строке. Во фрагменте выше приводится информация о единственном устройстве npx0, которая занимает три строки. Единственный способ узнать, что речь идет лишь об одном математическом процессоре, а не о трех, состоит в том, чтобы проверить порядковый номер устройства. В данном случае все три номера равны нулю, следовательно, это одно и то же устройство.

```
❶acpi0: <PTLTD RSDT> on motherboard
❷pcib0: <ACPI Host-PCI bridge> port 0xcf8-0xcff ❸on acpi0
❹pci0: <ACPI PCI bus> on ❺pcib0
```

Довольно интересно, что в загрузочных сообщениях приводится информация о том, как взаимосвязаны различные компоненты вашего компьютера. Здесь мы видим, что система ACPI находится непосредственно на материнской плате ❶, а мост (bridge) PCI ❷ подключен к acpi0 ❸. Здесь также видно, что шина PCI ❹ подключена к мосту PCI ❺, а продолжив просмотр сообщений, вам удастся определить, какие устройства подключены к этой шине. Возможно, сейчас эта информация даст вам не так много, но позднее, когда вам потребуется найти источник проблем, она станет для вас весьма ценной.

Все это хорошо, но эта информация малопригодна для использования в настоящий момент. А есть ли что-нибудь, чем можно было бы воспользоваться прямо сейчас?

```
❶firewire0: <IEEE1394(FireWire) bus> on fwohci0
❷fwe0: <Ethernet over FireWire> on firewire0
```

В этой системе имеется устройство FireWire ❶, и FreeBSD оказалась в состоянии идентифицировать и использовать его! Вот это действительно полезная информация, по крайней мере для тех, у кого имеется оборудование FireWire. Система FreeBSD поддерживает работу Ethernet через FireWire ❷? Это круто[1]!

Даже если у вас нет FireWire, есть вероятность, что у вас имеется некоторое устройство для подключения к сети.

```
❶re0: <RealTek 8169SB Single-chip ❷Gigabit Ethernet> port 0x1000-0x10ff mem
0xd2206800-0xd22068ff irq 19 at device 8.0 on pci0
```

Эта запись сообщает, что сетевая карта была связана с сетевым интерфейсом re0 ❶, что она поддерживает скорость обмена до одного гигабита ❷. Здесь также приводится дополнительная информация об адресах памяти, номере IRQ и шине PCI, к которой она подключена.

---

[1] Драйвер системы FreeBSD по умолчанию обеспечивает работу Ethernet через FireWire методом, который поддерживается только FreeBSD, главным образом потому, что она была первой, реализовавшей такую поддержку. Чтобы использовать Ethernet через FireWire способом, совместимым с Mac OS X и Windows XP, прочитайте страницу руководства fwip(4).

DMESG.BOOT

Информация, которая выводится на этапе загрузки, может быть полезна, но к тому моменту, когда она вам понадобится, она, скорее всего, скроется за пределами экрана. По этой причине загрузочные сообщения сохраняются в файле */var/run/dmesg.boot*. Это означает, что вы сможете увидеть сообщения ядра с информацией об аппаратном окружении даже через много месяцев после запуска системы.

Для каждого устройства в компьютере выводится одна или более записей, напоминающие те, что были показаны выше. Все вместе они довольно подробно описывают аппаратное окружение компьютера. Если выполнить загрузку в режиме подробного протоколирования, вы получите еще больше подробностей – возможно, намного больше, чем вам хотелось бы.

Одно важное обстоятельство заключается в том, что ядро отображает в загрузочных сообщениях названия устройств для каждого компонента аппаратного окружения. Эта информация важна для управления системой. У каждого аппаратного компонента имеется имя файла устройства, которое потребуется вам для его настройки. Например, ранее мы видели, что сетевой карте Ethernet было присвоено имя re0. Эта карта обслуживается драйвером re(4) и первый экземпляр этого драйвера имеет порядковый номер 0. Последующие сетевые карты аналогичного типа получат имена re1, re2 и т. д.

Большинство устройств, которые могут настраиваться и управляться, имеют соответствующие записи в каталоге */dev*. Например, наша сетевая карта представлена файлом */dev/net/re0*. Эти файлы называются *файлами устройств* и обеспечивают удобный способ обращения к различным аппаратным компонентам. К большинству файлов устройств невозможно обратиться как к обычным файлам – вы не сможете прочитать содержимое файла устройства с помощью команды cat(1) или скопировать в него содержимое другого файла. Однако имена файлов устройств можно передавать в качестве параметров командной строки специализированным программам. Например, жесткому диску, который во время загрузки был обнаружен как ad4, соответствует файл устройства */dev/ad4*. Когда вам потребуется смонтировать этот жесткий диск, вы можете использовать имя файла устройства и гарантировать тем самым, что будет смонтировано именно это устройство.

Запуск в многопользовательском режиме

Помимо однопользовательского режима существует еще и многопользовательский режим. Это стандартный режим работы UNIX-подобных

операционных систем. Если система выполняет реальную работу, значит она находится в многопользовательском режиме.

Как только FreeBSD завершает поиск аппаратных устройств и подключение соответствующих драйверов, она запускает сценарий командного интерпретатора */etc/rc*. Этот сценарий монтирует все файловые системы, запускает сетевые интерфейсы, настраивает устройства, идентифицирует доступные разделяемые библиотеки и производит все остальные действия, необходимые для подготовки системы к работе в нормальном режиме. Различные системы на запуске предъявляют различные требования. Несмотря на то, что практически любой сервер должен смонтировать жесткий диск, тем не менее требования, которые предъявляет веб-сервер, существенно отличаются от требований, которые предъявляет сервер баз данных, даже если они работают в совершенно идентичном аппаратном окружении. Это означает, что файл */etc/rc* должен быть максимально гибким. Достигается это за счет передачи некоторых полномочий другим сценариям командного интерпретатора, ответственным за отдельные аспекты системы.

Сценарий */etc/rc* контролируется файлами */etc/defaults/rc.conf* и */etc/rc.conf*.

/etc/rc.conf и /etc/defaults/rc.conf

Как и конфигурационный файл загрузчика, настройки, используемые сценарием */etc/rc*, размещаются в двух файлах: настройки по умолчанию – в файле */etc/defaults/rc.conf* и локальные настройки – в файле */etc/rc.conf*. Настройки в файле */etc/rc.conf* имеют более высокий приоритет перед любыми настройками в файле */etc/defaults/rc.conf* точно так же, как и в случае с загрузчиком.

Файл */etc/defaults/rc.conf* огромен. Он содержит довольно много переменных, часто называемых *knobs* (*кнопки*) или *tunables* (*элементы настройки*). Здесь не рассматриваются все эти переменные – не только потому, что knobs постоянно добавляются (такой список немедленно устареет), но также потому, что довольно много переменных не находит широкого применения на серверах. В стандартной системе FreeBSD переменной *rc.conf* может быть почти все – от раскладки клавиатуры до поведения TCP/IP. Полный перечень переменных вы найдете в своей системе, на странице руководства rc.conf(5).

В следующих разделах будут рассмотрены типичные записи из */etc/rc.conf*. Каждая запись представлена в */etc/defaults/rc.conf* один раз. Редактирование той или иной записи означает ее замену в */etc/rc.conf*. Для каждой переменной будет приводиться значение по умолчанию.

Параметры запуска

Следующие несколько параметров *rc.conf* управляют конфигурированием самой системы FreeBSD и тем, как она запускает другие про-

граммы. Эти параметры определяют, как будут запускаться все остальные программы и службы в системе.

Если у вас обнаруживаются проблемы с запуском самого сценария */etc/rc* и подчиненных ему сценариев, вам, возможно, потребуется разрешить их работу в отладочном режиме. Это позволит вам получить дополнительную информацию, способную прояснить, почему сценарий не запускается.

```
rc_debug="NO"
```

Если вам не нужен полный вывод в отладочном режиме, а хотелось бы получить лишь некоторую дополнительную информацию о работе сценария */etc/rc*, активируйте вывод информационных сообщений с помощью переменной `rc_info`:

```
rc_info="NO"
```

Одна из распространенных проблем, связанных с нехваткой памяти, – недостаточный объем пространства для свопинга. Подробнее о пространстве для свопинга мы поговорим в главе 19, но вы можете настроить использование дополнительного пространства для свопинга непосредственно на этапе загрузки системы:

```
swapfile="NO"
```

Параметры файловой системы

Система FreeBSD может использовать память как файловую систему, о чем подробно будет рассказываться в главе 8. Обычно эта возможность используется для создания файловой системы */tmp* в памяти, так как операции с памятью выполняются существенно быстрее, чем с жестким диском. Прочитав главу 8, вы сможете реализовать это на практике. В файле *rc.conf* имеются переменные, которые позволят вам создать файловую систему */tmp* в памяти и определить ее размер. Вы можете также указывать параметры, которые используются системой FreeBSD для работы с полноценными файловыми системами. (Наиболее нетерпеливые из вас могут озадачиться, что означает флаг `-S`. Этот флаг запрещает использование механизма *Soft Updates*. Если вы представления не имеете, о чем идет речь, тогда терпите до главы 8.) Если у вас появится необходимость реализовать файловую систему */tmp* в памяти, установите параметр `tmpmfs` в значение `YES` и определите желаемый размер файловой системы */tmp* с помощью параметра `tmpsize`.

```
tmpmfs="AUTO"
tmpsize="20m"
tmpmfs_flags="-S"
```

Еще одна популярная возможность файловых систем FreeBSD – это собственная реализация шифруемых разделов. Система FreeBSD «из коробки» поддерживает две различных файловых системы с шифрованием: GBDE и GELI. Файловая система с *шифрованием диска на низком*

уровне (*Geom Based Disk Encryption, GBDE*) была первой шифруемой файловой системой, предназначенной для использования в военном ведомстве. GELI – немного более дружественная технология, чем GBDE, и в отличие от GBDE соответствует иным стандартам. (Вам определенно стоит прочитать главу 18, прежде чем вы решитесь активировать один из этих механизмов!)

```
gbde_autoattach_all="NO"
gbde_devices="NO"
gbde_attach_attempts="3"
gbde_lockdir="/etc"
geli_devices=""
geli_tries=""
geli_default_flags=""
geli_autodetach="YES"
geli_swap_flags="-a aes -l 256 -s 4096 -d"
```

После перехода в многопользовательский режим FreeBSD по умолчанию монтирует корневой раздел в режиме чтения/записи. Если вам необходимо обеспечить доступ к корневому разделу в режиме только для чтения, вы можете установить следующую переменную в значение `NO`. Многие считают, что так безопаснее, но это может вступать в противоречие с потребностями некоторых программных продуктов и, кроме того, это помешает вам редактировать какие-либо файлы, расположенные в корневом разделе!

```
root_rw_mount="YES"
```

Когда во время загрузки FreeBSD пытается смонтировать файловые системы, она проверяет их внутреннюю целостность. Если ядро обнаруживает существенные проблемы, оно пытается автоматически исправить их с помощью `fsck -y`. Хотя в некоторых ситуациях это бывает совершенно необходимо, тем не менее в общем случае эта операция далеко не безопасна. (Обязательно прочтите внимательно главу 8, прежде чем активировать эту возможность!)

```
fsck_y_enable="NO"
```

Ядро может также обнаруживать незначительные неполадки в файловых системах, которые могут быть устранены «на лету» посредством запуска *fsck в фоновом режиме*, в то время как система работает в многопользовательском режиме, о чем подробно рассказывается в главе 8. При определенных обстоятельствах эта возможность может вызывать проблемы с безопасностью. Вы можете контролировать использование утилиты fsck в фоновом режиме и определять, как долго система должна ожидать, прежде чем запустить fsck в фоне.

```
background_fsck="YES"
background_fsck_delay="60"
```

Разнообразные сетевые демоны

В состав FreeBSD входит ряд небольших программ (или демонов), которые работают в фоновом режиме и обслуживают соответствующие сервисы. Мы будем рассматривать многие из этих сервисов на протяжении всей книги, но здесь упомянем некоторые настройки, которые будут интересны опытным системным администраторам. Один из наиболее известных демонов – syslog(8). Протоколирование работы системы – это Самое Мудрое Решение. Файлы журналов занимают настолько важное место, что большая часть главы 20 посвящена теме протоколирования средствами FreeBSD и для FreeBSD.

```
syslogd_enable="YES"
```

Как только вами будет принято решение о запуске демона протоколирования, вы сможете точно определить, как он должен запускаться, установив параметры командной строки. Система FreeBSD будет использовать эти параметры при запуске данного демона. Для всех программ, включенных в *rc.conf*, можно задать параметры командной строки в следующем формате.

```
syslogd_flags="-s"
```

Другой популярный демон – inetd(8), сервер малых сетевых служб. (Демон inetd будет рассматриваться в главе 15.)

```
inetd_enable="NO"
```

Одна из задач, которые обычно решаются с помощью FreeBSD, является обслуживание системы доменных имен (Domain Name System, DNS) с помощью демона named(8). DNS – это дорожная карта Интернета, которая позволяет простым людям пользоваться сетью. Поскольку прежде чем начать приносить пользу, сервер DNS должен быть сначала сконфигурирован, то по умолчанию запуск этого демона запрещен. DNS будет рассматриваться в главе 14.

```
named_enable="NO"
```

Очень часто используется демон Secure Shell (SSH), обеспечивающий безопасное подключение по сети. Если вам необходимо обеспечить сетевое соединение с удаленной системой, вам определенно потребуется служба SSH.

```
sshd_enable="NO"
```

Настройка демона SSH может выполняться через параметры командной строки, однако чаще всего для этих целей используются файлы в каталоге */etc/ssh*. Подробности приводятся в главе 15.

```
sshd_flags=""
```

Операционная система FreeBSD включает в себя самое разнообразное программное обеспечение, которое обеспечивает синхронизацию системных часов с остальным миром. Чтобы извлекать пользу из этого

программного обеспечения, его необходимо настроить. Эту тему мы рассмотрим в главе 15.

```
ntpd_enable="NO"
ntpd_flags="-p /var/run/ntpd.pid -f /var/db/ntpd.drift"
```

FreeBSD включает в себя также небольшой демон SNMP, который обеспечивает работу инструментальных средств управления на базе протокола SNMP. Мы будем рассматривать настройку SNMP в главе 19.

```
bsnmpd_enable="NO"
```

Сетевые параметры

Эти параметры управляют настройкой сетевых функций FreeBSD во время начальной загрузки. Каждый компьютер в Интернете должен иметь имя хоста (hostname). Имя хоста – это полное доменное имя системы, например *www.absolutefreebsd.org*. Многие программы не смогут нормально работать без такого имени.

```
hostname=""
```

В состав FreeBSD входят несколько различных пакетов, предназначенных для реализации межсетевого экрана. Мы коротко рассмотрим пакетный фильтр (Packet Filter, PF) в главе 9. Активация и деактивация пакетного фильтра производится в *rc.conf* с помощью следующей переменной:

```
pf_enable="NO"
```

TCP/IP – это довольно старый сетевой протокол[1], который неоднократно подвергался изменениям и расширениям. Некоторые изменения и дополнения были объединены в «TCP Extensions» (Расширения TCP), как описывается в главе 6. Многие операционные системы могут воспользоваться преимуществами расширений TCP, но наиболее старые на это не способны. Если у вас возникают сложности при взаимодействии с достаточно древними узлами сети, запретите использование расширений TCP.

```
tcp_extensions="YES"
```

Для вас может представлять интерес информация о неудачных попытках подключения сети к вашей системе. Описанная далее возможность поможет вам обнаружить попытки сканирования портов и несанкционированного вторжения, однако наряду с полезной информацией она будет накапливать в протоколах системы массу ненужного мусора. Для интереса можно включить эту функцию на короткое время, просто чтобы посмотреть, что в действительности происходит в вашей сети. (Тем не менее помните, что она *действительно* может вызывать изжогу у администратора.) Установите эту переменную в значе-

[1] Точнее, стек протоколов. – *Прим. перев.*

ние 1, чтобы система записывала в файл протокола информацию о неудачных попытках соединения.

```
log_in_vain="0"
```

Маршрутизаторы используют протокол ICMP для уведомления клиентских компьютеров о наличии сетевых шлюзов для конкретных маршрутов. Хотя существуют допустимые варианты использования ICMP, этот протокол широко используется хакерами для перехвата данных. Если в сети переадресация ICMP не применяется, установка этого параметра незначительно усилит защиту системы. О том, применяется ли переадресация ICMP, можно спросить у сетевого администратора.

```
icmp_drop_redirect="NO"
```

А если *вы и есть* сетевой администратор и не можете ответить на этот вопрос, просто включите протоколирование пакетов переадресации, получаемых вашей системой, в файл */var/log/messages*.[1] Обратите внимание: если ваш сервер атакуют злоумышленники, это быстро может привести к переполнению жесткого диска сообщениями в файле протокола.

```
icmp_log_redirect="NO"
```

Чтобы подключиться к сети, необходимо каждому сетевому интерфейсу присвоить IP-адрес. Некоторые подробности этой процедуры приводятся в главе 6. Получить перечень имеющихся сетевых интерфейсов можно с помощью команды ifconfig(8). Добавьте названия всех сетевых интерфейсов, каждый в отдельной строке, и укажите в кавычках информацию о настройках. Например, чтобы присвоить сетевому интерфейсу em0 IP-адрес 172.18.11.3 и сетевую маску 255.255.254.0, следует ввести следующую строку:

```
ifconfig_em0="inet 172.18.11.3 netmask 255.255.254.0"
```

Если в сети используется DHCP, тогда вместо IP-адреса следует указать значение `dhcp`.

```
ifconfig_em0="dhcp"
```

Аналогичным образом производится назначение сетевому интерфейсу *псевдонима* (*alias*). Псевдоним – это не фактический IP-адрес сетевого интерфейса, но интерфейс будет откликаться на этот IP-адрес, о чем подробнее рассказывается в главе 6. Система FreeBSD в состоянии обеспечить поддержку нескольких тысяч псевдонимов для единственного сетевого интерфейса при помощи записей в *rc.conf* следующего вида:

```
ifconfig_em0_alias number="address netmask 255.255.255.255"
```

[1] Если вы вообще никогда не слышали о перенаправлении ICMP, не просто идите, а бегом бегите в ближайший книжный магазин и купите книгу Чарльза М. Козиерока (Charles M. Kozierok) «The TCP/IP Guide» (No Starch Press, 2005). И сразу же принимайтесь за чтение.

Номера псевдонимов должны охватывать непрерывный ряд чисел и начинаться с 0. Если в порядковых номерах псевдонимов появится разрыв, то псевдонимы с номерами выше разрыва не будут установлены во время загрузки. (Это довольно типичная проблема, и если вы столкнетесь с ней, проверьте список псевдонимов.) Например, псевдоним адреса 192.168.3.4 должен быть определен как:

```
ifconfig_em0_alias0="192.168.3.4 netmask 255.255.255.255"
```

Параметры сетевой маршрутизации

Реализация сетевого стека во FreeBSD включает в себя массу возможностей маршрутизации интернет-трафика. Перечень параметров маршрутизации начинается с таких простых, как определение шлюза по умолчанию. Если IP-адрес обеспечивает нормальную работу в сети, то шлюз по умолчанию обеспечивает доступ ко всем узлам, которые находятся за пределами локальной сети.

```
defaultrouter=""
```

Устройства управления сетью, такие как межсетевые экраны, должны передавать пакеты между различными сетевыми интерфейсами. Эта возможность во FreeBSD по умолчанию выключена, но ее легко можно активировать. Просто сообщите системе, что она будет играть роль шлюза, и она обеспечит связь между несколькими сетями.

```
gateway_enable="NO"
```

Если системе необходима поддержка протокола Routing Information Protocol (протокол маршрутной информации), используйте параметр `router_enable`, чтобы активировать его на запуске системы. Могу заметить, что имя этого параметра не совсем точно отражает его суть – многие маршрутизаторы используют протоколы, отличные от RIP, однако такое название параметр носит уже несколько десятков лет. Если вам в действительности не требуется поддержка протокола RIP, оставьте этот параметр в покое!

```
router_enable="NO"
```

Параметры консоли

Параметры консоли управляют поведением монитора и клавиатуры. С их помощью можно изменить язык клавиатуры, размер шрифта на мониторе и многое-многое другое. Например, по умолчанию используется стандартная раскладка клавиатуры US, которая нередко называется «QWERTY». В каталоге */usr/share/syscons/keymaps* есть довольно много файлов раскладок. Я предпочитаю раскладку Dvorak, которой соответствует файл *us.dvorak*. Если указать в параметре `keymap` значение *us.dvorak*, после загрузки в многопользовательском режиме система будет использовать раскладку клавиатуры Dvorak.

```
keymap="NO"
```

Система FreeBSD гасит монитор спустя определенное время бездействия клавиатуры, которое указано в параметре `blanktime`. Если в этом параметре указать значение `NO`, тогда система вообще не будет гасить экран. Имейте в виду, что современные мониторы сами гасят экран через некоторое время в целях экономии электроэнергии. Если экран гаснет, даже если вы установили параметр `blanktime` в значение `NO`, проверьте настройки BIOS и посмотрите руководство монитора.

```
blanktime="300"
```

Кроме всего прочего, система FreeBSD позволяет использовать в консоли различные шрифты. Шрифты, используемые по умолчанию, вполне пригодны для сервера, однако у вас может появиться желание поменять их на настольном компьютере или на ноутбуке. В моем ноутбуке используется 17-дюймовый экран с пропорциями, позволяющими просматривать видеофильмы, и на таком экране шрифты по умолчанию выглядят довольно неприглядно. У вас есть возможность выбрать любой шрифт из каталога */usr/share/syscons/fonts*. Попробуйте несколько из них и посмотрите, как они будут выглядеть на вашем экране. Названия шрифтов включают в себя и их размеры, и вы можете подобрать подходящую переменную. Например, в файле *swiss-8x8.fnt* находится шрифт Swiss, размером 8 на 8 пикселей. Чтобы использовать его, необходимо установить его имя в параметре `font8x8`.

```
font8x16="NO"
font8x14="NO"
font8x8="NO"
```

Существует возможность использовать мышь в консоли, даже не имея графического интерфейса. По умолчанию FreeBSD сама пытается определить тип мыши. Если вы используете мышь PS/2 или USB, есть вероятность, что она заработает сама собой, без выполнения специальных настроек, как только вы активизируете демон управления мышью. Некоторые устаревшие или необычные типы мыши требуют ручной настройки, порядок которой описывается в странице руководства moused(8).

```
moused_enable="NO"
moused_type="AUTO"
```

Можно также изменить размеры изображения на экране монитора. Если размеры изображения не совпадают с размерами экрана, можно изменить число строк и их длину, изменить цвет текста, изменить форму курсора и его поведение и выполнить все остальные мелкие настройки. Полный список возможных вариантов можно получить в странице руководства vidcontrol(8).

```
allscreens_flags=""
```

Аналогичным образом можно управлять поведением клавиатуры в довольно широких пределах. Настроить можно все, начиная от скорости

автоповтора нажатой клавиши до переназначения функциональных клавиш. За дополнительной информацией обращайтесь к странице руководства kbdcontrol(8).

```
allscreens_kbdflags=""
```

Прочие параметры

Заключительное попурри параметров настройки может быть, а может и не быть, полезным для любого окружения, но они используются достаточно часто, чтобы быть достойными упоминания. Например, не все системы имеют доступ к принтеру, но в тех из них, которые имеют такой доступ, будет необходимо использовать демон печати lpd(8). Мы еще вернемся к настройке принтера в главе 15.

```
ldp_enable="NO"
```

Демон sendmail(8) управляет приемом и передачей электронной почты. Практически все системы должны обладать возможностью отправлять электронную почту, но значительная часть компьютеров, работающих под управлением FreeBSD, не нуждается в возможности принимать электронную почту. Параметр `sendmail_enable` отвечает за настройку обработки входящей электронной почты, а `sendmail_outbound_enable` позволяет системе отправлять почту. Подробнее об этом рассказывается в главе 16.

```
sendmail_enable="NO"
sendmail_outbound_enable="YES"
```

Одна из интересных особенностей FreeBSD – способность запускать программное обеспечение, созданное для других операционных систем. Наиболее часто используется режим совместимости с программным обеспечением, созданным для Linux, но FreeBSD поддерживает возможность запуска двоичных программ для SCO UNIX и SVR4. Эта особенность будет обсуждаться в главе 12. Не активируйте эти режимы совместимости, пока не прочитаете эту главу!

```
linux_enable="NO"
```

Жизненно важная часть любой UNIX-подобной операционной системы – это разделяемые библиотеки. У вас есть возможность определять перечень каталогов, где FreeBSD будет искать необходимые ей библиотеки. Обычно настройки по умолчанию вполне адекватны, но если вы обнаруживаете, что вам регулярно приходится изменять значение переменной окружения `LD_LIBRARY_PATH`, тогда вам следует воспользоваться этим параметром настройки. Дополнительные рекомендации будут даны в главе 12.

```
ldconfig_paths="/usr/lib /usr/X11R6/lib /usr/local/lib"
```

В системе FreeBSD реализована система обеспечения безопасности, которая позволяет администратору управлять базовыми характеристи-

ками системы. Вы можете глобально запретить возможность монтирования жестких дисков, ограничить доступ к определенным портам TCP/IP и даже запретить изменять файлы. Подробнее о том, как использовать эти функции, рассказывается в главе 7.

```
kern_securelevel_enable="NO"
kern_securelevel="-1"
```

Теперь, когда вы получили некоторое представление о параметрах настройки, поддерживаемых системой FreeBSD «из коробки», давайте посмотрим, как можно их использовать на практике.

Система сценариев запуска rc.d

Переход от однопользовательского режима к многопользовательскому система FreeBSD производит с помощью сценария командного интерпретатора */etc/rc*. Этот сценарий читает содержимое конфигурационных файлов */etc/defaults/rc.conf* и */etc/rc.conf* и запускает коллекцию других сценариев, основываясь на полученных параметрах настройки. Например, если был разрешен запуск демона USB, сценарий */etc/rc* запустит другой сценарий, созданный специально для запуска этого демона. В состав FreeBSD входят сценарии, предназначенные для запуска служб, монтирования дисков, настройки соединения с сетью и установки параметров безопасности. Эти сценарии можно использовать для запуска и остановки служб, точно так же, как это делает сама система, за счет чего обеспечивается поддержание целостности системы и упрощается ваша жизнь. Эти сценарии находятся в каталоге */etc/rc.d*.

Активировав функциональные возможности в *rc.conf*, вы сможете управлять ими с помощью сценариев *rc.d*. Например, предположим, что вам потребовалось запустить демон SSH, который раньше не использовался. Установите параметр `sshd_enable` в значение `YES` и перейдите в каталог */etc/rc.d*. Здесь вы найдете сценарий с именем *sshd*.

Что такое rcNG?

Когда-то давно FreeBSD включала в себя пригоршню монолитных сценариев */etc/rc*, которые выполняли настройку всей системы. Каждый демон или служба запускался с помощью нескольких строк, запрятанных в одном из этих сценариев. Для большинства систем такой подход вполне оправдывал себя, но он не отличался особой гибкостью. Современная система маленьких сценариев командного интерпретатора, предназначенных для запуска отдельных служб, была разработана в рамках системы NetBSD и затем быстро перекочевала во FreeBSD. В настоящее время только эта система используется во всех версиях FreeBSD, а любые упоминания об *rcNG* – это лишь отзвуки эпохи перехода.

```
#./sshd start
Starting sshd.
#
```

Ни один сценарий из *rc.d* не будет запущен, если не был активирован соответствующий ему параметр в *rc.conf*. Это гарантирует, что все, что работало до перезагрузки, будет запущено и после нее. Вы можете останавливать работу служб с помощью команды `stop`, проверять состояние с помощью команды `status` и перезапускать их с помощью команды `restart`. Если необходимо запустить программу всего один раз с помощью соответствующего ей сценария в *rc.d* и вам не нужно, чтобы она запускалась после перезагрузки, используйте команду `forcestart`.

Подробнее содержимое каталога *rc.d* будет рассматриваться в главе 12, когда мы будем обсуждать настройку и написание собственных сценариев *rc.d*.

Останов системы

Система сценариев запуска *rc.d* в системе FreeBSD играет двоякую роль – они не только должны производить запуск системных служб, они должны также производить их останов перед выключением питания. Одни сценарии должны демонтировать жесткие диски, другие отвечают за остановку демонов и удаление временных файлов, оставшихся после работы. Некоторые программы не переживают по поводу бесцеремонности остановки, когда система готовится к выключению на ночь – а ведь после остановки системы все клиенты, подключенные через SSH, будут отключены, и любые запрошенные веб-страницы просто не будут доставлены. Однако в случае с базами данных очень важно, как это программное обеспечение будет остановлено, поскольку бесцеремонная остановка может привести к повреждению данных. Многие другие программы, которые имеют дело с данными, также чувствительны к способу остановки: если не позволить им корректно завершить свою работу, потом можно сильно пожалеть об этом.

Когда выполняется остановка системы, с помощью команды shutdown(8) или reboot(8) FreeBSD вызывает сценарий */etc/rc.shutdown*. Этот сценарий вызывает сценарии из *rc.d* с параметром `stop`, в порядке, обратном порядку их запуска на этапе загрузки, давая тем самым серверным программам удалить временные файлы и завершить свою работу нормальным образом.

Теперь, когда вы знаете, как запускается и останавливается система FreeBSD, давайте перейдем к рассмотрению основных инструментальных средств, которые позволят вам обеспечить работоспособность системы даже после ваших экспериментов.

4

Прочтите это раньше, чем что-нибудь испортите! (Резервное копирование и восстановление)

В большинстве случаев сбои в системе происходят по вине человека, но случаются и аппаратные сбои. Хакеры находят все новые способы преодоления сетевой защиты и проникновения в систему через приложения, и вам неизбежно *придется* регулярно обновлять свою систему. (Обновлять ли систему полностью или отдельную ее часть – это уже отдельный вопрос.) Всякий раз, когда вы прикладываете руки к системе, существует опасность допустить ошибку, нарушить работу жизненно важной службы или вообще полностью разрушить систему. Достаточно вспомнить, как часто вы что-то меняли на компьютере (с любой операционной системой) и сталкивались впоследствии со странным поведением системы. Даже минимальные изменения конфигурации потенциально способны повредить данные. Поэтому вы всегда должны быть готовы к худшему. В случае с компьютерами это означает возможность повреждения данных на жестком диске, и вы должны быть готовы восстановить эти данные.

Хуже другое – читатель этой книги, вероятно, еще только изучает, как настраивать систему FreeBSD, и поэтому пока не готов к аварии. Новичку необходимо протестировать множество вариантов конфигурации и проанализировать «историю» изменений в конфигурации системы. Иногда случаются вещи, которые могут вызвать не только высказывание: «Но в последний месяц это работало, что же я изменил?», – но и более бурные эмоции. Сможете ли вы вспомнить все изменения, которые произвели в течение последних недель, месяцев или лет? А что если изменения были внесены вашими коллегами? На самом деле интенсивные эксперименты могут полностью нарушить работу системы, поэтому необходим способ восстановления важных данных.

Эта глава начинается с рассмотрения общего подхода – резервного копирования всего, что есть в компьютере. Однако такой подход недостаточно хорош для сохранения отдельных файлов, поэтому этот случай будет рассмотрен отдельно. Если файл изменяется три раза в день, а резервное копирование происходит раз в неделю, то при потере файла можно потерять важные данные. Кроме того, будут рассмотрены меры по восстановлению и перекомпоновке системы в случае частичных неполадок – с применением однопользовательского режима и загрузочной дискеты для ремонтных работ (fixit).

Резервное копирование системы

Резервное копирование системы необходимо, лишь если данные важны. Это не так глупо, как может показаться. Вопрос, который надо себе задать, таков: «Сколько будет стоить восстановление данных?» Например, недорогая система резервного копирования стоит несколько сотен долларов. Как дорого стоит ваше время и сколько его потребуется, чтобы восстановить систему с установочного носителя? Если самые драгоценные данные на вашем жестком диске – это файл с закладками веб-броузера, тогда, вероятно, нет смысла вкладывать деньги в систему резервного копирования. Но если речь идет о магистральном сервере компании, тогда вы будете относиться к инвестированию денег в систему резервного копирования очень серьезно.

Для операций по полному сохранению и восстановлению необходимы накопитель на магнитной ленте и сама лента. Кроме этого, данные можно сохранять в файлах, проводя резервное копирование по сети, или на съемных носителях, таких как компакт-диски или DVD-диски. Однако современным индустриальным стандартом является магнитная лента, поэтому внимание здесь будет сосредоточено именно на этом решении. Для работы с накопителем на магнитной ленте вам потребуется программа резервного копирования, и мы рассмотрим стандартные программы резервного копирования, которые распространяются в составе FreeBSD.

Накопители на лентах

FreeBSD поддерживает накопители на магнитных лентах с интерфейсами SCSI, USB и IDE. По сравнению с устройствами IDE устройства SCSI быстрее и надежнее, однако устройства IDE дешевле. Накопители с интерфейсом USB не всегда соответствуют стандартам и потому могут не поддерживаться системой FreeBSD. Обязательно ознакомьтесь с примечаниями к выпуску в архивах почтовых рассылок FreeBSD, чтобы убедиться, что ваш накопитель на магнитной ленте совместим с FreeBSD.

После физической установки накопителя на ленте необходимо убедиться, что FreeBSD распознала это устройство. Самый простой путь –

проверить файл */var/run/dmesg.boot*, как рассказывалось в главе 3. Накопители с интерфейсами SCSI и USB отображаются в нем как устройства «sa», тогда как IDE-накопители отображаются как устройства «ast». Например, следующие три строки из файла *dmesg.boot* описывают SCSI-накопитель, подключенный к этому компьютеру:

```
❶sa0 at ❷ahc0 bus 0 target ❸9 lun 0
sa0: <SONY ❹SDT-10000 0110> Removable Sequential Access SCSI-2 device
sa0: ❺40.000MB/s transfers (20.000MHz, offset 8, 16bit)
```

Из всей информации, что мы имеем об этом накопителе на магнитной ленте, самое важное, что FreeBSD опознала его как устройство sa0 ❶. Кроме того, это устройство подключено к SCSI-карте ahc0 ❷ со SCSI ID 9 ❸ и это модель «SONY SDT-10000 0110» ❹ с максимальной производительностью до 40 Мбайт в секунду ❺.

Файлы устройств накопителей на лентах, перемотка и извлечение

Прежде чем вы сможете использовать накопитель на магнитной ленте для нужд резервного копирования, необходимо научиться им управлять. Подобно многим другим старомодным устройствам UNIX, способ доступа к накопителю на ленте определяет его поведение. Для каждого типа накопителей на ленте используются различные файлы устройств, и все они ведут себя по-разному. Основной механизм управления накопителем – применение файла устройства, который используется для доступа к нему.

При наличии обычного SCSI-накопителя интерес представляют только три файла устройств: */dev/esa0*, */dev/nsa0* и */dev/sa0*. Точно так же для IDE-накопителя важны файлы */dev/east0*, */dev/nast0* и */dev/ast0*.

Накопители на лентах – это устройства с последовательным доступом; данные на ленте хранятся линейно. Для получения доступа к определенным данным на ленте необходимо перематывать ленту вперед и назад. Делать это или не делать – вопрос важный.

Примечание

Поведение различных файлов устройств накопителей на ленте различно в разных операционных системах. Различные версии UNIX обладают различным программным обеспечением, которое по-разному обслуживает эти накопители. Не делайте никаких предположений относительно вашего накопителя!

Если в команде указывается файл устройства, который соответствует имени устройства, лента будет автоматически перемотана после завершения операции. Возьмем устройство SCSI sa0; при запуске команды, в которой в качестве файла устройства указан */dev/sa0*, лента будет автоматически перемотана по завершении команды.

Бывает так, что после завершения операции ленту перематывать не надо, например, когда предполагается выполнить резервное копирование данных с другой машины и разместить их на ленте или вам необходимо перед перемоткой и извлечением ленты каталогизировать ее. В этом случае следует задействовать файл устройства, имя которого начинается с *n*. Скажем, если в команде указан */dev/nsa0*, лента не будет перемотана.

Чтобы лента извлекалась автоматически после завершения операции, следует указать файл устройства, имя которого начинается с *e*. Например, если операция подразумевает резервное копирование всей системы, для автоматического извлечения ленты по окончании сохранения следует задействовать устройство */dev/esa0*. Некоторые старые накопители на лентах могут не поддерживать автоматическое извлечение ленты; в этом случае необходимо нажать клавишу для приведения в действие рычага, который извлечет ленту из накопителя. Самый простой способ, позволяющий выяснить, возможно ли автоматическое извлечение ленты, состоит в том, чтобы попробовать извлечь ее за счет использования соответствующего файла устройства и посмотреть, что произойдет.

Переменная $TAPE

Во многих программах подразумевается, что накопитель на ленте представлен файлом устройства */dev/sa0*, однако такое предположение не всегда верно. Даже если в системе установлен только один накопитель SCSI, то возможно, что вам не потребуется автоматическая перемотка ленты (*/dev/nsa0*) или надо будет извлечь ленту по завершении резервного копирования (*/dev/esa0*). Либо в системе есть накопитель IDE, для которого используется файл устройства с совершенно другим именем.

Многие программы (но не все) ориентируются на переменную окружения `$TAPE`, значение которой они используют как имя файла устройства по умолчанию. Вы всегда сможете в командной строке указать иной файл устройства, а в переменную `$TAPE` записать наиболее часто используемое значение и тем самым избавить себя от доли раздражения.

Если вы используете командный процессор по умолчанию, значение переменной `$TAPE` можно установить следующей командой:

```
# setenv TAPE /dev/sa0
```

Определение состояние накопителя с помощью команды mt(1)

Теперь, когда вы узнали, как отыскать свой накопитель на ленте, можно выполнять основные операции, такие как перемотка, натяжка, очистка и т. д., с помощью команды mt(1). Чаще всего команда mt применяется для выяснения состояния накопителя на ленте:

```
#mt status
Mode      Density              Blocksize   bpi      Compression
```

```
Current: ❶0x25:DDS-3          variable   97000    ❷DCLZ
---------available modes---------
0:       0x25:DDS-3           variable   97000    DCLZ
1:       0x25:DDS-3           variable   97000    DCLZ
2:       0x25:DDS-3           variable   97000    DCLZ
3:       0x25:DDS-3           variable   97000    DCLZ
--------------------------------
❸ Current Driver State: at rest.
--------------------------------
File Number: 0   Record Number: 0      Residual Count 0
```

Большая часть информации, представленной здесь, особого значения не имеет, однако если эти данные необходимо изучить построчно, следует обратиться к странице руководства mt(1), хорошо описывающей все особенности команды. По крайней мере, если команда возвращает какую-то полезную информацию, следовательно, она обнаруживает накопитель.

Самое первое, что мы видим, – это характеристика плотности записи на ленту ❶. В устаревших моделях для разных целей могли использоваться ленты с разной плотностью записи, но современные устройства записывают данные настолько плотно, насколько это возможно. В данном случае привод обеспечивает плотность записи DDS-3. Вам могла бы потребоваться иная плотность записи, но DDS-3 – это все, что может вам предложить данное устройство. Далее видно, что устройство предлагает возможность аппаратной компрессии данных по алгоритму DCLZ ❷. Ближе к концу видно, что делает устройство в настоящий момент ❸.

Команда `status` может выводить самые разные сообщения. Наибольшие проблемы вызывает то, которое сообщает, что устройство не сконфигурировано:

```
# mt status
mt: /dev/nsa0: Device not configured
```

Это означает, что в системе нет накопителя на ленте, соответствующего файлу устройства, который указан в переменной `$TAPE`. С файлами устройств можно поэкспериментировать, если посредством ключа `-f` указать файл устройства (например, `mt -f /dev/nsa1 status`). Однако необходимую информацию можно получить из *dmesg.boot*.

Если вы уверены в работоспособности своего устройства, то, возможно, в приводе отсутствует лента или устройство требует чистки.

Еще один вариант ответа, который можно получить от команды `mt status`, – `mt: /dev/nsa0: Device busy`. Вы спрашиваете, в каком состоянии находится ваша лента, и вам отвечают: «Я не могу говорить сейчас, я занят». Попробуйте обратиться позднее или с помощью команды `ps -ax` посмотрите, кто пользуется приводом. В каждый конкретный момент времени доступ к ленте может получить только один экземпляр программы. Вы не можете вывести содержимое ленты в то время, когда с нее производится чтение файла.

Другие команды управления накопителем на ленте

С накопителем на магнитной ленте можно выполнять гораздо более интересные операции, чем просто получать отчет о его состоянии. Наиболее часто я использую команды `retension`, `erase`, `rewind` и `offline`.

Ленты имеют свойство растягиваться, особенно первое время. (Я отлично знаю, все современные производители лент заявляют, что они предварительно производят растяжку своих лент или что их ленты не растягиваются; но прибавляем к этим заявлениям два кусочка хлеба и получаем болонский сэндвич.) *Повторное натяжение* (*retensioning*) ленты производится простой ее перемоткой вперед и назад, с помощью команды `mt retension`. Повторное натяжение устраняет слабину и повышает надежность резервного копирования.

Операция *стирания* удаляет все данные с ленты. Но это не то надежное стирание, которое можно было бы использовать для сокрытия данных от фирмы, занимающейся восстановлением данных или от налогового управления США: `mt erase` просто прокручивает ленту и накладывает новую запись поверх существующей. Это может занять довольно много времени. Если нужно стереть ленту быстро, можно воспользоваться командой `mt erase 0`, которая просто пометит ленту как пустую.

Команда `mt rewind` перемотает ленту в начало, точно так же, как и обращение к устройству через файл устройства с тем же именем.

Когда устройство *отключается*, оно перематывает ленту и выталкивает ее, что дает возможность вставить новую ленту. Команда, которая делает это, выглядит немного странно — `mt offline`.

Характер накопителя на ленте

Не все накопители на магнитной ленте поддерживают полный перечень функций. Устаревшие модели накопителей очень капризны и требуют тщательной настройки. Если вы испытываете сложности с каким-либо приводом, загляните в архивы почтовых рассылок FreeBSD-questions и посмотрите, не было ли там сообщений с описанием аналогичной проблемы. Возможно, там вам удастся отыскать ответ.

Перематывать или нет?

О лентах важно постоянно помнить, что данные на ленте хранятся линейно. Каждый участок ленты хранит отдельный блок данных. Если на одну и ту же ленту производится копирование нескольких блоков данных, то после каждой операции копирования не нужно производить перемотку ленты. Представьте, вы записали на ленту резервную копию одной системы, перемотали ленту и записали резервную копию

другой системы. Вторая резервная копия просто затрет первую, потому что запись будет произведена на тот же самый участок ленты. Когда на одну и ту же ленту записывается несколько резервных копий, используйте соответствующий файл устройства, чтобы избежать автоматической перемотки ленты между операциями.

Программы для создания резервных копий

Два популярных пакета для резервирования системы – это tar(1) и dump(8). Разумеется, есть и другие инструменты, такие как pax и cpio. Вам также могут встретиться программы для FreeBSD, выполняющие резервное копирование по сети, такие как Amanda и Bacula, и позволяющие создать резервную копию всей сети. Подобные инструменты хорошо подходят для определенных сред, но они не такие универсальные, как tar и dump. Впрочем, если освоить dump и tar, то работать с другими программами будет просто.

Программа tar(1) предназначена для работы с файлами. Данные из резервных копий, созданных с помощью tar, можно восстановить почти на всех операционных системах. Программа dump(8) работает с дисковыми разделами и файловыми системами. Информацию из резервных копий, полученных командой dump, можно восстановить исключительно на той же операционной системе, на которой были созданы копии. Для резервного копирования всех файлов следует применять dump. Если необходимо сохранить небольшой объем данных либо их потребуется восстановить в совершенно другой среде, следует предпочесть tar.

tar

Утилита *tar*, сокращение от «tape archiver» (ленточный архиватор), может создавать резервные копии любой информации – от одного файла до всей системы. В отличие от dump tar работает только с файлами и каталогами и понятия не имеет о базовой файловой системе (это его преимущество и недостаток). Tar – это общий стандарт, признанный почти всеми производителями операционных систем. Программу tar можно запускать в средах Windows, Linux, UNIX, BSD, Mac OS X, AS/400, VMS, Atari, Commodore64, QNX и почти во всех других операционных системах.

Утилита tar(1) может сохранять файлы на ленте или в специальном файле. Такой файл называется *марбол* (*tarball*). Поскольку tar работает с файлами, извлечь и восстановить файл из тарбола очень легко.

В системе FreeBSD используется своя версия tar, которая называется *bsdtar*, написанная с самого начала для замены GNU tar. Утилита bsdtar может вести себя как GNU tar, а также в строгом соответствии с POSIX tar. Если вас интересуют различия между GNU tar, POSIX tar и bsdtar, прочитайте страницу руководства tar(1), где приводится масса

кровавых подробностей. Фактически утилита bsdtar основана на библиотеке libarchive(3), которая используется разработчиками для добавления поддержки архивирования в другие программы. Один из недостатков tar – молчаливость. Если файловая система повреждена, то одному богу известно, что сохранит tar. После этого он «успешно» восстановит файлы, которые были повреждены при первоначальном копировании. Такого рода проблемы редко случаются на практике, но когда это происходит, память о них остается надолго.

Режимы tar

Утилита tar(1) может выполнять несколько операций, управляемых посредством параметров командной строки. Эти операции называются режимами. Полное описание всех режимов tar приводится в странице руководства, а здесь будут рассмотрены наиболее типичные из них.

Создание архива

Режим *создания архива* (`-c`) применяется для создания нового архива. Если иное не указано, этот ключ инициирует сохранение всей информации на ленточном устройстве (`$TAPE` или */dev/sa0*, если переменная `$TAPE` не установлена). Для сохранения всей системы надо сообщить утилите tar о необходимости рекурсивно архивировать все файлы, начиная с корневого каталога:

```
# tar -c /
```

В ответ на ленточном накопителе должен загореться индикатор. Если на ленте достаточно места, в конечном итоге на ней будет сохранена вся система. Многие современные диски обладают значительно большей емкостью по сравнению с лентами, поэтому часто имеет смысл сохранять лишь отдельные, жизненно важные части системы. Так, если все файлы, которые подвергались изменениям, находятся в разделах */home* и */var*, то именно эти каталоги можно указать в командной строке:

```
# tar -c /home /var
```

Вывод содержимого архива

В режиме *вывода содержимого* (`-t`) выводятся имена всех файлов, находящихся в архиве. После завершения работы накопителя можно применить этот ключ, чтобы отобразить содержимое ленты:

```
# tar -t
.
.snap
dev
tmp
...
```

Эта операция выводит имена всех файлов в архиве и может занять продолжительное время. Примечательно, что в именах файлов отсутствует

первый символ слэша («/»). Например, имя файла */tmp* выводится как *tmp*. Эта особенность играет важную роль при восстановлении файлов.

Извлечение файлов из архива

В режиме *извлечения файлов* (`-x`) tar извлекает файлы из архива и копирует их на диск. Извлечение файлов производится в текущий каталог; чтобы перезаписать существующий системный каталог */etc*, надо прежде всего перейти в корневой каталог. С другой стороны, чтобы восстановить копию каталога */etc* в моем домашнем каталоге, я должен был бы сначала перейти в свой домашний каталог.

```
# cd /home/mwlucas
# tar -x etc
```

Помните замечание о том, что отсутствие первого символа слэша («/») играет важную роль? Теперь ясно, почему. Если бы в именах архивируемых файлов первым символом был «/», tar всегда извлекал бы файлы относительно корневого каталога. Восстановление файла */etc/rc.conf* из резервной копии всегда приводило бы к затиранию существующего файла */etc/rc.conf*. При отсутствии начального символа слэша («/») в резервной копии извлечение файлов может производиться в любой каталог, например, файл */etc/rc.conf* можно восстановить в виде файла */home/mwlucas/etc/rc.conf*. Если файлы извлекаются не на том компьютере, где они были заархивированы, было бы крайне нежелательно, чтобы они затерли существующие файлы на текущем компьютере. Желательно извлекаемые файлы сохранять в отдельном каталоге, чтобы они не оказывали влияния на текущую систему.

Проверка архива

Создав резервную копию, ее можно проверить, чтобы убедиться, что она соответствует вашей системе. Режим *поиска различий* (`-d`) сравнивает файлы на ленте с файлами на диске. Если все файлы на ленте соответствуют файлам в системе, то команда `tar -d` будет выполнена «молча». Однако абсолютное соответствие будет *удивительно*. Обычно при проведении резервного копирования растут файлы протоколов, поэтому они не будут совпадать с сохраненными файлами. Или, если в системе есть активная база данных, ее файлы могут не соответствовать сохраненным. Если вам необходимо полное соответствие резервной копии (которая еще называется *холодная копия*), вам следует загрузиться в однопользовательском режиме и создавать резервную копию из него. Вам следует решить, какие ошибки сравнения можно игнорировать, а с какими следует разбираться.

Другие возможности tar

Утилита tar обладает некоторыми другими особенностями, которые делают ее более дружественной и удобной. В число этих особенностей входят подробный режим отображения хода выполнения операции,

различные способы сжатия, восстановление прав доступа к файлам и самая популярная особенность – запись архивов в файлы вместо устройства накопителя на магнитных лентах.

Использование файла вместо ленты

Ключ `-f` позволяет задать устройство или файл в качестве места для создания архива. Во всех предыдущих примерах использовалось устройство по умолчанию */dev/sa0* или устройство, определяемое переменной `$TAPE`. В случае необходимости накопитель на ленте можно указать с помощью ключа `-f`:

```
# tar -c -f /dev/east0 /
```

Вместо записи резервной копии на ленту можно создать tar-файл. Исходный код, распространяемый Интернете, часто распространяется в виде файлов с расширением .tar (tarballs). Для сохранения резервной копии в файле можно использовать ключ `-f`. Так, для сохранения глав этой книги я время от времени создавал тарбол *bookbackup.tar*:

```
# tar -cf bookbackup.tar /home/mwlucas/absolutefreebsd/
```

Этот файл легко можно сохранить на любой другой машине, поэтому, даже если мой дом сгорит, книга будет сохранена. Далее можно задействовать телефонный канал и линию электропередачи соседа, позаимствовать ноутбук, отыскать открытую точку беспроводного доступа к сети, запустить `tar -xf bookbackup.tar` и работать в окружении головешек, ожидая представителя страховой компании. (Вряд ли я смог бы сделать намного больше в подобной ситуации.)

Подробный режим

Ключ `-v` делает утилиту tar словоохотливой. Обычно tar работает «молча», за исключением случаев, когда появляются ошибки. Чаще всего это очень хорошо (кому захочется читать длинные списки файлов на сервере всякий раз, когда запускается процедура архивирования?), но иногда возникает желание ощутить то теплое чувство, которое дает вид работающей программы. С ключом `-v` утилита покажет имена всех файлов, которые упаковываются в архив или распаковываются из архива. При рутинном архивировании или разархивировании подробный режим мешает следить за появлением ошибок.

gzip

Ключ `-z` вызывает обработку файлов программой сжатия gzip(1) – как при архивировании, так и при разархивировании. Сжатые тарболы обычно имеют расширение *.tar.gz* или *.tgz*, редко *.taz*. Сжатие файлов значительно уменьшает размер архива; на самом деле многие резервные копии можно сжать на 50% и более. Все современные версии tar, в отличие от старых, поддерживают gzip. Поэтому, если необходимо,

чтобы сжатые файлы мог прочесть абсолютно каждый, применять ключ -z не надо.

Сжатие

В противоположность предыдущему ключу все версии tar на всех версиях UNIX могут сжимать файлы с помощью ключа -Z, который вызывает compress(1). Программа compress не так эффективна, как gzip, но она все-таки уменьшает размер файла. Тарболы, сжатые с помощью -Z, имеют расширение *.tar.Z*.

Сжатие утилитой bzip

С помощью ключа y tar в составе FreeBSD поддерживает сжатие с помощью программы bzip, более эффективной, чем gzip. bzip требует больше времени процессора, чем gzip, но в наше время процессорное время не так ограничено, как во времена, кода появилась утилита gzip. Однако не все версии поддерживают bzip. Если сохраняемые файлы будут читаться только на системе FreeBSD или вы в состоянии будете установить bzip на другой платформе, используйте ключ -y.

Сжатие и утилита tar в системе FreeBSD

Библиотека libarchive в системе FreeBSD автоматически определяет тип сжатия, используемый для создания резервных копий. При создании архива вы должны явно указать желаемый тип сжатия, однако при извлечении файлов из архива можно позволить утилите tar(1) самой определять тип сжатия и Принять Правильное Решение самостоятельно.

Восстановление прав доступа к файлам

Ключ -p позволяет восстанавливать первоначальные права доступа к извлекаемым файлам. По умолчанию tar назначает владельцем извлекаемых файлов пользователя, который производит операцию разархивирования. Такой подход оправдывает себя при работе с исходными текстами программ, но при восстановлении системы необходимо восстанавливать первоначальные права доступа к файлам. (Попробуйте на протяжении некоторого времени восстанавливать эти права вручную; вы сразу поймете, почему сразу нужно все делать правильно.)

И еще, и еще, и еще...

Утилита tar имеет еще массу других функций, учитывающих изменения в резервном копировании, файлах, файловых системах и дисках, происходившие в течение десятилетий. Полный список этих функций вы найдете в странице руководства tar(1).

dump

dump(8) – инструмент для проведения резервного копирования, работающий на уровне дисковых блоков. В какой-то мере dump похож на tar(1). Существенное различие между ними состоит в том, что программа dump учитывает тип файловой системы и использует ее преимущества. Более подробно файловые системы будут обсуждаться в главе 8. Сейчас достаточно знать, что файловая система определяет порядок, в котором единицы и нули располагаются на физическом жестком диске. Утилита dump, в частности, совместима с файловой системой UFS2, используемой во FreeBSD. Скорее всего, начинающие системные администраторы хуже знакомы с dump, чем с tar, но dump эффективнее и безопаснее. Когда есть выбор, надо применять dump.[1]

Единственный недостаток dump в том, что эта утилита работает с файловыми системами, а не с отдельными файлами. Поэтому dump не подходит для сохранения каталога */etc*, если не надо сохранять весь корневой раздел. Тем не менее есть возможность восстанавливать файлы по отдельности.

Положительная особенность dump: для сохранения и восстановления информации dump применяет различные программы (dump(8) и restore(8) соответственно). Это означает, что ключи уже нельзя перепутать: при восстановлении данных файл в системе не запишется случайно поверх файла в архиве. Кроме того, dump работает значительно быстрее, чем tar.

Управление работой dump

Самое большое преимущество dump(8) в том, что она предоставляет пользователям возможность управлять своей работой. Например, они могут указать, какие файлы не надо включать в дамп, и они не будут копироваться. У многих пользователей есть файлы, не представляющие для них ценности. Они с радостью согласятся не сохранять эти файлы при условии, что важные для них файлы будут сохранены.

Для установки флага `nodump` воспользуйтесь chflags(1):

```
# chflags nodump filename
```

Содержимое каталога, к которому применен флаг `nodump`, и вложенных в него подкаталогов сохраняться не будет. Например, я использую ко-

[1] Некоторые системные администраторы не согласны с этим. Они настаивают, что tar(1) лучше. Такое расхождение во мнениях характерно для всего сообщества UNIX. Любые рекомендации, которые я дам, несомненно, вызовут гнев у администраторов, отдавших свои симпатии другому инструменту. Я твердо полагаю, что единственный способ уладить споры между людьми, фанатично преданными тому или иному инструменту, – это встретиться в поле на рассвете и разрешить споры с оружием в руках. Остальные будут просто продолжать жить.

манду chflags, чтобы избежать копирования каталога с дистрибутивами, загруженными из Интернета, и сэкономить время и место, потому что эти файлы всегда можно загрузить заново.

Уровни резервирования

Одна из наиболее интересных особенностей dump – возможность инкрементного копирования через *уровни резервирования* (*dump level*), которые варьируются от 0 до 9. По умолчанию принимается уровень 0, что подразумевает копирование всех файлов на диске, для которых не установлен флаг `nodump`. Более высокие уровни резервирования означают следующее: «сохранять файлы, которые были изменены или созданы с момента выполнения дампа более низкого уровня». Такая последовательность уровней позволяет выполнять инкрементные дампы – достаточно указать необходимый уровень резервирования как ключ командной строки.

Например, пусть каждый понедельник выполняется дамп уровня 0. Во вторник можно выполнить инкрементный дамп уровня 1. При этом будут сохранены только те файлы, которые изменились с момента проведения дампа в понедельник. Если в среду выполнить дамп уровня 2, то будут сохранено все, что изменилось со вторника. Однако если в следующий вторник выполнить еще один дамп уровня 1, будет сохранено все, что изменилось с первого понедельника, включая файлы, которые были сохранены в среду.

Хотя с помощью dump можно выполнять инкрементное резервирование, лучше создавать дампы уровня 0 – просто потому, что тогда будет намного легче восстанавливать файлы, чем в случае серии инкрементных дампов. Дампы уровня 0 выполняются дольше инкрементных дампов и требуют больше места, однако в большинстве случаев время, сбереженное при восстановлении данных, важнее, чем стоимость ленты. При надлежащем планировании дампы уровня 0 можно выполнять ночью.

Указывайте желаемый уровень резервирования как аргумент командной строки, например, чтобы создать дамп уровня 2, запустите команду `dump -2`.

dump, ленточные накопители и файлы

К сожалению, dump(8) не распознает переменную окружения `$TAPE` и склонен по умолчанию использовать файл устройства */dev/sa0*. Однако с помощью ключа `-f` можно указать другой накопитель. Как и tar, утилита dump позволяет с помощью ключа `-f` указать файл, в котором будет сохранена резервная копия. Хотя файлы, создаваемые dump, не так пригодны для распространения, как файлы, создаваемые tar, тем не менее, это прекрасная возможность поэкспериментировать и поближе познакомиться с dump.

Прежде чем dump приступит к созданию резервной копии, она пытается определить объем доступного пространства на ленте. К сожалению, эта ее возможность не усовершенствовалась уже много лет. Когда dump только появилась, производство приводов с лентами объемом 1 Мбайт было серьезным бизнесом, и каждый производитель выпускал ленты своих форматов, следуя своим собственным стандартам. В наше время накопители стали более универсальными и стандартными, и производители обеспечивают более широкую совместимость. Емкость лент также существенно выросла, например, работая над этой книгой, я использовал привод емкостью 40 Гбайт, от которого, вполне исправного, отказался предыдущий владелец только потому, что емкость его оказалась слишком маленькой. Вот так, с улучшением стандартизации и существенным ростом емкости, dump оказалась не в состоянии определять объем доступного пространства. Чтобы обойти эту проблему, лучше всего сообщить утилите о том, что она не должна определять размер ленты, а должна выполнять дамп до достижения физического маркера конца ленты и затем запросить смену ленты. Для этой цели используется флаг `-a`.

dump и задействованные файловые системы

Одна из проблем, которая возникает при создании резервных копий работающей системы, состоит в том, что файловая система постоянно изменяется в процессе копирования. Эта проблема отсутствует в файловых системах, предназначенных для хранения статичных данных, или когда изменения в одной части файловой системы не вызывают изменения в другой ее части. Но в случае хранения динамических, постоянно изменяющихся и/или взаимосвязанных данных, это превращается в серьезную проблему. Данная проблема характерна для большинства баз данных. Иногда бывает нежелательно останавливать базу данных только для того, чтобы получить качественную резервную копию, а иногда даже невозможно создать дамп базы данных в виде файла, поэтому приходится прибегать к холодному копированию. Чтобы обойти эту проблему, dump способна использовать возможность файловой системы UFS2 по созданию «снимков» и тем самым обеспечить внутреннюю целостность резервной копии. Подробнее о снимках файловой системы будет рассказываться в главе 8, а пока достаточно просто запомнить, что снимок – это образ диска, созданный в определенный момент времени. Даже когда данные на диске все время изменяются, снимок остается неизменным и статичным, поэтому его легко можно скопировать.

Чтобы создать дамп снимка, укажите ключ `-L`. Если производить копирование задействованной файловой системы UFS2 без этого ключа, dump сообщит об этом и предложит использовать ключ `-L`.

Разумеется, это не устраняет проблем с базами данных – нахождение файловой системы в непротиворечивом состоянии не означает, что база данных, которая расположена в этой файловой системе, также находит-

ся в непротиворечивом состоянии. Однако можно остановить базу данных на короткое время, запустить dump и снова запустить базу данных, позволив тем самым утилите dump записать снимок в тот момент, когда база данных была остановлена. Это уменьшает время бездействия, необходимое для создания резервной копии, до одной или двух секунд.

Временные метки и dump

В файл */etc/dumpdates* записывается все, что было сохранено в системе. Это наиболее полезно при инкрементном резервировании, когда для успешного восстановления системы необходимо знать дату последнего полного сохранения. Задав ключ `-u`, этот файл можно обновить.

Запуск dump

Теперь, объединив все полученные сведения, можно выполнить резервное копирование файловой системы. Здесь я задаю утилите dump: не вычислять количество необходимых лент, создавать резервную копию из снимка задействованной файловой системы и произвести копирование раздела */usr* на уровне резервирования 0.

```
# dump -auL /usr
❶ DUMP: Date of this level 0 dump: Sat Apr 5 08:26:03 2008
❷ DUMP: Date of last level 0 dump: the epoch
❸ DUMP: Dumping snapshot of /dev/ad0s1f (/usr) to /dev/sa0
❹ DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
❺ DUMP: estimated 3944900 tape blocks.
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
❻ DUMP: 11.54% done, finished in 0:38 at Sat Apr 5 09:09:25 2008
  DUMP: 28.12% done, finished in 0:25 at Sat Apr 5 09:01:40 2008
  DUMP: 40.69% done, finished in 0:21 at Sat Apr 5 09:02:58 2008
  DUMP: 57.26% done, finished in 0:14 at Sat Apr 5 09:01:02 2008
  DUMP: 72.60% done, finished in 0:09 at Sat Apr 5 09:00:33 2008
  DUMP: 87.49% done, finished in 0:04 at Sat Apr 5 09:00:24 2008
  DUMP: 99.99% done, finished soon
  DUMP: DUMP: 4095026 tape blocks on 1 volume
❼ DUMP: finished in 2130 seconds, throughput 1922 KBytes/sec
❽ DUMP: level 0 dump on Sat Apr 5 08:26:03 2008
  DUMP: Closing /dev/sa0
❾ DUMP: DUMP IS DONE
```

В этом выводе присутствует довольно много важных сведений. Во-первых, dump выводит текущую дату ❶ и дату последней операции резервного копирования. Дата, что показана здесь, `the epoch`, – это лишь необычный способ сказать: «от начала времен». Как известно, эпоха UNIX началась в 1970 году, что не так давно, как можно было бы подумать. В действительности же это означает, что в файле */etc/dumpdates* отсутствует какая-либо информация о резервном копировании этого

раздела раньше, но это совсем не означает, что резервирование *никогда* не производилось.

Прежде чем приступить к работе, dump напоминает, какой раздел будет копироваться и какой накопитель на магнитной ленте будет использоваться ❸.

Затем dump выполняет предварительный анализ указанного раздела, выясняет структуру файлов и каталогов и прикидывает, какой объем пространства на ленте потребуется ❹. Выполнив необходимые вычисления, dump выводит результаты ❺ и приступает к резервному копированию файлов ❻. Через каждые несколько минут dump сообщает, как далеко удалось продвинуться и сколько еще времени потребуется для полного завершения операции, благодаря чему у вас не будет появляться ощущение, что компьютер завис. Обратите внимание, как изменяется процент выполнения и немного варьируется примерное время завершения – суть в том, что любые программные таймеры, сообщающие пользователю примерное время, оставшееся до окончания операции, никогда не обладали и, наверное, никогда не будут обладать высокой точностью, независимо от используемой операционной системы.

Завершив операцию, dump сообщит объем заархивированных данных и среднюю скорость работы ❼, а затем еще раз выведет дату ❽. Это не дата окончания операции – это дата создания резервной копии. Операция завершилась в 9 часов утра, но резервное копирование производилось со снимка файловой системы, который был сделан в 8:26.

В заключение еще раз сообщается, что операция закончена ❾, и теперь можно вынимать ленту из привода.

Пропуск данных, помеченных флагом nodump

С помощью ключа `-h` системный администратор может указать, когда следует принимать во внимание флаг `nodump`. Этот ключ принимает уровень резервирования в качестве аргумента.

По умолчанию на уровне 0 dump архивирует все файлы, независимо от наличия флага `nodump`. На уровне 1 или выше dump учитывает флаг `nodump`. Ключ `-h` изменяет поведение утилиты по умолчанию и указывает минимальный уровень резервирования, начиная с которого следует учитывать флаг `nodump`. На любых других уровнях резервирования ниже указанного копироваться будут все файлы, независимо от наличия флага `nodump`.

Это дает возможность регулировать объем архивирования на уровне 0. Если резервная копия перестала умещаться на ленту, воспользуйтесь ключом `-h`, чтобы предотвратить резервирование файлов, отмеченных флагом `nodump`. Это даст вам передышку на пару дней, в течение которых вы сможете заказать новые ленты.

Восстановление из архива

Архивы – это замечательно, но они бесполезны, если с их помощью нельзя восстановить систему. Утилита для восстановления данных, restore(8), позволяет извлекать либо целые файловые системы, либо отдельные файлы. Как и в случае с tar и dump, ключ `-f` позволяет выбрать устройство или файл, из которого будут восстановлены данные.

Проверка содержимого архива

Для вывода списка файлов, находящихся в архиве, следует указать ключ `-t`.

```
# restore -t
❶ Dump date: Sat Apr 5 08:26:03 2008
Dumped from: the epoch
❷ Level 0 dump of /usr on test1.blackhelicopters.org:/dev/ad0s1f
Label: none
❸2 ❹.
3 ./.snap
94208 ./bin
97995 ❺./bin/bc
...
```

Утилита restore позволяет узнать, когда была сделана резервная копия ❶, что было скопировано ❷ и на каком уровне резервирования. После этого она выводит имена всех файлов в архиве и их местоположение в файловой системе. Для каждого файла выводится номер индексного дескриптора (inode) ❸. (Об индексных дескрипторах мы поговорим в главе 8.)

Следует заметить, что пути к файлам указаны относительно их местоположения в оригинальной файловой системе. Мы создали резервную копию корневой файловой системы, которая здесь представлена символом точки ❹. В действительности это корневой каталог файловой системы */usr*, или просто каталог */usr*. Полное имя файла *./bin/bc* ❺ в оригинальной файловой системе на самом деле было не */bin/bc*, а */usr/bin/bc*.

Об этом необходимо постоянно помнить при поиске конкретных файлов в архивах. Для проверки присутствия некоторого файла в архиве можно использовать ключ `-t`. Предположим, что необходимо восстановить файл */usr/home/mwlucas/.cshrc*. Первое, что приходит в голову, – это проверить наличие файла в архиве:

```
# restore -t /usr/home/mwlucas/.cshrc
...
./usr/home/mwlucas/.cshrc is not on the tape
```

Как так? Файл отсутствует на ленте? Где мои данные? Пора кричать «караул»? Конечно нет, подождите паниковать. Вспомните, этот архив понятия не имеет о существовании каталога */usr* – все сохраненные

пути файлов записаны относительно каталога */usr*. Нужно искать файл *home/mwlucas/.cshrc*.

```
# restore -t home/mwlucas/.cshrc
...
    871426 ./home/mwlucas/.cshrc
```

Уф-ф! Файл *.cshrc* присутствует в архиве. Попробуем теперь извлечь его.

Извлечение данных из архива

Как только стало известно, что тот или иной файл есть в архиве, его можно получить двумя способами: или извлекать файл за файлом, или восстановить всю файловую систему.

Восстановление файла

Если надо извлечь лишь некоторые данные, следует задать ключ `-x` и имя файла. В этом случае будет извлечен только указанный файл. Например, для восстановления *.cshrc* из архива, записанного на ленту, надо набрать следующую команду:

```
# restore -x home/mwlucas/.cshrc
You have not read any tapes yet.
If you are extracting just a few files, start with the last volume
and work towards the first; restore can quickly skip tapes that
have no further files to extract. Otherwise, begin with volume 1.
(Пока вы не прочитали ни одну ленту. Если вы не знаете, на каком томе
находятся файлы, необходимо начать с последнего тома и двигаться к первому;
restore может быстро пропускать ленты, где нет файлов, предназначенных для
извлечения. В противном случае начните с первого тома)
❶ Specify next volume #: 1
(Укажите следующий том)
❷ set owner/mode for '.'? [yn] y
(установить владельца/права доступа для '.' ?)
```

В первую очередь restore просит ввести номер тома ❶. Это порядковый номер ленты, из тех, что были использованы для создания данной резервной копии. Если архив записывался на несколько лент, dump(8) должна была сообщить вам номера лент, которые менялись в процессе копирования. (Надеюсь, вы правильно промаркировали ленты?) Если копия уместилась на одну ленту, то это будет том с номером 1.

Как только утилита restore отыщет файл, она предложит сохранить его с оригинальными правами доступа и признаком принадлежности файла ❷. В данном примере я хотел остаться владельцем файла, поэтому в ответ на вопрос я ввел `y`. После выполнения операции в текущем каталоге появится каталог *home/mwlucas* с файлом *.cshrc* внутри.

Восстановление файловой системы

Восстановить всю файловую систему достаточно просто, надо лишь помнить, что лучше не восстанавливать файловую систему поверх су-

ществующей. Если требуется серьезное восстановление, то безопаснее удалить раздел и начать все заново. Если необходимо сохранить некоторые файлы из поврежденной файловой системы, заархивируйте эти файлы в отдельный архив, отформатируйте раздел, восстановите файловую систему из резервной копии, а затем скопируйте отобранные ранее файлы на место.

В следующем примере будет удален раздел на втором жестком диске, а затем данные будут восстановлены с резервной ленты. Работа с диском здесь не будет подробно рассматриваться (дополнительную информацию см. в главе 8), однако все действия в общем виде можно представить так:

1. Создание новой файловой системы с помощью `newfs`.
2. Присоединение этой файловой системы к дереву каталогов, в `/mnt`.
3. Переход в этот каталог.
4. Восстановление файловой системы с ленточного устройства по умолчанию */dev/sa0*.

Вот эти команды:

```
# newfs /dev/ad1s1g
# mount /dev/ad1s1g /mnt
# cd /mnt
# restore -r
```

Все настолько просто, что у вас может даже появиться желание уничтожить содержимое файловой системы только ради того, чтобы восстановить ее, не так ли?

Восстановление и последующее резервирование

Когда бы ни проводилось полное восстановление диска, перед следующим после него инкрементным дампом необходимо выполнить дамп уровня 0. restore(8) изменяет положение данных на диске. Если попытаться создать инкрементный дамп вновь восстановленной файловой системы и попытаться использовать его вместе с прежним дампом уровня 0, это приведет к повреждению данных. Всегда создавайте дамп уровня 0 сразу после восстановления файловой системы, чтобы обеспечить корректность последующих инкрементных дампов. Кроме того, нужно помнить, что при выполнении полного резервирования жить будет намного легче.

Интерактивное восстановление

Одна из наиболее интересных особенностей restore(8) – интерактивный режим (`-i`), позволяющий «вскрыть» дамп и получить доступ к нему

с помощью инструмента командной строки. При этом можно указать файлы, которые необходимо восстановить. Интерактивный режим бывает весьма кстати, когда пользователь говорит примерно следующее: «Я случайно удалил файл с резюме. Он лежал в моем домашнем каталоге, а в его имени было слово *resume*. Точного имени не помню. Можно его восстановить?» Ясно, что ключ `-t` не поможет; точное название файла неизвестно! Вместо этого можно перейти в интерактивный режим и поискать этот файл. Это совсем несложно, а пользователь останется вам обязан.[1] Запустите restore с ключом `-i`, и вы получите доступ к интерактивному сеансу с командной строкой, которая ведет себя почти так же, как обычная командная строка UNIX, но поддерживает только команды, необходимые для восстановления. В зависимости от типа ленточного накопителя может потребоваться одна-две минуты, прежде чем появится командная строка.

```
# restore -i
restore > ls
.:
.snap/   bin/     games/  include/ libdata/ local/  ports/  share/
X11R6/   compat/  home/   lib/     libexec/ obj/    sbin/   src/
restore > cd home/mwlucas
```

Здесь нет ничего необычного – это содержимое каталога верхнего уровня в файле дампа, в данном случае – каталога */usr*. По нему можно продвигаться, используя команду `cd`, и выводить списки файлов с помощью команды `ls`, как в обычной командной оболочке. Найдя необходимый файл, его можно восстановить, для этого нужно добавить его в список извлекаемых файлов с помощью команды `add`. Когда все необходимые файлы будут добавлены в список, можно запустить процедуру восстановления с помощью команды `extract`.

```
restore > add ssh.tar
restore > add .cshrc
restore > extract
You have not read any tapes yet.
(Ни одна лента еще не была прочитана)
...
```

Последующий процесс выглядит точно так же, как и восстановление в неинтерактивном режиме, – restore предложит указать номер тома и спросит, необходимо ли восстанавливать права доступа к файлам. По завершении выбранные файлы появятся в текущем каталоге.

Завершив восстановление файлов, можно завершить интерактивный сеанс работы с помощью команды `quit`.

---

[1] Оставшись в долгу, пользователь будет стремиться не совершать подобных ошибок, потому что это даст вам возможность не только оплачивать уход за своей лужайкой. Секрет успеха системного администратора кроется в его профессионализме, в чувстве юмора и в бейсбольной бите.

Создание нескольких резервных копий на одной ленте

Если ленточный накопитель обладает достаточно большой емкостью, на одной ленте можно сохранять несколько резервных копий. dump(8) позволяет за один раз копировать только один раздел, и все же держать отдельные ленты для хранения копий каждого раздела весьма неэффективно. С помощью утилиты tar(1) можно сохранять несколько резервных копий на одну ленту, поскольку в этом случае можно с помощью команды mt(1) управлять перемоткой ленты.

Вспомните, использование файла устройства по умолчанию в утилитах mt, tar и dump приводит к автоматической перемотке ленты после каждой операции. Это означает, что вторая резервная копия затрет первую. Изменив используемый файл устройства, можно изменить поведение накопителя. Если запускать резервное копирование без перемотки, то вторая резервная копия запишется на лету как второй файл. Управляя текущей позицией ленты, можно выбирать, где производить запись или чтение данных.

Например, следующие команды запишут резервные копии трех файловых систем – /, */var* и */tmp* – на одну и ту же ленту:

```
# dump -f /dev/nsa0 -auL /
# dump -f /dev/nsa0 -auL /var
# dump -f /dev/nsa0 -auL /tmp
```

Теперь текущая позиция на ленте соответствует концу третьего файла. Ленту можно перемотать и извлечь из накопителя, надписать на ней имена содержащихся файлов, дату копирования и спрятать в надежное место.

Если необходимо определить количество резервных копий на ленте, достаточно запустить команду `mt status` и посмотреть на последнюю строку.

```
...
File Number: 3 Record Number: 0 Residual Count 0
```

Обратите внимание, что номер файла стал равен 3. То есть на ленте записано три файла.

Чтобы получить доступ к файлу, необходимо выполнить позиционирование ленты в накопителе на начало файла и с помощью tar или restore прочитать данные с ленты. После перемотки ленты текущая позиция соответствует началу первого файла. Чтобы переместиться к одному из следующих файлов, следует использовать команду `fsf` утилиты mt(1). Команда `mt fsf` принимает один аргумент – количество файлов, которые необходимо пропустить. Если предположить, что изначально считывающая головка накопителя находится в начале первого файла, тогда, чтобы переместиться ко второму файлу, достаточно выполнить команду:

```
# mt -f /dev/nsa0 fsf 1
```

В результате лента будет перемотана на один файл вперед. Обратите внимание на ключ `-f /dev/nsa0`, благодаря которому используется файл устройства, который «не перематывает» ленту. Было бы бессмысленно сначала выполнить переход на один файл вперед, а затем автоматически перемотать ленту.

Теперь с помощью ключа `-t` программы извлечения файлов из архива можно ознакомиться с содержимым этого файла. Резервную копию легко можно идентифицировать по ее содержимому, поскольку при использовании ключа `-t` restore(8) выводит даже дату и время копирования, а также имя раздела, с которого была сделана копия. Для перемещения по ленте в обратном направлении используется команда `mt bsf`, которой передается число файлов, которые нужно пропустить.

Ленты требуют немного иного мышления, чем диски, но в действительности это совсем несложно и к тому же ленты по-прежнему широко используются для резервного копирования. Теперь рассмотрим некоторые другие методы защиты вас, ваших данных, вашей системы и того, что мы, системные администраторы, со смехом называем нашими мозгами.

Управление версиями

Вообще говоря, управление версиями (revision control) – это процесс отслеживания изменений. В мире UNIX под этим понимаются изменения в исходном коде и в файлах конфигурации. Управление версиями позволяет разработчику видеть, как некий фрагмент кода выглядел в определенный момент. Администратор может представить, как была сконфигурирована система до возникновения неполадок. Управление версиями может быть полезно даже скромному писателю – он может увидеть, как менялась рукопись с течением времени. Если не применять этот механизм, работа будет труднее, чем ей положено быть.

Несмотря на то, что существует много систем управления версиями, от системы управления исходным кодом SCCS (Source Code Control System) в UNIX до Visual SourceSafe компании Microsoft, здесь будет рассмотрена система управления версиями RCS (Revision Control System), входящая почти во все системы UNIX. После изучения RCS работать с большинством других систем управления версиями будет просто.

При управлении версиями, по существу, сохраняется информация обо всех изменениях, сделанных в файле, а также описание причин, вызвавших эти изменения. Во-первых, файл помечается как захваченный (checked out). Это знак системе, что файл будет изменяться. Далее файл можно отредактировать, затем записать изменения в системе, а затем файл можно *передать* (check in) другим пользователям для последующего редактирования. Для поддержки этих действий RCS применяет три команды: ci(1) (check in, регистрация), co(1) (check out, захват) и rcs(1).

Передовые методы управления версиями

В моей многолетней практике системного администратора я встречал множество методов управления версиями. Пожалуй, самый распространенный способ состоит в сохранении копий всех файлов с датой сохранения в виде расширения (например, *rc.conf.20060510*). Однако такую методику никак нельзя назвать хорошей, более того – это плохая практика. Нет никакой гарантии, что при таком подходе будут сохраняться все изменения и отсутствует система слежения, с помощью которой можно было бы оставить комментарий, описывающий причины, вызвавшие эти изменения. Как сказал один мой товарищ: «RCS – это как горькое лекарство, которое необходимо пить, иначе вы не пойдете на поправку». Чтобы овладеть навыками работы с RCS, необходимо принуждать себя пользоваться ею, и через несколько месяцев вы будете удивляться – как вы вообще без нее обходились.

Управление версиями напоминает библиотеку – нет, не электронную библиотеку, а обычную, старую добрую библиотеку с книгами, отпечатанными на бумаге. Чтобы задействовать систему управления версиями, прежде всего надо указать системе RCS на необходимость отслеживания файла – поместить его в библиотеку. Чтобы приступить к работе с этим файлом, его необходимо получить, как книгу в библиотеке. Если файл выдан, то никто иной не может его сохранять или редактировать, хотя любой законный пользователь в то же время может просматривать, копировать, компилировать этот файл – обращаться к нему. После завершения работы с файлом он сдается обратно (check in) и становится доступен для редактирования другими пользователями. Такой порядок действий составляет основу любой системы управления версиями, а про файлы, находящиеся под контролем системы управлениями версиями, говорят, что они находятся *в RCS*.

В RCS каждый файл имеет номер версии. Всякий раз, когда отредактированный файл возвращается в систему, RCS сравнивает возвращаемый файл с выданным. При наличии хоть каких-то изменений эти изменения сохраняются, а номер версии увеличивается на единицу, вместе с датой и причиной изменений. Номер версии можно применять для идентификации отдельных версий файла.

Инициализация управления версиями

Процесс управления версиями начинается с регистрации файла с помощью ci(1), что напоминает сдачу книги в библиотеку. Например, хороший кандидат для защиты с помощью RCS – файл */etc/rc.conf*. Для начала надо ввести `ci` *filename*, как показано в следующем листинге:

```
# ci rc.conf
❶ rc.conf,v <-- rc.conf
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> ❷system configuration file
>> ❸.
❹ initial revision: 1.1
done
```

При первой регистрации файла с помощью ci эта команда создает или редактирует файл управления версиями. Этот файл получает то же имя, что и оригинальный файл, но с расширением *,v*. В данном примере файл *rc.conf* превращается в *rc.conf,v* ❶. Далее система запрашивает описание файла, и его надо ввести, чтобы позднее каждый пользователь RCS мог понять, что это за файл. Это описание не так уж важно для стандартных системных файлов, таких как *rc.conf*, – для него вполне достаточно краткого описания `system configuration file`, как задано в этом примере ❷. Более подробное описание будет очень полезно для исходного кода или файлов конфигурации самостоятельно разработанных или сложных программ. После того как описание набрано, для завершения работы ci следует ввести точку на отдельной строке ❸. После этого будет выведен номер версии файла ❹, который в первый раз всегда равен 1.1.

Если сразу после регистрации файла запустить `ls`, то можно увидеть, что файл как будто исчез. Вместо этого появится файл с тем же именем, но с замыкающими символами *,v*. Это файл RCS, в котором хранятся сам файл и его изменения. Если в RCS хранится много файлов, то файлы с расширением *,v* могут вносить некоторый хаос. Чтобы этого не происходило, достаточно создать каталог с именем RCS (все символы заглавные), и тогда программа ci будет сохранять файлы *,v* в этот каталог, не засоряя рабочий каталог.

Для некоторых файлов не важно, что они «исчезают» при передаче, но файлы с настройками или веб-страницами не могут просто исчезнуть. Чтобы решить эту проблему, при регистрации файла нужно оставить его копию в рабочем каталоге. Для этого надо запустить `ci -u`. Если файл исчез, а его копию необходимо поместить в рабочий каталог без редактирования файла, надо выполнить команду co(1).

```
# co rc.conf
❶ rc.conf,v --> rc.conf
❷ revision 1.1
done
```

Так как компьютер не в состоянии выполнить корректную загрузку без файла *rc.conf*, очень важно обеспечить доступность этого файла. Система RCS извлекла файл *rc.conf* из *rc.conf,v* ❶, и теперь версия 1.1 ❷ этого файла доступна для использования. Если внимательнее приглядеться к этим файлам, можно увидеть нечто удивительное.

```
# ls -l rc.conf*
-r--r--r--   1 root  wheel  321 Apr 10 21:40 rc.conf
-r--r--r--   1 root  wheel  527 Apr 10 21:33 rc.conf,v
```

Владелец этого файла – пользователь root, однако права доступа – «только для чтения» (`-r--r--r--`). Даже при том, что я знаю пароль пользователя root, я не имею права редактировать свои файлы! Дело в том, что файл не был захвачен *мной*. Файл был зарегистрирован и передан библиотекарю Revision Control System. Я могу просматривать этот файл, но для его редактирования нужно попросить разрешения у системы RCS.

Предупреждение для тех, кто использует редактор vi(1)

Предупреждение для пользователей vi: если вы или ваша группа владеете файлом, команда `w!` вызовет изменение прав доступа и позволит вам осуществлять запись в файл даже без его регистрации. Все будет выглядеть замечательно, однако следующий пользователь, который зарегистрирует файл, перезапишет ваши изменения! Будьте осторожны с этой командой; если vi говорит, что у вас нет прав на сохранение файла, это весомое предупреждение, к которому лучше прислушаться. Вы можете проигнорировать это предупреждение на свой страх и риск.

Редактирование файлов, находящихся в RCS

Чтобы отредактировать файл, его сначала необходимо получить и заблокировать для единоличного использования. Это не позволит редактировать файл другим пользователям, пока тот, кто заблокировал файл, не сделает необходимые изменения. Чтобы получить и заблокировать файл, следует использовать команду `co -l`:

```
# co -l rc.conf
rc.conf,v --> rc.conf
revision 1.1 ❶(locked)
done
```

Операция получения файла выглядит почти так же, как и раньше, но обратите внимание на слово `locked` ❶. Этот файл захвачен и заблокирован. Только пользователь, выполнивший блокировку, может сохранять этот файл, пока он не будет разблокирован. Запустив после этой операции `ls -l` еще раз, можно увидеть, что права доступа к файлу снова разрешают чтение и запись, а значит, этот файл можно сохранять (права доступа рассматриваются в главе 7). Любой, кто попытается захватить этот файл, получит предупреждение о том, что файл используется, при этом будет указано имя того, кто этот файл заблокировал.

Повторная регистрация

После завершения работы с файлом его можно зарегистрировать и снять блокировку, чтобы другие пользователи могли его редактировать. Чтобы оставить незаблокированную копию файла в текущем каталоге, следует использовать ключ `-u`.

```
# ci -u rc.conf
rc.conf,v <-- rc.conf
❶new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> ❷clean up unneeded services
>> ❸.
done
```

При передаче файла в систему RCS команда `ci` присвоит ему новый номер версии ❶ и запросит сообщение, которое будет записано в протоколе ❷. Здесь надо ввести краткое описание внесенных изменений. В многопользовательской системе, возможно, понадобится описать причины, по которым были сделаны изменения. Если у вас работает система регистрации сообщений об ошибках, будет нелишним перечислить в описании номера сообщений, благодаря этому пользователи смогут получать полную историю изменений, сославшись на номер сообщения. Как и при первой регистрации, текст описания следует завершить символом точки на отдельной строке ❸.

Благодаря сообщениям, хранимым в протоколе, другие пользователи легко узнают обо всех сделанных изменениях или могут увидеть, какие попытки предпринимались при ошибочных изменениях, когда кому-либо приходилось начать отладку. Протоколы RCS могут быть полезны и владельцу файла, спустя несколько месяцев удивленно пытающемуся вспомнить, о чем же он думал в то время.

Теперь, когда ясны основы регистрирования и захвата файлов, исследуем более интересные функции RCS: получение старых версий файлов, снятие блокировок, нахождение различий между версиями файлов и помещение идентификаторов RCS в файлы.

Просмотр протоколов RCS

Самый простой способ просмотреть историю изменений файла – это просмотреть протокол RCS с помощью команды rlog(1). Эта команда отобразит все сообщения, которые были введены для указанного файла. В приведенном ниже примере выполняется извлечение сообщений из протокола RCS для файла *rc.conf*, расположенного на другой машине:

```
# rlog rc.conf
❶ RCS file: RCS/rc.conf,v
Working file: rc.conf
❷ head: 1.3
branch:
```

```
locks: strict
access list:
symbolic names:
keyword substitution: kv
total revisions: 3; selected revisions: 3
description:
❸ system boot config
----------------------------
❹ revision 1.3
❺ date: 2006/02/17 22:23:45; ❻author: mwlucas; state: Exp; ❼lines: +2 -2
❽ rename new interfaces
----------------------------
revision 1.2
date: 2006/02/16 20:09:46; author: mwlucas; state: Exp; lines: +4 -0
❾ *** empty log message ***
----------------------------
revision 1.1
...
```

Здесь представлена вся полезная информация. Здесь видно, что файл RCS находится в каталоге с именем RCS ❶ и что файл имеет номер версии 1.3 ❷. Описание файла, которое было введено при первой регистрации, расположено первым ❸.

Затем следуют отдельные записи для каждой версии. Версия 1.3 ❹ была зарегистрирована 17 февраля 2006 года ❺ в 22:23. Файл был зарегистрирован пользователем mwlucas ❻ – очевидно, я допоздна задержался на работе. Изменения были не очень большими – в файл были добавлены две новые строки и две строки были удалены ❼. Наконец, протокольное сообщение говорит о том, что были переименованы два интерфейса ❽. Я совершенно не помню, чтобы делал это, впрочем, это и неудивительно, раз я работал так поздно!

Самое интересное, что отсутствует описание для версии 1.2. Очевидно, в этот день я был небрежен. Интересно, что я изменил?

Просмотр истории версий файла

Увидеть отличия между двумя версиями файла позволяет команда rcsdiff(1). Эта программа принимает три аргумента: два номера версий и имя файла, как показано ниже. Я рекомендую добавлять ключ `-u`, чтобы различия выводились в более удобочитаемом виде.

```
# rcsdiff -u -rolderversionnumber -rnewerversionnumber filename
```

Например, запустив команду `rcsdiff -u -r1.1 -r1.2 rc.conf`, я мог бы увидеть примерно следующее:

```
RCS file: RCS/rc.conf,v
retrieving revision 1.1
retrieving revision 1.2
❶ diff -u -r1.1 -r1.2
```

```
--- rc.conf 2006/02/10 18:09:54 1.1
+++ rc.conf 2006/02/16 20:09:46 1.2
@@ -10,12 +10,16 @@
ifconfig_sk0="name internet inet 172.16.88.3 netmask 255.255.255.0"
ifconfig_sk1="name dmz inet 192.168.3.1 netmask 255.255.255.0"
ifconfig_sk2="name mwlprivate inet 192.168.0.1 netmask 255.255.255.252"
❷ +
+
usbd_enable="YES"
sshd_enable="YES"
ntpd_enable="YES"
❸ -syslogd_flags="-l /var/run/log -l /var/named/var/run/log"
moused_enable="YES"
named_enable="YES"
apache21_enable="YES"
❹ +snmpd_enable="YES"
```

Команда `rcsdiff` просто извлекает две версии указанного файла и затем передает их утилите diff(1) ❶. Команда rcsdiff распознает большую часть ключей утилиты diff(1), поэтому, если вы знакомы с diff, то с помощью этих ключей вы можете настраивать вывод результатов под себя. Строки, начинающиеся с символа «плюс» (+) ❷, были добавлены в файл, а строки, начинающиеся с символа «минус» (-) ❸, были удалены. В данном случае было добавлено несколько пустых строк, удалена переменная `syslog_flags` (тем самым был выполнен возврат к поведению по умолчанию, которое описано в файле */etc/defaults/rc.conf*) и добавлен запуск демона snmpd ❹. Теперь все понятно. Жаль, что я не оставил описание для себя самого, чтобы потом не приходилось копаться в отличиях версий, но хотя бы я знаю, что было сделано. Наличие такой возможности очень удобно, особенно на рабочих системах, и особенно на тех, которые администрируются несколькими администраторами.

При запуске rcsdiff можно задавать произвольные номера версий. Это позволяет просмотреть изменения, выполненные в период между созданием любых двух версий. В предыдущем примере мы просмотрели различия между двумя соседними версиями, но точно так же я мог бы затребовать сведения о различиях между версиями 1.1 и 1.3 или даже все изменения, которые были произведены в течение предыдущего года.

Получение старых версий

Если различия между двумя версиями файла слишком велики, их просмотр может оказаться непростым делом. В некоторых случаях diff не в состоянии предоставить достаточный контекст, чтобы понять эти изменения. В подобной ситуации проще будет получить старую версию файла и ознакомиться с ней. Чтобы извлечь старую версию файла из RCS, следует использовать ключ `-r`. Номер версии следует указывать сразу вслед за ключом `-r`, без пробела:

```
# co -r1.1 rc.conf
rc.conf,v --> rc.conf
revision 1.1
done
```

Здесь была извлечена версия 1.1, которая перезаписала существующий файл *rc.conf*.

Минутку! Перезаписывать существующие файлы, особенно те, что имеют жизненно важное значение для системы, – это неправильно. Чтобы указать иное место сохранения извлекаемого файла, следует использовать ключ `-p`, благодаря которому содержимое файла будет выводиться на экран, и этот вывод можно перенаправить в файл.

```
# co ❶-r1.1 ❷-p rc.conf ❸> /tmp/rc.conf.original
rc.conf,v --> standard output
revision 1.1
```

Здесь выполняется получение файла *rc.conf* версии 1.1 ❶, благодаря наличию ключа `-p` ❷ содержимое файла будет выводиться непосредственно на терминал. Правая угловая скобка (`>`) ❸ перенаправляет вывод в файл, в данном случае */tmp/rc.conf.original*.

Снятие блокировок

Если вы не передали файл обратно в RCS, никто не сможет получить его и заблокировать. Если в течение дня вы уйдете домой, это может задеть других и может быть даже рассердить. Если вы ушли в отпуск, будьте готовы к тому, что его прервут. К счастью, нет никакой необходимости ждать вашего возвращения, чтобы разблокировать файл – ваши коллеги могут снять вашу блокировку и получить файл. Это как библиотекарь, который пришел к вам домой с бензопилой в руках и намекнул, что пора возвращать книгу.

Как потенциальный взломщик блокировок, вы должны быть предусмотрительны. Если кто-то действительно вносит изменения в файл в то время, как вы снимаете блокировку, то хозяина это не только не обрадует, но и серьезно разозлит. Поэтому постарайтесь сначала найти этого человека, прежде чем снимать блокировку! Однако, если вы все-таки решили снять блокировку, запустите для выбранного файла команду `rcs -u`. RCS попросит ввести сообщение, объясняющее причину снятия блокировки, которое будет послано по электронной почте пользователю, заблокировавшему файл.[1]

После снятия блокировки файл становится доступен другим пользователям для редактирования. Однако изменения, сделанные беспечным взломщиком, останутся в файле. Когда кто-то выполнит повторную

[1] Лично я рассматриваю такое сообщение как возможность обругать человека, который оставил файл заблокированным, но у вас могут быть иные взгляды.

передачу файла в RCS, эти изменения будут затерты. Поэтому, если эти изменения имеют большое значение, желательно скопировать файл куда-нибудь на временное хранение, пока вы не сможете получить и заблокировать файл снова.

Автоматизированный поиск блокировок

Поиск в Интернете позволит вам выяснить массу методов идентификации всех файлов, которые были оставлены в заблокированном состоянии. Если в вашей системе пользователи постоянно оставляют файлы заблокированными, можно попробовать выполнять поиск заблокированных файлов и тех, кто их заблокировал, с помощью cron и ежедневно автоматически отправлять списки нарушителей группе системных администраторов.

Получение нескольких файлов

Регистрация сразу нескольких файлов, участвующих в едином изменении, – довольно редкая ситуация. Например, изменение в DNS может потребовать вносить изменения сразу в два или более файла зон (глава 14). Для этих файлов обычно требуется оставить одно протокольное сообщение, а вам едва ли захочется вводить его несколько раз. Для примера я зарегистрирую файлы *blackhelicopters.org.db* и *absolutefreebsd.com.db* с протокольным сообщением `update for new mail server`.

```
# ci -u -m"update for new mail server" blackhelicopters.org.db
absolutefreebsd.com.db
```

Обратите внимание на отсутствие пробела между ключом `-m` и кавычками, окружающими текст сообщения. Система RCS зарегистрирует все файлы из списка и к каждому из них добавит заданное сообщение в протокол.

RCS и строки идентификации

Строки идентификации (ident strings) позволяют упростить просмотр сведений о том, кто и когда изменял тот или иной файл. Так, если последнюю неделю сервер ведет себя странно, полезно знать, что изменилось неделю назад. Конечно, можно запустить rlog(1) для каждого конфигурационного файла системы и проанализировать изменения, но это немного утомительно. Куда лучше просто взглянуть на файл, в котором представлена вся необходимая информация. Вот где нужны строки идентификации. Эти строки можно поместить в файлы, сохраняемые в RCS. Когда пользователь захватывает файл, RCS автоматически обновляет эти строки.

Строки идентификации имеют формат `$string$`. Например, в строке `$Id$` размещается информация о последнем изменении в файле. Всегда

полезно разместить `#$Id$` в первой строке важных конфигурационных файлов, таких как */etc/rc.conf*. Первый символ `#` сообщит сценарию */etc/rc*, что это комментарий и строку следует пропустить, но когда файл будет зарегистрирован в RCS, она примет следующий вид:

```
#$Id: rc.conf,v ❶1.3 ❷2006/04/12 ❸17:12:49 ❹mwlucas ❺Exp $
```

Простая строка идентификации была дополнена системой! Здесь можно увидеть номер версии файла ❶, дату ❷ и время ❸ последнего изменения и имя пользователя, выполнившего это изменение ❹. Это дает возможность легко ответить на вопрос: «Файл изменялся, и с кем теперь нужно поговорить об этом изменении?» Последний элемент строки – состояние RCS, произвольная строка, которую можно назначить с помощью ci(1) или rcs(1). Для файла можно задать произвольные состояния, которые позволяют понять назначение файла и условия его использования. Многие люди применяют эту строку, чтобы пометить файл как «экспериментальный» или «рабочий» или указать: «Ничего не меняйте ни по каким причинам». Обычно в системном администрировании строка состояния RCS не применяется.

Строка идентификации `$Id$` встречается чаще всего, но имеются и другие строки подобные строки, например `$Header$` и `$Log$`. `$Header$` сходна с `$Id$`, за исключением того, что она сообщает полный путь к файлу RCS, а не просто имя файла. `$Log$` – интересная строка идентификации, добавляющая протокольные сообщения RCS к самому файлу; при просмотре файла эти сообщения можно увидеть. Они могут переполнять интенсивно редактируемые файлы, но могут быть полезными для файлов, которые изменяются редко. Так, */etc/rc.conf* на рабочей системе может вообще не изменяться месяц. Если описываемую строку идентификации поместить в этот файл, в нем можно будет просматривать все протокольные сообщения RCS. Сразу станет ясно, что изменено, кем и почему. Большинство других строк идентификации по сути являются подмножеством строк `$Id$`, `$Header$` и `$Log$`. Полный перечень строк идентификации вы найдете в странице руководства ident(1).

Теперь, когда вы узнали, как создавать резервные копии системы и отслеживать вносимые изменения, вы можете более спокойно выполнять свою работу, зная, что в любой момент сможете восстановить систему в состояние, предшествующее последним изменениям.

Запись происходящих событий

Теперь вы можете создавать резервные копии системы, а также отслеживать изменения, производившиеся в единственном файле. Нам осталось лишь научиться выводить на экран информацию о происходящих событиях. Программа script(1) – один из редко упоминаемых, но довольно полезных инструментов, о которых должен знать каждый системный администратор. Она записывает все набранные команды, а также все, что появляется на экране. Этот инструмент можно ис-

пользовать для записи ошибок или большого вывода команд, который можно проанализировать впоследствии. Например, если программа всякий раз спотыкается на одном и том же месте, можно задействовать script для копирования набранных строк и ответов, появляющихся на экране. Это особенно полезно при обновлении системы или компоновке программного обеспечения из исходного кода; последние 30 строк (или около того) файла протокола представляют собой прекрасное дополнение к письму о помощи.

Чтобы запустить script(1), достаточно просто ввести команду `script`. После этого вы вернетесь в командную оболочку и сможете продолжить свою обычную работу. Когда потребуется остановить запись, достаточно просто ввести команду `exit` или нажать комбинацию клавиш Ctrl-D. Все ваши действия будут записаны в файл *typescript*. Если потребуется дать файлу другое имя или сохранить его в конкретном каталоге, достаточно просто передать команде `script` требуемое имя файла в виде параметра:

```
# script /home/mwlucas/debug.txt
```

В письмо о проблеме, отправляемое в почтовую рассылку *FreeBSD-questions@FreeBSD.org*, будет чрезвычайно полезно добавить точную запись того, что вводилось с клавиатуры и что выводилось на экран в ответ.

Диск восстановления

Наилучший способ изучить UNIX состоит в том, чтобы поиграть с этой системой, и чем труднее игра, тем больше можно узнать. Если игра идет не на шутку, что-нибудь обязательно «полетит». Восстановление неработоспособной системы, возможно, является самой короткой дорогой к получению знаний. Этот раздел предназначен для тех, кто помог системе перестать загружаться или планирует достаточно быстро набраться знаний, чтобы решиться на это. Вам придется научиться многому и быстро, хотя, в основном, и самостоятельно.

Однопользовательский режим (обсуждается в главе 3) дает доступ к множеству различных команд и инструментов. Что, если будет уничтожен один из таких инструментов? Возможно, вам даже удастся уничтожить программы в каталоге */rescue*. Это как раз та ситуация, когда в дело идет диск восстановления.

Диск восстановления – это образ «действующей файловой системы» FreeBSD на компакт-диске. Он включает в себя все программы, которые по умолчанию поставляются в составе FreeBSD. Можно начать загрузку с установочного носителя, но вместо установки ОС выбрать режим восстановления (*fixit mode*).

Чтобы успешно применять систему восстановления, необходимо в какой-то мере понимать системное администрирование. По существу, диск восстановления предоставляет в ваше распоряжение командную

строку и множество утилит UNIX. Для устранения неполадок необходимо задействовать свою голову и проанализировать сообщения об ошибках, выданные во время загрузки. Вы – против компьютера. В первых трех случаях из шести, когда я прибегал к помощи диска восстановления, победу одерживал компьютер. Однако время было потрачено не зря, поскольку теперь мне по силам восстановить поврежденную систему. Эту книгу определенно следует прочесть до конца, прежде чем пытаться что-то делать.

Пошаговое изложение процесса восстановления не представляется возможным; последовательность действий целиком зависит от ущерба, причиненного бедному, невинному компьютеру. Не надо отчаиваться – режим восстановления поможет избежать полной переустановки системы. Бывало, я случайно уничтожал каталог */etc* или «подвешивал» программу getty(1), которая отображает приглашения для ввода регистрационного имени (login prompt). Осмотрительное применение режима восстановления помогает найти выход в таких ситуациях за долю того времени, которое бы потребовалось для переустановки системы.

Примечание

Важно, чтобы диск восстановления примерно соответствовал запускаемой версии FreeBSD. Мелкие различия не играют роли, однако попытка восстановить систему 6.5 с помощью диска восстановления 8-current обречена на неудачу.

Начните загрузку с установочного носителя. В первом меню будет предоставлена возможность выбора режима восстановления. Выберите его. Далее выберите носитель: CD или дискету. Если CD есть, следует предпочесть его. (Дискета восстановления содержит только программы, которые можно разместить на одном гибком диске, тогда как на CD записан полный комплект программ, которые есть в комплекте FreeBSD.) Хотя на CD может не оказаться любимого редактора или предпочтительной оболочки, на нем есть все, что «действительно необходимо».

В некоторых случаях можно надеяться лишь на то, что удастся смонтировать жесткий диск и считать с него данные. Тут поможет компакт-диск восстановления, содержащий все инструменты для подключения системы к сети, благодаря чему можно будет смонтировать жесткий диск в режиме «только для чтения» и скопировать жизненно важные данные на другой компьютер. Это даст вам возможность создать последнюю резервную копию, прежде чем снести систему и выполнить ее переустановку. Если диск восстановления не содержит всех инструментов, необходимых для проведения восстановления, можно попробовать загрузиться с компакт-диска, содержащего загружаемую версию FreeBSD, например FreeSBIE. Мы еще коснемся FreeSBIE в главе 20.

Теперь, когда вы можете восстановить систему практически после любой ошибки, начинаем погружение в сердце FreeBSD – ядро.

5

Эксперименты с ядром

Обычно первый шаг в оптимизации FreeBSD – конфигурирование ядра (kernel). Новичков в администрировании UNIX слово *ядро* может устрашить. Ведь ядро – это одна из тех скрытых частей системы, которые не предназначены для дилетантского вмешательства простых смертных. Действительно – в некоторых версиях UNIX непосредственные манипуляции с ядром немыслимы. Компания Microsoft вообще замалчивает факт наличия ядра в своих операционных системах, как бы ставя под сомнение наличие мозгов у пользователей.[1] Хотя опытные пользователи могут получить досту п к ядру тем или иным способом, тем не менее такая возможность не предназначена для широких масс. Впрочем, в мире UNIX с открытым исходным кодом вмешательство в работу ядра считается приемлемым и даже ожидаемым способом улучшить производительность системы. Вероятно, если есть такая возможность, это наилучший способ и для настройки других систем.

Ядро FreeBSD можно настраивать динамически, или изменять на лету. При необходимости можно изменять большинство аспектов системной производительности. Далее будет обсуждаться интерфейс sysctl ядра и его применение для модификации рабочего ядра.

В то же время некоторые части ядра не подлежат изменению непосредственно при работе системы, а для настройки некоторых функций ядра необходима значительная реконфигурация. Например, существует возможность добавлять поддержку новых устройств или удалять поддержку устройств, которые не используются. Для этого лучше всего собрать собственное ядро, и далее будет показано, как это сделать.

Ядро FreeBSD имеет модульную архитектуру, что позволяет загружать и выгружать целые участки ядра, включая или выключая целые под-

---

[1] Здесь я мог бы добавить массу комментариев, но все они будут слишком мягкими. У меня есть *кое-какие* принципы, и вы об этом знаете.

системы по мере необходимости. Это очень удобно на данном этапе развития сменных устройств, таких как PC-карты и устройства USB. Загружаемые модули могут оказывать воздействие на производительность и поведение системы, а также на поддержку аппаратных устройств.

В заключение в этой главе будут рассмотрены основы отладки ядра, включая анализ порой устрашающих сообщений, а также поговорим о том, когда и как использовать альтернативные ядра.

Что такое ядро?

Ядро (kernel) имеет много разных определений. Значительная их часть абсолютно сбивает с толку. Некоторые определения являются технически корректными, но непонятными для начинающих пользователей, тогда как другие совершенно неправильные. Следующее определение не претендует на полноту, но оно понятно и удовлетворяет нашим целям: *ядро – это интерфейс между аппаратным и программным обеспечением*.

Ядро позволяет записывать данные на дисковые накопители и работать в сети. Когда программе требуется память, ядро выполняет все необходимые действия, организуя доступ к микросхемам физической памяти и выделяя ресурсы, необходимые для выполнения задания. Оно преобразует MP3 в поток нулей и единиц, которые понимает звуковая карта. Когда программа требует выделить ей квант процессорного времени, ядро обслуживает этот запрос и выделяет время. Проще говоря, ядро предоставляет интерфейсы программам, которым необходим доступ к аппаратным средствам.

Описывать работу ядра легко (по крайней мере, в этой упрощенной манере), но ее трудно выполнять. При взаимодействии с аппаратными средствами различные программы опираются на различные интерфейсы ядра, а аппаратные средства предоставляют свои ресурсы разными способами. Ядро должно справляться со всем этим. Например, ядро поддерживает несколько десятков типов сетевых карт, причем карты каждого типа предъявляют свои требования, которые ядро должно удовлетворить. Если ядро окажется не в состоянии обслуживать сетевую карту, система не сможет подключиться к сети. Программы запрашивают память различными способами, и если программа запрашивает память способом, который не поддерживается ядром, возникает затруднение. При загрузке ядро получает сведения об аппаратных средствах, и их дальнейшая работа определяется полученными сведениями. Значит, этим процессом надо управлять. Некоторые устройства идентифицируют себя довольно дружественным, по отношению к ядру, способом, а другие накрепко блокируются при попытке узнать, что это за устройства и для чего они предназначены.

Фактически, ядро и его модули – это файлы в каталоге */boot/kernel*. Файлы, находящиеся за пределами этого каталога, не являются частя-

ми ядра; эти файлы и программы называются *пользовательским пространством* (*userland*), то есть они предназначены для пользователей, даже если они во время работы обращаются к ядру.

Поскольку ядро – это всего лишь набор файлов, вы можете создавать свои, альтернативные ядра, предназначенные для особых случаев. В системе, где была выполнена сборка собственного ядра, можно обнаружить каталог */boot/kernel.old*, где находится ядро, работавшее до того, как была выполнена установка текущего ядра. Я предпочитаю хранить ядро, установленное вместе с системой в каталоге */boot/kernel.install*. Вы также можете создавать свои собственные ядра. Команда FreeBSD стремится к тому, чтобы сделать настройку и установку ядра как можно проще. Самый простой способ изменить ядро – это использовать интерфейс sysctl.

sysctl

Программа sysctl(8) позволяет взглянуть на параметры ядра и в некоторых случаях установить их значения. Запутаем дело: иногда эти параметры тоже называются sysctls. Sysctl – это мощный инструмент, поскольку во многих случаях он позволяет увеличивать производительность без повторной сборки ядра или переконфигурирования приложения. К сожалению, это палка о двух концах – таким образом можно нарушить работу программ и сделать пользователей очень несчастными.

Все операции над параметрами настройки ядра (sysctls) выполняются с помощью команды sysctl(8). По ходу изложения материала я буду обращать внимание на ту или иную операцию sysctl, которая изменяет поведение системы, но сначала эти операции необходимо понять. Прежде чем взяться за sysctls, выясним, какие из них доступны в системе. Следующая команда сохранит список операций в файле, который можно легко изучить:

```
# sysctl -A > sysctl.out
```

После выполнения этой команды файл *sysctl.out* будет содержать сотни параметров настройки ядра и их значений. Большая их часть пока неясна, но некоторые понять нетрудно:

```
kern.hostname: humvee.blackhelicopters.org
```

Этот параметр sysctl, называемый `kern.hostname`, имеет значение `humvee.blackhelicopters.org`. Система, на которой я запустил эту команду, называется *humvee.blackhelicopters.org*. По названию этого параметра ядра нетрудно догадаться, что он хранит имя компьютера. Если бы все было так просто...

```
kern.ipc.msqids: Format: Length:3520
Dump:0xe903e903e903e903c0010100168f7542...
```

Я понятия не имею, какую роль играет переменная kern.ipc.msqids, и еще меньше понимаю смысл ее значения. Все же, если у меня возникают трудности, я могу получить нужную информацию, обратившись за помощью к поставщику системы или в почтовую рассылку.

Организация sysctl

Параметры настройки ядра организованы по древовидной схеме, которая называется базой управляющей информации (Management Information Base, MIB). В ней есть несколько основных категорий, таких как net (network – сеть), vm (virtual memory – виртуальная память) и kern (kernel – ядро). В табл. 5.1 перечислены корневые элементы дерева MIB sysctl, присутствующие в системе, работающей на базе ядра GENERIC.

Таблица 5.1. Корни дерева sysctl MIB

Sysctl	Назначение
kern	Основные функции ядра
vm	Подсистема виртуальной памяти
vfs	Файловые системы
net	Сетевые функции
debug	Отладочная информация
hw	Информация об аппаратных средствах
user	Информация об интерфейсах пространства пользователя
p1003_1b	Параметры, управляющие характеристиками POSIX[a]
compat	Совместимость ядра с программным обеспечением других операционных систем (глава 12)
security	Параметры обеспечения безопасности
dev	Информация драйверов устройств

[a] POSIX – это международный стандарт, описывающий характеристики и возможности UNIX-подобных операционных систем. К сожалению, стандарт POSIX сильно изменился за последние годы, причем эти изменения иногда настолько непоследовательны, что системы, соответствующие требованиям одной версии POSIX, могут оказаться несовместимы с другими версиями стандарта. Если вы хорошо знакомы со стандартом POSIX, вы наверняка знаете, что это за различия. С помощью параметров sysctl вы сможете точно узнать о поведении FreeBSD и какой версии стандарта она соответствует.

Каждая категория, в свою очередь, подразделяется на подгруппы; например, категория net, охватывающая все сетевые sysctls, разделена на категории IP, ICMP, TCP и UDP. Термины sysctl MIB и sysctl часто употребляются попеременно. Концепция, лежащая в основе базы управляющей информации (MIB), используется и в других частях операционной системы, в чем вы сможете убедиться в главе 20 и на протяжении всей

своей карьеры. Термины *sysctl MIB* и *sysctl* часто употребляются попеременно. Уникальные имена переменных создаются из комбинации названия родительской категории и всех вложенных подкатегорий, например:

```
...
kern.maxfilesperproc: 11095
kern.maxprocperuid: 5547
kern.ipc.maxsockbuf: 262144
kern.ipc.sockbuf_waste_factor: 8
kern.ipc.somaxconn: 128
...
```

Здесь представлены пять параметров sysctl, извлеченные из категории `kern`. Первые два входят непосредственно в категорию `kern`, и с другими значениями их связывает лишь тот факт, что все они имеют отношение к ядру. Имена остальных трех параметров начинаются с `kern.ipc` – они являются частью раздела IPC (inter-process communication – взаимодействие между процессами). Если продолжить рассмотрение сохраненного ранее файла со списком параметров sysctl, можно заметить, что ветвление дерева sysctls может продолжаться на нескольких уровнях.

Значения параметров sysctl

Каждый MIB имеет значение, которое связано с неким буфером, настройками или особенностями, используемыми ядром. Изменение этого значения повлечет за собой изменение в характере работы ядра. Например, некоторые параметры настройки ядра управляют выделением памяти для каждого сетевого соединения. Если сетевая производительность низка, можно увеличить объем системных ресурсов, резервируемых для сетевых соединений.

Каждый параметр sysctl может представлять собой строку (string), целое число (integer), двоичное значение (binary value) либо неясный код (opaque). *String* – это текст свободного формата с произвольной длиной; *integer* – обыкновенное целое число; *binary value* – либо `0` (off), либо `1` (on); значение *opaque* представлено в машинном коде, который могут интерпретировать только специальные программы. Многие параметры настройки ядра недостаточно документированы. Нет отдельного документа, в котором перечислены все имеющиеся базы управляющей информации (sysctl MIB) и их назначение. Но описание каждого параметра присутствует на странице руководства к соответствующей функции, а иногда – только в исходных текстах. Например, исходная документация для MIB `kern.securelevel` (более подробно рассматривается в главе 7) представлена в init(8). Хотя за последние несколько лет объем документации по sysctl существенно вырос, тем не менее многие параметры по-прежнему вообще не имеют документации. В приложении А приведен список распространенных параметров настройки ядра и описано их назначение.

К счастью, некоторые параметры очевидны. Например, как будет говориться ниже, в этой же главе, следующий параметр имеет большое значение для тех, кто часто загружает систему с различными ядрами:

```
kern.bootfile: /boot/kernel/kernel
```

Если проводится отладка и последовательная загрузка с различных ядер, можно легко забыть, какое ядро было загружено (в действительности это происходит даже со мной), поэтому дополнительное напоминание будет совсем нелишним.

Просмотр параметров sysctl

Чтобы увидеть поддерево MIB, доступных в отдельном поддереве MIB, можно воспользоваться командой `sysctl`, передав имя поддерева, которое следует вывести. Например, чтобы увидеть все параметры в дереве, таком как `kern`, введите такую команду:

```
# sysctl kern
kern.ostype: FreeBSD
kern.osrelease: 7.0-CURRENT-SNAP010
kern.osrevision: 199506
...
```

Этот список может быть длинным. Он может пригодиться, когда необходимо выяснить, какие параметры sysctl представлены в ядре. Чтобы получить только значение отдельного параметра настройки ядра, задайте имя MIB как аргумент:

```
# sysctl kern.securelevel
kern.securelevel: -1
```

В этом случае `kern.securelevel` имеет целочисленное значение `-1`. Смысл этого значения обсуждается в главе 7.

Чтобы получить хоть какое-то представление о назначении того или иного параметра sysctl, можно использовать ключ `-d` вместе с полным именем MIB. В результате будет выведено краткое описание параметра:

```
# sysctl -d kern.maxfilesperproc
kern.maxfilesperproc: Maximum files allowed open per process
```

Это краткое описание сообщает, что данный параметр sysctl управляет именно тем, о чем можно было бы догадаться из его имени. Но это самый простой случай, в других, более сложных случаях, бывает сложно строить какие-либо предположения только на основе имени параметра.

Изменение параметров sysctl

Некоторые параметры sysctl предназначены только для чтения. Рассмотрим MIB-дерево `hw`:

```
hw.model: AMD Athlon(tm) 64 X2 Dual Core Processor 4200+
```

Разработчики FreeBSD еще не до конца разработали технологию переноса системы с AMD на аппаратные средства sparc64 с помощью программных настроек, поэтому данный параметр доступен только для чтения и его изменение может привести лишь к неработоспособности системы. Вот почему во FreeBSD такие параметры MIB доступны только для чтения. Попытки изменить их ничего не нарушат, однако будет выдано предупреждение о том, что сделать это нельзя. Но есть MIB, значения которых изменять можно. Рассмотрим следующий MIB:

```
vfs.usermount: 0
```

Его значение определяет, могут ли пользователи монтировать сменные носители, такие как CD или дискету. Изменение этого параметра не требует внесения обширных изменений в ядро или в аппаратное окружение, он всего лишь разрешает или запрещает функцию в ядре. Чтобы изменить значение параметра, следует вызвать команду `sysctl(8)`, передать ей имя параметра, знак «равно» (=) и желаемое значение:

```
# sysctl vfs.usermount=1
vfs.usermount: 0 -> 1
```

Команда sysctl(8) возвращает сообщение, в котором показаны предыдущее и новое значения. После этого параметр sysctl будет изменен. Параметры sysctl, которые подстраиваются «на лету», называются *sysctl, настраиваемые на этапе выполнения* (*run-time tunable sysctl*).

Автоматическая настройка параметров sysctl

Когда будут выполнены все необходимые настройки параметров ядра, было бы желательно, чтобы они автоматически устанавливались после перезагрузки. Для этих целей используется файл */etc/sysctl.conf*. Просто перечислите в этом файле все желаемые параметры с необходимыми значениями. Например, чтобы установить тот же самый параметр `vfs.usermount` на этапе загрузки, достаточно добавить в */etc/sysctl.conf* следующую строку:

```
vfs.usermount=1
```

Параметры sysctl, устанавливаемые на этапе загрузки

Некоторые этапы конфигурирования ядра должны предшествовать загрузке системы. Можно найти множество примеров параметров, значения которых должны определяться на этапе загрузки. Часто эти параметры связаны с низкоуровневой настройкой аппаратных устройств. Например, во время первоначального опроса ядром жесткого диска IDE драйвер устройства определяет, следует ли использовать DMA, PIO, кэширование записи или какие-либо другие особенности, характерные для жестких дисков. Это решение должно приниматься при обнаружении устройства, на этапе начальной загрузки, после чего решение изменить нельзя. Эти значения можно установить с помощью

системного загрузчика (system loader), определив необходимые настройки в файле */boot/loader.conf*, как уже говорилось в главе 3.

Настройки в *loader.conf*, как и в *sysctl.conf*, при неумелом использовании могут доставить компьютеру неприятности. Но нет причин для расстройства, потому что эти значения легко можно сбросить.

Слишком много настроек?

Не путайте между собой параметры sysctl, значения которых могут устанавливаться только на этапе загрузки, параметры, которые могут изменяться в процессе работы системы, и параметры, которые могут изменяться в процессе работы, но настроены для автоматической установки на этапе загрузки. Запомните, что настройки, выполняемые только на этапе загрузки, связаны с низкоуровневыми функциями ядра, тогда как настройки, позволяющие изменять свои значения «на лету», – с высокоуровневыми. Автоматическая настройка параметров sysctl во время загрузки – это просто пример экономии вашего времени; принадлежность параметров к той или иной категории не изменяет-

Выдача указаний драйверам устройств

Многие драйверы устройств требуют, чтобы флаги sysctl были установлены во время начальной загрузки. Узнать о них можно из страниц руководства, из этой книги и из другой документации. Несмотря на то, что эти параметры отсутствуют в файле по умолчанию *loader.conf*, тем не менее их можно добавлять в локальный *loader.conf*, чтобы автоматически задавать необходимые значения во время начальной загрузки. Например, чтобы отключить DMA для устройств ATAPI (которые будут рассматриваться в главе 8), достаточно просто поместить требуемое значение параметра sysctl в файл *loader.conf*:

```
hw.ata.atapi_dma="0"
```

Ядро установит этот флаг во время загрузки, и тем самым будет обеспечено желаемое поведение драйвера устройства.

Дополнительно многие устаревшие аппаратные устройства требуют, чтобы ядро обращалось к ним по сигналу прерывания, с четко определенным номером IRQ и по определенным адресам памяти. Если вам достаточно лет, вы должны помнить дискеты для «настройки устройств» и специальные разъемы для подключения плат расширения шин, вы знаете, о чем я говорю, и, возможно, одна из таких систем даже сейчас валяется у вас на заднем дворе. (Если же вы слишком молоды, купите выпить для одного из нас, старикашек, и слушайте[1] наши

[1] В действительности, слушать совершенно необязательно.

рассказы, наполненные ужасами.) Можно указать системе FreeBSD на необходимость опробовать все номера IRQ или указанные адреса памяти для работы с этими устройствами, что очень удобно, если имеется карта с известными параметрами настройки, но дискета с ними давно уже превратилась в прах. Загляните в файл */boot/device.hints*, где можно увидеть массу записей, таких как показано ниже:

```
    hint.❶ed.❷0.❸disabled="1"
❹ hint.ed.0.port="0x280"
❺ hint.ed.0.irq="10"
❻ hint.ed.0.maddr="0xd8000"
```

Данные записи – это указания для драйвера ed ❶. Запись предназначена для драйвера ed с номером 0 ❷. Ключевое слово `disabled` ❸ означает, что FreeBSD не будет выполнять проверку этого устройств автоматически, во время загрузки. Если будет найдена другая карта ed, ей может быть присвоен номер устройства 0. Если это устройство будет активировано, FreeBSD попробует обратиться к нему через порт с номером 0x280 ❹, по номеру IRQ 10 ❺ и по адресу памяти 0xd8000 ❻, и если карта будет обнаружена по указанным адресам, она получит имя устройства ed0. Конечно, если карта не поддерживается драйвером сетевых карт ed(4), у вас появятся другие проблемы!

Команда sysctl(8) дает вам власть выполнять настройку ядра, но настройка – это только начало пути. Следующий шаг – управление модулями ядра.

Тестирование настроек, выполняемых во время загрузки

Все эти указания драйверам и настройки, выполняемые во время загрузки, можно производить в интерактивном режиме, в командной строке загрузчика, о которой рассказывалось в главе 3. Вы можете протестировать настройки без редактирования файла *loader.conf*, отыскать правильное значение и только после этого сохранить его в файле.

Модули ядра

Как уже говорилось, модули ядра – это его компоненты, которые можно запускать (или загружать) и выгружать при необходимости. Модули ядра могут загружаться при подключении устройств и выгружаться при их отключении. Это позволяет экономить системную память и повышает гибкость системы.

Само ядро по умолчанию хранится в виде файла */boot/kernel/kernel*, модули ядра также хранятся в виде файлов в каталоге */boot/kernel*. За-

глянув в этот каталог, можно увидеть сотни файлов модулей ядра. Имена файлов всех модулей ядра имеют расширение *.ko*. Обычно файлы модулей ядра получают имена, соответствующие их функциональности. Например, в файле */boot/kernel/joy.ko* находится драйвер джойстика (от английского «joy»), описание которого находится в странице руководства joy(4). Этот драйвер делает джойстик видимым для системы в виде устройства joy0.

Просмотр списка загруженных модулей

В главе 3 было показано, как получить список загруженных модулей перед загрузкой, но этот метод не годится, после того как система загрузится. Получить список загруженных в настоящий момент модулей ядра можно с помощью команды kldstat(8).

```
    # kldstat
    Id  Refs   Address    Size     Name
❶   1    15   0xc0400000  6a978c   kernel
❷   2     1   0xc0aaa000  6228     snd_via8233.ko
❸   3     2   0xc0ab1000  23898    sound.ko
```

На этом ноутбуке загружено три модуля ядра. Первый – это ядро ❶, далее идут драйвер звуковой карты ❷ и звуковая подсистема ❸. (Это совершенно неудивительно, так как речь идет о ноутбуке.) Каждый модуль содержит один или более подмодулей, которые можно увидеть с помощью команды `kldstat -v`, но само ядро содержит несколько сотен подмодулей, поэтому будьте готовы получить очень длинный список.

Загрузка и выгрузка модулей

Загрузка и выгрузка модулей ядра производится с помощью команд kldload(8) и kldunload(8). Например, обычно мой ноутбук подключается к сети посредством проводного соединения Ethernet. При беспроводном подключении мне необходимо загрузить модуль ядра *wlan_wep.ko*, который использует WEP-шифрование. Для этого я пользуюсь командой `kldload`, которой передаю имя файла модуля ядра с требуемой функциональностью:

```
# kldload /boot/kernel/wlan_wep.ko
```

Как только беспроводное подключение станет ненужным, я выгружаю модуль.[1] Для этого требуется указывать не имя файла, а имя модуля, в том виде, как его отображает команда kldstat:

```
# kldunload wlan_wep.ko
```

Если бы все возможные функции компилировались непосредственно в ядро, оно имело бы очень большой размер. Модульная архитектура

[1] На самом деле я едва ли буду беспокоиться об этом, потому что я выключу ноутбук, но сама идея, я думаю, вам понятна.

позволяет уменьшить ядро, сделать его более эффективным, а редко используемые функции загружать только в случае необходимости.

Команды kldload(8) и kldunload(8) не требуют указания полного пути к модулю ядра, точно так же они не требуют указания расширения файла *.ko*. Если вы помните точное имя файла модуля ядра, можно использовать примерно такие команды:

```
# kldload wlan_wep
# kldunload wlan_wep
```

Лично я, со своим слабеньким мозгом, полагаюсь на функцию автоматического дополнения, присутствующую в командной оболочке, которая напоминает мне полные и правильные имена модулей.

Автоматическая загрузка модулей ядра

Для автоматической загрузки модуля надо добавить его в файл */boot/loader.conf*. Файл по умолчанию *loader.conf* содержит множество примеров загрузки модулей ядра, в которых используется один и тот же синтаксис. Нужно взять имя модуля ядра, убрать расширение файла *.ko* и добавить к нему строку `_load="YES"`. Например, чтобы автоматически загрузить модуль */boot/kernel/procfs.ko*, в файл *loader.conf* нужно добавить такую строку:

```
procfs_load="YES"
```

Самое сложное во всем этом – узнать точное имя модуля, который требуется загрузить. Пользоваться драйверами устройств проще простого – если ядро не поддерживает новую сетевую карту или устройство SCSI, тогда вместо перенастройки ядра можно просто загрузить модуль драйвера. В этом случае нужно выяснить, какой драйвер поддерживает подключенное устройство, и здесь вам помогут страницы руководства и поисковая система Google. Везде в этой книге я буду упоминать модули ядра, которые помогут решать те или иные проблемы.

Впрочем, подождите минутку – зачем FreeBSD вынуждать загружать драйверы устройств, если она сама определяет их во время загрузки? Отличный вопрос! Дело в том, что вы можете собрать свое собственное ядро и удалить поддержку неиспользуемого устройства. Вы до сих пор не знаете, как собрать свое собственное ядро? Сейчас мы исправим этот недостаток.

Сборка собственного ядра

В какой-то момент становится ясно, что существующее ядро уже не удается подгонять под стоящие задачи только с помощью модулей и sysctl. Единственное решение в этом случае – собрать собственное ядро. Не стоит волноваться – этот процесс очень прост, поскольку в данном случае речь идет не о написании программного кода, а всего лишь о редактировании текстового файла и о запуске пары команд. Если не-

укоснительно следовать инструкциям, то сборка ядра становится достаточно безопасным процессом. *Отказ* от следования рекомендациям – это как езда на автомобиле по встречной полосе. (В центре, в час пик.)

Ядро, поставляемое по умолчанию, называется GENERIC. Это ядро предназначено для работы с широким спектром аппаратных средств, но оно не обязательно будет работать хорошо или оптимально. GENERIC прекрасно запускается на системах, выпущенных в последнее десятилетие или около того. Однако новейшие устройства обладают возможностями оптимизации своей работы, которые не поддерживаются ядром GENERIC, ориентированном на «наименьший общий знаменатель». Даже в этом случае оно прекрасно подходит для повседневного использования. При настройке ядра можно подключить необходимые функции оптимизации, добавить поддержку новых устройств, удалить поддержку устройств, которые не используются, или активировать какие-либо новые возможности.

Процесс сборки ядра нередко считают неким таинственным ритуалом, но сообщество поддержки FreeBSD не будет долго думать над вопросом – собирать или не собирать новое ядро, чтобы включить или исключить какую-либо особенность. Не надо думать, что пересборка ядра способна решить любую проблему; тем не менее этим умением должен обладать любой системный администратор.

Подготовка

Для сборки ядра необходим его исходный код. Если читатель последовал совету, данному в главе 2, то все уже готово. Если нет, надо вернуться в программу установки и загрузить исходный код ядра или перейти к главе 13 и использовать команду cvsup(8). Если неизвестно, инсталлирован ли исходный код ядра, поищите каталог */sys*. Если он существует, и в нем имеются файлы и подкаталоги, значит, искомый код у вас есть.

Перед созданием нового ядра выясните, какие аппаратные средства присутствуют в системе. Порой это нелегко, поскольку торговая марка на компоненте необязательно имеет хоть какое-то отношение к возможностям и отличительным чертам устройства. Многие компании занимаются ребрэндингом универсальных компонентов – я помню одного производителя, который выпускал четыре разных типа сетевых карт с одним и тем же названием и не утруждал себя необходимостью указывать номер версии на первых трех из них. Единственный способ отличить их состоял в том, чтобы пробовать использовать разные драйверы устройств, пока не будет найден тот, который работает. Похожим образом многие компании производят сетевые карты, совместимые с NE2000. Даже если на коробке написано другое название, данное производителем, микросхемы чипа могут сообщать о себе как об устройствах *NE2000*. К счастью, некоторые производители используют стандартную архитектуру для своих устройств и драйверов. Так,

вы всегда можете быть уверены, что сетевые карты компании Intel будут распознаваться драйверами устройств Intel.

Чтобы узнать, какие аппаратные средства обнаружила система FreeBSD, лучше всего обратиться к */var/run/dmesg.boot*, о чем уже говорилось в главе 3. Каждая запись в этом файле представляет либо аппаратное устройство, либо программную особенность ядра. Приступая к сборке нового ядра, всегда держите под рукой файл *dmesg.boot*.

Шины и подключения

Каждое устройство в компьютере подключается к некоторому другому устройству. Если внимательно просмотреть файл *dmesg.boot*, можно увидеть целые цепочки устройств, подключенных друг к другу. Ниже приводится несколько сообщений из файла, в демонстрационных целях отредактированных:

```
❶ acpi0: <PTLTD RSDT> on motherboard
❷ acpi_ec0: <Embedded Controller: GPE 0xb> port 0x62,0x66 on ❸acpi0
❹ cpu0: <ACPI CPU> on acpi0
  cpu1: <ACPI CPU> on acpi0
❺ pcib0: <ACPI Host-PCI bridge> port 0xcf8-0xcff on acpi0
❻ pci0: <ACPI PCI bus> on ❼pcib0
```

Первое устройство в данной системе – это acpi0 ❶. Скорее всего, вы не знаете, что это за устройство, но вы всегда можете воспользоваться командой `man acpi`, чтобы ликвидировать этот недостаток. (Или прочитать оставшуюся часть главы.) Следующее устройство в системе – acpi_ec ❷, и оно подключено к устройству acpi0 ❸. Микропроцессоры ❹ также подключены к acpi0, как и мост PCI ❺. Наконец, имеется первая шина PCI, pci0 ❻, подключенная к мосту PCI, а не к acpi0. Обычно устройства PCI подключаются к иерархии шин PCI, которая в свою очередь подключается к мосту PCI, чтобы иметь возможность обмениваться данными с остальными компонентами компьютера. Можно было бы ознакомиться со всем содержимым файла *dmesg.boot* и нарисовать дерево подключения всех устройств в системе – хотя это и не является необходимым условием, тем не менее знание схемы включения устройств увеличивает вероятность успешной сборки нового ядра.

Если у вас что-то вызывает сомнения, воспользуйтесь утилитой pciconf(8), чтобы увидеть, что в действительности имеется в вашей системе. Команда `pciconf -lv` перечислит все подключенные устройства PCI, имеющиеся в системе, независимо от того, были они обнаружены ядром или нет.

Сохранение рабочего ядра

Из-за плохого ядра система может перестать загружаться, поэтому абсолютно необходимо всегда иметь под рукой хорошее ядро. В процессе установки нового ядра старое сохраняется на всякий случай в виде файла */boot/kernel.old*. Это замечательно – иметь возможность вер-

нуться к рабочему ядру в случае ошибки, но я рекомендую двинуться дальше.

Если хорошего надежного ядра нет, события могут развиваться так. При сборке нового ядра вы вдруг обнаруживаете, что допустили незначительную оплошность, и решаете устранить ее, еще раз пересобрав ядро. В этом случае заново собранное ядро становится текущим, а предыдущее (с изъяном) в свою очередь заменит старое, рабочее ядро, которое исчезнет. Когда обнаружится, что новое ядро содержит ту же самую или даже более серьезную ошибку, останется только сожалеть о том, что надежное ядро утеряно.

Обычное место для хранения хорошо зарекомендовавшего себя ядра – */boot/kernel.good*. Перед настройкой ядра сохраните надежное работающее ядро:

```
# cp -Rp /boot/kernel /boot/kernel.good
```

Не бойтесь хранить несколько ядер. Дисковое пространство в наше время очень дешево. Некоторые администраторы даже размещают ядра в каталогах, названных по дате ядра. Благодаря этому они всегда могут вернуться к более ранней версии ядра. Кроме того, многие хранят копию ядра GENERIC в виде файла */boot/kernel.GENERIC* для нужд тестирования и отладки. Ядер слишком много бывает лишь в том случае, если они совершенно заполнили корневой раздел.

Формат конфигурационного файла

Ядро FreeBSD конфигурируется через текстовый файл. Для конфигурирования ядра отсутствуют какие-либо графические утилиты, управляемые системой меню, – процесс конфигурирования ядра не изменился со времен 4.4 BSD. Если вы чувствуете дискомфорт при работе с текстовыми конфигурационными файлами, тогда сборка нового ядра – не для вас.

Каждая конфигурационная запись располагается на отдельной строке. Каждая запись начинается с идентификационной метки, указывающей тип записи, за которой следует описание функции. Среди записей можно встретить множество комментариев, которые начинаются с символа решетки (#), как в следующей записи, соответствующей поддержке файловой системе FFS:

```
options FFS # Berkeley Fast Filesystem
```

В каждом конфигурационном файле ядра присутствуют записи пяти типов: `cpu`, `ident`, `makeoptions`, `options` и `devices`. Наличие или отсутствие этих записей определяет наличие поддержки аппаратных устройств или функциональных возможностей.

`cpu` Данная метка указывает, какие типы процессоров поддерживаются ядром. Конфигурационный файл ядра, предназначенного для устаревших персональных компьютеров, включал в себя несколько

записей с указанием типа процессора, такие как 486 (I486_CPU), Pentium (I586_CPU) и для серии процессоров от Pentium Pro до современных Pentium 4 (I686_CPU). Конфигурация ядра для аппаратной платформы amd64/EM64T включает в себя единственную запись с указанием типа процессора. Конфигурация ядра может включать несколько типов процессоров при условии, что они принадлежат одной и той же архитектуре, – можно собрать ядро, которое будет работать и на процессорах 486, и на процессорах Pentium, но невозможно получить ядро, которое могло бы работать как на Intel-совместимых процессорах, так и на процессорах Sparc.

`ident` Строка, начинающаяся с метки `ident`, содержит имя ядра. Именно таким образом ядро GENERIC получает свое имя. Это может быть любая произвольная строка.

`makeoptions` Данная строка содержит инструкции для программного обеспечения, выполняющего сборку ядра. Наиболее распространенный параметр – `DEBUG=-g`, который сообщает компилятору о необходимости включения в ядро отладочной информации. Отладочная информация помогает разработчикам в разрешении возникающих проблем.

`options` Записи этого типа описывают функции ядра, которые непосредственно не связаны с аппаратным обеспечением. Сюда входят файловые системы, сетевые протоколы и отладчики, встроенные в ядро.

`devices` Записи этого типа описывают устройства или *драйверы устройств*, они содержат инструкции, которые описывают, как ядро должно взаимодействовать с определенными устройствами. Если вам необходимо, чтобы система осуществляла поддержку некоторого аппаратного обеспечения, ядро должно включать драйвер этого аппаратного устройства. Некоторые записи этого типа соответствуют так называемым псевдоустройствам, которые в действительности не являются аппаратными устройствами, а обеспечивают поддержку целых категорий аппаратных устройств, таких как сетевые карты, генераторы случайных чисел или электронные диски. Здесь вполне может появиться вопрос – чем псевдоустройства отличаются от записей типа `options`. Дело в том, что псевдоустройства тем или иным способом отображаются в системе как устройства, тогда как функциональные возможности типа `options` не имеют отличительных особенностей, присущих устройствам. Например, петлевое (loopback) псевдоустройство – это сетевой интерфейс, который позволяет подключаться только к локальному компьютеру. В данном случае отсутствует какое-либо аппаратное обеспечение, тем не менее программы могут подключаться через петлевой интерфейс и обмениваться данными с программами, исполняемыми на том же самом компьютере.

Конфигурационные файлы

К счастью, обычно не требуется создавать конфигурационные файлы на пустом месте – вместо этого достаточно скопировать существующий файл и отредактировать его. Начнем с ядра GENERIC, предназначенного для вашей аппаратной архитектуры. Найти этот конфигурационный файл можно в каталоге */sys/<arch>/conf* – например, конфигурационные файлы ядра для архитектуры i386 находятся в каталоге */sys/i386/conf*, для архитектуры amd64 – в каталоге */sys/amd64/conf* и т. д. В каталоге находятся несколько файлов, из которых наиболее важными являются *DEFAULTS*, *GENERIC*, *GENERIC.hints*, *MAC* и *NOTES*:

DEFAULTS Это список параметров и устройств, поддержка которых включена по умолчанию для данной архитектуры. Наличие этого файла совершенно не означает, что можно скомпилировать и запустить ядро *DEFAULTS*, – это лишь отправная точка, позволяющая собрать ядро с минимальными возможностями.

GENERIC Этот файл содержит конфигурацию стандартного ядра. В нем присутствуют настройки, обеспечивающие поддержку стандартного аппаратного окружения, необходимую для запуска ядра на данной архитектуре – этот конфигурационный файл используется инсталлятором системы.

GENERIC.hints Это файл с указаниями, который впоследствии устанавливается как */boot/device.hints*. В данном файле содержится конфигурационная информация, необходимая для настройки устаревших аппаратных устройств.

MAC Этот конфигурационный файл ядра обеспечивает поддержку обязательного контроля доступа (Mandatory Access Controls) – системы многоуровневого управления доступом, используемой в средах с высокой степенью защиты. Этот конфигурационный файл требуется только в случае использования MAC.

NOTES Это комплексная конфигурация ядра для заданной аппаратной архитектуры. В файл *NOTES* включены все особенности, характерные для той или иной платформы. Платформонезависимые особенности можно найти в файле */usr/src/sys/conf/NOTES*.

Не редактируйте эти файлы непосредственно в каталоге с конфигурациями. Вместо этого скопируйте файл *GENERIC* в файл, содержащий имя компьютера, и редактируйте копию. Например, мой ноутбук имеет имя *humvee.blackhelicopters.org*: я мог бы скопировать файл GENERIC в файл *HUMVEE* и открыть файл *HUMVEE* в своем любимом текстовом редакторе. Ниже приводится отрывок из конфигурационного файла, точнее, та его часть, которая описывает устройства ATA:

```
# ATA and ATAPI devices
device          ata
device          atadisk         # ATA disk drives
device          ataraid         # ATA RAID drives
```

```
device          atapicd         # ATAPI CDROM drives
device          atapifd         # ATAPI floppy drives
device          atapist         # ATAPI tape drives
options         ATA_STATIC_ID   # Static device numbering
```

С символа «решетка» (#) начинаются комментарии – все, что находится после этого символа и до конца строки, игнорируется. Этот символ отделяет текст, предназначенный для компьютера, от текста, предназначенного для человека. Например, первая строка в этом фрагменте сообщает, что следующие записи отвечают за настройку устройств ATA и ATAPI. В других строках присутствуют комментарии, начинающиеся в середине строк, которые информируют о назначении каждой отдельной записи.

Сравните эти записи с парой записей в */var/run/dmesg.boot*, которые соответствуют устройствам ATA:

```
ata1: <ATA channel 1> on atapci1
acd0: DVDR <PIONEER DVD-RW DVR-K16/1.33> at ata1-master UDMA33
```

Конфигурационный файл ядра описывает шину ATA, устройство ata, и в файле *dmesg.boot* присутствует упоминание о канале ATA – ata1. Привод DVD-RW подключен к ata1. Без наличия устройства ata в конфигурационном файле ядро не смогло бы определить наличие шины ATA. Даже если бы системе удалось определить наличие устройства DVD, она все равно не знала бы, как производить обмен данными с ним. Ваша конфигурация ядра должна включать в себя все промежуточные устройства для драйверов, которые предполагают их наличие. С другой стороны, если в системе отсутствуют устройства ATA RAID, накопители на гибких магнитных дисках или ленточные приводы, вы можете удалить эти устройства из своего ядра.

Уменьшение ядра

Когда-то, давным-давно, память стоила очень дорого и карты памяти имели небольшую емкость. Когда в системе имелось 128 Мбайт, приходилось экономить каждый бит, поэтому было важно получить ядро как можно меньшего объема. В наши дни даже ноутбуки имеют по 2 Гбайта ОЗУ, и размер ядра перестал играть такую важную роль. Удаление ненужных драйверов из ядра перестало быть насущной необходимостью, но мы не будем замалчивать эту тему и будем учиться собирать ядро так, чтобы потом, когда появится такая необходимость, вам не пришлось бы изучать все сначала. В качестве учебного примера мы пройдем процесс уменьшения размеров ядра, но не считайте это жизненно важным или необходимым этапом.

Удаление записей из конфигурации ядра производится простым комментированием строк.

Типы CPU

Для большинства архитектур FreeBSD поддерживает всего два типа микропроцессоров (CPU). В архитектуре i386 – три. Удаление ненужных типов CPU из конфигурации приведет к созданию ядра, которое полностью использует преимущества, предлагаемые имеющимся микропроцессором. Например, рассмотрим процессоры Pentium и Pentium II с поддержкой инструкций MMX. Если эти инструкции поддерживаются микропроцессором, было бы желательно воспользоваться их преимуществами. С другой стороны, удаление из конфигурации упоминания о неиспользуемых типах CPU приведет к созданию ядра, которое будет способно работать только с одним типом CPU.

Вам необходимо оставить только тот тип CPU, который имеется в системе. Если тип CPU неизвестен, узнать его можно из файла *dmesg.boot*. Файл *dmesg.boot* на моем ноутбуке содержит следующие строки:

```
CPU: AMD Athlon(tm) 64 X2 Dual Core Processor 4200+ (2200.10-MHz 686-class CPU)
  Origin = "AuthenticAMD" Id = 0x20fb1 Stepping = 1

Features=0x178bfbff<FPU,VME,DE,PSE,TSC,MSR,PAE,MCE,CX8,APIC,SEP,MTRR,PGE,MCA,C
MOV,PAT,PSE36,CLFLUSH,MMX,FXSR,SSE,SSE2,HTT>
...
```

Жирным шрифтом выделена часть строки, которая сообщает, что в системе присутствует микропроцессор 686-class CPU. Это означает, что я могу удалить инструкцию `cpu`, описывающую микропроцессор I486_CPU, что сделает мое ядро меньше и быстрее. В результате при компиляции ядра, вместо медленного универсального кода, будут реализованы оптимизации, характерные для процессора 686-class. (Кроме того, это 64-битовый i386-совместимый процессор, который позволяет запускать версии FreeBSD для архитектур i386 и amd64, как обсуждалось в главе 1. Однако в примерах мы остановимся на архитектуре i386, которой наверняка обладает большинство читателей.)

Основные параметры

Вслед за записями, определяющими выбор типа CPU, имеется целый список параметров основных служб FreeBSD, таких как TCP/IP и файловые системы. Для средней системы необязательно присутствие всех этих особенностей, но их наличие увеличивает гибкость системы. Также можно обнаружить редко используемые параметры, которые можно удалить. Здесь не обсуждаются все возможные параметры ядра, а представлены только наиболее общие из них и специфичные примеры для их различных типов. Особым образом будут отмечены те, что имеют отношение к серверу Интернета. Рассмотрим следующие «опции»:

```
#options        SCHED_ULE       # ULE scheduler
options         SCHED_4BSD      # 4BSD scheduler
options         PREEMPTION      # Enable kernel thread preemption
```

Эти параметры управляют внутренним алгоритмом планирования FreeBSD. Планирование будет обсуждаться в главе 12. Возможность вытеснения (preemption) повышает эффективность FreeBSD в многозадачном режиме.

```
options         INET            # InterNETworking
options         INET6           # IPv6 communications protocols
```

Эти параметры обеспечивают поддержку сетевых протоколов. Параметр INET обеспечивает поддержку старомодного стека протоколов TCP/IP, тогда как параметр INET6 – поддержку IPv6. Программное обеспечение UNIX-подобных систем зависит от наличия поддержки TCP/IP, поэтому параметр INET следует оставить. Версия IPv6 пока не получила широкого распространения – рано или поздно вы придете к пониманию, что ее время еще не наступило, поэтому при желании этот параметр можно удалить.

```
options         FFS             # Berkeley Fast Filesystem
options         SOFTUPDATES     # Enable FFS soft updates support
options         UFS_ACL         # Support for access control lists
options         UFS_DIRHASH     # Improve performance on big directories
```

FFS – это стандартная файловая система для FreeBSD, а остальные параметры связаны с ней. Параметр SOFTUPDATES – это метод обеспечения целостности диска даже в случае некорректного завершения системы. Списки управления доступом UFS позволяют определять весьма подробные права доступа к файлам, а параметр UFS_DIRHASH обеспечивает поддержку индексирования каталогов, что повышает скорость работы с каталогами, содержащими тысячи файлов. Более подробно параметры FFS будут рассматриваться в главе 8.

```
options         MD_ROOT         # MD is a potential root device
```

Этот параметр (и все остальные параметры, оканчивающиеся на _ROOT) позволяет использовать в качестве дискового устройства для корневого раздела другие файловые системы, отличные от FFS. Инсталлятор использует в качестве корневой файловой системы устройство памяти (MD). Если вы используете бездисковые системы (глава 20), вам потребуется поддержка NFS в качестве корневого раздела. Если вы устанавливаете FreeBSD на стандартный компьютер – с жестким диском и клавиатурой, то все эти параметры вам не потребуются.

```
options         NFSCLIENT       # Network Filesystem Client
options         NFSSERVER       # Network Filesystem Server
```

Эти два параметра обеспечивают поддержку сетевой файловой системы (Network File System, NFS) (глава 8). Параметр NFSCLIENT позволит монтировать разделы, которые обслуживаются другим компьютером в сети, а параметр NFSSERVER позволит обеспечить доступ к своим разделам для других компьютеров.

```
options         MSDOSFS         # MSDOS filesystem
```

```
options         CD9660          # ISO 9660 filesystem
options         PROCFS          # Process filesystem (requires PSEUDOFS)
options         PSEUDOFS        # Pseudo-filesystem framework
```

Эти параметры обеспечивают поддержку редко используемых файловых систем, таких как FAT, CD, файловой системы процессов и инфраструктуры псевдофайловой системы. Все эти файловые системы будут рассматриваться в главе 8, но все функциональные возможности, которые необходимы для работы с ними, доступны в виде модулей ядра, в тех редких случаях, когда в этом возникает необходимость.

```
options         COMPAT_43TTY    # BSD 4.3 TTY compat [KEEP THIS!]
options         COMPAT_FREEBSD4 # Compatible with FreeBSD4
options         COMPAT_FREEBSD5 # Compatible with FreeBSD5
options         COMPAT_FREEBSD6 # Compatible with FreeBSD6
```

Эти параметры совместимости позволяют запускать программное обеспечение, скомпилированное для устаревших версий FreeBSD, а также программное обеспечение, которое требует определенной версии ядра, что было необходимо в более ранних версиях FreeBSD. Если вы устанавливали систему с самого начала, тогда вам, возможно, не требуется совместимость с FreeBSD 4, 5 или 6, однако существует достаточно большое число программных продуктов, ориентированных на работу в среде 4.3 BSD. Оставьте параметр `COMPAT_43TTY`, иначе ваша система может оказаться *неработоспособной*.

```
options         SCSI_DELAY=5000 # Delay (in ms) before probing SCSI
```

Параметр `SCSI_DELAY` задает количество миллисекунд, в течение которых после обнаружения контроллеров SCSI система FreeBSD ожидает перед опрашиванием устройств SCSI, давая им время «раскрутиться» и идентифицировать себя на шине SCSI. Если в системе нет устройств SCSI, данную строку можно удалить. Если установлены древние аппаратные устройства SCSI, которые поднимаются медленно, как больной слон, то паузу можно увеличить до 15 000 (15 секунд).

```
options         SYSVSHM         # SYSV-style shared memory
options         SYSVMSG         # SYSV-style message queues
options         SYSVSEM         # SYSV-style semaphores
```

Эти параметры обеспечивают поддержку разделяемой памяти и межпроцессных взаимодействий в стиле System V. На эту возможность опираются многие программы управления базами данных.

```
options         AHC_REG_PRETTY_PRINT # Print register bitfields in debug
                                     # output. Adds ~128k to driver.
options         AHD_REG_PRETTY_PRINT # Print register bitfields in debug
                                     # output. Adds ~215k to driver.
```

Параметры, подобные этим, эффективны только в случае использования аппаратных устройств, на которые они ссылаются. В противном случае они могут быть удалены.

Более одного процессора

Следующие две записи активируют симметричную многопроцессорную обработку (Symmetric Multiprocessing, SMP) в ядрах, предназначенных для работы в архитектуре i386:

```
options         SMP             # Symmetric MultiProcessor Kernel
device          apic            # I/O APIC
```

Параметр `SMP` предписывает ядру производить планирование работы процессов на нескольких CPU, а параметр `apic` обеспечивает поддержку ввода/вывода для ядер SMP. Реализация i386 SMP в системе FreeBSD поддерживает только те многопроцессорные системы, которые отвечают спецификации SMP компании Intel и куда не входят 386 или 486 SMP-системы. (Для других платформ, таких как sparc64, имеются свои собственные стандарты, поэтому для них реализация SMP выполнена на очень высоком уровне.) Система FreeBSD поддерживает многопроцессорные системы, построенные только на базе процессоров Pentium и выше. В более старых версиях FreeBSD ядра с поддержкой SMP плохо или вообще не загружались на однопроцессорных системах – непроизводительные затраты на управление данными для нескольких процессоров создавали дополнительные задержки в системе. С течением времени эта проблема была ликвидирована и теперь FreeBSD распространяется с поддержкой SMP, включенной по умолчанию.

Драйверы устройств

Вслед за параметрами в файле конфигурации расположены записи с настройками драйверов устройств, которые сгруппированы вполне очевидным образом.

Первые записи – это шины, например `device pci` и `device eisa`. Убирать их следует лишь в том случае, если в системе действительно нет таких шин. Велико количество систем «без поддержки традиционных устройств» (legacy-free), в которых шина ISA скрыта где-то внутри.

Далее следует то, что большинством рассматриваются как *драйверы устройств,* – записи для накопителей на гибких магнитных дисках, контроллеров SCSI, контроллеров RAID и т. д. Если ваша цель уменьшить размер ядра, то данный раздел – хорошее место для решительного сокращения. Удалите все драйверы устройств, которые отсутствуют в компьютере. Здесь вы также найдете раздел драйверов устройств для клавиатур, видеокарт, портов PS/2 и т. д. Эти записи едва ли стоит удалять.

Раздел драйверов сетевых карт содержит довольно длинный список и по своему размеру напоминает разделы с драйверами устройств IDE и SCSI. Вы можете удалить из конфигурации любые сетевые карты, которые не используете.

Мы не будем перечислять здесь все драйверы устройств, поскольку в этом мало практической пользы, разве что ознакомиться с перечнем

оборудования, которое поддерживалось на момент, когда я писал этот раздел. Чтобы ознакомиться с перечнем оборудования, поддерживаемым вашей версией FreeBSD, загляните в примечания к выпуску (release notes).

Псевдоустройства

В нижней части файла настройки ядра GENERIC представлен список псевдоустройств. Как подсказывает их название, они всецело создаются программными методами. Ниже приводится описание некоторых из наиболее часто используемых псевдоустройств.

```
device          loop            # Network loopback
```

Это петлевой интерфейс, позволяющий системе взаимодействовать с самой собой через сетевые сокеты, с использованием сетевых протоколов. Сетевые соединения во всех подробностях будут рассматриваться в следующей главе. Вас может удивить, как много программ используют петлевой интерфейс; его следует оставить.

```
device          random          # Entropy device
```

Это устройство обеспечивает генерацию псевдослучайных чисел, необходимых в операциях шифрования и таких важных программах, как игры. FreeBSD обеспечивает поддержку разнообразных источников случайных чисел, и все они объединяются в виде устройств псевдослучайных чисел */dev/random* и */dev/urandom*.

```
device          ether           # Ethernet support
```

Ethernet обладает массой особенностей, характерных для устройств, и потому FreeBSD представляет его как устройство. Оставьте эту строку, если ваша цель не изучение системы.

```
device          sl              # Kernel SLIP
device          ppp             # Kernel PPP
```

Псевдоустройство `sl` предназначено для поддержки межсетевого протокола для последовательного канала (Serial Line Internet Protocol, SLIP), а устройство `ppp` поддерживает протокол «точка-точка» (Point to Point Protocol, PPP) на уровне ядра. Оба они давно устарели и не требуются для поддержки PPP в пространстве пользователя, если, конечно, у вас нет особых требований.

```
device          tun             # Packet tunnel.
```

Псевдоустройство `tun` – это *логический пакетный канал* (*logical packet tunnel*). Используется различными программами для обмена пакетами с ядром. Такое псевдоустройство необходимо для поддержки PPP средствами, не входящими в ядро (userland PPP), – обычными коммутируемыми соединениями.

```
device          pty             # Pseudo-ttys (telnet etc)
```

Псевдоустройство pty – это псевдотерминал. Когда вы подключаетесь к системе по протоколам telnet или SSH (глава 15), FreeBSD должна иметь возможность отслеживать терминальные сеансы, выводить символы на экран и читать ввод с клавиатуры. Система обслуживает удаленное соединение как обычный физический монитор и клавиатуру. Псевдотерминал – это псевдоустройство, напоминающее терминал, связанное с соединением.

```
device          md              # Memory "disks"
```

Псевдоустройство md позволяет хранить файлы в памяти. Это очень удобно для организации очень быстрых хранилищ временных данных, но об этом мы поговорим в главе 8. Для большинства (но не для всех) серверов Интернета диски в памяти – это просто непродуктивное растрачивание оперативной памяти. Кроме того, диски в памяти могут использоваться для монтирования и доступа к образам дисков. Если вы не используете диски в памяти, этот параметр можно удалить из ядра.

Отключаемые устройства

Наконец, после псевдоустройств в конфигурационном файле GENERIC можно найти поддержку FireWire и устройств USB. Эти функциональные возможности всегда оформляются в виде модулей, так как представляют устройства, которые могут присутствовать, а могут и не присутствовать в системе.

Сборка ядра

После прочтения предыдущего раздела вы должны быть в состоянии спроектировать минимальную конфигурацию ядра. Перед тем как добавлять в ядро поддержку каких-либо других возможностей, я рекомендую собрать и запустить описанное минимальное ядро. Это позволит понять, что действительно необходимо для надлежащей работы ядра и какая требуется дополнительная настройка.

Вам необходимо указать имя файла, который содержит конфигурацию вашего ядра, либо в командной строке, либо в файле */etc/make.conf*, либо в файле */etc/src.conf*, в виде переменной окружения KERNCONF.

```
# cd /usr/src
# make KERNCONF=MYKERNEL kernel
```

Процесс сборки начинается с запуска команды config(8), для поиска синтаксических ошибок в файле конфигурации. Если config(8) обнаружит какие-либо ошибки, она сообщит о них и прекратит работу. Некоторые сообщения об ошибках довольно очевидны. Например, можно случайно удалить поддержку файловой системы UNIX (UNIX File System, UFS), но включить поддержку инициализации UFS. Одна функция не может обойтись без другой, поэтому config точно скажет, в чем состоит ошибка. Другие сообщения об ошибках могут выглядеть странно

и непонятно, и тогда для выяснения причин может потребоваться длительное время, как, например, при получении следующего сообщения:

```
HUMVEE: unknown option "NET6"
```

NET6 – это же параметр, активирующий поддержку IPv6, разве не так? Нет, эту поддержку активирует параметр *I*NET6. Сообщения об ошибках в достаточной мере описывают проблему, особенно теперь, когда вы познакомились со всеми параметрами, поддерживаемыми ядром. Внимательно читайте сообщения об ошибках!

Если config(8) не нашла ошибок, остается только ждать. Процесс сборки ядра на 486 компьютере может занимать несколько часов, но меньше часа на современной и быстрой системе. При этом по экрану будут пробегать загадочные сообщения о ходе компиляции. По окончании сборки система переименует текущее ядро в */boot/kernel.old*, а новое ядро будет сохранено в виде файла */boot/kernel*. Когда все закончится, перезагрузите сервер и посмотрите на сообщения, появляющиеся во время загрузки.

```
Copyright (c) 1992-2006 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
        The Regents of the University of California. All rights reserved.
❶ FreeBSD 7.0-CURRENT #0: Thu May 11 01:16:19 EDT 2006
❷ mwlucas@humvee.blackhelicopters.org:/usr/src/sys/compile/HUMVEE
...
```

Во время загрузки FreeBSD точно сообщит, какое ядро было загружено ❶ и где это ядро было собрано ❷. Поздравляю, вы только что собрали ядро!

Выявление неполадок при создании ядра

Если собрать ядро не удалось, то для выявления неполадок прежде всего необходимо изучить последние строки, выданные при компиляции. Некоторые из этих строк могут показаться непонятным шифром, а другие достаточно очевидны. Важно помнить, что сообщения вида «Stop in *имя некоторого каталога*» абсолютно бесполезны – нужные нам сообщения об ошибках находятся перед ними. О том, как решать эти проблемы, мы говорили в главе 1: берете текст сообщения об ошибке и обращаетесь за помощью к поисковой системе. Как правило, ошибки времени компиляции вызваны ошибками в конфигурации.

К счастью, во FreeBSD перед установкой чего-либо надо полностью скомпилировать все ядро. Благодаря этому система защищена, а результаты неудачной компиляции по-прежнему находятся в каталоге сборки.

Загрузка с запасного ядра

Итак, что делать, если новое ядро не работает или работает неустойчиво? Возможно, при конфигурировании не был указан драйвер устройства либо удалено псевдоустройство `ppp` и не удается установить

коммутируемое соединение с Интернетом. Без паники! Старое ядро сохранено, верно? Хорошо. Вот что следует предпринять.

В главе 3 мы обсуждали механизм загрузки альтернативного ядра. Мы пройдем этот процесс еще раз, но, чтобы увидеть некоторые пояснения, касающиеся подробностей управления загрузчиком, вам может потребоваться вернуться к более раннему разделу. А теперь сосредоточим свое внимание на причинах, которые могут вынудить произвести загрузку с запасного ядра, и на том, как сделать это правильно.

Для начала необходимо решить, какое ядро загружать. Старое ядро должно находиться в каталоге */boot* – в этом разделе предполагается, что необходимо загрузить ядро */boot/kernel.good*. Запустите перезагрузку и прервите ее, чтобы получить доступ к командной строке загрузчика. Вспомните, мы говорили, что к моменту, когда FreeBSD предоставит доступ к командной строке загрузчика, ядро уже было загружено, поэтому в первую очередь необходимо выбросить плохое ядро за борт:

```
ok unload
```

Теперь можно загрузить требуемое ядро, а также модуль ACPI (если он используется) и любые другие модули ядра, которые обычно загружаются на этапе запуска системы:

```
ok load /boot/kernel.good/kernel
ok load /boot/kernel.good/acpi.ko
ok boot
```

После этого система будет загружена на старом ядре.

Включения, исключения и расширения ядра

Теперь, когда вы научились собирать ядро, можно попробовать включить свою фантазию и посмотреть, как можно использовать включения, ключевые слова с префиксом *no* и файл *NOTES*.

NOTES

Ядро FreeBSD включает в себя самые разнообразные функциональные возможности, которые не включены в GENERIC. Многие из этих возможностей предназначены для построения весьма специфических систем или сетей особого вида. Полный перечень особенностей, зависящих от аппаратного обеспечения, можно найти в файле *NOTES*, в каталоге с конфигурациями ядра для каждой из аппаратных архитектур, например */sys/amd64/conf/NOTES*. Перечень особенностей, поддерживаемых системой FreeBSD и не зависящих от аппаратного окружения, можно найти в файле */sys/conf/NOTES*. Если у вас имеется аппаратное устройство, которое не поддерживается ядром GENERIC, загляните в файл *NOTES*. Некоторые из перечисленных в этом файле особенностей имеют неясное назначение, но если у вас имеется определенное

устройство, то вы сможете разобраться, какая запись ему соответствует. Давайте взглянем на типичную запись из файла *NOTES*:

```
# CPU_SOEKRIS enables support www.soekris.com hardware.
#
...
options CPU_SOEKRIS
```

Soekris – это название производителя небольших систем для встраиваемых устройств. Об этих системах мы поговорим в главе 21 – они не настолько распространены, чтобы эта запись попала в GENERIC, но если у вас имеется одно из таких устройств, вы можете использовать параметр `CPU_SOEKRIS`, чтобы активировать особые функциональные возможности этого микропроцессора.

Если в файле *NOTES* перечислены все особенности для каждого из возможных устройств, почему бы не использовать его как основу для сборки собственного ядра? Во-первых, такое ядро расходовало бы больший объем памяти, чем ядро GENERIC. Несмотря на то, что даже небольшие современные компьютеры имеют достаточно памяти для запуска ядра GENERIC без каких-либо проблем, тем не менее, если ядро увеличится в размерах в десять раз без заметного увеличения функциональности, люди начнут недоумевать. Кроме того, многие параметры настройки являются взаимно исключающими. В файле *NOTES* можно отыскать параметры для микропроцессоров Athlon, Cyrix, IBM, параметры, предназначенные для учета некоторых особенностей различных версий плат, и т. д.

Включения и исключения

Механизм конфигурирования ядра FreeBSD обладает двумя интересными возможностями, которые способны упростить обслуживание ядра: ключевые слова с префиксом *no* и включения.

Возможность подключения позволяет добавить отдельный файл в конфигурацию ядра. Например, если имеется конфигурация ядра, которая описана как «GENERIC с парой дополнительных настроек», к ней можно было бы подключить конфигурацию GENERIC с помощью оператора `include`:

```
include GENERIC
```

Так, например, если необходимо собрать ядро, которое обладает функциональностью GENERIC, но кроме того поддерживает микропроцессор Soekris, можно было бы создать конфигурацию, составленную из следующих строк:

```
ident          MYKERNEL
include        GENERIC
options        CPU_SOEKRIS
```

Вы можете подумать, что такой подход дает немного больше, чем простое копирование и редактирование файла GENERIC, и будете правы.

Какие преимущества дает такой подход? Дело в том, что по мере обновления FreeBSD конфигурация GENERIC может изменяться. Например, конфигурация GENERIC в версии FreeBSD 7.1 несколько отличается от той, что используется в версии 7.0. При рассматриваемом подходе ваша конфигурация будет корректной в обеих версиях FreeBSD и в обеих по праву может называться «GENERIC плюс дополнительные параметры».

Такой подход отлично подходит для случаев включения элементов конфигурации, но не годится для случая исключения элементов из ядра. Например, любые микропроцессоры Soekris используются в небольших системах, не имеющих карты SCSI или RAID. В этом случае можно было бы дополнить свою конфигурацию объявлениями `nodevice`, исключающими поддержку всех этих устройств. Запись `nodevice` отменяет ранее объявленное включение устройства. Аналогичным образом ключевое слово `nooption` запрещает активизацию указанного параметра.

Наглядным примером использования такой методики может служить конфигурация ядра PAE. PAE – это конфигурация, которая используется в системах i386 с объемом памяти более 4 Гбайт, о чем будет говориться ниже, в этой же главе. Многие устаревшие устройства не в состоянии работать на компьютерах с таким объемом памяти, поэтому они должны быть исключены из конфигурации. Неважно, насколько велики изменения, произошедшие в вашей системе FreeBSD в результате обновления, вы всегда точно будете знать, что ядро PAE – это «GENERIC плюс возможности PAE минус все драйверы, которые не способны работать в конфигурации PAE».

Как с помощью параметров ядра исправлять проблемы

Некоторые параметры настройки ядра могут использоваться исключительно в случае появления определенных проблем. Например, пару лет назад у моего знакомого были веб-серверы, собранные на аппаратных средствах i386. Когда одному серверу пришлось обслуживать по нескольку сотен веб-страниц в секунду, на консоли стали появляться сообщения, подобные этому:

```
Jun 9 16:23:17 ralph/kernel: pmap_collect: collecting pv entries --
suggest increasing PMAP_SHPGPERPROC
```

Спустя пару часов после появления сообщения происходила авария системы. Совершенно очевидно, что мы с большой серьезностью отнеслись к этому случаю, и я произвел поиск в Интернете и просмотрел содержимое файла *NOTES*, где обнаружил следующую запись:

```
# Set the number of PV entries per process. Increasing this can
# stop panics related to heavy use of shared memory. However, that can
# (combined with large amounts of physical memory) cause panics at
# boot time due the kernel running out of VM space.
# (Устанавливает количество записей (pv-entry) таблицы страниц
# на один процесс. Увеличение этого числа может предотвратить неполадки,
```

```
# связанные с нехваткой разделяемой памяти. Однако при большой емкости
# физической памяти это может вызвать неполадки во время начальной загрузки
# из-за выхода за пределы пространства виртуальной памяти.)
#
# If you're tweaking this, you might also want to increase the sysctls
# "vm.v_free_min", "vm.v_free_reserved", and "vm.v_free_target".
# (При настройке этого параметра надо также увеличить значения:
# sysctls "vm.v_free_min", "vm.v_free_reserved" и "vm.v_free_target".)
#
# The value below is the one more than the default.
# (Значение, приведенное ниже, на единицу превышает значение по умолчанию.)
#
options PMAP_SHPGPERPROC=201
```

Прочитав это объяснение, мы приступили к решению проблемы. Прежде всего, сохранили старое ядро в */boot/kernel.pmap-crash*. Это было не самое лучшее ядро, но оно обеспечивало работоспособность системы хотя бы на несколько часов. Затем мы увеличили значение `PMAP_SHPGPERPROC` до 400, а емкость оперативной памяти увеличили до 192 Мбайт. (Да, эта недорогая система обслуживала несколько сотен веб-страниц в секунду, имея в распоряжении 64 Мбайт оперативной памяти, один диск IDE и CPU Celeron 433!) После сборки нового ядра проблема была устранена, а период работоспособного состояния сервера достиг нескольких месяцев. В версии FreeBSD 7 данный параметр ядра можно настроить через sysctl, прямо во время работы системы, однако в ядре существует еще множество параметров, которые можно изменить только таким способом.

Не будь возможности тонкой настройки ядра, единственным выходом стала бы покупка дополнительных аппаратных средств. Даже при том, что аппаратное обеспечение этой системы характеризуется низкой производительностью, нам удалось обеспечить устойчивость к высоким нагрузкам всего лишь за счет программной настройки и покупки небольшого количества памяти. В системе FreeBSD имеется большое число параметров, подобных этому, которые предназначены для особых ситуаций, и они скажут вам об этом, стоит только уделить им чуть больше внимания. (Замечу, что если вы просто пытаетесь направо и налево тратить свои деньги, я готов проявить милосердие к свободной наличности и, уверяю вас, смогу предоставить этим деньгам новый кров, где они будут высоко оценены.)

Распространение ядра

При наличии нескольких идентичных серверов не надо вручную собирать ядро на каждом из них; ядро, специально созданное для одного сервера, можно установить на остальных. В конце концов, ядро и модули к нему – это всего лишь файлы на диске.

Прежде всего, создайте и установите одно ядро и тщательно протестируйте его. Затем заархивируйте */boot/kernel* с помощью tar и скопи-

руйте тарбол на остальные серверы. Сохраните текущее ядро на каждом из них и разархивируйте тарбол, чтобы установить новый файл */boot/kernel*. Перезагрузитесь, и дело сделано!

Удаленное тестирование ядра

Вполне обычное дело, когда администрирование системы FreeBSD производится удаленно, например, когда сервер находится в другом, закрытом помещении. У вас может не быть последовательной консоли, вследствие чего становится невозможным получить доступ к командной строке загрузчика. Самое большее, на что можно надеяться в таких ситуациях, это то, что у вас будет некто, кто сможет подойти к компьютеру и нажать кнопку питания. Как в таких обстоятельствах можно протестировать новое ядро? Здесь хорошо бы иметь возможность попробовать загрузить новое ядро и если что-то пойдет не так – выполнить перезагрузку с рабочим ядром. И такая возможность есть, она называется однократная тестовая загрузка. Здесь вам пригодится nextboot(8).

nextboot(8) – это способ сказать: «Загрузи это ядро в следующий раз, но только один раз». Команда nextboot принимает единственный аргумент `-k`, параметром которого является имя каталога с тестовым ядром, вложенного в каталог */boot*. Например, я собрал и установил новое ядро, мне нужно однократно загрузить новое ядро, а если оно окажется неработоспособным, то на следующей перезагрузке должно быть загружено предыдущее ядро. Прежде всего, необходимо скопировать новое ядро в каталог, отличный от */boot/kernel*, а в */boot/kernel* должно находиться рабочее ядро.

```
# mv /boot/kernel /boot/kernel.test
# mkdir /boot/kernel
# cp /boot/kernel.good/* /boot/kernel/
```

Можно заранее предусмотреть установку ядра в данный каталог, установив значение переменной `INSTKERNAME` при сборке ядра. Дайте ядру имя во время установки ядра, и утилита make(1) автоматически поместит собранное ядро в нужное место:

```
# cd /usr/src
# make KERNCONF=TESTKERNEL INSTKERNAME=test kernel
```

Эта команда установит ядро как */boot/kernel.test*.

Теперь, когда имеется рабочее ядро, установленное как ядро по умолчанию, и тестовое ядро как */boot/kernel.test*, можно вызвать команду nextboot(8) и передать ей тестовое ядро в виде параметра ключа `-k`, указав имя каталога, вложенного в каталог */boot*:

```
# nextboot -k kernel.test
```

На следующей перезагрузке системы загрузчик запустит ядро из */boot/kernel.test* вместо */boot/kernel* и заодно сотрет файл с настройками, где

говорится о необходимости загрузить тестовое ядро. После этого при следующей перезагрузке будет загружено стандартное ядро. Вся хитрость состоит в том, чтобы рабочее ядро находилось в каталоге */boot/kernel*. Если новое ядро хорошо себя зарекомендовало, можно выполнить следующие команды:

```
# mv /boot/kernel /boot/kernel.previous
# mv /boot/kernel.test /boot/kernel
```

Вуаля! И тестовое ядро превратилось в рабочее.

> **Осторожно – nextboot!**
>
> В руководстве говорится, что nextboot(8) выполняет запись в корневую файловую систему до того, как она пройдет проверку целостности на этапе загрузки. Это означает, что в случае некорректного завершения системы до момента загрузки нового ядра возможно, что nextboot повредит файловую систему. В действительности утилита nextboot(8) не выделяет новое дисковое пространство, она просто редактирует файл, который уже существует в системе, поэтому риск повреждения файловой системы сводится к минимуму.

Составляющие ядра, о которых следует знать

Для этого раздела лучше подошел бы заголовок «Параметры ядра, о которые можно споткнуться, если не знать, для чего они нужны». Если кто-то сообщает вам, что у вас наблюдаются проблемы с ACPI, вы должны понимать, о чем идет речь. Когда система извергает страшное проклятие: «lock order reversals» (аннулирован запрос на блокировку), вы должны знать, что неприятности вам гарантированы. Основные источники проблем – это ACPI, PAE, SMP и аннулирование запроса на блокировку.

ACPI

Усовершенствованный интерфейс управления конфигурированием и энергопотреблением (Advanced Configuration and Power Interface, ACPI) отвечает за низкоуровневое конфигурирование аппаратного обеспечения, режима электропитания и т. п. Это следующий этап развития устаревших протоколов, таких как Plug-and-Play, системы настройки аппаратного обеспечения PCI BIOS и APM (Advanced Power Management – усовершенствованное управление питанием). Как и в других UNIX-подобных операционных системах, при реализации ACPI в системе FreeBSD была использована справочная информация, представленная компанией Intel.

Это просто протокол конфигурирования аппаратного обеспечения, какие тут могут быть ошибки? Скажем так, не все производители аппаратных компонентов реализуют поддержку ACPI в точном соответствии со спецификациями, скорее, они реализуют достаточный объем функций, чтобы обеспечить совместимость своей аппаратуры с операционными системами компании Microsoft. Это означает, что все остальные, которые используют другие операционные системы, должны искать способы обхода этих недостатков. FreeBSD также включает в себя различные обходные пути для таких неполных реализаций ACPI, но иногда до их выхода в свет проходит некоторое время.

Кроме того, в первых моделях аппаратных средств, использовавших ACPI, часто обнаруживались проблемы с самим ACPI. Как и в случае с любым другим сложным протоколом, первым разработчикам приходилось учиться применять этот протокол на практике. Если ваши аппаратные компоненты были собраны в этот период, вам может потребоваться вообще запретить использование ACPI, поместив строку `hint.acpi.0.disabled=1` в файл */boot/loader.conf*. Если у вас есть подозрения на проблемы с ACPI в вашем аппаратном окружении, можно попробовать запретить использование ACPI однократно, выбрав соответствующий пункт меню загрузки, как рассказывалось в главе 3.

PAE

В течение многих лет компьютеры с архитектурой i386 имели свойственное им ограничение – они могли иметь не более 4 Гбайт памяти. В то время компьютеры со 128 Мбайт памяти рассматривались как высокопроизводительные серверы, но к настоящему моменту развитие технологий сделало преодоление этого барьера не только мыслимым, но и реальным. Подобные ограничения существовали и раньше, начиная от предела в 640 Кбайт памяти для IBM PC и заканчивая пределом в 8 Гбайт для жестких дисков. Разработчики архитектуры должны выбрать некоторые пределы и попытаться установить их как можно выше, чтобы потом не волноваться о них в течение ближайших десяти лет или что-то около того. Механизм расширения пространства физических адресов (Physical Address Extensions, PAE) позволил поднять планку ограничения на объем памяти до 64 Гбайт, чего должно хватить на ближайшие несколько лет. К тому времени, когда такие объемы памяти станут обычным делом, большинство новых систем будут работать в 64-битовом режиме, что позволит поднять ограничения намного выше.

Но не все устройства совместимы с PAE, поэтому PAE не может включаться в ядро GENERIC. Конфигурацию ядра с поддержкой PAE можно найти в файле */sys/i386/conf/PAE*. Поддержка PAE необходима только для архитектуры i386 – amd64, sparc64 и другие новейшие архитектуры имеют другие ограничения на объем памяти, которые еще не были достигнуты.

Поддержку PAE нельзя запретить на этапе загрузки или во время работы системы – она либо присутствует в ядре, либо нет.

Симметричная многопроцессорная обработка

Симметричная многопроцессорная обработка (Symmetric Multiprocessing, SMP) означает наличие в системе нескольких микропроцессоров общего назначения. В теории операционная система равномерно распределяет рабочую нагрузку между несколькими процессорами. Но на самом деле все гораздо сложнее, чем кажется на первый взгляд, по причинам, которые будут рассматриваться в главе 12. Для обслуживания нескольких процессоров ядро должно включать `options SMP` и `device apic`.

В некоторых случаях в отдельных частях системы могут появляться проблемы, связанные с поддержкой SMP. До выяснения источника проблем может оказаться полезным временно отключить поддержку SMP. Если проблема присутствует при включенной поддержке SMP и пропадает при ее отключении, это поможет идентифицировать проблему. Запретить поддержку SMP можно на этапе загрузки, установив значение параметра `kern.smp.disabled` равным `1`. Аналогичным образом можно отключить поддержку APIC, побочным эффектом чего является отключение поддержки SMP. Запретить поддержку APIC можно, установив значение параметра `hint.apic.0.disabled="1"`.

Аннулирование запроса на блокировку

Ключевым элементом реализации поддержки SMP является блокировка ядра. Хотя вы, как пользователь, не должны беспокоиться по поводу блокировок ядра, тем не менее иногда при работе с определенными версиями FreeBSD (в частности, -current) в консоли могут появляться сообщения об аннулировании запросов на блокировку. Эти сообщения генерируются механизмом отладки ядра WITNESS (глава 13) и означают, что блокировки в ядре выполняются не так, как было задумано разработчиками. По большей части эти сообщения не представляют ничего опасного, но они выглядят устрашающе и действительно свидетельствуют о потенциальных проблемах.

Если вам встретятся подобные сообщения, лучше всего узнать, не сообщалось ли что-нибудь об этом конкретном сообщении ранее. Для этого можно выполнить поиск в Интернете по тексту сообщения. Можно также поискать на странице Lock Order Reversal веб-сайта FreeBSD (к моменту написания этих строк поддерживалась Бьерном Зибом (Bjoern Zeeb)) и узнать, упоминается ли данное сообщение здесь. Если поиск сообщения, встретившегося у вас, не дал результатов, вам следует сообщить об этом разработчикам FreeBSD, написав письмо в рассылку *FreeBSD-hackers@FreeBSD.org*.

Теперь у вас имеется объем знаний, достаточный для обслуживания ядра FreeBSD. Пойдем дальше и посмотрим, что можно сделать с сетью.

6

Работа в сети

FreeBSD известна своей сетевой производительностью. Сетевой протокол TCP/IP разрабатывался в BSD, и именно BSD использовала первую реализацию TCP/IP. Несмотря на жесткую конкуренцию сетевых протоколов в 80-х годах прошлого века, и благодаря удобству, гибкости и либеральным условиям лицензирования, стек протоколов TCP/IP стал фактическим стандартом.

Многие современные системные администраторы в целом знакомы с основами работы в сетях, но при этом имеют весьма смутное представление о внутренних взаимосвязях. Однако хороший системный администратор понимает, как работает сеть. Знания об адресации IP, смысле сетевой маски и отличии номеров портов от номеров протоколов – необходимый шаг к овладению профессией. В этой главе будут рассмотрены некоторые из этих тем. Для начала вы должны разобраться в том, что такое сетевые уровни.

Хотя эта глава и дает достаточно объемный обзор TCP/IP, в ней не рассматриваются многочисленные подробности. Если вам необходимы более глубокие знания о TCP/IP, обращайтесь к толстым книгам, посвященным этой теме. Для начала рекомендую прочитать замечательную книгу «The TCP/IP Guide» Чарльза М. Козиерока (Charles M. Kozierok) (No Starch Press, 2005).

Эта книга, в частности, охватывает TCP/IP версии 4. Его наследник, TCP/IP версии 6, к моменту написания этих строк еще не получил достаточно широкого распространения.

Сетевые уровни

Каждый уровень решает определенные задачи в сети и взаимодействует только с вышестоящим и нижестоящим уровнями. Изучающие TCP/IP часто смеются, когда слышат, что многоуровневая организация

упрощает сетевые взаимодействия, тем не менее это так. Прямо сейчас важно запомнить, что каждый уровень взаимодействует только с двумя соседними уровнями, расположенными выше и ниже.

В классической диаграмме сетевых протоколов OSI представлено семь уровней. Это полностью законченная схема, охватывающая все ситуации. Однако Интернет – это только одна из ситуаций, и эта книга не посвящена работе в сети в общем. Мы ограничимся обсуждением таких сетей TCP/IP, как Интернет и типичные корпоративные сети, поэтому будем рассматривать только четыре уровня.

Физический уровень

Самый нижний уровень – физический. Он образован сетевой картой и проводом, оптоволокном или радиоканалом. Кроме того, этот уровень включает в себя коммутатор (switch), сетевой концентратор (hub) или базовую станцию, провода, идущие к маршрутизатору, а также оптоволокно, соединяющее ваш офис с телефонной компанией. Коммутатор телефонной компании так же является частью физического уровня, как и оптоволокно и провода, соединяющие континенты. Любые каналы связи являются частью физического уровня. С этого момента мы будем называть физический уровень *проводом*, хотя это может быть любая другая среда распространения информации.

Этот уровень – самый простой для понимания. Если провод не поврежден и соответствует требованиям, задаваемым физическим протоколом, то дело в шляпе. Если нет, дело табак. Без физического уровня работа сети становится невозможной, точка, конец истории. Одна из функций маршрутизаторов Интернета состоит в том, чтобы преобразовывать пакеты при переходе с физического уровня одного вида на физический уровень другого вида, например из Ethernet в T1/E1. На физическом уровне отсутствует какая-либо логика, он не способен принимать решения, все, что передается через этот уровень, исходит от уровня физического протокола.

Уровень физического протокола

На уровне физического протокола дело становится интереснее. Физический протокол обеспечивает преобразование информации в фактические нули и единицы, которые передаются через физический уровень соответствующим образом. Например, Ethernet использует адреса протокола управления доступом к среде (Media Access Control, MAC) и протокол разрешения адресов (Address Resolution Protocol, ARP); для коммутируемого соединения используется протокол «точка-точка» (Point to Point Protocol, PPP). В дополнение к популярным физическим протоколам Ethernet и PPP, FreeBSD поддерживает и другие протоколы, включая асинхронную систему передачи (Asynchronous Transfer Mode, ATM), протокол высокого уровня управления каналом передачи данных (High Level Data Link Control, HDLC) и протокол межсетевого

обмена пакетами (Internetwork Packet Exchange, IPX), а также такие комбинации протоколов, как PPP через Ethernet (PPPoE), который используется в широкополосных сетях. Наличие поддержки всех этих протоколов в системе FreeBSD не означает, что она поддерживает *все* физические протоколы, которые когда-либо использовались. Если у вас имеются какие-то нетипичные требования, обращайтесь к документации по вашей версии FreeBSD, чтобы узнать, поддерживаются ли требуемые протоколы.

Некоторые физические протоколы имеют реализации, поддерживающие самые разные физические уровни. Например, протокол Ethernet может использоваться для передачи данных по биаксиальным, коаксиальным кабелям; по витой паре CAT3, CAT5, CAT6, CAT7; по оптоволоконным линиям; через радиоэфир и посредством почтовых голубей. С незначительными изменениями в драйверах устройств физический протокол может использоваться в любых типах линий передачи данных. Это один из примеров, как многоуровневая организация позволяет упростить реализацию сетевых взаимодействий. Далее мы подробно будем обсуждать протокол Ethernet как наиболее часто используемый для организации работы систем FreeBSD в сети. Разобравшись с протоколом Ethernet в системе FreeBSD, вы сможете управлять и другими протоколами, разумеется, как только вы начнете их понимать!

Физический протокол обменивается информацией с физическим уровнем и сетевым уровнем (уровнем логического протокола).

Сетевой уровень

«Сетевой уровень? Разве это не вся сеть целиком?»

Да, но сетевой уровень более специфичен. Он отвечает за обеспечение связи между узлами сети и отвечает на такие вопросы, как: «Где искать другие узлы сети?» и «Можно ли соединиться с каким-то конкретным узлом сети?». Этот логический протокол обеспечивает непротиворечивый интерфейс для программ, использующих сетевые соединения, независимо от типа физического уровня. Сетевой уровень Интернета основан на использовании протокола IP (Internet Protocol). В соответствии с протоколом *IP* каждому узлу сети присваивается уникальный адрес[1], который называется *IP-адресом*, по которому он может быть обнаружен любым другим узлом в сети.

Сетевой уровень взаимодействует с уровнем физического протокола, лежащим ниже, и транспортным уровнем, лежащим выше.

---

[1] Да, я помню о существовании механизма трансляции сетевых адресов (Network Address Translation, NAT), который позволяет использовать неуникальные IP-адреса. NAT – это путь, полный хлопот, спросите любого, кто использует NAT в достаточно крупных масштабах. Но даже в случае использования NAT в Интернете вы все равно имеете один или несколько уникальных адресов.

Транспортный уровень

Транспортный уровень имеет дело с реальными данными для реальных приложений и, возможно даже, для реальных людей. На транспортном уровне используются три основных протокола – ICMP, TCP и UDP.

Протокол управляющих сообщений Интернета (*Internet Control Message Protocol, ICMP*) управляет передачей сообщений, обеспечивающих взаимодействие между узлами сети с присвоенными IP-адресами. Если протокол IP – это способ добраться до требуемого адреса, то ICMP – это своего рода светофор и дорожный знак, обозначающий выезд на магистраль. В большинстве случаев протокол ICMP работает совершенно незаметно, и вам никогда не придется вспоминать о его существовании.

Другие, широко известные транспортные протоколы, – это протокол пользовательских дейтаграмм (User Datagram Protocol, UDP) и протокол управления передачей данных (Transmission Control Protocol, TCP). Насколько широко они распространены? Скажем так, комплекс протоколов Интернета вообще называется *TCP*/IP. Эти протоколы предоставляют такие возможности, как мультиплексирование по номеру порта и передачу пользовательских данных. *UDP* – это исключительно транспортный протокол, предоставляющий минимальный объем возможностей, необходимых для передачи данных через сеть. TCP обеспечивает более широкие возможности, такие как определение заторов и проверку целостности.

Помимо этих трех, существует еще множество протоколов, работающих поверх IP. Полный перечень транспортных протоколов, основанных на механизме IP, можно найти в файле */etc/protocols*. Здесь не удастся отыскать протоколы, не использующие IP, такие как LAT компании Digital, но здесь перечислено огромное число протоколов, которые могут вам встретиться в реальном мире. Например, здесь, как показано ниже, присутствуют наиболее типичные для Интернета протоколы IP и ICMP:

```
❶ip    ❷0  ❸IP    ❹# Internet protocol, pseudo protocol number
icmp    1   ICMP   # Internet control message protocol
```

Каждая строка в */etc/protocols* состоит из трех основных полей: неофициальное название ❶, номер протокола ❷ и псевдонимы ❸. Номер протокола используется в сетевых запросах для идентификации трафика. Вы увидите его, если воспользуетесь программой перехвата пакетов (sniffer) или по каким-либо причинам углубитесь в изучение своей сети. Как видно из фрагмента выше, протокол IP имеет номер 0, а протокол ICMP имеет номер 1 – если не это основа всего сущего, сложно представить, что могло бы быть ею! Протокол TCP имеет номер 6, а протокол UDP – 17. Кроме того, здесь же можно видеть комментарии ❹, в которых приводится краткое описание каждого протокола.

Транспортный уровень взаимодействует с сетевым уровнем, лежащим ниже, и с приложениями – выше.

Приложения

Приложения также являются частью сети. Приложения посылают запросы на открытие сетевых соединений, передают данные в сеть, получают данные из сети и обрабатывают эти данные. Веб-броузеры, клиенты электронной почты, серверы JSP и т. п. – все это сетевые приложения. Приложения взаимодействуют только с сетевым уровнем и с пользователем. Проблемы на уровне пользователя и выше далеко выходят за рамки этой книги.[1]

Сеть на практике

Итак, вы имеете представление, как все работает в комплексе, и готовы двигаться дальше, правильно? Нет, неправильно. Давайте посмотрим, как этот комплекс работает в реальных условиях. Некоторые из последующих объяснений будут касаться материала, который будет рассматриваться в следующих главах, но раз уж вы взялись за чтение этой книги, значит, вы имеете представление о сетях, достаточное, чтобы двигаться дальше. Если что-то будет вам непонятно, перечитайте этот раздел еще раз, после прочтения этой главы. (Просто купите еще один экземпляр этой книги, вырежьте страницы с данным разделом и вклейте их в конце этой главы.)

Предположим, что пользователь, имеющий подключение к Интернету, хочет попасть на веб-сайт Yahoo! Для этого он запускает свой веб-броузер и набирает адрес URL.

Веб-броузер должен знать, как взаимодействовать с транспортным уровнем, лежащим ниже. После преобразования запроса пользователя в соответствующую форму броузер просит транспортный уровень установить соединение с указанным IP-адресом, с номером порта 80. (Поборники точности могут заметить, что я пропустил этап определения IP-адреса по доменному имени, но это не такое большое отступление, которое только усложнило бы пример.)

Транспортный уровень анализирует запрос, поступивший от броузера. Поскольку приложение обратилось с запросом на установление соединения TCP, транспортный уровень выделяет для этого соответствующие системные ресурсы. Запрос разбивается на части допустимого размера, которые передаются ниже, сетевому уровню.

Сетевой уровень не думает о реальном запросе. Ему вручили блок данных, который он должен доставить по указанному адресу через Интер-

[1] Если мои исследования в области переформатирования и переустановки пользователей принесут плоды, я обязательно опубликую полученные результаты.

нет. Как почтальон, доставляющий письма, ничего не зная об их содержимом, сетевой уровень просто упаковывает данные TCP в «конверт» с соответствующим адресом. Получившийся блок данных называется *пакетом*. Далее сетевой уровень передает пакеты на уровень физического протокола.

Уровень физического протокола совершенно не интересует содержимое пакета. Его не интересуют ни IP-адрес, ни маршрут движения пакета. Ему вручили блок нулей и единиц, и он должен выполнить задание по доставке их через сеть. Единственное, о чем он знает, – это как выполнить передачу. Уровень физического протокола может добавлять в пакет дополнительную информацию, которая будет использоваться на физическом уровне, создавая тем самым *кадр* (*frame*). Наконец, все это передается с уровня физического протокола в провода, в эфир или другую среду передачи информации.

Физический уровень полностью лишен интеллекта. Уровень физического протокола передает ему последовательность нулей и единиц, а он передает их другому физическому устройству. Он понятия не имеет, какой протокол используется или как вывести эти нули и единицы через коммутатор, концентратор или повторитель, но предполагает, что один из хостов является маршрутизатором.

Когда маршрутизатор принимает блок нулей и единиц, он передает их на уровень выше – на уровень физического протокола. На этом уровне отбрасывается информация о кадре и полученный пакет вручается сетевому уровню внутри маршрутизатора. Сетевой уровень маршрутизатора анализирует пакет и на основе имеющейся таблицы маршрутизации решает, что с ним делать дальше. После этого пакет вручается соответствующему уровню физического протокола. Это может быть другой интерфейс Ethernet или интерфейс PPP, ведущий в линию T1.

Каждый внутри другого?

Да, именно так, оригинальный веб-запрос был инкапсулирован в протокол TCP. Этот запрос в свою очередь был инкапсулирован на транспортном уровне в протокол IP и затем в физический протокол. Все эти заголовки вставлялись в начало и в конец первоначального запроса. Приходилось ли вам когда-нибудь видеть картинку, где маленькую рыбку глотает рыба побольше, которую в свою очередь пытается проглотить рыба еще большего размера и так далее? Эта картинка очень точно отражает то, что происходит в нашем случае. Или, если хотите, кадр можно сравнить с самой большой коробкой, в которую вложена серия коробок разного размера и в самой внутренней коробке спрятан подарок. Развернув один протокол, вы обнаружите другой.

На пути данных тип физического носителя может меняться. Например, линия T1 может перейти в оптоволоконную линию DS3, которая в свою очередь стыкуется с линией OC192, протянутой по всей стране. Благодаря многоуровневой организации и абстракции ни ваш компьютер, ни ваш пользователь не обязаны что-либо знать о таких вещах.

Достигнув места назначения – компьютера на другой стороне соединения, кадр начинает обратный путь по стеку протоколов. Физический уровень принимает нули и единицы и передает их на уровень физического протокола. Уровень физического протокола очищает кадры от заголовков Ethernet и передает получившиеся пакеты транспортному уровню. На транспортном уровне пакеты собираются в поток данных, который затем направляется приложению, в данном случае – веб-серверу. Приложение обрабатывает запрос и возвращает ответ, который опускается вниз по стеку протоколов и отправляется в сеть, где они, в случае необходимости, перепрыгивают через границы различных физических протоколов. Такой процесс подразумевает огромный объем работы, часто только лишь для того, чтобы вы могли получить сообщение об ошибке «404 Page Not Found» (запрошенная страница не найдена).

Этот пример наглядно демонстрирует, почему многоуровневая организация имеет такое большое значение. Каждый уровень знает лишь самое необходимое об уровнях, находящихся выше и ниже, поэтому при желании можно менять целые уровни. Когда создается новый физический протокол, другим уровням не о чем беспокоиться; сетевой протокол просто передает правильно сформированный запрос на уровень физического протокола и оставляет за ним всю внутреннюю работу. При появлении сетевой карты нового типа надо лишь написать драйвер для физического протокола; стек сетевых протоколов, включая и приложение, здесь роли не играют. Представьте себе драйвер устройства, который было бы необходимо устанавливать в веб-броузер, в клиент электронной почты и в любые другие приложения на вашем компьютере, включая ваши собственные. Это быстро привело бы вас к разочарованию и стало бы похоже на прыжок с парашютом с наковальней на шее.

Двоичные и шестнадцатеричные значения

Будучи системным администратором, вы частенько будете встречать такие термины, как *48-битовый адрес* или *18-битовая сетевая маска*. Я часто встречал системных администраторов, которые только кивают и улыбаются, когда слышат нечто подобное, и при этом думают: «Да неважно, ты прямо скажи, что я должен делать». К сожалению, математика – это неотъемлемая часть работы, и вы *обязаны* знать двоичное исчисление. Понимание этой области является одним из признаков, которые отличают любителей от профессионалов. Вам не нужно читать эту книгу, если вы хотите остаться любителем.

Возможно, вы уже бормочете себе под нос: «Но я знаю это!» Тогда просто пропустите этот раздел. Но не обманывайте себя, если вы этого еще не знаете.

Возможно, вы уже знаете, что компьютер интерпретирует данные как нули и единицы и что одна единица или один ноль составляют один бит. Когда протокол задает биты, он указывает это число согласно представлениям компьютера. 32-битовое число имеет 32 цифры, каждая из которых может быть представлена нулем или единицей. Возможно, вы уже имеете представление о двоичных операциях, или операциях с числами *по основанию 2*, которые наверняка изучались вами в средней школе и быстро забылись. Двоичная система – это просто другой способ представления чисел, с которыми люди работают каждый день.

Десятичная математика (числа с основанием 10) ежедневно используется нами при покупке пиццы или для подсчета баланса чековой книжки. В ней задействованы цифры от 0 до 9. Чтобы получить число, превышающее самую большую цифру, слева к этой цифре добавляется другая, а текущая становится равной нулю. Это правило «переноса единицы», которое вы изучили много лет назад и сейчас используете неосознанно. В двоичной математике есть только цифры 0 и 1. Чтобы получить число, превышающее самую большую цифру, слева к этой цифре добавляется другая, а текущая становится равной 0. Та же самая десятичная математика, только с восемью пропавшими пальцами. В качестве примера в табл. 6.1 показаны первые несколько десятичных чисел, преобразованные в двоичные значения.

Таблица 6.1. Десятичные и двоичные числа

Десятичные	Двоичные
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000

Число из 32 бит, например IP-адрес, представляет собой строку из 32 нулей и единиц. MAC-адреса, используемые протоколом Ethernet, представляют собой 48-битовые числа и состоят из 48 нулей и единиц.

Ради интереса могу заметить, что в UNIX в некоторых случаях используются шестнадцатеричные числа (например, MAC-адреса и сетевые маски). Шестнадцатеричные числа имеют длину 4 бита. Двоичное

число 1111, в котором все четыре бита представлены единицами, эквивалентно десятичному числу 15, а это означает, что в шестнадцатеричной системе используются цифры от 0 до 15. Сейчас некоторые из вас, глядя на двузначное число 15, которое должно представлять единственную цифру, думают – какую травку я курю и нельзя ли организовать поставку такой замечательной травки себе. Для представления чисел от 10 до 15 в шестнадцатеричной системе счисления используются символы от A до F. Когда счет доходит до самой большой цифры и нужно добавить еще единицу, текущая цифра устанавливается равной нулю, а слева добавляется еще одна цифра. Например, последовательность чисел от 1 до 17 в шестнадцатеричной системе выглядит следующим образом: «1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11». Снимайте обувь и считайте на пальцах, пока не поймете.

Шестнадцатеричные числа обычно помечаются префиксом *0x*. Например, число *0x12* – это шестнадцатеричный эквивалент десятичного числа 18, тогда как 18 – это просто число 18. Если шестнадцатеричное число не помечено префиксом 0x, значит оно находится на месте, где допустимы только шестнадцатеричные числа, например в MAC-адресе. Наличие символов от A до F однозначно говорит о том, что число шестнадцатеричное, но это не самый надежный признак, потому что шестнадцатеричные числа могут не содержать шестнадцатеричных цифр, так же как многие десятичные числа могут состоять только из двоичных цифр.

Когда вы работаете с шестнадцатеричными, десятичными и двоичными числами, было бы желательно пользоваться научным калькулятором. Современные средние калькуляторы имеют функции преобразования чисел между этими тремя системами счисления, как и большинство программных калькуляторов.

Биты по байтам

Компьютеры преимущественно работают с байтами, где каждый символ представлен 8-битовым числом. Единственное исключение – сетевой стек, где все сущее представлено отдельными битами. Так, например, файл размером 5 мегабайтов может находиться в компьютере, подключенном к сети со скоростью передачи данных 10 мегабитов (*в секунду – прим. научн. ред*). Не путайте их!

TCP/IP в деталях

Теперь, когда у вас имеется некоторое представление о том, как работает сеть, я предлагаю подробнее рассмотреть реальный сетевой протокол. Доминирующим протоколом в Интернете является TCP/IP, или

Transmission Control Protocol over Internet Protocol (протокол управления передачей данных, реализованный поверх межсетевого протокола). TCP – это транспортный протокол, тогда как IP – это сетевой протокол, но они настолько переплелись между собой, что обычно рассматриваются как единое целое. Мы начнем рассмотрение темы с протокола IP, а затем перейдем к TCP и UDP.

IP-адреса и сетевые маски

IP-адрес – это уникальное 32-битовое число, присвоенное конкретному узлу сети. Некоторые адреса остаются более или менее постоянными, как, например, адреса серверов в Интернете. Другие изменяются в зависимости от потребностей сети, как, например, IP-адреса, которые выдаются клиентам, подключающимся к Интернету по коммутируемым линиям связи. Отдельные машины в общедоступных сетях получают IP-адреса из непрерывного диапазона адресов.

Обычно IP-адреса рассматриваются не как 32-битовое число, а делятся на четыре группы по 8 бит в каждой, и записываются в виде десятичных чисел, разделенных точками. Хотя запись 192.168.1.1 – это то же самое, что и 11000000.10101000.00000001.00000001, или число 11000000101010000000000100000001, тем не менее запись в виде четырех чисел воспринимать проще.

При подключении компании к Интернету провайдер выделяет ей блок IP-адресов. Как правило, этот блок невелик, скажем, 16 или 32 IP-адреса. Если речь идет о небольшой группе идентичных серверов, компания получит лишь несколько IP-адресов.

Сетевая маска определяет размер блока IP-адресов в локальной сети. Размер блока IP-адресов определяется сетевой маской, или, иначе говоря, сетевая маска определяет количество IP-адресов, имеющихся в вашем распоряжении. Если вы имеете давнее знакомство с сетями, вам наверняка приходилось встречаться с сетевой маской 255.255.255.0 и вы знаете, что она описывает блок из 256 IP-адресов. Кроме того, возможно, вы даже знаете, что неверная сетевая маска будет препятствовать нормальной работе системы в сети. Сейчас такие простые сетевые маски встречаются все реже и реже. Чтобы понять, в чем тут дело, необходимо погрузиться в историю IP-адресов. Много лет назад IP-адреса выделялись блоками трех размеров, соответствующих классу A, классу B и классу C. Со временем эта терминология устарела, но мы возьмем ее за основу.

Класс A был очень простым: первое из четырех чисел IP-адресов было фиксированным. Агентство по выпуску IP-адресов в обращение (в прошлом – InterNIC) могло выделить блок адресов класса A, скажем, 10.0.0.0. Оставшиеся три числа владелец мог назначать по своему усмотрению, при условии, что IP-адреса будут начинаться с 10. Например, адреса с 10.1.0.0 по 10.1.1.255 можно было выделить для информационного центра, адреса с 10.1.2.0 по 10.1.7.255 – для офиса в Детройте

и т. д. Блоки адресов класса A получали только очень большие компании, такие как Ford и Xerox, а также влиятельные академические организации, связанные с компьютерными технологиями, такие как Массачусетский технологический институт (MIT). Просмотрев список первоначальных владельцев класса A, можно увидеть, что в основном это учебные заведения, имевшие большое влияние в 80-х годах прошлого века.

В блоках класса B фиксированными были первые два из четырех чисел, составляющих IP-адрес. Блок адресов класса B мог выглядеть, например, как 172.16.0.0. Каждый IP-адрес, использовавшийся во внутренней сети, начинался с 172.16, а по своему усмотрению можно было назначать последние два числа. Многие компании среднего размера получили блок адресов класса B.

Подобным образом в блоке класса C фиксированы первые три числа адреса. Обычно такие блоки получали маленькие компании. Провайдер услуг Интернета выделял номер, подобный 198.22.63.0, и предоставлял владельцу блока возможность назначать последнее число каждого адреса.

Такая схема приводила к растрачиванию IP-адресов. Многим мелким фирмам не нужны были 256 IP-адресов, а многие компании среднего размера занимали больше 256, но меньше 65 000 адресов в блоке класса B. И почти никому не требовались все 16 миллионов адресов из блока класса A. Однако было принято такое решение, и до бума Интернета такое положение всех устраивало. Тогда, в 80-х годах, дети хотели просто поиграть с компьютером, теперь же они мечтают о своем собственном сайте электронной коммерции. Это привело к увеличению потребности в IP-адресах.

Сегодня IP-адреса выделяются вместе с *префиксной длиной* (*prefix length*), значение которой отделяется от IP-адреса символом *слэша* (*slash*). Блок IP-адресов может выглядеть так: 192.168.1.128/25. Подобная запись может быть не очень понятной, но делает возможным более тонкое деление классов адресов. Известно, что каждое из четырех чисел в IP-адресе состоит из 8 бит. При использовании классов «фиксированным» считается определенное количество бит – их нельзя изменить в локальной сети. Адрес класса A имеет 8 фиксированных бит, класса B – 16 бит и класса C – 24 бита.

В приведенных примерах IP-адреса того или иного класса не были представлены в двоичной форме, и я не буду заставлять вас выполнять это преобразование. Однако под IP-адресом следует понимать строку из двоичных чисел. В адресах локальной сети можно изменить биты в правой части каждого адреса, но не биты в левой части. Единственный вопрос, который осталось выяснить: «Где проходит граница, которая отделяет правую и левую части?» Нет никаких причин, по которым длина каждой части адреса должна быть кратной 8. Префиксная длина – это просто некоторое количество фиксированных битов. За-

пись /25 означает, что фиксированными являются 25 бит, то есть на один бит больше, чем в адресе класса C. В этом случае в адресе можно изменять последние 7 бит. В следующей записи фиксированные биты представлены единицами, а изменяемые – нулями:

```
11111111.11111111.11111111.10000000
```

11111111 – это число 255, а 10000000 – 128. Таким образом, данная сетевая маска приобретает вид 255.255.255.128. Все очень просто, если мыслить двоичными категориями. С двоичными числами вовсе не обязательно работать каждый день, но тем, кто не понимает базовые концепции двоичной математики, преобразование чисел из двоичной формы в десятичную кажется совершенно непонятным. В ходе практической деятельности можно научиться видеть десятичные числа в логичной двоичной записи.

Что это означает на практике? Прежде всего, блоки IP-адресов выделяются в количестве, кратном 2. Если изменяемыми являются 4 бита, то возможны 16 адресов (2×2×2×2 = 16), а если 8 бит, то (2^8) 256 IP-адресов. Если кто-либо говорит, что доступно 19 IP-адресов, то либо речь идет о разделяемой сети Ethernet, либо высказанное утверждение ошибочно.

Нередко можно увидеть IP-адрес хоста вместе с его сетевой маской, например 192.168.3.4/26. Такая форма записи содержит всю информацию, необходимую для подключения компьютера к локальной сети. (Поиск шлюза по умолчанию – это уже другая проблема, но, как правило, адрес шлюза находится либо в самом верху, либо в самом низу диапазона адресов.)

Вычисление сетевых масок в десятичном виде

Преобразование из десятичной формы в двоичную и обратно не всегда доставляет удовольствие. Это не только неудобно, это еще увеличивает вероятность допустить ошибку. Следующая хитрость позволит вам вычислять сетевые маски, оставаясь в рамках десятичной системы счисления.

Прежде всего нужно выяснить, сколько реальных IP-адресов есть в наличии. Это число будет кратно 2. Почти всегда количество адресов меньше 256. Из числа 256 нужно вычесть количество адресов, имеющихся в наличии, и получим последнее число сетевой маски. При этом все еще необходимо уметь определять размеры сети. Если предположить, что IP-адрес имеет вид 192.168.1.100/26, вы должны понимать, что запись /26 – это 26 фиксированных битов, или 64 IP-адреса. Взгляните на последнее число в IP-адресе – 100. Конечно, оно находится не в интервале между 0 и 63, а в интервале между 64 и 127. Другие хосты этого блока имеют IP-адреса от 192.168.1.64 до 192.168.1.127, а сетевая маска имеет вид 255.255.255.192 (256 – 64 = 192).

В этом месте я должен заметить, что сетевые маски часто записываются в шестнадцатеричном виде. У вас может возникнуть желание бросить

все, как безнадежное дело, но упростить жизнь вам поможет табл. 6.2, в которой приводятся сетевые маски и количество доступных IP-адресов для сетей /24 и меньше.

Таблица 6.1. Сетевые маски и IP-адреса

Префикс	Двоичная маска	Десятичная маска	Шестнадцатеричная маска	Доступные IP-адреса
/24	00000000	255.255.255.0	0xffffff00	256
/25	10000000	255.255.255.128	0xffffff80	128
/26	11000000	255.255.255.192	0xffffffc0	64
/27	11100000	255.255.255.224	0xffffffe0	32
/28	11110000	255.255.255.240	0xfffffff0	16
/29	11111000	255.255.255.248	0xfffffff8	8
/30	11111100	255.255.255.252	0xfffffffc	4
/31	11111110	255.255.255.254	0xfffffffe	2
/32	11111111	255.255.255.255	0xffffffff	1

Неиспользуемые IP-адреса

Теперь вы понимаете, что объединяет сетевые маски и IP-адреса и что, например, запись /28 подразумевает 16 IP-адресов. К сожалению, не все адреса в блоке можно использовать. Первый IP-адрес блока – это *номер сети* (*network number*). Он предназначен для внутреннего управления сетевыми ресурсами.

В любой группе IP-адресов последний номер является *широковещательным адресом* (*broadcast address*). Согласно спецификации IP, каждая машина в сети должна откликаться на запрос, посылаемый по этому адресу. Такая возможность позволяет проверять доступность адресатов и выяснять, какие IP-адреса используются в сети. Например, в типичной сети /24 широковещательный адрес будет иметь вид: x.y.z.255. В конце 90-х годов отправка пакетов по широковещательному адресу применялась при организации атак на сети. Сейчас эта возможность отключена в большинстве операционных систем и сетевых устройств.[1] В системах BSD поддержку широковещательной адресации можно включить с помощью установки параметра `net.inet.icmp.bmcastecho` в 1.

В любом случае интерфейсу нельзя назначить первый и последний IP-адреса сети, не вызвав появление проблем. Разные системы по-разному реагируют на эти проблемы. Но можно попробовать, причем желательно после окончания рабочего дня, иначе вам придется рассказывать эту историю уже на другой работе.

[1] За исключением многих принтеров, по определенным причинам. Размещайте свои принтеры за межсетевыми экранами!

Присвоение IP-адресов

Можно было бы подумать, что каждый компьютер в сети имеет IP-адрес, но это не всегда верно. IP-адреса присваиваются каждому сетевому *интерфейсу*. Большинство компьютеров имеют единственный сетевой интерфейс, поэтому для них эта разница несущественна. Однако, если в компьютере имеется несколько сетевых карт, то каждой из них присваивается свой IP-адрес. С другой стороны, можно присвоить несколько IP-адресов одному сетевому интерфейсу, используя псевдонимы. Напротив, в некоторых конфигурациях можно связать несколько сетевых карт с единственным сетевым интерфейсом, дав компьютеру единственный виртуальный интерфейс, несмотря на имеющееся количество сетевых карт. Хотя эти различия являются незначительными, о них следует помнить при выяснении возникающих проблем.

ICMP

Протокол управляющих сообщений Интернета (Internet Control Message Protocol, ICMP) – это стандартный способ передачи информации о маршрутах и сообщений о доступности сети. Такие инструменты, как ping(8) и traceroute(8) используют ICMP для сбора необходимых сведений. Протокол ICMP необходим для обеспечения надлежащей производительности сети, но некоторые виды ICMP-сообщений могут использоваться злоумышленниками для получения дополнительных сведений о сети. Если вам необходимо блокировать трафик ICMP по причинам, связанным с обеспечением безопасности, вы должны делать это выборочно.

UDP

Протокол пользовательских дейтаграмм (User Datagram Protocol, UDP) – это основной транспортный протокол передачи данных, работающий поверх протокола IP. В нем не предусмотрена возможность обработки ошибок, он обладает минимальными возможностями проверки целостности и не имеет никакой защиты от потери данных. Несмотря на эти недостатки, UDP является неплохим выбором для передачи некоторых видов данных, и многие службы Интернета используют его.

Когда хост передает данные по UDP, он не знает, доставлены ли они до места назначения. Когда хост получает данные по UDP, он просто прослушивает сеть и принимает все, что из нее приходит. Программа, принимающая данные по UDP, не может проверить, откуда пришли эти данные. Хотя пакеты UDP включают в себя адрес отправителя, его легко подделать. По этой причине UDP называют протоколом *без установления соединения* (*connectionless*) или протоколом *без установления состояния* (*stateless*).

Почему тогда используется UDP, если он обладает такими недостатками? Приложения, использующие UDP, часто сами реализуют методы обработки ошибок, которые могут не совпадать с методами, предостав-

ляемыми по умолчанию такими протоколами, как TCP. Например, простые клиентские запросы к DNS должны иметь тайм-аут не более нескольких секунд, в противном случае это будет вызывать раздражение у пользователя. Тайм-аут для соединений TCP составляет две минуты. Поскольку реакция на неудачные запросы к DNS должна следовать как можно быстрее, то запросы к DNS отправляются по протоколу UDP. В случаях, когда сервер DNS должен отправлять большие объемы данных (например, передача информации о зонах), он переходит на использование протокола TCP. Для передачи потоковых данных, как в случае с видеоконференциями, также используется протокол UDP. Если бы потеря нескольких пикселей при передаче картинки видеоконференции в реальном времени вызывала повторную передачу данных, это приводило бы лишь к перегрузке сети. Нельзя вернуться назад во времени и восполнить недостающие участки картинки! Аналогичными причинами объясняется применение протокола UDP практически всеми использующими его сетевыми приложениями.

UDP – это также протокол обмена *дейтаграммами*, в том смысле, что каждый передаваемый блок данных является самодостаточным и законченным и воспринимается как отдельный блок. При этом сами приложения, использующие UDP, чаще всего не считают одиночные пакеты UDP завершенным запросом, каковыми они воспринимаются сетью. Протокол TCP полностью отличается от UDP.

TCP

Протокол управления передачей данных (Transmission Control Protocol, TCP) обладает такими замечательными особенностями, как коррекция ошибок и восстановление данных. Принимающая сторона должна подтвердить получение каждого пакета, в противном случае отправитель будет повторять передачу неподтвержденных пакетов. Приложения, использующие TCP, вправе ожидать надежной передачи данных. В отличие от UDP, протокол TCP называют протоколом *с установлением соединения* (*connected*).

Кроме того, TCP является *потоковым* протоколом, то есть при его использовании единственный запрос может быть разбит на несколько сетевых пакетов. Несмотря на то, что отправитель может посылать пакеты по порядку, один за другим, принимающая сторона может получать их в ином порядке или вообще в фрагментированном виде. Поэтому принимающая сторона должна следить за тем, какие части она получает, и выполнять их сборку для организации нормальных сетевых взаимодействий.

Два хоста, использующие TCP для обмена данными, должны организовать между собой канал, по которому будут передаваться данные. Первый хост запрашивает открытие соединения, второй хост отвечает на этот запрос, и только после этого первый хост начинает передачу данных. Этот процесс называется *тройным рукопожатием* (*three-way*

handshake). Подробности на данном этапе нас не интересуют, нам достаточно знать, что это происходит. Аналогичным образом по окончании передачи системы должны выполнить некоторые действия, чтобы закрыть соединение.

Протокол TCP обычно используется такими программами, как программы электронной почты, FTP-клиенты и веб-броузеры, по этой причине для протокола были выбраны наиболее универсальные интервалы тайм-аутов и набор функциональных возможностей.

Транспортные протоколы следующего поколения

Одной интересной особенностью FreeBSD 7.0 является поддержка протокола управления передачей потоков данных (Stream Control Transmission Protocol, SCTP). Это транспортный протокол следующего поколения, разработанный для передачи сложных потоков данных. Реализация протокола во FreeBSD спонсировалась компанией Cisco Systems, которая, по всей видимости, рассматривала свободно распространяемую систему FreeBSD как один из лучших способов вывести данный протокол в мир.

Как протоколы взаимодействуют между собой

Стек сетевых протоколов можно сравнить с семьей, сидящей за обеденным столом. Родственники передают блюда друг другу. Физический протокол (ARP, в случае использования Ethernet) позволяет увидеть всех сидящих за этим столом. Протокол IP выделяет каждому свое место за столом, кроме трех юных племянников, присевших на скамейку NAT. Протокол ICMP предоставляет информацию о маршрутизации, например: «Самый короткий путь к гороху – это попросить дядюшку Криса передать его вам». Протокол TCP вступает в игру, когда вы передаете кому-то блюдо, и он должен сказать: «Спасибо» прежде, чем вы отпустите тарелку. Наконец, протокол UDP напоминает бросок рулета тетушке Бетти, которая может быть поймает его, а может быть он будет пойман собакой, следящей за обедающими. (Да, воскресные обеды в моей семье проходят гораздо интереснее, чем в большинстве других семей.)

Порты транспортного протокола

Вы когда-нибудь замечали, что компьютеры имеют слишком много портов? Мы собираемся добавить в список порты TCP и UDP. *Порты транспортных протоколов* позволяют одному серверу предоставлять различные сетевые услуги посредством единственного транспортного протокола, обеспечивая возможность организации множественных соединений между компьютерами.

Когда запускается программа-сервер, она подключается к одному или более логических портов. Логический номер порта – это просто число от 1 до 65535. Например, почтовые серверы принимают соединения на порту TCP с номером 25. Каждый пакет TCP или UDP, поступающий в систему, имеет поле, которое содержит номер порта доставки. Каждый входящий запрос помечен номером порта, куда он должен быть доставлен. Если входящий запрос приходит на порт с номером 25, он передается почтовому серверу, ожидающему запросы на соединения на этом порте. Это означает, что разные порты могут обслуживаться разными программами, клиенты могут взаимодействовать с серверами на разных портах, и никто не путается, кроме системного администратора.

Файл */etc/services* содержит список номеров портов и сервисов, которые с ними ассоциированы. Почти любой сервис можно запускать на произвольном порту, но таким образом можно сбить с толку другие хосты Интернета, которые пытаются соединиться с вашей системой. Если кто-то пытается отправить электронную почту, его программа автоматически пытается соединиться с портом 25 вашей системы. Если служба электронной почты работает на порту с номером 77, а веб-сервер на порту 25, вы никогда не сможете получить свою электронную почту, а ваш веб-сервер начнет получать спам. Формат файла очень прост – он содержит всего пять колонок:

```
❶qotd ❷17/❸tcp ❹quote ❺#Quote of the Day
```

Эта запись описывает службу qotd ❶, которая работает на порту с номером 17 ❷ протокола TCP ❸. Эта служба известна также под названием quote ❹. В последней колонке содержится комментарий ❺, более подробно описывающий службу, в частности здесь говорится, что название *qotd* происходит от английского «Quote of the Day» (цитата дня). Службам присваиваются одни и те же номера портов для протоколов TCP и UDP, даже несмотря на то, что обычно они могут использовать только один протокол, например, со службой qotd связаны порты 17/tcp и 17/udp.

Многие программы читают */etc/services*, чтобы выяснить, к какому порту осуществить привязку (bind to), а клиентские программы читают */etc/services*, чтобы выяснить, к какому порту обращаться для установления соединения. Чтобы вынудить сервер использовать другой порт, надо отредактировать этот файл и сообщить серверу, какой порт следует использовать.

Подобно всем другим стандартам, содержимое файла */etc/services* не является догмой. Служба sshd обычно занимает порт 22/tcp, однако по разным причинам я запустил ее на портах 23 (telnet), 80 (HTTP) и 443 (HTTPS). Подобные изменения зависят от программы, задействованной для предоставления того или иного сервиса. Примеры использования такого подхода мы будем рассматривать в главе 15.

Зарезервированные порты

Порты протоколов TCP и UDP с номерами 1024 и ниже называются *зарезервированными портами*. Они зарезервированы для ключевых служб инфраструктуры Интернета, таких как DNS, SSH, HTTP, LDAP и других, которые должны предоставляться только системой или сетевым администратором. Только программы, обладающие привилегиями суперпользователя (root-level), могут привязываться к зарезервированным портам (портам с низкими номерами). Пользователь может, например, запустить игровой сервер на одном из портов с высокими номерами, если политика безопасности системы допускает это, но назначение компьютера игровым сервером отличается от настройки официального веб-сервера, доступного всем желающим! Привязка портов к базовым протоколам словно высечена на каменных скрижалях.

Просмотреть и изменить зарезервированные порты можно с помощью параметров sysctl `net.inet.ip.portrange.reservedhigh` и `net.inet.ip.portrange.reservedlow`.

Многие часто думают, что если бы можно было запретить правило «привязки только при наличии прав суперпользователя», это повысило бы степень защищенности системы – в конце концов, если приложение будет работать не с правами root, а с правами обычного пользователя, разве это не повысило бы безопасность системы? В действительности большинство программ, использующих зарезервированные порты, запускаются с привилегиями пользователя root, выполняют привязку к порту, а затем понижают свои привилегии до уровня специально созданного пользователя, который обладает еще меньшими привилегиями, чем обычный пользователь. Эти программы разрабатывались так, чтобы запускаться с привилегиями суперпользователя, и часто ведут себя иначе, когда их запускают с привилегиями обычного пользователя. Некоторые программы, такие как веб-сервер Apache, написаны так, чтобы сразу запускаться с привилегиями обычного пользователя, но другие не предусматривают такой возможности.

Ethernet

Ethernet является наиболее популярным носителем в корпоративных и домашних сетях и является наиболее типичным носителем для систем FreeBSD. Ethernet – это общая сеть; самые разные компьютеры могут подключаться к одной и той же сети Ethernet и напрямую взаимодействовать друг с другом. Это одно из преимуществ Ethernet перед другими сетевыми протоколами. Однако Ethernet налагает ограничения на физические расстояния между системами, поэтому этот протокол подходит только для офисов, внутренних сетей хостинг-провайдеров (co-location facilities) и других сравнительно небольших сетей.

На протяжении многих лет для реализации Ethernet применялось много различных физических носителей. Было время, когда большинство

кабелей Ethernet представляли собой толстый коаксиальный кабель, а сегодня в большинстве сетей Ethernet задействованы сравнительно тонкие кабели категории 5 (CAT5) с восемью тонкими проводами внутри. Ethernet можно реализовать по оптоволокну или через радиоэфир. В рамках нашего разговора предположим, что Ethernet в данном случае реализована по кабелю CAT5 – сегодня это наиболее распространенный носитель. Независимо от того, какой физический носитель используется, теория Ethernet от этого не меняется – не забывайте, что физический уровень для нас лишь абстракция.

Протоколы и аппаратное обеспечение

Ethernet – это широковещательный (broadcast) протокол – каждый пакет, посылаемый по сети, может доставляться на каждую рабочую станцию. (Обратите внимание на оговорку «*может*» – некоторые устройства Ethernet ограничивают возможность приема таких широковещательных посылок.) Драйвер сетевой платы на той или иной рабочей станции отделяет данные, предназначенные для этого компьютера, от всех остальных передаваемых пакетов. Побочный эффект широковещательной сущности Ethernet – возможность подслушивать трафик, предназначенный для других компьютеров сети. Хотя такая возможность очень полезна при диагностировании неполадок, она добавляет хлопот с точки зрения безопасности: перехват пароля – тривиальная операция в старых сетях Ethernet. Часть сети Ethernet, где все хосты могут напрямую взаимодействовать друг с другом, не прибегая к услугам маршрутизатора, называется *доменом коллизий* (*collision domain*), или *сегментом*.

Сегменты Ethernet соединяются между собой с помощью сетевых коммутаторов или концентраторов. *Концентратор* (hub) Ethernet – это центральная часть аппаратного обеспечения, обеспечивающего физическое соединение устройств Ethernet. Он просто осуществляет переадресацию и пересылку информации уровня Ethernet между устройствами, которые к нему подключены. Весь поступающий трафик Ethernet концентраторы переадресуют каждому присоединенному хосту и другим концентраторам. А каждый хост отвечает за фильтрацию нежелательного трафика. Это традиционный Ethernet.

Коммутаторы в значительной степени вытеснили концентраторы. Коммутатор предоставляет усовершенствованный способ соединения устройств Ethernet, при котором скорость передачи данных в сети Ethernet улучшается за счет отслеживания IP и MAC-адресов всех подсоединенных устройств и перенаправления пакетов тому устройству, для которого они предназначены. Поскольку каждый хост Ethernet имеет ограниченную пропускную способность, коммутация снижает нагрузку на отдельные системы за счет ограничения объема данных, передаваемых каждому устройству.

Отказы коммутаторов

Случается так, что несмотря на надежность устройств, выпускаемых компанией Cisco, коммутаторы иногда выходят из строя. Некоторые поломки видны сразу, как, например, когда из-под крышки прибора валит таинственный черный дым. Дым перестает идти, а прибор прекращает работать. Другие поломки не так очевидны, и на первый взгляд создается ощущение, что коммутатор в порядке.

Каждый производитель коммутаторов сам решает, как устройство должно реагировать на всевозможные отказы. Либо коммутатор прекращает передачу данных, пока к нему не будет проявлено должное внимание, либо он пытается предупредить администратора и продолжает передачу пакетов, хоть как-нибудь. Если вы производитель, ваш выбор вполне очевиден – прибор должен работать, пусть хоть кое-как, благодаря этому у покупателя даже не появляется мысли, что ваши коммутаторы никуда не годятся. Это означает, что коммутатор начинает действовать как концентратор и никто может даже не подозревать об этом. Плохо здесь то, что если вы рассчитываете на коммутатор, как на одно из средств, предотвращающих утечку секретной информации, вас ждет разочарование. В моей практике не раз случались поломки коммутаторов, поэтому особо не удивляйтесь, если это произойдет и с вами.

Установка и использование сервера syslog (глава 19) поможет уменьшить этот риск. Этот сервер не сможет предотвратить поломки коммутаторов, зато он поможет прислушиваться к их жалобам.

Скорость и двунаправленная передача данных в Ethernet

Первоначально Ethernet поддерживал скорость передачи всего в несколько мегабит в секунду, но со временем он стал поддерживать скорости в десятки гигабит. В большинстве случаев используются скорости 10/100 мегабит в секунду (Mbps). Если на сетевой карте имеется наклейка с надписью *10/100 Mbps*, это совсем не значит, что она в действительности способна пропускать данные с такой скоростью, – мне приходилось видеть сетевые карты с пометкой 100 Mbps, которые едва дотягивали до скорости 10 Mbps, и гигабитные карты, которые начинали «задыхаться» на скорости 100 Mbps. Большое значение имеет качество сетевой карты, если вам нужна высокая скорость, и не менее важно качество самого компьютера, когда речь идет о высоких скоростях.

Такая характеристика, как *duplex* (дуплекс, двунаправленная передача), определяет – может ли сетевая карта выполнять прием и передачу данных одновременно. Соединение типа *half-duplex* (полудуплекс, односторонняя передача) означает, что в каждый конкретный момент времени сетевая карта может выполнять либо только прием, либо только передачу данных – она не в состоянии выполнять обе операции одновременно. В случае соединения *fullduplex* (полный дуплекс) прием и передача данных могут осуществляться одновременно.

MAC-адреса

Каждая сетевая карта Ethernet имеет уникальный идентификатор, или *MAC-адрес* (Media Access Control – управление доступом к среде передачи данных). MAC-адрес – это 48-битное число, которое иногда называют *адресом Ethernet*. Когда одна система хочет передать данные другой через Ethernet, она сначала посылает широковещательный запрос, суть которого можно выразить так: «Какой MAC-адрес отвечает за этот IP-адрес?» Если хост отвечает, отправляемые данные помечаются в соответствии с его MAC-адресом. Этот процесс известен как *протокол разрешения адресов* (Address Resolution Protocol, ARP).

Получить данные из таблицы ARP в системе FreeBSD можно с помощью команды arp(8). Обычно ее запускают как `arp -a`, что позволяет увидеть MAC-адреса и имена всех хостов в сети:

```
# arp -a
gw.blackhelicopters.org (192.168.3.1) at 00:00:93:34:4e:78 on fxp0
[ethernet]
sipura.blackhelicopters.org (192.168.3.5) at 00:00:93:c2:0f:8c on fxp0
[ethernet]
```

Этот листинг информации ARP называется *таблицей ARP* или *таблицей MAC*. (Термины MAC и ARP часто используются как взаимозаменяемые, поэтому не нужно беспокоиться по этому поводу.) В этом примере хост *gw.blockhelicopters.org* имеет IP-адрес 192.168.3.1 и MAC-адрес 00:00:93:34:4e:78, а подключиться к этому хосту можно через интерфейс fxp0.

Если MAC-адрес показан как `incomplete` (неполный), установить соединение с хостом не удастся. В этом случае необходимо проверить физический уровень (провод), удаленную систему и конфигурацию системы.

Настройка подключения к Ethernet

Прежде чем приступать к настройке системы для доступа к сети Ethernet, у вас должна иметься основная информация об IP-адресе. Если в сети используется протокол динамической настройки конфигурации хоста (Dynamic Host Configuration Protocol, DHCP) и ваш компьютер является клиентом этой службы, вы просто должны использовать ее для доступа к сети, как и любой другой клиент. Если компьютер играет роль сервера, для него больше подходит статический IP-адрес. Для каждого сервера необходимы:

- IP-адрес сервера
- Сетевая маска для этого IP-адреса
- IP-адрес шлюза по умолчанию

Вооружившись этой информацией, вы сможете подключить свою систему к сети с помощью команд ifconfig(8) и route(8).

ifconfig(8)

Утилита ifconfig(8) выводит список сетевых интерфейсов компьютера и позволяет настраивать их. Начнем с исследования списка интерфейсов в системе, который выводит команда ifconfig(8) без аргументов:

```
# ifconfig
❶fxp0: flags=8843<❷UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=3<RXCSUM,TXCSUM>
        inet ❸198.22.63.43 netmask ❹0xfffffff0 broadcast 198.22.63.47
        ether ❺00:02:b3:be:df:f5
        media: ❻Ethernet autoselect (10baseT/UTP)
        status: ❼active
rl0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> mtu 1500
        options=8<VLAN_MTU>
        ether 00:20:ed:72:3b:5f
        media: Ethernet autoselect (10baseT/UTP)
        status: ❽no carrier
❾lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
        inet 127.0.0.1 netmask 0xff000000
```

Первый сетевой интерфейс в списке – fxp0 ❶, или это описание первой сетевой карты, для управления которой используется драйвер fxp(4). Страница руководства fxp(4) сообщает, что это сетевая карта Intel EtherExpress PRO. Далее следует ряд основных сведений об этой карте ❷, включая слово UP, которое сообщает, что карта либо уже работает, либо пытается включиться в работу. Далее следует IP-адрес 198.22.63.43 ❸, присвоенный этому интерфейсу, и сетевая маска 0xfffffff0 ❹ (или 255.255.255.240, согласно табл. 6.2). Здесь также приводится MAC-адрес ❺ и скорость соединения ❻. Наконец запись `status` сообщает, что данная сетевая карта активна ❼: кабель подключен и горит индикатор подключения.

Для второй сетевой карты rl0 утилита не выводит ничего подобного. Ключевым фактором здесь является отсутствие несущего сигнала (`no carrier`) ❽: кабель не подключен и индикатор не горит. Эта карта не используется.

Последний сетевой интерфейс lo0 ❾ – это петлевой интерфейс. Данному интерфейсу на любом компьютере присваивается IP-адрес 127.0.0.1. Этот адрес используется для организации взаимодействия сетевых приложений, исполняющихся на одной и той же машине. Это стандартный программный интерфейс, который не связан с каким-либо аппаратным обеспечением. Не пытайтесь удалить петлевой интерфейс или присвоить ему другой IP-адрес – в этом случае многие приложения не смогут работать. FreeBSD поддерживает и другие программные интерфейсы, такие как disc(4), faith(4), gif(4) и многие другие.

Назначение IP-адреса интерфейсу

В процессе установки системы выполняется настройка всех сетевых карт, имеющихся в компьютере. Если в ходе установки были настроены

не все сетевые карты или если уже после установки были удалены существующие или добавлены новые сетевые карты, утилита ifconfig(8) позволит присвоить IP-адреса сетевым картам. Сетевой карте необходимо присвоить IP-адрес и сетевую маску.

```
# ifconfig interface-name IP-address netmask
```

Например, предположим, что сетевой карте fxp0 необходимо присвоить IP-адрес 192.168.1.250 и установить сетевую маску 255.255.255.0; в этом случае необходимо запустить такую команду:

```
# ifconfig fxp0 192.168.1.250 255.255.255.0
```

Сетевую маску можно указывать в точечной нотации, как это сделано выше, или в шестнадцатеричном формате (0xffffff0). Но, пожалуй, самый простой способ состоит в том, чтобы указать маску через символ слэша:

```
# ifconfig fxp0 192.168.1.250/24
```

Утилита ifconfig(8) способна выполнять и любые другие настройки сетевых карт, такие как выбор типа носителя сигнала и установка дуплексного режима. Перечень поддерживаемых типов носителей сигнала и порядок установки дуплексного режима можно найти в странице руководства к драйверу карты. Выбор типа носителя производится с помощью ключевого слова `media`, а установка дуплексного режима – с помощью ключевого слова `mediaopt`. Некоторые комбинации сетевых карт и сетевых коммутаторов не в состоянии автоматически «договориться» о параметрах подключения, поэтому иногда возникает необходимость вручную устанавливать скорость подключения и дуплексный режим с той или с другой стороны. Некоторые сетевые карты поддерживают как полудуплексный, так и полнодуплексный режимы на скорости 100 Mbps, но на скорости 10 Mbps – только полнодуплексный режим. (Стандарт Ethernet со скоростями от 1 гигабита требует, чтобы устройства сами договаривались о выборе режима, поэтому принудительная установка – не самый лучший вариант.) Некоторые сетевые карты имеют несколько разъемов или допускают подключение носителей разных типов к одному и тому же разъему. Безусловно, все необходимые настройки можно объединить в одной команде.

```
# ifconfig fxp0 192.168.1.250/24 media 1000baseTX mediaopt full-duplex
```

Чтобы настройки сохранялись после перезагрузки, следует добавить строку, которая сообщит системе все необходимые настройки, в файл */etc/rc.conf*. Строка имеет вид `ifconfig_`*`имя_интерфейса`*`="`*`аргументы ifconfig`*`"`. Например, настройка сетевой карты re0 могла бы выглядеть примерно так:

```
ifconfig_re0="192.168.1.250 255.255.255.0 media 1000baseTX mediaopt full-
duplex"
```

Как только будет найдена работающая конфигурация для интерфейса, достаточно просто скопировать аргументы ifconfig(8) в */etc/rc.conf*.

Проверка интерфейса

Теперь, когда интерфейсу присвоен IP-адрес, попробуем выполнить ping, задав IP-адрес шлюза по умолчанию. Если ответ получен, как показано в следующем листинге, то машина успешно подключена к сети. Выполнение ping можно прервать, нажав клавиши Ctrl-C.

```
# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=1.701 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.436 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.436/1.569/1.701/0.133 ms
```

Если ответ не был получен, следовательно, ваш сетевой интерфейс не работает. Дело либо в плохом соединении (проверьте кабель и свечение индикатора), либо в неверной настройке интерфейса.

Установка маршрута по умолчанию

Маршрут по умолчанию – это адрес, по которому система по умолчанию (т. е. если не указано иного для конкретных внешних адресов – *прим. научн. ред.*) посылает весь трафик, который отправляется за пределы локальной сети. Если в результате использования команды ping с адресом маршрутизатора по умолчанию был получен ответ, можно установить этот IP-адрес в качестве маршрута по умолчанию с помощью команды route(8).

```
# route add default 192.168.1.1
```

Вот и все! Теперь ping должен проходить по любому IP-адресу Интернета.

Если в ходе установки системы вы не указали используемые серверы имен, то вместо имен хостов вам придется указывать их IP-адреса. Настройка DNS будет рассматриваться в главе 14, а пока замечательным источником IP-адресов могут служить корневые серверы имен, перечисленные в файле */etc/namedb/named.root*.

Маршрутизатор по умолчанию, устанавливаемый при начальной загрузке системы, можно задать оператором `defaultrouter` в */etc/rc.conf*:

```
defaultrouter="192.168.1.1"
```

Несколько IP-адресов на одном сетевом интерфейсе

Одна система FreeBSD может отвечать на запросы по нескольким IP-адресам на одном интерфейсе. Такая возможность широко применяется в серверах Интернета, особенно для защищенных (SSL) веб-сайтов. Возможно, один сервер должен поддерживать сотни или тысячи доменов, и ему нужен IP-адрес для каждого из них. Дополнительные IP-адреса

можно добавить с помощью команды ifconfig(8) и ключевого слова alias. Сетевая маска для псевдонима (alias) всегда устанавливается равной /32, независимо от количества адресов в сети с основным адресом.

```
# ifconfig fxp0 alias 192.168.1.225/32
```

После запуска предыдущей команды дополнительные IP-адреса появятся в выводе команды ifconfig(8). Основной IP-адрес всегда идет первым, а за ним следуют псевдонимы:

```
# ifconfig fxp0
fxp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      options=b<RXCSUM,TXCSUM,VLAN_MTU>
      inet6 fe80::202:b3ff:fe63:e41d%fxp0 prefixlen 64 scopeid 0x1
      inet 192.168.1.250 netmask 0xffffff00 broadcast 192.168.1.255
      inet 192.168.1.225 netmask 0xffffffff broadcast 192.168.1.255
      ether 00:02:b3:63:e4:1d
...
```

Остальные хосты, запустив команду ping с IP-адресами-псевдонимами, будут получать ответ от этого сервера.

Настроив псевдонимы, можно добавить еще один оператор ifconfig с параметрами их настройки в файл */etc/rc.conf*:

```
ifconfig_fxp0_alias0="192.168.1.225/32"
```

Единственное реальное различие между этой записью и стандартной записью *rc.conf* – это фрагмент alias0. Ключевое слово alias сообщает системе FreeBSD, что этот IP-адрес является псевдонимом, а 0 – это уникальный номер псевдонима. Каждый псевдоним, устанавливаемый в */etc/rc.conf*, должен иметь уникальный номер, причем номера должны быть последовательными. Если пропустить какой-нибудь номер, то псевдонимы после пропуска не будут установлены при начальной загрузке. Это самая распространенная ошибка в конфигурировании интерфейсов; система FreeBSD требует перезагрузки так редко, что ошибки в */etc/rc.conf* могут месяцами оставаться незамеченными!

Многие демоны могут выполнить привязку только одного IP-адреса, поэтому, чтобы иметь возможность обслуживать несколько IP-адре-

Псевдонимы и исходящие соединения

Все исходящие соединения используют действительные IP-адреса. К сетевой карте может быть привязано 2000 IP-адресов, однако когда вы соединяетесь с внешним сервером по ssh, соединение исходит с первичного IP-адреса. Об этом следует помнить при написании правил для межсетевого экрана и других фильтров, управляющих доступом к системе. Инициация соединений с адресов-псевдонимов блокируется, но об этом мы поговорим в главе 9.

сов, вам придется запускать несколько экземпляров программ. Например, named(8) (глава 14) способен привязать только один IP-адрес, но есть возможность запустить несколько экземпляров named(8) на одном компьютере, используя различные IP-адреса для каждого из экземпляров.

Переименование интерфейсов

FreeBSD присваивает сетевым интерфейсам имена, исходя из названия драйвера устройства, который используется для обслуживания сетевой карты. Это старая добрая традиция в мире UNIX, и она соблюдается в большинстве промышленных операционных систем. Некоторые операционные системы дают имена сетевым интерфейсам, исходя из их типа, например в Linux интерфейсы Ethernet получают имена eth0, eth1 и т. д. Иногда появляется необходимость переименовать интерфейс – либо для обеспечения соответствия стандартам, либо для большей наглядности. Например, представим, что имеется устройство с двенадцатью сетевыми интерфейсами, каждый из которых подключен к отдельной сети. Каждая сеть имеет свои имена, такие как *test*, *QA* и т. д. Определенно имеет смысл переименовать сетевые интерфейсы, чтобы их имена наглядно говорили о том, к какой сети подключен каждый из них.

Система FreeBSD отличается высокой гибкостью в выборе имен интерфейсов, но некоторые программы предполагают, что имя интерфейса – это короткое слово, за которым следует число. Такое положение дел едва ли изменится в ближайшем будущем, поэтому для интерфейсов желательно выбирать короткие имена, оканчивающиеся цифрой. Для переименования сетевого интерфейса используется ключевое слово `name` команды ifconfig(8). Например, чтобы переименовать интерфейс fxp1 в test1, можно было бы запустить следующую команду:

```
# ifconfig fxp1 name test1
```

Запустив после этого команду ifconfig(8) без аргументов, можно убедиться, что интерфейс был переименован.

```
...
test1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
       options=b<RXCSUM,TXCSUM,VLAN_MTU>
...
```

Чтобы сохранить эти изменения, следует добавить параметр `ifconfig_`*`interface`*`_name` в файл */etc/rc.conf*.

```
ifconfig_fxp1_name="test1"
```

Переименование интерфейсов во время загрузки FreeBSD должно производиться до назначения IP-адреса или других значений. То есть при настройке параметров интерфейса должно использоваться не старое, а новое имя. Полная настройка интерфейса, включая его переименование, может выглядеть примерно так:

```
ifconfig_fxp1_name="dmz2"
ifconfig_dmz2="192.168.1.2 netmask 255.255.255.0"
ifconfig_dmz2_alias0="192.168.1.3"
```

DHCP

Существуют такие сети, в которых назначение IP-адресов компьютерам, включая и серверы, выполняется посредством протокола DHCP. Сервер DHCP выполняет назначение IP-адресов, сетевых масок, серверов имен и шлюзов по умолчанию. Если в вашей сети конфигурирование серверов производится посредством DHCP, вам необходимо настроить сетевую карту так, чтобы она принимала всю необходимую информацию с помощью DHCP, например, так:

```
ifconfig_fxp0="DHCP"
```

Перезагрузка!

Теперь, когда все сетевые интерфейсы настроены, выполните перезагрузку системы, чтобы убедиться, что все параметры настройки, указанные в файле */etc/rc.conf*, правильно устанавливаются во время загрузки. Если во время загрузки FreeBSD обнаружит ошибки в */etc/rc.conf*, особенно в настройках сети, вы не сможете взаимодействовать с системой с удаленного рабочего места. Лучше все ошибки выявить сразу, пока имеется доступ к серверу, чем потом терять часы работы.

Деятельность в сети

Теперь, когда подключение к сети установлено, как можно увидеть, что происходит в ней? Существует несколько способов заглянуть в сеть, и мы рассмотрим все эти способы по порядку. В отличие от многих коммерческих операционных систем, информацию о сети в системе FreeBSD можно получить с помощью команд netstat(8) и sockstat(1).

Текущая активность в сети

netstat(8) – это многоцелевая программа наблюдения за сетями, которая отображает различные сведения в зависимости от ключей, с которыми она была запущена. Наиболее часто возникает вопрос: «Какой объем трафика проходит через компьютер прямо сейчас?» Команда netstat(8) с ключом `-w` покажет количество пакетов и байтов, обработанных системой. Ключ `-w` принимает один аргумент – количество секунд между обновлениями. Ключ `-d` сообщит netstat(8) о необходимости включить в вывод информацию о сброшенных пакетах. Ниже представлена команда, которая заставляет netstat(8) обновлять информацию на экране каждые 5 секунд:

```
# netstat -w 5 -d
            input       (Total)           output
   packets  errs     bytes    packets  errs      bytes  colls  drops
```

```
          ❶34     ❷0      ❸44068          ❹23      ❺0        ❻1518     ❼0      ❽0
    33  0  42610  23  0  1518  0  0
...
```

Сразу после ввода команды никаких видимых изменений на экране не происходит, но через несколько секунд выводится единственная строка с информацией. Первые три колонки описывают входящий трафик, а следующие три – исходящий. Здесь можно увидеть число пакетов, принятых с момента последнего обновления ❶, число ошибок интерфейса при приеме входящего трафика с момента последнего обновления ❷ и число байтов, принятых с момента последнего обновления ❸. В следующих трех колонках выводится число пакетов, отправленных компьютером с момента последнего обновления ❹, количество ошибок передачи с момента последнего обновления ❺ и количество отправленных байтов ❻. Далее выводится количество коллизий в сети, обнаруженных с момента последнего обновления ❼, и количество сброшенных пакетов ❽. Например, в данном случае система приняла 34 пакета ❶, начиная с момента запуска команды `netstat -w 5 -d`.

Через пять секунд netstat(8) вывела вторую строку, описывающую сетевую активность, начиная с момента вывода первой строки.

Интервал обновления информации можно регулировать по своему желанию. Если необходимо выводить сведения раз в секунду, просто запустите команду `netstat -w 1 -d`. Если достаточно интервала в одну минуту, запустите команду `netstat -w 60 -d`. Когда я занимаюсь мониторингом сетевой активности, наиболее подходящим для меня является интервал в пять секунд, однако вы сами быстро поймете, какой интервал лучше подходит для ваших нужд.

Чтобы остановить работу утилиты, нужно нажать комбинацию клавиш Ctrl-C.

Открытые порты

Другой не менее популярный вопрос: «Какие порты открыты и какие программы ожидают соединения на них?» В состав системы FreeBSD входит утилита sockstat(1). Этот дружелюбный инструмент позволит получить ответ на столь важный вопрос. Он покажет список активных соединений и портов, доступных для клиентов.

Программа sockstat(1) не только перечислит порты, доступные для соединения, но и другие порты (или сокеты) в системе. Так как основной интерес для нас представляют открытые порты для протокола TCP/IP версии 4, то нужно не забывать использовать ключ `-4`.[1] Ниже приводится сокращенный вывод команды sockstat(1), запущенной на *очень* маленьком сервере:

[1] Ключ `-4` используется для протокола TCP/IP версии 4. Не забывайте, нас сейчас интересует только сеть, и мы игнорируем IPv6.

```
# sockstat -4
USER     COMMAND    PID   FD PROTO  LOCAL ADDRESS      FOREIGN ADDRESS
mwlucas ❶sshd       11872 4   tcp4   ❷198.22.63.43:22   ❸24.192.127.92:62937
❹ root   sendmail   11433 4   tcp4   *:25               *:*
❺ www    httpd      9048  16  tcp4   *:80               *:*
❻ root   sshd       573   3   tcp4   *:23               *:*
❼ root   sshd       426   3   tcp4   *:22               *:*
❽ bind   named      275   20  udp4   198.22.63.8:53     *:*
```

В первой колонке выводится имя пользователя, с привилегиями которого исполняется программа, выполнившая привязку порта. Вторая колонка – это имя команды. Далее указаны идентификатор процесса (PID) программы и дескриптор файла, соответствующего сокету. В следующей колонке указано название транспортного протокола, который используется сокетом, – это либо `tcp4` для TCP в TCP/IP версии 4, либо `udp4` для UDP в TCP/IP версии 4. Далее выводится локальный IP-адрес с номером порта, а в последней колонке указываются IP-адрес и номер порта удаленной системы, в случае открытого соединения.

Взгляните на самую первую запись. Она соответствует работающей программе sshd ❶. Страница справочного руководства sshd(8) рассказывает о демоне SSH. Основной демон sshd(8) запустил дочерний процесс от моего имени для обслуживания соединения с моей рабочей станцией, поэтому в списке имеется несколько экземпляров sshd(8) с различными идентификаторами процесса. В данном случае соединение установлено с локальным IP-адресом 198.22.63.43 ❷ и с портом TCP 22. Удаленный хост, участвующий в этом соединении, имеет IP-адрес 24.192.127.92 ❸ и соединение выполнено с портом 62937. Это SSH-соединение было выполнено на локальный компьютер с удаленной системы.

В число других соединений входят Sendmail ❹, сервер электронной почты, принимающий запросы на порту 25. Обратите внимание: в этой записи отсутствуют IP-адреса. Данный сокет ожидает получения запросов на соединение. Процесс httpd ❺ ожидает получение запросов на порту с номером 80.

Наиболее проницательные читатели могут заметить, что на этом сервере имеются два демона SSH, ожидающие получения запросов на подключение, один на порту с номером 23 ❻ и другой на порту с номером 22 ❼. В файле */etc/services* указывается, что SSH обычно работает с портом 22, а порт 23 зарезервирован за службой telnet. Любой, кто попытается подключиться с помощью команды telnet, будет подключен к демону SSH, который работает совсем не так, как ожидается. Наиболее подозрительные читатели могут предположить, что данный сервер SSH был настроен для компенсации недостатков в настройках межсетевого экрана, который фильтрует трафик, исходя из исходящего и входящего номеров портов, а не из фактически используемого протокола. (У меня нет комментариев по поводу таких утверждений.)

Последняя запись в списке соответствует серверу имен named ❽, ожидающему запросы на порту с номером 53. Здесь видно, что прослушивание производится по протоколу UDP (а не TCP) и соединения принимаются на IP-адресе 198.22.63.8.

Подробнее об открытых портах

Утилита sockstat(1) предоставляет обзорную информацию о доступных сетевых службах, однако с помощью netstat(8) можно получать более подробные сведения об отдельных соединениях. Чтобы увидеть открытые сетевые соединения, можно использовать команду netstat(8) с ключом `-a`. Ключ `-n` сообщает команде netstat(8), что она не должна выполнять преобразование IP-адресов в имена хостов, – такое преобразование не только замедляет вывод информации, но и может породить неоднозначность полученных результатов. Наконец, параметр `-f inet` требует от netstat(8) беспокоиться только о сетевых соединениях. Ниже приводится пример вывода, полученного от команды netstat с того же компьютера, на котором только что был получен вывод команды sockstat(1):

```
# netstat -na -f inet
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address     (state)
tcp4       0     48  198.22.63.43.22        24.192.127.92.62937  ESTABLISHED
tcp4       0      0  *.25                    *.*                  LISTEN
tcp4       0      0  *.23                    *.*                  LISTEN
tcp4       0      0  *.80                    *.*                  LISTEN
tcp4       0      0  *.22                    *.*                  LISTEN
udp4       0      0  198.22.63.43.53         *.*
```

Здесь нетрудно догадаться, какая программа к какому порту подключена. Первая колонка во всех записях – это название транспортного протокола, используемого сокетом, – здесь в большинстве случаев используется протокол TCP, и только в последней записи – UDP.

Колонки `Recv-Q` и `Send-Q` показывают, сколько байт в том или ином соединении ожидают обслуживания. Если в колонке `Recv-Q` ненулевые значения присутствуют постоянно, значит, система не может обрабатывать входные данные достаточно быстро. Точно так же постоянно присутствующие значения в `Send-Q` говорят о том, что либо сеть, либо другая сторона соединения не могут принять данные с той скоростью, с какой они посылаются. Случайные пакеты, ожидающие своей очереди, – это нормально. Исследуйте свою систему, чтобы определить, какая ситуация нормальна, а какая – нет.

Очевидно, что `Local Address` – это IP-адрес и номер порта локальной системы, на котором локальная система ожидает получение запросов на соединение. Сетевой порт находится в конце записи и отделяется от IP-адреса точкой. Например, 198.22.63.43.22 – IP-адрес 198.22.63.43 и порт 22. Если запись представлена как «звездочка, точка, номер

порта», это означает, что система прослушивает этот порт на всех IP-адресах. В приведенном примере система готова принимать соединения по указанным портам.

Колонка `Foreign Address` (внешний адрес) показывает удаленный адрес (remote address) и номер порта для каждого соединения.

Наконец, колонка `(state)` показывает состояние рукопожатия TCP. В данный момент не требуется знать все возможные состояния TCP-соединения; достаточно понять, какие из них являются нормальными. Состояние `ESTABLISHED` означает, что соединение установлено и данные, по всей видимости, передаются. Состояния `LAST_ACK`, `FIN_WAIT_1` и `FIN_WAIT_2` означают, что соединение закрывается. Состояния `SYN_RCVD`, `ACK` и `SYN+ACK` – это этапы нормального создания соединения. Состояние `LISTEN` говорит, что порт готов принимать входящие соединения. В предыдущем примере одно соединение TCP активно и четыре ожидают входящих соединений. Протокол UDP не устанавливает соединений, поэтому в данном листинге для этого протокола отсутствует информация о состоянии.

Ознакомившись с этой информацией и сопоставив ее со сведениями, полученными от sockstat(1), можно разобраться, какая программа ведет себя вполне нормально, а какая является узким местом в системе.

Если вас не интересуют сведения о сокетах, ожидающих входящие соединения, а интерес представляют только активные соединения, можно вместо ключа `-a` использовать ключ `-b`. Например, команда `netstat -nb -f inet` отобразит только установленные соединения с удаленными системами.

Пропускная способность сетевой подсистемы в ядре

Система FreeBSD оптимизирует сетевую работу с помощью mbufs. *mbuf* – это отдельный участок памяти ядра, выделенный для сетевых операций. При просмотре документации, описывающей сетевой стек FreeBSD, вы постоянно будете спотыкаться о термин mbuf, поэтому очень важно иметь хотя бы поверхностное представление об этом.

Во время загрузки FreeBSD автоматически выделяет память для нужд сетевой подсистемы, исходя из общего объема физической памяти. Предполагается, что если в системе имеется 4 Гбайт памяти, то было бы желательно выделить для нужд сетевой подсистемы больше памяти, чем ее выделяется на маломощном компьютере с 128 Мбайт памяти. Узнать, как FreeBSD распределяет эти ресурсы, можно с помощью команд `netstat -s` и `netstat -m`. Рассмотрим сначала самый короткий вариант.

Команда `netstat -m` позволяет получить общее представление об использовании памяти ядра для сетевой подсистемы. Полученные результаты делятся на две большие категории: как много памяти используется, и сколько запросов на соединение потерпело неудачу. Ниже приводится сокращенный вывод этой команды, который включает

в себя всего несколько примеров, но все они следуют одному и тому же формату:

```
# netstat -m
...
❶32/372/404/❷25600 mbuf clusters in use (current/cache/total/max)
...
0/0/0 requests for mbufs ❸denied (mbufs/clusters/mbuf+clusters)
...
```

Здесь видно, сколько кластеров mbuf используется ❶. Если вы сделали предположение, что они как-то связаны с mbufs, то вы были правы. Вам пока не обязательно точно знать, что такое кластеры mbuf, важно знать количество, которое может быть выделено ❷, и что этот предел еще не достигнут.

Также здесь можно видеть количество запросов на выделение mbufs, отвергнутых ядром ❸. В данном случае система не отвергла ни одного запроса, а это означает, что в ней отсутствуют проблемы производительности, связанные с нехваткой памяти. Если система начинает отвергать запросы на выделение mbufs, следовательно, в сетевой подсистеме ощущается нехватка памяти. О решении этой проблемы рассказывается в разделе «Оптимизация производительности сети» ниже.

Команда `netstat -m` производит всего с десяток строк результатов, а вот команда `netstat -s` выводит данные страница за страницей. Она выводит статистические данные о производительности для каждого из протоколов. Как и в случае с командой `netstat -m`, эти данные можно разбить на категории: как много было сделано и сколько проблем было обнаружено. Время от времени запускайте обе эти команды и просматривайте полученные результаты, чтобы быть в курсе событий и вовремя обнаруживать возникающие проблемы.

Оптимизация производительности сети

Как теперь, имея возможность увидеть, что происходит с сетевой подсистемой FreeBSD, можно повысить ее производительность? При рассмотрении вопроса оптимизации существует простое эмпирическое правило: *ничего не предпринимать*. Вообще производительность сетевой подсистемы ограничивается только возможностями аппаратных средств. С другой стороны, многие приложения не в состоянии обрабатывать данные с той же скоростью, с какой их поставляет сетевая подсистема. Если у вас появилась мысль о необходимости оптимизации производительности системы, скорее всего вы смотрите не в ту сторону. Прочитайте главу 19, где приводятся рекомендации для тех, кто занимается поиском узких мест.

Вообще говоря, сетевая подсистема нуждается в корректировке только в том случае, если вы испытываете сложности при работе в сети. Это означает, что вы должны иметь результаты работы команд `netstat -m`

и `netstat -s`, которые явно указывают на нехватку ресурсов, выделяемых ядром. Если ядро начинает отвергать запросы или закрывать соединения из-за нехватки ресурсов, прочитайте рекомендации, которые приводятся в этом разделе. В первую очередь, при появлении проблем или при желании повысить производительность системы обратите внимание на аппаратные средства.

Оптимизация сетевых аппаратных средств

Не все сетевые устройства одинаковы. То, о чем часто слышал любой специалист в области информационных технологий, открытая природа FreeBSD делает очевидным. Например, ниже приводится комментарий, взятый из исходных текстов драйвера сетевой карты rl:

```
The RealTek 8139 PCI NIC redefines the meaning of 'low end.' This is
probably the worst PCI ethernet controller ever made, with the possible
exception of the FEAST chip made by SMC. The 8139 supports bus-master
DMA, but it has a terrible interface that nullifies any performance
gains that bus-master DMA usually offers.
(Перевод: Сетевая карта RealTek 8139 PCI по-новому заставляет оценить понятие
'низкокачественный'. Это, пожалуй, самый худший PCI-контроллер Ethernet
из когда-либо производившихся, за исключением разве что чипа FEAST,
выпускаемого компанией SMC. Карта 8139 поддерживает работу с DMA,
но интерфейс карты настолько ужасен, что сводит на нет весь выигрыш
в производительности, который обычно способна дать поддержка DMA.)
```

Этот комментарий может быть перефразирован так: «Эта карта не выдерживает никакой критики. Купите другую карту». Это самый едкий комментарий, который я видел в исходных текстах FreeBSD, впрочем, в наше время эти устройства очень сложно найти. В других драйверах содержатся более корректные комментарии в адрес других сетевых карт. Оптимизировать производительность сетевой подсистемы с помощью низкокачественных устройств – это все равно, что пытаться ставить гоночную трансмиссию на Chevrolet Chevette 1982 года. Скорее поможет замена дешевой карты. Например, Intel выпускает вполне приличные сетевые карты, эта компания предоставляет свой драйвер FreeBSD для своих проводных карт и обеспечивает поддержку сообществу FreeBSD, которая помогает сопровождать этот драйвер. (Беспроводные карты – это уже другая история.) Подобным же образом многие другие компании, выпускающие серверы, считают для себя обязательным использовать высококачественные сетевые карты. Некоторые компании не предоставляют никакой документации к своим устройствам, но поставляют свои драйверы для FreeBSD. Это означает, что драйвер наверняка будет работать, но вы впадаете в зависимость от поставщика при планировании будущих обновлений системы. Устройства от производителей, специализирующихся на выпуске недорогого потребительского сетевого оборудования, – это не самый лучший выбор для установки на высокопроизводительный сервер. В конце концов, обычный средний пользователь понятия не имеет,

как выбирать сетевые карты, и в основном руководствуется ценой на изделие. Если у вас возникают сомнения, обращайтесь в архивы почтовых рассылок, где можно найти достаточно свежие рекомендации по выбору сетевых карт.

Точно так же в широких пределах варьируется и качество сетевых коммутаторов. Если в сопроводительной документации утверждается, что коммутатор поддерживает соединения со скоростями 10/100 Mbps, это совершенно не означает, что вы в действительности получите скорость 100 Mbps! У меня есть сетевой концентратор, рассчитанный на скорость 10 Mbps, который обеспечивает скорость всего 0.5 Mbps, и коммутатор на 100 Mbps, который дает всего 15 Mbps. Скорость работы коммутатора можно представить себе как протокол или язык: я могу заявлять, что говорю по-китайски, но через 20 лет после окончания обучения я смогу выговорить не более трех слов в минуту. Опять же, коммутаторы, предназначенные для домашнего пользования, не самый лучший выбор для использования в промышленных условиях.

Если замена устройств не решила ваших проблем, тогда читайте дальше.

Память

При принятии решения, какой объем памяти выделить для нужд сетевой подсистемы, FreeBSD исходит из общего объема физической памяти. Память, выделяемая для mbufs, не может использоваться для каких-либо других целей, поэтому выделение гигабайтов памяти под mbufs может негативно сказаться на общей производительности системы. Не следует корректировать число mbufs, если команда `netstat -m` явно не укажет на нехватку пространства под mbufs. Если сетевой подсистеме не хватает памяти, самый действенный способ ликвидировать эту проблему заключается в увеличении общего объема памяти компьютера. Это вынудит FreeBSD пересчитать объем для mbufs во время загрузки и тем самым решит проблему. В противном случае вы лишь переместите проблему в другую часть системы или в другое приложение. Можно было бы выделить больше памяти для сетевой подсистемы и посадить на голодный паек сервер базы данных. Если вы уверены в своих действиях, то можно предпринять следующее.

Распределением памяти для mbufs управляют два параметра sysctl – `kern.maxusers` и `kern.ipc.nmbclusters`. Первый из них, `kern.maxusers`, является настройкой, выполняемой на этапе загрузки. Система автоматически определяет соответствующее значение `kern.maxusers` во время загрузки, исходя из аппаратной конфигурации. Корректировка этого значения – пожалуй, лучший способ масштабировать систему в целом. В старых версиях FreeBSD значение `kern.maxusers` отвечало за предварительное выделение памяти для сетевой подсистемы, вследствие чего она становилась недоступной для решения других задач, поэтому чрезмерное увеличение значения `kern.maxusers` могло приводить

к ужасным последствиям для других частей системы. В современных версиях FreeBSD предварительное распределение памяти для сетевой подсистемы не производится, а данное значение обозначает лишь верхний предел, до которого может расти объем памяти сетевой подсистемы. Если значение `kern.maxusers` слишком мало, в файле протокола */var/log/messages* начнут появляться предупреждения (глава 19).

Числом mbufs, выделяемых системой, управляет параметр `kern.ipc.nmbclusters`. Хотя этот параметр и допускает настройку во время работы системы, тем не менее лучше устанавливать его на ранних этапах загрузки, добавив запись в файл */etc/sysctl.conf* (глава 5). Это позволит отобрать память у других задач и передать ее сетевой подсистеме или наоборот. Если это значение выбрать слишком большим, это может привести к нехватке памяти для других задач и даже к аварийному завершению системы.

```
# sysctl kern.ipc.nmbclusters
kern.ipc.nmbclusters: 25600
```

Память для mbufs выделяется блоками, которые называются *nmbclusters* (иногда *mbuf clusters*). Хотя размер одного блока mbuf может варьироваться, размер одного кластера остается постоянным и составляет порядка 2 Кбайт. С помощью несложных вычислений можно выяснить, сколько памяти выделяется для текущего количества nmbclusters, а затем рассчитать приемлемые значения для системы и приложений. В данном примере имеется 25600 nmbclusters, то есть ядро зарезервировало для нужд сетевой подсистемы порядка 50 Мбайт памяти. Это не так много, учитывая, что речь идет о ноутбуке с объемом памяти 1 Гбайт, но это слишком много для какого-нибудь антикварного Pentium.

Чтобы найти наиболее подходящее количество кластеров mbuf, нужно запустить `netstat -m`, когда сервер испытывает серьезную нагрузку. Во второй строке результатов вы получите число mbuf, находящихся в использовании, и общее их число. Если в периоды пиковой нагрузки сервер не использует полный объем доступных кластеров mbuf, следовательно, вы идете по ложному пути – прекратите эксперименты с mbufs и замените, наконец, аппаратные устройства.[1] Например:

```
❶32/❷372/❸404/❹25600 mbuf clusters in use (current/cache/total/max)
```

В настоящий момент система использует 32 кластера ❶, и в кэш были помещены 372 кластера ❷, использовавшиеся ранее. При количестве кластеров, равном 404 ❸, и общем объеме выделенной памяти 25600 кластеров ❹ мы получаем, что используется всего 1,5 процента. Если

[1] Некоторые читатели уже заменили старые устройства новыми, прежде чем приступить к чтению этого раздела. Такие читатели могут воспринимать этот комментарий, как ничем не оправданный. Я приношу свои искренние извинения – всем троим из вас.

это реальная нагрузка на систему, то имеет смысл уменьшить число nmbclusters. Однако есть вероятность, что ваш компьютер вообще имеет низкую нагрузку, тогда любая оптимизация не имеет смысла.

Мое личное эмпирическое правил гласит: сервер должен иметь такое количество mbufs, чтобы оно в два раза превосходило потребность при высокой нагрузке. Если сервер потребляет 25 000 кластеров в часы пик, значит общее число кластеров должно составлять 50 000, чтобы иметь возможность противостоять пиковым нагрузкам.

Планирование пропускной способности сетевой подсистемы

Придет день, когда к вам придет ваш шеф и спросит: «Какой нужен сервер, чтобы иметь возможность обслужить одновременно сто тысяч клиентов?» Я не смогу помочь вам в оценке объемов памяти, которая потребуется вашему приложению (ладно, ладно – я смогу это сделать, но это тема уже другой книги), но рассчитать объем памяти, необходимый для сетевой подсистемы, вам вполне по силам. Каждое TCP-соединение требует наличия приемного и передающего буфера, тогда как для входящих UDP-соединений требуется только приемный буфер. В ходе сеанса FreeBSD может изменять размеры этих буферов, но изначально они имеют размеры, заданные значениями по умолчанию. Узнать значения по умолчанию можно из параметров sysctl – *net.inet.tcp.sendspace*, *net.inet.tcp.recvspace* и *net.inet.udp.recvspace*.

```
# sysctl net.inet.tcp.sendspace
net.inet.tcp.sendspace: 32768
# sysctl net.inet.tcp.recvspace
net.inet.tcp.recvspace: 65536
# sysctl net.inet.udp.recvspace
net.inet.udp.recvspace: 41600
```

Предположим, что у вас имеется веб-сервер, который должен обрабатывать одновременно сто тысяч подключений. Протокол HTTP работает поверх протокола TCP, поэтому для каждого соединения потребуется выделить приемный и передающий буферы. Для каждого соединения потребуется 96 Кбайт памяти (32 768 байт + 65 538 байт = 98 304 байт, или 96 Кбайт). Для одновременного обслуживания ста тысяч пользователей потребуется 9 Гбайт памяти (100 000 × 96 Кбайт = 9 Гбайт)! Кроме того, на этом компьютере должно работать какое-то приложение. Вам потребуется предусмотреть какое-то решение, связанное с распределением нагрузки или кэшированием данных, или готовьтесь писать чертовы объяснительные по поводу единственного компьютера.

Задайте следующий далее вопрос тому, кто попытается предположить, что вам придется *одновременно* обслуживать сто тысяч пользователей. Если только вы не работаете в Yahoo! или в одной из конкурирующих компаний, вам едва ли придется видеть такие нагрузки. Даже если у вашего веб-приложения и наберется 100 000 зарегистрированных пользователей, то они, скорее всего, никогда не будут работать

«Раз в жизни» и стандартная нагрузка

Когда вступил в действие регистрационный сайт правительства США «do not call» с черным списком телефонов, с которых поступают рекламные звонки, одновременно на этом сайте попытались зарегистрироваться миллионы пользователей. Первый день сайт работал ужасно медленно, но уже через неделю сервер прекрасно справлялся с нагрузкой. Это наглядный результат правильного планирования пропускной способности. Нужно отличать пиковые нагрузки, которые случаются один раз в жизни, от обычных повседневных нагрузок.

с ним одновременно. Сколько денег вы готовы потратить, чтобы гарантировать работу в этом крайне редком случае?

Максимальное число входящих соединений

Ядро FreeBSD обеспечивает пропускную способность, необходимую для обработки определенного числа новых входящих TCP-соединений. Это ограничение не касается уже установленных и обрабатываемых подключений, оно относится к числу новых соединений, которые пытаются установить одновременно. Например, в это число не попадают веб-страницы, которые к настоящему моменту уже были отправлены клиентам, – сюда относятся входящие запросы, которые еще даже не достигли веб-сервера.

Параметр `kern.ipc.somaxconn` определяет максимальное число одновременных попыток установить соединение, которые будут обслужены системой. По умолчанию оно равно 128, что может оказаться недостаточным для веб-сервера, работающего под высокой нагрузкой. Если вы занимаетесь сопровождением высокопроизводительного сервера, который, как ожидается, будет получать более 128 новых запросов одновременно, скорее всего, вам потребуется увеличить этот параметр sysctl. Если от пользователей начинают поступать жалобы, что они не могут подключиться к серверу, причина может крыться в значении этого параметра. Конечно, очень немногие приложения способны принимать такое количество новых подключений, поэтому, прежде чем изменять этот параметр, вам, возможно, следует настроить само приложение.

Опрос

Механизм опроса берет проверенную временем идею прерываний и IRQ и выкидывает ее из окна, заменяя регулярными проверками сетевой активности. В классической модели, управляемой прерываниями, всякий раз, когда пакет поступает в сетевую карту, она требует от центрального процессора уделить ей внимание, генерируя прерывание. Процессор прекращает обычную последовательность операций и при-

ступает к обработке этих данных. Это очень хорошо и даже желательно при условии, что карта не занимается обработкой трафика большого объема. Но как только система начинает иметь дело с огромными объемами данных, сетевая карта начинает генерировать прерывания непрерывно. Более эффективный способ заключается в том, чтобы ядро извлекало данные из сетевой карты через регулярные интервалы времени. Такая регулярная проверка называется *опросом* (*polling*). Вообще говоря, использовать механизм опроса полезно только в случае объемного трафика.

К моменту написания этих строк механизм опроса невозможно собрать в виде модуля ядра, так как он требует изменений в драйверах устройств. Также следует иметь в виду, что не все сетевые карты обладают поддержкой механизма опроса, поэтому обязательно ознакомьтесь с полным списком, который приводится в странице руководства polling(4). Чтобы активировать механизм опроса, следует добавить параметр `DEVICE_POLLING` в конфигурацию ядра. После перезагрузки системы разрешить использование механизма опроса отдельно для каждого из устройств можно с помощью команды ifconfig(8).

```
# ifconfig re0 polling
```

Точно так же, с помощью аргумента `-polling`, можно запретить использование этого механизма. Кроме того, команда ifconfig(8) показывает, был ли активирован механизм опроса для интерфейса.

Так как активация и деактивация механизма опроса возможны прямо в процессе работы системы, попробуйте активировать его в часы высокой нагрузки и проверить, не приведет ли это к повышению производительности.

Изменение размера окна

Для обслуживания входящих соединений система FreeBSD использует три буфера: *net.inet.tcp.sendspace*, *net.inet.tcp.recvspace* и *net.inet.udp.recvspace*. Гуру TCP/IP эти параметры настройки известны как *размер окна* (*window size*). При изменении размеров этих буферов изменяется и производительность сетевой подсистемы. Значения по умолчанию были выбраны не просто так: *они работают*. С увеличением размеров буферов увеличивается объем памяти ядра, которая должна выделяться для каждого соединения. Если одновременно не увеличить количество nmbclusters соответственным образом, может не хватить памяти для работы сетевых служб.

В версии FreeBSD 7.0 размеры этих буферов регулируются автоматически, в зависимости от сетевой нагрузки. В Интернете вам часто будут попадаться ссылки на информацию, описывающую порядок настройки этих параметров sysctl, но это лишь отзвуки многолетнего усовершенствования FreeBSD. В действительности вам не следует выполнять настройку этих параметров вручную.

Прочие способы оптимизации

В системе FreeBSD имеется порядка 150 параметров sysctl, которые имеют отношение к сетевой подсистеме. В ваших руках есть все необходимые инструменты, которые позволят оптимизировать систему до такой степени, что она вообще не будет пропускать трафик. Будьте крайне осторожны, экспериментируя с оптимизацией сетевой подсистемы. Многие параметры позволяют избавиться только от одного набора проблем, тут же принося другие. Некоторые производители программного обеспечения (например, Samba) рекомендуют вносить изменения в отдельные параметры настройки сети. Будьте осторожны с ними, прежде чем принять их как новые значения по умолчанию, обязательно проверяйте на появление побочных эффектов в других программах. TCP/IP очень, очень сложный протокол, и настройки по умолчанию, принятые в системе FreeBSD, отражают многолетний опыт тестирования и страданий системных администраторов.

Группировка сетевых адаптеров

Серверы сети становятся все более жизненно важными для бизнеса, и все более важной становится избыточность. У нас имеются избыточные жесткие диски на сервере и избыточная полоса пропускания в информационном центре, а как быть с избыточностью полосы пропускания на сервере? Операционная система FreeBSD может трактовать две сетевых карты как нечто единое целое, позволяя иметь несколько подключений к единственному коммутатору. Это обычно называется *группировкой сетевых адаптеров* (*network adapter teaming*). Группировка реализована в системе FreeBSD через lagg(4) – интерфейс агрегированного канала (link aggregation interface).

lagg(4) – это модуль ядра, который создает виртуальный интерфейс lagg0. Вы можете связать физические интерфейсы с интерфейсом lagg0, сделав их частью агрегированного канала. Существует возможность использовать lagg(4) и с единственным физическим интерфейсом, но создавать агрегированный канал имеет смысл только при наличии двух или более физических интерфейсов. lagg(4) позволяет реализовать бесшовный роуминг между проводными и беспроводными сетями, обеспечить устойчивость к отказам оборудования и поддержку несколько различных агрегатных протоколов.

Агрегатные протоколы

Не все сетевые коммутаторы поддерживают агрегатные протоколы. Система FreeBSD имеет базовую реализацию некоторых достаточно сложных и высокопроизводительных протоколов, а также включает основные параметры обеспечения отказоустойчивости. Я рекомендую три из них: Fast EtherChannel, LACP и failover. (Есть и другие схемы включения, о которых можно узнать в странице руководства lagg(4).)

Fast EtherChannel (*FEC*) – это агрегатный протокол, разработанный компанией Cisco, но он работает только с коммутаторами Cisco высокой и средней производительности, работающими под управлением операционной системы компании Cisco. Если у вас используется неуправляемый коммутатор, то вы не сможете использовать протокол Fast EtherChannel. Он очень сложен в настройке (на стороне коммутатора), поэтому я могу его порекомендовать только тем, у кого этот протокол используется как корпоративный стандарт.

Протокол управления агрегированным каналом (Link Aggregation Control Protocol, LACP) – это индустриальный стандарт построения агрегированных каналов. Физические интерфейсы объединяются в единый виртуальный интерфейс с пропускной способностью, приблизительно равной сумме пропускных способностей объединяемых физических интерфейсов. Протокол LACP обеспечивает превосходную отказоустойчивость, и практически все коммутаторы поддерживают его. Я рекомендую LACP, только если у вас отсутствует необходимость использовать Fast EtherChannel и ваш коммутатор способен обеспечить необходимую пропускную способность при использовании LACP.

Если ваш коммутатор «задыхается» при использовании LACP, используйте *failover*. При использовании метода failover трафик отправляется через один физический интерфейс. Если этот интерфейс выходит из строя, происходит переключение на следующий интерфейс из пула. Несмотря на то, что этот метод не дает увеличения пропускной способности, тем не менее вы получаете возможность подключить сервер к нескольким коммутаторам для повышения отказоустойчивости.

Так как LACP обычно является лучшим выбором, в наших примерах мы будем использовать именно его.

Настройка lagg(4)

Интерфейс lagg является виртуальным в том смысле, что в компьютере отсутствует физическое устройство, на которое можно было бы указать пальцем и сказать: «Это интерфейс lagg0». Прежде чем приступать к настройке, интерфейс сначала необходимо создать. FreeBSD позволяет создавать интерфейсы с помощью команды `ifconfig` *`intrfacename`* `create`, однако то же самое можно сделать с помощью оператора `cloned_interfaces` в файле */etc/rc.conf*.

Настройка lagg(4) в файле *rc.conf* производится в три этапа: создание интерфейса, запуск в работу физических интерфейсов и их агрегирование. В следующем примере из двух гигабитных сетевых карт Intel em0 и em1 создается единственный интерфейс lagg0.

```
cloned_interfaces="lagg0"
ifconfig_em0="up"
ifconfig_em1="up"
ifconfig_lagg0="laggproto lacp laggport em0 laggport em1 192.168.1.1 netmask
255.255.255.0"
```

Сначала указывается имитируемый интерфейс lagg0; FreeBSD создаст его во время загрузки. Затем производится запуск интерфейсов em0 и em1, при этом их настройка не производится. Наконец интерфейсу lagg0 сообщается, какой агрегатный протокол будет использоваться, какие физические интерфейсы будут ему принадлежать, а также сетевые параметры интерфейса. Эти несколько строк в конфигурационном файле обеспечат вас отказоустойчивым Ethernet-соединением.

Это было длинное путешествие через исследование сетевых возможностей и вы, вероятно, узнали даже больше, чем предполагали. Давайте немного постоим на месте и рассмотрим основы обеспечения безопасности системы.

7

Организация защиты системы

В защищенной системе доступ к ресурсам компьютера открыт лишь авторизованным пользователям в дозволенных целях. Даже если в системе нет важных данных, большое значение имеют время процессора, память и полоса пропускания. Вообще многие администраторы, считавшие свои системы маловажными и не заботившиеся об их защите, обнаруживали, что их машины невольно стали ретрансляторами (relays) – плацдармом для атак, наносящих урон целым корпорациям. Кому хочется проснуться утром от восхитительных звуков, сопровождающих появление в вашем доме сотрудников правоохранительных органов, потому что ваш незащищенный компьютер был использован для проникновения в банк.

Конечно, есть вещи похуже, чем подростки, захватившие власть над серверами, – скажем, перелом обеих ног. Однако вторая по значимости неприятность – однажды прийти на работу и обнаружить, что веб-страница компании теперь сообщает: «Ха-ха, здесь был Вася!» Еще более масштабные вторжения вызывают куда больше головной боли. Большинство вторжений, с которыми мне приходилось иметь дело (не в качестве злоумышленника, а в качестве консультанта, нанятого пострадавшей стороной), исходили из стран, где доступ к Интернету находится под цензурой правительства, а анализ трафика показал, что злоумышленники лишь искали неограниченный доступ к новостным сайтам. Хотя я и сочувствую этим людям, но когда речь заходит о стабильности работы моих серверов, подобные вторжения я считаю недопустимыми.

К сожалению, за последние несколько лет стало намного проще взламывать удаленные компьютеры. Программы с «мышиным» интерфейсом, позволяющие громить компьютеры, легко можно найти, зайдя на поисковые машины. Стоит только одному способному взломщику написать программу, вскрывающую уязвимые места в защите, как тысячи скучающих подростков не находят ничего лучшего, как загрузить этот код

и усложнить жизнь некоторым из нас. Даже если вас не заботит сохранность ваших данных, системные ресурсы все равно надо защищать.

Вообще говоря, взламываются не операционные системы, а программы, работающие в них. Даже изначально безопасная операционная система, самая надежная в мире, не может защитить плохо написанные программы от них самих. Иногда проблемы, заложенные в таких программах, накладываются на особенности операционной системы и подвергают ее нешуточному риску. Чаще всего это выражается в *переполнении буферов*, когда программа злоумышленника попадает прямо в область памяти, из которой процессор берет команды для выполнения в данный момент, а операционная система выполняет этот код. Система FreeBSD прошла всестороннее тестирование и доработку, призванную исключить возможность переполнения буферов, однако нет никакой гарантии, что такая возможность полностью устранена. Новые функции и программы появляются каждый день, а их взаимодействие с устоявшимися функциями может быть непредсказуемым.

Система FreeBSD имеет большое число инструментальных средств, которые помогут обезопасить систему и сделать ее неуязвимой для нападающих, как извне, так и изнутри. Ни один из этих инструментов не является достаточным для обеспечения надежной защиты, поэтому было бы желательно использовать их все. Все, что вы узнаете о безопасности системы, следует воспринимать как один из инструментов, а не как решение всех ваших проблем. Например, простое повышение уровня безопасности системы не сделает ее полностью безопасной. Этот шаг поможет только в комбинации с расстановкой прав доступа, флагами файлов, регулярным наложением «заплат», внимательным отношением к паролям и другими приемами, составляющими общую политику безопасности. Расширенные способы обеспечения безопасности будут рассматриваться в главе 9, но без базовых приемов защиты, которые рассматриваются здесь, эти способы не смогут обезопасить систему.

Кто враг?

Прежде всего, я произвольно распределю потенциальных взломщиков на четыре группы: малыши со скриптами, бот-сети, недовольные пользователи и опытные взломщики. Более точную градацию взломщиков можно найти в книгах, посвященных безопасности, но такие подробности не относятся к теме данной книги. Эти категории легко объяснить, легко понять; под них подпадают 99% всех взломщиков, с которыми вам, вероятно, придется столкнуться.

Малыши со скриптами

Наиболее многочисленная группа взломщиков – *малыши со скриптами* (*script kiddies*). Малыши со скриптами – не системные администраторы. Они не искусны. Они загружают маленькие программы с «мыши-

ным» интерфейсом, предназначенные для организации атак, и ищут потенциальные цели. Малыши со скриптами похожи на грабителей, которые охотятся за кошельками старых леди, держащих свои сумки слишком свободно. К счастью, от них защититься легче всего; для этого достаточно своевременно обновлять систему и программы на сервере, ставить на них заплатки. Они как саранча, их легко раздавить, но их *слишком много*!

Бот-сети

Бот-сети состоят компьютеров, зараженных червями или вирусами. Авторы вирусов контролируют бот-сети и используют их для всего, чего угодно, начиная от поиска дополнительных уязвимых хостов и рассылки спама, и до вторжения на защищенные сайты. Подавляющее большинство бот-сетей состоят из компьютеров, работающих под управлением операционной системы Windows, но это не говорит, что компьютеры, работающие под управлением UNIX-подобных систем, не могут быть ассимилированы в бот-сети. Например, в операционной системе Solaris 10 демон telnet (in.telnetd) имел уязвимость, которая впоследствии эксплуатировалась червем, вызывавшим хаос и разрушение в домашних каталогах пользователей.

К счастью, защититься от бот-сетей ничуть не сложнее, чем от малышей со скриптами, – достаточно своевременно обновлять систему и следовать общепринятым рекомендациям.

Недовольные пользователи

Третья группа, вызывающая большую часть проблем безопасности, – это ваши пользователи. И правда, недовольные служащие пробивают большинство брешей в защите, потому что они как никто другой знают уязвимости в системе безопасности, считают, что правила к ним не относятся, и имеют время на взлом системы. Если вы сообщите сотруднику, что политика компании запрещает ему доступ к ресурсам компьютера, а сам сотрудник будет считать, что он должен обладать доступом к этим ресурсам, он наверняка начнет искать возможность обойти установленные ограничения. Любой, кто считает себя особенным, и считает, что к нему правила не имеют отношения, представляет риск для безопасности системы. Вы можете на всех ваших серверах своевременно устанавливать заплатки, задать человеконенавистнические правила фильтрации трафика, но если в коммутационном шкафу есть модем, который позволяет любому желающему, знающему пароль, войти в сеть, то ждите неприятностей.

Лучший способ остановить таких людей – не быть сентиментальным. Когда сотрудник покидает компанию, удалите его учетную запись, измените все административные пароли и сообщите всем служащим о его уходе и о том, что с ним больше не надо делиться конфиденциальной информацией. Выработайте политику обеспечения безопасности

с действенными наказаниями и неуклонно проводите ее в жизнь. Избавьтесь от незащищенного модема, недокументированного telnet-сервера, доступного на порту с нестандартным номером, и любого другого серьезного инструмента, в спешке размещенного там, где его якобы никто не сможет найти.

Опытные взломщики

Последняя группа действительно опасна: опытные взломщики. Это квалифицированные системные администраторы, исследователи систем безопасности и специалисты по преодолению защиты, желающие заполучить специфические ресурсы. Проникновение в компьютерные системы в наше время превратилось в доходный преступный бизнес, особенно если у жертвы имеются ресурсы, достаточные для проведения распределенных атак типа «отказ в обслуживании» (Distributed Denial of Service, DDoS) или массовой рассылки спама. Взлом веб-сайта, состоящего из нескольких компьютеров, и превращение его в инструмент злоумышленника оплачивается достаточно высоко. Если у вас имеется важная секретная информация компании, вы можете стать целью для одного из таких злоумышленников. Если один из них *действительно* захочет проникнуть в систему, он, вероятно, это сделает.

Тем не менее надлежащие мероприятия по обеспечению защиты, которые остановят первые три группы злоумышленников, могут заставить

Хакеры, злоумышленники и другие негодяи

Нередко можно услышать, как слово *хакер* употребляется для описания людей, которые взламывают компьютерные системы. Это слово имеет различные значения в зависимости от того, кто его произносит. В техническом мире хакером называют человека, который проявляет интерес к внутренней работе технических систем. Одни хакеры интересуются буквально всем, другие имеют более узкую область интересов. В сообществе FreeBSD *хакер* – это знак уважения. Основной технический список рассылки называется *FreeBSD-hackers@FreeBSD.org*. Если вам действительно интересен термин *хакер*, прочитайте статью в словаре жаргонных слов, которая доступна по адресу: *http://www.catb.org/jargon/html/H/hacker.html*.

Я рекомендую во избежание путаницы вообще не употреблять это слово. В этой книге взломщики систем будут именоваться *злоумышленниками*.[1] Технических кудесников можно величать по-разному, но они редко возражают против обращения «О, Великий и Могучий!»

[1] Лично я даю им гораздо менее приятные имена.

опытных взломщиков изменить тактику. Вместо того чтобы проникать в компьютеры по сети, они будут вынуждены прикидываться ремонтниками телефонной компании и тайком устанавливать анализатор пакетов либо разгребать мусор в поисках листков с записанными паролями. В этом случае им придется быть у всех на виду, и они могут посчитать, что проникновение в систему не стоит возможных неприятностей. Если вам удастся заставить злоумышленника составлять план взлома, напоминающий голливудский сценарий, *независимо от того, насколько полной информацией о вашей сети он обладает*, значит вам удалось достаточно высоко поднять уровень безопасности.

Сообщения, относящиеся к безопасности FreeBSD

Лучший способ остановить всех взломщиков – своевременно обновлять систему. Другими словами, надо знать, когда обновлять систему, что именно обновлять и как обновлять. Устаревшие системы – это лучшие друзья малышей со скриптами.

В Проекте FreeBSD есть команда разработчиков, специализирующихся на аудите исходного кода и обеспечении безопасности как базовой операционной системы, так и добавляемого программного обеспечения. Эти разработчики поддерживают почтовую рассылку с небольшим трафиком, *FreeBSD-security-notifications@FreeBSD.org*, на которую стоит подписаться. Общие уведомления появляются и в других почтовых рассылках (таких как BugTraq и CERT), но рассылка *security-notifications* представляет собою единственный источник информации о проблемах обеспечения безопасности, относящейся к FreeBSD. Порядок оформления подписки на эту рассылку описывается на странице *http://lists.freebsd.org/mailman/listinfo*. Группа безопасности FreeBSD публикует свои рекомендации в этой рассылке сразу же, как только они будут выработаны.

Внимательно читайте эти рекомендации и действуйте незамедлительно, если они имеют к вам отношение, так как малыши со скриптами уже вышли на поиск уязвимых систем. Лучшее, что можно сделать, – это ликвидировать проблему как можно быстрее.

Безопасность и пользователи

Помните, я говорил, что наибольшую опасность представляют ваши собственные пользователи? В этом разделе вы узнаете, как заставить этих маленьких детишек следовать установленным правилам. В операционной системе FreeBSD имеется большое разнообразие средств, которые позволят пользователям выполнять свою работу, но не дадут им безраздельно властвовать над системой. Здесь мы рассмотрим самые важные инструменты, начав с тех, которые позволяют добавлять новых пользователей.

Создание учетной записи пользователя

В операционной системе FreeBSD используются стандартные для UNIX программы управления пользователями, такие как passwd(1), pw(8) и vipw(8). Кроме того, FreeBSD включает в себя удобную интерактивную программу adduser(8), которая позволяет добавлять новых пользователей. Разумеется, создать новую учетную запись может только пользователь root. Просто введите в командной строке команду `adduser`, и вы попадете в интерактивную оболочку.

На первом запуске программа adduser(8) попросит установить соответствующие значения по умолчанию, которые будут использоваться при создании всех новых учетных записей. Следуя за примером сеанса работы с программой, который приводится ниже, вы сможете определить значения по умолчанию для своей системы.

```
# adduser
❶ Username: gedonner
❷ Full name: Gregory E Donner
❸ Uid (Leave empty for default):
```

Здесь username ❶ – это имя учетной записи. Имена учетных записей в моей системе составляются из инициалов и фамилии пользователя. Вы можете выбирать имена для учетных записей, исходя из любой другой схемы. Full name ❷ – это настоящее имя пользователя. Далее FreeBSD предложит выбрать числовой идентификатор пользователя (UID) ❸. Нумерация идентификаторов пользователей в системе FreeBSD начинается с числа 1000, хотя вы можете выбрать любое число. Все идентификаторы, лежащие ниже 1000, зарезервированы для нужд системы. В этом месте я рекомендую просто нажать клавишу Enter, чтобы выбрать первый незанятый UID.

```
❶ Login group [gedonner]:
❷ Login group is gedonner. Invite gedonner into other groups? []: webmasters
❸ Login class [default]:
❹ Shell (sh csh tcsh nologin) [sh]: tcsh
❺ Home directory [/home/gedonner]:
```

Группа пользователя по умолчанию ❶ – это очень важно, не забывайте, что права доступа в UNIX основаны на принадлежности владельцу и группе. По умолчанию FreeBSD создает для каждого пользователя отдельную группу, что наиболее предпочтительно для большинства систем. Во всех толстых книгах по системному администрированию предлагается несколько схем группировки – вы можете использовать ту из них, которая окажется наиболее близка к вашим потребностям. Помимо группы по умолчанию, пользователя можно добавить и в другие группы ❷, если в этом есть необходимость.

Login class (класс доступа) ❸ определяет, к каким ресурсам имеет доступ создаваемый пользователь. Подробнее о классах доступа мы поговорим немного ниже, в этом же разделе.

Shell (оболочка) ❹ – это интерпретатор команд. Хотя в системе по умолчанию выбирается интерпретатор */bin/sh*, я предпочитаю для начинающих пользователей выбирать интерпретатор команд tcsh.[1] Если у вас имеется глубокая привязанность к другому интерпретатору – выбирайте его. Опытные пользователи могут самостоятельно изменить выбор интерпретатора команд.[2]

Home directory (домашний каталог) ❺ – это каталог на диске, где будут находиться файлы пользователя. Владельцами этого каталога будут назначены сам пользователь и группа по умолчанию.

```
❶ Use password-based authentication? [yes]:
❷ Use an empty password? (yes/no) [no]:
❸ Use a random password? (yes/no) [no]: y
❹ Lock out the account after creation? [no]: n
```

Использование паролей дает определенную степень гибкости. Если все ваши пользователи знают, что такое SSH и общедоступные ключи, возможно, вам удастся обойтись без паролей. А пока большинство из нас предпочитают использовать пароли ❶.

Use an empty password (использовать пустой пароль) ❷ – эта возможность предусмотрена на тот случай, если необходимо, чтобы пользователь сам назначил себе пароль. Любому, кто попытается подключиться к системе с указанной учетной записью, будет предложено сначала установить пароль. Это делает идею оставить пароль пустым похожей на мысль заполнить дирижабль водородом, а потом зажечь спичку внутри него.

С другой стороны, отличным выбором будет назначение случайного пароля (random password) ❸ для новой учетной записи. Генератор случайных паролей в системе FreeBSD неплохо подходит для повседневного использования. Случайные пароли очень сложны для запоминания, и это лишний раз подталкивает пользователей поскорее изменить его.

Если учетная запись заблокирована (locked) ❹, никто не сможет использовать ее для входа в систему. Вообще говоря – это неэффективно.

После ввода всей необходимой информации программа adduser повторно выведет ее на экран, чтобы вы могли подтвердить или отменить

---

[1] Для интерактивной работы – да. Но никогда, никогда, *никогда* не используйте C-подобные командные интерпретаторы для написания программ. Прочитайте классический труд Тома Кристиансена (Tom Christiansen) «Csh Programming Considered Harmful», где вы найдете подробные объяснения. (Найти его можно по адресу: *http://www.faqs.org/faqs/unix-faq/shell/csh-whynot*)

[2] По умолчанию во многих версиях FreeBSD и так устанавливается */bin/tcsh*, хотя намного удобнее работать в bash, и поэтому после установки системы имеет смысл с помощью adduser установить bash в качестве интерпретатора команд по умолчанию для всех будущих пользователей. – *Прим. научн. ред.*

ее. Как только будет получено подтверждение, adduser проверит параметры учетной записи и выведет пароль, сгенерированный случайным образом. Затем программа предложит создать другую учетную запись.

Настройка adduser: /etc/adduser.conf

Процедура создания новых учетных записей в некоторых версиях UNIX сопряжена с необходимостью вручную редактировать файл */etc/passwd*, перестраивать базу паролей, редактировать файл */etc/group*, создавать домашний каталог, устанавливать права доступа к этому каталогу, устанавливать скрытые файлы (имена которых начинаются с символа «точка» (.)) и т. д. Все это вынуждает вас выработать свою собственную процедуру настройки – если все параметры устанавливаются вручную, вы легко можете управлять своими локальными учетными записями. adduser(8) сама выполняет большую часть рутинных операций и использует некоторые значения по умолчанию. Для серверов с различными требованиями вы можете определить различные значения по умолчанию в файле */etc/adduser.conf*, что позволит обеспечить соответствие предъявляемым требованиям и одновременно сохранить высокий уровень автоматизации.

Чтобы создать свой первый файл *adduser.conf*, нужно запустить команду `adduser -C` и ответить на ряд вопросов.

```
❶ Login group []:
❷ Enter additional groups []: cvsup
❸ Login class [default]:
❹ Shell (sh csh tcsh nologin) [sh]: tcsh
❺ Home directory [/home/]: /nfs/u1/home
❻ Use password-based authentication? [yes]:
Use an empty password? (yes/no) [no]:
Use a random password? (yes/no) [no]: yes
Lock out the account after creation? [no]: no
```

Login group ❶ – это группа пользователя по умолчанию. Пустое значение означает, что в качестве группы по умолчанию пользователю будет назначаться его собственная группа (принято в FreeBSD по умолчанию).

Существует возможность указать дополнительные группы ❷, к которым по умолчанию будет принадлежать новая учетная запись, а также класс доступа ❸.

Укажите командный интерпретатор по умолчанию ❹ для своих пользователей.

Ваш выбор местоположения домашних каталогов ❺ пользователей может отличаться от устоявшегося стандарта, принятого в системе FreeBSD. В данном примере я указал, что домашние каталоги пользователей будут размещаться на разделе NFS, где находятся домашние каталоги пользователей, имеющих учетные записи на нескольких компьютерах.

В заключение определяется порядок установки пароля ❻ для нового пользователя.

Эти параметры по умолчанию помогут уберечь вас от ручного ввода с клавиатуры, но как только будет создан файл с базовыми настройками, у вас появляется возможность добавить в него более изощренные функции. В табл. 7.1 приводятся дополнительные параметры для файла *adduser.conf*, которые я считаю наиболее полезными.

Таблица 7.1. Полезные параметры настройки для файла adduser.conf

Параметр	Назначение
`defaultGroup`	Имя группы по умолчанию, к которой будут добавляться новые пользователи (если значение не определено, для каждого пользователя будет создаваться его собственная группа)
`defaultclass`	Класс доступа по умолчанию
`passwdtype`	Может иметь значения `no` (учетная запись останется заблокированной, пока root не назначит пароль), `none` (пароль не установлен), `yes` (пароль устанавливается при создании учетной записи) или `random` (будет назначен случайный пароль)
`homeprefix`	Каталог, где будут размещаться домашние каталоги пользователей (например, */home*)
`defaultshell`	Командная оболочка, назначаемая по умолчанию (здесь можно указать любую командную оболочку из */etc/shells*)
`udotdir`	Каталог, где находятся заготовки файлов пользователя, имена которых начинаются с символа «точка»
`msgfile`	Файл, содержащий текст электронного письма, отправляемого каждому пользователю сразу после создания учетной записи.

Используя эти параметры, вы сможете изменять поведение по умолчанию программы adduser, чтобы оно наиболее точно соответствовало вашим потребностям.

Изменение учетных записей: passwd(1), chpass(1) и другие

Управление пользователями заключается не только в создании и удалении учетных записей. Время от времени эти учетные записи необходимо изменять. Система FreeBSD включает в себя некоторые инструменты, предназначенные для редактирования учетных записей, самыми простыми из которых являются passwd(1), chpass(1), vipw(8) и pw(8). Все они работают с тесно взаимосвязанными файлами */etc/master.passwd*, */etc/passwd*, */etc/spwd.db* и */etc/pwd.db*. Для начала мы рассмотрим эти файлы, а затем перейдем к обзору указанных выше инструментов.

Файлы */etc/master.passwd*, */etc/passwd*, */etc/spwd.db* и */etc/pwd.db* хранят информацию об учетных записях пользователей. Каждый файл имеет свой формат и свое назначение. Файл */etc/master.passwd* является

источником информации для аутентификации и содержит пароли пользователей в зашифрованном виде. У обычных пользователей нет прав для просмотра содержимого файла */etc/master.passwd*. Однако им должна быть доступна основная информация, хранящаяся в учетных записях, иначе как непривилегированные программы смогут идентифицировать пользователя? В файле */etc/passwd* перечислены все учетные записи без привилегированной информации (например, без зашифрованных паролей). Содержимое этого файла доступно для чтения любому пользователю, оттуда он может извлекать основные сведения об учетной записи.

Информация из учетной записи требуется многим программам, а синтаксический анализ текстовых файлов, как известно, – процедура достаточно медленная. В наши дни мощных ноутбуков слово *медленная* теряет свой смысл, но это было насущной проблемой во времена, когда стиль диско шагал по Земле. По этой причине в системах BSD появились файлы базы данных, которые создаются на основе */etc/master.passwd* и */etc/passwd*. (Другие UNIX-подобные системы обладают похожей функциональностью, реализованной на основе других файлов.) Файл */etc/spwd.db* создается непосредственно из */etc/master.passwd* и содержит секретную информацию о пользователях, этот файл доступен для чтения только пользователю root. Файл */etc/pwd.db* доступен для чтения всем пользователям, но содержит ограниченный набор сведений, содержащихся в файле */etc/passwd*.

Всякий раз, когда какая-либо программа управления пользователями изменяет информацию об учетной записи, хранящейся в файле */etc/master.passwd*, FreeBSD запускает pwd_mkdb(8) для обновления трех других файлов. Например, все три программы, passwd(1), chpass(1) и vipw(8), позволяют вносить изменения в основной файл с паролями и все три программы вызывают pwd_mkdb(8) для обновления информации во взаимосвязанных файлах.

Изменение пароля

Для изменения пароля используется программа passwd(1). Пользователи могут изменять свои собственные пароли, а пользователь root имеет право изменять пароли любых пользователей. Чтобы изменить свой собственный пароль, достаточно просто ввести в строке приглашения к вводу команду `passwd`.

```
# passwd
Changing local password for mwlucas
Old Password:
New Password:
Retype New Password:
```

Если вы изменяете свой собственный пароль, passwd(1) сначала потребует ввести текущий пароль. Сделано это для того, чтобы никто другой не смог изменить пароль без ведома самого пользователя. Вообще, ко-

гда вы покидаете терминал, было бы желательно всегда завершать сеанс работы с системой, но даже если вы этого не делаете, эта простая проверка, которую проводит passwd(1), обезопасит вас от проделок шутников. После этого нужно будет дважды ввести новый пароль и все. Если вы обладаете привилегиями суперпользователя и вам требуется изменить пароль другого пользователя, просто передайте программе passwd имя этого пользователя в виде аргумента.

```
# passwd mwlucas
Changing local password for mwlucas
New Password:
Retype New Password:
```

Обратите внимание: пользователю root необязательно знать старый пароль другого пользователя – пользователь root может изменить учетную информацию любого пользователя в системе, как ему заблагорассудится.

Управление пользователями и переменная $EDITOR

Такие инструменты управления пользователями, как chpass и vipw (как и многие другие инструменты администрирования системы), запускают текстовый редактор, когда возникает необходимость внести изменения. Вообще говоря, эти программы определяют предпочтительную программу текстового редактора, исходя из содержимого переменной окружения `$EDITOR`. С помощью этой переменной можно назначить редактором по умолчанию vi, Emacs или какой-нибудь другой имеющийся редактор. Я рекомендую использовать Vigor (*/usr/ports/editors/vigor*), клон редактора vi(1) с анимированным помощником в виде скрепки, благодаря которому пользователи, привыкшие к Microsoft Office, будут чувствовать себя увереннее.

Изменение учетных записей с помощью chpass(1)

В учетной записи хранится не только пароль, но и масса другой информации. Утилита chpass(1) дает пользователям возможность редактировать все доступные для их учетной записи сведения. Например, если chpass запустить из командной строки, откроется текстовый редактор со следующим содержимым:

```
#Changing user information for mwlucas.
Shell: /bin/tcsh
Full Name: Michael W Lucas
Office Location:
Office Phone:
```

```
Home Phone:
Other information:
```

В данном случае мне разрешено редактировать шесть информационных полей, входящих в мою учетную запись. В первом поле, оболочка (Shell), можно установить любой командный интерпретатор из перечисленных в файле */etc/shells* (раздел «Интерпретаторы команд и /etc/shells» на стр. 244). Я могу изменить свое полное имя (Full Name), если, к примеру, мне захочется указать свое отчество полностью или я захочу, чтобы другие пользователи системы знали меня как *мистера Зануду*. Я могу изменить адрес своего офиса (Office Location) и номер рабочего телефона (Office Phone), по которым мои коллеги смогут меня отыскать. Это одна из особенностей, которая была полезна в университетских городках, где выросла BSD и где пользователи системы почти не знали о физическом местоположении других пользователей. Теперь, когда в нашем распоряжении имеются электронные каталоги и масса компьютеров, это поле перестало быть таким полезным. Обычно в поле с номером домашнего телефона (Home Phone) я указываю номер 911 (999 – в Великобритании), а в поле с прочей информацией (Other) могу внести некоторые дополнительные сведения. Кроме того, следует отметить, какую информацию я *не могу* изменять, будучи обычным пользователем. Местоположение моего домашнего каталога определяется системным администратором, и я не могу изменить его, даже если в систему был установлен новый жесткий диск огромного объема, который так пригодился бы для размещения моих файлов MP3. Аналогичным образом мои числовые идентификаторы пользователя и группы (UID и GID) присваиваются системой или системным администратором.

С другой стороны, если пользователь root запустит команду `chpass mwlucas`, его привилегированное положение позволит увидеть иную картину.

```
#Changing user information for mwlucas.
Login: mwlucas
❶ Password: $1$4d.nOPmc$uuBQy6ZL6hPQNTQef1jty.
Uid [#]: 1001
Gid [# or name]: 1001
Change [month day year]:
Expire [month day year]:
Class:
Home directory: /home/mwlucas
Shell: /bin/tcsh
Full Name: Michael W Lucas
Office Location:
Office Phone:
Home Phone:
Other information:
```

Как суперпользователь вы сможете сделать с бедным пользователем все, что угодно. Изменение имени учетной записи на *megaloser* – это только начало. Вы даже имеете доступ к зашифрованному паролю пользователя ❶. Не изменяйте его значение, если вы не в состоянии

производить шифрование в уме. Программа passwd(1) позволяет сделать это более безопасно. Вам будет даже позволено изменить местоположение домашнего каталога пользователя, при этом следует помнить, что chpass(1) не выполняет перемещение файлов пользователя – вам придется сделать это вручную.

Вы также можете установить дату изменения пароля и срок действия учетной записи. Срок действия пароля (Change) удобно определять в случаях, когда необходимо вынудить пользователя задать свой пароль при первом входе в систему. Срок действия учетной записи (Expire) удобно назначать, когда кто-то просит создать ему учетную запись на ограниченный период времени. Вы можете забыть вовремя удалить эту учетную запись, но FreeBSD никогда не забудет. В обоих случаях требуется указать дату в формате *месяц день год*, при этом следует указывать только первые три символа названия месяца. Например, чтобы пароль пользователя перестал действовать 8 июня 2008 года, я мог бы ввести в поле `Change` значение `Jun 8 2008`. Как только пользователь изменит свой пароль, поле срока действия пароля будет очищено, а что касается срока действия учетной записи, то только системный администратор сможет продлить его.

Большая кувалда: vipw(8)

Программу chpass удобно использовать для редактирования отдельных учетных записей, а как быть, если возникает необходимость отредактировать сразу множество записей? Предположим, что в системе имеются сотни пользователей, и в нее был установлен новый жесткий диск, предназначенный специально для размещения домашних каталогов, – неужели у вас хватит терпения запускать chpass(1) несколько сотен раз? Это тот случай, когда на помощь приходит программа vipw(8).

vipw позволяет редактировать непосредственно файл */etc/master.passwd*. По окончании внесения изменений vipw проверит синтаксис файла паролей, чтобы убедиться, что вы ничего не разрушили, затем сохранит обновленный файл паролей и запустит pwd_mkdb(8). vipw может защитить ваш файл паролей от массы досадных ошибок, но не стоит выпускать процесс из-под контроля. Чтобы должным образом использовать vipw(8), вам необходимо знать формат файла паролей.

Если информация в файле */etc/master.passwd* не совпадает с информацией в других файлах, программа предполагает, что верная информация находится в файле */etc/master.passwd*. Например, в файле */etc/group* отсутствуют группы, которые определены как основные группы пользователей. Группа, указанная в качестве основной в файле */etc/master.passwd*, считается правильной, даже если она отсутствует в файле */etc/group*.

Каждой учетной записи соответствует отдельная строка в файле */etc/master.passwd*, которая состоит из 10 полей, разделенных двоеточиями. Это следующие поля:

Имя учетной записи

Это или имя учетной записи, созданной системным администратором, или имя учетной записи, созданной во время установки какой-либо системной службы. Система FreeBSD включает такие имена учетных записей для нужд системного администрирования, как root, daemon, games и т. д. Каждая из таких учетных записей является владельцем некоторой части основной системы. В этот перечень входят также учетные записи для таких наиболее распространенных служб, как пользователь www, необходимый для работы веб-сервера. Дополнительное программное обеспечение может добавлять собственные учетные записи.

Зашифрованный пароль

Второе поле в строке – это пароль в зашифрованном виде. Системные пользователи не имеют пароля, поэтому вы не сможете войти в систему под одной из таких учетных записей. Для обычных пользователей это поле содержит случайные, на первый взгляд, последовательности символов.

Числовой идентификатор пользователя (UID)

Третье поле – это *числовой идентификатор пользователя*, или *UID*. Каждый пользователь имеет свой, уникальный UID.

Числовой идентификатор группы (GID)

Четвертое поле – это *числовой идентификатор группы*, или *GID*. В этом поле приводится идентификатор основной группы пользователя. Обычно он совпадает с UID, а сама группа имеет то же имя, что и учетная запись пользователя.

Класс доступа

Следующее поле – это класс доступа, определение которого находится в файле */etc/login.conf* (раздел «Ограничение на использование ресурсов системы» на стр. 255).

Срок действия пароля

Это то же самое поле срока действия пароля, что и в программе chpass(1), однако здесь время хранится в виде числа секунд, прошедших с начала времен. Для преобразования реальной даты в секунды можно использовать команду `date -j` с форматом вывода `+%s`. Чтобы перевести полночь 1 июня 2008 года в число секунд, прошедших от начала эпохи, можно воспользоваться командой `date -j 200806010000 '+%s'`.

Срок действия учетной записи

Чтобы заблокировать учетную запись в заданный день, нужно установить значение этого поля так же, как если бы это было поле, определяющее срок действия пароля.

Личные данные

Это поле известно также под названием *gecos* по малопонятным историческим причинам. Это поле содержит действительное имя пользователя, адрес офиса, рабочий телефон и домашний телефон. Все элементы этого поля отделяются друг от друга запятыми. Не используйте двоеточия внутри этого поля – двоеточия используются для разделения полей самой записи из */etc/master.passwd*.

Домашний каталог пользователя

Девятое поле – это домашний каталог пользователя. По умолчанию в это поле записывается значение в формате */home/<username>*, но вы можете указать любой другой каталог по своему усмотрению. Кроме того, при изменении этого поля вам потребуется вручную переместить файлы пользователя в новый каталог. Пользователи с несуществующим домашним каталогом по умолчанию не могут заходить в систему, хотя такое поведение можно изменить с помощью параметра настройки `requirehome` в файле *login.conf*.

Командный интерпретатор

Последнее поле – это командный интерпретатор пользователя. Если это поле оставить пустым, система будет предоставлять старый, скучный интерпретатор */bin/sh*.

Если программа chpass(1) дает вам возможность калечить отдельные учетные записи, то vipw(8) вверяет вам в руки всю базу данных о пользователях. Будьте внимательны при работе с ней!

Удаление пользователя

Учетные записи пользователей удаляются с помощью программы rmuser(8). Программа попросит ввести имя учетной записи, которую необходимо удалить, и спросит, следует ли удалять домашний каталог пользователя. Вот и все, что необходимо сделать, – разрушать всегда легче, чем созидать.

Использование pw(8) в сценариях

Команда pw(8) предоставляет мощный интерфейс командной строки для доступа к ученым записям пользователей. В то время как программа useradd(8) проведет вас по всем этапам создания учетной записи в диалоговом режиме, pw(8) позволит выполнить все необходимые действия одной-единственной командой. Я считаю, что pw(8) слишком громоздка для повседневного использования, но если вам приходится администрировать большое число учетных записей, ее помощь будет неоценима.

Я часто использую pw(8) для блокировки учетных записей. Хотя заблокированная учетная запись остается активной, никто не может войти с ней в систему. Я использую эту замечательную возможность, когда сумма на счете клиента опускается ниже нуля, – пользователи

очень быстро реагируют на отсутствие возможности войти в систему, даже при том, что их веб-сайты продолжают работать, а электронная почта продолжает поступать.

```
# pw lock mwlucas
```

Разблокировка учетной записи производится командой `pw unlock` *username*.

Если вам потребуется писать сценарии для управления своими пользователями, вам определенно стоит прочитать страницу руководства pw(8).

Интерпретаторы команд и /etc/shells

Интерпретатор команд (*shell*) – это программа, которая предоставляет в распоряжение пользователя командную строку. Различные интерпретаторы команд отличаются поведением, поддерживают различные комбинации клавиш для быстрого вызова и предоставляют различные функциональные возможности. Многие пользователи привязаны к определенным командным интерпретаторам и выражают свое неудовольствие, если любимый командный интерпретатор отсутствует в системе. Вы можете выполнить установку большого числа командных интерпретаторов из «портов» (глава 11).

Файл */etc/shells* содержит список командных интерпретаторов, доступных пользователю. При установке командного интерпретатора из «порта» или «пакета» в */etc/shells* будет добавлена соответствующая запись. Однако если не применять «порт», а компилировать исходный код, то в этот файл необходимо будет добавить полный путь к исполняемому файлу командного интерпретатора.

Демон FTP отклонит подключение пользователя, командный процессор которого не перечислен в */etc/shells*. Если вы используете */sbin/nologin* в качестве пользовательского командного процессора исключительно для доступа по FTP, то вы должны эту программу добавить в */etc/shells*. Однако более правильный путь состоит в том, чтобы управлять такими пользователями через классы доступа, о чем будет рассказываться далее в этой главе.

root, группы и права доступа

Система безопасности UNIX считается в некотором смысле грубой, поскольку один суперпользователь, *root*, может делать все, что угодно. Другие пользователи – безропотные пеоны, покорно носящие кандалы, в которые их заковал root. Проблема в том, что у суперпользователя не такой уж широкий выбор кандалов, и он не может подбирать их индивидуально для каждого пользователя. Это в общем так, но приличный администратор может комбинированием групп и прав доступа решать почти все вопросы безопасности.

Пароль root

Для некоторых действий нужен полный контроль над системой, в том числе возможность манипулировать ключевыми системными файлами, такими как файлы ядра, драйверов устройств и файлы системы аутентификации. Учетная запись root разработана как раз для выполнения таких действий.

Для использования пароля root надо либо войти в систему с консоли как root, либо, если вы – член группы wheel, выполнить команду смены пользователя (switch user) su(1). (Группы будут обсуждаться ниже, в этом же разделе.) Я рекомендую su; информация о пользователях, выполнивших su, заносится в протокол. Кроме того, su можно применять на удаленной системе. Выполнить эту команду очень просто:

```
# su
Password:
#
```

Проверьте текущее имя пользователя с помощью команды id(1):

```
# id
uid=0(root) gid=0(wheel) groups=0(wheel), 5(operator)
#
```

Теперь вы владеете системой – да, она *в вашей власти*. Будьте внимательны при каждом нажатии клавиши; небрежность может превратить жесткий диск в неформатированный кусок металла – такой же чистый, каким он был изначально. Если уж использовать пароль root, то ответственно, ибо каждый, кто знает пароль root, может причинить непоправимый вред системе.

Не забывайте, только пользователи из группы wheel могут воспользоваться паролем root, чтобы получить привилегии этого пользователя с помощью команды su(1). И любой может воспользоваться паролем root с консоли, вот почему так важно обеспечить физическую защиту системы. Если пароль root попадет в руки обычного пользователя, не имеющего физического доступа к консоли, он может вводить команду `su`, пока не надоест, – команда все равно работать не будет.

Естественно, возникает вопрос: «А кому нужен доступ с правами root?» Для большинства действий по конфигурированию, обсуждаемых в этой книге, необходим пароль root. Но как только система настроена должным образом, доступ к паролю root можно значительно сократить и даже приостановить. Для решения остальных задач, которые требуют привилегий пользователя root, я рекомендую использовать sudo (*/usr/ports/security/sudo*). Один из самых простых способов уменьшить необходимость передачи прав root состоит в правильном использовании групп.

Группы пользователей

В UNIX пользователи классифицируются по *группам*, в каждую из которых входят пользователи, выполняющие сходные административные функции. Системный администратор может определить группу с именем *webmasters*, добавить в нее учетные записи пользователей, редактирующих веб-страницы, и определить права доступа к соответствующим файлам так, чтобы члены группы могли редактировать эти файлы. Он может также создать группу *email*, добавить в нее учетные записи администраторов, управляющих почтовым сервером, и соответствующим образом определить права доступа к файлам, имеющим отношение к электронной почте. Такое использование групп представляет собой мощный инструмент управления системой.

Любой пользователь может определить свою принадлежность к группам с помощью команды id(1). В предыдущем примере видно, что пользователь root входит в состав групп wheel и operator. Однако root – это специальный пользователь, он может делать все, что пожелает. Ниже приводится моя учетная запись, более похожая на запись обычного среднего пользователя:

```
# id
uid=1001(mwlucas) gid=1001(mwlucas) groups=1001(mwlucas), 0(wheel),
68(dialer), 1006(cvsup)
```

Мой идентификатор пользователя (UID) равен 1001, а имя учетной записи – mwlucas. Идентификатор моей основной группы (GID) равен 1001, а моя основная группа также называется mwlucas. Это типичный пример первого созданного пользователя, но и последующие пользователи будут отличаться друг от друга только числовыми идентификаторами пользователя и группы. Гораздо интереснее узнать, для чего я назначил остальные группы: помимо своей основной группы я вхожу также в группы wheel, dialer и cvsup. Члены группы wheel могут использовать пароль root, чтобы получить привилегии root, члены группы dialer могут пользоваться программой tip(1) без необходимости получать права суперпользователя, а члены группы cvsup могут пользоваться репозиторием CVS в локальной системе. В моей системе каждая из этих групп имеет особые привилегии и как член этих групп, я наследую эти привилегии.

Информация о группах находится в файле */etc/group*.

/etc/group

Файл */etc/group* содержит информацию о всех группах, за исключением основных групп пользователей (которые определяются вместе с учетными записями в файле */etc/master.passwd*). Каждая строка в файле */etc/group* содержит четыре поля, разделенных двоеточиями: имя группы, пароль группы, числовой идентификатор группы и список членов группы. Ниже приводится пример записи:

```
wheel:*:0:root,mwlucas,gedonner
```

Имя группы – это понятное для человека название группы. Данная группа называется wheel. Имена групп могут быть совершенно произвольными. При желании группу можно назвать, например, minions (любимчики). Впрочем, имена для групп лучше выбирать, исходя из их предназначения. Вы можете запомнить, что члены группы minions могут редактировать веб-страницы, но догадаются ли об этом ваши коллеги?

Второе поле содержит пароль группы. Пароли групп способствовали укоренению плохих, с точки зрения безопасности, привычек, поэтому в большинстве современных систем UNIX они не поддерживаются. Однако прежние программы написаны в расчете на присутствие этого поля, поэтому вместо того, чтобы оставить это поле пустым или удалить его, используйте звездочку (*) в качестве «заполнителя».

Третье поле содержит уникальный числовой идентификатор группы (GID). Для идентификации групп в большинстве внутренних программ FreeBSD применяются именно GID, а не имена. Группа wheel имеет числовой идентификатор, равный 0, а максимальное значение GID равно 65 535.

Последнее поле – список всех пользователей, входящих в эту группу. Имена пользователей разделяются запятыми. В этом примере в состав группы wheel входят пользователи root, mwlucas и gedonner.

Изменение состава групп

Для добавления пользователя в группу добавьте имя его учетной записи в конец строки с определением требуемой группы. Например, в группу wheel входят пользователи, которые обладают правом пользоваться паролем root. В следующем примере я добавил пользователя rwatson в группу wheel:

```
wheel:*:0:root,mwlucas,gedonner,rwatson
```

Конечно, мои шансы убедить пользователя rwatson (президент фонда FreeBSD Foundation) взять на себя обязанности по администрированию любой из моих систем находятся в диапазоне от ничтожно малых до вообще никаких, но попробовать стоит.

Создание групп

Чтобы создать новую группу, достаточно иметь имя группы и числовой идентификатор группы. Технически вам даже не нужно иметь членов группы – некоторые программы работают с правами группы, и сама система FreeBSD использует привилегии группы для управления такими программами точно так же, как это делают пользователи.

Традиционно в качестве значения GID выбирается первое свободное число, превышающее максимальное значение GID в списке имеющихся групп. Вообще говоря, числовые идентификаторы групп со значе-

ниями ниже 1000 зарезервированы для нужд операционной системы. Программы, которым требуется отдельное значение GID, обычно используют одно из чисел в этом диапазоне. Значения идентификаторов групп для учетных записей пользователей начинаются с числа 1001. Идентификаторы некоторых специализированных групп начинают получать значения от 65 535 и ниже.

Использование групп во избежание передачи пароля root

Кроме проблемы обеспечения безопасности, передача пароля суперпользователя может вызвать разногласия в любой организации. Многие системные администраторы не торопятся передавать пароль root тем, кто отвечает за сопровождение отдельных частей системы, но, не предлагая альтернативы, они тем самым препятствуют тому, чтобы люди выполняли свои обязанности. Другие системные администраторы бесконтрольно раздают пароль root почти всем желающим, а потом жалуются, что система стала нестабильной. В конечном итоге оба варианта малопригодны. Будучи пользователем, я совсем не претендую на получение пароля root, но я настаиваю, чтобы системный администратор создал группу, которая обладала бы достаточными привилегиями для решения поставленных задач. Хотя это довольно удобно – обладать привилегиями суперпользователя, тем не менее не нести ответственность за нарушения в работе системы еще удобнее.

Одна из типичных ситуаций – когда младший системный администратор отвечает за некоторую часть системы. В моем подчинении было много администраторов DNS[1] – они никогда не устанавливали программное обеспечение, не пересобирали ядро и не выполняли другие обязанности системного администратора. Они лишь отвечали на электронные письма, обновляли файлы зон и перезапускали демон named. Начинающие системные администраторы часто считают, что им необходим пароль root для выполнения работ такого рода. Создав группы для тех, кто выполняет сходные административные функции, вы избежите бесконтрольной передачи пароля root и позволите людям выполнять свою работу. В этом разделе мы реализуем на основе групп управление доступом к файлам сервера имен. Те же самые принципы применимы к любым файлам, которые требуется обезопасить. Конфигурационные файлы веб-сервера и сервера электронной почты – другой наиболее вероятный кандидат для организации управления доступом на основе групп.

Системные учетные записи

В операционной системе FreeBSD некоторые учетные записи резервируются для встроенных программ. Например, сервер имен работает под учетной записью с именем bind и в группе bind. Не нужно входить

[1] Некоторым даже удалось выжить.

в систему под учетной записью программного пользователя, чтобы выполнять такого рода работу! Если злоумышленник взломает сервер имен, он получит доступ к системе всего лишь с привилегиями пользователя bind.

Более того, не допускайте, чтобы системные учетные записи и системные группы выступали владельцами файлов, созданных специально для таких программ. Создайте отдельного пользователя и группу, которые будут владеть файлами программ. В этом случае наш гипотетический взломщик сервера имен не сможет даже отредактировать файлы, которые используются сервером DNS, благодаря чему сводятся к минимуму возможные разрушения. Если программа регулярно обновляет файлы (например, файлы базы данных), то необходимо дать программе право на запись, но при этом вполне возможно, что человеку никогда не потребуется редактировать такие файлы. Точно так же нет никаких причин, по которым сервер базы данных должен иметь право на запись в свои собственные конфигурационные файлы.

Создание административных групп

Самый простой способ создать группу, которая будет владеть файлами, – это создать нового пользователя с помощью adduser(8) и затем использовать основную группу этого пользователя для назначения выбранным файлам привилегий этой группы. Так как у нас уже есть пользователь bind, мы создадим административного пользователя dns. Имя учетной записи не играет большой роли, но лучше выбирать такие имена, которые лучше запоминаются.

В качестве командной оболочки для этого пользователя укажите *nologin*, что соответствует интерпретатору */sbin/nologin*. Это предотвратит попытки войти в систему под этой учетной записью.

При желании для пользователей этой категории можно выбирать определенные значения UID и GID. Я выбираю такие значения UID и GID, которые напоминали бы числовые идентификаторы учетных записей, используемых соответствующими системными службами. Например, UID и GID пользователя bind имеют значение 53. Для простоты запоминания я мог бы присвоить UID пользователя dns значение 10053. В других случаях я присваиваю административным группам числовые идентификаторы, начиная с 65535 и далее вниз.

Не добавляйте таких административных пользователей в другие группы. И уж ни при каких обстоятельствах не добавляйте их в привилегированные группы, такие как wheel! Каждый пользователь должен иметь домашний каталог. Для административного пользователя вполне подойдет каталог с именем */nonexistent*. В конце концов, файлы этого пользователя находятся в другом месте. И, наконец, в программе adduser(8) нужно сделать учетную запись неактивной. Хотя выбранный командный интерпретатор предотвратит возможность входа, но дополнительная защита не помешает.

Теперь, после создания административного пользователя и группы можно назначить этого пользователя владельцем требуемых файлов. Каждому файлу поставлены в соответствие пользователь и группа, владеющие им. Увидеть права доступа к файлу и владельца можно с помощью команды `ls -l`. (Если вы не помните, как используются права доступа в UNIX, прочитайте страницы руководства ls(1) и chmod(1).) Многие системные администраторы уделяют самое пристальное внимание правам доступа для владельца файла, чуть меньше – для всех остальных и весьма поверхностное – правам доступа для группы.

```
# ls -l
total 3166
-rw-r----- 1 mwlucas mwlucas 79552 Nov 11 17:58 rndc.key
-rw-rw-r-- 1 mwlucas mwlucas 3131606 Nov 11 17:58 absolutefreebsd.com.db
```

Для этого примера были созданы два файла. Первый файл, *rdnc.key*, доступен для чтения и записи пользователю mwlucas, для чтения – всем, кто входит в состав группы mwlucas, все остальные не имеют никаких прав доступа к этому файлу. Файл *absolutefreebsd.com.db* доступен для чтения и записи пользователю mwlucas и всем, кто входит в состав группы mwlucas, все остальные имеют только право на чтение. Если вы входите в состав группы mwlucas, вы сможете редактировать файл *absolutefreebsd.com.db* и вам для этого совершенно не нужны привилегии пользователя root.

Изменить владельца и группу файла можно с помощью команды chown(1). При этом необходимо знать имя пользователя и группы, которые будут владеть файлом. В данном случае нам требуется, чтобы файлами владели пользователь dns и группа dns.

```
# chown dns:dns rndc.key
# chown dns:dns absolutefreebsd.com.db
# ls -l
total 3166
-rw-r----- 1 dns dns 79552 Nov 11 17:58 rndc.key
-rw-rw-r-- 1 dns dns 3131606 Nov 11 17:58 absolutefreebsd.com.db
```

Теперь этими файлами владеют пользователь dns и группа dns. Любой, кто входит в состав группы dns, сможет редактировать файл *absolutefreebsd.com.db*, не используя пароль root. Наконец, этот файл доступен для чтения серверу имен, работающему с привилегиями пользователя bind. Теперь стоит добавить администраторов DNS в группу dns, в файле */etc/group*, и они тут же смогут приступить к выполнению своих обязанностей.

Администраторы могут думать, что для перезапуска сервера имен им необходимо знать пароль root. Однако это легко решается с помощью rndc(8). Другие задачи могут решаться с помощью заданий в cron или с помощью программы sudo(8).

Интересные группы, создаваемые по умолчанию

В состав FreeBSD входит несколько групп, создаваемых по умолчанию. Большая часть из них используется системой и не требует к себе внимания системного администратора – вам достаточно знать об их существовании. Тем не менее здесь представлены самые полезные, интересные и необычные группы по умолчанию (табл. 7.2). Добавление собственных новых групп упрощает администрирование, однако примите к сведению, что в каждой системе FreeBSD есть группы, приведенные в этом списке.

Таблица 7.2. Интересные группы FreeBSD

Имя группы	Назначение
audit	Группа, обладающая правом доступа к информации audit(8)
authpf	Группа аутентификации фильтра пакетов PF
bin	Группа, владеющая основными двоичными файлами и программами в системе
bind	Группа для встроенного сервера DNS (глава 14)
daemon	Группа, используемая различными системными сервисами, например системой печати
_dhcp	Группа для выполнения операций с клиентами DHCP
dialer	Группа пользователей, имеющих доступ к последовательным портам; удобна для работы с модемами и программой tip(1)
games	Группа для игровых программ и файлов
guest	Группа для гостей системы (практически никогда не используется)
kmem	Группа для программ, обращающихся к памяти ядра, например fstat(1), netstat(1) и т. д.
mail	Группа для системы электронной почты (глава 16)
mailnull	Группа по умолчанию для Sendmail (глава 16)
man	Группа, владеющая страницами руководства
network	Группа, владеющая программами для работы с сетью, такими как ppp(8)
news	Группа для программ, предоставляющих доступ к Usenet (если установлено)
nobody	Основная группа для пользователя nobody, не имеющего никаких привилегий
nogroup	Группа без привилегий
operator	Группа, имеющая доступ к приводам; обычно используется для выполнения резервного копирования
_pflogd	Группа для ведения протокола PF
proxy	Группа для FTP-прокси в фильтре пакетов PF

Таблица 7.2 (продолжение)

Имя группы	Назначение
smmsp	Группа для Sendmail Submission User (глава 16)
sshd	Владелец сервера SSH (глава 15)
staff	Администраторы системы
sys	Системная группа для прочих нужд
tty	Группа для программ, имеющих право вывода на терминалы, таких как wall(1)
uucp	Группа для программ, использующих протокол UNIX-to-UNIX Copy Protocol
wheel	Пользователи, которые имеют право использовать пароль root
www	Группа для программ веб-сервера (но не для веб-файлов)

Настройка безопасности пользователей

Вы можете ограничить активность пользователей с целью предотвращения значительного потребления системных ресурсов, таких как память или процессорное время, каким-то одним пользователем. В наше время это не так важно, потому что сейчас даже небольшие компьютеры оснащаются быстрыми процессорами и большими объемами памяти, но это по-прежнему полезно в системах с десятками и сотнями пользователей. Кроме того, можно ограничить места, откуда пользователи смогут входить в систему.

Ограничение возможности входа

Всякий раз, когда пользователь пытается войти в систему, FreeBSD проверяет содержимое файла */etc/login.access*. Если там есть правила, запрещающие вход в систему того или иного пользователя, то его попытка зарегистрироваться немедленно провалится. По умолчанию этот файл пуст – нет никаких ограничений для пользователей, имеющих имя и пароль.

В файле */etc/login.access* есть три поля, разделенных двоеточиями. Первое поле предоставляет (+) или отнимает (-) право на вход в систему; второе поле – список пользователей или групп; третье – список источников подключений. Можно также использовать выражения `ALL` (все) и `ALL EXCEPT` (все, за исключением), позволяя администратору задавать простые, но выразительные правила. Программа входа в систему проверяет правила, руководствуясь первым совпадением, а не лучшим. Когда программа login(1) находит правило, которому соответствуют и группа, и источник подключения, соединение немедленно разрешается или отклоняется. Значит, порядок правил очень важен. Например, чтобы разрешить регистрацию с системной консоли только членам группы wheel, можно попробовать следующее:

```
+:wheel: console
```

Однако с этим правилом не все в порядке: на самом деле оно не отменяет полномочий пользователей на вход в систему. По умолчанию регистрация разрешена, а все, что делает это правило, – явно предоставляет право регистрироваться пользователям из группы wheel, поэтому данное правило не будет препятствовать регистрации других пользователей. Если я не вхожу в эту группу и попытаюсь зарегистрироваться с системной консоли, это правило не запретит мне доступ.

Можно было бы попробовать создать два правила:

```
+:wheel: console
-:ALL:console
```

Это даст желаемый эффект, но этот вариант можно упростить, если использовать выражение `ALL EXCEPT`.

```
-:ALL EXCEPT wheel: console
```

Это правило отклоняет подключения быстрее. Кроме того, снижается вероятность ошибки администратора. Как правило, при создании списков в *login.access* лучше запрещать регистрации, а не разрешать их. Как только система дойдет до этого правила, она немедленно откажет в регистрации с консоли пользователям, не входящим в группу wheel.

Последнее поле в файле *login.access*, источник подключения, может содержать имена хостов, адреса хостов, номера сетей, имена доменов или специальные значения `LOCAL` и `ALL`.

Имена хостов

Имена хостов опираются на DNS или файл *hosts*. Если есть подозрение, что сервер имен подвергается атакам взломщиков, не стоит использовать доступ по имени соединяющегося хоста; злоумышленники могут назначить имени хоста любой IP-адрес и обмануть вашу систему при подключении. Тем не менее можно предпринять следующее:

```
-:ALL EXCEPT wheel:fileserver.mycompany.com
```

Пользователи из группы wheel могут входить в систему с fileserver, а все остальные – нет.

Адреса хостов и сети

Адреса хостов похожи на имена хостов, но они невосприимчивы к манипуляциям с DNS.

```
-:ALL EXCEPT wheel:192.168.1.5
```

Номер сети – это усеченный IP-адрес:

```
-:ALL EXCEPT wheel:192.168.1.
```

Такое правило позволит каждому члену группы wheel регистрироваться с машины, IP-адрес которой начинается с 192.168.1, а также запретит доступ всем остальным.

LOCAL

Самое сложное местоположение – `LOCAL`. Ему соответствует любое имя хоста без точки (обычно это хосты в локальном домене). Например, *www.absolutefreebsd.com* предполагает, что любой хост в домене *absolutefreebsd.com* соответствует `LOCAL`. Такая функциональность реализована на основе метода обратного преобразования адресов в DNS (глава 14), который более уязвим для мистификаций. Мой ноутбук может заявить, что его имя – *humvee.blackhelicopters.org*, но его IP-адрес соответствует записи обратного DNS, утверждающей, что он находится где-то в сети моего поставщика услуг Интернета. Машина в домене *absolutefreebsd.com* подумает, что моему ноутбуку присвоено имя хоста, принадлежащее другому домену, а значит, он не является локальным. Таким образом, я не могу применять `LOCAL` в качестве метода проверки.

Аналогичным образом, любой, кто владеет блоком IP-адресов, сможет дать им любые имена в записях обратного DNS. Поэтому ограничение `LOCAL` не особенно полезно в реальных условиях.

ALL и ALL EXCEPT

Выражение `ALL` соответствует всем хостам, а `ALL EXCEPT` – всем, за исключением указанных далее. На мой взгляд – это наиболее полезные выражения при построении правил, ограничивающих источники подключения. Например, если необходимо обеспечить доступ к системе только с двух рабочих станций, можно записать такое правило:

```
-:ALL EXCEPT wheel:ALL EXCEPT 192.168.89.128 192.168.170.44
```

Собрать все вместе

Назначение этих правил заключается в обеспечении политики входа в систему, которая соответствует вашей политике безопасности. Если вы оказываете услуги общего характера, но удаленный доступ к системе разрешается только для системных администраторов, однострочный *login.access* позволит предотвратить попытки входа посторонних пользователей:

```
-:ALL EXCEPT wheel:ALL
```

Это замечательно, если с подобными ограничениями можно жить и работать. Однако мне случалось работать с несколькими поставщиками услуг Интернета, которые использовали FreeBSD. Обычным клиентам не разрешалось регистрироваться на серверах, если в их учетных записях не был указан командный интерпретатор. Системные администраторы, а также администраторы DNS и веб-сервера (члены групп dns и webmasters) могли выполнять вход с удаленных рабочих мест. Но с консоли могли выполнять вход только системные администраторы.

```
-:ALL EXCEPT wheel:console
-:ALL EXCEPT wheel dns webmasters:ALL
```

При наличии этой записи пользователи не смогут войти в систему, если их не добавить в группу, которой разрешен доступ.

Ограничение на использование ресурсов системы

Существует возможность организовать более точное управление с помощью классов доступа. Описания классов доступа находятся в файле */etc/login.conf* и определяют, какие данные и какие ресурсы могут предоставляться пользователям. Каждый пользователь закрепляется за определенным классом, и у каждого класса есть свои ограничения на доступ к системным ресурсам. Изменения ограничений для класса затрагивают всех пользователей, которые к нему принадлежат. Класс пользователя задается при создании его учетной записи. Изменить класс можно с помощью команды chpass(1).

Определения классов

По умолчанию *login.conf* начинается с класса default, который применяется для учетных записей, не относящихся к другим классам. По сути, этот класс предоставляет пользователям неограниченный доступ к системным ресурсам и больше подходит для приложений серверов с ограниченным числом пользователей. Если это соответствует вашим потребностям, не трогайте этот файл вообще.

Каждое определение класса состоит из последовательности операций присваивания значений переменным, которые определяют среду окружения пользователя, порядок учета и ограничения на системные ресурсы. Каждая запись в определении класса начинается и заканчивается двоеточием. Символ обратного слэша – это символ продолжения, он означает, что определение класса продолжается на следующей строке. За счет этого файл можно представить в удобочитаемом формате. Вот как может начинаться описание класса:

```
❶default:\
❷:passwd_format=❸md5:\
:copyright=/etc/COPYRIGHT:\
:welcome=/etc/motd:\
...
```

Этот класс называется `default` ❶. Я показал три из нескольких десятков переменных в этом классе. Например, переменной `passwd_format` ❷ присвоено значение `md5` ❸. Такие операции присваивания значений переменным и само имя класса составляют описание класса, и вы можете влиять на работу пользователя в системе, причисляя его к тому или иному классу.

Некоторые переменные *login.conf* не имеют значений; они изменяют свойства учетной записи своим наличием. Например, воздействие переменной `requirehome` определяется упоминанием ее в классе. Если эта переменная присутствует в определении класса, то для входа в систему пользователь обязан будет иметь домашний каталог.

```
:requirehome:\
```

После редактирования *login.conf* необходимо обновить базу данных login, чтобы изменения вступили в силу:

```
# cap_mkdb /etc/login.conf
```

Эта команда перестроит файл базы данных */etc/login.conf.db*, который используется для ускорения поиска, как описанный ранее файл */etc/spwd.db*. По умолчанию файл */etc/login.conf* включает в себя несколько примеров классов пользователей. Чтобы понять, какие ограничения можно накладывать на пользователей в различных ситуациях, справляйтесь с этими примерами. В следующем разделе рассказывается о том, что можно устанавливать в классе доступа. Полный перечень доступных параметров в вашей версии FreeBSD можно найти на странице руководства login.conf(5).

Ограничения на ресурсы

Ограничения на ресурсы позволяют управлять объемом системных ресурсов, которые одновременно может задействовать один пользователь. Если к машине подключаются несколько сотен пользователей, а один из них решает скомпилировать OpenOffice.org, то этот пользователь может потреблять намного больше памяти и времени процессора, чем ему положено по справедливости. Если ограничить ресурсы, которые один пользователь может монополизировать одновременно, то система будет более отзывчивой к остальным пользователям.

В табл. 7.3 описаны переменные файла *login.conf*, отвечающие за ограничения ресурсов.

Таблица 7.3. Переменные login.conf для ограничения ресурсов

Переменная	Описание
cputime	Максимальное время процессора, которое может использовать любой процесс
filesize	Максимальный размер одного файла
datasize	Максимальный объем памяти, который может потреблять один процесс для хранения данных
stacksize	Максимальный объем стека, доступный одному процессу
coredumpsize	Максимальный размер дампа памяти
memoryuse	Максимальный объем памяти, который процесс может заблокировать
maxproc	Максимальное количество процессов, которые могут быть одновременно запущены одним пользователем
openfiles	Максимальное количество открытых файлов на один процесс
sbsize	Максимальный размер буфера сокета, который может задействовать приложение пользователя

Обратите внимание: зачастую ограничения на ресурсы привязаны к процессам по отдельности. Если каждый процесс получает 20 Мбайт памяти, а каждый пользователь может запускать 40 процессов, то тем самым вы разрешаете каждому пользователю получить 800 Мбайт памяти. Возможно, в вашей системе очень много памяти, но действительно ли так много?

Текущие и максимальные ограничения на ресурсы

Помимо ограничений, перечисленных выше, имеется возможность задать текущие (current) и максимальные ограничения на ресурсы. *Текущие* ограничения обычно носят рекомендательный характер, и пользователь может менять их по своему желанию. Это удобно в окружении, основанном на совместной работе, когда пользователи легко разделяют ресурсы, но вам необходимо предупреждать пользователей, превышающих стандартные ограничения. *Максимальные* ограничения абсолютны, и пользователь не может их раздвинуть.

Если не указывать тип ограничения, текущее или максимальное, система будет воспринимать его как максимальное.

Для задания текущего ограничения добавьте `-cur` к имени переменной. Для установления жесткого лимита добавьте `-max`. Например, для установления текущего ограничения на количество процессов, которые могут быть у одного пользователя, выполните следующее:

```
...
:maxproc-cur: 30:\
:maxproc-max: 60:\
...
```

Помимо наложения ограничений можно использовать функции учета ресурсов. Учет ресурсов не так уж важен сегодня по сравнению с временами, когда недорогой компьютер стоил десятки тысяч долларов, поэтому учет ресурсов в этой книге не рассматривается. Гораздо важнее иметь возможность накладывать ограничения на отдельного пользователя, чтобы не позволить ему потреблять больше процессорного времени, чем положено. Тем не менее полезно знать, что такая возможность существует.

Задание параметров среды по умолчанию

Существует возможность определять параметры среды в файле */etc/login.conf*. Это лучше, чем устанавливать их в пользовательских файлах *.cshrc* или *.profile*, поскольку настройки в *login.conf* окажут влияние на все учетные записи пользователей, подключающихся после задания этих параметров. Некоторые командные интерпретаторы, такие как zsh(1), не просматривают упомянутые конфигурационные файлы, поэтому описание среды в классе позволит правильно установить значения переменных окружения для пользователей, работающих с такими интерпретаторами.

Каждое поле окружения распознает два специальных символа. Тильда (~) замещается домашним каталогом пользователя, а знак доллара ($) – именем пользователя. Следующие несколько примеров из класса default иллюстрируют их использование:

```
:setenv=MAIL=❶/var/mail/$,BLOCKSIZE=K,FTP_PASSIVE_MODE=YES:\
:path=/sbin /bin /usr/sbin /usr/bin /usr/games /usr/local/sbin /usr/local/
bin /usr/X11R6/bin ❷~/bin:\
```

Например, переменной окружения MAIL присваивается значение */var/mail/<username>* ❶. Так же последний каталог в определении переменной PATH – это каталог *bin* в домашнем каталоге пользователя.

В табл. 7.4 перечислены типичные параметры настройки среды, которые используются в файле *login.conf*.

Таблица 7.4. Типичные параметры настройки среды в login.conf

Переменная	Описание
hushlogin	Если присутствует в определении класса, системная информация не выдается при входе в систему.
ignorenologin	Если присутствует в определении класса, пользователь может войти в систему, даже когда файл */var/run/nologin* существует.
ftp-chroot	Если присутствует в определении класса, пользователи при работе с FTP помещаются chroot-окружение (глава 15).
manpath	Список каталогов для переменной окружения $MANPATH.
nologin	Если присутствует, пользователь не может войти в систему.
path	Список каталогов для переменной окружения $PATH.
priority	Приоритет (nice) пользовательских процессов по умолчанию (глава 19).
setenv	Список переменных окружения, разделенных запятыми, с их значениями.
umask	Значение umask по умолчанию (см. builtin(1)). Это значение всегда должно начинаться с 0.
welcome	Полный путь к файлу, содержащему приветственное сообщение.
shell	Полный путь к командному процессору, который будет запущен после входа в систему. Эта запись подменяет командный процессор, указанный в */etc/master.passwd*. Однако переменная окружения $SHELL будет указывать на командный процессор, заданный в файле паролей, поэтому окружение будет противоречивым. Изменение значения этой переменной – замечательный способ раздосадовать пользователей.
term	Тип терминала по умолчанию. Чуть ли не каждая программа, пытающаяся установить тип терминала, подменяет эту запись.
timezone	Значение по умолчанию переменной окружения $TZ.

Управление свойствами пароля и входа в систему

В отличие от параметров среды окружения, которые можно настроить не только с помощью классов доступа, большая часть параметров настройки входа в систему и аутентификации могут выполняться только в определении класса доступа. Ниже приводятся некоторые параметры настройки аутентификации:

`minpasswordlength`

Задает минимальную длину пароля. Это значение возымеет эффект лишь при следующем изменении пароля пользователем; никакой проверки на соответствие уже установленных паролей этой длине не проводится. В следующем примере минимальная длина пароля устанавливается равной 28 символам (это лучший способ побудить пользователей использовать ключи SSH).

```
\:minpasswordlen=28:\
```

`passwd_format`

Задает алгоритм, применяемый для шифрования паролей в */etc/master.passwd*. Значение по умолчанию – `md5`. Другие допустимые значения – `des` (DES), `blf` (Blowfish) и `nthash` (Windows NT). DES наиболее полезен, когда необходимо иметь одинаковые пароли на компьютерах с различными операционными системами. Blowfish – очень ресурсоемкий алгоритм, но в современных системах процессорное время стоит дешевле грязи, поэтому его тоже вполне можно использовать. Алгоритм `nthash` очень удобен, когда *возникает потребность*, чтобы кто-то взломал вашу систему, иного предназначения этого алгоритма я не вижу.

`mixpasswordcase`

Если это свойство задано, то при следующем изменении паролей пользователи не смогут задавать в них только строчные буквы.

`host.allow`

Пользователи в классе с этим значением могут применять rlogin и rsh. Такая установка настоятельно не рекомендуется.

`host.deny`

Это значение используется при работе с rlogin и rsh. Избегайте их как несвежего мяса.

`times.allow`

Определяет промежуток времени, когда пользователь может входить в систему. Для задания времени необходимо в полях, разделенных двоеточиями, указать дни и периоды времени. Дни обозначаются двумя первыми буквами названия дня недели (Su, Mo, Tu, We, Th, Fr и Sa). Время указывается в стандартном 24-часовом формате. Например, если пользователю разрешено входить в систему только по средам между 8.00 и 17.00, подойдет такая запись:

```
        :times.allow=We8-17:\
```

`times.deny`

Определяет промежуток времени, когда пользователю нельзя входить в систему. Отметим, что эта запись не «выкинет» пользователя, если он уже вошел в систему. Формат записи такой же, как и в `times.allow`. Если `times.allow` и `times.deny` перекрываются, то `times.deny` имеет преимущество.

Флаги файлов

Все UNIX-подобные операционные системы обладают одними и теми же наборами прав доступа к файлам в файловой системе, такими как права на запись, на чтение и на исполнение файла для владельца файла, для группы и для всех остальных. Система FreeBSD расширяет схему прав, вводя *флаги файлов*. Вместе с правами доступа эти флаги усиливают защиту системы. Некоторые флаги используются для функций, которые не связаны с безопасностью, однако здесь будут рассматриваться флаги, имеющие к ней отношение. Полный список флагов файлов вы найдете в странице руководства chflags(1).

Многие флаги оказывают различное влияние в зависимости от уровня безопасности системы, кратко рассмотренного в следующем разделе. Для понимания уровней безопасности необходимо понимать флаги файлов, и при этом флаги файлов опираются на уровни безопасности. Пока кивайте головой и улыбайтесь при упоминании об этих уровнях; все станет ясно через несколько страниц.

Ниже представлены флаги, относящиеся к безопасности:

`sappnd`

Системный флаг «только добавление», который может устанавливать только root. В файл с этим флагом можно добавлять записи, однако удалять или редактировать этот файл нельзя (это особенно удобно для файлов протоколов). Если учетная запись каким-то образом «скомпрометирована», то установка флага `sappnd` файла *.history* может принести пользу. Тактика типичного злоумышленника состоит в том, чтобы удалить *.history* или сделать из него символическую ссылку на */dev/null*, чтобы администратор не узнал о предпринятых действиях. Благодаря установке флага `sappnd` малыши со скриптами не смогут замести следы таким образом. Забавно наблюдать за кем-либо, кто пытается удалить файл с флагом `sappnd`. Можно увидеть, как растет разочарование взломщика с каждой новой попыткой.[1] Этот флаг нельзя изменить, когда система работает на уровне безопасности 1 и выше.

[1] На самом деле это не так забавно, потому что злоумышленник все-таки проник в систему, но это хотя бы доставит краткий миг радости в тяжелый день.

`schg`

Системный флаг «неизменяемости», который может устанавливать только root. Файлы с установленным флагом `schg` нельзя изменять вообще: редактировать, перемещать, заменять. Сама файловая система пресекает все попытки коснуться этого файла каким-либо образом. Этот флаг нельзя изменить, когда система работает на уровне безопасности 1 и выше.

`sunlnk`

Системный флаг «запрет на удаление», который может устанавливать только root. Файл можно редактировать или изменять, но нельзя удалить. Этот флаг не настолько безопасен, как предыдущие два: если файл можно редактировать, то его можно очистить. Впрочем, это удобно при определенных обстоятельствах. Я использовал эту возможность, когда программа настаивала на удалении собственного файла протокола в случае аварии. Обычно установка любых стандартных флагов не приносит пользы. Этот флаг нельзя изменить, когда система работает на уровне безопасности 1 и выше.

`uappnd`

Пользовательский флаг «только добавление», который может устанавливать только владелец файла или root. Как в случае с установленным системным флагом `sappnd`, в файл с флагом `uappnd` можно добавлять записи, однако этот файл нельзя удалять или редактировать. Такая возможность наиболее полезна для протоколов, которые ведут личные программы пользователя. Установка такого файла предохраняет пользователей от их собственных ошибок. Владелец файла и root могут удалить этот флаг в любое время.

`uchg`

Пользовательский флаг «неизменяемости», который может устанавливать только владелец файла или root. Как в случае с флагом `schg`, описанным ранее, флаг «неизменяемости» защищает пользователя от изменения файла. Как и в предыдущих случаях, root может преодолеть действие этого флага. Кроме того, сам пользователь может убрать его, на каком бы уровне безопасности ни работала система. Этот флаг помогает предотвратить ошибки, но не обеспечивает безопасность системы.

`uunlnk`

Пользовательский флаг «запрет на удаление», который может установить только владелец файла или root. Файл с установленным флагом `uunlnk` владелец удалить не может, а root может. Пользователь может снять этот флаг в любой нужный момент.

Просмотр и установка флагов файлов

Флаги можно устанавливать командой chflags(1). Например, чтобы защитить ядро от подмены, можно сделать следующее:

```
# chflags schg /boot/kernel/kernel
```

Теперь никто не заменит ядро: ни злоумышленник, ни вы.

С помощью ключа `-R` можно рекурсивно изменять флаги, продвигаясь по дереву каталогов. Например, чтобы сделать каталог */bin* неизменяемым, воспользуйтесь такой командой:

```
# chflags -R schg /bin
```

Ура! Базовые исполняемые файлы теперь нельзя изменить.

Флаги файла можно увидеть с помощью команды `ls -lo`:

```
# ls -lo log
-rw-r--r--   1 mwlucas   mwlucas   sappnd 0 Nov 12 12:37 log
```

Символы `sappnd` здесь говорят о том, что флаг «только добавление» установлен на этот файл протокола. Для сравнения приведем строку в случае, когда у файла не установлены флаги:

```
# ls -lo log
-rw-r--r--   1 mwlucas   mwlucas   - 0 Nov 12 12:37 log
```

Дефис вместо имени флага говорит о том, что флаги файловой системы не были установлены.

В свежеустановленных системах FreeBSD лишь немногие файлы отмечены таким образом. Впрочем, флаги можно установить как угодно. На одной системе, считавшейся взломанной, я неистово запускал `chflags -R schg` в различных системных каталогах, чтобы не позволить никому заменить системные исполняемые файлы на троянские версии. Это не остановило бы взломщика, но я почувствовал себя лучше, представив, как бы он был расстроен, если бы добрался до командной строки.

Для снятия флага надо в команде `chflags` предварить имя флага словом `no`. Например, чтобы снять флаг `schg`, установленный на ядро, введите следующую команду:

```
# chflags noschg /boot/kernel/kernel
```

Надо сказать, что снятие этого флага подразумевает работу системы на уровне безопасности -1. Поэтому, не откладывая в долгий ящик, обсудим уровни безопасности.

Уровни безопасности

Уровни безопасности – это параметры настройки ядра, которые изменяют поведение базовой системы, запрещая те или иные действия. С повышением уровня безопасности ядро будет вести себя немного иначе. Например, на низких уровнях безопасности флаги файлов, рассмотренные выше, могут быть сняты. Например, флаг «неизменяемости» можно снять, отредактировать файл, а затем снова установить

флаг. На более высоких уровнях безопасности флаг файла не может быть снят. Подобные изменения имеют место в других частях системы. В целом, изменение поведения системы, вызванное повышением уровня безопасности, либо сорвет планы злоумышленника, либо остановит его. Уровень безопасности системы, вступающий в силу при начальной загрузке, можно задать с помощью параметра `kern_securelevel_enable="YES"` в файле *rc.conf*.

Уровни безопасности усложняют эксплуатацию системы, поскольку накладывают определенные ограничения на ее поведение. В конце концов, многие операции, выполняемые при обычном администрировании, может применить и злоумышленник, заметающий следы. Например, на некоторых уровнях безопасности невозможно отформатировать или смонтировать новый жесткий диск. С другой стороны, уровни безопасности препятствуют злоумышленнику больше, чем вам.

Установка уровней безопасности

Существует пять уровней безопасности: -1, 0, 1, 2 и 3, где -1 является низшим уровнем, а 3 – наивысшим. Как уже говорилось, разрешить использование уровней безопасности можно с помощью параметра `kern_securelevel_enable` в файле *rc.conf*. После этого уровень безопасности можно автоматически задавать при загрузке, указав его в переменной `kern_securelevel` файла *rc.conf*. Когда бы вы ни увеличили уровень безопасности системы, его нельзя снизить без перезагрузки в однопользовательском режиме. Если бы снизить уровень можно было без перезагрузки, этим воспользовался бы злоумышленник!

Уровень безопасности -1

Уровень безопасности -1, устанавливаемый по умолчанию, не подразумевает применения дополнительных средств защиты, предоставляемых ядром. Если вы изучаете FreeBSD и часто изменяете конфигурацию, оставьте уровень безопасности -1 и применяйте права доступа к файлам, встроенные в систему FreeBSD, а также другие меры безопасности UNIX, достаточные в большинстве ситуаций.

Уровень безопасности 0

Уровень безопасности 0 используется только в начале загрузки системы. Он не предлагает никаких специальных функций. Когда система переходит в многопользовательский режим, уровень безопасности автоматически увеличивается до 1. Установка `kern_securelevel=0` в */etc/rc.conf* эквивалентна установке `kern_securelevel=1`. Однако это может быть полезно, если во время загрузки системы запускаются сценарии, которые не могут выполнять необходимые действия на более высоких уровнях безопасности.

Уровень безопасности 1

На уровне безопасности 1 ситуация интереснее:

- Системные флаги файлов не могут быть сняты.
- Нельзя загружать и выгружать модули ядра (см. главу 5).
- Программы не могут записывать данные напрямую в системную память через устройства */dev/mem* или */dev/kmem*.
- Закрыт доступ к */dev/io*.
- На монтированные диски нельзя записывать данные напрямую, а значит, нельзя форматировать разделы. (Файлы можно записывать на диск через стандартный интерфейс ядра, нельзя лишь обращаться к диску, как к физическому устройству.)

Самое наглядное следствие уровня безопасности 1 в том, что нельзя изменять флаги файловой системы, специфичные для BSD. Если файл помечен системным флагом «неизменяемый», то заменить его не удастся.

Уровень безопасности 2

Уровень безопасности 2 предоставляет все преимущества уровня безопасности 1 с двумя добавлениями:

- Нельзя записывать данные напрямую в монтированные и немонтированные файловые системы.
- Системное время за раз можно изменить не более чем на 1 секунду.

Начинающий системный администратор может посчитать эти особенности несущественными, однако это важные приемы обеспечения безопасности. Хотя UNIX предоставляет удобные инструменты, например текстовые редакторы для записи данных в файл, без них можно обойтись. Точно так же можно обойтись и без файловой системы и получить прямой доступ к нулям и единицам, которые записаны на диске. В таком случае можно изменить любой файл вне зависимости от прав доступа. На практике такая возможность применяется лишь при установке нового жесткого диска. Обычно напрямую записывать данные на диск может только пользователь root. При уровне безопасности 2 это не позволяется даже ему.

Другой хакерский прием состоит в том, чтобы изменить системное время, отредактировать файл и установить прежнее время. Когда администратор ищет файлы, которые могли вызвать неполадки, искаженный файл выглядит так, словно к нему годами не притрагивались, а значит, не привлечет к себе пристального внимания.

Уровень безопасности 3

Уровень безопасности 3 называют *сетевым режимом безопасности*. Он проявляет себя точно так же, как уровень безопасности 2, но не допускает изменений правил пакетного фильтра. Правила брандмауэра невозможно изменить на этом уровне безопасности. Если в системе

включена фильтрация пакетов или управление полосой пропускания, а их правила хорошо настроены и вряд ли будут изменяться, то можно установить уровень безопасности 3.

Какой уровень безопасности выбрать?

Уровень безопасности, необходимый для той или иной операционной среды, зависит от ситуации. Например, если машина FreeBSD только что введена в эксплуатацию и ее требуется точно настроить, следует оставить уровень безопасности -1. По окончании настройки уровень безопасности можно увеличить. Для большинства систем вполне подходит уровень безопасности 2.

Если применяется одна из программ FreeBSD, предназначенных для фильтрации пакетов и организации брандмауэра, то стоит задуматься об уровне безопасности 3. Однако убедитесь в надежности правил брандмауэра, прежде чем выбрать уровень 3! Уровень безопасности 3 не позволит изменять конфигурацию брандмауэра без разрыва соединения с Интернетом. Вы должны быть уверены на все 100%, что ни один из ваших клиентов не скажет: «Я доплатил, удвойте мою полосу пропускания».

Когда уровни безопасности и флаги файлов не помогают?

Рассмотрим случай, когда взломщик находит «дыру» в сценарии CGI на веб-сервере Apache, получает через него доступ к командному интерпретатору (shell), а затем с его помощью получает права root.

Если уровень безопасности выбран правильно, планы взломщика, вероятно, сорвутся, так как он не сможет заменить ядро системы на свое, специально скомпилированное ядро. Тем не менее он сможет заменить множество системных исполняемых файлов на «троянских коней» с тем, чтобы при следующем входе в систему ваш пароль был отправлен на анонимный почтовый ящик с веб-интерфейсом или в сетевую телеконференцию.

Итак, для защиты своих ключевых файлов вы запускаете `chflags schg -R /bin/*, chflags schg -R /usr/lib` и т. д. Отлично. Если вы забыли про один файл, скажем, какой-нибудь непонятный */etc/rc.bsdextended*, то хакер сможет отредактировать его, добавив `chflags -R noschg /`. Позднее он может перезагрузить систему – ночью, когда вы этого не заметите. Как часто вы садитесь и основательно проверяете файлы */etc/rc*?

Вы думаете, что все файлы полностью защищены и система в безопасности. А что у вас в каталоге */usr/local/etc/rc.d*, где находятся локальные программы, запускаемые при начальной загрузке? В процессе загрузки система будет выполнять все файлы с расширением *.sh*, найденные в этом каталоге. Значит, гипотетический хакер может причинить немало вреда, разместив здесь простенький сценарий командного интерпретатора. В конце концов, */etc/rc* увеличивает уровень безопасно-

сти в последнюю очередь, когда все программы уже запущены. А что если хакер создаст сценарий, который убьет запущенный */etc/rc* перед тем, как он увеличит уровень безопасности, а затем запустит свой собственный */var/.hidden/rc.rootkit*, чтобы завершить подключение к сети?

Конечно, это только один путь, есть и другие, количество которых ограничивается только способностями взломщика. Важно помнить, что организация защиты системы – это нелегкая задача, для которой не существует простого решения. Как только взломщик получил доступ к командной строке, начинается схватка. И, если он имеет достаточно хорошую подготовку, о его появлении вы узнаете слишком поздно. Следуйте надежным рекомендациям по обеспечению безопасности и постоянно обновляйте систему, это самое первое средство предотвратить вторжения. Не позволяйте себе обленившись, положившись на уровни безопасности!

Жизнь с уровнями безопасности

Если вы использовали флаг `schg` направо и налево, вы скоро обнаружите, что установка обновлений или «заплат» стала неудобной. На самом деле, те условия, которые усложняют жизнь хакерам, могут сделать и вашу жизнь невыносимой, если не знать обходных путей. Итак, каковы же обходные пути?

Если файл */etc/rc.conf* защищен флагом `schg`, то для редактирования системных файлов сначала надо понизить уровень безопасности. Конечно, параметр, задающий уровень безопасности, находится в */etc/rc.conf*, поэтому для редактирования этого файла необходимо получить контроль над системой до запуска */etc/rc*. Для этого загрузите систему в однопользовательском режиме (как рассказывалось в главе 3), смонтируйте требуемые файловые системы, запустите `chflags noschg` на задействованных файлах и продолжите загрузку. Можно даже полностью отключить уровни безопасности в */etc/rc.conf* и потом спокойно работать. В этом случае процесс обновлений пойдет быстрее, но защита с помощью флагов файлов будет потеряна.

После внесения необходимых изменений уровень безопасности можно увеличить (но не уменьшить), не прибегая к перезагрузке. Для этого достаточно изменить параметр sysctl `kern.securelevel`, установив желаемый уровень безопасности.

```
# sysctl kern.securelevel=3
```

Теперь, когда вы можете управлять изменениями файлов, рассмотрим управление доступом к системе из сети.

Цели нападения из сети

Как правило, взламывается не операционная система, а программы, которые предоставляют возможность подключения из сети. Операци-

онная система может способствовать или не способствовать защите программ от сетевых атак, но вторжение всегда начинается с приложений. Один из способов сократить количество нападений на сервер состоит в том, чтобы составить полный перечень работающих программ, которые ожидают подключения из сети, и запретить запуск тех из них, которые не являются необходимыми. Операционная система FreeBSD содержит программу sockstat(1), которая предоставляет наиболее простой способ узнать, какие программы «прослушивают» сеть.

Программу sockstat мы уже рассматривали достаточно подробно в главе 6 – команда `sockstat -4` продемонстрирует все открытые порты TCP/IP IPv4. Каждый открытый сетевой порт можно рассматривать как потенциальную брешь и цель для нападения. Закройте все сетевые порты, ненужные для работы сервера, и обеспечьте защиту для тех, которые необходимы.

Список открытых портов следует просматривать постоянно, это поможет вам узнать кое-что, что может удивить вас. Например, вы можете с удивлением обнаружить, что некоторое программное обеспечение, установленное вами, имеет сетевой компонент, о существовании которого вы даже не подозревали, который преспокойно «прослушивал» сеть.

Но как отключить ненужные службы после того, как вы узнаете, что таковые действительно имеются? Лучший способ закрыть ненужные порты заключается в том, чтобы не запускать программы, которые их открывают. Обычно сетевые демоны запускаются из двух мест: либо из */etc/rc.conf*, либо из сценария запуска в */usr/local/etc/rc.d*. Программы, интегрированные в основную систему FreeBSD, такие как sendmail(8), sshd(8) и rpcbind(8), имеют флаги в */etc/rc.conf*, с помощью которых можно разрешать или запрещать запуск этих программ, а также многих других дополнительных приложений. Некоторые дополнительные программы, такие как веб-серверы, запускаются из сценариев

Безопасность серверов и рабочих станций

Во многих компаниях, которые мне приходилось видеть, уделяют очень много внимания безопасности серверов и практически не уделяют внимания безопасности рабочих станций. Однако для злоумышленника совершенно неважно, что взламывать, – сервер или рабочую станцию. На многих серверах и в системах сетевой защиты имеются специальные правила для рабочих станций системных администраторов. Злоумышленник с радостью взломает рабочую станцию и воспользуется ею для проникновения на сервер. Безусловно, безопасность сервера имеет очень большое значение, но не пренебрегайте защитой рабочих станций, особенно если это ваша рабочая станция!

в */usr/local/etc/rc.d*. Более подробно вопрос разрешения или запрета запуска программ во время загрузки рассматривается в главе 3.

Собрать все вместе

Когда в системе открыты только необходимые порты и ясно, какие программы их используют, то о защите этих программ и следует позаботиться. Если разработчики, обеспечивающие безопасность FreeBSD, отправляют извещение о неполадках сервиса, который вы не применяете, это сообщение можно спокойно проигнорировать. Если же речь идет о «дыре» в используемой программе, на это необходимо обратить внимание и как можно скорее внести необходимые исправления. Способность быстро реагировать на реальные опасности поможет вам защититься от большинства нападений. Такие инструменты, как флаги файлов или уровни безопасности, позволят минимизировать ущерб, который могут причинить взломщики. Наконец, использование групп для ограничения действий самих системных администраторов может защитить компьютеры от случайного или преднамеренного причинения вреда.

8

Диски и файловые системы

Важность управления файловыми системами и дисками трудно переоценить. (Попробуйте переоценить их важность, а я подожду.) Для операционной системы надежность и гибкость работы с дисками должны быть первостепенно важны, потому что на дисках хранятся ваши данные. Операционная система FreeBSD предоставляет выбор файловых систем и различных вариантов их обслуживания. В этой главе мы рассмотрим типичные компоненты, с которыми имеет дело системный администратор.

Жесткие диски

Большинство воспринимают жесткий диск как «черный ящик». При неаккуратном обращении можно добиться того, что жесткий диск будет визжать и греметь, а при очень плохом обращении из него может пойти дымок и он просто выйдет из строя. Чтобы по-настоящему понимать, что такое файловые системы, вам необходимо немножко понимать, что происходит внутри привода. Если у вас есть старый жесткий диск, который вам больше не нужен, вскройте его корпус и читайте дальше.

Внутри корпуса накопителя на жестких магнитных дисках находится стопка круглых алюминиевых или пластиковых дисков, которые обычно называют *пластинами*. Во время работы привода эти пластины вращаются со скоростью несколько тысяч оборотов в минуту. Число RPM (revolutions per minute), которое приводится на корпусе жесткого диска, – это и есть скорость вращения пластин. Пластины покрыты тонким слоем магнитного материала. Этот магнитный материал распределен по тысячам концентрических колец, которые называются *дорожками* (*tracks*). Они располагаются как годичные кольца в стволе дерева, от сердцевины к внешнему краю. На этих дорожках и хранятся данные в виде последовательностей нулей и единиц. Каждая дорожка

делится на *секторы*. Сектора на внешних дорожках вмещают больше данных, чем на внутренних, и скорость чтения данных, записанных не внешних дорожках, также выше, потому что каждая точка на внешней дорожке имеет более высокую линейную скорость.

Головки, смонтированные над каждой из пластин, записывают и считывают данные с вращающихся пластин точно так же, как игла патефона. Головки могут выполнять чтение и запись очень быстро, но им приходится ждать, пока под ними не окажется нужная позиция на диске. Производительность накопителей на жестких магнитных дисках в основном зависит от скорости вращения пластин, вот почему такой показатель, как скорость вращения, имеет важное значение.

ATA, SATA, SCSI и SAS

Я полагаю, вы уже знакомы с основными понятиями технологии хранения данных на жестких дисках. Если еще нет, тогда я рекомендую провести некоторое время в Интернете и прочитать что-нибудь по этой теме, например статьи в Wikipedia. Я иногда буду ссылаться на эти понятия, но рассказ о том, что такое LUN или SCSI ID, – это тема отдельной книги.

Файлы устройств

В главе 3 мы коротко коснулись файлов устройств, а теперь рассмотрим их более подробно. Файлы устройств – это специальные файлы, представляющие аппаратные средства системы. Они служат логическим интерфейсом между пользовательскими программами с одной стороны и драйверами устройств или физическими устройствами – с другой. Выполняя команду с файлом устройств, посылая информацию файлу устройств или читая данные из него, ядро выполняет те или иные операции с физическим устройством. Для разных устройств эти операции могут существенно различаться, – в конце концов, запись данных на диск дает совсем другие результаты, нежели передача данных звуковой карте. Файлы устройств можно найти в каталоге */dev*.

Чтобы приступить к работе с диском или дисковыми разделами, необходимо знать имя устройства или имя файла, который представляет это устройство. Как правило, они основаны на имени драйвера соответствующего аппаратного средства, а имена драйверов устройств, в свою очередь, основаны на названии микросхемы, применяемой в устройстве, а не на его назначении.

В табл. 8.1 приводятся наиболее типичные файлы устройств дисковых накопителей. Более подробное их описание вы найдете в соответствующих страницах руководства.

Таблица 8.1. Дисковые накопители и типы

Файл устройства	Страница руководства	Описание
*/dev/fd**	fdc(4)	Накопители на гибких магнитных дисках
*/dev/acd**	acd(4)	CD-ROM с интерфейсом IDE
*/dev/ad**	ad(4)	Жесткие диски ATA и SATA и разделы
*/dev/cd**	cd(4)	CD-ROM с интерфейсом SCSI
*/dev/da**	da(4)	Жесткие диски с интерфейсом SCSI или SAS, устройства хранения данных с интерфейсом USB и flash-накопители

Для отдельных дисков, подключенных к аппаратным контроллерам RAID, файлы устройств не используются. Вместо этого контроллеры RAID представляют по виртуальному диску для каждого контейнера RAID, для чего к имени драйвера RAID добавляется имя устройства. Например, драйвер amr(4) представляет такие виртуальные диски, как */dev/amrd**. Для работы с некоторыми контроллерами RAID используется промежуточный уровень абстракции cam(4), и в этом случае диски отображаются как устройства */dev/da**.

Жесткие диски и разделы

В главе 2 мы уже рассматривали понятие раздела, теперь мы посмотрим на разделы с точки зрения драйвера устройства. Первый диск ATA, подключенный к первому контроллеру ATA, называется */dev/ad0*. За ним следуют диски */dev/ad1*, */dev/ad2* и т. д. Подразделы на каждом диске начинаются с этого имени. К данному названию добавляются дополнительные символы, например */dev/ad0s1b*. Диск можно считать одним целым, но в каталоге */dev*, как правило, есть масса файлов подразделов, начинающихся с */dev/ad0*:

```
# ls /dev/ad*
/dev/ad0      /dev/ad0s1a    /dev/ad0s1c    /dev/ad0s1e
/dev/ad0s1    /dev/ad0s1b    /dev/ad0s1d    /dev/ad0s1f
```

Итак, что означают все эти подразделы? Вспомните, как вы распределяли дисковое пространство для FreeBSD. Если вы следовали рекомендациям этой книги, весь диск отдан FreeBSD. А может быть, вы создали второй раздел для другой операционной системы или даже разбили диск на две секции FreeBSD. В мире Microsoft и Linux эти секции называются разделами (partitions), а на территории FreeBSD – *участками* (*slices*). Символы `s1` в предыдущем листинге представляют большие разделы, или участки. Диск ad0, представленный в листинге, имеет один участок ad0s1, который разбит на подразделы, отмеченные символами.

Во FreeBSD *раздел* – это сегмент внутри участка. Разделы внутри участка создаются во время установки системы. Каждый раздел имеет

уникальный файл устройства, название которого образуется добавлением уникального символа к названию файла участка. Например, разделы внутри участка */dev/ad0s1* представлены как */dev/ad0s1a*, */dev/ad0s1b*, */dev/ad0s1c* и т. д. Каждому разделу, создаваемому во время установки системы (*/usr*, */var* и другим), назначается соответствующий файл устройства.

Название файлу устройства, соответствующего тому или иному разделу, можно давать произвольным образом с некоторыми ограничениями. Традиционно файл устройства, имя которого заканчивается на *a* (в нашем примере */dev/ad0s1a*), – это корневой раздел (root), а файл с замыкающим символом *b* (*/dev/ad0s1b*) предназначен для раздела свопинга. Метка *c* служит признаком всего участка, от начала до конца, а символы от *d* до *h* можно назначить любому разделу по своему усмотрению. В одном участке может быть не больше восьми разделов, а на одном диске – не больше четырех участков. Например, файл устройства */dev/ad0s1a* соответствует диску с номером 0, участку 1, разделу 1 и, возможно, файловой системе root. Файл устройства */dev/ad1s2b* указывает на диск 2 и, возможно, участок свопинга.

Для приводов, отличных от ATA, вместо /dev/ad следует подставить соответствующие названия устройств.

Нумерация дисков ATA

То, что первый диск ATA будет называться */dev/ad0*, вовсе не означает, что у вас обязательно должен быть установлен жесткий диск */dev/ad0*. В моем ноутбуке жесткий диск имеет имя */dev/ad4*, потому что он подключен не к встроенному контроллеру ATA, а к контроллеру RAID. Жесткие диски SCSI и SAS нумеруются немного иначе – первый диск получает название */dev/da0* независимо от того, куда он подключен. При необходимости можно заставить систему нумеровать жесткие диски ATA, начиная с 0, для этого достаточно удалить параметр ядра `ATA_STATIC_ID`.

Таблица файловых систем: /etc/fstab

Так как же операционная система отображает все эти имена устройств на разделы жесткого диска? С помощью таблицы файловых систем */etc/fstab*. В этом файле каждая файловая система представлена отдельной строкой, в которой также находятся параметры, используемые командой mount(8). Ниже приводится пример одной из таких строк:

```
/dev/ad4s2a     /     ufs     rw     1     1
```

Первое поле в каждой строке соответствует имени устройства.

Второе поле указывает точку монтирования, или каталог, в котором можно найти файловую систему. Каждый раздел, куда можно записывать файлы, подключается к точке монтирования, например */usr*, */var* и т. д. Некоторые специальные разделы – например, разделы свопинга, могут не иметь точки монтирования (none). Вы не можете записывать файлы в пространство для свопинга, и вам придется решить, для каких целей будет использоваться раздел – либо для хранения файлов, либо для свопинга.

Далее следует тип файловой системы. Для стандартных разделов FreeBSD применяются файловые системы типа `ufs` (Unix Fast File System – быстрая файловая система UNIX). В примере ниже присутствуют `swap` (пространство для свопинга), `cd9660` (CD) и `nfs` (Network File System – сетевая файловая система). Перед монтированием раздела необходимо знать, какая файловая система на ней находится. Легко догадаться, что попытка монтировать дискету DOS как файловую систему FFS не даст удовлетворительных результатов.

В четвертом поле представлены параметры mount, которые применяются для данной файловой системы. Они описывают, как ядро должно интерпретировать файловую систему. Далее параметры монтирования будут обсуждаться более подробно, а здесь рассмотрим несколько специальных параметров, применяемых только в */etc/fstab*:

`ro`

Файловая система монтируется только для чтения. Даже root не может осуществлять запись в файлы этой файловой системы.

`rw`

Файловая система монтируется для чтения-записи.

`noauto`

Файловая система не будет монтироваться автоматически во время начальной загрузки или при запуске `mount -a`. Этот параметр применяется для приводов на сменных носителях, в которых во время загрузки системы может не оказаться сменного носителя.

Пятое поле сообщает программе dump(8), надо ли в этой файловой системе выполнять дамп. Если значение в этом поле равно `0`, программа dump не будет сохранять файловую систему. Если в поле стоит ненулевое значение, то заданное число определяет минимальный уровень дампа, необходимый для сохранения файловой системы. За более подробной информацией обращайтесь к главе 4.

Последнее поле `Pass#` сообщает системе, когда следует выполнять проверку целостности файловой системы во время начальной загрузки. Все файловые системы с одним и тем же значением `Pass#` проверяются утилитой fsck(8) параллельно. Только корневая файловая система имеет значение параметра `Pass#`, равное `1`, и она проверяется первой. Все остальные файловые системы имеют значение `2` в этом поле, и это означает, что они будут монтироваться после корневой файловой системы.

Пространство для свопинга и носители, доступные только для чтения, не требуют проверки на целостность, поэтому для них установлено значение `0`.

Теперь, имея необходимые сведения, можно взглянуть на полный файл */etc/fstab*.

```
# Device        Mountpoint      FStype      Options          Dump    Pass#
❶ /dev/ad4s1b   none            swap        sw               0       0
❷ /dev/ad4s2a   /               ufs         rw               1       1
❸ /dev/ad4s1a   /amd64          ufs         rw               2       2
  /dev/ad4s1f   /amd64/usr      ufs         rw               2       2
  /dev/ad4s1d   /amd64/var      ufs         rw               2       2
❹ /dev/ad4s1e   /tmp            ufs         rw               2       2
❺ /dev/ad4s2e   /usr            ufs         rw               2       2
❻ /dev/ad4s2d   /var            ufs         rw               2       2
❼ /dev/ad4s3d   /home           ufs         rw               2       2
❽ /dev/acd0     /cdrom          cd9660      ro,noauto        0       0
❾ data:/mp3     /mp3            nfs         rw,noauto,soft   0       0
```

Первая строка в этом листинге, */dev/ad4s1b* ❶, – это пространство для свопинга. Этот раздел никуда не монтируется, FreeBSD использует пространство для свопинга как дополнительную память.

Вторая строка ❷ – это корневой раздел. Обратите внимание на имя устройства – тогда как раздел для свопинга располагается в разделе `b` участка 1, корневая файловая система находится в разделе `a` участка 1. Корневой каталог и пространство для свопинга располагаются в разных участках!

Третий раздел – */dev/ad4s1a* ❸. Судя по имени, можно было бы ожидать, что в этом разделе будет размещаться корневая файловая система, но вместо этого он монтируется в каталог */amd64*. Следующие два раздела также располагаются в участке 1, но монтируются в подкаталоги каталога */amd64*.

Файловая система */tmp* ❹ – это раздел в участке 1, который содержит файловую систему каталога */amd64*.

Следующие строки, */usr* ❺ и */var* ❻, – это обычные разделы в участке 2.

Следующий раздел, */home* ❼, располагается на том же диске, в участке 3, в разделе `d`. И откуда здесь взялся третий участок?

Привод компакт-дисков монтируется в каталог */cdrom* ❽. Во время загрузки это устройство автоматически не монтируется.

Последняя строка начинается не с имени файла устройства. Эта строка соответствует сетевой файловой системе (Network File System – NFS) и говорит, что в каталог */mp3* будет смонтирован раздел `mp3`, расположенный на удаленном компьютере с именем `data`. О сетевых файловых системах мы поговорим ниже, в этой же главе.

Эта таблица файловых систем была взята с компьютера, где установлено сразу две операционных системы: FreeBSD/i386 и FreeBSD/amd64.

Именно поэтому данная таблица выглядит не совсем обычно. Я могу получить доступ к файловой системе amd64 и пользоваться пространством для свопинга amd64 при работе в системе FreeBSD/i386.

Что смонтировано сейчас?

Если не все файловые системы монтируются во время загрузки и системный администратор может выполнять монтирование дополнительных файловых систем, как узнать, какие файловые системы смонтированы в настоящий момент? Запуск команды mount(8) без аргументов позволит увидеть список всех смонтированных файловых систем.

```
# mount
/dev/ad4s2a on / (ufs, local)
devfs on /dev (devfs, local)
/dev/ad4s1e on /tmp (ufs, local, soft-updates)
/dev/ad4s2e on /usr (ufs, local, soft-updates)
/dev/ad4s2d on /var (ufs, local, soft-updates)
/dev/ad4s3d on /usr/home (ufs, local, soft-updates)
```

Здесь мы видим, что почти все файловые системы представляют разделы UFS. Слово `local` говорит о том, что разделы находятся на локальном жестком диске, подключенном к данному компьютеру. Здесь также видно слово `soft-updates`, эта особенность реализована во FreeBSD и будет рассматриваться далее в этой главе. Если при монтировании разделов применяются такие функции, как NFS или SMB, они будут показаны в выводе.

Вызов mount(8) – еще один быстрый способ получить не только имена устройств для каждого из разделов, но и информацию об использовании некоторых других функций.

Монтирование и демонтирование дисков

Команда mount(8) монтирует файловые системы, размещенные на дисковых устройствах. Если вам прежде никогда не приходилось экспериментировать с монтированием файловых систем, перезагрузите систему FreeBSD, выберите однопользовательский режим и продолжайте чтение.

При загрузке в однопользовательском режиме система монтирует корневой раздел в режиме «только для чтения». Этот раздел содержит всю необходимую информацию для базовой настройки, запуска основных служб системы и поиска остальных файловых систем. Однако эти остальные файловые системы остаются не смонтированными, поэтому их содержимое недоступно. Попробуйте заглянуть в каталог */usr*, и вы убедитесь, что этот каталог в однопользовательском режиме пуст. Система не потеряла содержимое этого каталога, просто раздел с этими файлами еще не смонтирован. Чтобы выполнить мало-мальски инте-

ресные операции в однопользовательском режиме, надо монтировать другие файловые системы.

Монтирование стандартных файловых систем

Чтобы вручную смонтировать файловую систему, перечисленную в файле */etc/fstab*, например */var* или */usr*, нужно вызвать команду mount(8) и передать ей в качестве аргумента имя файловой системы, которую требуется смонтировать.

```
# mount /usr
```

Эта команда смонтирует раздел с учетом всех дополнительных параметров, перечисленных в файле */etc/fstab*. Если необходимо смонтировать все файловые системы, перечисленные в */etc/fstab*, можно вызвать команду mount(8) с ключом `-a`.

```
# mount -a
```

Монтирование с параметрами

Иногда бывает необходимо смонтировать файловую систему, которая расположена в необычном месте. У меня такая потребность обычно возникает при установке нового диска. В этом случае команде mount(8) следует передать имя монтируемого устройства и каталог, в который оно будет смонтировано. Например, если предположить, что раздел */usr* находится на устройстве */dev/ad0s1e* и мне необходимо смонтировать его в каталог */mnt*, то я могу выполнить следующую команду:

```
# mount /dev/ad0s1e /mnt
```

Демонтирование раздела

Для отключения файловой системы используется команда umount(8), которая сообщает операционной системе о необходимости демонтировать раздел. (Обратите внимание: команда называется `umount`, а не `unmount`.)

```
# umount /usr
```

Если файловая система используется какой-либо программой, демонтировать ее будет невозможно. Если вам не удалось демонтировать раздел, вполне возможно, что какая-то программа работает с ним. Демонтировать раздел не удастся и тогда, когда текущим рабочим каталогом является каталог в этом разделе.

Насколько заполнен раздел?

Чтобы узнать объем свободного пространства на каждом разделе, можно воспользоваться командой df(1). Эта команда выведет список смонтированных разделов, объем использованного пространства на каж-

дом из них и точки монтирования. Один из неприятных моментов, связанных с командой df(1), заключается в том, что информация об объеме выводится в виде количества блоков, размером 1 Кбайт. Если раньше, когда диски были достаточно маленькими, это было вполне удобно, то сейчас числа получаются слишком большими, чтобы их можно было достаточно легко воспринимать одним взглядом. К счастью, команда df(1) имеет флаги `-h` и `-H`, которые вынуждают команду выводить информацию в удобочитаемом для человека виде. При использовании ключа `-h` один мегабайт вычисляется как степень 2 и составляет 1024 Кбайт, или 1 048 576 байт, а при использовании ключа `-H` мегабайт вычисляется как степень 10 и составляет 1000 Кбайт, или 1 000 000 байт. Сетевые администраторы и производители жестких дисков обычно используют степень 10, тогда как системные администраторы предпочитают использовать степень 2.[1] Если вы обладаете достаточно богатым опытом работы, вы уже знаете, какой ключ выбрать. Я – сетевой администратор, поэтому вам придется следовать моим предпочтениям при исследовании примеров, независимо от того, что думает по этому поводу мой технический редактор.

```
# df -H
Filesystem    Size    Used    Avail Capacity  Mounted on
/dev/ad4s2a   520M    301M    177M     63%    /
devfs         1.0k    1.0k      0B    100%    /dev
/dev/ad4s1e   520M    2.4M    476M      0%    /tmp
/dev/ad4s2e   11G     4.1G    5.9G     41%    /usr
/dev/ad4s2d   1.0G    322M    632M     34%    /var
/dev/ad4s3d   49G     43G     2.0G     96%    /usr/home
```

Здесь видны имена разделов, размер каждого раздела, объем использованного пространства, объем свободного пространства, процент использованного пространства и точки монтирования. Например, каталог */home* в данной системе заполнен на 96% и имеет еще 2 Гбайт свободного пространства. Корневой раздел заполнен всего на 63%, но свободным осталось только 177 Мбайт.

Файловая система FFS занимает порядка 8% дискового пространства для нужд оптимизации. Это пространство используется при перемещении файлов и для уменьшения степени фрагментирования. При переполнении диска вы можете даже увидеть отрицательный объем свободного дискового пространства. Когда это происходит, производительность диска падает катастрофически. Поэтому лучше всегда оставлять на разделах немного свободного пространства, чтобы FFS могла постоянно дефрагментировать себя. Хотя существует возможность уменьшить размер пространства, резервируемого файловой системой, но это отрицательно сказывается на производительности, и поступать

[1] Спорить на эту тему еще более бесполезно, чем, например, дискутировать на тему «Emacs против vi». Держу пари, что вы и не предполагали о существовании такого ключа!

так будет неразумно. Если вы действительно хотите попробовать уменьшить объем резервируемого пространства, обращайтесь к команде tunefs(8).

Очевидный вопрос: «Что такого хранится на диске, что заняло столько места?» Если ваша система напоминает мою, использованное пространство на диске продолжает расти без каких-либо видимых причин. С помощью команды `ls -l` можно отыскать отдельные крупные файлы, но рекурсивно выполнять эту команду в каждом каталоге как минимум непрактично.

Объем использованного пространства в отдельном каталоге можно получить с помощью команды du(1). Сначала результаты, которые дает эта команда, кажутся непонятными и могут отпугивать неопытных пользователей. Ниже приводится пример использования du(1) для поиска чего-то, что занимает почти все место в моем домашнем каталоге:

```
# cd $HOME
# du
1       ./bin/RCS
21459   ./bin/wp/shbin10
53202   ./bin/wp
53336   ./bin
5       ./.kde/share/applnk/staroffice_52
6       ./.kde/share/applnk
...
```

Команда продолжает и продолжает выводить информацию, перечисляя каталоги и их размеры в блоках. В данном случае каталог *$HOME/bin* занимает 53 336 блоков, или порядка 53 Мбайт. Я мог бы устроиться поудобнее и позволить команде du(1) перечислить все каталоги и подкаталоги, но тогда мне придется переварить информации гораздо больше, чем мне требуется в действительности. Кроме того, блоки – это не совсем удобная единица измерения, особенно, когда числовые значения выравниваются по левому краю.

Попробуем вывести результаты в удобочитаемом виде. Во-первых, команда du(1) поддерживает флаг `-h`, как и команда df. Кроме того, мне

$BLOCKSIZE

Многие утилиты, предназначенные для работы с дисками, отображают размеры в блоках размером по 512 байт, или полкилобайта. Если в переменную окружения `$BLOCKSIZE` записать значение `k`, команда df(1) и многие другие программы будут отображать размеры в блоках по 1 Кбайту, что намного удобнее. Можно использовать значение `1M`, тогда размеры будут выводиться в мегабайтах.

совсем не требуется рекурсивное перечисление всех вложенных подкаталогов. Управлять глубиной вложенности можно с помощью ключа `-d`. Этот ключ принимает один аргумент – глубину вложенности подкаталогов, которые требуется перечислить в выводе. Например, при использовании ключа `-d0` не будет перечислен ни один из вложенных подкаталогов, и команда просто выведет общий размер всех файлов в указанном каталоге.

```
# du -h -d0 $HOME
 37G     /home/mwlucas
```

В моем домашнем каталоге накопилось файлов на 37 Гбайт? Попробуем копнуть поглубже и найти наиболее объемный подкаталог.

```
# du -h -d1
 38K    ./bin
 56M    ./mibs
 ...
 34G    ./mp3
 ...
```

Похоже, мне придется искать другое место для хранения своих данных, поскольку все файлы в моем домашнем каталоге слишком важны для меня, чтобы удалять их.

Если вам флаг `-h` чем-то не нравится, для поиска наиболее объемного подкаталога можно использовать команду sort(1), например так: `du -kxd 1 | sort -n`.

Fast File System

Операционная система FreeBSD в качестве основной использует файловую систему Fast File System (FFS), которая является прямой наследницей файловой системы, распространявшейся в составе BSD 4.4. Один из авторов исходной файловой системы продолжает развивать эту файловую систему FreeBSD, добавив немало интересных функциональных возможностей за последние годы. Иногда FFS FFS называют UFS (UNIX File System). Многие системные утилиты по-прежнему называют разделы FFS разделами UFS. FreeBSD – не единственная операционная система, где продолжает использоваться файловая система 4.4 BSD или производные от нее. Если производитель UNIX не заявляет явно о своей «улучшенной и усовершенствованной» файловой системе, то почти наверняка он использует FFS.

При разработке FFS предполагалось создать быструю и надежную файловую систему, которой по силам эффективно справляться с обычными (и необычными) ситуациями. FreeBSD поставляется с FFS, которая сконфигурирована в расчете на извлечение максимальной пользы из современных аппаратных средств, однако FFS можно оптимизировать для обслуживания триллионов маленьких файлов или полудюжины файлов размером 30 Гбайт. Знать детали внутренней организации FFS

необязательно, но такие основные понятия, как блоки, фрагменты и индексные дескрипторы (inodes) знать необходимо.

Блоки – это сегменты диска, содержащие информацию. По умолчанию в системе FreeBSD блок имеет размер 16 Кбайт. Не все файлы имеют размеры, кратные 16 Кбайт, поэтому в файловой системе FFS для хранения остаточных кусков файлов используются *фрагменты*. Стандартный размер фрагмента составляет одну восьмую блока, или 2 Кбайт. Например, файл размером 20 Кбайт заполнит один блок и два фрагмента. *Индексные дескрипторы* – это специальные блоки, которые содержат базовую информацию о файлах, включая права доступа, размер и список блоков, занятых файлом. Данные, хранящиеся в индексных дескрипторах, называются *метаданными*. Это данные о данных (metadata). Такой способ хранения данных не уникален для FFS – другие файловые системы, такие как NTFS, также используют блоки для хранения данных и индексные дескрипторы. Однако в каждой файловой системе используется свой, уникальный принцип индексации.

Любая файловая система имеет определенное число индексных дескрипторов, число которых пропорционально размеру файловой системы. Современные диски могут иметь сотни тысяч индексных дескрипторов на каждом разделе, что необходимо для обеспечения возможности хранения сотен тысяч файлов. Если у вас имеется огромное число очень маленьких файлов, возможно, вам потребуется перестроить свою файловую систему, чтобы добавить в нее дополнительные индексные дескрипторы. Увидеть количество свободных индексных дескрипторов, доступных в файловой системе, можно с помощью команды `du -i`. Если вам действительно необходимо перестроить файловую систему, чтобы увеличить число индексных дескрипторов, обращайтесь к главе 18.

Виртуальные индексные дескрипторы

Во времена молодости UNIX, когда жесткие диски постоянно были подключены к компьютерам, индексные дескрипторы и блоки работали замечательно. Однако с годами вошло в практику перемещать диски между различными машинами и даже различными операционными системами. Набрали популярность компакт-диски со своей уникальной файловой системой, доступной только для чтения. Для дискет файловая система FAT32 стала стандартом. В других версиях UNIX разработаны свои варианты файловых систем. Поскольку BSD надо было «общаться» с разнообразными системами, потребовался иной уровень абстракции.

Такая абстракция была реализована в виде виртуальных индексных дескрипторов (virtual nodes), или vnodes. Виртуальными дескрипторами никогда не манипулируют напрямую, однако ссылки на них есть в документации системы. *Виртуальный дескриптор* играет роль переводчика информации, передаваемой между ядром и той или иной фай-

ловой системой. Каждый инструмент, который обращается к диску для чтения или записи данных, действует через виртуальные дескрипторы, которые устанавливают соответствие между данными и файловой системой на физическом носителе. Если файл записывается в файловой системе FFS, виртуальный дескриптор обращается к индексному, а если файл записывается в системе FAT32, виртуальный дескриптор обращается к таблице размещения файлов FAT32. Ссылки на индексные дескрипторы (inodes) можно встретить только в связи с файловыми системами FFS, а виртуальные дескрипторы (vnodes) применяются в работе с любой файловой системой.

Типы монтирования FFS

В отличие от файловых систем Windows, разделы FFS можно интерпретировать по-разному в зависимости от варианта монтирования. Способ монтирования раздела называется *типом монтирования* (*mount type*). Для монтирования файловой системы вручную надо указать параметры монтирования в командной строке, но при монтировании файловой системы, перечисленной в файле /etc/fstab, достаточно указать точку ее монтирования, при этом параметры монтирования должны быть указаны в поле `Options`.

Чтобы указать параметры монтирования в командной строке, команде mount следует передать ключ `-o` *`mounttype`* или определить параметры в файле */etc/fstab*, в поле `Options`.

Монтирование в режиме «только для чтения»

Если надо только просматривать содержимое диска и нет необходимости записывать данные, можно монтировать раздел «только для чтения». Бесспорно, это самый безопасный способ монтирования диска. В то же время это один из бессмысленных способов для большинства серверных приложений, поскольку невозможно изменять данные или записывать новые.

Многие системные администраторы монтируют «только для чтения» корневой раздел (root), и возможно, даже */usr*, чтобы минимизировать ущерб в результате выключения электропитания или ошибок в программном обеспечении. Даже если вы «потеряете» физический жесткий диск при всплеске напряжения либо из-за других неполадок в оборудовании, данные на пластинах останутся невредимы. В этом заключается преимущество монтирования в режиме «только для чтения»; недостатком является сложность эксплуатации такой файловой системы из-за невозможности записи на диски, монтированные «только для чтения»!

Монтирование в режиме «только для чтения» особенно удобно использовать при повреждении компьютера. Монтирование в этом режиме возможно даже тогда, когда операционная система отказывается монтировать поврежденные файловые системы в режиме чтения/записи. Это дает шанс восстановить данные с поврежденной системы.

Для монтирования файловой системы в режиме «только для чтения» следует использовать параметр монтирования `rdonly` или `ro`. Оба значения дают один и тот же эффект.

Синхронное монтирование

Синхронное (*sync*) *монтирование* – это старинный способ монтирования файловой системы. При записи на синхронно смонтированный диск ядро ждет завершения операции, чтобы сообщить программе о выполнении записи. Если запись не завершена, программа вправе предпринимать соответствующие действия.

Синхронное монтирование обеспечивает целостность данных в случае аварии, однако работа с такой файловой системой происходит медленно. Понятие «медленно» в наши дни относительно, т. к. даже дешевый диск может превосходить диски, которые считались лучшими несколько лет назад. Если вы скрупулезно относитесь к соблюдению целостности данных, задумайтесь о применении синхронного монтирования. Однако в большинстве случаев это избыточная мера.

Для монтирования файловой системы в синхронном режиме следует использовать параметр монтирования `sync`.

Асинхронное монтирование

В наше время *асинхронный режим монтирования* вытесняется такой функциональной возможностью, как soft updates, но вы по-прежнему можете услышать упоминания об этом режиме. Для ускорения доступа к данным, с более высоким риском их потери, монтируйте разделы асинхронно. Когда диск монтирован асинхронно, ядро записывает данные на диск и сообщает программе о том, что запись прошла успешно. При этом ядро не ждет от диска подтверждения того, что данные действительно записаны. Асинхронное монтирование – прекрасный выбор для второстепенных машин, однако оно не подходит для хранения важных данных. Разница в производительности между работой файловой системы, смонтированной в асинхронном режиме и в режиме noasync с использованием soft updates, очень невелика.

Для монтирования файловой системы в асинхронном режиме следует использовать параметр монтирования `async`.

Монтирование noasync

Наконец, рассмотрим комбинацию sync и async. Этот метод называется *noasync* и применяется во FreeBSD по умолчанию. При этом данные, имеющие отношение к индексным дескрипторам, записываются на диск синхронно, а фактические данные – асинхронно. В сочетании с soft updates (см. раздел «Soft Updates» далее в этой главе) режим noasync применяется для создания по-настоящему устойчивых систем.

Для монтирования файловой системы в режиме noasync никаких дополнительных параметров указывать не требуется.

Параметры монтирования FFS

В дополнение к типам монтирования система FreeBSD поддерживает несколько дополнительных параметров монтирования. Эти параметры оказывают влияние на поведение монтированных файловых систем.

noatime

В FFS у каждого файла есть «отпечаток» времени доступа, называемый *atime*, в котором отражается время последнего доступа к файлу. Если у вас большое количество файлов и нет необходимости в таких «отпечатках», можно монтировать диск с параметром noatime и не обновлять информацию о времени доступа. Это особенно удобно для работы с flash-накопителями или с дисками, которые испытывают серьезную нагрузку, например, с дисками, где хранятся буфера телеконференций Usenet.

noexec

Запрещает выполнение любых программ в этом разделе. В разделе */home*, монтированном с параметром noexec, пользователи не смогут запускать свои программы, однако для полного предотвращения таких действий параметр noexec следует применять при монтировании */tmp*, */var/tmp* и других разделов, в которые пользователи могут записывать свои файлы. Обратите при этом внимание, что данный параметр не предотвращает возможность запуска пользователями сценариев командного интерпретатора, которые представляют собой простой набор инструкций программы. Кроме того, параметр noexec иногда используется в случаях, когда на сервере имеются исполняемые двоичные файлы для других операционных систем или других аппаратных архитектур, и вам необходимо предотвратить возможность запуска этих файлов на сервере.

nosuid

Предотвращает запуск setuid-программ на вашей системе. setuid-программы разрешают пользователям запускать программы от имени других пользователей. Например, некоторые программы, такие как login(1), требуют привилегий пользователя root, но должны запускаться обычными пользователями. Очевидно, setuid-программы должны разрабатываться особенно тщательно, чтобы предотвратить возможность их взлома и применения для неавторизованного доступа к системе. Вот почему многие администраторы обычно делают недоступными все ненужные setuid-программы. Для этого можно применить nosuid, однако это не принесет пользы при наличии сценариев-оберток, таких как suidperl, которые дают возможность обойти это ограничение.

nosymfollow

Запрещает применять символические ссылки на файлы, или псевдонимы. *Символические ссылки* в основном применяются для создания

псевдонимов файлов, расположенных в других разделах. Для создания псевдонима файла в том же самом разделе лучше использовать обычные ссылки. Более подробную информацию о ссылках можно получить в странице руководства ln(1).

Для создания псевдонимов каталогов всегда используются символические ссылки, так как для этих целей жесткие ссылки (hard links) использовать нельзя.

Soft Updates и функция журналирования в FFS

Soft Updates – это технология организации и упорядочения операций записи на диск, при использовании которой метаданные файловой системы на диске остаются непротиворечивыми, производительность близка к производительности при асинхронном монтировании, а надежность соответствует надежности синхронного монтирования. Это не означает, что на диск будут записаны все данные – неполадки с электропитанием по-прежнему могут повредить данные. Однако Soft Updates предотвращают значительную часть трудностей. В случае отказа операционной системы, функция soft updates может продолжить работу со своей файловой системой и выполнить ее проверку. На мой взгляд, функция soft updates больше подходит для использования с разделами, чей объем не превышает 80 Гбайт.

Начиная с версии FreeBSD 7.0, файловая система FFS обзавелась поддержкой *журналирования*. Журналируемая файловая система записывает все изменения, производимые в файловой системе, на отдельный раздел диска, поэтому в случае неожиданного завершения работы системы измененные данные могут быть восстановлены автоматически во время загрузки. Благодаря этому отпадает необходимость запускать утилиту fsck(8). Любая журналируемая файловая система использует порядка 1 Гбайт для хранения журнала, что делает ее слишком расточительной для применения на небольших файловых системах. Однако принцип ведения журнала в системе FreeBSD отличается от аналогичных принципов, принятых в других журналируемых файловых системах, тем, что ведение журнала поддерживается на уровне диска, ниже уровня самой файловой системы. Такой способ ведения журнала является новинкой в FreeBSD и по-прежнему находится на экспериментальной стадии.

Например, к моменту написания этих строк, в моем распоряжении имеется компьютер регистрации. Большая часть дисковых разделов на нем имеет размер менее 10 Гбайт, но раздел */var* занимает почти два терабайта. Для раздела */var* я использую функцию журналирования, а для других – soft updates. По сравнению с запуском fsck в фоновом режиме восстановление данных в журналируемой системе после неожиданного падения напряжения выполняется быстро и безболезненно.

Подробнее о журналировании и функции gjournal мы поговорим в главе 18.

Кэширование записи

Файловая система FFS лучше всего работает на жестких дисках SCSI и SAS в силу устойчивости архитектуры SCSI. Не менее замечательно FFS работает на жестких дисках архитектуры ATA, за одним важным исключением: многие современные диски IDE поддерживают *кэширование записи*.

Диски IDE, поддерживающие кэширование записи, имеют встроенную микросхему, которая запоминает данные, предназначенные для записи на диск. Такой механизм может не подойти для функции Soft Updates, поскольку Soft Updates уповают на «честность» жесткого диска – когда тот сообщает о записи данных на диск, механизм Soft Updates предполагает, что данные находятся на пластине. Однако на самом деле механизм кэширования записи IDE сообщает о благополучном сохранении данных в дисковом кэше, а не о том, что данные записаны на диск. Фактическая запись данных на диск может произойти спустя секунду или более.

Этот разрыв во времени не опасен при единичных случаях его возникновения, однако на сервере это происходит постоянно. Поэтому, если вы беспокоитесь о своих данных, отключите кэширование записи, добавив в */boot/loader.conf* следующую строку:

```
hw.ata.wc=0
```

Отключение кэширования записи замедлит работу диска IDE, однако ваши данные будут в безопасности. Я благополучно применяю кэширование записи на настольном компьютере и ноутбуке, где данные не записываются на диск постоянно, однако на сервере ни в коем случае нельзя оставлять кэширование записи включенным. Вы можете сообщить своему руководству, что диски ATA не обеспечивают необходимой для сервера производительности, и вам требуется дополнительное аппаратное обеспечение или сказать, что из-за перегрузки аппаратных средств происходят потери данных. Лично я предпочитаю первый вариант.

Мгновенные снимки файловой системы

Файловая система FFS с активированной функцией Soft Updates может создавать мгновенные образы диска, или снимки (*snapshot*), в конкретные моменты времени. Эти снимки можно найти в каталоге *.snapshot*, который располагается в корневом каталоге каждого раздела.

Хотя вы можете не производить администрирование этих снимков, они используются различными инструментальными средствами во время работы. Например, dump(8) выполняет архивирование снимков, а не работающей файловой системы, благодаря чему обеспечивается внутренняя непротиворечивость резервных копий. Утилита fsck, работающая в фоновом режиме (о чем будет говориться ниже, в этой

же главе), также использует снимки в своей работе. Снимки можно монтировать и получать отображение файловой системы в конкретный момент времени. Хотя это достаточно интересная информация, но большинство системных администраторов считают снимки не слишком полезными. Тем не менее при работе с FreeBSD вы частенько будете встречать упоминания о снимках.

«Грязные» диски

Нет, диски не пачкаются при эксплуатации (хотя пыль на пластине причинит ей вред, да и вода, безусловно, будет некстати). «Грязный» дисковый раздел FFS находится в неопределенном состоянии: операционная система запрашивает информацию, которая находится в этом разделе, но данные не были записаны полностью. Возможно, записана только часть блоков данных, либо изменен индексный дескриптор, а данные не записаны, либо имеется комбинация этих двух вариантов. Если при записи на диск выключается питание, в системе появляется *«грязный» диск*.

fsck(8)

В состав операционной системы FreeBSD входит мощный инструмент для проверки файловых систем – fsck(8). Если при загрузке системы будет найден «грязный» диск, fsck автоматически проверит диск и постарается устранить неполадки. Данные, не записанные на диск, будут утеряны, но fsck сделает все возможное, чтобы привести в порядок те данные, что остались. При благоприятном исходе все встанет на свои места – за исключением незаписанных данных!

Неудачный автоматический запуск fsck

Иногда перезагрузка заканчивается неудачно, и вы попадаете в однопользовательский режим, где вас просят запустить fsck(8) вручную. Здесь у вас есть выбор: либо запустить fsck, либо нет. Если ввести команду `fsck`, эта программа проверит каждый блок и индексный дескриптор на диске. Вероятно, она найдет все блоки, не связанные с соответствующими индексными дескрипторами. Далее программа постарается выяснить былое соответствие. Однако она не сможет сказать, к какому каталогу принадлежат эти файлы.

fsck спросит, надо ли выполнить перегруппировку. Если вы ответите `n`, программа удалит поврежденные файлы. В случае ответа `y` потерянный файл будет помещен в каталог находок (например, */usr/lost+found*) в разделе, в котором он был найден. В качестве названия файла будет выступать некое число. Если таких файлов немного, их можно попробовать идентифицировать вручную, а в случае большого числа файлов их можно просмотреть с помощью таких инструментов, как file(1) или grep(1), и произвести идентификацию.

Отключение приглашения fsck

Если диск стал «грязным» во время масштабной операции, на нем может появиться множество потерянных файлов. Чтобы не тратить время, снова и снова набирая `y` и предписывая fsck восстанавливать каждый файл, можно запустить команду `fsck -y`, заранее ответив `y` на все вопросы. Это намного проще, чем сидеть и без конца набирать `y`.

Систему можно настроить на автоматический запуск `fsck -y` при начальной загрузке. Однако я не рекомендую такой шаг: если есть хотя бы малейший шанс, что моя файловая система вознесется в цифровые небеса, я хочу знать об этом. Я хочу сам набирать опасную команду и трепетать, наблюдая за работой fsck(8). Кроме того, всегда неприятно обнаружить «мусор» в системе, не имея ни малейшего понятия о том, что произошло. Однако если вы более безрассудны, чем я, установите параметр `fsck_y_enable="YES"` в файле */etc/rc.conf*.

> **ВНИМАНИЕ!**
>
> Запуск fsck с ключом `-y` не гарантирует полную безопасность. Когда я работал с экспериментальными файловыми системами в ветке -current или выполнял не совсем обычные операции, благодаря `fsck -y` в каталогах */usr/lost+found* и */var/lost+found* иногда оказывалось содержимое всего диска. После этого восстанавливать файлы было чрезвычайно трудно. Следует заметить, что в стабильной версии FreeBSD-stable со стандартной файловой системой UFS я не испытывал никаких трудностей.

Что вместо fsck –y

Какой у вас есть выбор вместо `fsck -y`? Программы fsdb(8) и clri(8) позволяют отлаживать файловую систему и перенаправлять файлы в надлежащее место. Можно восстановить имена файлов и разместить их в исходных каталогах. Однако это трудно[1] – такой путь рекомендуется только Мастерам Файловых Систем из Тайного Клана Ниндзя.

Запуск fsck в фоновом режиме

Возможность запуска fsck в фоновом режиме придает файловой системе FFS некоторые преимущества журналируемых файловых систем, без фактического использования части жесткого диска для ведения журнала. Когда после загрузки FreeBSD обнаруживает наличие fsck, работающей в фоновом режиме, она монтирует «грязные» диски в режиме

[1] В первом издании книги я сравнил использование fsdb(8) и clri(8) с восхождением на Эверест в сандалиях и шортах. Со временем мне не раз пришлось убедиться, что я был неправ – это напоминает восхождение на Эверест в сандалиях, но без шорт.

чтения/записи. Утилита fsck(8) работает в фоновом режиме, несмотря на то, что сервер продолжает свою работу, отыскивает потерянные куски файлов и ставит их на место.

Фактически при работе в фоновом режиме fsck проходит две основных стадии. Когда во время инициализации процесса загрузки FreeBSD обнаруживает «грязные» диски, она запускает fsck(8) для выполнения предварительной оценки их состояния. fsck(8) решает, возможно ли их восстановление в ходе работы системы или требуется полноценный запуск fsck в однопользовательском режиме. В большинстве случаев fsck предполагает, что работа может быть продолжена, и дает разрешение на продолжение загрузки системы. Когда система достигает однопользовательского режима, fsck запускается в фоновом режиме с низким приоритетом и выполняет проверку разделов одного за другим. Результаты работы fsck выводятся в файл протокола */var/log/messages*.

Пока fsck работает в фоновом режиме, можно ожидать снижения производительности приложений, выполняющих действия с жестким диском. fsck(8) забирает на себя значительную долю производительности диска. Если система медленная, она станет еще медленнее.

По окончании работы fsck в фоновом режиме можно заглянуть в файл */var/log/messages* и проверить наличие сообщений об ошибках. На предварительном этапе fsck может ошибиться в своих оценках, и, возможно, для восстановления раздела действительно потребуется полноценный запуск fsck в однопользовательском режиме. Если будет обнаружено подобное сообщение, предусмотрите несколько часов простоя системы на проведение восстановительных работ. Очень неприятно останавливать систему для проведения запланированных работ, однако гораздо неприятнее производить незапланированную остановку, вызванную падением напряжения и работой команды `fsck -y` в однопользовательском режиме.

Принудительное монтирование «грязных» дисков в режиме чтения/записи

Если вам очень захочется вынудить FreeBSD монтировать «грязный» диск в режиме чтения/записи без использования fsck в фоновом режиме, вы можете это сделать. Но результаты вам не понравятся. Очень. Но поскольку о такой возможности упоминается в странице руководства mount(8), некоторые читатели могут решить, что в этом нет ничего плохого, если заранее не знают, чем это грозит. Для принудительного монтирования в режиме чтения/записи используются флаги `-w` (read-write – чтение/запись) и `-f` (force – принудительно) команды mount(8).

Монтирование «грязного» раздела в режиме чтения/записи приведет к повреждению данных. Обратите внимание на отсутствие в этом предложении слова *может*. Кроме того, в этом предложении отсутствует слово *восстановление*. Монтирование «грязной» файловой системы может привести к краху системы. Эта операция может разрушить еще

оставшиеся на разделе данные и даже уничтожить саму файловую систему. Принудительное монтирование «грязной» файловой системы в режиме чтения/записи – это табу. Никогда не производите его.

Синхронизация и остановка FFS

Во время остановки операционной системы FreeBSD ядро производит синхронизацию всех данных на жестком диске, помечает диск как «чистый» и останавливается. Эти операции с диском выполняются процессом ядра с именем *syncer*. В ходе подготовки к остановке системы syncer сообщает о ходе синхронизации данных на жестком диске.

В процессе остановки вы будете наблюдать малопонятные сообщения от процесса syncer. В действительности syncer не опускается до работы с блоками и индексными дескрипторами, требующими обновления, – он работает с виртуальными индексными дескрипторами, требующими синхронизации. Благодаря механизму Soft Updates операция записи в один виртуальный индексный дескриптор может приводить к появлению другого виртуального дескриптора, требующего обновления. Вы можете увидеть, как быстро уменьшающееся число буферов, требующих записи на диск, вдруг раз-другой прыгает между нулем и каким-нибудь небольшим значением – это результат фактической синхронизации с жестким диском.

Подлинные подробности о FFS

Если вы хотите побольше узнать о FFS, загрузите схему внутреннего устройства ядра FFS, которую можно найти в документе *http://phk.freebsd.dk/misc/ufs.pdf*. Чтобы распечатать ее, вам потребуется широкоформатный архитектурный принтер или 18 листов обычной бумаги, а также новая лента для принтера.

Запуск fsck в фоновом режиме, fsck –y, обычный запуск fsck – вот напасть–то!

В реальной жизни могут встретиться все эти варианты использования fsck(8), но как понять, когда FreeBSD использует каждую из этих команд? При принятии решения, когда и как запускать fsck(8), FreeBSD использует следующие условия:

- Если файловая система не содержит ошибок, она монтируется без запуска fsck(8).
- Если во время загрузки файловая система не задействует механизм Soft Updates и окажется «грязной», FreeBSD запускает fsck. Если будут обнаружены серьезные повреждения файловой системы, FreeBSD остановится и запросит вашего вмешательства. Здесь вы

сможете либо запустить команду `fsck -y`, либо вручную подтвердить все попытки восстановления файлов.

- Если во время загрузки файловая система задействует механизм Soft Updates и окажется «грязной», FreeBSD выполнит оценочную проверку с помощью fsck(8). В случае умеренных повреждений FreeBSD запустит fsck(8) в фоне, в многопользовательском режиме. Если будут обнаружены серьезные повреждения, FreeBSD прервет загрузку и потребует вмешательства либо в виде запуска команды `fsck -y`, либо в виде принятия или отклонения каждой обнаруженной проблемы.
- Если во время загрузки файловая система задействует механизм журналирования и окажется «грязной», FreeBSD восстановит данные из журнала и продолжит загрузку. Журналируемые файловые системы редко требуют восстановления с помощью fsck(8).

Использование неродных файловых систем

Для наших целей любой раздел или диск, который не является разделом FFS, есть *неродная (foreign) файловая система*. К счастью, во FreeBSD реализована серьезная поддержка неродных файловых систем. Отметим, что будут работать только те функции, которые поддерживаются в самой файловой системе. Например, файловая система FAT32 компании Microsoft не поддерживает права доступа, а файловые системы, используемые в операционной системе Linux, не поддерживают флаги файлов, характерные для BSD.

Каждая неродная файловая система должна иметь поддержку в ядре FreeBSD. К счастью, в случае необходимости программа mount(8) автоматически загружает соответствующий модуль ядра.

Для монтирования неродной файловой системы необходима та же информация, что и для монтирования FFS: имя устройства и точка монтирования. Также необходимо знать тип файловой системы и имя команды, которая монтирует файловые системы данного типа. Например, для монтирования CD-ROM в системе FreeBSD имеется точка монтирования */cdrom*. Первое устройство IDE CD в системе имеет имя */dev/acd0*. Для компакт-дисков используется файловая система ISO 9660, а команда FreeBSD, которая выполняет их монтирование, имеет вид: `mount -t cd9660(8)`. Следующая команда выполняет монтирование компакт-диска в каталог */cdrom*:

```
# mount -t cd9660 /dev/acd0 /cdrom
```

Теперь можно перейти в каталог */cdrom* и ознакомиться с его содержимым. Просто, не правда ли?

При попытке монтировать диск командой, не поддерживаемой в данной файловой системе, будет выдано сообщение об ошибке. Например, у меня дома все дискеты содержат либо файловую систему FAT32, ли-

бо FFS. Накопителю на гибких магнитных дисках соответствует файл устройства */dev/fd0*, а каталог */media* является стандартной точкой монтирования дискет.

```
# mount /dev/fd0 /media
mount: /dev/fd0 on /media: incorrect super block
(Перевод: монтирование: /dev/fd0 в /media: некорректный суперблок)
```

Эта дискета содержит файловую систему FAT32. Если бы я запустил команду `mount -t msdosfs`, никаких ошибок не было бы.

Независимо от типа монтированной файловой системы ее можно размонтировать с помощью umount(8):

```
# umount /cdrom
```

Утилита umount(8) не делает никаких предположений о типе файловой системы. Она просто отключает раздел диска.

Поддерживаемые типы неродных файловых систем

Ниже приводится список наиболее распространенных неродных файловых систем с кратким описанием и соответствующими командами монтирования.

FAT (MS-DOS)

Во FreeBSD реализована значительная поддержка файловой системы FAT (DOS/Windows 9x File Allocation Table), обычно используемой для сменных носителей и в случае одновременной установки на компьютер двух операционных систем. Данная поддержка охватывает такие разновидности FAT, как FAT12, FAT16 и FAT32. Впрочем, *можно* форматировать дискеты в FFS, поэтому не надо слепо полагать, что все дискеты имеют формат FAT. Поскольку дискеты чаще всего используются для переноса файлов с одного компьютера на другой, то *обычно* они форматируются в файловой системе FAT32. Тип монтируемой системы – `msdosfs` (`mount -t msdosfs`).

Если вам приходится монтировать много устройств с файловой системой FAT32, ознакомьтесь как следует с каталогом */usr/ports/tools/mtools*. Здесь собраны программы для работы с файловой системой FAT, обладающие более высокой гибкостью, чем стандартные инструменты FreeBSD.

ISO 9660

ISO-9660 – это стандартная файловая система для компакт-дисков. FreeBSD позволяет считывать данные с CD-ROM, а также записывать их при наличии записывающего привода. Чуть ли не каждый компакт-диск, с которым вы столкнетесь, будет иметь формат ISO 9660. Команда монтирования – `mount -t cd9660`.

Пакет инструментальных средств cdrtools, который можно найти в */usr/ports/sysutils/cdrtools*, содержит множество полезных инструментов для работы с образами компакт-дисков, включая средства создания ISO-образов из файлов на диске.

UDF

Файловая система UDF (Universal Data Format – универсальный формат хранения данных) появилась как замена ISO 9660. Файловую систему UDF можно обнаружить на некоторых DVD-дисках и flash-устройствах с интерфейсом USB, емкость которых превышает 32 Гбайт – верхней границы, поддерживаемой файловой системой FAT32. С ростом емкости сменных носителей вам все чаще и чаще будет встречаться файловая система UDF. Команда монтирования: `mount -t udf`.

NTFS

Стандартная файловая система Windows NT/2000/XP, NTFS, тесно интегрирована с ядром Microsoft. Чтобы записывать данные в раздел NTFS, необходимо глубоко понимать работу этой файловой системы. К сожалению, Microsoft не предоставляет такую информацию, поэтому FreeBSD может безопасно монтировать разделы NTFS в режиме «только для чтения», а функциональность режима чтения/записи имеет существенные ограничения. Команда монтирования – `mount -t ntfs`.

Возможность монтирования разделов NTFS наиболее часто бывает необходима при переходе с систем Windows на систему FreeBSD – просто извлеките жесткий диск из Windows-машины, смонтируйте его в FreeBSD-машину и скопируйте необходимые данные. Кроме того, поддержка NTFS бывает необходима, когда на компьютере установлены сразу две операционные системы, и одна из них – Windows.

Так как спецификации NTFS закрыты, а кодирование данных производится малоизвестным способом, поддержка NTFS в системе FreeBSD не гарантирует корректную работу.

ext2fs и ext3fs

Стандартные файловые системы Linux – ext2fs и ext3fs – поддерживают многие функции файловой системы FreeBSD, позволяя безопасно читать и записывать данные. Подобно монтированию NTFS монтирование файловых систем Linux довольно полезно при чрезвычайных обстоятельствах или при наличии двух операционных систем на компьютере. Несмотря на различия в названиях, обе файловые системы монтируются одной и той же командой `mount -t ext2fs`.

Пользователи файловых систем Linux могут найти полезные инструментальные средства в каталоге */usr/ports/sysutils/e2fsprogs*. Кроме всего прочего эти инструменты позволят запускать команду fsck(8) и выполнять оценку повреждений файловых систем Linux.

ReiserFS

Файловая система ReiserFS имеет не слишком много приверженцев среди пользователей Linux. FreeBSD поддерживает монтирование разделов ReiserFS исключительно в режиме «только для чтения». Поддержка реализована непосредственно в программе mount(8). Команда монтирования: `mount -t reiserfs` *раздел точка_монтирования*.

XFS

Операционная система FreeBSD поддерживает возможность чтения разделов с файловой системой XFS, созданной компанией SGI, однако, к моменту написания этих строк, реализация операции записи находилась еще на экспериментальной стадии. Файловая система XFS – это самая старая журналируемая файловая система, с хорошо отлаженной реализацией. Однако XFS распространяется на основе лицензии GPL, что делает маловероятным включение этой файловой системы в базовую систему FreeBSD. Журналирование в FreeBSD поддерживается командой gjournal, рассматриваемой в главе 18.

Программы форматирования, монтирования и управления разделами XFS можно найти в каталоге */usr/ports/sysutils/xfsprogs*.

ZFS

Начиная с версии 7.0, операционная система FreeBSD включает в себя экспериментальную поддержку файловой системы ZFS, перенесенной с платформы OpenSolaris. Хотя инсталлятор FreeBSD и не поддерживает ZFS, тем не менее в случае необходимости вы сможете использовать расширенные возможности ZFS. Порядок лицензирования этой файловой системы не позволяет сделать ее основной в системе FreeBSD, однако ее высокая производительность может оказаться востребованной для определенных применений в многотерабайтных файловых системах и в 64-битных системах. В 32-битных системах ZFS испытывает некоторые проблемы с памятью, но в 64-битных системах репутация ZFS безукоризненна.

Права доступа и неродные файловые системы

Метод, применяемый для монтирования файловой системы, и пользователь, который ее монтирует, определяют права доступа монтированной файловой системы. Например, XFS и ext3fs хранят права доступа в файловой системе, ставя их в соответствие идентификаторам пользователей (UIDs). Поскольку права доступа в них очень напоминают права доступа в FFS, а вся информация о правах доступа находится в самой файловой системе, FreeBSD соблюдает эти права доступа.

Файловая система NTFS имеет свою систему прав доступа. Поскольку NTFS лишь отдаленно похожа на UNIX-подобные системы, права доступа NTFS будут отброшены при монтировании этой файловой системы во FreeBSD. Она будет рассматриваться как аналог CD-ROM или дискеты.

По умолчанию монтировать файловые системы может только root. Он же владеет всеми неродными файловыми системами. Если это не соответствует вашим предпочтениям, то с помощью ключей `-u` и `-g` можно указать ID пользователя и группы при монтировании файловых систем FAT32, NTFS или ISO 9660. Например, следующая команда подойдет при монтировании устройства USB с файловой системой FAT32 для пользователя «cstrzelc», причем этот пользователь сможет редактировать содержимое дискеты:

```
# mount -t msdosfs -u cstrzelc -g cstrzelc /dev/da0 /mnt
```

Теперь пользователь cstrzelc является владельцем файлов этой дискеты.

Монтируя носители под пользователей, можно заработать мигрень, особенно если это приходится делать на нескольких десятках компьютеров. Чтобы разрешить пользователю монтировать файловые системы, присвойте параметру sysctl `vfs.usermount` значение `1` и убедитесь, что пользователь владеет точкой монтирования. В результате пользователь cstrzelc не сможет монтировать сменные носители в каталог */mnt*, зато сможет монтировать их в каталог */home/cstrzelc/mnt*.

Файловые системы на съемных носителях

Популярность съемных носителей сильно выросла за последние несколько лет. Совсем недавно мы носились с дискетами, а теперь у нас есть компакт-диски и устройства USB. Вы должны уметь управляться с любыми съемными носителями, которые будут пересекать порог вашего информационного центра. Далее мы рассмотрим работу с файловыми системами на дискетах, устройствах USB и компакт-дисках.

Из соображений безопасности я не рекомендую подключать съемные носители к рабочим серверам, если на то нет веских оснований. Кто знает, что в действительности находится на устройстве USB данного производителя? Я предпочитаю сначала смонтировать такие устройства на своей рабочей станции, проверить содержимое и лишь затем скопировать необходимые данные на сервер. Съемные носители чрезвычайно удобны и распространены, и, конечно, указанное выше правило не действует, когда речь заходит о моем личном устройстве USB.

Форматирование носителей с файловой системой FAT32

И дискеты, и устройства USB обычно форматируются в файловой системе FAT32. В устройствах USB используется файловая система FAT32, и, как правило, они продаются уже отформатированными. Так как устройства USB имеют ограниченное число циклов чтения/записи, не рекомендуется переформатировать их слишком часто. Это правило не относится к дискетам, которые часто нуждаются в переформатировании при интенсивном использовании. Процесс, который большинство пользователей Windows называют «форматированием дискеты», на самом деле является многоэтапным. Под этим подразумевается форматирова-

ние диска, присвоение диску метки и создание файловой системы. Во FreeBSD при создании дискеты надо выполнить все эти операции.

Нестандартные дискеты

Предположим, что есть гибкий диск емкостью 1,44 Мбайт, который был стандартом для аппаратных средств x86 свыше двух десятилетий. Если у вас имеется привод, рассчитанный на работу с дискетами емкостью 800 Кбайт или какой-нибудь другой нестандартной емкостью, потребуется внести поправки, однако последовательность шагов не изменится.

Низкоуровневое форматирование

При форматировании диска первым делом надо выполнить низкоуровневое форматирование с помощью fdformat(1). Эта программа принимает только два аргумента: размер дискеты и имя устройства.

```
# fdformat -f 1440 /dev/fd0
Format 1440K floppy '/dev/fd0.1440'? (y/n): y
```

Когда вы наберете y, fdformat(1) начнет низкоуровневое форматирование, подготавливая диск к переносу на него файловой системы. Низкоуровневое форматирование – самая медленная операция в процессе подготовки дискеты. Windows выполняет такое форматирование, только когда будет затребовано полное форматирование.

Создание файловой системы FFS

После низкоуровневого форматирования, если вы создаете дискету FFS, то диску надо присвоить метку с помощью disklabel(8). Эта программа записывает на дискету базовую идентификационную информацию, устанавливает данные о разделах и даже может обозначить диск как загрузочный. Обозначение диска в качестве загрузочного не подразумевает копирование на него необходимых программ; маркер, размещаемый на диске, нужен для того, чтобы системный BIOS воспринял этот диск как загрузочный. Если вам нужна загрузочная дискета, скопируйте соответствующий образ диска из установочного комплекта. В следующем примере мы добавим простую метку диска без каких бы то ни было специальных характеристик:

```
# disklabel -r -w /dev/fd0 fd1440
```

Ключ -r предписывает disklabel обратиться к диску на низком уровне. Это необходимо, поскольку на диске еще нет файловой системы. Ключ -w предписывает осуществить запись на диск: мы записываем в */dev/fd0* и устанавливаем стандартную метку для дискеты 1,44 Мбайт. Полный список меток дискет можно найти в */etc/disktab*. Там же располагаются метки для устройств других типов.

Наконец, создайте файловую систему с помощью newfs(8).

```
# newfs /dev/fd0
```

Далее вы увидите несколько строк текста, которые выведет программа newfs, после чего управление вернется командной строке.

Следует иметь в виду, что при использовании FFS на дискете впустую расходуется некоторое пространство, кроме того, FFS не обеспечивает должной защиты файлов на дискете, как того можно было бы ожидать. Так как информация о правах доступа сохраняется на дискете, обладатель дискеты легко может их преодолеть. Кроме того, не забывайте, что FFS резервирует порядка 8% общей емкости диска для собственных нужд. Вам действительно нужна дискета емкостью 1.32 Мбайт? Файловая система FFS хороша, но для гибких дисков лучше подойдет FAT32.

Создание файловой системы FAT32

Для перемещения данных между разными компьютерами требуется подготовить дискету в формате FAT32. Как и в предыдущем случае, сначала необходимо выполнить низкоуровневое форматирование, как уже обсуждалось ранее, однако в данном случае не надо присваивать метку. Просто воспользуйтесь программой newfs_msdos(8):

```
# newfs_msdos /dev/fd0
```

После того как по экрану пробежит несколько строк текста, вы получите готовую дискету.

Использование съемных носителей

Работа со съемными носителями ничем не отличается от работы с обычными устройствами, такими как жесткие диски. Вам требуется знать тип файловой системы на устройстве, имя файла устройства и точку монтирования.

Для компакт-дисков используется файловая система ISO 9660, тогда как для дисков DVD – либо UDF, либо комбинация ISO 9660 и UDF. Для устройств USB и дискет обычно используется FAT32. На новейших устройствах USB может использоваться файловая система UDF. Я предполагаю, что UDF станет наиболее привычной файловой системой для съемных устройств, особенно с увеличением емкости flash-накопителей.

Имя файла устройства зависит от типа самого устройства. Приводы компакт-дисков с интерфейсом IDE получают имя */dev/acd0*, тогда как приводы с интерфейсом SCSI – */dev/cd0*. Накопителям на гибких магнитных дисках соответствует файл устройства */dev/fd0*. Устройства USB получают первое свободное имя, производное от */dev/da*. При подключении устройства USB в файле протокола */var/log/message* и на экране консоли появляется сообщение с указанием имени файла устройства и его типа.

Последнее, что вам потребуется, – это точка монтирования. По умолчанию в системе FreeBSD присутствует точка монтирования */cdrom* для монтирования дисков CD и DVD. Здесь также можно обнаружить точку монтирования */media*, которая предназначена для монтирования *съемных носителей*. Вы можете создать дополнительные точки монтирования, какие только пожелаете, поскольку любая точка монтирования – это самый обычный каталог. Для монтирования разнообразных устройств на короткий срок FreeBSD предлагает */mnt*.

Так, чтобы смонтировать устройство USB */dev/da0* с файловой системой FAT32 в точку монтирования */media*, следует запустить следующую команду:

```
# mount -t msdosfs /dev/da0 /media
```

Иногда встречаются устройства, которые содержат настоящую таблицу участков, и такие устройства должны монтироваться как */dev/da0s1*, а не как */dev/da0*. Это зависит от того, как было отформатировано устройство, а не от самой FreeBSD.

Единственный раздражающий фактор, связанный с приводами компакт-дисков, состоит в том, что приводам с интерфейсами SCSI и ATA соответствуют разные программные интерфейсы. Значительная часть программного обеспечения написана в расчете исключительно на интерфейс SCSI, поскольку этот интерфейс считается более надежным. Если вам придется столкнуться с таким программным обеспечением, обратите внимание на модуль ядра atapicam(4), который реализует уровень эмуляции для приводов CD-ROM с интерфейсом ATA.

Извлечение съемных носителей

Перед извлечением съемного носителя необходимо демонтировать файловую систему. Лоток CD-ROM не удастся выдвинуть, если компакт-диск не будет демонтирован, аналогичным образом не удастся извлечь дискету из накопителя. Безусловно, устройство USB можно извлечь в любой момент, но если файловая система в этот момент не будет демонтирована, это может привести к краху системы и даже к повреждению данных на устройстве. Чтобы демонтировать файловую систему, используйте команду umount(8) как для любой другой файловой системы:

```
# umount /cdrom
```

Съемные носители и /etc/fstab

Можно упростить монтирование съемных носителей, добавив соответствующие записи в файл */etc/fstab*. Если для файловой системы на съемном носителе имеется запись в */etc/fstab*, то при монтировании можно будет не указывать имя устройства и тип файловой системы. Другими словами, не надо помнить имена всех устройств и точные команды для монтирования тех или иных файловых систем. Вероятно, в вашем файле */etc/fstab* уже имеется запись для монтирования CD-ROM.

```
/dev/acd0       /cdrom          cd9660          ro,noauto       0       0
```

Я уверен, что вы помните назначение каждого поля в */etc/fstab*, тем не менее напомню, что данная запись означает следующее: «монтировать */dev/acd0* в */cdrom*, с использованием файловой системы ISO 9660, в режиме только для чтения, и не монтировать автоматически это устройство во время загрузки». Используя этот пример в качестве шаблона, вы сможете дополнить */etc/fstab* аналогичными записями для устройств USB и накопителей на гибких магнитных дисках.

```
/dev/fd0        /mnt            msdosfs         rw,noauto       0       0
❶/dev/da0       /medi           msdosfs         rw,❷noauto      0       0
```

В системе FreeBSD эти записи по умолчанию отсутствуют, но я нахожу очень удобным иметь их в системах, где регулярно приходится пользоваться съемными носителями. Проверьте, что следующее доступное устройство da – это именно */dev/da0* ❶, иначе, если этот файл устройства соответствует уже смонтированному жесткому диску, данная запись окажется бесполезной.

Когда будете добавлять записи в файл */etc/fstab*, не забудьте добавить флаг `noauto` ❷. В противном случае при отсутствии съемного носителя загрузка остановится в однопользовательском режиме из-за отсутствия файловой системы.

Прочие файловые системы FreeBSD

Помимо FFS и неродных файловых систем, FreeBSD поддерживает различные файловые системы специального назначения. Некоторые из них интересны сами по себе, но обычно не используются, например mount_umapfs(8), а другие используются очень часто или даже используются в обязательном порядке. Далее мы поговорим о файловых системах в памяти – популярном способе оптимизации производительности, который используется в некоторых случаях, о создании файловых систем внутри файлов, а также о файловых системах procfs и fdescfs, которые необходимы для нормальной работы некоторых программ.

Файловые системы в памяти

Операционная система FreeBSD позволяет создавать файловые системы не только в дисковых разделах, но также в памяти, в файлах или в комбинации того и другого. Одно из наиболее популярных применений этой возможности заключается в создании *файловых систем в памяти* (*memory filesystems*), или *дисков памяти* (*memory disks*). Операции чтения и записи в файлы, находящиеся в памяти, выполняются намного быстрее, чем доступ к файлам, находящимся на диске, что позволяет существенно повысить производительность некоторых приложений. Однако при завершении работы системы все содержимое диска памяти теряется безвозвратно.

/tmp как диск памяти

Наиболее часто файловые системы в памяти используются для размещения каталога */tmp*. Делается это так часто, что FreeBSD имеет для этих целей даже специальный параметр в *rc.conf*. Взгляните на следующие определения в *rc.conf*:

```
❶ tmpmfs="YES"
❷ tmpsize="20m"
❸ tmpmfs_flags="-S"
```

Установив значение `YES` в параметре `tmpmfs` ❶, вы тем самым предписываете FreeBSD автоматически создавать во время загрузки файловую систему */tmp* в памяти. Размер файловой системы */tmp* определяет параметр `tmpsize` ❷. Назначение флагов ❸ мы будем рассматривать немного ниже, в этом же разделе.

Если на этом ваш интерес к файловым системам в памяти исчерпывается, можно считать, что вы достигли желаемого. Если же вам интересно узнать, как с помощью mdmfs(8) создаются и используются специальные устройства памяти, читайте дальше.

Типы дисков памяти

Диски памяти могут быть трех типов: невытесняемые (malloc-backed), вытесняемые (swap-backed) и файловые (vnode-backed).

Невытесняемые диски всегда находятся в памяти. Невытесняемые диски никогда не перемещаются в своп, даже если система испытывает нехватку памяти. Создание невытесняемых дисков памяти большого объема – отличный способ исчерпать доступную память и вызвать крах системы. Диски такого типа наиболее полезны для встраиваемых устройств, не имеющих файла подкачки, с которыми мы познакомимся в главе 20.

Вытесняемые диски большую часть времени находятся в памяти, но они могут вытесняться в системный раздел подкачки (swap). Если система испытывает нехватку памяти, она вытесняет давно не использовавшиеся участки памяти в раздел подкачки, как описывается в главе 19. Вытесняемые диски – это наиболее безопасный способ использования файловых систем в памяти.

Файловые диски – это обычные дисковые файлы. Вы можете использовать обычные файлы в качестве носителей дисков памяти, однако по-настоящему эта возможность полезна только разработчикам файловых систем, желающим выполнить серию тестов или смонтировать образ диска.

Тип диска памяти определяется областью его применения. Если в системе отсутствует пространство свопинга, а файловая система доступна только для чтения, единственный доступный тип – это невытесняемые диски. Если речь идет о сервере или рабочей станции, где было бы желательно создать быструю файловую систему для временного хранения

данных, то предпочтительнее использовать вытесняемые диски. Файловые диски необходимы только для монтирования образов дисков.

Как только вы определитесь с типом диска памяти, можно приступать к выполнению необходимых операций с помощью mdmfs(8).

Создание и монтирование дисков памяти

Утилита mdmfs(8) представляет собой удобный интерфейс к таким программам, как mdconfig(8) и newfs(8). Она берет на себя тяжелый труд по настройке устройств и созданию файловых систем на этих устройствах. Она упрощает операцию создания дисков памяти настолько, насколько это вообще возможно. Чтобы создать диск памяти, достаточно знать лишь размер диска, его тип и точку монтирования.

По умолчанию создаются вытесняемые диски памяти. Для этого достаточно передать утилите mdmfs(8) размер диска и точку монтирования. Следующая команда создает вытесняемый диск памяти размером 8 Мбайт, смонтированный в каталог */home/mwlucas/test*:

```
# mdmfs -s 8m md /home/mwlucas/test
```

Ключ `-s` задает размер диска. Если теперь запустить mount(8) без аргументов, можно будет увидеть устройство диска памяти */dev/md0*, смонтированное в указанный выше каталог.

Чтобы создать и смонтировать невытесняемый диск памяти, следует добавить ключ `-M`.

Чтобы смонтировать файловый диск памяти, следует добавить ключ `-F` и указать путь к файлу образа.

```
# mdmfs -F diskimage.file md /mnt
```

В предыдущих командах в качестве имени устройства использовалось имя `md`, что означает: «Меня не интересует, как будет названо устройство, я лишь хочу получить первое незанятое имя». Однако в случае необходимости можно явно определить желаемое имя устройства. Следующая команда создаст устройство */dev/md9*, которое будет играть роль диска памяти:

```
# mdmfs -F diskimage.file md9 /mnt
```

Недостатки дисков памяти

Диски памяти – это звучит слишком хорошо, чтобы быть правдой, особенно для высокопроизводительных систем. Они имеют свои ограничения, о которых вам следует знать, прежде чем вы приступите к повсеместному их внедрению.

Самая большая проблема заключается в том, что при выключении системы содержимое таких дисков исчезает безвозвратно. На первый взгляд эта проблема может показаться несущественной, однако мне в своей практике не раз приходилось удивляться пропаже файлов из файловых систем, если я забывал, что это были диски памяти.

Кроме того, невытесняемые диски *никогда* не отдают память. Когда вы записываете файл на диск памяти, а затем стираете его, этот файл все равно продолжает занимать память. При использовании вытесняемого диска памяти этот файл, в конечном счете, будет вытеснен в раздел подкачки в случае нехватки памяти. Диск памяти в системах, работающих продолжительное время, рано или поздно исчерпает всю оперативную память системы.

Единственный способ освободить использованную память – это демонтировать диск памяти, уничтожить устройство и создать новый диск памяти. По своему опыту могу сказать, что даже на очень нагруженном сервере вполне достаточно выполнять эту операцию раз в месяц.

За пару месяцев до выхода в свет FreeBSD 7.0 в нее была добавлена файловая система tmpfs(5), размещаемая в памяти, которая, как предполагается, освобождает неиспользуемую память и возвращает ее системе. Однако эта файловая система еще достаточно новая и рассматривается как экспериментальная. Если вы читаете эти строки, когда уже вышли в свет более новые версии FreeBSD, я порекомендовал бы обратить внимание на tmpfs(5), но если вы установили версию 7.0 или 7.1, то на вашем месте я бы предпочел, чтобы отловом ошибок занимался кто-то другой.

Удаление дисков памяти

Чтобы удалить диск памяти, нужно демонтировать раздел и уничтожить устройство диска. В момент уничтожения устройства диска происходит освобождение памяти, использованной этим устройством, что бывает полезно при существенном увеличении нагрузки на систему. Чтобы узнать имя файла устройства, запустите команду mount(8) и поищите в списке раздел диска памяти. Он должен выглядеть примерно так:

```
/dev/md41 on /mnt (ufs, local, soft-updates)
```

Здесь видно диск памяти */dev/md41*, монтированный в каталог */mnt*. Давайте демонтируем его и уничтожим.

```
# ❶umount /mnt
# mdconfig ❷-d ❸-u 41
```

Демонтирование раздела диска памяти выполняется точно так же, как и любой другой файловой системы, – с помощью команды umount ❶. Однако команда mdconfig(8) вам еще не знакома. Утилита mdconfig(8) используется для непосредственного управления устройствами памяти. Ключ `-d` ❷ означает *уничтожить* (*destroy*), а ключ `-u` ❸ определяет номер устройства. Команда, приведенная выше, уничтожит устройство */dev/md41* или устройство md с номером 41. Теперь память, которая использовалась устройством, свободна для других нужд.

Диски памяти и /etc/fstab

Если добавить информацию о дисках памяти в файл */etc/fstab*, FreeBSD будет автоматически создавать их на этапе загрузки. Записи, соответствующие дискам памяти, выглядят сложнее, но все не так страшно, если вы поняли, как работать с командами mdmfs(8), которые мы использовали до сих пор. Чтобы освежить вашу память, приведу единственную строку из файла */etc/fstab*, которая соответствует стандартной файловой системе.

```
# Device        Mountpoint      FStype      Options         Dump    Pass#
/dev/ad4s2a     /               ufs         rw              1       1
```

Чтобы как-то обозначить, что устройство является диском памяти (memory disk), мы дадим ему имя *md*. В качестве точки монтирования можно выбрать произвольный каталог, как и для любого другого устройства, а в качестве типа файловой системы укажем `mfs`. В поле `Options` укажем параметр `rw` (read-write – чтение/запись) и параметры командной строки, которые потребуются для создания данного устройства. Для создания нашей файловой системы объемом 8 Мбайт, монтированной в каталог */home/mwlucas/test*, используется следующая запись в файле */etc/fstab*:

```
md              /home/mwlucas/test  mfs         rw,-s 8m        0       0
```

Выглядит достаточно просто, не так ли? Единственная незадача – длинная строка нарушает аккуратный вид файла */etc/fstab*. И этот пример не единственный, делающий файл трудночитаемым, в чем вы вскоре убедитесь.

Монтирование образов дисков

Диски памяти могут также использоваться для монтирования и доступа к образам дисков. Эта очень удобная возможность позволяет ознакомиться с содержимым образов компакт-дисков, прежде чем записывать их на болванки. Для этого достаточно подключить диск памяти к файлу с помощью команды mdconfig(8), с ключом `-a`. Ниже приводится пример подключения ISO-образа с дистрибутивом FreeBSD к устройству памяти:

```
# mdconfig ❶-a -t ❷vnode -f ❸/home/mwlucas/7.0-CURRENT-snap.iso
❹md0
```

Эта команда предписывает команде mdconfig(8) подключить ❶ файловый (vnode-backed) ❷ диск памяти к указанному файлу ❸. После этого mdconfig(8) сообщает, что устройство ❹ подключено. Теперь можно просто смонтировать устройство соответствующей командой монтирования, не забыв указать тип файловой системы:

```
# mount -t cd9660 /dev/md0 /mnt
```

Одна из распространенных ошибок, которую допускают здесь, – это попытка смонтировать образ без указания файловой системы. В ре-

зультате вы можете получить сообщение об ошибке, а можете благополучно смонтировать образ, в котором не окажется данных, потому что по умолчанию утилита mount(8) предполагает, что имеет дело с файловой системой FFS.

Закончив просмотр содержимого образа, его обязательно нужно демонтировать, а устройство уничтожить, так же как и любое другое устройство памяти. Хотя диски памяти файлового типа и не потребляют системную память, бесконтрольное создание устройств памяти может привести к тому, что через несколько месяцев у вас появятся вопросы – откуда они взялись в каталоге */dev*. Узнать о наличии устройств памяти в системе можно с помощью команды `mdconfig -l`, которая выведет список всех устройств md(5).

```
# mdconfig -l
md0 md1
```

Как? У меня два устройства памяти? Чтобы узнать тип устройства памяти, следует добавить ключ `-u` и номер устройства. Давайте посмотрим, что это за устройство с номером 1 (*/dev/md1*):

```
# mdconfig -l -u 1
md1     vnode     456M  /slice1/usr/home/mwlucas/iso/omsa-51-live.iso
```

Оказывается, у меня смонтирован еще один ISO-образ? Ух ты! Возможно, через несколько месяцев мне придется выполнить перезагрузку. Впрочем, это слишком трудоемко для меня, поэтому я просто демонтирую файловую систему.

Файловые системы в файлах

Одна из интересных возможностей, используемых для создания клетки (глава 9) и размещения самодельных встроенных систем (глава 20), заключается в создании полных образов файловой системы в локальной файловой системе. В предыдущем разделе мы видели, как можно использовать диски памяти для монтирования и доступа к образам компакт-дисков. Вы можете использовать тот же самый прием для создания и доступа к образам дисков с файловой системой FFS. Недостаток такого способа состоит в том, что объем такого диска равен размеру файла образа. Если создать образ диска размером 500 Мбайт, он займет 500 Мбайт на диске.

Для работы с файловой системой в файле необходимо создать файл требуемого размера, подключить файл к устройству памяти, разместить файловую систему на устройстве и смонтировать его.

Создание пустого файла

Файл, внутри которого предполагается создать файловую систему, изначально не должен содержать никаких данных – это всего лишь файл нужного размера. Можно было бы усесться поудобнее и вручную набить огромное число нулей, чтобы создать такой файл, однако Free-

BSD имеет неограниченный источник «ничего», который избавит вас от такой неприятной работы. Устройство */dev/zero* до краев наполнено пустотой, которую можно использовать для заполнения файла.

Здесь вам поможет та же самая команда dd(1), которую вы использовали для копирования установочных дискет в файл с файловой системой. Следующая команда копирует данные из устройства */dev/zero* в файл *filesystem.file*.

```
# dd ❶if=/dev/zero ❷of=filesystem.file ❸bs=1m ❹count=1k
1048576+0 records in
1048576+0 records out
1073741824 bytes transferred in ❺24.013405 secs (❻44714268 bytes/sec)
```

Эта команда берет данные их входного файла */dev/zero* ❶ и выводит их в выходной файл *filesystem.file* ❷. Передача данных производится блоками размером 1 Кбайт ❹, и так один миллион раз ❸. Это занимает всего несколько секунд ❺, а поскольку устройству */dev/zero* не приходится всякий раз тратить время на создание очередного символа, файл наполняется с очень высокой скоростью ❻. Если теперь заглянуть в текущий каталог, там можно будет увидеть файл *filesystem.file*, размером 1 Гбайт.

Очень часто в заблуждение вводят такие параметры команды dd(1), как размер блока (block size) и счетчик (count). Вычисление конечного размера файла, создаваемого командой dd(1), напоминает перемещение груды песка – вся работа делится на три этапа: погрузка, перемещение и выгрузка определенной порции. Вы можете сделать несколько рейсов с тачкой, в несколько раз больше – с ведром или много-много рейсов с ложкой. Объем, переносимый за раз, – это размер блока, а число рейсов – счетчик. Вы можете носить песок в пригоршнях, и тогда размер блока будет маленький, а число рейсов – большим. Возможно, у вас имеется тачка, куда поместится блок среднего размера, и тогда число рейсов будет не так велико. А возможно, у вас есть экскаватор, который способен переместить весь песок за один рейс. Чем больше размер блока, тем выше нагрузка на систему при его перемещении. Команда dd(1) способна воспринимать различные сокращения, используемые для обозначения одного блока и количества перемещений, как показано в табл. 8.2.

Допустим, что вам требуется получить файл размером 1 Гбайт. Не забывайте, что `1k` обозначает один килобайт. Один мегабайт – это тысяча килобайт, а один гигабайт – одна тысяча мегабайт. Если использовать блоки размером 1 байт, а количество перемещений задать равным одному `1g`, вашей системе придется совершить 1 073 741 824 рейсов, чтобы переместить песок. Каждый рейс будет очень легким, но их будет слишком много! С другой стороны, если выбрать размер блока равным `1g`, а счетчик равным 1, тогда системе придется переместить всю кучу за один рейс. Я думаю, ее это не обрадует. Впрочем, как и вас. Вообще говоря, при размере блока `1m` и небольшом числе рейсов конечная цель будет достигнута достаточно быстро, причем без особого перенапряжения

системы. Если использовать размер блока, равный 1m, то счетчик будет обозначать размер файла в мегабайтах. При счетчике, равном 1k, будет создан файл, размером 1 Гбайт, при счетчике, равном 2k, – 2 Гбайт, при счетчике, равном 32, – 32 Мбайт и т. д.

Таблица 8.2. Сокращения, которые распознает команда dd

Символ	Значение	Множитель
b	Дисковый блок	512
k	Кило	1024
m	Мега	1048576
g	Гига	1073741824
w	Целое	Размер целого числа в байтах на вашей платформе

Создание файловой системы в файле

Чтобы создать файловую систему в файле, для начала необходимо связать файл с устройством диска памяти файлового типа. Делается это точно так же, как и в предыдущем разделе:

```
# mdconfig -a -t vnode -f filesystem.file
md0
```

Теперь можно создавать файловую систему на этом устройстве. Сама процедура очень напоминает создание файловой системы FFS на дискете с помощью команды newfs(8). Механизм Soft Updates весьма благоприятно сказывается при работе с файловыми системами, расположенными внутри файлов, поэтому желательно указать ключ -U, который активирует этот механизм.

```
# newfs -U /dev/md0
/dev/md0: 1024.0MB (2097152 sectors) block size 16384, fragment size 2048
        using 6 cylinder groups of 183.77MB, 11761 blks, 23552 inodes.
        with soft updates
super-block backups (for fsck -b #) at:
 160, 376512, 752864, 1129216, 1505568, 1881920
```

Утилита newfs(8) вывела такие основные сведения о диске, как размеры блоков и фрагментов, а также число индексных дескрипторов.

Теперь, когда файловая система создана, ее можно монтировать:

```
# mount /dev/md0 /mnt
```

Поздравляю! Теперь в вашем распоряжении имеется файловая система, расположенная внутри файла, размером 1 Гбайт. В нее можно копировать файлы, можно создать резервную копию этой файловой системы на магнитной ленте и делать все то, что можно делать с любой другой файловой системой. Но, кроме того, ее можно перемещать из каталога в каталог, подобно любому другому файлу. Мы будем использовать такую возможность, когда в следующей главе будем рассматривать

клетки, которые играют важную роль при создании собственных встроенных систем.

Файловые системы внутри файлов и /etc/fstab

Файловые системы, расположенные внутри файлов, можно монтировать автоматически, во время загрузки системы, если добавить соответствующую строку в */etc/fstab*, точно так же, как в случае с другими дисками памяти. Для этого нужно просто указать имя файла с ключом `-F` и добавить ключ `-P`, чтобы предотвратить создание новой файловой системы и использовать уже существующую. Следующая строка предполагает автоматическое монтирование созданной выше файловой системы, находящейся внутри файла, в каталог */mnt* во время загрузки. (Я предупреждал вас, что нам доведется увидеть примеры строк в */etc/fstab*, которые делают его еще более трудночитаемым, не так ли?)

```
md      /mnt      mfs     rw,-P,-F/home/mwlucas/filesystem.file   0       0
```

Прочие файловые системы

FreeBSD поддерживает несколько других, малоизвестных файловых систем. Большинство из них используется в достаточно необычных ситуациях, но в системном администрировании такие необычные ситуации случаются ежедневно.

devfs(5) – это файловая система устройств, которая располагается в каталоге */dev*. В этой файловой системе нельзя хранить обычные файлы, она поддерживает только файлы устройств. Содержимым этой файловой системы напрямую управляют ядро и демон файловой системы устройств – devd(8).

procfs(5) – это файловая система процессов, она содержит огромный объем информации о процессах. Эта файловая система представляет угрозу безопасности и потому не рекомендуется к использованию в современных версиях FreeBSD. Однако при наличии монтированной файловой системы процессов можно узнать много интересного о процессах. Некоторые старые приложения требуют, чтобы эта файловая система была смонтирована в каталог */proc* – если приложение сервера требует наличия procfs, попробуйте найти аналогичное приложение, которое не предъявляет таких требований.

При использовании режима совместимости с Linux (глава 12) может потребоваться монтировать файловую систему linprocfs(5). Значительная часть программного обеспечения для Linux требует наличия файловой системы процессов и при выборе режима совместимости с Linux операционная система FreeBSD предлагает установить linprocfs в каталог */compat/linux/proc*. Я рекомендую устанавливать linprocfs, только если имеется программное обеспечение, требующее ее присутствие.

fdescfs(5), файловая система дескрипторов, предоставляет возможность просматривать дескрипторы файлов для каждого процесса. Су-

ществуют программы, требующие наличия fdescfs(5). Она представляет меньшую угрозу безопасности, чем procfs, тем не менее использовать ее нежелательно.

Привязка устройств

Диски с интерфейсом SCSI не всегда включаются в одном и том же порядке, но нумерация устройств SCSI в системе FreeBSD соответствует порядку, в каком они подключены к шине SCSI. Кроме того, если изменить порядок подключения устройств к шине SCSI, это приведет к изменению порядка, в каком они будут опрашиваться. Диск, который имел порядковый номер 0 при установке FreeBSD, после добавления нового диска может получить номер 1. Такое изменение может привести к тому, что разделы будут монтироваться в неверные точки монтирования, и может даже вызвать повреждение данных. Похожая проблема может возникнуть при добавлении нового контроллера SCSI, так как в этом случае произойдет переупорядочение шин SCSI! Например, устройство, имевшее имя */dev/da0* при установке FreeBSD, может превратиться в устройство */dev/da1* или даже */dev/da17* после подключения нового накопителя на магнитных лентах. Это приведет к тому, что разделы будут монтированы в неверные точки монтирования.

Во избежание этих трудностей порядок нумерации дисков можно жестко закодировать в ядре. Процесс, избавляющий от подобной путаницы, называется *привязкой* (*wiring down*) устройств SCSI. Чтобы выполнить привязку устройств, надо знать SCSI ID, номер шины SCSI и LUN (если применяется) каждого устройства в цепочке SCSI, доступных в */var/run/dmesg.boot*. Например, в моей тестовой системе есть следующие записи dmesg для адаптера SCSI:

```
ahc0: <Adaptec 2940B Ultra2 SCSI adapter> port 0xe000-0xe0ff mem 0xe8042000-
0xe8042fff irq 11 at device 20.0 on pci0
aic7890/91: Ultra2 Wide Channel A, SCSI Id=7, 32/253 SCBs
```

В первой строке сообщается, что основная карта SCSI – это адаптер «Adaptec 2940B Ultra2». Во второй содержится дополнительная информация об адаптере на этой карте. На самом деле здесь есть только одна физическая карта. Хост-адаптер использует SCSI ID 7. LUN не применяется.

Далее, в файле *dmesg.boot* содержатся записи для всех дисков SCSI. Эти записи включают в себя такие сведения, как емкость диска, название модели, скорость и дополнительные особенности, но я приведу здесь только первые строки записей для каждого диска:

```
❶da0 at ❷ahc0 ❸bus 0 ❹target 8 ❺lun 0
da1 at ahc0 bus 0 ❻target 9 lun 0
da2 at ahc0 bus 0 target 10 lun 0
...
```

Эти строки сообщают, что диск da0 ❶ подсоединен к карте SCSI ahc0 ❷, на шине 0 ❸, SCSI ID 8 ❹, LUN равен 0 ❺. Диск da1 расположен на той же самой карте SCSI и той же шине, однако SCSI ID равен 9 ❻.

Чтобы выполнить привязку дисков, сообщите ядру, какой номер шины SCSI к какой карте подключается, SCSI ID и LUN для каждого диска. Эту информацию следует поместить в файл */boot/device.hints*:

```
hint.❶scbus._0.at="❸ahc0"
hint.❹da.0.at="❺scbus0"
hint.da.0.target="❻8"
hint.da.1.at="scbus0"
hint.da.1.target="❼9"
```

Здесь ядру FreeBSD сообщается, что шина SCSI ❶ с номером 0 ❷ должна подключаться к карте ahc0 ❸. Диск da0 ❹ подключен к шине SCSI с номером 0 ❺ и имеет SCSI ID 8 ❻, а диск da1 подключается к той же шине и имеет SCSI ID 9 ❼. Во время следующей перезагрузки устройства будут нумероваться так, как указано здесь. Если добавить еще одну карту SCSI или дополнительные жесткие диски SCSI, FreeBSD присвоит номера новым шинам и устройствам с учетом номеров, зарезервированных для этих устройств. При наличии нескольких устройств с разными номерами LUN в файле */boot/device.hints* можно использовать параметр `lun`.

Добавление новых жестких дисков

Прежде чем можно будет использовать новый жесткий диск, его надо отформатировать, разместить на нем файловые системы, смонтировать эти файловые системы и перенести на них данные. Хотя во FreeBSD имеются инструменты командной строки, позволяющие выполнить все эти операции, тем не менее действия можно упростить с помощью sysinstall(8). Если вы спешите, то наверняка выберете sysinstall. Предположим, что вы добавляете диск в существующую систему, а ваша конечная цель состоит в том, чтобы перенести на этот диск данные.

Резервное копирование, резервное копирование и еще раз резервное копирование!

Прежде чем выполнять какие-либо операции с жесткими дисками, следует создать полную резервную копию. Одно неверное движение при выполнении таких операций может разрушить вашу систему! Несмотря на то, что корневая файловая система форматируется очень редко, тем не менее, если такое случается, вам наверняка захочется восстановить ее как можно быстрее.

Создание участков

В первую очередь при работе с новым жестким диском следует создать на нем участки и разделы. Выполните следующие шаги:

1. Войдите в систему как root и запустите sysinstall(8). Мы будем конфигурировать систему уже после установки, поэтому выбираем Configure и затем Disk.
2. Меню покажется знакомым; вы применяли его при установке FreeBSD. (См. снимки экранов на рис. 2.4 главы 2.) Вы увидите существующий диск FreeBSD и новый диск. Выберите новый диск.
3. Если диск взят с другого сервера, на нем может оказаться файловая система. Можно сохранить содержимое диска, а можно очистить его и начать с нуля. Обычно проще всего удалить существующие разделы и файловые системы. С помощью клавиш-стрелок перейдите к существующему разделу и нажмите клавишу D для его удаления.
4. Далее создайте новый участок, нажав клавишу C, или задействуйте весь диск (клавиша A). На сервере лучше задействовать весь диск. Выбрав конфигурацию участков, сохраните внесенные изменения нажатием клавиши W. Появится следующее предостережение:

```
WARNING: This should only be used when modifying an EXISTING installation.
If you are installing FreeBSD for the first time then you should simply
type Q when you're finished here and your changes will be committed in one
batch automatically at the end of these questions. If you're adding a
disk, you should NOT write from this screen, you should do it from the
label editor.
(ПРЕДОСТЕРЕЖЕНИЕ: Такой вариант следует применять только при изменении
СУЩЕСТВУЮЩЕЙ системы. Если вы устанавливаете FreeBSD с нуля, то после
внесения изменений нажмите <Q>. В этом случае все изменения будут
одновременно приняты после получения ответов на данные вопросы. Если вы
добавляете диск, вам НЕ следует осуществлять запись из этого экрана,
делать это следует в редакторе меток.)
```

5. Абсолютно ли вы уверены в своем выборе?
6. Да, вы абсолютно уверены. Перейдите к пункту Yes и нажмите Enter.
7. Далее будет задан вопрос о необходимости установки на этот диск менеджера загрузки. На дополнительном диске он не нужен, поэтому спускаемся к пункту Standard и нажимаем клавишу пробела. Программа sysinstall должна сообщить, что информация fdisk записана. Теперь на диске есть участок FreeBSD. Нажмите клавишу Q, чтобы покинуть этот экран systinstall.

Создание разделов

Для создания разделов на диске выполните следующие шаги:

1. Выберите пункт Label в sysinstall(8), в том же подменю, где находится FDISK. Выберите новый диск, чтобы перейти в редактор меток. С помощью команды C здесь можно создать новый раздел, указав

его размер в мегабайтах, гигабайтах, дисковых блоках или дисковых цилиндрах. Кроме того, необходимо определиться с предназначением каждого нового раздела – будет ли это файловая система или пространство свопа. Когда программа потребует указать точку монтирования, введите каталог */mnt*. Программа sysinstall временно будет монтировать новые разделы в этот каталог.

2. По окончании создания разделов нажмите `W` для подтверждения изменений. Должен появиться текст предупреждения, который вы уже видели в меню fdisk, а затем сообщение от newfs(8).

По окончании работы с newfs(8) завершите sysinstall.

Конфигурирование /etc/fstab

Теперь следует добавить записи с информацией о новых дисках в */etc/fstab*. Порядок конфигурирования новых разделов свопа и файловых систем выполняется по-разному. Любое пространство свопа в */etc/fstab* описывается следующим образом:

```
devicename     none            swap    sw            0       0
```

Если представить, что новому разделу свопа соответствует устройство */dev/da10s1b*, тогда в */etc/fstab* следует добавить следующую строку:

```
/dev/da10s1b   none            swap    sw            0       0
```

Во время следующей загрузки система FreeBSD найдет эту запись и активизирует новое пространство свопа. Чтобы активизировать новое пространство свопа без перезагрузки, можно ввести команду `swapon -a` *`devicename`*.

Если был создан новый раздел данных, добавьте новую запись, как описывалось выше в этой главе, например, такую:

```
/dev/da10s1d   /usr/obj        ufs     rw             2       2
```

Теперь можно просто отмонтировать новый раздел от точки временного монтирования, запустить команду `mount /usr/obj`, и ваш новый диск будет готов к приему файлов.

Перенос существующих файлов на новые диски

Вероятно, новый диск призван заменить существующий раздел FreeBSD либо взять на себя часть данных существующего раздела. Для этого надо предоставить системе доступ к новому разделу во временной точке монтирования. Переместите файлы из старого раздела в новый. Затем заново смонтируйте раздел в требуемом месте.

В примере выше новый раздел был смонтирован в каталог */mnt*. Теперь надо перенести файлы из их текущего местоположения в новый раздел, не изменяя права доступа к файлам. Такая задача решается довольно просто с помощью tar(1). Вы можете просто сохранить существующие данные на ленте или в файле с помощью tar, а затем разар-

хивировать их в новом месте. Однако последовательно выполнять эти операции неудобно. Объединив команды tar, можно обойтись без промежуточного этапа:

```
# tar cfC - /old/directory . | tar xpfC - /tempmount
```

Эта строка может озадачить тех, кто не говорит на языке UNIX на вечеринках. Разберем ее по частям. Во-первых, вы переходите в старый каталог и архивируете все его содержимое. Затем вывод этой команды переносится в новый каталог, где данные разархивируются. Например, для перемещения */usr/src* на новый раздел, временно монтированный в */mnt*, подойдет такая последовательность команд:

```
# tar cfC - /usr/src . | tar xpfC - /mnt
```

Проверьте содержимое точки временного монтирования и убедитесь, что все файлы действительно были скопированы. Как только вы убедитесь, что файлы были благополучно скопированы, удалите файлы из старого каталога и смонтируйте новый диск в новое местоположение. Например, после копирования файлов из */usr/src* можно было бы выполнить следующую последовательность команд:

```
# rm -rf /usr/src/*
# umount /mnt
# mount /usr/src
```

Перенос активных файлов

Вам не удастся благополучно переместить файлы, если они в этот момент подвергаются изменению. Например, при переносе почтового спула на новый раздел выключите почтовые сервисы. В противном случае файлы изменятся, когда вы будете их копировать. Избежать необходимости останавливать сервисы не удастся, но с помощью таких инструментов, как rsync (*/usr/ports/net/rsync*), время простоя можно уменьшить.

Составное монтирование

Предположим, что вам нет дела до старых данных, и вы просто хотите освободить место на диске. Вы планируете восстановить свои данные из резервной копии. Разумно. Все файловые системы BSD – *составные* (*stackable*). Они обладают расширенными возможностями, которые не особо полезны в повседневном администрировании. Однако эти возможности придут на помощь, когда потребуется разделить один раздел на два.

Предположим, что ваши данные хранятся в */usr/src*. Посмотрите объем используемого дискового пространства и затем смонтируйте новый раздел в */usr/src*. Если вы зайдете в каталог /usr/src, то увидите, что он пуст.

Есть одна загвоздка: новый раздел монтирован «поверх» старого диска, а все данные по-прежнему находятся на старом диске. На старом разделе не появится свободное место, пока вы не переместите данные. Если размонтировать новый раздел и снова проверить каталог, вы увидите, что данные чудесным образом восстановлены! Новый раздел скрывал старый.

Вы не можете это увидеть, но данные на старом диске по-прежнему занимают определенное пространство. Если вы перераспределяете диск для получения дополнительного пространства, а новый диск монтируете поверх старого, дисковое пространство на исходном диске будет недоступно. Мораль: даже при восстановлении данных с резервной копии надо убедиться, что данные перемещены с исходного диска и дисковое пространство освобождено.

Сетевые файловые системы

Сетевая файловая система обеспечивает возможность доступа по сети к файлам, расположенным на другом компьютере. В качестве сетевых файловых систем наиболее часто используются Network File System (NFS), изначально реализованная в UNIX, и CIFS (известна также как SMB), которая используется в Microsoft Windows. Мы рассмотрим обе файловые системы, но начнем с NFS, старого стандарта для UNIX.

Организовать совместный доступ к каталогам и разделам для UNIX-подобных систем проще всего с помощью сетевой файловой системы NFS (Network File System). Система FreeBSD изначально обладает поддержкой NFS. Настройка NFS часто пугает начинающих администраторов, но, выполнив эту процедуру пару раз, вы увидите, что в ней нет ничего сложного.

Все соединения NFS следуют модели клиент-сервер. Один компьютер играет роль сервера – он предоставляет доступ к своим файловым системам для других компьютеров. Это называется *экспортирование NFS* (*NFS exporting*), а предоставляемые файловые системы называ-

Функциональная совместимость NFS

Все реализации NFS имеют некоторые различия. Можно заметить незначительные различия NFS в Solaris, Linux, BSD и других UNIX-подобных системах. NFS должна работать между всеми этими системами, но иногда может потребоваться выполнить некоторые настройки. Если вы испытываете сложности при работе с другими UNIX-подобными операционными системами, проверьте архив почтовой рассылки *FreeBSD-net* – ваша проблема наверняка уже обсуждалась здесь.

ются *предметами экспорта* (*exports*). Клиенты могут монтировать предметы экспорта почти так же, как локальные файловые системы.

Интересно заметить, что NFS не имеет информации о своем состоянии, она не следит за состоянием соединения. Сервер NFS можно перезагрузить, и клиент при этом не пострадает. Конечно, пока сервер не заработает, он не сможет обращаться к файлам, находящимся в предметах экспорта, но как только сервер запустится, клиент окажется там же, где и остановился. Другие сетевые файловые системы не всегда так дружелюбны. Безусловно, отсутствие информации о состоянии вызывает некоторые проблемы, например, клиент не узнает, что файл, открытый им, был в это же время изменен другим клиентом.

Как для NFS-серверов, так и для NFS-клиентов необходимо выполнить настройку ядра, но различные команды NFS динамически загружают необходимые модули ядра. Ядро FreeBSD GENERIC имеет поддержку NFS, поэтому можно не беспокоиться по поводу настройки ядра.

NFS – это одна из тем, которым посвящены целые книги. Мы не будем рассматривать NFS во всех подробностях, а сфокусируем свое внимание на выполнении основных операций. Если необходимо развернуть NFS сложной конфигурации, то вам придется произвести дополнительные исследования этой темы. Но даже такая простая конфигурация, которая обсуждается ниже, поможет вам в решении многих сложных задач.

Запуск сервера NFS

Чтобы включить поддержку сервера NFS, необходимо добавить в файл *rc.conf* следующие параметры настройки. Не все из этих параметров являются обязательными для всех возможных случаев, тем не менее все вместе они обеспечивают высокий уровень функциональной совместимости NFS и достаточно приличную производительность.

```
❶ nfs_server_enable="YES"
❷ rpcbind_enable="YES"
❸ mountd_enable="YES"
❹ rpc_lockd_enable="YES"
❺ rpc_statd_enable="YES"
```

Первый параметр сообщает FreeBSD о необходимости загрузить модуль *nfsserver.ko* ❶, если он еще не загружен. rpcbind(8) ❷ отображает удаленные вызовы процедур (remote procedure calls, RPC) в локальные сетевые адреса. Все клиенты NFS спрашивают у демона rpcbind(8), работающего на стороне сервера, где они могут отыскать демон mountd(8), чтобы соединиться с ним. mountd(8) ❸ ожидает получения запросов клиентов на соединение на одном из портов верхнего диапазона. При активации сервера NFS также запускается демон nfsd(8), который обслуживает фактический доступ к файлам. rpc.lockd(8) ❹ обеспечивает блокировку файлов в NFS, а rpc.statd(8) ❺ следит за клиентами NFS, чтобы сервер NFS мог освобождать ресурсы после отключения клиентов.

Эти службы можно запускать из командной строки, если вы знакомы с NFS, но после выполнения необходимых настроек лучше выполнить перезагрузку системы. После этого вы сможете увидеть rpc.lockd, rpc.statd, nfsd, mountd и rpcbind в выводе команды sockstat(1). Если какой-то из этих демонов отсутствует в списке, просмотрите содержимое файла */var/log/messages* на наличие сообщений об ошибках.

Настройка экспорта NFS

Теперь необходимо сообщить серверу, какие каталоги являются предметом экспорта. Можно было бы просто экспортировать все каталоги и файловые системы сервера, но любой администратор, компетентный в вопросах безопасности, предпочтет внести свои ограничения. Как и в любом другом случае, доступным следует делать лишь то, что действительно необходимо. Например, в большинстве случаев клиентам не требуется монтировать корневую файловую систему сервера NFS.

Перечень клиентов и доступных им файловых систем и каталогов находится в файле */etc/exports*. Для каждого диска на стороне сервера в этом файле отводится отдельная строка, в которой также перечисляются клиенты, обладающие правом доступа к этому устройству. Строки могут содержать до трех частей:

- Экспортируемые каталоги или разделы (обязательное поле)
- Параметры экспорта
- Клиенты, которые могут монтировать этот каталог

Каждой комбинации клиентов и дискового устройства в этом файле может соответствовать всего одна строка. То есть, если на одном и том же разделе находятся каталоги */usr/ports* и */usr/home*, и оба эти каталога необходимо сделать доступными для одного и того же клиента, они должны быть перечислены в одной и той же строке. Невозможно экспортировать */usr/ports* и */usr/home* одному и тому же клиенту с разными правами доступа. Не обязательно экспортировать весь диск, можно экспортировать единственный каталог. Этот каталог не может содержать символических ссылок.

Из трех частей записи в */etc/exports* обязательной является только первая – имя каталога. Например, если бы мне захотелось экспортировать свой домашний каталог всем желающим в Интернете, я мог бы добавить в файл */etc/exports* следующую строку:

```
/home/mwlucas
```

Здесь отсутствуют какие-либо параметры и ограничения. Это, конечно, глупо, но вполне возможно.[1]

[1] Почему же не предусмотрена защита от таких вот «самострелов»? Надо сказать, операционная система UNIX не считает тех, кто способен на такой шаг, достойными своей дружбы. За такое негуманное поведение многие не устают проклинать UNIX, но, по-моему, так даже интереснее.

После редактирования файла /etc/exports необходимо сообщить демону mountd о том, чтобы он перечитал его.

```
# /etc/rc.d/mountd restart
```

Любые ошибки заносятся демоном mountd(8) в файл протокола */var/log/messages*. Сообщения в файле протокола носят довольно загадочный характер: mountd(8) обычно сообщает, что строка содержит ошибку, но, как правило, не говорит – какую. Наиболее типичные ошибки, из тех, с которыми я сталкивался, связаны с наличием символических ссылок.

Активация клиента NFS

Настройка клиента выглядит намного проще, для этого достаточно поместить в */etc/rc.conf* следующую строку:

```
nfsclient="YES"
```

После этого нужно перезагрузить систему или запустить команду /etc/rc.d/nfsclient start. В обоих случаях будет активирована поддержка клиента NFS.

Теперь можно монтировать каталоги и файловые системы, экспортируемые серверами NFS, только вместо имени устройства следует указывать сетевое имя сервера NFS и монтируемый каталог. Например, чтобы смонтировать каталог */home/mwlucas*, экспортируемый сервером *sardines*, в каталог */mnt*, можно запустить такую команду:

```
# mount sardines:/home/mwlucas /mnt
```

После этого с помощью df(1) можно убедиться, что каталог смонтирован.

```
# df
Filesystem             1K-blocks      Used    Avail Capacity   Mounted on
/dev/ad4s1a              1012974    387450   544488      42%   /
devfs                          1         1        0     100%   /dev
/dev/ad4s1f            109009798  12959014 87330002      13%   /usr
/dev/ad4s1e              1012974     42072   889866       5%   /var
sardines:/home/mwlucas 235492762 150537138 66116204      69%   /mnt
```

Смонтированный каталог NFS выглядит как обычный раздел и позволяет читать и записывать файлы, какие понравятся. Ну, может быть, *не все*, какие понравятся…

NFS и пользователи

Принадлежность файла и права доступа к нему определяются по числовым идентификаторам UID. Для идентификации владельца NFS тоже использует UID. Например, в моем ноутбуке пользователь mwlucas имеет UID, равный 1001. На сервере NFS пользователь mwlucas также имеет UID, равный 1001. Это упрощает мне жизнь, потому что я не должен беспокоиться по поводу принадлежности файлов – у меня

одинаковые привилегии и на сервере, и на ноутбуке. В больших сетях, где пользователи обладают привилегиями root на своих машинах, это может вызывать проблемы. Лучший способ избежать их – создать центральный репозитарий авторизованных пользователей с доступом через Kerberos. В небольших сетях или в сетях с небольшим числом пользователей NFS таких проблем обычно не возникает – можно просто синхронизировать файлы */etc/master.passwd* на всех системах или просто присвоить одно и то же значение UID каждому пользователю в каждой системе.

Однако пользователь root обслуживается несколько иначе. Сервер NFS не доверяет пользователям root с других компьютеров выполнять операции на сервере от имени root. В конечном итоге едва ли кто-то захочет, чтобы его сервер был остановлен, если злоумышленник взломает сервер NFS. Запросы, поступающие от root, можно отобразить на любую другую учетную запись. Например, можно потребовать, чтобы все запросы, поступающие от пользователя root клиентского компьютера, обслуживались с привилегиями пользователя на сервере nfsroot. При внимательном отношении к группам можно было бы дать пользователю nfsroot ограниченный доступ к файлам. Чтобы отобразить пользователя root в другого пользователя, следует указать параметр `-maproot`. В следующей строке производится отображение UID 0 (root) на стороне клиента в пользователя с UID 5000 на стороне сервера.

```
/usr/home/mwlucas -maproot=5000
```

Если действительно необходимо, чтобы пользователь root клиентского компьютера обладал привилегиями root на сервере, следует использовать отображение `-maproot` в UID 0. Такой подход может оказаться подходящим для домашней сети или для тестовой системы.

По умолчанию, если не использовать отображение `-maproot`, NFS будет отображать учетную запись удаленного пользователя root в учетную запись `nobody:nobody` на сервере.

Не забывайте перезапускать mountd(8) после редактирования файла */etc/exports*.

Экспорт нескольких каталогов

Из раздела */usr* можно было бы экспортировать множество каталогов. Среди наиболее вероятных кандидатов на экспорт можно было бы назвать каталоги */usr/src*, */usr/obj* и */usr/ports/distfiles*. Список всех каталогов, находящихся на одном и том же разделе, должен располагаться в одной строке файла */etc/exports*, сразу вслед за первым каталогом; разделителем должен быть пробел. Мой файл */etc/exports* сейчас выглядит так:

```
/usr/home/mwlucas /usr/src /usr/obj /usr/ports/distfiles -maproot=5000
```

Здесь нет ни идентификаторов, ни разделителей между отдельными частями строки. Конечно, файл было бы проще читать, если бы описа-

ние каждого экспортируемого каталога находилось в отдельной строке, но это невозможно, так как все эти каталоги находятся на одном и том же разделе. Группа разработчиков FreeBSD могла бы ликвидировать эту проблему, но тогда формат файла */etc/exports* FreeBSD окажется несовместим с форматом файла в других версиях UNIX.

Как и в случае со многими другими конфигурационными файлами, можно использовать символ обратного слэша для деления одной конфигурационной записи на несколько строк. Так, строка, которая была показана выше, может быть записана более удобочитаемым образом:

```
/usr/home/mwlucas \
    /usr/src \
    /usr/obj \
    /usr/ports/distfiles \
    -maproot = 5000
```

Ограничение клиентов

Чтобы дать право доступа к экспортируемым каталогам NFS только определенным клиентам, их следует перечислить в конце записи в файле */etc/exports*. Ниже показана запись, которая разрешает доступ к разделяемым ресурсам только с одного IP-адреса:

```
/usr/home/mwlucas /usr/src /usr/obj /usr/ports/distfiles \
    -maproot=5000 192.168.1.200
```

Подобным образом, используя квантификаторы `–network` и `–mask`, можно дать право доступа только клиентам из определенной сети:

```
/usr/home/mwlucas /usr/src /usr/obj /usr/ports/distfiles \
    -maproot=5000 -network 192.168.0 -mask 255.255.255.0
```

Эта запись позволит клиентам, чьи IP-адреса начинаются с 192.168.0, получить доступ к серверу NFS. Подобные настройки я использую для быстрого обновления клиентов. Я собираю новые пакеты и ядро на сервере NFS, а затем позволяю клиентам смонтировать эти разделы и установить скомпилированные двоичные файлы через NFS.

Комбинирование клиентов и экспортируемых каталогов

Каждая строка в файле */etc/exports* определяет экспортируемые каталоги из одного раздела и хост или несколько хостов клиентов. Для разных хостов можно создать совершенно разные определения экспорта.

```
/usr/home/mwlucas /usr/src /usr/obj /usr/ports/distfiles \
    -maproot=5000 192.168.1.200
/usr -maproot=0 192.168.1.201
```

В этом примере несколько каталогов из раздела */usr* экспортируются клиенту NFS с IP-адресом 192.168.1.200. А клиент с адресом 192.168.1.201 может смонтировать весь раздел */usr*, и даже с правами root.

Производительность NFS и параметры

FreeBSD использует консервативные настройки NFS по умолчанию, поэтому она прекрасно работает с любыми другими UNIX-подобными операционными системами. Если NFS используется исключительно в окружении систем FreeBSD или сеть состоит только из высокопроизводительных систем UNIX, можно повысить производительность NFS за счет использования дополнительных параметров монтирования.

Во-первых, по умолчанию NFS работает через протокол UDP. Если воспользоваться параметром `tcp` или `-T`, для отправки запросов на монтирование клиенты будут использовать протокол TCP.

Программы предполагают, что файловая система не может исчезнуть, но когда речь идет о файловой системе NFS, всегда есть вероятность, что сервер пропадет из сети. В результате программы на стороне клиента, пытающиеся получить доступ к файловой системе NFS, могут зависнуть навсегда. Если с помощью параметра `intr` сделать операцию монтирования NFS *прерываемой* (*interruptible*), можно будет с помощью комбинации `Ctrl-C` прерывать работу процессов, зависших при попытке смонтировать недоступный раздел NFS.

FreeBSD может уведомлять клиентов в случае недоступности файловой системы. В результате вместо зависания программы будут терпеть неудачу при попытке получить доступ к файловой системе.

Наконец, можно установить размеры запросов на чтение и запись. Значения по умолчанию отлично подходили для сетей начала 90-х годов, но вы с помощью ключей `-r` и `-w` можете указать иные размеры, более подходящие для современных условий. Я установил, что для обоих параметров отлично подходит значение 32768. А теперь, объединив все вышесказанное, я мог бы на стороне клиента использовать такую команду монтирования файловой системы NFS:

```
# mount -o tcp,intr,soft,-w=32768,-r=32768 server:/usr/home/mwlucas /mnt
```

Соответствующая ей запись в файле */etc/fstab* выглядит так:

```
server:/usr/home/mwlucas /mnt nfs rw,-w=32768,-r=32768,tcp,soft,intr 0 0
```

Такая конфигурация NFS обеспечивает мне хорошую пропускную способность в локальной сети, ограниченную только возможностями аппаратных средств.

Хотя настройка NFS для простых случаев выполняется достаточно легко, тем не менее ее расширение и улучшение может занять массу времени. Если вы предполагаете выстраивать сложные схемы построения окружения NFS, не полагайтесь на это короткое введение, а лучше потратьте свое время на изучение хорошей книги по данной теме.

FreeBSD и CIFS

В типичной офисной сети стандартным протоколом совместного использования файлов в сети является общая межсетевая файловая система (Common Internet File Sharing), или CIFS. (Некогда CIFS была известна как блок сообщений сервера (Server Message Block), или SMB.) Это то самое «Сетевое окружение» («Network Neighborhood»), доступ к которому имеют пользователи Windows. Изначально этот протокол поддерживался только операционной системой Windows, но позднее он стал чем-то вроде стандарта. К счастью, в настоящее время имеется реализация сервера CIFS с открытыми исходными текстами под названием Samba. Многие коммерческие продукты предоставляют свои услуги через этот протокол. Система FreeBSD также включает в себя модули ядра, обеспечивающие поддержку файловой системы и программ для поиска, монтирования и использования разделяемых ресурсов CIFS.

Подготовительные операции

Прежде чем использовать сетевую файловую систему компании Microsoft, необходимо собрать следующую информацию о сети Windows:

- Название рабочей группы или домена
- Имя пользователя Windows и пароль
- IP-адрес Windows-сервера DNS

Поддержка в ядре

Поддержка CIFS в операционной системе FreeBSD осуществляется несколькими модулями ядра. Базовые операции CIFS поддерживаются модулем *smbfs.ko*. Модули *libmchain.ko* и *libiconv.ko* предоставляют функции поддержки и загружаются автоматически вместе с модулем *smbfs.ko*. Вы можете скомпилировать эти модули в ядре статически:

```
options NETSMB
options LIBMCHAIN
options LIBICONV
options SMBFS
```

Безусловно, эти модули можно загружать во время загрузки системы, поместив соответствующие записи в файл */boot/loader.conf*.

Конфигурирование CIFS

Настройки CIFS могут находиться в конфигурационном файле *$HOME/.nsmbrc* или */etc/nsmb.conf*. Параметры настройки в файле */etc/nsmb.conf* имеют приоритет перед настройками, которые размещаются в домашних каталогах. Конфигурационный файл делится на разделы с помощью меток, заключенных в квадратные скобки. Например, настройки, которые применяются к каждому соединению CIFS, находятся в разделе `[default]`. Создайте свои собственные разделы

с указанием серверов, пользователей и разделяемых ресурсов в следующих форматах:

```
[servername]
[servername:username]
[servername:username:sharename]
```

Информация, которая применяется ко всему серверу, помещается в раздел, который следует сразу за именем сервера. Информация, которая применяется к конкретному пользователю, помещается в раздел с именем этого пользователя, а информация, применяемая к конкретному разделяемому ресурсу, вставляется в раздел, заголовок которого включает имя разделяемого ресурса. Если у вас нет информации, конкретной для каждого пользователя или разделяемого ресурса, тогда все сведения можно поместить в один большой раздел `[servername]`.

В конфигурационных записях используются значения из системы CIFS, например, моя учетная запись во FreeBSD имеет имя mwlucas, а в Windows – lucas_m, поэтому я использую имя lucas_m в файле *nsmb.conf*.

Ключевые слова в nsmb.conf

Параметры настройки в файле *nsmb.conf* определяются с помощью ключевых слов и значений в соответствующих разделах. Например, серверы имеют IP-адреса, а пользователи – нет, поэтому параметр, определяющий IP-адрес, должен находиться только в разделе описания сервера. Чтобы использовать ключевое слово, ему нужно присвоить значение через знак равенства: *keyword=value*. Далее перечисляются наиболее часто употребляемые – полный список можно найти в странице руководства nsmb.conf(5).

workgroup=string

Ключевое слово `workgroup` определяет имя домена Windows или рабочей группы, доступ к которой требуется получить. Это типичная настройка по умолчанию, используемая для всех серверов.

addr=a.b.c.d

Ключевое слово `addr` определяет IP-адрес сервера CIFS. Данное ключевое слово может находиться только в разделе с меткой `[servername]`.

nbns=a.b.c.d

Ключевое слово `nbns` определяет IP-адрес сервера имен NetBIOS (WINS). Вы можете добавить эту строку в раздел по умолчанию или в раздел с конкретным сервером. Если у вас имеется служба ActiveDirectory (которая основана на DNS), можно использовать имена хостов в DNS. Добавление сервера WINS не повредит вашей конфигурации и поможет проверить базовые настройки CIFS.

password=string

Ключевое слово `password` определяет пароль в открытом текстовом виде для пользователя или разделяемого ресурса. Если вам придется хранить пароли в */etc/nsmb.conf*, вы должны обеспечить доступность этого файла на чтение только для пользователя root. Хранение паролей в многопользовательских системах в файле *$HOME/.nsmbrc* вообще недопустимо.

Пароли Windows можно зашифровать с помощью команды `smbutil crypt`. Она сгенерирует строку, которую можно использовать как значение этого ключевого слова. Зашифрованная строка начинается с двух символов доллара (`$$`). Шифрование предотвращает случайное раскрытие пароля, но злонамеренный пользователь легко может расшифровать его.

Разрешение имен CIFS

Давайте создадим типичный файл *nsmb.conf*. Для начала нам необходимо как минимум иметь возможность находить хосты в сети, а это означает, что нам необходимо знать имя рабочей группы. У нас имеется контроллер домена в виде сервера WINS. Кроме того, у меня имеется учетная запись на серверах Windows, предоставляющих доступ к разделяемым ресурсам, поэтому я буду использовать их как настройки по умолчанию в *nsmb.conf*.

```
[default]
workgroup=BIGLOSER
nbns=192.168.1.66
username=unix
```

Взяв эту информацию на вооружение, мы сможем производить базовые запросы на получение имен SMB.

```
# smbutil lookup ntserv1
Got response from 192.168.1.66
IP address of ntserv1: 192.168.1.4
```

Если все получилось, значит в вашем распоряжении имеется базовая функциональность CIFS.

Прочие функции smbutil(1)

Прежде чем можно будет монтировать разделяемые ресурсы Windows, необходимо иметь возможность зарегистрироваться на компьютере Windows. Эта операция доступна только пользователю root.

```
# smbutil login //unix@ntserv1
Password:
```

Итак, наша конфигурация правильная. Давайте с помощью команды `smbutil view` посмотрим, какие ресурсы предлагает этот сервер.

```
# smbutil view //unix@ntserv1
Password:
Share        Type       Comment
-------------------------------
IPC$         pipe       Remote IPC
ADMIN$       disk       Remote Admin
C$           disk       Default share
unix      disk
4 shares listed from 4 available
```

Вы получите список всех разделяемых ресурсов на сервере CIFS. Теперь предположим, что мы завершили работу с сервером, и выходим из него.

```
# smbutil logout //unix@ntserv1
```

Монтирование разделяемого ресурса

Теперь, закончив исследования, можно выполнить монтирование ресурса с помощью mount_smbfs(8). Эта команда имеет следующий синтаксис:

```
# mount_smbfs //username@servername/share /mount/point
```

У меня имеется разделяемый ресурс *MP3* на компьютере, работающем под управлением Windows, к которому я хочу получить доступ из системы FreeBSD. Чтобы смонтировать его в каталог */home/mwlucas/smbmount*, я мог бы запустить следующую команду:

```
# mount_smbfs //unix@ntserv1/MP3 /home/mwlucas/smbmount
```

mount(8) и df(1) показывают, что этот ресурс подключен к системе и теперь можно обращаться к документам на этом сервере, как к файлам в любой другой файловой системе.

Другие параметры mount_smbfs

Утилита mount_smbfs поддерживает несколько ключей, влияющих на поведение монтируемых файловых систем CIFS. Ключ `-f` используется для выбора другого режима назначения прав доступа к файлам, а ключ `-d` – для выбора другого режима назначения прав доступа к каталогам. Например, чтобы определить права доступа к монтируемому каталогу, которые позволят обращаться к нему только мне, я мог бы запустить команду `mount_smbfs -d 700`. Это позволило бы сделать права доступа из системы FreeBSD более строгими, чем привилегии в Windows, и в моем случае это было бы именно то, что нужно. Владельца файла можно изменить с помощью ключа `-u`, а группу – с помощью ключа `-g`.

Имена файлов в файловых системах корпорации Microsoft не чувствительны к регистру символов, а в системах UNIX – наоборот. Файловая система CIFS по умолчанию оставляет регистр символов в именах файлов без изменений, но это может оказаться нежелательно. Ключ `-c` вынуждает mount_smbfs(8) менять регистр символов в именах файлов

и каталогов: `-c l` изменяет регистр всех символов на нижний, а `-c u` — на верхний.

Примеры записей в nsmb.conf

Ниже приводятся примеры записей в файле *nsmb.conf* для различных ситуаций. В этих примерах предполагается, что все представленные записи являются частью конфигурации, в которой уже определены рабочая группа, сервер имен NetBIOS и имя пользователя с привилегиями доступа к разделяемым ресурсам CIFS.

Уникальный пароль на отдельной системе

Если предположить, что имеется некоторый компьютер с сетевым именем desktop, на котором открыт разделяемый ресурс, защищенный паролем, то можно использовать запись, аналогичную приведенной ниже. Многие системы Windows 9x используют такую возможность.

```
[desktop:shareusername]
password=$$1789324874ea87
```

Доступ ко второму домену

В этом примере описывается доступ ко второму домену с именем development. В этом домене используются имя пользователя и пароль, которые отличаются от тех, что используются в домене по умолчанию.

```
[development]
workgroup=development
username=support
```

Принадлежность файлов в CIFS

Различия в определении принадлежности файлов в UNIX-подобных операционных системах и в Windows могут порождать проблемы. Начнем с того, что имена пользователей FreeBSD, возможно, не получится отобразить в имена пользователей Windows, а кроме того, в UNIX и в Windows используются совершенно непохожие схемы назначения прав доступа.

Если в Windows вы используете отдельную учетную запись для доступа к разделяемому ресурсу, вы обладаете теми правами доступа, которые имеет эта учетная запись в Windows, однако в FreeBSD для этого монтируемого ресурса вам необходимо назначить права доступа, используемые системой FreeBSD. По умолчанию утилита mount_smbfs(8) определяет те же самые права доступа для нового разделяемого ресурса, какие указаны для точки монтирования. В предыдущем примере владельцем каталога */home/mwlucas/smbmount* был пользователь mwlucas, а для каталога были определены права 755. Эти права доступа говорят, что пользователь mwlucas обладает возможностью редактировать все, что находится в этом каталоге, но все остальные лишены

такого права. Но даже когда FreeBSD позволяет данному пользователю редактирование указанных файлов, Windows может не позволить ему изменять файлы, находящиеся на разделяемом ресурсе.

Обслуживание разделяемых ресурсов CIFS

Точно так же, как операционная система FreeBSD может обращаться к разделяемым ресурсам CIFS, с помощью Samba она может предоставлять доступ клиентам CIFS к своим разделяемым ресурсам. Найти пакет Samba можно в каталоге */usr/ports/net/samba3*. На веб-сайте проекта Samba (*http://www.samba.org*) кроме всего прочего можно найти множество полезных руководств. Обслуживание разделяемых ресурсов в операционной системе FreeBSD выполняется намного сложнее, чем доступ к ним, поэтому мы закончим обсуждение этой темы, чтобы книга не стала слишком толстой.

devfs

devfs(5) – это динамическая файловая система, предназначенная для управления файлами устройств. Не забывайте, что в UNIX-подобных системах *все сущее* является файлами. Это относится и к физическим устройствам. Для каждого устройства в системе имеется свой файл устройства в каталоге */dev*.

Когда-то давно системный администратор нес ответственность за создание этих файлов устройств. Наиболее удачливые администраторы получали систему, поставляемую вместе с файлом сценария на языке командного интерпретатора, который создавал файлы устройств и определял права доступа к ним. Если производитель операционной системы не прилагал подобный сценарий, или к серверу были подключены необычные устройства, которые не обрабатывались этим сценарием, то системному администратору приходилось создавать файлы устройств с помощью магических заклинаний и утилиты mknod(8). В этом случае при малейшей ошибке устройство могло не работать. Как вариант можно было бы распространять операционную систему с файлами устройств для всех мыслимых аппаратных средств. Тогда системные администраторы могли бы быть уверены, или почти уверены, что нужный файл устройства находится где-то в каталоге */dev* среди тысяч других файлов.

Безусловно, ядро точно знает, какими характеристиками должен обладать каждый файл устройства. Посредством devfs(5) операционная система FreeBSD просто спрашивает у ядра, какие файлы устройств, по его мнению, должны иметься в системе, и создает только те, которые действительно необходимы. Этот подход оправдывает себя в большинстве случаев. Однако наш с вами случай не попадает в это «большинство». Мы должны быть готовы к появлению многих неожиданностей. Возможно, нам потребуется создавать файлы устройств с други-

ми именами, изменять права доступа или настраивать устройства уникальным образом. В операционной системе FreeBSD задача управления файлами устройств имеет три подзадачи: конфигурирование устройств на этапе загрузки, глобальная видимость и права доступа, а также конфигурирование устройств с помощью devd(8), которые появляются динамически уже после загрузки.

Управление устройствами и серверы

В большинстве случаев управление файлами устройств на серверах не предполагает каких-либо дополнительных настроек и вмешательства со стороны человека. Чаще всего с файлами устройств приходится возиться на ноутбуках и иногда на рабочих станциях. Инструментальные средства операционной системы FreeBSD, предназначенные для управления файлами устройств, обладают высокой гибкостью и включают поддержку таких ситуаций, с которыми я и не предполагал столкнуться в ближайшие сто лет. Мы затронем лишь самые основы. Не надо думать, что вам обязательно нужно овладеть навыками работы с devfs(5), чтобы обеспечить бесперебойную работу своего сервера!

devfs на этапе загрузки: devfs.conf

Самая большая проблема системного администратора, связанная с динамической файловой системой devfs, состоит в том, что все сделанные изменения исчезают при перезагрузке системы. Когда файлы устройств постоянно хранились на диске, системный администратор мог создавать символические ссылки на эти файлы или изменять права доступа к ним, не беспокоясь, что его изменения могут исчезнуть. С появлением автоматизированной и динамической файловой системы устройств такой уверенности больше нет. (Конечно, вам больше не нужно овладевать оккультными командами mknod(8), так что вам, скорее, повезло.) Среди изменений файлов устройств можно назвать следующие:

- Создание файлов устройств под различными именами
- Изменение принадлежности файлов устройств
- Сокрытие файлов устройств от пользователей

На этапе загрузки devfs(5) создает файлы устройств в соответствии с правилами в файле */etc/devfs.conf*.

devfs.conf

С помощью файла */etc/devfs.conf* можно создавать символические ссылки, изменять принадлежность файлов устройств и определять права

доступа к устройствам на этапе загрузки. Каждое правило имеет следующий формат:

```
операция        устройство      желаемое_значение
```

В качестве операции можно употреблять `link` (создание ссылки), `perm` (установка прав доступа) и `own` (установка принадлежности). Под устройством понимается существующий файл устройства, а последнее поле в записи – это желаемое значение. Например, ниже для файла устройства создается новое имя:

```
❶link           ❷acd0           ❸cdrom
```

Нам требуется создать символическую ссылку ❶ на файл устройства */dev/acd0* ❷ (привод ATAPI CD), и чтобы ссылка имела имя */dev/cdrom* ❸. Если перезагрузиться с этой записью в */etc/devfs.conf*, наше устройство */dev/acd0* будет также доступно как */dev/cdrom*, как того ожидают многие мультимедийные программы.

Чтобы изменить права доступа к файлу устройства, эти права должны быть определены в восьмеричной форме в поле желаемого значения:

```
perm            acd0            666
```

Здесь определены такие права доступа к устройству */dev/acd0* (все тот же привод компакт-дисков), которые позволят любому пользователю в системе выполнять запись и чтение с этого устройства. Запомните, изменение прав доступа к символической ссылке */dev/cdrom* никак не отразится на правах доступа к файлу устройства – это всего лишь символическая ссылка.

Наконец, можно изменить владельца устройства. Изменение владельца обычно указывает, что вы пытаетесь решить проблему не совсем правильным способом и, возможно, вам следует остановиться и подумать. Однако FreeBSD не будет возражать против такого вмешательства в систему, если вы настаиваете на этом. Следующая строка дает конкретному пользователю безраздельный контроль над дисковым устройством */dev/da20*:

```
own             da20            mwlucas:mwlucas
```

Однако это может и не дать желаемый эффект, поскольку некоторые программы по-прежнему могут считать, что вы должны обладать правами root для выполнения операций с устройством. Я встречал такие программы, которые завершались самостоятельно, если их запускали с привилегиями обычного пользователя, при этом они даже не пытались обратиться к файлу устройства. Изменение прав доступа к файлу устройства никак не отразится на поведении таких программ, если они будут запускаться без привилегий пользователя root.

Процедура конфигурирования devfs(5) позволит решить многие проблемы, но далеко не все. Если необходимо, чтобы устройство было просто невидимо и недоступно, следует использовать правила devfs.

Глобальные правила devfs

Каждый экземпляр devfs(5) ведет себя в соответствии с правилами, которые определены в файле *devfs.rules*. Правила devfs применяются как к устройствам, которые обнаруживаются на этапе загрузки системы, так и к устройствам, которые подключаются и отключаются во время работы системы. Правила позволяют изменять принадлежность файлов устройств, права доступа к ним, а также могут делать их видимыми или невидимыми. Но с помощью правил devfs нельзя создать символическую ссылку на файл устройства.

FreeBSD использует файлы */etc/devfs.rules* и */etc/defaults/devfs.rules* подобно файлам */etc/rc.conf* и */etc/defaults/rc.conf*. Добавляйте свои правила в файл */etc/devfs.rules* и не трогайте содержимое файла по умолчанию.

Формат определения набора правил devfs

Каждый набор правил devfs начинается с имени и номера в квадратных скобках. Например, ниже приводятся несколько базовых правил из конфигурации по умолчанию:

```
[❶devfsrules_hide_all=❷1]
❸add hide
```

Первый набор правил называется `devfs_hide_all` ❶ и имеет номер 1 ❷. Этот набор содержит всего одно правило ❸.

Содержимое набора правил

Все правила devfs (в файлах) должны начинаться со слова `add`, которое добавляет правило в набор. После него можно указать ключевое слово `path` и определить регулярное выражение, которое будет соответствовать именам устройств, либо `type`, и указать тип устройства. В конце правила должно находиться определение выполняемого *действия* или команды. Ниже приводится пример правила devfs:

```
add path da* user mwlucas
```

Это правило назначает пользователя mwlucas владельцем всех файлов устройств, чьи имена начинаются на `da`. В многопользовательских системах или в системах, оснащенных жесткими дисками SCSI, этого делать не следует. На ноутбуке, где на `da` начинаются только имена USB-устройств хранения данных, это не такая плохая идея.

При определении устройств с помощью ключевого слова `path` используются стандартные регулярные выражения командного интерпретатора. Если необходимо, чтобы правило применялось к нескольким устройствам, используйте символ звездочки в качестве шаблонного символа. Например, определение `path ad1s1` соответствует единственному устройству */dev/ad1s1*, а определение `path ad*s*` будет соответствовать всем устройствам, имена которых начинаются с `ad`, за которыми могут

следовать символы, далее находится символ s, вслед за которым также могут следовать дополнительные символы. Точно определить, какие устройства будут соответствовать шаблону, можно с помощью командной строки:

```
# ls /dev/ad*s*
```

Эта команда выведет все участки и разделы на жестких дисках ATA, но не выведет файлы устройств, которые соответствуют целым дискам.

Ключевое слово type указывает, что данное правило применяется ко всем устройствам заданного типа. Здесь можно использовать ключевые слова disk (дисковые устройства), mem (устройства памяти), tape (накопители на магнитных лентах) и tty (устройства терминалов, включая псевдотерминалы). Ключевое слово type используется достаточно редко именно потому, что оно охватывает целые группы устройств.

Если в правило не включается ключевое слово type или path, то оно применяется ко всем файлам устройств, что практически во всех случаях нежелательно.

В качестве выполняемого действия может быть использовано одно из следующих: group, user, mode, hide и unhide. Действие group позволяет определять принадлежность файла устройства к группе, название которой передается в виде дополнительного аргумента. Похожим образом действие user позволяет назначить владельца устройства. Следующие правила делают пользователя desktop и группу usb владельцами дисков da:

```
add path da* user desktop
add path da* group usb
```

Действие mode позволяет определять права доступа к устройству в стандартной восьмеричной форме:

```
add path da* mode 664
```

Ключевое слово hide позволяет скрывать файлы устройств, а unhide – делать их видимыми. Так как ни одна программа не может использовать файл устройства, если он невидим, данная возможность находит ограниченное применение, за исключением случаев использования jail(8). Мы будем рассматривать эту специфическую область применения команд hide и unhide в следующей главе.

После того как набор правил devfs будет определен, его можно будет активировать в файле */etc/rc.conf*. Следующая строка активирует набор правил devfs с именем laptoprules:

```
devfs_system_rulesets="laptoprules"
```

Не забывайте, что правила devfs применяются как к устройствам, которые определяются на этапе загрузки, так и к устройствам, которые подключаются динамически, после запуска системы. В заключение посмотрим на устройства, подключаемые динамически.

Управление динамическими устройствами с помощью devd(8)

Устройства, допускающие подключение в процессе работы системы, в наше время становятся более привычными, чем, например, десять лет назад. В прошлом столетии сетевые карты для ноутбуков считались ультрасовременными устройствами. Хотя вы можете менять компакт-диски в лотке привода CD-ROM, тем не менее вам едва ли пришлось бы на ходу менять сам привод. В наши дни ситуация полностью изменилась. Flash-накопители, которые по сути можно рассматривать как жесткие диски, подключаемые к порту USB, допускают подключение и отключение в любой момент времени, так же как и другие USB-устройства, такие как клавиатуры, мыши и даже сетевые карты.

Файловая система devfs динамически создает новые файлы устройств при подключении таких устройств и удаляет их при отключении, что существенно упрощает использование таких устройств. devd(8) позволяет даже запускать пользовательские программы при включении или отключении устройств.

Конфигурирование devd

Демон devd(8) читает свои настройки из файла */etc/devd.conf* и любых других файлов в каталоге */usr/local/etc/devd/*. Чтобы упростить дальнейшее обновление системы, я рекомендую размещать свои локальные правила в файле */usr/local/etc/devd/devd.conf*. Если конфигурация devd(8) становится слишком сложной, можно разместить правила, касающиеся устройств разных типов, в отдельных файлах. Для devd(8) существует четыре типа правил: `attach`, `detach`, `nomatch` и `notify`.

Правила `attach` вызываются, когда соответствующие аппаратные устройства подключаются к системе. Например, когда подключается сетевая карта, правило attach может выполнить настройку IP-адреса и запустить ее в работу.

Правила `detach` вызываются, когда соответствующие аппаратные устройства отключаются от системы. Правила `detach` используются редко, так как ядро автоматически помечает ресурсы как недоступные при отключении устройств, однако вы сможете найти им применение.

Правила `nomatch` вызываются, когда новое оборудование подключено, но не связано с драйвером устройства, то есть когда для устройства отсутствует драйвер в текущем ядре.

Правила `notify` применяются демоном devd(8), когда ядро отправляет соответствующее уведомление в пространство пользователя. Примером события типа `notify` может служить вывод на консоль сообщения о том, что сетевой интерфейс подключен. Обычно извещения выводятся на консоль или в файл протокола */var/log/messages*.

Правила devd(8) имеют систему приоритетов, где 0 обозначает самый низкий приоритет. Обрабатываются только высокоприоритетные правила, низкоприоритетные – пропускаются. Ниже приводится пример правила devd(8):

```
❶notify ❷0 {
          match "system"          ❸"IFNET";
          match "type"            ❹"ATTACH";
          action ❺"/etc/pccard_ether $subsystem start";
};
```

Это правило `notify` ❶, которое применяется, когда ядро отправляет сообщение в пространство пользователя. Так как правило имеет приоритет 0 ❷, оно будет применяться только в том случае, если будет отсутствовать более высокоприоритетное правило с тем же критерием. Это правило применяется, только если сообщение отправлено сетевой подсистемой `IFNET` ❸ и только если сообщение имеет тип ATTACH ❹ или, другими словами, – когда происходит запуск сетевого интерфейса. При этих обстоятельствах devd(8) запускает команду настройки сетевого интерфейса ❺.

devd(8) поддерживает массу параметров на все случаи жизни. Если необходимо автоматически монтировать USB flash-диски в определенные точки монтирования, этого можно добиться, проверяя серийные номера каждого подключаемого USB-устройства. Если необходимо настроить сетевую карту Intel, отличающуюся от сетевых карт 3Com, это также вполне возможно. Далее будет рассказываться достаточно много о конфигурировании devd(8), в основном на примерах, – мы не будем слишком углубляться в подробности.

Пример конфигурирования devd(8): ноутбуки

Я пользуюсь своим ноутбуком и на работе, и дома. На работе у меня проводное подключение к сети, дома – беспроводное. В обоих случаях требуются весьма специфические установки, различные серверы DNS и прочие параметры настройки. Я мог бы схитрить и использовать в моей домашней сети адреса, которые совпадали бы с адресами в сети на работе, но мой опыт подсказывает, что домашняя сеть будет существовать много дольше, чем любая моя работа. (Возможно, я не прав.[1]) Я хочу, чтобы мой ноутбук сам выполнял необходимые настройки дома – при подключении беспроводной карты. Я хочу, чтобы мой ноутбук сам выполнял необходимые настройки на работе – при подключении кабеля сети к интегрированному сетевому интерфейсу.

[1] Было бы неплохо когда-нибудь получить работу, которая не закончилась бы неприятным моментом, когда четыре гориллы из корпоративной службы безопасности выбрасывают меня за двери, а мой бывший босс сзади кричит что-то о федеральных органах регулирования, крем-брюле и возбужденных лемурах. Это всего лишь случайность, заметьте, так как никто не может меня обвинить в чем-либо.

Если проводной интерфейс настроен на использование DHCP, то при подключении кабеля FreeBSD должна запустить dhclient(8) и выполнить настройку сети. При подключении беспроводного интерфейса FreeBSD должна проверить конфигурацию и попытаться подключиться к сети. Ниже приводится конфигурация из */etc/rc.conf* для моего беспроводного интерфейса (в действительности это одна строка, но я разделил ее, чтобы уместить на книжную страницу):

```
ifconfig_wi0="inet dhcp ssid WriteFaster wepkey
0xdeadbeefbadc0decafe1234567 weptxkey 1 wepmode on channel 1"
```

Когда я подключаю свою беспроводную карту, ядро подключает к ней драйвер wi0. В результате этого возникает событие `attach`, по которому devd(8) отыскивает соответствующее правило. Этому событию соответствует следующая запись:

```
❶attach ❷0 {
        ❸media-type "802.11";
        ❹action "/etc/pccard_ether $device-name start";
};
```

Итак, в ноутбук была вставлена беспроводная карта, и FreeBSD подключила к ней драйвер wi(4). В результате возникло событие `attach` ❶. Это правило имеет приоритет 0 ❷, поэтому другое соответствующее правило будет запущено до этого правила по умолчанию. Этому правилу соответствуют любые устройства типа 802.11 ❸ (беспроводные). Когда по событию `attach` будет найдено соответствующее правило, FreeBSD запустит команду ❹ конфигурирования сети. Я мог бы написать и запускать свой сценарий настройки, но к чему делать лишнюю работу, когда имеющаяся во FreeBSD встроенная поддержка обеспечивает все, что мне необходимо?

В отличие от беспроводного интерфейса, который я использую дома, интерфейс Ethernet, используемый на работе, постоянно подключен к моей системе. Интерфейс может быть не подключен к сети, но сама карта всегда подключена. Мне совсем не нужно, чтобы этот интерфейс настраивался при каждой загрузке системы, потому что если интерфейс не подключен к сети, то будут возникать длительные задержки, когда демон SSH и другие службы, работающие на моем ноутбуке, будут ожидать истечения тайм-аута запросов DNS. На работе у меня обычно смонтированы некоторые разделы NFS, а также выполняются некоторые нестандартные настройки, недоступные при использовании DHCP.

Это означает, что я должен запустить свой собственный сценарий на языке командного процессора для настройки параметров под сетевое окружение на работе, но только при подключении кабеля сети. Для этого надо, чтобы devd дождался извещения о подключении кабеля к интерфейсу и затем запустил мой сценарий. Для этого я создал следующую запись, которую поместил в файл */usr/local/etc/devd/devd.conf*:

```
❶notify ❷10 {
          match "system" ❸"IFNET";
          match "type" ❹"LINK_UP";
          media-type ❺"ethernet";
❻#         action "/etc/rc.d/dhclient start $subsystem";
         ❼action "/home/mwlucas/bin/jobnetwork.sh";
};
```

Это правило описывает реакцию devd на извещение ❶, которое применяется, когда ядро посылает системе соответствующее извещение. Кроме того, это правило имеет приоритет 10 ❷, поэтому оно будет иметь преимущество перед любыми правилами по умолчанию. Данное правило применяется, когда сетевая подсистема ❸ обнаруживает подключение ❹ к сети (то есть подключение кабеля к сетевому интерфейсу) при условии, что это сеть Ethernet ❺. Когда будут соблюдены все эти условия, FreeBSD запустит сценарий ❼, расположенный в моем домашнем каталоге.

Обратите внимание: это правило содержит закомментированную строку ❻. Я несколько погрешил против истины, когда говорил, что написал это правило. В действительности я скопировал правило по умолчанию из */etc/devd.conf* и несколько изменил его. Операционная система FreeBSD уже имеет правила, которые применяются при подключении сетевых интерфейсов к сети, но мне требовалось выполнить ряд действий, которые не предусмотрены правилом по умолчанию. Вместо того, чтобы вспоминать все необходимые условия, я просто нашел запись, которая почти удовлетворяла меня, и немного изменил ее. Я рекомендую вам поступать точно так же. Достаточно лишь присвоить своему правилу более высокий приоритет, чем правило по умолчанию, и тогда FreeBSD будет использовать ваше правило вместо своего собственного.

Еще один пример конфигурирования devd: flash-устройства

К сожалению, не всегда удается копировать записи. Теперь мы рассмотрим пример, когда FreeBSD не предлагает правил по умолчанию. Я хотел бы, чтобы при подключении flash-накопителя он автоматически монтировался бы в каталог */media*. Первое устройство USB, подключенное к ноутбуку, появится в системе как */dev/da0*, но имя устройства будет umass0. (Если у вас в системе уже имеются подключенные устройства USB или SCSI, номер устройства в вашем случае будет иным.) Вообще говоря, все flash-накопители отформатированы в файловой системе FAT32, поэтому для их монтирования следует использовать команду `mount -t msdosfs`.

При подключении устройства FreeBSD подключает его к ядру, поэтому следует использовать правило `attach`:

```
attach 10 {
       match "device-name" ❶"umass0";
       action "/home/mwlucas/bin/mountusb.sh";
};
```

Я присвоил этому правилу приоритет 10, поэтому оно будет иметь преимущество перед любыми другими правилами с приоритетом 0, которые могли бы быть установлены при обновлении версии FreeBSD. (Если FreeBSD когда-нибудь окажется в состоянии делать это автоматически, я, вероятно, воспользуюсь этой возможностью, но пока я не хочу, чтобы любое обновление изменило поведение системы без моего ведома.) Критерием в этом правиле является имя устройства ❶. При подключении устройства система запускает сценарий командного интерпретатора, расположенный в моем домашнем каталоге.

Почему бы просто не запустить команду mount(8)? Дело в том, что устройствам USB требуется одна-две секунды на разогрев и стабилизацию, и только потом к ним можно будет обращаться. Кроме того, некоторые flash-накопители необходимо «подергать», прежде чем они действительно начнут работать. Сценарий, который я использую, весьма незамысловат, но меня он вполне устраивает:

```
#!/bin/sh
sleep 2
/usr/bin/true > /dev/da0
/sbin/mount -t msdosfs /dev/da0s1 /media
```

Такой прием поможет смонтировать устройство, но devd(8) не в состоянии демонтировать его. Операционная система FreeBSD предполагает, что, прежде чем вынимать устройство, системный администратор демонтирует файловые системы, находящиеся на нем. Это ничем не отличается от процедуры демонтирования компакт-диска (или дискеты) прежде, чем вынимать его.

Демон devd(8) имеет множество более сложных особенностей, тем не менее эти два примера наглядно демонстрируют все его возможности, которые мне необходимы и которые я использую уже несколько лет. Если вам необходима дополнительная информация, обязательно прочитайте страницы справочного руководства (man). А теперь, когда вы чуть больше знаете о файловых системах FreeBSD, давайте поговорим о более эффективных инструментах обеспечения безопасности.

9

Расширенные средства защиты

В состав FreeBSD входит множество инструментов, обеспечивающих защиту сетевого трафика и пользователей. Например, средства управления трафиком позволяют разрешить и запретить установку соединений с определенными сегментами Интернета. Реализовать это можно разными способами. Вы также можете засадить пользователей в виртуальную машину, называемую *клеткой* (*jail*), где они имеют доступ ко всему, что важно для них, и ни к чему, что важно для вас. Функция фильтрации пакетов позволит вам управлять доступом к системе. В данной главе будут рассмотрены указанные инструменты и методы, а также мониторинг защиты системы и шаги, которые необходимо предпринять при атаке злоумышленника.

Начнем с основ – с непривилегированных учетных записей.

Непривилегированные учетные записи

Непривилегированные учетные записи – это специализированные учетные записи, созданные для решения специфических задач. Многие программы работают с правами непривилегированных пользователей или используют такие учетные записи для выполнения ограниченных действий.

«Пользователь должен обладать только теми правами, которые позволят ему решать поставленные задачи», – разве это правило не подходит для всех учетных записей обычных пользователей? Да, это так, но все-таки учетные записи, используемые людьми, предоставляют более широкие права, чем это необходимо многим программам. Любой пользователь, обладающий доступом к командному интерпретатору, имеет домашний каталог. Обычный пользователь может создавать файлы в этом каталоге, запускать текстовые редакторы или работать с электронной почтой. Типичный пользователь системы нуждается в ко-

мандном интерпретаторе, а программы – нет. Ограничивая права программы, в частности – сетевого демона, вы тем самым ограничиваете объем повреждений, которые может нанести злоумышленник при взломе этой программы.

В составе FreeBSD имеется несколько непривилегированных учетных записей. Загляните в файл */etc/passwd*, где можно увидеть такие учетные записи, как audit, bind, uucp и www. Все это – непривилегированные учетные записи, которые используются демонами-серверами. Посмотрим, что у них общего.

Непривилегированные пользователи не имеют нормального домашнего каталога. Для многих из них в качестве домашнего каталога указан каталог */nonexistent*, хотя некоторые из них, например sshd, имеют специальный домашний каталог */var/empty*. Наличие домашнего каталога, недоступного для чтения и записи, ограничивает возможности учетной записи, которых, впрочем, вполне достаточно для работы демона. Эти пользователи могут являться владельцами некоторых файлов в системе, но обычно они не имеют права на запись в них.

Точно так же никто и никогда не должен входить в систему под этими учетными записями. Если учетная запись bind зарезервирована для сервера DNS, никто не должен входить в систему под учетной записью bind! Для такой учетной записи должен быть определен такой командный процессор, который запрещает регистрацию в системе, например */usr/sbin/nologin*. Каким образом все это повышает уровень безопасности? Давайте рассмотрим на примере.

Обычно веб-сервер работает под непривилегированной учетной записью www. Предположим, что злоумышленник обнаружил уязвимость в версии программы веб-сервера, которой вы пользуетесь, и получил возможность заставить веб-сервер исполнять произвольный программный код. Самая неприятная брешь в системе безопасности – когда злоумышленник оказывается в состоянии заставить программу сервера выполнять программный код с ее привилегиями. Что *имеется* в пределах возможностей программы?

Скорее всего, злоумышленник пожелает получить доступ к командной строке. Командная строка в UNIX-подобных системах – это дверь в мир широких возможностей. Непривилегированному пользователю назначен командный процессор, который не допускает возможности входа в систему. Благодаря этому перед злоумышленником возникает дополнительный барьер, существенно осложняющий получение доступа к командной строке.

Если злоумышленник достаточно хитер, то командный процессор nologin не остановит его. Предположим, что злоумышленник хитростью сумел заставить веб-сервер запустить простой командный процессор, такой как */bin/sh*, и получил в свое распоряжение командную строку. Теперь он может дать волю своему воображению и нанести серьезный ущерб… или не может?

У него нет домашнего каталога и нет права создавать каталоги. Это означает, что любые файлы, которые ему понадобятся, должны размещаться в общедоступных каталогах, таких как */tmp* или */var/tmp*, что демаскирует злоумышленника. Конфигурационный файл веб-сервера Apache принадлежит пользователю root или группе администраторов веб-сервера, но пользователь www не принадлежит этой группе. У злоумышленника может быть доступ к веб-серверу, но нет возможности изменить его конфигурацию. Он не сможет изменить файлы, так как пользователь www не владеет ими. В действительности пользователь www вообще не имеет никакого доступа к системе. Взломав веб-сервер, злоумышленник теперь должен взломать систему или веб-приложение.

Однако следует понимать, что использование непривилегированных учетных записей не решает всех проблем безопасности. Получив доступ к веб-серверу под учетной записью www, злоумышленник может просматривать содержимое исходных файлов веб-приложения. Если приложение написано неграмотно или в его исходных текстах, в открытом виде, хранятся пароли доступа к базам данных, это грозит существенными неприятностями. Однако, если вы своевременно обновляете систему, злоумышленнику придется немало потрудиться, чтобы проникнуть в саму систему FreeBSD.

Учетная запись nobody

В течение многих лет системные администраторы использовали учетную запись *nobody* в качестве типичной непривилегированной учетной записи. Под этой учетной записью они запускали веб-серверы, прокси-серверы и другие программы. Для безопасности это было гораздо лучше, чем запускать те же программы с правами root, но не так хорошо, как иметь отдельные учетные записи для каждого демона. Если злоумышленнику удавалось взломать одну из таких программ, он получал доступ сразу к ним ко всем. У нашего гипотетического злоумышленника, взломавшего веб-сервер, сразу появлялся бы доступ не только к веб-серверу, но и ко всем программам, запущенным с привилегиями этого пользователя! Если вы используете NFS, помните, что по умолчанию учетная запись удаленного пользователя root отображается в учетную запись nobody. В общем случае использование непривилегированных учетных записей должно помочь минимизировать возможный ущерб в случае успешного вторжения.

Для нужд тестирования вы можете использовать учетную запись nobody, но не используйте ее для развертывания служб на рабочих серверах. Используйте отдельные непривилегированные учетные записи.

Пример непривилегированной учетной записи

Ниже перечислены параметры типичной непривилегированной учетной записи:

Имя пользователя: присваивайте такое имя пользователя, которое явно указывало бы на выполняемые функции. Например, учетная запись для веб-сервера имеет имя www.

Домашний каталог: */nonexistent*

Командный процессор: */usr/sbin/nologin*

UID/GID: числовые значения идентификаторов пользователя и группы желательно выбирать из диапазона, специально отведенного для непривилегированных пользователей.

Полное имя: присваивайте такое имя, которое описывало бы функциональное назначение учетной записи.

Пароль: с помощью chpass(1) назначьте пользователю пароль, состоящий из одного символа звездочки. Это запретит использование пароля учетной записи.

Эти параметры сделают непривилегированную учетную запись действительно непривилегированной. С помощью утилиты adduser(8) вы легко сможете создать учетную запись без пароля, с нужным домашним каталогом и соответствующим командным интерпретатором.

Управление трафиком

Системный администратор должен уметь управлять входящим и исходящим трафиком, чтобы не подпускать к системе незваных гостей. FreeBSD предоставляет множество инструментов, позволяющих управлять доступом к системе, осуществляемым из внешнего мира. Здесь внимание будет сосредоточено на TCP-wrappers и фильтрации пакетов – двух популярных методов управления доступом.

Программа TCP Wrappers управляет доступом к сетевым демонам. Серверные программы должны быть написаны с поддержкой TCP Wrappers, и большинство программного обеспечения обладают такой поддержкой уже в течение многих лет. Конфигурировать TCP Wrappers довольно просто и для этого не нужны глубокие знания сетей.

Средства фильтрации пакетов регулируют, какие пакеты система будет принимать, а какие отклонять. Большинство брандмауэров и пакетных фильтров имеют неплохой графический интерфейс с пользователем, но также вы можете использовать фильтр пакетов FreeBSD и программное обеспечение прокси-сервера для построения надежной защиты сети. Отклоненный запрос на соединение не доходит до программ, запущенных на сервере; он отклоняется на низком уровне сетевого стека. Средства фильтрации пакетов могут управлять трафиком к любой программе, сервису или сетевому порту, но требуют хорошего знания сетей.

В любом случае перед реализацией управления трафиком необходимо выбрать одну из двух стратегий: принимать все пакеты по умолчанию (accept) или отвергать их (deny).

По умолчанию принимать или отвергать?

Один из важнейших вопросов в любой системе защиты – принимать ли все пакеты по умолчанию или же отвергать их. *Принятие по умолчанию* означает разрешение соединений любого типа, за исключением тех, которые запрещены явным образом. *Отвергать по умолчанию* значит разрешать соединения лишь с указанных адресов Интернета, а все другие соединения запретить. Действие по умолчанию выполняется всегда, если явно не определено правило, диктующее иную реакцию. Выбрав стратегию по умолчанию, можно регулировать доступ к системе, ограждая или открывая те или иные сервисы. На самом деле выбор между принятием или запрещением по умолчанию – это выбор между предоставлением сервисов всему миру или же немногим избранным.

Например, политика компании может диктовать условия, при которых корпоративный веб-сервер должен быть доступен только для пользователей корпоративной сети. Такая стратегия означает запрет по умолчанию и явное указание тех, кто может устанавливать соединение с системой. Альтернативный подход – открыть систему для всех, за исключением некоторых сегментов Интернета. Такая стратегия означает принятие по умолчанию. Я всегда рекомендую использовать политику запрещения доступа по умолчанию. Если вы еще не определились в выборе – выбирайте политику разрешения доступа по умолчанию.

Следует отметить, что выбор той или иной стратегии не означает, что все сервисы в системе должны следовать установке по умолчанию. Для своих общедоступных веб-серверов я выбираю политику запрета по умолчанию и явным образом разрешаю всеобщий доступ к веб-сайтам. Запросы на установление соединения с другими программами, расположенными на этой машине, отклоняются, если они не исходят с нескольких специально указанных IP-адресов. Это совершенно приемлемая стратегия запрета по умолчанию.

Различные средства обеспечения безопасности реализуют эти политики различными способами. Например, при использовании программы TCP Wrappers применяется *первое* совпавшее правило. Если последнее правило отвергает все попытки соединения, то тем самым вы применили политику, которая гласит: «Запретить любые соединения, если выше отсутствует правило, которое разрешает прохождение этого трафика». С другой стороны, в пакетном фильтре используется логика *последнего* совпавшего правила. Если первое правило блокирует все виды трафика, то тем самым вы применили политику, которая гласит: «Запретить любые соединения, если ниже отсутствует правило, которое разрешает прохождение этого трафика».

Обе эти политики осложняют жизнь системного администратора. Если по умолчанию вы разрешаете все соединения, то вам постоянно придется тратить время на затыкание дыр. Если по умолчанию вы запрещаете все соединения, то вам придется тратить время на разрешение доступа

тем, кому это необходимо. При использовании политики запрета по умолчанию вам постоянно придется говорить фразу: «Я только что открыл вам доступ к службе. Приношу свои извинения за доставленные неудобства». При использовании политики разрешения по умолчанию вам постоянно придется говорить: «...именно поэтому злоумышленники смогли проникнуть в нашу базу данных, из-за чего компания потеряла миллионы долларов». В последнем случае вам едва ли удастся отделаться фразой: «Приношу свои извинения за доставленные неудобства».

TCP Wrappers

В главе 6 говорилось, что сетевые соединения устанавливаются различными программами, ожидающими запросов на соединение. Когда программа собрана с поддержкой TCP Wrappers, она проверяет полученный запрос в соответствии с правилами TCP Wrappers. Если правила требуют отвергнуть запрос, программа немедленно его отвергает. Вопреки своему названию, TCP Wrappers работает как с сетевыми соединениями TCP, так и с соединениями UDP. TCP Wrappers – это установившийся стандарт UNIX, который был внедрен в FreeBSD. Тем не менее отдельные программы могут и не работать с TCP Wrappers. Почти все программы, составляющие основную часть FreeBSD, работают с TCP Wrappers, однако некоторые программы сторонних разработчиков – нет.

Программное обеспечение TCP Wrappers реализовано в виде разделяемой библиотеки с именем *libwrap*. Как будет показано в главе 12, разделяемая библиотека – это библиотека программного кода, который может совместно использоваться несколькими программами. Любые программы, скомпилированные с поддержкой libwrap, могут использовать функции TCP Wrappers.

Наиболее часто Wrappers применяются для защиты inetd – программы, которая запускает меньшие демоны. Программа inetd(8) обсуждается в главе 15. Хотя в примерах этой главы TCP Wrappers будут применяться для защиты программ inetd(8), этим же способом можно защитить любую другую программу. Несмотря на наличие защиты для inetd(8), вам нужно убедиться, что inetd(8) не предлагает ненужных служб, как вы делаете то же самое для всей системы в целом.

Конфигурирование Wrappers

TCP Wrappers проверяют каждый входящий запрос на соединение, последовательно сверяясь с правилами в */etc/hosts.allow*. Как только найдено первое совпавшее правило, процесс немедленно прекращается. Значит, порядок правил очень важен. Каждое правило занимает отдельную строку. Она состоит из трех частей, разделенных двоеточиями: имени демона, списка клиентов и списка параметров. Вот простой пример такой строки:

```
ftpd : all : deny
```

В этом примере имя демона – ftpd, а список клиентов – `all`, что означает все хосты. Наконец, параметр `deny` означает «отвергать все соединения». Никто не сможет подключиться к серверу FTP на этом хосте, если более раннее правило не предоставило такой доступ.

В предыдущих примерах мы ссылались только на два параметра: `accept` и `deny`. Соответственно, они разрешают или запрещают соединения. Есть много других параметров, которые будут рассмотрены позже.

Имя демона

Имя демона – это название программы, набираемое в командной строке. Например, inetd(8) запускает программу ftpd(8), когда получает входящий FTP-запрос. Веб-сервер Apache запускает программу, называемую httpd. Значит, если ваша версия Apache поддерживает wrappers, используйте имя демона `httpd`. (Обратите внимание: inetd(8) не используется для запуска сервера Apache, тем не менее он может обладать поддержкой TCP Wrappers.) Специальное имя демона `ALL` соответствует всем демонам, которые поддерживают wrappers.

Если за системой закреплено несколько IP-адресов, можно указать разные правила для каждого IP-адреса, за которые отвечает демон. В этом случае IP-адрес представляет собой часть имени демона:

```
ftpd@192.168.1.1 : ALL : deny
ftpd@192.168.1.2 : ALL : accept
```

В этом примере есть два имени демона: `ftpd@192.168.1.1` и `ftpd@192.168.1.2`. Для каждого из них есть отдельное правило TCP Wrappers.

Список клиентов

Список клиентов – это список разделенных пробелами конкретных IP-адресов, блоков сетевых адресов, имен хостов, имен доменов и ключевых слов. Имена хостов и IP-адреса просты, и их надо лишь перечислить.

```
ALL: netmanager.absolutefreebsd.com 192.168.1.5 : allow
```

При наличии такого правила в самом начале файла */etc/hosts.allow* TCP Wrappers позволит подключаться к любым службам системы моему компьютеру с именем netmanager и любому другому хосту с IP-адресом 192.168.1.5. (При этом я мог бы заблокировать доступ другими средствами.)

Номер сети в списке клиентов представляет собой IP-адрес и сетевую маску, разделенные символом слэша, как обсуждалось в главе 6. Например, если малыши со скриптами атакуют с группы различных адресов, которые начинаются с 216.136.204, их можно блокировать так:

```
ALL: 216.136.204.0/255.255.255.0 : deny
```

В шаблоне клиента можно также указывать имена доменов, предваряя их точкой. При таком способе используется обратное разрешение

имен DNS, а это значит, что любой, кто контролирует сервер DNS, ответственный за данный блок адресов, сможет обойти это ограничение.

```
ALL : .mycompany.com : allow
```

Если список клиентов достаточно длинный, то его можно хранить в файле и прописывать к нему путь в поле клиента в */etc/hosts.allow*. Исходя из моего опыта, в сети ISP возможно большое количество хостов с произвольно разбросанными адресами, а в корпоративной сетевой среде рабочие станции, управляющие сетью, могут быть разбросаны по всему миру. Каждая рабочая станция, как и любая другая рабочая станция, использовала одно и то же правило TCP Wrappers и занимала полдюжины строк в */etc/hosts.allow*. Поддерживая список всех рабочих станций в одном файле, я смог централизовать все изменения.

Помимо явно указанных адресов и имен клиентов можно использовать специальные ключевые слова, позволяющие добавить в список группу клиентов. Список ключевых слов и их описания приводятся в табл. 9.1.

Таблица 9.1. Ключевые слова TCP Wrappers

Ключевое слово	Назначение
ALL	Под это ключевое слово подпадает любой возможный хост.
LOCAL	Под это ключевое слово подпадает любой хост, чье имя не содержит точку. Обычно это машины в локальном домене. Даже компьютеры, находящиеся на противоположном конце земного шара, но использующие то же имя домена, согласно этому правилу считаются «локальными».
UNKNOWN	Под это слово подпадают машины с неопределимыми именами хостов или пользователей. Как правило, если машина устанавливает соединение IP, ее IP-адрес известен. Однако для определения имен хостов необходим DNS, а для отслеживания имен пользователей – identd(8). Этот параметр следует использовать очень внимательно, так как временные неполадки в DNS могут сделать неразрешимыми даже имена локальных хостов, а в большинстве хостов identd(8) не работает по умолчанию. Никто не хочет, чтобы машина стала недоступной только потому, что сервер имен неверно настроен, особенно если эта машина и есть сервер имен!
KNOWN	Под это ключевое слово подпадает любой хост с определимым именем и IP-адресом.
PARANOID	Под это ключевое слово подпадает любой хост, имя которого не соответствует его IP-адресу. Представим запрос на соединение с хоста с адресом 192.168.84.3, который выдает себя за *mail.absolutefreebsd.com*. Получив запрос, TCP Wrappers проверит, какой IP-адрес у *mail.absolutefreebsd.com*. Если TCP Wrappers получит IP-адрес, отличающийся от исходного IP-адреса, то хост подпадает под это правило. Системные администраторы, не имеющие времени на сопровождение своего сервера DNS, скорее всего не имеют времени на обновление своей системы и наложение «заплат».

Большинству ключевых слов, перечисленных в табл. 9.1, необходим работающий сервер DNS. Применяя эти ключевые слова, вы должны обеспечить высокую надежность службы DNS и должны помнить о существенной связи между DNS и другими программами. Если DNS не работает, то демоны, использующие wrappers и ключевые слова, не смогут распознать ни один хост. Это значит, что все хосты подпадут под правила UNKNOWN, которое обычно создается таким, что отвергает все запросы на соединение. Кроме того, недействующий DNS на стороне удаленных пользователей может сорвать их доступ к вашим серверам, поскольку ваши серверы DNS не смогут получить необходимую информацию с клиентских серверов DNS. Наконец, при активном использовании правил, построенных на базе DNS, злоумышленнику достаточно вызвать перегрузку сервера DNS, чтобы провести распространенную атаку под названием «отказ в обслуживании» (Denial of Service).

Другие ключевые слова доступны, но не так полезны или безопасны. Например, разрешать соединения можно на основе имени пользователя на удаленной машине, от которой исходит запрос. Однако не следует полагаться на имя пользователя или сетевое имя удаленной системы. Если настройки TCP Wrappers на моей домашней системе разрешают подключение лишь пользователю с именем «mwlucas», то при желании кто-нибудь легко добавит учетную запись с этим именем на своей машине FreeBSD. Кроме того, доступ по имени пользователя опирается на ранее упомянутый протокол identd, поддерживаемый небольшим количеством хостов. На странице руководства hosts_access(5) можно найти другие малоизвестные и такие же бесполезные параметры.

Ключевые слова ALL и ALL EXCEPT

Ключевые слова ALL и ALL EXCEPT можно использовать как для имени демона, так и для списка клиентов. Под ключевое слово ALL подпадает абсолютно все. Например, по умолчанию файл */etc/hosts.allow* начинается с правила, разрешающего все подключения, с любых адресов и к любому демону:

```
ALL : ALL: accept
```

Доступ разрешен ко всем программам от всех клиентов. Его можно ограничить, если конкретизировать список клиентов или список демонов:

```
ALL : 192.168.1.87 : deny
```

В этом примере отвергаются все подключения с хоста 192.168.1.87.

Категорически заблокировать доступ от всех хостов – не самая лучшая идея. Однако следует помнить, что программа TCP Wrappers читает правила последовательно и останавливается на первом подходящем правиле. Ключевое слово ALL позволяет легко установить запрет или разрешение по умолчанию. Рассмотрим следующий набор правил:

```
ALL : 192.168.8.3 192.168.8.4 : accept
```

```
ftpd : ALL : accept
ALL : ALL : deny
```

В этом примере рабочим станциям 192.168.8.3 и 192.168.8.4 (возможно, это рабочие станции системных администраторов) разрешен доступ ко всем демонам. Далее, любой желающий может подключиться к сервису FTP на этой машине. Наконец, все остальные соединения запрещены. Это удобно при выборе стратегии запрета по умолчанию.

Ключевое слово `ALL EXCEPT` применяется для еще большего сокращения предыдущего набора правил. `ALL EXCEPT` позволяет перечислять хосты как исключения; под правило подпадает то, что не перечислено в списке. Рассмотрим тот же набор правил, переписанный с применением `ALL EXCEPT`:

```
ALL : 192.168.8.3 192.168.8.4 : accept
ALL EXCEPT ftpd : ALL : deny
```

Разумеется, это правило основано на стратегии запрета по умолчанию, которая допускает возможность соединения с сервером FTP.

Одни администраторы считают более понятными правила с `ALL`, другие – с `ALL EXCEPT`. Важно помнить, что первое подходящее правило останавливает сличение, поэтому применение правила `ALL` требует осторожности.

Если вы лишь приступаете к изучению TCP Wrappers, лучше всего будет разрешить любые соединения с локального хоста – вы наверняка обнаружите массу программ, которые не в состоянии работать в отсутствие возможности подключаться к локальному компьютеру. Добавьте в начало файла hosts.allow следующее правило:

```
ALL : localhost : allow
```

Параметры

Мы уже видели в действии два параметра: `allow` и `deny`. Параметр `allow` предписывает разрешить соединение, `deny` – блокировать. По умолчанию файл *hosts.allow* начинается с правила, которое соответствует всем демонам и всем клиентам и разрешает все возможные соединения. Если необходимо защитить сервисы, то это правило не должно быть первым в списке. В то же время оно хорошо подойдет в качестве последнего правила при использовании стратегии разрешения соединений

Длинные правила

При использовании большого числа параметров правила TCP Wrappers становятся чересчур длинными. Для повышения удобочитаемости в файле *hosts.allow* можно использовать символ обратного слэша (\) как знак переноса правила на следующую строку.

по умолчанию. Точно так же правило `ALL:ALL:deny` прекрасно подойдет в качестве последнего правила при использовании стратегии запрета соединений по умолчанию. Помимо простых `allow` и `deny`, TCP Wrappers поддерживают и другие параметры, увеличивая гибкость наборов правил.

Протоколирование

Как только сделан выбор между принятием и отклонением запроса на подключение, информацию о запросе можно зарегистрировать в протоколе. Предположим, вы хотите разрешить все запросы, исходящие от конкурента, но занести их в протокол. Точно так же может потребоваться знать о том, сколько запросов на соединение было отклонено от машин из-за неполадок в DNS, особенно если указано ключевое слово `PARANOID`. Протоколирование – это удобно. Чем больше объем протокола, тем лучше. Дисковое пространство стоит много дешевле, чем ваше время.

Параметр `severity` (строгость) посылает сообщение программе системного протоколирования (system log), syslogd(8). Программу syslogd можно настроить на переадресацию сообщений в произвольный файл, в зависимости от источника (facility) syslogd и выбранного уровня важности сообщения (глава 20):

```
telnetd: ALL: severity auth.info : allow
```

Это правило подразумевает протоколирование всех соединений telnet.

Возврат сообщений

Параметр `twist` позволяет выполнять произвольные команды интерпретатора (shell) и сценарии, когда кто-либо пытается установить соединение с защищаемым демоном TCP, и возвращает вывод команд удаленному пользователю. `twist` работает только с соединениями TCP. (Не забывайте, что UDP – это протокол без установления соединения; не существует соединения, по которому можно вернуть ответ. Значит, необходимо предпринять изощренные и трудоемкие усилия, чтобы `twist` мог работать с UDP. Кроме того, те программы, которые используют UDP, обычно не ожидают подобного ответа и не готовы его принять и интерпретировать. Применение `twist` для UDP не стоит возникающих трудностей.) `twist` принимает команду интерпретатора как аргумент и действует по правилу «отклонить и что-нибудь сделать». Для использования `twist` необходимо знать основы создания сценариев командного интерпретатора; возможны очень сложные варианты применения `twist`, но здесь будут представлены более простые.

Если используется запрет по умолчанию, то `twist` полезен для последнего правила. Параметр `twist` можно использовать для возврата ответа лицу, пытающемуся установить соединение. Например, так:

```
ALL : ALL : twist /bin/echo "Вы не можете использовать этот сервис."
```

Чтобы не предоставлять конкретный сервис тому или иному хосту, можно явно указать имя демона и список клиентов:

```
sendmail : .spammer.com : twist /bin/echo \
    "Как спамер, вы не можете использовать этот сервис."
```

В действительности это неэффективный способ борьбы со спамом, но очень наглядный пример. Кроме того, такое грубое сообщение может вызывать отрицательные эмоции у добропорядочных пользователей и побудить к дальнейшим поискам недобропорядочных пользователей.

Если вы настроены дружелюбно, можете оповещать людей о причине отклонения их запросов на соединения. Следующее правило с `twist` отклоняет все запросы на установление соединения, исходящие от хостов, чьи имена не соответствуют их IP-адресам, и объясняет причину отказа:

```
ALL : PARANOID : twist /bin/echo \
"Ваш DNS неработоспособен. Возвращайтесь, когда устраните неполадки."
```

`twist` удерживает сетевое соединение открытым, пока команда интерпретатора не закончит работу. Если она выполняется долго, система будет удерживать больше открытых соединений, чем планировалось. Это может значительно снизить производительность системы. Малыши со скриптами могут использовать правила с параметром `twist` для увеличения нагрузки на систему с целью проведения атаки «отказ в обслуживании». Правила с `twist` должны быть просты и выполняться быстро.

Обработка

Подобно `twist`, параметр `spawn` отклоняет запрос на соединение и запускает указанную команду интерпретатора. В отличие от `twist`, `spawn` не возвращает результаты клиенту. Параметр `spawn` следует применять, когда при получении запроса на соединение система FreeBSD должна выполнить те или иные команды, о которых не должен знать клиент. Эти команды выполняются в фоновом режиме, а их результаты не возвращаются клиенту. Правило из следующего примера разрешает соединение, но заносит IP-адрес клиента в файл протокола:

```
ALL : PARANOID : spawn (/bin/echo %a >> /var/log/misconfigured) \
    : allow
```

Минутку, откуда возьмется значение `%a` в этой команде? Программа TCP Wrappers поддерживает множество переменных, которые можно применять в командах `twist` и `spawn`. Эти переменные раскрываются перед выполнением команд, поэтому с их помощью можно легко видоизменять ответы на запросы. Переменная `%a` содержит адрес клиента. Будучи размещенной в команде, она преобразуется в IP-адрес до выполнения этой команды. Эта и другие переменные представлены в табл. 9.2.

Таблица 9.2. Переменные, которые можно использовать в командах twist и spawn

Ключевое слово	Назначение
%a	Адрес клиента
%A	IP-адрес сервера
%c	Вся доступная информация о клиенте
%d	Имя демона
%h	Имя хоста клиента (если доступно) или IP-адрес
%H	Имя хоста сервера (если доступно) или IP-адрес
%n	Имя хоста клиента; если оно не найдено, выдается UNKNOWN. Если имя хоста не соответствует IP-адресу, это эквивалентно PARANOID
%N	Имя хоста сервера; если оно не найдено, возвращается либо UNKNOWN, либо PARANOID
%p	Идентификатор процесса демона
%s	Вся доступная информация о сервере
%u	Имя пользователя клиента
%%	Символ %

Эти переменные можно применять везде, где предоставляемая ими информация используется в сценариях интерпретатора. Например, чтобы занести всю доступную информацию о клиенте в файл протокола при каждом подключении к защищаемой программе, можно использовать spawn:

```
ALL : ALL : spawn (/bin/echo %c >> /var/log/clients) : allow
```

Символы пробела и обратного слэша в имени хоста могут вызвать нарекания у командного интерпретатора, поскольку это недопустимые символы. При обычных обстоятельствах они не должны присутствовать в именах хостов, но Интернет – это по определению не обычный случай. Для обеспечения безопасности программа TCP Wrappers замещает все символы, которые могут вызвать сложности у командного интерпретатора, на символ подчеркивания (_). Проверьте наличие таких символов в файлах протоколов; они могут свидетельствовать о попытках проникновения в систему или о том, что кому-то нравится символ подчеркивания в именах хостов.

В заключение о TCP Wrappers

Возьмем все примеры, уже рассмотренные в этом разделе, и создадим файл */etc/host.allow* целиком, чтобы защитить гипотетическую систему, подключенную к сети. Для начала составим перечень сетевых ре-

сурсов, предлагаемых системой, а также список IP-адресов, закрепленных за системой, и пользователей, которым разрешено подключение:

- Диапазон IP-адресов – 192.168.0.0/16. Система предлагает POP3, ftpd(8), SSH и portmap(8).
- Диапазон IP-адресов конкурента, которому запрещен доступ к системе: 10.5.4.0/23. (Вообще говоря, блокирование конкурента и предоставление услуг всем остальным – это не самая эффективная тактика, поскольку в этом случае конкуренты просто пойдут в ближайшее интернет-кафе и воспользуются беспроводным подключением. Однако у вас могут быть и другие причины блокирования определенных адресов.)
- Примем отчасти параноидальное решение: хосты с неверной информацией DNS могут принадлежать взломщикам, поэтому запросы на соединение, исходящие от этих машин, будут отклоняться.
- С любого адреса Интернета можно запрашивать доступ к серверам FTP и SSH и POP3.
- Хосты из локальной сети могут пользоваться услугами демона portmap, а из других сетей – нет.
- Локальный хост может взаимодействовать сам с собой.
- Все, что не разрешено, – запрещено.

Хотя эти требования довольно сложны, они сводятся к очень простому набору правил:

```
# отклонять все запросы на соединение, исходящие от конкурента и от хостов
# с неверными записями DNS
ALL : PARANOID 10.5.4.0/23 : deny
# локальный хост может взаимодействовать сам с собой
ALL : localhost : allow
#разрешить использовать portmap в локальной сети
#и запретить для всех остальных.
portmap : ALL EXCEPT 192.168.0.0/16 : deny
# Разрешить доступ к SSH, pop3 и ftp и запретить все остальное
sshd, POP3, ftpd : ALL : allow
ALL : ALL : deny
```

Примеры с более подробными комментариями можно найти в файле */etc/hosts.allow* или на страницах руководства hosts_allow(5) и hosts_access(5).

Фильтрация пакетов

Инструменты FreeBSD для фильтрации пакетов на уровне ядра используются, когда необходимо управлять доступом к сетевым приложениям, не поддерживающим TCP Wrappers, или когда требуется нечто, превосходящее TCP Wrappers по своим возможностям. Если вам

потребовался фильтр пакетов, то лучше полностью заменить им реализацию TCP Wrappers. Применение двух инструментов на одном компьютере будет просто запутывать вас.

При фильтрации пакетов каждый сетевой пакет, поступающий в систему, сверяется со списком правил. Когда соответствующее правило найдено, ядро действует, исходя из этого правила. Правила могут пропустить, удалить или видоизменить пакет. Однако здесь нельзя применять замечательные параметры, разрешенные в TCP Wrappers; клиенту не отправляется сравнительно дружелюбное «сообщение о запрете» – вместо этого соединение разрывается на сетевом уровне еще до того, как пакеты будут доставлены приложению.

Идея фильтрации пакетов достаточно проста, но первая реализация этой стратегии может стать нелегким испытанием – как говорится, «ценным опытом». Будьте готовы потратить несколько часов на эксперименты с фильтрацией пакетов и не отчаивайтесь при неудачах. По своему опыту могу сказать, что источником разочарований является не пакетный фильтр, а незнание TCP/IP. Пытаться фильтровать трафик – занятие неблагодарное и бессмысленное. Единственный способ по-настоящему изучить протокол TCP/IP состоит в том, чтобы работать с ним. Если информации в главе 6 для вас оказалось недостаточно, приобретите книгу «The TCP/IP Guide» Чарльза М. Козиерока (Charles M. Kozierok) (No Starch Press, 2005).

Операционная система FreeBSD обладает богатым набором пакетных фильтров: IPFW, IP Filter и PF.

IPFW – это программное обеспечение фильтрации пакетов, изначально входившее в состав FreeBSD. Оно очень тесно интегрировано в операционную систему – файлы с говорящими названиями, */etc/rc.firewall* и */etc/rc.firewall6*, предназначены исключительно для нужд IPFW. Это очень мощное программное обеспечение, пользующееся большой популярностью у опытных системных администраторов FreeBSD, но для начинающих оно является довольно сложным.

Второй пакетный фильтр – IP Filter – не является программным обеспечением построения брандмауэров, разработанным исключительно для FreeBSD, но поддерживается некоторыми другими UNIX-подобными операционными системами. Этот фильтр разрабатывался в основном одним человеком – Дарреном Ридом (Darren Reed), который приложил героические усилия, написав большую часть программного кода и выполнив перенос на разные операционные системы. Однако IP Filter не получил широкой популярности.

Основное внимание мы будем уделять продукту с названием PF, или *packet filter* (*фильтр пакетов*). Впервые PF появился в OpenBSD и разрабатывался как очень мощный, гибкий и простой в использовании. Обычный администратор FreeBSD может, используя PF, добиться любых эффектов, которые можно получить с помощью двух других фильтров пакетов.

Примечание

Более подробную информацию о PF вы найдете в книге Питера Н. М. Ханстена (Peter N. M. Hansteen) «The Book of PF» (No Starch Press, 2007) или в моей книге «Absolute OpenBSD» (No Starch Press, 2003), в которую включено несколько глав о PF. Можно также заглянуть в сборник часто задаваемых вопросов PF FAQ в Интернете, но там вы не найдете простых ответов.

Активизация PF

Фильтр пакетов PF состоит из модуля ядра *pf.ko* и программы, работающей в пространстве пользователя, – pfctl(8). Прежде чем можно будет начать использовать PF, необходимо загрузить модуль ядра. Проще всего это сделать, поместив в *rc.conf* следующую строку:

```
pf_enable="YES"
```

По умолчанию фильтр PF пропускает все пакеты; это означает, что простая активизация брандмауэра не обеспечивает никакой защиты.

Фильтрация пакетов: принимать по умолчанию или отвергать

Стратегии принятия и запрета по умолчанию чрезвычайно важны в фильтрации пакетов. Если выбрано принятие по умолчанию и необходимо защитить систему или сеть, то для блокирования всех возможных атак потребуется множество правил. При выборе запрета по умолчанию необходимо явно открыть «лазейки» для каждого предлагаемого сервиса. В большинстве случаев предпочтительнее использовать стратегию запрета по умолчанию. Хотя при такой стратегии управлять фильтром значительно сложнее, тем не менее повышенный уровень безопасности с лихвой окупит все трудозатраты.

При запрете по умолчанию очень легко полностью заблокировать удаленный доступ к системе даже для самого себя. Если у вас имеется SSH-соединение с удаленной системой, и вы по ошибке нарушите правило, разрешающее доступ по SSH, у вас появятся проблемы. Такое случается со всеми, хотя бы один раз, поэтому особо не смущайтесь, если это произойдет и с вами. Однако будет лучше, если вы начнете изучать принципы фильтрации пакетов не на удаленной машине – начните с машины, к которой у вас имеется физический доступ, чтобы проще было исправить свою ошибку. Я много раз закрывал доступ самому себе – обычно, когда не утруждал себя размышлениями при устранении неполадок, связанных с фильтрацией пакетов. Единственное решение – проклиная себя, лезть в машину, ехать на удаленную точку, долго извиняться перед людьми, которым я причинил неудобство, и на ходу устранять проблему. К счастью, с возрастом такое случается все реже и реже.[1]

[1] Зато я научился искусно водить автомобиль и извиняться.

Тем не менее в большинстве случаев стратегия запрета по умолчанию будет верной. Начинающий администратор может хорошо изучить фильтрацию пакетов лишь при наличии удобного доступа к системной консоли. Если нет полной уверенности в проведенной настройке, то посылать систему фильтрации пакетов через всю страну можно лишь при наличии компетентного местного администратора или удаленной консоли.

Основы фильтрации пакетов и контроль за состоянием соединения

Вспомним из главы 6, что соединение TCP может пребывать в различных состояниях. Оно может быть открытым, открываться, закрываться и т. д. Например, пытаясь открыть соединение, клиент посылает серверу пакет SYN, запрашивая синхронизацию. Если сервер ожидает получение запросов на соединение, в ответ он посылает клиенту пакет SYN-ACK, что означает следующее: «Я получил ваш запрос на соединение. Вот базовая информация для установления соединения». Клиент подтверждает прием информации, отвечая пакетом ACK, что означает: «Я получил и подтверждаю вашу информацию о соединении». Каждая часть процесса «тройного рукопожатия» должна быть выполнена, чтобы соединение было действительно установлено. Правила фильтрации пакетов должны разрешать каждую часть «тройного рукопожатия», а также саму передачу данных. Разрешение на получение входящих запросов соединений бесполезно, если правила фильтрации пакетов не позволяют отправить обратно уведомление.

В 1990-х годах фильтры пакетов проверяли каждый пакет по отдельности. Если пакет отвечал правилу, он проходил дальше. Система не анализировала предыдущие пакеты и не могла определить, был ли тот или иной пакет частью легитимной транзакции или нет. Например, если пакет SYN-ACK, привязанный к адресу «изнутри» фильтра пакетов, прибывал к нему «снаружи», фильтр пакетов решал, что этот пакет должен быть ответом на пакет, одобренный ранее. Для завершения «тройного рукопожатия» такой пакет *необходимо* было принять. В результате злоумышлениники могли подделывать пакеты SYN-ACK и использовать их для обхода казавшихся надежными устройств. Поскольку фильтру пакетов не был известен отправитель предыдущего пакета SYN, он не мог отклонить такие подложные пакеты SYN-ACK. Когда пакеты, отправленные злоумышленниками, проникали в сеть, они инициировали ответ от того или иного устройства и выведывали информацию о системе.

Контроль состояния, введенный в большинстве современных пакетных фильтров, призван противодействовать таким атакам. *Контроль состояния* – это сохранение информации о каждом соединении и его текущем состоянии. Если входящий пакет SYN-ACK представляется частью действующего соединения, но никто не посылал соответствующий запрос SYN, он отклоняется. Хотя это усложняет работу ядра, на-

писать правила фильтрации пакетов для stateful inspection на самом деле легче. Фильтр пакетов должен отслеживать множество других возможных состояний, то есть это труднее, чем может показаться; особенно, если добавить контроль за фрагментацией пакетов, противодействие мистификации адресов и т. п.

Те, кто подумал: «А-а, фильтрация пакетов – это все равно, что брандмауэр», по сути правы. Слово *брандмауэр* применимо к множеству устройств, предназначенных для защиты сети. Некоторые из них довольно замысловаты, некоторые по уровню своего интеллекта сравнимы с бетонной стеной. Сейчас слово брандмауэр превратилось в модное словечко из лексикона маркетоидов, не имеющее точного значения. Оно подобно слову *автомобиль*. Имеете ли вы в виду Gremlin 1972 года с двигателем в шесть лошадиных сил и по количеству выхлопных газов не соответствующий Киотскому договору, или сверкающий Chevy SSR 2005 года, причудливой трехцветной раскраски, оснащенный пятисотсильным двигателем и стереосистемой Apocalypse? Оба автомобиля нашли свою нишу, но один из них, очевидно, эффективнее. Хотя Gremlin'ы среди брандмауэров находят свою нишу, лучше приобрести что-нибудь получше.

Итак, система FreeBSD может стать таким крепким брандмауэром, каким ее желает сделать администратор. Фильтрация пакетов – это только начало, в каталогах */usr/ports/net* и */usr/ports/security* можно найти множество приложений-представителей (proxy), которые позволят вашей системе FreeBSD стать вровень с Checkpoint или PIX и даже победить в состязании, затратив на десятки тысяч долларов меньше.

Конфигурирование PF

Параметры настройки PF хранятся в файле */etc/pf.conf*. В этом файле содержатся определения и правила, формат которых меняется в зависимости от конфигурируемых функций. Порядок следования правил имеет чрезвычайно важное значение, но не менее важен и порядок настройки функциональных особенностей. Если, например, попытаться реализовать контроль состояния до сборки фрагментированных пакетов, то соединения не будут устанавливаться должным образом.

Файл по умолчанию */etc/pf.conf* содержит примеры правил, расположенные в нужном порядке, но если вы боитесь запутаться в правилах, я рекомендую в нужных местах отделять разделы комментариями, набранными заглавными буквами. (Комментарии в *pf.conf* начинаются с символа решетки (#).) Функциональные конструкции конфигурации должны вводиться в следующем порядке:

- Макроопределения
- Таблицы
- Параметры
- Нормализация пакетов

- Управление полосой пропускания
- Преобразование адресов
- Перенаправление
- Фильтрация пакетов

Да, PF – это больше, чем просто фильтр пакетов. Это многоцелевой инструмент манипулирования протоколом TCP/IP. Мы не будем рассматривать все его возможности, потому что это тема отдельной книги.

Макроопределения

Макроопределения позволяют определять переменные, которые упрощают создание и чтение правил. Например, ниже приводятся макроопределения, которые определяют сетевой интерфейс и его IP-адрес:

```
interface="fxp0"
serveraddr="192.168.1.2"
```

Потом в правилах можно будет ссылаться на сетевой интерфейс по имени `$interface`, а на IP-адрес – по имени `$serveraddr`. Благодаря этому в случае смены IP-адреса сервера или замены сетевой карты достаточно будет внести всего одно изменение в *pf.conf*, чтобы приспособить правила под новые условия.

Таблицы и параметры

Фильтр пакетов PF может хранить в таблицах длинные списки адресов. Мы не будем использовать эту возможность, но вы должны знать о ее существовании.

Точно так же в PF имеются различные параметры, которые позволяют управлять синхронизацией сетевых соединений, размерами таблиц и другими внутренними настройками. Значения параметров по умолчанию вполне адекватны для использования в большинстве нормальных (и ненормальных) ситуаций.

Нормализация пакетов

Пакеты TCP/IP по пути следования могут быть разбиты на фрагменты, что повышает нагрузку на систему и увеличивает объем работы, которую придется выполнять серверу, чтобы иметь возможность обслуживать запросы и фильтровать пакеты. Прежде чем передать полученные данные клиентской программе, система должна собрать эти фрагменты, попутно решая, что делать с любыми другими случайно пришедшими частями. Сборка фрагментов в PF называется *очисткой* (*scrubbing*). Например, следующее правило выполняет сборку всех фрагментов, полученных сетевым интерфейсом, отвергает все фрагменты, слишком маленькие, чтобы считаться легитимными, и выполняет другие функции обработки потока входных данных:

```
scrub in all on $interface
```

Это правило будет воздействовать на все пакеты, поступающие в компьютер.

Несмотря на то, что очистка кажется хотя и удобной, но маловажной функцией, в действительности она имеет большое значение, так как фильтры PF могут работать только с полными пакетами. В отсутствие функции сборки работа с фрагментами становится чрезвычайно сложной и требует специальной обработки. Без очистки трафика будут наблюдаться проблемы обеспечения связи.

Полоса пропускания, преобразование адресов и перенаправление

Принципы управления полосой пропускания будут обсуждаться немного ниже, а пока лишь замечу, что PF может управлять объемом трафика, пропускаемого к определенному порту или IP-адресу.

Функции преобразования сетевых адресов (Network Address Translation, NAT) и перенаправления портов являются критически важными частями брандмауэра. Фильтр пакетов PF обладает множеством функций, обеспечивающих поддержку NAT и перенаправление, которые, впрочем, мы не будем рассматривать, так как перед нами не стоит задача построения брандмауэра.

Правила фильтрации трафика

Теперь рассмотрим то, что нам действительно необходимо. Правила фильтрации трафика в общем случае имеют следующий формат:

```
❶pass ❷out ❸on $interface proto ❹{ tcp, udp } all ❺keep state
```

Первым в строке стоит ключевое слово ❶, которое определяет тип правила. Для каждого типа правила имеется свое ключевое слово. В данном случае – это правило фильтрации. Далее указывается, в каком направлении ❷ движутся пакеты. Кроме того, правила PF содержат наименование интерфейса ❸, к которому применяется это правило. Остальное содержимое правила зависит от его типа. В данном случае под действие правила подпадают только те пакеты, которые покидают систему через интерфейс, заданный макроопределением `$interface`, и только если они соответствуют остальным условиям.

Далее определяется тип трафика, соответствующий правилу. Тип трафика определяется по протоколу, номеру порта или флагам TCP/IP ❹. Данному простому правилу соответствует весь трафик, который передается с использованием протоколов TCP и UDP. Далее говорится, что PF должен использовать механизм контроля состояния соединения, разрешая прохождение дальнейшего трафика, являющегося частью этого соединения.

Если говорить в целом, то это правило пропускает весь трафик TCP и UDP, покидающий систему. Данный сервер может открывать любые исходящие соединения, что весьма типично для сервера Интернета.

Обработка входящих соединений выполняется немного сложнее. Следующее правило разрешает доступ к веб-серверу извне:

```
pass in on $interface proto ❶tcp from ❷any to ❸($interface) ❹port 80
❺flags S/SA keep state
```

Благодаря предыдущему примеру многое здесь вы сможете понять сами. Это правило пропускает входящий трафик при условии, что это трафик TCP ❶. Допустимым считается трафик, пришедший из любого источника ❷, входящий через указанный сетевой интерфейс ❸. (Круглые скобки, окружающие имя интерфейса, означают: «независимо от IP-адреса, присвоенного интерфейсу», что очень удобно при использовании DHCP.) Далее уточняется, что пропускаться должен только тот трафик, который идет на порт с номером 80 ❹.

Единственное сложное место в этом правиле – это ключевое слово `flags` ❺. Первый символ, стоящий перед символом слэша, указывает, какой флаг TCP должен быть установлен в пакете, а символы, стоящие за символом слэша, указывают флаги, которые должны проверяться. Символ `S` обозначает SYN, а символ `A` – ACK. Это означает: «Из флагов SYN и ACK должен быть установлен только флаг SYN». Каким бы сложным не выглядело это утверждение, в действительности оно означает лишь: «Данному правилу соответствуют только входящие запросы на открытие соединения».

Дополнительное выражение `keep state`, находящееся в конце правила, предписывает фильтру PF отслеживать это соединение и пропускать весь остальной трафик, принадлежащий этому соединению.

Если говорить в целом, то это правило гласит: «Разрешить открытие входящих соединений по протоколу TCP с портом 80». Порт с номером 80 обычно используется веб-серверами.

> **С флагами или без флагов?**
>
> В версии FreeBSD 6.0 и более ранних ключевое слово `flags` было обязательным. В версии FreeBSD 7.0 оно подразумевается по умолчанию. Если у вас возникают сомнения, включайте ключевое слово `flags` в правила.

Пример полного правила PF

Ниже приводится набор правил PF, реализующих защиту небольшого сервера Интернета. Вы можете принять этот набор за основу и приспособить его под свои требования.

```
interface="em0"
scrub in all
❶ block in on $interface
```

```
❷ # разрешить входящий трафик SSH и POP3 из локальной сети
pass in on $interface proto tcp from 192.168.1.0/24 to $interface port 22
pass in on $interface proto tcp from 192.168.1.0/24 to $interface port 110
❸ # разрешить входящий трафик SMTP (25), HTTP (80) и HTTPS (443)
pass in on $interface proto tcp from any to $interface port 25
pass in on $interface proto tcp from any to $interface port 80
pass in on $interface proto tcp from any to $interface port 443
❹ # разрешить входящие запросы к серверу DNS
pass in on $interface proto tcp from any to $interface port 53
pass in on $interface proto udp from any to $interface port 53
❺ # разрешить исходящий трафик
pass out on $interface proto { tcp, udp } all
```

Набор правил начинается с макроопределения, которое задает имя сетевого интерфейса, благодаря чему после замены сетевой карты нам не придется переписывать все правила в наборе. Далее производится очистка входящего трафика. Оба эти правила были взяты непосредственно из предыдущих примеров.

Первое интересное правило, которое встречается в данном наборе, – это установка стратегии запрета по умолчанию с помощью оператора `block` ❶. Все, что явно не разрешено, – запрещено.

Следующие два правила пропускают трафик по определенным протоколам, исходящий с определенных IP-адресов ❷. Благодаря этим правилам имеется возможность соединяться с сервером по протоколам POP3 и SSH из сети 192.168.1.0/24, которая, скорее всего, является локальной офисной или административной сетью.

Для всеобщего пользования предлагаются службы электронной почты, веб-сервера и HTTPS ❸. Доступ к этим службам обеспечивается с помощью правил, которые уже встречались в предыдущих примерах, но с другими номерами портов. Кроме того, всем желающим разрешается доступ к серверу DNS ❹, но для этой цели используются несколько иные правила. Служба DNS работает как с протоколом TCP, так и с протоколом UDP. Такого рода знания можно получить только из толстых книг, посвященных описанию протоколов TCP/IP, или в архивах почтовых рассылок, попутно вырывая волосы на голове[1] при попытках заставить эти правила работать. Как могли бы заметить ярые сторонники PF, эти правила можно ужать и оптимизировать, но для небольшого сервера вполне пригоден и такой стиль. Наконец, сервер может инициировать исходящие соединения TCP и UDP с любыми портами ❺.

Такая простая стратегия определяет основные правила, обеспечивающие возможность взаимодействий с нашим сервером. Несмотря на несовершенство, этот набор может стать серьезной преградой для злоумышленников. Предположим, что кто-то смог взломать веб-сервер и получить доступ к командной строке с привилегиями суперпользователя root через порт с номером 10000. Однако все его усилия пойдут

[1] Именно по этой причине я рекомендую делать очень короткую стрижку.

прахом, так как брандмауэр не пропускает входящие соединения с этим портом.

Активизация правил PF

Управление фильтром пакетов PF производится с помощью pfctl(8). Если правила не содержат ошибок, pfctl(8) не выводит никаких сообщений – эта утилита осуществляет вывод на экран только при наличии ошибок. Так как ошибки в брандмауэре могут причинить немало бед, лучше тщательно проверить их перед активизацией. Правила проверяются только на наличие грамматических ошибок, так как активизация правил с грамматическими ошибками либо оставит систему незащищенной, либо заблокирует доступ к ней. Для проверки файла с правилами на наличие проблем используется ключ `-n`, чтобы указать имя файла, используется ключ `-f`.

```
# pfctl -nf /etc/pf.conf
```

Если были обнаружены ошибки – исправьте их и повторите попытку. Если программа не вывела никаких сообщений, можно активизировать набор правил, убрав ключ `-n`.

```
# pfctl -f /etc/pf.conf
```

Изменение настроек PF выполняется очень быстро, таким образом, у вас могут быть несколько наборов правил для различных ситуаций. Например, вы можете разрешать доступ к определенным службам только в определенное время суток, для чего достаточно запланировать запуск pfctl для активизации соответствующего набора правил в заданный момент времени. У вас также может иметься набор правил, специально предназначенный для установки на период ликвидации аварийных ситуаций, или специальные правила на случай потери соединения с Интернетом. Реализовать все это с помощью pfctl(8) не составляет большого труда.

При желании, использовав команду `pfctl -sr`, можно увидеть текущий набор правил.

```
# pfctl -sr
❶ No ALTQ support in kernel
ALTQ related functions disabled
scrub in all ❷fragment reassemble
block drop in on ❸em0 all
pass in on wi0 inet proto tcp from 192.168.1.0/24 to 192.168.1.201 port = ssh
flags S/SA keep state
...
```

На запуске PF сообщает о недоступности чего-то, что называется ALTQ ❶. Что это такое, вы узнаете в следующем разделе. Затем выводятся правила с дополнительными параметрами, используемыми по умолчанию ❷. При выводе правил на место макроопределений pfctl(8) подставляет истинные значения ❸, которые были указаны в конфигура-

ции, затем утилита подтверждает, что эти правила используются уже несколько недель или месяцев с момента, когда они были загружены.

Наконец, чтобы удалить все правила, используются ключи `-Fa` (flush all – очистить все).

```
# pfctl -Fa
```

Вам время от времени придется наблюдать, как PF очищает все правила, настройки NAT и все остальное. Не требуется очищать наборы правил перед загрузкой новых наборов – достаточно просто загрузить новые правила, и старые автоматически будут удалены.

Фильтр пакетов PF обладает очень широкими возможностями и может нарушить работу TCP/IP любыми способами, которые вам понравятся (и некоторыми, которые вам точно не понравятся). Мы лишь слегка коснулись того, что лежит на поверхности. Для получения более глубоких и более подробных сведений обращайтесь к источникам информации, перечисленным на стр. 349.

Шифрование открытым ключом

Многие функции обеспечения безопасности, реализованные в серверных демонах, опираются на шифрование открытым ключом. Оно обеспечивает конфиденциальность, целостность и достоверность данных. Различные сервисы Интернета также используют такое шифрование. Знание его основ поможет организовать защищенные веб-страницы (https) и защищенную почту POP3 (pop3ssl). Если вы уже знакомы с шифрованием открытым ключом, можете просто пропустить этот раздел. Если нет – приготовьтесь к весьма сжатому введению в эту тему.

В системах шифрования ключ применяется для преобразования удобочитаемых сообщений с открытым текстом (cleartext) в версию с зашифрованным текстом (ciphertext), а также для обратного преобразования. Хотя в терминах *открытый текст* и *зашифрованный текст* есть слово *текст*, сообщения не ограничены только текстом. В них могут быть графические, исполняемые файлы и любые другие пересылаемые данные.

У каждой криптографической системы есть три основных назначения: поддерживать целостность и конфиденциальность данных и обеспечивать невозможность отказа от авторства. *Целостность* означает, что сообщение не будет искажено. *Конфиденциальность* подразумевает, что сообщение сможет прочесть только предопределенный круг лиц. *Невозможность отказа от авторства* означает, что автору не удастся впоследствии заявить, будто он не писал того или иного сообщения.

Старые системы шифрования предполагали наличие единого ключа для шифрования и дешифрования сообщений. Для преобразования сообщения требовалось выполнить много операций, как в случае с системой Enigma, применявшейся коалицией союзников во время второй

мировой войны. Как бы то ни было, единый ключ сделал шифрование возможным. Типичным примером является любой код, для которого необходим ключ или пароль. Планшеты для создания разовых сообщений, которые стали известны благодаря фильмам про шпионов, служат примером первых систем шифрования одним ключом.

В отличие от систем шифрования одним ключом, в системах шифрования открытым ключом (несимметричное шифрование) применяются два ключа: секретный (private) и открытый (public). Сообщения шифруются одним ключом, а расшифровываются другим. Математическое объяснение довольно сложно, чтобы приводить его здесь, однако такая схема работоспособна. Система основана на свойствах очень больших чисел. Владелец ключа держит секретный ключ в тайне, а открытый ключ раздает направо и налево. Владелец ключа применяет секретный ключ, а все остальные – открытый. Владелец ключа может шифровать сообщения, которые может открыть любой из указанных им субъектов. Однако сообщения, зашифрованные открытым ключом, может прочесть только владелец ключа.

Криптография с открытым ключом прекрасно удовлетворяет наши потребности в целостности, конфиденциальности и невозможности отказа от авторства. Если автор хочет, чтобы его сообщение смог прочесть любой желающий, но никто не смог его исказить, он зашифровывает сообщение своим секретным ключом. Все, у кого есть его открытый ключ, смогут дешифровать и прочесть это сообщение. Искажение зашифрованного сообщения сделает его нечитаемым.

Такое шифрование сообщений гарантирует, что у автора есть секретный ключ. Если автор хочет послать сообщение, которое разрешено прочесть только указанному получателю, он сможет зашифровать текст открытым ключом получателя. И только получатель, у которого есть соответствующий секретный ключ, сможет прочесть это сообщение.

Описанная система работает отлично, пока секретный ключ остается секретным. Если он похищен, потерян или раскрыт, то становится бесполезным. Неосторожный субъект, у которого похитили ключ, может столкнуться, например, с тем, что кто-то другой подписывает документы его именем. Внимательно относитесь к ключам. Иначе кто-нибудь, воспользовавшись вашим сертификатом, сделает срочный заказ на профессиональные графические станции стоимостью в полмиллиона долларов и укажет адрес уединенного дома в центре Детройта![1]

Конфигурирование OpenSSL

В операционной системе FreeBSD для шифрования открытым ключом используется пакет OpenSSL. Функциональность OpenSSL использует-

[1] Это чистая правда. И прежде чем вы спросите, я скажу, что я этого не делал! Те самые профессиональные графические станции дал мне мой друг.

ся многими программами, однако системному администратору редко приходится напрямую работать с OpenSSL. Хотя OpenSSL прекрасно справляется со своими обязанностями при настройках по умолчанию, тем не менее я предпочитаю изменять некоторые из них, чтобы упростить себе жизнь. Конфигурация OpenSSL располагается в файле */etc/ssl/openssl.cnf*. Большинство параметров настройки имеют вполне приемлемые значения, и вам не стоит изменять их, если, конечно, вы не являетесь специалистом в области криптографии. Те немногие параметры, которые желательно изменить, отвечают за функции создания криптографических сигнатур. Каждое значение по умолчанию отмечено строкой `_default`. Параметры настройки OpenSSL общего характера, представляющие наибольший интерес, показаны ниже со значениями по умолчанию:

```
❶ countryName_default          = AU
❷ stateOrProvinceName_default  = Some-State
❸ 0.organizationName_default   = Internet Widgits Pty Ltd
```

Значение параметра `countryName_default` ❶ – это двухсимвольный код страны. В моем случае – это `US`. Значение параметра `stateOrProvinceName_default` ❷ – это название административной области, в которой вы проживаете; может иметь произвольную длину. Я присваиваю этому параметру значение `Michigan`. Значение параметра `0.organizationName_default` ❸ – это название компании. Если бы я приобрел подписанный сертификат, тогда здесь я поместил бы название компании, указанное в сертификате. Если вы просто проверяете возможность работы программ с SSL, то можно не указывать настоящее название компании. Я мог бы указать здесь название компании, в которой я работаю, или придумать свое название.

Следующие значения отсутствуют в файле *openssl.cnf*, но если добавить их, в командной строке OpenSSL они будут отображаться как параметры со значениями по умолчанию. Я считаю следующие настройки полезными, даже при том, что изменять их приходится гораздо чаще, чем предыдущие – для меня они играют роль подсказок.

```
❶ localityName_default           = Detroit
❷ organizationalUnitName_default  = Authorial Division
❸ commonName_default              = www.absolutefreebsd.com
❹ emailAddress_default            = mwlucas@absolutefreebsd.com
```

Значение параметра `localityName_default` ❶ – это название города. Значение параметра `organizationalUnitName_default` ❷ – подразделение компании, для которого был выдан данный сертификат. Следующий параметр OpenSSL, `commonName_default` ❸, зачастую становится камнем преткновения для системных администраторов – это имя хоста, как оно отображается при обратном преобразовании имен в DNS (глава 14), для которого был выдан сертификат. Наконец, параметр `emailAddress_default` ❹ – это адрес электронной почты администратора. Все эти значения отображаются в строке приглашения к вводу команды OpenSSL

в виде значений по умолчанию, и помогут избавить вас от лишнего раздражения.

Сертификаты

Интересная особенность шифрования открытым ключом заключается в том, что автор и получатели сообщений необязательно должны быть людьми – это могут быть программы. Безопасная командная оболочка (Secure Shell, SSH) и протокол защищенных сокетов (Secure Sockets Layer, SSL) представляют собой две возможности взаимодействия программ между собой, защищая их данные от просмотра злоумышленниками. Криптография с открытым ключом – это важный компонент *цифровых сертификатов* (*digital sertificates*), используемых защищенными вебсайтами и почтовыми службами. Покупая что-нибудь с помощью Firefox в режиме онлайн, возможно, вы и не догадываетесь, что броузер неистово зашифровывает и расшифровывает веб-страницы. Вот почему ваш компьютер может пожаловаться на «недействительные сертификаты» (invalid certificates) – когда истек срок действия открытого ключа либо когда используется самозаверяющий сертификат (self-signed).

Многие компании, например VeriSign, предоставляют услуги по заверению открытых ключей. Такие компании называются *Certificate Authorities* (CAs, центры сертификации). Другие компании, которым необходимо заверить сертификаты, предоставляют доказательства подлинности предоставляемой о себе информации, например корпоративные документы и деловые книги, а Certificate Authority применяет свой сертификат для заверения сертификатов этих компаний. Заверяя сертификат, Certificate Authority говорит: «Я проверил документы этой персоны (организации). Он (или она) является тем, за кого себя выдает». Однако Certificate Authority не гарантирует, что эта персона не применит сертификат для веб-сервера, продающего поддельные или опасные продукты, или для шифрования требования выкупа. Подписанный (заверенный) сертификат гарантирует определенную техническую безопасность, а не персональную честность или одностороннюю техническую безопасность. Сертификаты не являются волшебной таблеткой, обеспечивающей безопасность вашей системы.

В веб-броузерах и других программах, применяющих сертификаты, есть сертификаты основных CAs. Получая сертификат, подписанный Certificate Authority, броузер принимает его. По сути, он говорит: «Я доверяю Certificate Authority, а Certificate Authority доверяет этой компании; следовательно, я буду доверять этой компании». Пока вы доверяете источнику сертификатов, схема работает.

Большинство CAs, предоставляющих услуги по выдаче заверенных сертификатов, являются крупными компаниями. Особняком в этом ряду стоит некоммерческая организация CACert, которая бесплатно предоставляет сертификаты SSL всем, кто может подтвердить свою идентификационную информацию. Популярность CACert растет с каждым

годом, и, хотя корневой сертификат CACert отсутствует в Internet Explorer, в ближайшем будущем я надеюсь увидеть его в Mozilla Firefox.

Для нужд тестирования вполне подойдет сертификат, не заверенный Certificate Authority. Кроме того, незаверенные сертификаты могут использоваться для внутренних нужд компании, где можно установить сертификат в веб-броузер или сообщить пользователям, что данному сертификату можно доверять. Далее мы рассмотрим оба способа.

В обоих случаях использования сертификата необходимо сгенерировать ключ хоста.

Ключ хоста SSL

Как для заверенных, так и для самозаверенных сертификатов необходимо создать секретный ключ хоста. Ключ хоста – это всего лишь тщательно сгенерированное случайное число. Следующая команда создаст 1024-битовый ключ хоста и поместит его в файл *host.key*:

```
# openssl genrsa 1024 > host.key
```

После этого вы увидите сообщение от OpenSSL о том, что создается ключ хоста, и точки, бегущие по экрану, отражающие процесс генерирования ключа. Всего через несколько секунд вы получите файл, содержащий ключ. Это обычный текстовый файл, начинающийся со слов `BEGIN RSA PRIVATE KEY`, вслед за которыми следует блок случайных символов.

Храните в секрете свой ключ хоста! Назначьте суперпользователя root владельцем этого файла и сделайте его доступным для чтения только для root. После того как вы начнете использовать свой сертификат, любой, кто завладеет этим ключом, сможет использовать его для расшифровывания секретных сообщений.

```
# chown root host.key
# chmod 400 host.key
```

Поместите этот ключ в каталог с теми же правами доступа, какие были определены для самого файла.

Создание запроса на получение сертификата

Для получения любого сертификата, как заверенного, так и незаверенного, необходимо создать запрос. Мы не будем углубляться в подробности работы OpenSSL, поэтому мы не будем анализировать работу команды. Перейдите в каталог, где находится ключ хоста, и введите такую команду:

```
# openssl req -new -key host.key -out csr.pem
```

В ответ на экране появятся инструкции и будет задан ряд вопросов. По нажатии клавиши Enter будут вводиться ответы по умолчанию. Если предварительно вы произвели конфигурирование OpenSSL, ответы по умолчанию должны быть верными.

```
❶ Country Name (2 letter code) [US]:
(Название страны (2-буквенный код))
❷ State or Province Name (full name) [Michigan]:
(Название области или края (полное название))
❸ Locality Name (eg, city) [Detroit]:
(Название местности (например, города))
❹ Organization Name (eg, company) [Absolute FreeBSD]:
(Название организации (например, компании))
❺ Organizational Unit Name (eg, section) [Authorial Division]:
(Название подразделения компании (например, отдела))
❻ Common Name (eg, YOUR name) [www.absolutefreebsd.com]:
(Принятое имя (например, ВАШЕ имя))
❼ Email Address [mwlucas@absolutefreebsd.com]:
(Адрес электронной почты)
```

Двухбуквенные коды стран ❶ определены в стандарте ISO 3166, поэтому вы быстро сможете найти код своей страны в Интернете. Если вы не знаете название области (края, республики) ❷ и города ❸ – спросите у кого-нибудь, кто хотя бы изредка покидает машинный зал. Название организации ❹ – это, скорее всего, название вашей компании, затем следует указать название подразделения или отдела ❺. Если вы не работаете, тогда введите свои имя и фамилию или какую-нибудь другую информацию, идентифицирующую вас уникальным образом, а для самозаверенного сертификата здесь вообще можно написать все, что угодно. Различные центры сертификации предъявляют разные требования к заполнению этого поля частными лицами, поэтому обязательно ознакомьтесь с инструкциями центра сертификации.

Принятое имя ❻ оказывается камнем преткновения для большинства системных администраторов. Это не «ВАШЕ» имя, это имя сервера, которое возвращается службой DNS при обратном разрешении имен. Если не указать здесь имя сервера, запрос будет бесполезен.

В качестве адреса электронной почты ❼ я рекомендую использовать общий корпоративный адрес. В этом случае я *являюсь* Absolute FreeBSD, независимо от того, что бы это могло значить. Не следует указывать адрес электронной почты конкретного сотрудника, который может покинуть компанию по каким-либо причинам.

```
Please enter the following 'extra' attributes
to be sent with your certificate request
❶ A challenge password []:
❷ An optional company name []:
(Пожалуйста, введите 'дополнительные атрибуты', которые будут
посланы вместе с запросом сертификата
Запрашиваемый пароль []:
Необязательное название организации []:)
```

Запрашиваемый пароль ❶ также называют *идентификационной фразой* (*passphrase*). Повторюсь: он должен храниться в тайне, поскольку тот, кому станет известна ваша идентификационная фраза, сможет ис-

пользовать ваш сертификат! Использование запрашиваемого пароля является необязательным. Но если вы используете его, вам придется вводить его на каждом запуске сервера, то есть в случае краха веб-сервера веб-сайт не будет работать, пока кто-нибудь не введет идентификационную фразу. Я рекомендую использовать идентификационную фразу, но в некоторых случаях это может оказаться неприемлемым. Нажмите клавишу Enter, чтобы использовать пустую идентификационную фразу.

Так как вы уже вводили несколько названий компании, вводить название в третий раз ❷ нет смысла.

После того как вы вернетесь в командную строку, в текущем каталоге появится файл *csr.pem*. Он выглядит почти так же, как файл с ключом хоста, за исключением того, что в первой строке вместо слов `BEGIN RSA PRIVATE KEY` находится фраза `BEGIN CERTIFICATE REQUEST`.

Получение заверенного сертификата

Отправьте файл *csr.pem* в выбранный центр сертификации, и обратно вы получите файл, который будет очень похож на предыдущие файлы. Сохраните этот файл под именем *signature.pem* и запустите следующие команды:

```
# cp csr.pem cert.pem
# cat signature.pem >> cert.pem
```

В результате будет создана копия ключа хоста с присоединенной сигнатурой. Полученный файл и будет являться полным сертификатом. Данный сертификат можно использовать для любых служб, защищенных SSL, включая веб-страницы, pop3ssl и любых других демонов, обладающих поддержкой SSL.

Получение самозаверенного сертификата

Самозаверенный сертификат технически ничем не отличается от сертификата, заверенного центром сертификации, но он не передается на подпись Certificate Authority. Вместо этого вы создаете свою собственную сигнатуру. Большинство клиентов отказываются принимать самозаверенные сертификаты от рабочих служб, но для нужд тестирования такие сертификаты вполне пригодны. Чтобы подписать сертификат своей собственной подписью, выполните следующие команды:

```
# openssl x509 -req -days ❶365 -in csr.pem -signkey host.key -out
❷selfsigned.crt
Signature ok
subject=/C=US/ST=Michigan/L=Detroit/O=Absolute FreeBSD/OU=Authorial Division/
CN=www.absolutefreebsd.com/emailAddress=mwlucas@absolutefreebsd.com
Getting Private key
#
```

Вот и все! Теперь у вас есть самозаверенный сертификат, который будет действовать 365 дней ❶, в файле *selfsigned.crt* ❷. Этот ключ можно

использовать точно так же, как и заверенный сертификат, столько времени, сколько вы будете готовы терпеть предупреждения приложений.

Если вы подписываете собственные сертификаты, клиентские программы будут выдавать предупреждения о том, что «сторона, подписавшая сертификат, неизвестна» (certificate signer is unknown). Ничего удивительного, в конце концов, люди за стенами моего офиса понятия не имеют, кто такой Майкл Лукас и почему он подписывает сертификаты веб-сайта! Компании VeriSign и другие центры сертификации пользуются доверием. Мне доверяют люди, которые меня знают[1], но нельзя ожидать, что мне будет доверять весь мир. По этой причине не следует применять самозаверенные сертификаты в системах, где пользователи его увидят, поскольку выдаваемые предупреждения смутят их, раздосадуют или даже отпугнут. Потратьте сотню долларов или около того и получите настоящую CA-подпись своих рабочих сертификатов.

Соединение с портами, защищенными SSL

Я уже говорил, что мы не будем слишком углубляться в использование OpenSSL, и это правда. Однако существует одна особенность, которая слишком удобна, чтобы отказываться от нее; и как только вы узнаете о ее существовании, вы будете пользоваться ею по крайней мере раз в месяц и будет довольны, что она существует.

На протяжении всей книги мы проверяем работу сетевых служб, используя telnet(1) для подключения к портам, с которыми работают демоны, и запуска различных команд. Такой подход хорошо зарекомендовал себя при работе с текстовыми службами, такими как SMTP, POP3 и HTTP. Но он непригоден для работы с такими службами, как HTTPS, использующими шифрование. Необходима программа, которая способна выполнять шифрование при работе с такими службами. OpenSSL включает команду `openssl s_client`, которая предназначена именно для такой отладки клиентов. Кроме того, что вам будет доступна масса криптографической информации, вы еще получите возможность запускать текстовые команды и получать ответы. Чтобы установить соединение, команде `openssl s_client -connect` следует передать имя хоста и номер порта, разделенные двоеточием. Ниже приводится команда подключения к защищенному веб-серверу *www.absolutefreebsd.com*:

```
# openssl s_client -connect www.absolutefreebsd.com:443
CONNECTED(00000003) depth=1 /O=VeriSign Trust Network/OU=VeriSign, Inc./
OU=VeriSign International Server CA - Class 3/OU=www.verisign.com/CPS
Incorp.by Ref.
...
```

Вы увидите массу информации о цепочках доверия и ограничении ответственности, а также строки, напоминающие содержимое цифровых

[1] Во всяком случае большинство из них. Многие из них. По крайней мере хоть кто-нибудь из них. Да бог с ними!

сертификатов. После всего этого вы увидите пустую строку без приглашения к вводу. Сейчас вы общаетесь с демоном напрямую. А так как это веб-сервер, попробуем ввести команду HTTP:

```
GET /
```

Система ответит:

```
Object Moved
This document may be found <a href="https://www.nostarch.com/">here</
a>read:errno=0
```

Эта команда позволяет протестировать защищенные службы так же легко, как и незащищенные.

Некоторые из вас могут задаться вопросом – а зачем тогда шифровать службу, если общаться с ней так просто? Шифрование не защищает демона – шифрование защищает поток данных между клиентом и сервером. Шифрование средствами SSL препятствует просмотру данных обмена посторонними лицами – шифрование не защищает ни клиента, ни сервер. SSL не сможет предотвратить несанкционированное вторжение в ваш компьютер.

После всего сказанного я рассчитываю, что вы поняли назначение этой команды и принципы ее использования.

Клетки

Один из самых старых механизмов защиты UNIX построен на идее измененного корневого каталога (changed root), или *chroot*, согласно которой пользователь или программа ограничивается подразделом файловой системы, что защищает других пользователей и остальную часть файловой системы. chroot подходит для таких служб, как named(8), но ограничивает работу сложных программных комплексов, которым необходим доступ ко всей системе. Использование механизма chroot для организации веб-сервера или почтовой службы потребует приложить немалые усилия, направленные на добавление большого числа программ в chroot-окружение. Если вы работаете в компании, предоставляющей услуги веб-хостинга, вашим клиентам точно не понравится оказаться в chroot!

Клиенты, которые понимают мощь UNIX, нередко предъявляют запросы, усложняющие жизнь администратора. Например, хотят установить то или иное программное обеспечение или переконфигурировать веб-сервер, чтобы включить последний крутой модуль Apache. В двух словах, они хотят получить доступ с правами root, а в большинстве систем UNIX нельзя раздавать доступ root клиентам на сервере с множеством пользователей.

Нельзя, если у вас не FreeBSD. Администраторы FreeBSD столкнулись с этой проблемой довольно давно и решили ее, значительно улучшив механизм chroot. По сути, они решили ее так хорошо, что в системе

FreeBSD можно создавать на диске целые виртуальные машины и изолировать их от остальной части системы. Такая виртуальная машина называется *клеткой* (*jail*).

Клетку можно представить как среду клиент-сервер. Основной сервер – это хост, а каждая система-клетка – это клиент. Изменения, проводимые в хосте, можно отразить на всех системах, но изменения в клетке не затронут основную систему (если только не позволить клетке заполнить весь диск или что-либо подобное).

С точки зрения пользователя клетка очень похожа на полноценную систему FreeBSD, в которой отсутствует всего несколько устройств. Находясь в клетке, клиенты могут иметь доступ root и даже устанавливать любые программы по своему желанию. Это не окажет влияния на основную систему. Все процессы, запускаемые в клетке, ограничены ее средой. Ядро не предоставляет им доступ к информации, лежащей за пределами клетки. Поскольку программы и процессы в клетке ничего не знают о «внешнем мире» и не могут читать или получить к чему-либо доступ за его пределами, пользователь заблокирован. Мало того, что клиент не может вырваться из клетки, – если в клетку проник злоумышленник, дальше нее он не пройдет. Такой механизм позволяет обезопасить систему и удовлетворить потребности клиентов.

На современном оборудовании, с недорогими (но не дешевыми!) дисками и избытком памяти, система FreeBSD может обслуживать десятки клеток с веб-серверами. С точки зрения продвижения машина-клетка представляет собой хороший промежуточный вариант между виртуальным доменом на общем сервере и собственным локальным сервером.

Конфигурирование сервера для использования клеток

Перед использованием клеток необходимо убедиться, что сервер настроен правильно. Клетки налагают специальные требования на сервер. Самое досадное из них – демоны нельзя привязать ко всем доступным IP-адресам.

Каждая клетка привязана к определенному IP-адресу и определяется этим IP-адресом. Для каждой клетки необходимо к сетевой карте добавить псевдоним IP-адреса, как обсуждалось в главе 6.

Клетка должна иметь исключительный доступ к этому адресу; никто иной не может его использовать. Этот IP-адрес является единственным, который сможет использовать клетка.[1] Если в основном сервере есть демон, привязанный ко всем доступным IP-адресам системы, то

[1] В момент написания этих строк мне известно, что организация FreeBSD Foundation продолжает работу над созданием виртуального стека IP для использования в клетках. Это позволит клетке использовать более одного IP-адреса и получить более полный доступ к стеку TCP/IP; эта работа может быть уже завершена к тому моменту, когда вы читаете эти строки.

этот демон воспрепятствует запуску клетки. Взглянув на вывод команды sockstat(1), можно заметить несколько записей, в которых локальный адрес включает в себя символ звездочки «*». Это означает, что демон «прослушивает» все доступные IP-адреса.

```
# sockstat -4
...
root     inetd    895  5  tcp4  ❶*:21                *:*
root     sshd     822  4  tcp4  ❷*:22                *:*
root     syslogd  601  7  udp4  ❸*:514               *:*
...
```

В этом примере видно, что демон inetd прослушивает все IP-адреса на порту с номером 21 ❶, демон sshd – все IP-адреса на порту с номером 22 ❷ и демон syslogd – все IP-адреса на порту с номером 514 ❸. Нам необходимо заново сконфигурировать эти демоны, чтобы каждый из них прослушивал только один IP-адрес.

Для правильной настройки сервера клеток следует понять, что основной сервер – это лишь хост для клеток и не предоставляет никаких других услуг. Вам необходимы службы sendmail, named и другие? Нет проблем! Создайте клетку для этих служб и заключите в нее все необходимые демоны. Это не только упростит настройку всех этих программ, но и обеспечит дополнительную защиту для остальных клеток. В случае вторжения в хост-систему злоумышленник автоматически получает доступ ко всем клеткам, а вторжение в единственную клетку сильно ограничит его возможности границами единственной клетки.

В этом случае вам не придется настраивать программы, заключаемые в клетку, на использование единственного IP-адреса. Процесс в клетке обладает доступом лишь к одному IP-адресу, и потому демоны, стремящиеся подключиться ко всем доступным IP-адресам, будут вести себя, как и ожидается.

Ниже перечислены некоторые распространенные демоны, которые могут быть источниками проблем при запуске на хост-сервере. Во всех этих примерах предполагается, что клетка имеет IP-адрес 192.168.1.1.

syslogd

syslogd – это системный регистратор (system logger), который открывает сокет, а значит, может посылать регистрируемые сообщения другим серверам, даже если они не принимают никаких сообщений. Чтобы подключить syslogd(8) к конкретному IP-адресу, следует использовать ключ `-b`, в файле *rc.conf*:

```
syslogd_flags="-b 192.168.1.1"
```

Чтобы заставить замолчать syslogd, следует использовать ключ `-ss`. Однако в этом случае невозможным будет удаленное ведение протоколов. Программа syslogd(8) подробно рассмотрена в главе 19.

inetd

Вообще говоря, этот демон лучше запускать именно в клетке. Демон inetd(8) тоже может быть настроен на использование единственного IP-адреса с помощью ключа `-a`, например:

```
inetd_flags="-wW -C 60 -a 192.168.1.1"
```

Обратите внимание: inetd обычно запускается с ключами по умолчанию, которые определены в файле */etc/defaults/rc.conf*. В моей версии FreeBSD inetd по умолчанию запускается с ключами `-wW -C 60`. Поэтому, помимо ограничения привязкой к единственному IP-адресу, я добавил еще и эти ключи.

sshd

Параметр настройки `ListenAddress` в файле */etc/sshd/sshd_config* указывает IP-адрес, который может использоваться демоном sshd для приема запросов на соединение.

```
ListenAddress 192.168.1.1
```

Будет очень разумно обеспечить работу службы SSH на сервере клеток. Если sshd(8) – это единственная служба, предоставляемая сервером, значит вы все сделали правильно.

NFS

Программы NFS, такие как rpcbind(8) и nfsd(8), прослушивают все IP-адреса системы, и повлиять на такое их поведение очень сложно. Не запускайте сервер NFS на сервере клеток. Тем не менее NFS можно использовать внутри клетки. Если вам все-таки необходимо иметь сервер NFS, не используйте основной сервер, а настройте работу NFS внутри клетки.

Клетки и ядро

Теперь, когда сетевая подсистема готова к размещению клеток на сервере, давайте заглянем в ядро. Настройки клеток по умолчанию имеют вполне приемлемые значения, однако системный администратор имеет возможность выполнить дополнительные регулировки. В большинстве случаев изменение этих параметров настройки не создает ничего, кроме проблем.

Система клеток имеет свое дерево параметров sysctl, `security.jail`. Эти параметры можно изменять только из основной системы, поскольку из клеток они недоступны, а изменения затрагивают все клетки, запущенные на сервере.

security.jail.set_hostname_allowed

По умолчанию в клетке пользователь root может устанавливать имя хоста для этой клетки. Поскольку клетка использует это имя для

взаимодействия с «основным» хостом, изменение имени хоста может легко запутать администратора, ответственного за управление клетками. Если установить этот sysctl в значение 0, изменение имени хоста будет запрещено.

security.jail.socket_unixiproute_only

По умолчанию с клеткой можно взаимодействовать только через протокол IP и локальные сокеты UNIX. Хотя маловероятно, что пользователь будет применять, скажем, протокол IPX, теоретически это вполне возможно. Однако система клеток поддерживает только IP. Позволить пользователям применять другие протоколы – значит, позволить им «выбираться» из клетки. Система клеток виртуализирует только IP, но не IPX или другие сетевые протоколы, поэтому все остальные протоколы, за исключением IP, можно использовать для доступа ко всем клеткам. Такой доступ, в общем, безопасен, однако неразумно считать себя умнее любого злоумышленника. Значение 1, используемое по умолчанию, ограничивает доступ к системе только протоколом IP; значение 0 позволит использовать любой сетевой протокол, отличный от IP.

security.jail.sysvipc_allowed

System V IPC – это стандарт UNIX для разрешения межпроцессного взаимодействия (Interprocess Communication, IPC) через сегменты разделяемой памяти. По существу, связанные друг с другом программы могут использовать один участок памяти для хранения информации. Многие приложения баз данных используют IPC. Система клеток не создает отдельные области памяти для каждой клетки. Разрешение IPC позволило бы информации просачиваться из клетки и обратно. Впрочем, воспользоваться таким слабым местом способен лишь квалифицированный взломщик. Значение 0, принятое по умолчанию, запрещает использование механизмов IPC, значение 1 – разрешает.

security.jail.enforce_statfs

Данный параметр управляет возможностью пользователя, находящегося в клетке, видеть информацию о файловой системе. По умолчанию (security.jail.enforce_statfs=2) программы в клетках могут видеть только точки монтирования, находящиеся непосредственно в клетке, а путь к корневому каталогу усечен так, что из клетки нельзя определить, где на диске находится каталог, заключенный в клетку. Если значение этого параметра установить равным 1, пользователь попрежнему сможет видеть только точки монтирования, находящиеся непосредственно в клетке, но при этом он сможет узнать полный путь к своему каталогу в хост-системе. Если значение этого параметра установить равным 0, пользователь сможет видеть все точки монтирования. Причины, которые могли бы служить веским основанием для изменения значения по умолчанию, встречаются крайне редко.

security.jail.allow_raw_sockets

Низкоуровневые сокеты обеспечивают прямой доступ к сетевой подсистеме. Если пользователи и программы, заключенные в клетки, не вызывают доверия, не разрешайте им доступ к низкоуровневым сокетам. Низкоуровневые сокеты TCP/IP используются такими программами, как ping и traceroute, поэтому некоторые клиенты могут изъявить желание иметь такую возможность. Однако, с точки зрения безопасности, слишком мало причин за то, чтобы разрешать эту возможность, и слишком много причин против. Значение `0`, используемое по умолчанию, запрещает доступ к низкоуровневым сокетам, `1` – разрешает.

security.jail.chflags_allowed

По умолчанию пользователи, заключенные в клетки, не могут использовать флаги файловой системы и утилиту chflags(1) (глава 7). Напомню, что многие из этих флагов невозможно сбросить без перезагрузки в однопользовательском режиме. При неумелом обращении с chflags(1) ваши клиенты могут навлечь на себя неприятности, и только вы сможете ликвидировать их. Это верное средство для службы поддержки заработать себе головную боль. Чтобы разрешить использование утилиты chflags(1), измените значение по умолчанию `0` на `1`.

security.jail.jailed

Это значение, доступное только для чтения, покажет, запускается ли sysctl(8) из хост-системы (`0`) или из клетки (`1`).

security.jail.list

В хост-системе этот параметр содержит список всех активных клеток.

Установка клиента

Для начала необходимо решить, где будут размещаться клетки. Я рекомендую использовать для этих целей отдельный раздел, чтобы расходование дискового пространства вашими клиентами никак не сказывалось на хост-системе. Некоторые администраторы пристально следят за своими клиентами и поднимают плату для пожирателей дискового пространства. Файловые диски (глава 8) – это самый простой способ создания отдельных разделов для клеток, который не требует создания дополнительных физических разделов на диске. В данном примере мы установим первую клетку в каталог */var/jail/jail1*.

Процесс установки клетки по своей сути напоминает процесс обновления FreeBSD: сборка исполняемых файлов из исходных текстов и их установка. Только в случае с клеткой установка производится в другое местоположение. Если вы производили обновление системы, значит вы уже выполняли сборку программ, если нет – перейдите в каталог */usr/src* и запустите команду:

```
# make buildworld
```

После того как система будет собрана, ее нужно просто установить. Для выбора иного каталога установки следует использовать параметр командной строки DESTDIR. Однако, чтобы установить каталоги */etc* и */var*, нам придется выполнить дополнительные действия. (Они не требуются при обновлении системы, так как в хост-системе эти каталоги уже присутствуют.)

```
# make installworld DESTDIR=/var/jail/jail1
# make distribution DESTDIR=/var/jail/jail1
# mount -t devfs devfs /var/jail/jail1/dev
```

Последняя команда производит установку файлов устройств в клетку. Хотя это и не является необходимым, тем не менее многие программы предполагают наличие доступа к таким основным устройствам, как терминалы, генераторы случайных чисел и т. п. Файловая система devfs в клетке вам потребуется как минимум на этапе установки.

Уменьшение дискового пространства, занимаемого клеткой

В Интернете можно найти множество рекомендаций по уменьшению дискового пространства, требуемого клетке. Объединение точек монтирования и NFS поможет снизить требования, предъявляемые к объему дискового пространства, однако чем больше уловок вы будете использовать, тем меньше поддержки от сообщества FreeBSD вы получите.

Украшаем клетку: установка внутри клетки

К этому моменту все готово к первой загрузке клетки. Не нужно путать понятия: *первая загрузка* и *готова к использованию*, потому что до полной готовности нам еще нужно выполнить некоторые действия внутри клетки. Воспользуйтесь командой jail(8) для запуска клетки:

```
# jail <path to jail> <jail hostname> <jail IP> <command>
```

Например, наша первая клетка располагается в каталоге */var/jails/jail1*. Ей должно быть присвоено имя хоста *jail.absolutefreebsd.com* и IP-адрес 192.168.1.4. Для этого следует ввести команду:

```
# jail /var/jails/jail1 jail.absolutefreebsd.com 192.168.1.4 /bin/sh
```

Самый ранний этап, где система может предоставить вам доступ к командной строке, – это однопользовательский режим. Это не совсем однопользовательский режим клетки, поскольку в ней пока нет других программ, кроме */bin/sh*.

Если присмотреться внимательнее, можно заметить, что фактически была установлена FreeBSD в минимальном варианте. Здесь нет никаких программ, кроме тех, что составляют основу FreeBSD: у вас нет ни учетной записи, ни пароля root, ни сетевых демонов и никаких дополнительных программ. Прежде чем клетку можно будет использовать, следует установить все необходимое.

Создание /etc/fstab

Многие программы предполагают наличие в системе файла */etc/fstab* и не могут работать при его отсутствии. Этот файл имеет большое значение для настоящего сервера, но он бесполезен для системы в клетке. Однако, чтобы обеспечить работоспособность таких программ, лучше будет создать пустой файл.

```
# touch /etc/fstab
```

Настройка разрешения имен DNS

Вероятно, вам потребуется настроить DNS в файле */etc/resolv.conf*. Для этого можно скопировать в клетку файл */etc/resolv.conf* из хост-системы. Обратите внимание: вы должны скопировать файл из хост-системы, потому что клетка не имеет доступа к каталогу */etc* хоста.

sendmail

Вам не нужно выполнять настройку Sendmail(8), по крайней мере, пока, но вы будете постоянно получать предупреждения о том, что устарела база данных псевдонимов. Чтобы такие предупреждения не появлялись, запустите команду newaliases(8).

/etc/rc.conf

Добавьте в файл */etc/rc.conf* клетки следующие строки:

```
network_interfaces=""
sshd_enable="YES"
```

Вы не сможете настроить сетевой интерфейс внутри клетки, но сценарии начальной загрузки предполагают, что это будет делаться. Предупредите систему FreeBSD, чтобы она не волновалась по этому поводу, так как возможность настройки интерфейса вам недоступна.

Доступ к клетке можно получить только по сети, поэтому вам потребуется активировать SSH.

Пароль root и учетная запись

В клетке пока еще отсутствует пароль root. Установите его командой passwd(1). Вам также потребуется создать учетную запись пользователя. Запустите команду adduser(8) и создайте хотя бы одну учетную запись.

Прочие настройки

При желании в клетке можно определить часовой пояс с помощью команды tzsetup(8), установить дополнительные пакеты, скопировать личные файлы в домашний каталог и прочее, но все эти действия не являются необходимыми на данном этапе. Покиньте командный интерпретатор, и клетка завершит работу.

Клетка и /etc/rc.conf

Теперь, когда клетка готова к эксплуатации, можно сообщить об этом системе. Самый простой способ управления клетками заключается в использовании параметров файла */etc/rc.conf*. Ниже приводятся несколько примеров настройки системы для работы с клетками:

```
❶ jail_enable="YES"
❷ jail_list="jail1 jail2 jail3"
```

Эти два параметра управляют общими настройками клеток. Установив значение `YES` в параметре `jail_enable` ❶, вы тем самым предписываете сценариям начальной загрузки системы отыскать и обработать дополнительные параметры настройки клеток. Параметр `jail_list` ❷ содержит перечень всех клеток в системе, разделенных пробелами. Установив значения этих двух параметров, вы сможете использовать систему запуска FreeBSD для управления вашими клетками. Кроме того, каждая клетка имеет свой собственный набор параметров настройки, которые содержат информацию о том, где находится каждая клетка и как она должна настраиваться.

```
❶ jail_jail1_rootdir="/var/jails/jail1"
❷ jail_jail1_hostname="jail.absolutefreebsd.com"
❸ jail_jail1_ip="192.168.1.4"
❹ jail_jail1_devfs_enable="YES"
❺ jail_jail1_devfs_ruleset="devfsrules_jail"
```

Эти записи описывают конфигурацию единственной клетки. Похожие записи следует добавить для каждой создаваемой вами клетки, замещая название `jail1` в именах переменных. Для каждой клетки должны быть определены корневой каталог ❶, имя хоста ❷ и IP-адрес ❸.

Многие программы выводят предупреждения или вообще отказываются работать, если не находят определенные файлы устройств в каталоге */dev*. В клетках желательно создать некоторые файлы устройств, такие как терминалы или генератор случайных чисел. Другие файлы, как, например, файлы дисковых устройств или сетевых интерфейсов, создавать не нужно. Можно предоставить доступ к устройствам с помощью devfs ❹, но тогда необходимо применить правила ❺, которые обеспечивали бы доступ только к определенным устройствам. О devfs мы говорили в главе 8. Операционная система FreeBSD включает определенный набор правил для файловой системы devfs, заключенной в клетку, в виде параметра `devfsrules_jail` в файле */etc/defaults/devfs.rules*.

В файле */etc/defaults/rc.conf* можно отыскать и другие параметры, управляющие определенными аспектами клеток, но те, что были описаны выше, охватывают все самое необходимое.

Запуск и останов клетки

Выполнив вышеупомянутые настройки в файле *rc.conf*, можно будет использовать сценарий */etc/rc.d/jail* для управления клетками, как всеми сразу, так и по отдельности. Для запуска всех клеток достаточно просто запустить команду:

```
# /etc/rc.d/jail start
```

Можно запустить и единственную клетку, указав ее имя в качестве аргумента:

```
# /etc/rc.d/jail start jail1
```

Запустив клетку, можно использовать SSH для входа в нее. После этого можно добавлять пакеты, настраивать локальную систему и вообще делать все, что угодно со своей «новой» машиной FreeBSD. Поиграйте немного, попробуйте, к примеру, сбежать из клетки. Попытайтесь выйти в известный вам каталог, находящийся за пределами клетки. Даже обладая привилегиями root в клетке, вы не сможете просмотреть или обратиться к процессам, работающим на ведущей системе. Даже с помощью таких мощных инструментов, как Perl и cc(1), позволяющих создавать любые инструменты, вы не сможете вырваться в ведущую систему. Вы можете даже использовать csup в клетке и полностью перекраивать ее мир, хотя это и *не* самая лучшая идея. Не забывайте, что функциональные возможности ядра должны соответствовать требованиям пользовательских программ, в противном случае, хоть это может и не вызвать крах ведущей системы, но несоответствие версий ядра и пользовательских программ станет причиной некорректного поведения последних.

Управление клетками

В клетках происходит сложное управление процессами. Если зарегистрироваться на сервере с клетками, можно увидеть все процессы во всех клетках. Какие из них принадлежат самому серверу, а какие – клеткам?

Команда `ps -ax`, выполненная в «основной» системе, покажет все работающие процессы, даже процессы клеток. Символ `J` в поле `STAT` означает, что процесс запущен в клетке. Если на сервере есть несколько клеток, то исходя из предназначения каждой из них, можно догадаться, к какой клетке относится тот или иной процесс. Например, если есть только один сервер имен, и он находится в клетке, можно смело держать пари на принадлежность процесса named(8). Однако в большинстве случаев все не так хорошо, как кажется. И здесь нам на помощь приходят два инструмента: jls(8) и jexec(8).

jls

Программа jls(8) выводит список всех клеток, запущенных в системе.

```
# jls
   JID  IP Address   Hostname                    Path
   ❶1 ❷192.168.1.4 ❸jail.absolutefreebsd.com  ❹/var/jails/jail1
    2   192.168.1.5  jail2.blackhelicopters.org  /var/jails/jail2
...
```

Каждая клетка имеет уникальный числовой идентификатор (Jail ID), или JID ❶. Идентификатор *JID* очень похож на числовой идентификатор процесса, в том смысле, что каждая клетка имеет единственный идентификатор, но его точное значение может изменяться. Если остановить клетку и снова запустить ее, ей будет присвоено другое значение JID. Кроме того, утилита jls(8) выводит IP-адрес ❷, имя хоста ❸ и корневой каталог ❹ клетки.

jexec

Утилита jexec(8) позволяет администратору ведущей системы исполнять команды внутри любой запущенной клетки без необходимости входить в нее. Это позволит сохранить у владельцев клеток чувство личного пространства – в конце концов, вам не нужно знать пароли root в клетках и даже не требуется входить в их системы. Но для использования jexec(8) вам необходимо знать значение JID клетки. Чтобы узнать, какие процессы работают внутри клетки, достаточно запустить в ней команду `ps -ax`. Например, предположим, что наша первая клетка, *jail.absolutefreebsd.com*, имеет ID со значением 1. Чтобы выполнить команду `ps -ax` внутри этой клетки, следует запустить такую команду:

```
# jexec 1 ps -ax
```

После этого вы увидите список процессов, запущенных в клетке, как если бы вы вошли в нее.

Утилита jexec(8) позволяет запускать любые команды в клиентской клетке из ведущей системы. Вы можете устанавливать программное обеспечение, перезапускать демоны, останавливать процессы или изменять пароли пользователей.

Процессы и procfs

Помните, как в главе 8 я говорил, что не стоит использовать файловую систему procfs? Сервер клеток – единственное исключение из этого правила. В истории безопасности файловой системы procfs имеется множество темных пятен, но на сервере клеток не должно быть никаких служб и посторонних пользователей. Единственное предназначение файловой системы процессов в данном случае – это идентификация клеток по идентификаторам процессов. Если, например, вы увидели, что некоторое приложение баз данных использует слишком большой объем памяти или процессорного времени, то с помощью

файловой системы */proc* вы можете быстро идентифицировать клетку по числовому идентификатору процесса (PID).

Для каждого запущенного процесса в файловой системе */proc* имеется отдельный каталог. Чтобы узнать, в какой клетке исполняется процесс, нужно отыскать каталог процесса по его идентификатору и посмотреть содержимое файла с именем *status*. Последнее слово в данном файле – это имя хоста клетки, в которую заключен процесс. Если процесс не заключен в клетку, последним словом в файле будет символ дефиса.

Останов клетки

При останове основного хоста также будут остановлены различные клиентские клетки. Останов клетки без останова хоста ненамного сложнее. Чтобы остановить все клетки или какую-нибудь одну клетку, следует запустить команду `/etc/rc.d/jail stop`. Ниже приводится пример останова клетки `jail1`:

```
# /etc/rc.d/jail stop jail1
```

Эта команда запустит стандартную процедуру останова внутри клетки и выйдет. В результате клетка исчезнет из списка команды jls(8).

Программы shutdown(8) и reboot(8) бесполезны для останова клетки, поскольку их основная обязанность состоит в синхронизации и размонтировании дисков, отсоединении от сети и т. д. В виртуальной машине это не требуется.

Что может быть не так с клетками

Клетки обеспечивают высокую гибкость при невысоких требованиях к аппаратному обеспечению. На небольшом жестком диске емкостью 80 Гбайт легко можно организовать десяток клеток и даже более, используя приемы, описанные в этой книге. А если вы ознакомитесь с секретными техниками ниндзя клеток, вы сможете утроить это число. Почему бы тогда не заключить все службы в отдельные клетки, а ведущую систему не использовать в качестве сервера клеток?

Удобство обслуживания – вот главная причина.

После создания всех этих клеток существенно увеличивается объем работ по обслуживанию одной системы. Например, если у вас имеется два десятка веб-серверов, заключенных в клетки на одной машине, а в Apache обнаружится какая-либо проблема безопасности, вам придется накладывать «заплаты» на два десятка веб-серверов в клетках. При надлежащем уровне планирования, умении писать сценарии и богатой практике это не очень большая проблема, но множество системных администраторов недостаточно сильны в планировании и не обладают богатыми практическими навыками.[1] Вы можете так поступить,

[1] Нередко недостаток практических навыков и опыта в планировании мы компенсируем созданием неплохих сценариев.

но вы должны понимать все трудности, с которыми вам придется столкнуться при развертывании скопища клеток.

С другой стороны, существует масса компаний, имеющих положительный опыт развертывания сотен клеток и управления ими в автоматическом режиме. Клетки способны решить массу проблем, но при этом они порождают другие проблемы и дают вам роскошный выбор той или иной головной боли.

Подготовка к вторжению с помощью mtree(1)

Одна из самых больших неприятностей, которые могут произойти с системным администратором, это подозрение, что кто-то проник в его систему. Если вы вдруг обнаруживаете странные файлы в каталоге */tmp*, дополнительные команды в */usr/local/sbin* или просто появляется ощущение что «что-то не так», сразу возникает вопрос – а не проник ли кто-нибудь посторонний в систему. Самое отвратительное во всем этом, что нет способов доказать обратное. Грамотный взломщик может заменить системные исполняемые файлы своими версиями, чтобы иметь возможность производить действия, которые не регистрируются в системе, и затруднить возможность его обнаружения. Даже Шерлок Холмс с лупой в руках ничего не сможет обнаружить в вашей системе, если лупа предоставлена самим злоумышленником и имеет особенность скрывать его! У некоторых имеется даже взломанный системный компилятор, который позволяет добавлять во вновь создаваемые исполняемые файлы программный код, представляющий собой «черный ход» для умельца.[1] Хуже того, компьютеры способны творить самые фантастические вещи. Операционные системы имеют чрезвычайно сложное устройство, а приложения содержат ошибки. Возможно, тот самый файл в каталоге */tmp*, что вызвал ваше беспокойство, это реакция вашего текстового редактора на ваши попытки слишком быстро нажимать клавиши, а возможно, след, оставленный злоумышленником.

Единственный способ восстановить взломанную систему заключается в том, чтобы заново переустановить ее, восстановить данные из резервной копии и уповать, что брешь в системе защиты, которая сделала вторжение возможным, была заделана. Но надежда на это настолько слабая, а сомнения возвращаются настолько быстро, что многие системные администраторы в конечном счете прекращают беспокоиться или лгут сами себе вместо того, чтобы жить с постоянным чувством тревоги.

---

[1] Я мог бы сказать *злоумышленник*, если бы таким человеком не был Кен Томпсон (Ken Tompson), один из создателей UNIX и языка программирования C. Он обладал удивительной способностью проникать в любую систему UNIX, находящуюся в любой точке мира, включая системы, которые были созданы спустя годы после того, как Кен прекратил работу над UNIX. Подробности вы найдете по адресу: *http://www.acm.org/classics/sep95*.

В большинстве случаев злоумышленники подменяют файлы, которые уже существуют в системе. А в операционной системе FreeBSD имеется утилита mtree(1), которая может записывать информацию о правах доступа, размерах и датах файлов в системе, а также их контрольные суммы. Если вы запишете все эти характеристики сразу после установки системы, вы будете знать, как должны выглядеть неиспорченные файлы. Если злоумышленник подменит какие-либо файлы, это можно будет обнаружить простым сравнением. Когда у вас появится ощущение, что система была взломана, вы сможете повторно снять характеристики с существующих файлов и узнать – не изменился ли какой-нибудь из них.

Запуск mtree(1)

Следующая команда запускает утилиту mtree(1) для сбора информации в корневой файловой системе и сохраняет в файле контрольные суммы md5 и sha256 для последующего анализа:

```
# mtree ❶-x ❷-ic ❸-K cksum ❹-K md5digest ❺-K sha256digest ❻-p / ❼-X /
home/mwlucas/mtree-exclude > ❽/tmp/mtree.out
```

Хотя mtree(1) можно использовать для сбора информации со всего сервера, тем не менее большинство предпочитают запускать утилиту для исследования одного раздела за раз, с помощью ключа `-x` ❶. Нет смысла записывать контрольные суммы для файловых систем, содержимое которых меняется слишком часто, как например, разделов с базами данных. Флаги `-ic` ❷ предписывают утилите mtree(1) выводить результаты на экран, с отступами для каждого следующего уровня вложенности в файловой системе. Такой формат соответствует структуре файлов в */etc/mtree*. Ключ `-K` принимает несколько необязательных ключевых слов – в данном случае предписывается генерировать стандартные контрольные суммы ❸, контрольные суммы md5 ❹ и контрольные суммы sha256 ❺. Ключ `-p` ❻ указывает, какой раздел должен проверяться. Практически в каждом разделе имеются файлы или каталоги, которые изменяются регулярно и которые по этой причине проверять было бы нежелательно. С помощью ключа `-X` ❼ можно указать файл, содержащий список каталогов, которые не должны проверяться. В заключение производится перенаправление вывода этой команды в файл */tmp/mtree.out* ❽.

Составление списка каталогов, которые не должны проверяться, может оказаться непростым делом. Обычно я не выполняю проверку содержимого каталогов */tmp* и */var/tmp*. Я могу указать оба каталога в одном и том же файле исключений. При использовании файла исключений, содержащем следующую строку, из проверки будет исключаться каталог *tmp*, находящийся в корне исследуемой файловой системы.

```
./tmp
```

Вообще говоря, злоумышленнику скорее потребуется подменить некоторые файлы в корневом разделе или в разделе */usr*. Если ваш сервер выполняет функции веб-сервера, целью злоумышленника могут также стать CGI и PHP-приложения.

Вывод mtree(1): файл спецификации

Вывод команды mtree(1) еще называют спецификацией, или *spec*. Первоначально эта спецификация предназначалась для установки программного обеспечения. Спецификация начинается с комментариев, в которых указываются имя пользователя, запустившего команду, имя компьютера, на котором была запущена команда, анализируемая файловая система и дата выполнения проверки. Первая настоящая запись в спецификации определяет значения по умолчанию для установки, которую мы не запускали, и начинается с ключевого слова `/set`.

```
/set type=file uid=0 gid=0 mode=0755 nlink=1 flags=none
```

mtree(1) выбирает эти значения по умолчанию на основе анализа файлов раздела. По умолчанию объектом файловой системы является файл, которым владеют UID 0 и GID 0, с правами доступа 0755, с одной жесткой ссылкой и без флагов файловой системы. Затем для каждого файла и каталога, присутствующего в файловой системе, создается отдельная запись. Ниже приводится пример записи для корневого каталога:

```
❶.          ❷type=dir ❸nlink=24 ❹size=512 ❺time=1171639839.0
```

Это файл с именем . (точка) ❶, или *текущий каталог*. Этот файл является каталогом ❷ и имеет 24 жестких ссылки ❸ на него. Размер файла каталога составляет 512 байтов ❹, а последнее его изменение производилось в момент времени, отстоящем на 1 171 639 839.0 секунд от начала эпохи UNIX ❺. Эпоха UNIX началась 1 января 1979 года.

Запись с информацией о каталоге не содержит почти ничего интересного. В конце концов, не может же злоумышленник подменить сразу целый каталог! Ниже приводится запись, соответствующая фактическому файлу, находящемуся в корневом каталоге.

Секунды от начала эпохи и реальные даты

У вас нет желания подсчитывать, сколько секунд прошло от начала эпохи UNIX? Чтобы преобразовать число секунд, прошедших от начала эпохи, в обычные дату и время, можно воспользоваться командой `date -r число_секунд`. Не забудьте при этом исключить завершающую пару символов `.0`, присутствующих в выводе mtree(1), – date(1) принимает только целые числа.

```
.cshrc     mode=0644 nlink=2 size=801 time=1141972532.0 \
❶cksum=3359466860 ❷md5digest=67b0d2664a9c2fcccb517e3069ca8125 \
❸sha256digest=6624a34d5e068cca1f64142e077b8c64869dc2207bf7ff4f292a7f1f3e237b4
```

Здесь мы видим имя файла, права доступа, количество жестких ссылок, размер и время, однако, по сравнению с каталогом, здесь появилось несколько дополнительных элементов: контрольная сумма ❶, контрольная сумма md5 ❷ и контрольная сумма sha256 ❸. Эти контрольные суммы вычисляются на основе содержимого файла. Хотя теоретически возможно, чтобы файл, подмененный злоумышленником, имел контрольную сумму, совпадающую с контрольной суммой оригинального файла, а специалисты в области криптографии ищут практические способы создания файлов, которые соответствовали бы произвольным контрольным суммам md5 и sha256, тем не менее крайне маловероятно, чтобы злоумышленнику удалось создать поддельный файл, который соответствовал бы сразу всем трем контрольным суммам, содержал «черный ход» и при этом продолжал выполнять привычные функции, чтобы владелец системы не сразу заметил подмену. К тому времени, когда такое станет возможным, у нас в арсенале появятся дополнительные алгоритмы вычисления контрольных сумм, способные противостоять методам, которые используются злоумышленниками, и нам останется лишь переключиться на них.

Сохранение файла спецификаций

Файл спецификаций содержит информацию, необходимую для проверки целостности системы после предполагаемого вторжения. Если оставить файл спецификаций на сервере, злоумышленник сможет отредактировать его и замести следы. Не следует хранить этот файл в самой системе! Время от времени я слышал предложения вычислять контрольную сумму самого файла спецификаций и хранить все на сервере. В этом нет никакого смысла: если кто-то изменит содержимое файла спецификаций и его контрольную сумму, как вы тогда узнаете об этом? Или еще хуже – если кто-то изменит файл спецификаций и вы обнаружите этот факт, то как тогда узнать, какие изменения были сделаны!

Храните копию файла спецификаций в безопасном месте, предпочтительно на съемном носителе, например на дискете или на компакт-диске.

Реакция на вторжение

Когда у вас появляются подозрения, что система была взломана, создайте новый файл спецификаций и сравните его с созданным ранее. Утилита mtree(1) может отыскивать различия между двумя файлами спецификаций. В результате будут сгенерированы тысячи строк текста, поэтому не забудьте перенаправить вывод в файл, чтобы упростить дальнейший анализ.

```
# mtree -f savedspec -f newspec > mtree.differences
```

Информация о файлах, которые не изменились, будет располагаться в одной строке, например, так:

```
sbin/conscontrol file
```

Файл */sbin/conscontrol* не изменялся между двумя проверками. Ниже приводятся две строки, которые были получены для файла */sbin/devd*:

```
❶sbin/devd file cksum=3950957068 size=328396 md5digest=14d0cc1dbe
a86c69ebd4af1dec2312d0 sha1digest=7acd1bdf46581b1d6a3231ca62b2c47e6e1dcf07
❷sbin/devd file cksum=4141842183 size=328428 md5digest=5ab5ec4213
03ca770ac2cccd546bcf11 sha1digest=51f5fbc24d47f11115474714040b8cce0eb9b6c9
```

Сравните старую ❶ и новую ❷ контрольные суммы файла */sbin/devd*, а также размеры. Они совершенно разные. Что-то изменилось в двоичном файле devd(8). Не торопитесь жать на кнопку с надписью «Караул», а начните задавать жесткие вопросы системным администраторам. Если вам не удастся получить вразумительный ответ, почему мог измениться двоичный файл, поищите резервную копию у себя на лентах.

Мониторинг системы безопасности

Итак, вы считаете, что сервер в безопасности. Может быть.

К сожалению, существуют взломщики, которые не находят лучшего занятия, кроме как держать себя в курсе последних «прорех» в защите. Опираясь на эти знания, они пытаются проникнуть в систему, которая, по их мнению, уязвима. Даже если вы добросовестно читаете сообщения рассылки *FreeBSD-security* и накладываете все появляющиеся «заплатки», однажды защита системы может быть взломана. Хотя сказать точно, что система взломана, не удастся, следующие приемы помогут узнать о таких попытках:

- Стремитесь понимать работу своих серверов. Регулярно запускайте на них `ps -axx`. Изучите, какие процессы обычно бывают запущены. Если тот или иной процесс не распознается, его надо исследовать.
- Обращайте внимание на открытые сетевые порты с помощью `netstat -na` и `sockstat`. Какие порты TCP и UDP должен «прослушивать» сервер? Если порт не распознается, исследуйте его. Наверное, это что-то безобидное, но это может быть и черным ходом злоумышленника.
- Необъяснимые неполадки в системе также являются подсказкой. Многие злоумышленники неуклюжи, и у них мало навыков системного администратора. Они затевают бессмысленные атаки и считают себя «крутыми».
- По-настоящему квалифицированные взломщики не только заметают следы, но и стараются не причинить вреда системе, чтобы не насторожить администратора. Поэтому необычайно высокая стабильность системы также должна настораживать.

- Перезагрузки, происходящие без видимой причины, могут быть сигналом установки нового ядра. Также они могут свидетельствовать о неполадках в аппаратном обеспечении и неверной конфигурации, поэтому в любом случае такие факты необходимо исследовать.
- FreeBSD отсылает вам каждый день электронные письма с информацией о состоянии системы. Читайте их. Храните их. Если что-то покажется подозрительным – исследуйте.

Существует два инструмента защиты, которые я особенно рекомендую для понимания работы системы. Первый инструмент – это */usr/ports/sysutils/lsof*, который перечисляет все файлы, открытые на компьютере. Чтение вывода lsof(8) – это обучение само по себе; возможно, вы и не догадывались, что на сервере запущено столько лишнего. Странные открытые файлы свидетельствуют либо о недостаточном понимании системы, либо о чьих-то нежелательных действиях.

Второй инструмент – */usr/ports/security/nessus*. Это автоматизированный сканер слабых мест в системе защиты. Аудит безопасности с помощью nessus(8) – это замечательный способ узнать, что может увидеть взломщик в вашей системе.

Если система взломана

Что делать, если система взломана? Простого ответа нет. На эту тему написаны толстенные книги. Однако вот несколько общих рекомендаций.

Прежде всего, взломанной системе нельзя доверять. Если злоумышленник получил доступ root к серверу Интернета, он может заменить любую программу в системе. Даже если закрыть «дыру», через которую он проник, в системе может остаться установленная им программа login(8), которая будет посылать имя регистрирующегося пользователя и пароль в какой-нибудь канал IRC. Не доверяйте такой системе. Ее обновление делу не поможет, поскольку даже sysinstall(8) и компилятор находятся под подозрением.

Несмотря на то, что такие инструменты, как rkhunter (*/usr/ports/security/rkhunter*), могли бы помочь проверить присутствие злоумышленника, ни один инструмент не сможет достоверно сказать, что злоумышленника *нет* на сервере. Не стесняйтесь написать по адресу *FreeBSD#security@FreeBSD.org* и попросить совет. Опишите ситуацию и объясните, почему вы думаете, что система взломана. Однако будьте готовы и к такому ответу: переустановите операционную систему с безопасных носителей (FTP или CD-ROM) и восстановите данные с резервной копии. Надеюсь, вы прочли главу 4?

Соблюдение приемов обеспечения безопасности поможет уменьшить вероятность взлома так же, как соблюдение правил дорожного движения уменьшает вероятность автомобильной аварии. Удачи.

10

Каталог /etc

Каталог */etc* содержит основные сведения о конфигурации, необходимые для загрузки UNIX-подобных систем. Сталкиваясь с незнакомой системой UNIX, я прежде всего присматриваюсь к */etc*. Кратчайшая дорога от младшего «админа» UNIX к админу-середнячку – чтение */etc*. Да, надо читать все его содержимое. Да, читать придется много. Но понимание */etc* равнозначно пониманию взаимосвязи всех компонентов системы. По мере профессионального роста администратор UNIX в любом случае постепенно осваивает эту информацию, поэтому, если добавить ее к багажу знаний в самом начале пути, путь будет легче.

Многие файлы */etc* обсуждаются в тех главах, к которым они имеют самое непосредственное отношение, например, */etc/services* рассматривается в главе 6, а */etc/fstab* – в главе 8. Кроме того, некоторые файлы просто исчезают или представляют исключительно исторический интерес. В этой главе речь пойдет о важных файлах, для рассмотрения которых нет более подходящего места.

Через разные версии UNIX

Различные системы UNIX используют разные файлы */etc*. В большинстве случаев эти файлы есть не что иное, как переименованные или перестроенные файлы из BSD. Например, в первый раз столкнувшись с IBM AIX, я стал искать файл */etc/fstab*, подобный эквивалентному файлу BSD. Его там не было. Однако недолгие поиски навели меня на */etc/filesystems*, который оказался переделанной версией */etc/fstab*, учитывающей специфику IBM. Очевидно, в компании IBM решили, что название *fstab*, являющееся сокращением от *filesystem table* (таблица файловых систем), будет вводить в заблуждение, и выбрали для него другое название. Знание, что данная информация содержится где-то в */etc*, и что каких-то файлов там, очевидно, нет, значительно сокращает время поиска.

Даже совершенно разные системы FreeBSD имеют почти идентичные каталоги */etc*. Хотя некоторые дополнительные программы добавляют сюда свои файлы[1], можно ожидать, что определенные файлы будут присутствовать в любой системе FreeBSD.

Не забывайте – каталог */etc* является сердцем системы FreeBSD, и изменения в файлах из этого каталога могут повредить или даже разрушить систему. Перед изменением любого файла из каталога */etc* освежите в памяти информацию о системе управления версиями (Revision Control System, RCS), почерпнутую в главе 4. Я настоятельно рекомендую создать каталог */etc/RCS* и после этого оттачивать мастерство использования ci(1) и co(1). Ведь хотя восстановление перепутанной таблицы файловых систем помогает превратить сведущего администратора в отличного, такой путь – один из наименее приятных для достижения этой цели.

/etc/adduser.conf

Этот файл позволяет определить значения по умолчанию, которые будут использоваться программой adduser при создании новых учетных записей. Более подробную информацию вы найдете в главе 7.

/etc/amd.map

В операционной системе FreeBSD имеется возможность автоматического монтирования и демонтирования файловых систем NFS по запросу через демона автоматического монтирования amd(8). Более подробную информацию вы найдете в страницах руководства.

/etc/bluetooth, /etc/bluetooth.device.conf и /etc/defaults/bluetooth.device.conf

Операционная система FreeBSD поддерживает технологию Bluetooth, ставшую стандартом для организации беспроводной связи на коротких расстояниях. В отличие от 802.11, технология Bluetooth предназначена для организации высокоуровневых сервисов, таких как голосовой обмен. В этой книге речь идет о серверах, поэтому мы не будем рассматривать технологию Bluetooth, но вы должны знать, что при желании ноутбук, работающий под управлением FreeBSD, может взаимодействовать с сотовым телефоном, обладающим поддержкой Bluetooth, и подключаться к Интернету.

---

[1] Есть также дополнительные программы, которые добавляют файлы конфигурации не в */etc*, а в */usr/local/etc*. – *Прим. научн. ред.*

/etc/crontab

Демон cron(8) дает пользователям возможность планировать запуск различных задач. Подробности и примеры использования вы найдете в главе 15.

/etc/csh.*

Файлы */etc/csh.** содержат значения по умолчанию для csh и tcsh, задаваемые в масштабе всей системы. Когда пользователь регистрируется с любым из этих процессоров, процессор выполняет все команды, найденные в */etc/csh.login*. Подобным образом при выходе пользователя из системы выполняется */etc/csh.logout*. В */etc/csh.cshrc* можно разместить общую информацию о конфигурации оболочки.

/etc/devd.conf

Демон devd(8) управляет съемным оборудованием, например устройствами USB, PCCard и Cardbus. Когда к ноутбуку подключается беспроводная сетевая карта, devd(8) замечает это событие и запускает соответствующие системные процессы для настройки карты в соответствии с параметрами, заданными в файле */etc/rc.conf*. Мы уже рассматривали devd(8) в главе 8, однако, если вам кажется, что в файл */etc/devd.conf* необходимо внести некоторые корректировки, скорее всего вы идете по неверному пути.

/etc/devfs.conf, /etc/devfs.rules и /etc/defaults/devfs.rules

Управление файлами устройств в системе FreeBSD производится с помощью devfs(5) – виртуальной файловой системы, которая динамически выполняет необходимые операции с файлами устройств во время загрузки, при подключении съемных устройств и при их отключении. Более подробную информацию вы найдете в главе 8.

/etc/dhclient.conf

Многие операционные системы предлагают лишь базовые средства конфигурирования клиента DHCP, которые не предоставляют возможности тонкой настройки или подгонки; либо вы используете то, что есть, либо не используете ничего. В большинстве случаев пустой файл клиента DHCP */etc/dhclient.conf* дает полную функциональность DHCP, однако он работает правильно не во всех ситуациях. Возможно, в вашей локальной сети есть неполадки или, когда вы находитесь на конференции, какой-нибудь малыш со скриптами решил забавы ради настроить второй сервер DHCP и направить весь трафик на свой ком-

пьютер, чтобы иметь возможность перехватывать пароли. Настройка конфигурации DHCP поможет преодолеть эти трудности.

Сервер лучше не настраивать через DHCP (если речь не идет о бездисковой станции), поэтому мы не будем углубляться в эту тему. Однако вы должны знать, что FreeBSD предоставляет возможность настраивать функциональность клиента DHCP.

/etc/disktab

Когда-то давно жесткие диски были редкими и экзотическими устройствами с незначительным числом разновидностей. В файле */etc/disktab* вы найдете низкоуровневые описания различных типов дисков, начиная от дискет емкостью 360 Кбайт до жесткого диска Panasonic, предназначенного для ноутбуков, емкостью 60 Мбайт. (Да, когда-то ноутбуки оснащались такими жесткими дисками, и мы были счастливы иметь их.)

В наши дни этот файл в основном используется при работе со съемными носителями, такими как дискеты емкостью 1.44 Мбайт и ZIP-диски. В главе 8 мы рассматривали процедуру форматирования стандартных дискет, но в данном файле содержится информация, необходимая для форматирования и других съемных носителей. Если у вас появится необходимость отформатировать диск LS 120 или ZIP-диск, в этом файле вы найдете требуемую вам метку в самом начале записи.

Изменение содержимого файла */etc/disktab* может потребоваться, только если у вас имеется несколько идентичных жестких дисков, которые должны иметь идентичный формат и деление на разделы. Если вам потребуется добавить свои собственные записи, ознакомьтесь со страницей руководства disktab(5).

/etc/freebsd-update.conf

Этот файл используется программой freebsd-update(8), когда производится установка на сервер новых версий программ. Более подробную информацию вы найдете в главе 13.

/etc/fstab

Обсуждение таблицы файловых систем, */etc/fstab*, вы найдете в главе 8.

/etc/ftp.*

Эти файлы используются демоном FTP – ftpd(8), чтобы определить, кто имеет право доступа к системе по протоколу FTP и что им доступно в случае успешного подключения. Подробные сведения можно найти в описании, представленном в главе 17.

/etc/group

Включение пользователей в состав групп очень подробно рассматривалось в главе 7.

/etc/hosts

Это жестко определенный список имен хостов с соответствующими им IP-адресами, о чем будет говориться в главе 14.

/etc/hosts.allow

Файл */etc/hosts.allow* управляет доступом к демонам, скомпилированным с поддержкой TCP Wrappers. Этот файл исключительно подробно рассмотрен в главе 9.

/etc/hosts.equiv

Файл */etc/hosts.equiv* используется для организации работы r-служб (rlogin, rsh и других) и позволяет доверенным удаленным системам подключаться и выполнять команды в локальной системе без предоставления пароля и даже без регистрации. Предполагается, что доверенные хосты выполняют аутентификацию пользователей самостоятельно, а значит, локальная система может не беспокоиться о повторной аутентификации.

Такое явное доверие очень удобно использовать в «дружественных» сетях; так же удобно, как оставить открытой дверь вашей квартиры на Манхеттене, чтобы не рыться в карманах в поисках ключа от дверного замка. В наши дни нет таких вещей, как «дружественная сеть». На самом деле с помощью этого сервиса любой рассерженный служащий может нарушить работу корпоративной сети, а машина с r-службами становится легкой добычей для первого малыша со скриптами, который на нее набредет. На самом деле */etc/hosts.equiv* и относящиеся к нему сервисы заставали врасплох даже превосходных экспертов по безопасности, которые считали, что вполне могут использовать их. Я рекомендую оставить этот файл пустым и даже сделать его неизменяемым (глава 9).

/etc/hosts.lpd

Файл */etc/hosts.lpd* – это один из самых простых файлов в */etc*. Хосты, перечисленные здесь, каждый в отдельной строке, могут отправлять задания на печать на принтеры, которыми управляет эта машина. Можно задавать имена хостов, но неполадки в DNS способны «перекрыть кислород» заданиям на печать, поэтому вместо имен следует указывать IP-адреса.

В отличие от многих других конфигурационных файлов UNIX, этот не допускает номера сетей и сетевые маски, здесь необходимо указывать отдельные имена хостов или IP-адреса.

/etc/inetd.conf

Демон inetd(8) обслуживает входящие сетевые подключения для меньших демонов, которые не запускаются часто. Более подробную информацию можно найти в разделе «Демон inetd» в главе 15.

/etc/localtime

В этом файле содержится информация о часовом поясе системы, которая была указана при использовании программы tzsetup(8). Это двоичный файл, и его нельзя редактировать обычными средствами. В действительности этот файл просто копируется программой tzsetup(8) из подкаталога */usr/share/zoneinfo*. При изменении часового пояса вам необходимо обновить FreeBSD, с тем чтобы получить новые файлы с информацией о часовых поясах, а затем перезапустить tzsetup(8), чтобы установить правильный часовой пояс.

/etc/locate.rc

Программа locate(1) отыскивает все файлы с заданным именем. Например, чтобы найти файл *locate.rc*, введите следующее:

```
# locate locate.rc
/etc/locate.rc
/usr/share/examples/etc/locate.rc
/usr/src/usr.bin/locate/locate/locate.rc
```

Можно увидеть, что *locate.rc* есть в трех местах: в основном каталоге */etc*, в каталоге с примерами системных файлов и в исходном коде системы.

Раз в неделю система FreeBSD сканирует свои диски, создает список всех найденных файлов и сохраняет его в базе данных. Программа, создающая такой список, locate.updatedb(8), использует значения по умолчанию, указанные в */etc/locate.rc*. Изменение переменных, которые перечислены ниже, позволяет управлять процессом создания базы данных locate:

- `TMPDIR` содержит путь к временному каталогу, который используется программой locate.updatedb(8) и по умолчанию имеет значение */tmp*. При недостатке дискового пространства в каталоге */tmp* с помощью этой переменной можно указать любой другой каталог.
- Местоположение базы можно изменить с помощью переменной `FCODES`. Однако это может оказать влияние на другие компоненты системы FreeBSD, которые предполагают наличие базы в местопо-

ложении по умолчанию, поэтому будьте готовы к странным результатам, особенно если старая база locate осталась на прежнем месте, в каталоге */var/db/locate.database*.

- В переменной SEARCHPATHS перечислен каждый каталог, в котором требуется проводить поиск. По умолчанию это каталог /, то есть весь диск. Чтобы индексировать только часть диска, укажите здесь конкретное значение.
- В переменной PRUNEPATHS перечислены каталоги, которые индексировать не надо. По умолчанию исключаются временные каталоги, которые обычно содержат только недолговечные файлы.
- Переменная FILESYSTEMS содержит список файловых систем, которые необходимо индексировать. По умолчанию locate.updatedb(8) индексирует только файловые системы UFS (FreeBSD) и ext2fs (Linux). Нежелательно включать в этот список сетевую файловую систему NFS (глава 8) – если все серверы начнут индексировать файловый сервер, сеть будет сильно перегружена.

/etc/login.*

С помощью файлов */etc/login.access* и */etc/login.conf* можно указать, кто имеет право входить в систему, и какие ресурсы должны быть доступны этим пользователям. Подробности приводятся в главе 7.

/etc/mail/mailer.conf

Файл */etc/mail/mailer.conf* позволяет указать почтовую программу, которая будет использоваться в системе, как описывается в главе 16.

/etc/make.conf

make – означает создание, или компиляцию программы на машинном языке из ее исходного кода. Этот процесс обсуждается в главе 11. Файл */etc/make.conf* управляет процессом сборки программного обеспечения и позволяет определять параметры, значения которых оказывают влияние на сборку программ. Запомните, все, что будет добавлено в файл *make.conf*, будет оказывать влияние на сборку всего программного обеспечения в системе, включая и обновления. Это может вызвать ошибки процесса обновления.[1] Многие параметры в *make.conf* представляют интерес только для разработчиков.

[1] Если при выполнении обновления системы в файле *make.conf* окажется всякий мусор, и вы попросите о помощи, вас могут поднять на смех. Однако коммерческие службы поддержки программного обеспечения не откажут вам в такой помощи, так что тут все в порядке.

Если вам желательно определять значения параметров сборки, которые оказывают влияние только на обновление системы, то вместо *make.conf* следует использовать файл */etc/src.conf*.

Ниже приводятся некоторые из наиболее часто используемых параметров в файле *make.conf*. Все определения должны следовать синтаксису, используемому программой make(1). В большинстве случаев следующие рекомендации являются наиболее оптимальными.

CFLAGS

Задает параметры оптимизации для сборки программ, не относящихся к ядру. Многие другие UNIX-подобные операционные системы предлагают компилировать программное обеспечение с определенными *флагами компилятора* (*compiler flags*), или `CFLAGS`. Такая практика не приветствуется в системе FreeBSD. Системные компоненты, требующие установки определенных флагов, уже имеют необходимые настройки в файлах конфигурации, а настройки для дополнительного программного обеспечения определяются в отдельных файлах. Хотя некоторые администраторы могут порекомендовать другие настройки `CFLAGS`, любые другие опции Проектом FreeBSD не поддерживаются.

Вообще код FreeBSD лучше всего компилируется без дополнительных параметров. Их применение лишь ухудшит производительность. Если при использовании тех или иных нестандартных флагов программа работает со сбоями, вернитесь к стандартным флагам и пересоберите программу.

COPTFLAGS

Параметры оптимизации `COPTFLAGS` используются только для сборки ядра. Повторюсь: установки, отличные от принятых по умолчанию, могут привести к созданию неработоспособного ядра.

CXXFLAGS

Параметр `CXXFLAGS` содержит флаги, которые используются при компиляции исходных текстов, написанных на языке программирования C++. Чтобы добавить в этот параметр инструкции в дополнение к тем, что указаны в параметрах сборки пакета, следует использовать оператор `+=`. Все, что раньше было сказано о `CFLAGS`, в равной степени применимо и к `CXXFLAGS`.

CPUTYPE=i686

Некоторые пакеты программного обеспечения могут быть оптимизированы под различные типы микропроцессоров (CPU). Если указать тип CPU в файле */etc/make.conf*, компилятор попытается оптимизировать программное обеспечение под указанный микропроцессор.

Это может положительно сказаться на производительности в случае обновления операционной системы из исходных кодов (глава 13). FreeBSD оптимизирует под конкретный процессор не только саму операционную систему, но и любое программное обеспечение сторонних производителей, которое будет устанавливаться впоследствии.

На отдельной машине всегда устанавливайте значение параметра `CPUTYPE` в соответствии с фактическим типом микропроцессора. Если сборка производится на одной машине, а затем через NFS экспортируются каталоги */usr/obj* и */usr/src* для других машин, то чтобы выполнять на них обновление, минуя этап компиляции, устанавливайте значение `CPUTYPE` равным самому слабому CPU всех систем.

Таблица 10.1. Типы микропроцессоров

Архитектура	Подходящие типы
архитектура i386, Intel	core2, core, nocona, pentium4m, pentium4, prescott, pentium3m, pentium3, pentium-m, pentium2, pentiumpro, pentium-mmx, pentium, i486, i386
архитектура i386, VIA	c3, c3-2
архитектура i386, AMD	opteron, athlon64, athlon-mp, athlon-xp, athlon-4, athlon-tbird, athlon, k8, k7, k6-2, k6, k5
архитектура amd64, Intel	Nocona, Prescott, core, core2
архитектура amd64, AMD	opteron, athlon64

INSTALL=install -C

По умолчанию при установке собранной программы система FreeBSD записывает новый исполняемый файл поверх старого. Параметр `install -C` заставляет инсталлятор install(1) сравнивать новую версию программы со старой. Если они идентичны, то новый исполняемый файл не устанавливается. Благодаря этому предотвращается бессмысленное изменение значения времени последнего изменения файлов в */usr/include*, и программа make(1) не будет считать, что программа устарела только потому, что заголовочные файлы были заменены более новыми, хотя и совершенно идентичными версиями. Кроме того, за счет этого можно также ускорить процесс обновления и сократить количество операций записи на диск. Последнее обычно не имеет большого значения, но при необходимости этим можно воспользоваться.

Примечание

В главе 11 мы будем рассматривать некоторые параметры *make.conf*, полезные для коллекции портов, а в главе 20 поговорим о некоторых необычных параметрах, которые помогут уменьшить объем операционной системы FreeBSD.

/etc/master.passwd

Этот файл содержит базовую информацию обо всех учетных записях пользователей, о чем рассказывалось в главе 7.

/etc/motd

Файл *motd*, или *message of the day* (*сообщение дня*), отображается на экранах пользователей, когда они входят в систему. В этом файле можно разместить системные извещения или другую информацию, которую должны увидеть пользователи, интерактивно входящие в систему (shell users). Заметим, что правом на просмотр этого файла управляет параметр `welcome` в */etc/login.conf*. В системе может быть несколько файлов сообщений – по одному для каждого login-класса.

/etc/mtree

mtree(1) строит иерархии каталогов с правами доступа в соответствии с предопределенным стандартом. Каталог */etc/mtree* хранит этот стандарт для базовой системы FreeBSD. Записи mtree используются в процессе обновления системы для обеспечения корректной установки программного обеспечения. В случае случайного изменения прав доступа к файлу или каталогу в системе можно будет с помощью mtree(1) восстановить их. В редактировании этих файлов нет необходимости, но знать, почему они здесь находятся, полезно.

/etc/namedb

Файлы */etc/namedb* управляют системным сервером имен. Работа этих файлов подробно описана в главе 14.

/etc/netstart

Этот сценарий на языке командной оболочки создан специально для активизации сетевой подсистемы в однопользовательском режиме. Наличие подключения к сети в однопользовательском режиме может оказаться чрезвычайно полезным по целому ряду причин, от монтирования файловых систем NFS до подключения к удаленным системам с целью проверки конфигурации. Достаточно просто запустить */etc/netstart*. Этот сценарий не оказывает никакого влияния на систему в многопользовательском режиме.

/etc/network.subr

Этот сценарий на языке командного интерпретатора не предназначен для использования человеком – он представляет собой библиотеку подпрограмм, используемых другими сценариями настройки сети.

/etc/newsyslog.conf

Этот файл настраивает ротацию и удаление файлов протоколов. Дополнительную информацию можно найти в главе 19.

/etc/nscd.conf

Программа nscd(8) кэширует результаты обращения к службе имен с целью оптимизации производительности системы. Подробности приводятся в главе 15.

/etc/nsmb.conf

Система монтирования разделяемых ресурсов Windows в операционной системе FreeBSD использует файл */etc/nsmb.conf* для определения параметров доступа к системам Windows, как описано в главе 8.

/etc/nsswitch.conf

Функция выбора службы имен (Name Service Switching) рассматривается в главе 14.

/etc/opie*

OPIE, или *One-time Passwords in Everything* (одноразовые пароли повсюду), – это система одноразовых паролей, производная от S/Key. Хотя эта система кое-где еще используется, тем не менее она больше не пользуется популярностью. Если вам интересно, то можете прочитать страницу руководства opie(4). По большей части система OPIE была вытеснена другими системами, такими как Kerberos.

/etc/pam.d/*

Подключаемые модули аутентификации (*Pluggable Authentication Modules, PAM*) позволяют системному администратору использовать различные системы аутентификации, авторизации и управления доступом. Если вы используете Kerberos или какую-нибудь другую централизованную систему аутентификации, вам придется столкнуться с PAM.

/etc/pccard_ether

Этот сценарий запускает и останавливает съемные сетевые карты, например сетевые карты с интерфейсом Cardbus или USB. Название этого файла – наследие прошлых лет, когда существовали только карты PC Cards. Обычно этот сценарий запускается демоном devd(8), когда в этом возникает необходимость, о чем уже рассказывалось в главе 8.

/etc/periodic.conf и /etc/defaults/periodic.conf

periodic(8) запускается каждый день для выполнения повседневных операций по обслуживанию системы. Это источник сообщений о состоянии, которые каждый день посылаются пользователю root. Эта программа просто выполняет различные сценарии командной оболочки, хранящиеся в */etc/periodic* и */usr/local/etc/periodic*. Каждый из этих сценариев включается или отключается в */etc/periodic.conf*.

periodic запускает программы ежедневно, еженедельно или ежемесячно. Для каждой группы программ есть свои значения параметров, например, программы, запускаемые ежедневно, конфигурируются отдельно от программ, запускаемых ежемесячно. Этими параметрами управляют записи в файле */etc/periodic.conf*. Хотя мы рассмотрим только примеры из сценариев, запускаемых ежедневно, тем не менее похожие настройки используются для сценариев, запускаемых раз в неделю или раз в месяц.

daily_output="root"

Если необходимо, чтобы результаты ежедневных проверок отправлялись пользователям, отличным от root, здесь следует перечислить этих пользователей. Если у вас нет пользователя, ответственного за чтение почты periodic, лучше всего оставить значение по умолчанию и перенаправлять почту пользователю root, чью почту вы читаете. В качестве альтернативы здесь можно указать полное имя файла, и тогда newsyslog(8) (глава 19) может осуществлять ротацию протокола periodic.

daily_show_success="YES"

Если для `daily_show_success` установлено значение `YES`, в ежедневные сообщения будет включена информация обо всех успешных проверках.

daily_show_info="YES"

Если для `daily_show_info` установлено значение `YES`, в ежедневные сообщения будет включена общая информация о запущенных командах.

daily_show_badconfig="NO"

В случае значения `YES` в ежедневное сообщение будет включена информация о командах periodic, выполнить которые не удалось. Обычно эти сообщения совершенно безобидны и имеют отношение к подсистемам, которые не поддерживаются или не входят в состав вашей системы.

daily_local="/etc/daily.local"

Вы можете определить свои сценарии, которые должны запускаться программой periodic(8) ежедневно, еженедельно или ежемесячно. По умолчанию эти сценарии должны располагаться в каталогах */etc/*

daily.local, */etc/weekly.local* и */etc/monthly.local*, но вы можете указать любые другие каталоги.

Каждый из сценариев, предназначенных для запуска ежедневно, еженедельно или ежемесячно, размещенных в подкаталогах */etc/periodic*, имеет краткое описание в самом начале файла, и для большинства из них имеются параметры в файле */etc/defaults/periodic.conf*. Обратите на них внимание, возможно, там вы найдете то, что вас заинтересует. Параметры, которые по умолчанию включены, подходят в большинстве случаев, однако в некоторых системах можно включить в */etc/periodic.conf* и дополнительную информацию. Например, если вы используете диски на базе использования функциональных особенностей GEOM, для вас могут оказаться полезными ежедневные отчеты GEOM о состоянии. Все, что я здесь перечислю, может устареть еще до передачи рукописи в редакцию, не говоря уже о времени, которое пройдет до момента получения книги вами. Поэтому я не буду вдаваться в подробности работы различных сценариев.

/etc/pf.conf

Основы работы с фильтром пакетов PF рассматривались в главе 9.

/etc/pf.os

Фильтр пакетов PF может идентифицировать тип операционной системы по передаваемым ею пакетам, что позволяет создавать, например, такие правила: «Пользователям FreeBSD показать мою настоящую домашнюю страницу, а пользователям Windows показать страницу, где они смогут приобрести настоящую операционную систему». За дополнительной информацией обращайтесь к странице руководства pf.os(5).

/etc/phones

Те, кто пользуется модемом, могут хранить в этом файле номера телефонов для удаленных модемов, чтобы вместо полного номера можно было набирать, например, `home`. Этот файл используется только программами tip(1) и cu(1), поэтому этот файл не так полезен, как могло бы показаться.

/etc/portsnap.conf

Программа portsnap обеспечивает обновление дерево «портов», о чем будет рассказываться в главе 11.

/etc/ppp

Поддержка модемов в операционной системе FreeBSD обеспечивается с помощью ppp(8). За более подробной информацией обращайтесь к страницам руководства.

/etc/printcap

Это файл содержит информацию о настройках принтера. Вывод на печать в UNIX-подобных системах может оказаться делом далеко не простым, особенно если учесть, какое разнообразие принтеров существует. Однако настройка процесса передачи заданий серверу печати в FreeBSD совсем не сложна. Эта тема будет рассматриваться в главе 15.

/etc/profile

Файл */etc/profile* содержит учетную информацию, задаваемую по умолчанию для интепретатора */bin/sh*, подобно тому, как файлы */etc/csh.** хранят информацию для пользователей csh/tcsh. При каждом входе в систему пользователи */bin/sh* наследуют значения, задаваемые в этом файле. Однако пользователь может подменить эти значения своим файлом *.profile*. Bash и другие модификации sh также используют этот файл

Хотя tcsh – это стандартный командный процессор FreeBSD, также довольно популярны sh и его модификации (особенно bash). Стоит задуматься о синхронизации настроек в */etc/profile* и */etc/csh.login*, чтобы избежать проблем в будущем.

/etc/protocols

В главе 6 были кратко рассмотрены сетевые протоколы. Как объясняется в этой главе, файл */etc/protocols* содержит список протоколов Интернета, с которыми вы можете встретиться.

/etc/rc*

Когда бы система ни загружалась, в определенный момент, когда уже можно выполнять команды userland, она запускает сценарий */etc/rc*. Этот сценарий командной оболочки монтирует все файловые системы, «поднимает» сетевые интерфейсы, конфигурирует devfs(5), устанавливает разделяемые библиотеки и выполняет все остальные задачи, необходимые для загрузки системы.

В различных системах могут решаться во время загрузки совершенно различные задачи. Сервер терминалов, оснащенный тремя платами с 48 последовательными портами на каждой, работает совершенно не так, как веб-сервер. Вместо единого сценария */etc/rc*, выполняющего все операции, в некоторых системах стартовые задачи распределяются между сценариями поменьше, что позволяет создавать сценарии для решения конкретных задач.

Дополнительно можно найти несколько сценариев непосредственно в каталоге */etc*, такие как */etc/rc.firewall* и */etc/rc.initdiskless*. Эти сценарии появились задолго до появления системы запуска rcNG и были

оставлены, чтобы не терять совместимость с устаревшим программным обеспечением.

Система запуска FreeBSD рассматривалась в главе 3.

/etc/remote

Этот файл содержит параметры настройки в машинном виде для подключения к удаленным системам через последовательные линии связи. В наши дни этот файл может представлять интерес, только если ваша система используется, например, как клиент, и вы собираетесь подключить ее к последовательной консоли. Последовательные консоли будут рассматриваться в главе 20.

/etc/rpc

Удаленный вызов процедур (*Remote Procedure Call, RPC*) – это метод выполнения команд на удаленном компьютере. Как и стек протоколов TCP/IP, RPC имеет службы и номера портов. Файл */etc/rpc* содержит список служб с соответствующими им номерами портов. Наиболее широко RPC используется сетевой файловой системой (NFS), которая рассматривалась в главе 8.

/etc/security/

Этот каталог содержит конфигурационную информацию для утилиты системы безопасности audit(8).

/etc/services

Этот файл содержит список служб с соответствующими им номерами портов TCP/IP. Файл */etc/services* подробно рассматривался в главе 6.

/etc/shells

Этот файл содержит список командных процессоров, доступных пользователю, как обсуждалось в главе 7.

/etc/snmpd.config

FreeBSD включает в себя базовую реализацию SNMP, которая будет рассматриваться в главе 19.

/etc/src.conf

Этот файл содержит инструкции по сборке FreeBSD из исходных текстов. Он используется как аналог файла *make.conf* для единственного

дерева исходного кода. Значения, установленные в файле */etc/make.conf*, также оказывают влияние на сборку FreeBSD из исходных текстов. Разница в том, что */etc/src.conf* оказывает влияние только на сборку FreeBSD, но не на «порты» и пакеты.

/etc/sysctl.conf

Этот файл содержит информацию о том, какие параметры sysctls ядра устанавливаются в процессе начальной загрузки. См. главу 5.

/etc/syslog.conf

Этот файл определяет, какие данные будут протоколироваться системой, и где файлы протоколов будут размещаться. Подробности – в главе 19.

/etc/termcap

Этот файл содержит настройки и характеристики различных типов терминалов. Давным-давно, когда терминалы выпускались в великом разнообразии типов и производители добавляли новые модели чуть ли не ежедневно, этот файл являлся жизненно необходимой частью системы. Однако теперь, когда почти все пришли к единому стандарту vt100, конфигурация по умолчанию подходит практически для всех.

/etc/ttys

Этот файл содержит информацию обо всех терминальных устройствах (окнах, содержащих приглашение командной строки). Имя файлу было дано еще в те времена, когда терминалы представляли собой физические мониторы. В наше время в большинстве случаев используются виртуальные терминалы, созданные командами telnet или ssh. Мы будем использовать этот файл для организации входа в систему через последовательное устройство в главе 20.

11

Делаем систему полезной

В таких операционных системах, как Microsoft Windows и Red Hat Linux, базовая установка подразумевает инсталляцию всех компонентов, которые когда-либо могут понадобиться пользователю. В системах BSD это не так: компоненты устанавливаются разрозненно, и в этом есть свое преимущество. Например, в системах Windows в основном системном каталоге размещаются тысячи объектов, причем большая часть из них – это разделяемые библиотеки. Когда бы система ни запускалась, большая часть этих библиотек загружается в системную память. Я не знаю, для чего предназначена каждая библиотека, но уверен, что никогда не буду использовать многие из них. Единственные программы, которые я применяю, – SSH и Firefox. Все остальные программы только поглощают оперативную память. Да, именно так Microsoft разрабатывает операционные системы – пользователям надо дать все, что только имеется, то есть все от первой до последней мелочи. При установке Red Hat Linux инсталлируется программное обеспечение примерно того же объема, но оно хотя бы не загружается в память автоматически при загрузке системы, и пользователь может удалить ненужные ему программы.

При установке BSD инсталлируется столько программ, *сколько требуется* для запуска системы, плюс дополнительные утилиты, традиционно включаемые в состав систем UNIX. Программа установки предлагает выбор: инсталлировать дополнительные программы и исходный код – или нет. Однако даже полная система BSD занимает намного меньше дискового пространства, чем Windows или Red Hat Linux. По сравнению с Windows при полной установке FreeBSD инсталлируется намного меньше компонентов. Система Windows, поддерживающая только пакеты SSH и Firefox, была бы намного меньше и проще – совсем как FreeBSD.

Преимущество такой разбивки в том, что можно устанавливать лишь необходимые компоненты системы. При этом существенно упрощается

поиск и устранение неполадок, а ненужные разделяемые библиотеки не вызовут сбоев в системе. Обратная сторона медали: придется определить, что именно вам требуется, и установить дополнительные программы, обеспечивающие эту функциональность. Однако разработчики FreeBSD позаботились о том, чтобы максимально упростить установку программного обеспечения.

Сборка программного обеспечения

Сборка программного обеспечения – сложный процесс, поскольку исходный код нужно особым образом обработать, чтобы созданные исполняемые файлы были работоспособными – и работали бы правильно! Программисты могли бы включать в свои программы подробные инструкции по установке, например: `Теперь наберите ar cru .libs/lib2O_zlib_plugin.a istream-zlib.o zlib-plugin.o`, но это был бы явный садизм с их стороны. Администраторы UNIX сами склонны к садизму, но они категорически против насилия, когда оно направлено против них, – если что-то может быть автоматизировано, то оно будет автоматизировано.

Основной инструмент сборки программного обеспечения – утилита make(1). В текущем каталоге она ищет файл с именем *Makefile*, полный жутких команд, подобных примеру из предыдущего абзаца. Программа make(1) читает команды и выполняет их, автоматизируя процесс установки независимо от степени его сложности. На самом деле знать начинку *Makefile* вовсе не обязательно, поэтому здесь этот файл не анализируется.

Каждый *Makefile* включает в себя различные *целевые объекты* (*targets*), или наборы исполняемых инструкций. Например, команда `make install` ищет в *Makefile* целевой объект (процедуру) с именем *install*. Если такая процедура найдена, она выполняется. Имя целевого объекта обычно говорит о процедуре, которая выполняется этим объектом. Так, совершенно очевидно, что команда `make install` производит установку программного обеспечения. В файле *Makefile* можно найти целевые объекты для установки, настройки и удаления программного продукта. Программа make(1) может выполнять огромное количество функций, в том числе таких, о которых создатели даже не предполагали. Но в этом и прелесть UNIX!

Исходный код и программное обеспечение

Исходный код – это понятные человеку инструкции, из которых создается собственно машинный код, то есть программа. Возможно, вы уже встречали исходный код в той или иной форме. Чтобы увидеть его, достаточно открыть любой файл из каталога */usr/src*. Читать исходный код вам не обязательно, но уметь распознавать его необходимо в двух случаях из трех.

Имеющийся исходный код программы вы можете собрать (или *скомпилировать*) в системе, в которой она будет выполняться. (Выстраивание программного обеспечения на иной платформе с помощью так называемой «кросс-компиляции» потребовало бы от вас гораздо большего объема знаний о сборке программ, да это и не всегда возможно.) Если программа написана для операционной системы, в значительной степени сходной с той платформой, на которой вы ее компилируете, то компиляция будет успешной. Если же ваша платформа существенно отличается от оригинальной, программа не будет скомпилирована. После успешной сборки программного обеспечения на целевой платформе получившуюся программу (то есть *двоичный исполняемый файл*, или *binary*) можно копировать на другие идентичные платформы – она должна работать и там.

Некоторые программы написаны настолько хорошо, что их можно компилировать на множестве различных платформ. В отдельные программы специально включена поддержка платформ, значительно отличающихся друг от друга. Например, веб-сервер Apache можно компилировать как в Windows, так и в UNIX, – для этого достаточно набрать `make install`. Впрочем, это нетипичный случай, свидетельствующий о поистине героических усилиях разработчиков данного программного обеспечения.

Вообще говоря, если удается собрать программу из исходного кода, то она будет выполняться. Достаточно опытный администратор, изучив исходный код и сообщения об ошибках, может сказать, почему программа не будет собрана или выполнена. В большинстве случаев проблема проста, и обойти ее можно с минимальными усилиями. Это одна из причин, объясняющих важность доступа к исходному коду.

Раньше каждый администратор UNIX был программистом, а отладка поглощала значительную часть его времени. Каждая UNIX-система немного отличалась от других, поэтому каждый администратор должен был знать свою платформу и платформу, для которой программа была написана, а также отличия этих платформ. Лишь в этом случае можно было рассчитывать на успешный запуск данной программы. Приходилось затрачивать двойные усилия, что было просто ужасно.

Со временем были созданы инструменты, облегчающие кросс-компилирование, такие как autoconf. Но эти инструменты применялись не во всех программах, поэтому при сбоях администратор начинал все с нуля. Системным администраторам приходилось редактировать исходный код и файлы *Makefile*, чтобы обеспечить хотя бы вероятность того, что программа запустится. Но *вероятность того, что программа запустится*, это совсем не то же самое, что *устойчивая работа* программы, не говоря уже о *правильной* ее работе.

Для упрощения процесса сборки программ в системе FreeBSD была создана коллекция «портов».

Система «портов» и пакетов

«Порты» (*ports*) – это команды для компиляции программ во FreeBSD, а *пакеты* – это предварительно скомпилированные «порты».

Пакеты устанавливаются быстрее и проще. «Порты» устанавливаются медленнее, но их можно настроить для конкретного окружения. В целом система называется *коллекция «портов»* (*Ports Collection*), *дерево «портов»* (*ports tree*), или просто «*порты*» (*ports*). Все эти термины относятся непосредственно к «портам» – системе сборки портов и пакетов.

Основная идея системы «портов» и пакетов очень проста: если программу необходимо модифицировать для запуска в операционной системе FreeBSD, то эти модификации надо автоматизировать. Если для сборки данной программы из исходного кода или для ее работы требуется другое программное обеспечение, следует это записать и отслеживать. Для автоматизации внесения изменений полезно иметь перечень всех файлов, входящих в состав программы, чтобы затем легко инсталлировать и деинсталлировать ее. Так как процесс сборки программного обеспечения каждый раз выдает один и тот же результат, и у вас записано, какие файлы создаются процессом, то можно скопировать все исполняемые файлы и установить их на любой похожей системе FreeBSD.

«Порты»

«Порт» – это набор команд, задающих правила наложения *заплаток* (*patches*) на файлы исходного кода. Комбинируя заплатки и команды установки, FreeBSD может вести полный учет всего процесса установки программного обеспечения. Такой подход избавляет от трудностей установки программ и позволяет сконцентрироваться на их конфигурировании.

Установка дерева «портов»

Если вы следовали инструкциям по установке, представленным в главе 2, то дерево «портов» установлено в */usr/ports*. В этом каталоге должны находиться несколько файлов и пара десятков каталогов. Если у вас в каталоге */usr/ports* нет ничего, то вы не сможете следовать за примерами этой главы. Ничего страшного – вам просто нужно установить дерево «портов». В главе 13 мы будем рассматривать программу portsnap(8), но уже сейчас с ее помощью можно установить дерево «портов»:

```
# portsnap fetch
Looking up portsnap.FreeBSD.org mirrors... 3 mirrors found.
Fetching public key from portsnap3.FreeBSD.org... done.
Fetching snapshot tag from portsnap3.FreeBSD.org... done.
Fetching snapshot metadata... done.
Fetching snapshot generated at Thu May 15 20:09:15 EDT 2008:
a2b71859b1a44878d19f879e2d1c801d785761670cc745 5% of 47 MB 29 kBps 26m32s
(Перевод: Поиск зеркал portsnap.FreeBSD.org... найдено 3 зеркала.
Получение открытого ключа от portsnap3.FreeBSD.org... выполнено.
```

```
Получение тега копии из portsnap3.FreeBSD.org... выполнено.
Получение метаданных копии... выполнено.
Получение копии, созданной во вторник 15 мая в 20:09:15 2008 года:
a2b71859b1a44878d19f879e2d1c801d785761670cc745 5% из 47 MB 29 kBps 26м32с
)
```

Программа portsnap отыскивает зеркала, где размещаются копии дерева «портов», проверяет целостность файлов на сервере, загружает файлы и проверяет целостность загруженных файлов. После загрузки установка дерева «портов» производится командой:

```
# portsnap extract
```

В результате будет получено текущее дерево со всеми последними «портами» FreeBSD.

Содержимое дерева «портов»

Большинство каталогов, которые вы здесь увидите, соответствует категориям программных продуктов. В каждом каталоге есть подкаталоги, вмещающие отдельные программные продукты. К моменту написания этих строк FreeBSD содержала почти 17 000 «портов», поэтому важно уметь пользоваться деревом каталогов и понимать разделение ПО на категории. Ниже описаны файлы и каталоги, которые не относятся к категориям программных продуктов.

В файле *CHANGES* перечислены изменения, произведенные в инфраструктуре «портов» FreeBSD. В основном этот файл используют разработчики «портов» и те, кто интересуется внутренним устройством коллекции «портов».

Файл *COPYRIGHT* содержит информацию о лицензировании для коллекции «портов» в целом. У каждой отдельной программы из коллекции «портов» есть собственная информация об авторских правах и лицензировании, но сама коллекция «портов» распространяется на основе лицензии BSD.

Файл *GIDs* содержит перечень всех числовых групповых идентификаторов (GID), используемых программным обеспечением в коллекции «портов». Многие программные продукты работают с непривилегированными учетными записями, поэтому в коллекции «портов» должна быть информация о числовых идентификаторах групп для этих учетных записей. Данный файл позволяет предотвратить возможные конфликты между программными продуктами, так как каждому порту присвоено свое значение GID в файле *GIDs*.

Файл *KNOBS* содержит перечень всех параметров настройки, доступных в коллекции «портов». Эти настройки можно выполнить из командной строки или указать в файле */etc/make.conf*, чтобы активизировать соответствующие им функциональные возможности программного обеспечения, если таковые поддерживаются. Подробнее об этих настройках мы поговорим ниже в этой же главе.

В файле *LEGAL* можно увидеть список всех правовых ограничений в программном обеспечении, включенном в коллекцию «портов». Некоторые программные продукты накладывают определенные ограничения, например оговаривается их некоммерческое использование, ограничение на распространение, ограничение на получение денежной выгоды и т. д. Эти ограничения также перечисляются в отдельных «портах», а данный список – всего лишь сборник ограничений из всех «портов».

В файле *MOVED* находится список «портов», которые были перемещены из одной категории в другую. Так как команда разработчиков «портов» FreeBSD периодически создает новые категории, они могут перемещать «порты» из одной категории в другую. Для поиска перемещенных «портов» необходимо использовать такие средства автоматизированного управления, как portmaster(8).

Файл *Makefile* содержит высокоуровневые инструкции для всей коллекции «портов» в целом.

Подкаталог *Mk* содержит подробные низкоуровневые инструкции для всей коллекции «портов» в целом. Многие программы предполагают совместную установку, и эти файлы гарантируют, что части одного набора инструментов будут собраны и установлены совместимым образом. Например, окружения рабочего стола KDE и GNOME состоят из десятков и сотен небольших программ, и каждая из них должна быть собрана так, чтобы она могла поддерживать возможность взаимодействия с другими программами. Если заглянуть в этот каталог, можно увидеть такие файлы, как *bsd.gnome.mk* и *bsd.kde.mk*, предназначенные для настройки этих программ, а также файлы для Apache, Emacs, GCC, Perl и многих других разновидностей программных продуктов. Если вы действительно хотите изучить работу коллекции «портов», исследуйте этот каталог.

Файл *README* содержит описание верхнего уровня организации коллекции «портов».

Каталог *Templates* содержит заготовки файлов, используемые другими разделами коллекции «портов».

Каталог *Tools* содержит программы, сценарии и другие средства автоматизации, которые в основном используются разработчиками «портов».

Файл *UIDs* содержит перечень всех числовых идентификаторов пользователей (UID), используемых программным обеспечением в коллекции «портов». Как и файл *GIDs*, он позволяет разработчикам «портов» предотвратить конфликты между непривилегированными учетными записями, с которыми работает программное обеспечение, устанавливаемое из «портов».

Файл *UPDATING* содержит примечания, используемые при обновлении программного обеспечения. Здесь перечислены обновления, требующие специального вмешательства, отсортированные по дате в обратном порядке. Этот файл будет рассматриваться в главе 13.

Каталог *distfiles* содержит исходный код программного обеспечения, устанавливаемого из «портов». Исходный код, загружаемый при установке «порта», сохраняется в каталоге */usr/ports/distfiles*.

Все остальные каталоги хранят программное обеспечение «портов» по категориям. Ниже показано содержимое «порта» arabic, где находится программное обеспечение поддержки арабского языка. Большая часть программного обеспечения в «портах» поддерживает арабский язык, но программное обеспечение данной категории предназначено исключительно для поддержки арабского языка – например, шрифтов, перевода некоторых документов и т. д. Мало кому понадобится эта категория, но она удобна в качестве примера благодаря скромным размерам. Некоторые категории «портов» содержат сотни элементов.[1]

```
Makefile       ae_fonts_ttf    kacst_fonts     khotot          php_doc
Makefile.inc   arabtex         katoob          koffice-i18n
ae_fonts_mono  aspell          kde3-i18n       libitl
```

Файл *Makefile* содержит инструкции для сборки всех «портов» в каталоге. Файл *Makefile.inc* содержит метаинструкции для «портов» в этом каталоге. Все остальные каталоги – это отдельные пакеты программного обеспечения. Мы займемся исследованием этих каталогов в разделе «Применение „портов“».

Поиск программного обеспечения

Некоторые категории содержат сотни «портов». Как же найти в них *что-либо*? Файл */usr/ports/INDEX-7* содержит список всех «портов» в алфавитном порядке. Это индекс «портов», в котором каждый «порт» описан в отдельной строке, а поля разделены символами вертикальной черты (|). Такой формат очень удобен для различных системных инструментов, но не для чтения. Запустите команду `make print-index` в каталоге */usr/ports*, чтобы получить более длинный, но удобный для чтения индекс. Этот индекс заполнен записями, как показано ниже:

```
Port:   p5-Compress-Bzip2-2.09
Path:   /usr/ports/archivers/p5-Compress-Bzip2
Info:   Perl5 interface to bzip2 compression library
Maint:  demon@FreeBSD.org
Index:  archivers perl5
B-deps: perl-5.8.8
R-deps: perl-5.8.8
E-deps: perl-5.8.8
P-deps: perl-5.8.8
F-deps:
WWW:    http://search.cpan.org/dist/Compress-Bzip2/
```

[1] 17 000 «портов», 60 с лишним категорий. В некоторых категориях – до 11 элементов. Считайте сами.

Индекс начинается с имени «порта» и полного пути к каталогу «порта». Поле `Info` содержит краткое описание порта. В поле `Maint` находится список лиц или групп, взявших на себя ответственность за поддержку и интеграцию данного программного обеспечения в коллекцию «портов». В поле `Index` перечислены категории, где этот «порт» может быть зарегистрирован. В поле `B-deps` перечислены зависимости, то есть программное обеспечение, которое должно быть установлено до выполнения сборки этого «порта». Некоторые программные компоненты должны быть извлечены или разархивированы определенными инструментами, список которых приводится в поле `E-deps`. В поле `P-deps` перечислены все зависимости, необходимые для наложения исправлений, – изредка исправления должны накладываться с помощью определенных инструментов. Поле `F-deps` определяет *зависимости получения* (*fetch dependencies*), то есть специальное программное обеспечение, с помощью которого должен загружаться исходный код программы. Наконец, поле `WWW` содержит адрес домашней страницы программы.

Поиск по имени

Знать содержимое индекса – это прекрасно, но как с его помощью найти ту или иную программу? Если точное имя программного пакета известно, можно выполнить поиск с помощью команды `make search`. Ниже приводятся результаты поиска для net-snmp:

```
# cd /usr/ports
# make search name=net-snmp
❶ Port: net-snmp-5.3.1_3
Path:   /usr/ports/net-mgmt/net-snmp
Info:   An extendable SNMP implementation
Maint:  kuriyama@FreeBSD.org
B-deps: autoconf-2.59_2 libtool-1.5.22_4 m4-1.4.8_1 openssl-0.9.8e perl-5.8.8
R-deps: openssl-0.9.8e perl-5.8.8
WWW:    http://net-snmp.sourceforge.net/

Port:   p5-Net-SNMP-5.2.0
...
Port:   p5-Net-SNMP-365-3.65
...
```

К моменту написания этих строк во FreeBSD имелось три «порта», имена которых содержали строку `net-snmp`. Один из них – собственно набор программных средств net-snmp ❶; два других – это библиотеки Perl, позволяющие использовать в программах SNMP и никак не связанные с net-snmp. Поля в описании были получены непосредственно из файла *INDEX*, который мы рассматривали выше.

Если вам не требуется такое подробное описание в результатах поисков, можно воспользоваться командой `make quicksearch`.

Однако не все программные компоненты можно отыскать с помощью этой команды. Например, можно попробовать найти популярный файловый менеджер Midnight Commander с помощью такой команды:

```
# make search name=midnight
#
```

Да, ничего не вышло. Попробуем выполнить более общий поиск.

Поиск по ключевому слову

Если точное имя программы неизвестно, попробуйте выполнить поиск по ключевому слову. В этом случае будет просканировано больше полей и возвращено больше совпадений. Однако если искать обычное слово, то по ключевому слову можно получить слишком много информации.

```
# make search key=midnight
```

Такая команда вернет все «порты» со строкой `midnight` в полях с описанием, именем или зависимостями. Вы быстро поймете, что Midnight Commander можно найти в каталоге */usr/ports/misc/mc*.

Другие способы просмотра коллекции «портов»

Если вы предпочитаете работать с веб-броузером, соберите индекс в формате HTML. Перейдите в */usr/ports* и как root наберите `make readmes` для создания файла *README.html* с содержимым дерева «портов». В нем можно щелкать мышью на различных категориях и даже просматривать подробное описание каждого «порта».

Если ни один из этих вариантов не дал результатов, попробуйте обратиться к сервису поиска FreeBSD Ports Tree по адресу *http://www.freebsd.org/cgi/ports.cgi*. Кроме того, имеется и другая, очень неплохая служба поиска FreshPorts, расположенная по адресу *http://www.freshports.com*.

Веб-броузер и поисковая машина помогут вам найти нужную программу.

Правовые ограничения

Хотя большинство программ в коллекции «портов» открыты для некоммерческого использования, у некоторых из них есть более строгие лицензионные ограничения. В файле */usr/ports/LEGAL* перечислены правовые ограничения для содержимого коллекции «портов». Наиболее типичное ограничение – запрет на передачу третьим лицам. Проект FreeBSD не включает такие программы в дистрибутивы, поставляемые на компакт-дисках, и не помещает их на свои FTP-серверы, но предоставляет инструкции по их сборке. Например, довольно долго FreeBSD не имела лицензии на Java. Проекту не было разрешено распространять исходный код Java или скомпилированные исполняемые файлы. Впрочем, можно было распространять инструкции по сборке исходного кода Java компании Sun. Пользователь FreeBSD, желающий установить Java, мог зайти на веб-страницу Sun Microsystems, загрузить исходный код Java и собрать свою версию Java на FreeBSD.

В наше время проект FreeBSD Foundation распространяет лицензионный пакет Java, который можно быстро установить.

Для некоторых программ также запрещено коммерческое использование либо включение в состав коммерческих продуктов. Некоторые программы нельзя экспортировать из Соединенных Штатов в соответствии с правилами Министерства торговли, ограничивающими экспорт криптографических программных продуктов.[1] Если вы собираете системы FreeBSD для последующей продажи (раздачи), экспорта или коммерческого использования, вам определенно следует изучить этот файл.

К счастью, большая часть программного обеспечения в коллекции «портов» открыта как для коммерческого, так и для некоммерческого использования. Пакеты с ограничениями – это исключение из правил.

Применение пакетов

Пакеты – это программы из коллекции «портов», уже скомпилированные для определенной версии FreeBSD. Сначала мы обсудим именно пакеты, поскольку, как правило, работать с ними легче, чем с «портами». После рассмотрения пакетов обратимся к «портам».

Установка программы как пакета может сэкономить много времени, потому что оно не будет тратиться на компиляцию исходного кода. Если на распространение программы в компилированном виде не наложены правовые ограничения, то она доступна как пакет. Другие программы, такие как Adobe Acrobat, доступны только в компилированном виде. Пакеты доступны на компакт-дисках или через FTP и называются так же, как соответствующие им «порты».

Пакеты на компакт-дисках

В комплекте компакт-дисков FreeBSD есть довольно внушительная коллекция скомпилированных пакетов. Для их использования нужно лишь смонтировать CD и прочитать файл пакета. Если у вас имеются образы CD, еще не записанные на физический носитель, можно смонтировать эти образы и устанавливать пакеты из образов. (Порядок работы со съемными носителями и монтирования образов дисков подробно описан в главе 8.)

После монтирования CD загляните в каталог *packages*. Ниже приведено содержимое каталога с самого свежего установочного компакт-диска FreeBSD:

```
# cd /cdrom/packages/
# ls
```

[1] Большинство этих программ доступны из источников за пределами США – их можно загрузить откуда угодно. Кроме того, бывшие шифровальщики КГБ с удовольствием снабдят всех желающих надежными алгоритмами шифрования по самым низким ценам без всяких ограничений.

```
❶ All          emulators       linux           textproc        x11-servers
INDEX          graphics        perl5           x11
devel          lang            print           x11-fonts
```

Листинг кажется знакомым. Да, он такой же, как и листинг каталога */usr/ports*. Пакеты – это всего лишь скомпилированные «порты», поэтому они хранятся в тех же каталогах-категориях, что и «порты», из которых они собраны. На одном компакт-диске недостаточно места для хранения всех 17 000 пакетов FreeBSD. Установочные компакт-диски содержат только основные пакеты, востребованные практически в любой системе, такие как X Window System и Perl. Второй образ компакт-диска FreeBSD содержит более полный набор пакетов. Например, ниже приведено содержимое каталога *x11* с установочного компакт-диска FreeBSD 6.2.

```
# ls x11
libdrm-2.0.2.tbz              xorg-documents-6.9.0.tbz        xterm-220.tbz
xorg-6.9.0.tbz                xorg-libraries-6.9.0.tbz
xorg-clients-6.9.0_3.tbz      xorg-manpages-6.9.0.tbz
```

В дереве «портов» категория x11 содержит 342 «порта». На установочном компакт-диске их всего 7, но это лишь основные компоненты X Window System. Дерево портов содержит, например, инструменты настройки сенсорной панели (touchpad) Synaptics в GNOME, которые важны для тех, у кого имеется сенсорная панель и кто использует GNOME, но не настолько ценны, чтобы занимать место на компакт-диске.

Интересное отличие между деревом «портов» и деревом пакетов заключается в каталоге *All* ❶. Этот каталог содержит все пакеты, присутствующие на компакт-диске. В других каталогах присутствуют лишь ссылки на фактические файлы в каталоге *All*.

Чтобы узнать назначение пакета, можно выполнить поиск в каталоге */usr/ports*, как мы делали это ранее в этой главе, когда искали «порты» по именам. Следует, однако, заметить, что в каталоге *packages* имеется файл *INDEX*, содержащий описания пакетов, присутствующих только на этом диске. Вы не можете воспользоваться функцией `make search name=`, как в случае с «портами», но можете произвести поиск непосредственно по файлу *INDEX* с применением утилиты grep(1). Пример поиска пакета libdrm:

```
# cd /cdrom/packages
# grep ^libdrm INDEX
libdrm-2.0.2|/usr/ports/graphics/libdrm|/usr/local|❶Userspace interface to
kernel Direct Rendering Module services|/usr/ports/graphics/libdrm/pkg-descr|
x11@FreeBSD.org|graphics x11|||http://dri.freedesktop.org||||1
#
```

Для тех, кто мало знаком с утилитой grep(1), поясню, что здесь выполняется поиск строки `libdrm` в файле *INDEX*. Символ «крышки» (^) указывает, что последовательность символов `libdrm` должна находиться

в начале строки. Файл *INDEX* содержит всю информацию, извлекаемую командой `make search`, но в менее удобочитаемом формате. Символ вертикальной черты (`|`) служит разделителем полей: четвертое поле ❶ содержит описание пакета.

Пакеты на сервере FTP

Зачастую пакет не представлен на CD, потому что пространство в наборе компакт-дисков Проекта FreeBSD ограниченно и не может вместить больше 17 000 пакетов. Кроме того, программное обеспечение на CD собрано для определенного «выпуска» FreeBSD. Если вы установили FreeBSD 7.1, затем обновили ее до версии 7.3 и теперь хотите установить пакеты для версии 7.3, то компакт-диск с пакетами для версии 7.1 не поможет. Кроме того, похожие проблемы будут наблюдаться после обновления до версии -stable или -current (глава 13).

Если пакет отсутствует на компакт-диске, его можно установить только через FTP. Проект FreeBSD предоставляет возможность установить через FTP пакеты практически для всех 17 000 с лишним элементов коллекции «портов». Любое FTP-зеркало FreeBSD содержит пакеты для самых последних выпусков FreeBSD, а на некоторых зеркалах можно найти пакеты для старых (и даже очень старых) выпусков. О выборе наиболее подходящего вам зеркала говорилось в главе 2. Каждое официальное FTP-зеркало FreeBSD хранит пакеты по адресу:

```
ftp://зеркало.freebsd.org/pub/freebsd/ports/архитектура/packages-версия/
```

Например, пакеты FreeBSD 6.2 для архитектуры i386 можно найти по адресу *ftp://ftp.freebsd.org/pub/freebsd/ports/i386/packages-6.2-release* и на любых других серверах в том же самом каталоге. Если заглянуть на FTP-сайт, можно увидеть все категории в каталоге */usr/ports* и многие другие. В коллекции «портов» каждый «порт» помещается в какую-нибудь категорию, однако некоторые «порты» запросто могут попасть сразу в несколько категорий. Например, «порты» Perl SNMP можно классифицировать и как средства управления сетью, и как программы Perl, в зависимости от выбранной точки зрения. На серверах FTP достаточно места, чтобы хранить «порты» во всех возможных категориях, что упрощает и ускоряет их поиск. Чем быстрее вы найдете искомый «порт» и покинете зеркало, тем легче жизнь у администраторов зеркала.

Так же, как и на компакт-диске, на каждом FTP-сайте имеется собственный каталог *All*, где хранятся фактические файлы пакетов. Однако, в отличие от CD, на FTP-сайте можно найти тысячи и тысячи пакетов!

Установка пакетов

Если файл пакета находится на локальном диске, установить его можно с помощью pkg_add(1):

```
# pkg_add xorg-6.9.0.tbz
```

Программа pkg_add(1) извлечет и проверит сжатый файл пакета и установит его в соответствии с инструкциями, находящимися в пакете. В большинстве случаев pkg_add(1) не выводит никаких сообщений и по окончании просто возвращает вас к приглашению командной строки. Иногда, например, если требуемый пакет уже установлен, программа установки выведет сообщение. Не оставляйте без внимания такие сообщения, так как в них содержатся рекомендации по обеспечению корректной работы программного обеспечения.

Если программа pkg_add(1) обнаружит, что пакет имеет зависимости – другие программы, необходимые для корректной работы пакета, – она постарается установить и эти пакеты из того же источника. Набор компакт-дисков FreeBSD собран с учетом этого обстоятельства. Например, пакету xorg, показанному выше, требуется несколько необходимых ему программ, и все они присутствуют на установочных компакт-дисках. Если нужный пакет отсутствует, pkg_add сообщит об этом и прервет установку. В этом случае можно отыскать этот пакет на другом диске и установить его или просто установить недостающий пакет через FTP.

Кроме того, программа pkg_add(1) поддерживает возможность автоматической установки через FTP. Ключ `-r` предписывает выйти в Интернет и загрузить недостающие пакеты с FTP-сервера FreeBSD.

```
# pkg_add -r xorg
```

Преимущество такого подхода: система автоматически найдет подходящий сервер FTP, загрузит подходящую версию пакета и все программы, от которых он зависит, а затем установит их. Обратная сторона медали: если процесс установки пакета сорвется, система уничтожит все загруженные файлы пакетов. Предотвратить уничтожение файлов пакетов в текущем каталоге можно с помощью ключа `-K`.

Можно также вручную скачать пакеты с сайта FTP и установить их из командной строки. При таком способе зависимости не устанавливаются автоматически, но при попытке установить пакет программа pkg_add(1) выведет полный список всех необходимых, но отсутствующих пакетов. Вы можете загрузить их и установить удаленно. Этот способ наиболее полезен в тех случаях, когда вы находитесь за брандмауэром и должны приложить дополнительные усилия для загрузки файлов.

В случае больших пакетов я часто использую комбинированный подход. Например, пакет OpenOffice.org имеет размер более 100 Мбайт и требует предварительной установки нескольких небольших пакетов. Я загружаю OOo с сервера FTP и затем пытаюсь установить его. Программа pkg_add(1) прерывает установку из-за отсутствия необходимых программ. Тогда я устанавливаю эти дополнительные пакеты командой `pkg_add -r` и снова пытаюсь установить OOo из загруженного ранее пакета.

pkg_add(1) настройка окружения

Поведением программы pkg_add(1) можно управлять с помощью переменных окружения. Переменные окружения определяют, куда распаковываются пакеты, откуда они загружаются, другие каталоги в системе, где могут находиться пакеты и т. д. Эти переменные окружения лучше устанавливать в сценарии входа в систему, чтобы использовать их согласованно.

Ниже описаны наиболее полезные настройки среды окружения для программы pkg_add(1).

PKG_TMPDIR

Переменная окружения PKG_TMPDIR задает каталог, в который распаковываются временные файлы. Пакет – это архив tarball с файлами программ и дополнительными инструкциями по установке. Чтобы установить пакет, его нужно разархивировать (untar). Если в стандартных каталогах мало места, то разархивирование не будет завершено, и установка прервется. По умолчанию программа pkg_add(1) обращается к каталогу, задаваемому переменной окружения TMPDIR. Если эта переменная не определена, pkg_add ищет свободное место в таком порядке: */tmp*, */var/tmp* и */usr/tmp*. С помощью переменной PKG_TMPDIR можно задать другой каталог, где хватает свободного места:

```
# setenv PKG_TMPDIR /home/mwlucas/garbage
```

PACKAGEROOT

По умолчанию pkg_add -r пытается скачивать пакеты с *ftp://ftp.freebsd.org*. Однако, скорее всего, это далеко не лучший выбор для вас. Основное зеркало FreeBSD часто сильно перегружено и может находиться далеко от вашего компьютера. Нередко можно добиться большей производительности, выбрав более близкое и менее нагруженное зеркало. Переменная окружения PACKAGEROOT предписывает программе pkg_add(1) использовать другой сервер FTP. Присвойте переменной PACKAGEROOT протокол и имя конкретного сервера (без полного пути). Например, для загрузки пакетов с *ftp5.us.freebsd.org* введите:

```
# setenv PACKAGEROOT ftp://ftp5.us.freebsd.org
```

PACKAGESITE

Это полный путь к хранилищу (репозитарию) пакетов. Применяется, если требуется использовать пакеты определенного «выпуска» или в системе есть локальное хранилище пакетов. (Установка локального хранилища пакетов рассматривается ниже в этой же главе.) Присвоим переменной PACKAGESITE значение в виде абсолютного URL, например, для устаревшей и неподдерживаемой версии FreeBSD:

```
# setenv PACKAGESITE ftp://ftp-archive.freebsd.org/pub/FreeBSD-Archive/
oldreleases/i386/5.4-RELEASE/packages/All
```

PKGDIR

Этот каталог определяет место для размещения копий пакетов, загруженных командой `pkg_add -Kr`, и позволяет организовать хранение загруженных пакетов.

```
# setenv PKGDIR /usr/ports/packages/All
```

Что устанавливает пакет?

Теперь, когда программа установлена, как найти ее в системе? В конце концов, здесь нет меню Пуск! Беспокоиться не о чем. Для получения полного списка установленных программ обратитесь к */var/db/pkg*. Подкаталоги этого каталога соответствуют всем «портам» или пакетам, установленным в системе, и содержат списки всех программных компонентов. К примеру, теперь, после установки пакета xorg 6.9, у нас появился каталог */var/db/pkg/xorg-6.9.0*. В нем находятся следующие файлы:

```
# ls /var/db/pkg/xorg-6.9.0/
+COMMENT       +CONTENTS       +DESC     +MTREE_DIRS
```

Файл *+COMMENT* содержит краткое описание пакета, а файл *+DESC* – более подробное описание. Файл *+MTREE_DIRS* содержит описание пакета в формате mtree(1). Особый интерес представляет файл *+CONTENTS*, в котором перечислены все файлы, установленные пакетом, все пакеты зависимостей и все инструкции по удалению пакета. (Теперь, когда пакет установлен, инструкции по установке больше не нужны, но инструкции по удалению могут пригодиться, если вы вдруг решите удалить пакет.)

```
# more /var/db/pkg/xorg-6.9.0/+CONTENTS
❶ @comment PKG_FORMAT_REVISION:1.1
❷ @name xorg-6.9.0
❸ @comment ORIGIN:x11/xorg
❹ @cwd /usr/local
❺ @pkgdep expat-2.0.0_1
❻ @comment DEPORIGIN:textproc/expat2
@pkgdep libdrm-2.0.2
@comment DEPORIGIN:graphics/libdrm
...
```

Первая строка – это номер версии формата ❶, в котором хранится запись о пакете. Если FreeBSD изменит формат хранения пакетов, инструменты управления пакетами смогут определить по этому номеру, как обслуживать пакет. Далее следует имя пакета ❷ и затем определение `ORIGIN` ❸, которое указывает – к какой категории относится пакет и в каком каталоге его можно отыскать. Вслед за меткой `cwd` ❹ указано место, куда нужно установить файлы пакета. Все пути к файлам указываются относительно этого каталога. Комментарий `pkgdep` ❺ указывает дополнительные пакеты, от которых зависит данный пакет. В данном

случае – это пакет expat-2. Здесь также указывается каталог ❻ в дереве «портов», где этот дополнительный пакет находится.

Однако пакет xorg является чем-то вроде «обманки» – при установке этого пакета не устанавливаются никакие файлы! В операционной системе FreeBSD пакет xorg присутствует только для описания зависимостей, необходимых для X Window System. Единственная запись, которую можно обнаружить в файле *+CONTENTS*, это ссылка на другие пакеты. Давайте рассмотрим содержимое файла *+CONTENTS* другого пакета, который действительно включает набор файлов. Вот содержимое пакета */usr/ports/archivers/zip*:

```
# cd /var/db/pkg/zip-2.32
# more +CONTENTS
...
❶ @cwd /usr/local
❷ man/man1/zip.1.gz
❸ @comment MD5:1c3dc2c955ff2e3d9a6b38b1f6150ca9
❹ @unexec rm -f %D/man/cat1/zip.1 %D/man/cat1/zip.1.gz
❺ bin/zip
@comment MD5:af81366669c5e1cd1fca37ea5fd70d62
bin/zipcloak
...
```

Здесь мы снова видим рабочий каталог ❶, но за ним следует собственно файл ❷. Объединив имя каталога и имя файла, можно определить, что при установке пакета zip был установлен файл */usr/local/man/man1/zip.1.gz*. Это страница руководства, с которой стоит ознакомиться. Далее следует комментарий, содержащий контрольную сумму MD5 ❸ этого файла и инструкция ❹ по удалению файла на случай, если потребуется удалить пакет. Однако этот пакет содержит не только страницу руководства, в состав пакета входит также двоичный файл ❺. Вы уже наверняка поняли, что утилита zip(1) была установлена в виде файла */usr/local/bin/zip*. При просмотре файла *+CONTENTS* можно определить имена и местоположение всех установленных файлов.

Основную информацию о файлах и каталогах можно получить с помощью программы pkg_info(1), но часто гораздо проще отыскать необходимую информацию самостоятельно.

Деинсталляция пакетов

Для деинсталляции пакетов применяется pkg_delete(1):

```
# pkg_delete xorg-6.9.0
```

Можно в качестве имени пакета указать путь к каталогу в базе данных пакетов – например, следующая команда отработает ничуть не хуже:

```
# pkg_delete /var/db/pkg/xorg-6.9.0
```

При деинсталляции пакета не производится деинсталляция пакетов, от которых он зависит. Так, поскольку пакет xorg является всего

лишь коллекцией зависимостей, которые должны быть удовлетворены для установки X Window System, деинсталляция в действительности не приводит к удалению каких-либо файлов. Каждую из зависимостей нужно деинсталлировать отдельно.

Деинсталляция пакета, необходимого для работы других пакетов, чревата проблемами. Например, пакет xorg требует установки expat, libdrm, xorg-libraries и многих других пакетов. Деинсталляция любого из них без деинсталляции пакета xorg приведет к нарушениям в работе пакета xorg и почти всех остальных программных компонентов, связанных с ним. Практически любой пакет наверняка перестанет работать, если удалить программные компоненты, от которых он зависит. Для принудительной деинсталляции пакета можно применить `pkg_delete -f`. Программа pkg_delete(1) выведет предупреждение, но так или иначе свою работу выполнит.

Информация о пакете

Устанавливая программное обеспечение, вы обычно знаете, что оно делает и какие зависимости были установлены. Однако большая часть этой информации наверняка забудется, как только программа заработает, а спустя несколько недель или месяцев вы вообще ничего не сможете вспомнить. Лично я радуюсь, если мне удается вспомнить хотя бы названия пакетов, не говоря об их версиях.

В состав операционной системы FreeBSD входит утилита pkg_info(1) – более удобный инструмент изучения установленных пакетов, чем ручное сканирование */var/db/pkg*. В своей работе программа pkg_info(1) опирается на содержимое */var/db/pkg*, однако при этом автоматически выполняет поиск информации и ее сортировку. Если запустить pkg_info(1) без дополнительных параметров, она перечислит все установленные пакеты с кратким описанием:

```
# pkg_info
915resolution-0.5.2_1,1 Resolution tool for Intel i915 video cards
ORBit-0.5.17_3      High-performance CORBA ORB with support for the C language
ORBit2-2.14.7       High-performance CORBA ORB with support for the C language
OpenSSH-askpass-1.2.2.2001.02.24 Graphical password applet for entering SSH
passphrase
Xaw3d-1.5E_1         A 3-D Athena Widget set that looks like Motif

(Перевод: 915resolution-0.5.2_1,1 - инструмент настройки разрешения экрана
для видеокарт Intel i915
ORBit-0.5.17_3 - высокопроизводительный компонент CORBA ORB с поддержкой
языка C
ORBit2-2.14.7 - высокопроизводительный компонент CORBA ORB с поддержкой языка C
OpenSSH-askpass-1.2.2.2001.02.24 - графический апплет для ввода пароля SSH
Xaw3d-1.5E_1 - набор трехмерных графических элементов управления Athena
Widget, внешне похожих на компоненты Motif)
...
```

Ого! В моем ноутбуке масса всякой всячины. Эта утилита поможет, если мне вздумается почистить систему и понадобится краткий перечень, благодаря которому я смогу решить, что удалить, а что оставить.

Другие аргументы pkg_info(1)

Программа pkg_info(1) может предоставлять самую разнообразную информацию обо всех установленных пакетах, от подробного описания до сценария деинсталляции пакета. При использовании аргументов pkg_info(1) нужно указывать либо имя пакета, либо ключ `-a`, который означает *for all packages* (*для всех пакетов*). Например, чтобы выяснить, каким пакетам в системе требуются другие пакеты, можно использовать команду `pkg_info -aR`:

```
# pkg_info -aR
Information for 915resolution-0.5.2_1,1:

Information for ORBit-0.5.17_3:

Required by:
gnome-libs-1.4.2_6
gnomecanvas-0.22.0_5
gnome-print-0.37_3
...
```

Здесь видно, что пакет 915resolution стоит особняком: ему ничего не требуется, и сам он не нужен никакому другому пакету. Если этот пакет вам не нужен, его можно спокойно удалить. С другой стороны, пакет ORBit востребован многими другими пакетами, поэтому его нельзя удалить просто так.

Чтобы определить объем дискового пространства, необходимый для файлов того или иного пакета, следует использовать ключ `-s`. Заметим, что здесь будут учитываться только файлы, установленные пакетом, а файлы, созданные при его работе, – это совсем другое дело. В конце концов, мы ведь не считаем MP3-файлы частью проигрывателя!

Другой типичный вопрос – к какому пакету относится тот или иной файл. Просматривая */usr/local/bin* в субботу вечером[1], я неоднократно натыкался на неизвестный файл, которого никогда не использовал и в котором, скорее всего, не нуждался. Ключ `-W` команды pkg_info позволяет выполнить «реверсивный поиск» файлов и определить, из каких пакетов они произошли.

```
# pkg_info -W /usr/local/bin/gtail
/usr/local/bin/gtail was installed by package coreutils-6.7
```

[1] Я мог бы найти на выходные занятие и повеселей чтения каталога */usr/local/bin*. Но жене не нравится, когда я весь вечер экспериментирую с микроволновкой, засовывая туда что попало, так что пришлось.

Проблемы с пакетами

Систему пакетов FreeBSD можно назвать великолепной. Почти. С этой схемой связано несколько проблем, особенно с процессом «портирования» программного обеспечения и его синхронизации.

Подавляющее большинство пакетов производят сторонние организации, выпуская свои программы по расписанию, совершенно независимому от FreeBSD. При выходе обновленной версии их программного продукта те, кто отвечает за сопровождение «порта», обновляют и соответствующий «порт» FreeBSD, но не сразу. Маленький «порт», пользующийся большой популярностью, можно обновить за считанные часы, а на обновление крупного и редко используемого программного обеспечения может уйти несколько дней или недель. Огромные программные пакеты с большим числом зависимостей, такие как GNOME или KDE, могут не обновляться несколько недель, в течение которых проводится тестирование. Многие пользователи полагают, что вслед за выходом их любимого инструмента тут же должно следовать обновление пакета FreeBSD для данного программного продукта, но это технически невозможно. Ребята из FreeBSD не торопятся включить в «порты» программное обеспечение, в котором наверняка есть ошибки, причем не только из-за ответственного подхода к работе, но и потому, что именно на их головы обрушится поток претензий к работе программ. Если некоторая задержка позволяет лучше удовлетворить пользователей, то не стоит спешить.

Кроме того, пакеты взаимозависимы; для надлежащей работы многих из них необходимы другие пакеты. Когда команда, поддерживающая «порты» FreeBSD, изменяет пакет, это изменение отзывается во всех зависимых пакетах. Вот почему можно увидеть пакеты с такими именами, как sudo-1.6.8.12_1. Эта программа называется sudo, номер версии – 1.6.8.12, а 1 означает номер версии «порта» FreeBSD. Это вторая версия «порта» для sudo 1.6.8.12 (первая версия не имеет номера). «Порт» немного изменился, возможно, потому что изменился список зависимостей или метод сборки. Зачастую эти изменения исключительно внутренние, не влияющие на работу программы или ее производительность.

Если вы применяете пакеты, относящиеся только к одному «выпуску» FreeBSD, взаимозависимость не настолько важна. В конце концов, пакеты, собранные для «выпуска», не изменяются. Возможно, вы запускаете предшествующую версию FreeBSD, но хотите выполнять программу, которая только что появилась. Можно постоянно обновлять операционную систему и при этом иметь старые версии программных продуктов.

Например, пакету OpenOffice-2.2 необходим пакет libiconv-1.9.2_2. Если пакет libiconv-1.9.1 уже установлен, программа pkg_add отметит, что установлены не все требуемые пакеты. Возможно, она все равно установит программу, но эта программа может работать некорректно –

в зависимости от изменений в самой программе. Самое лучшее, что можно предпринять, – это обновить libiconv перед установкой пакета.

Один из способов обойти эту проблему заключается в том, чтобы всегда применять пакеты с одной и той же датой или временем. Если переменная окружения PACKAGESITE указывает на каталог пакетов для определенного «выпуска» FreeBSD, пакеты всегда будут согласованными. Во многих случаях это совершенно приемлемо, поскольку вовсе не обязательно устанавливать последнюю версию той или иной программы. Я полагаю, что Emacs 19 работает ничуть не хуже, чем Emacs 21, поэтому различия между Emacs 21.3 и Emacs 21.1 меня вообще не волнуют.

Конечно, это означает отказ от установки обновлений, вызванных проблемами безопасности в пакетах. Для ноутбука такой подход в принципе оправдан, но для сервера он совершенно не годится. В таком случае я рекомендую применять «порты», а не пакеты. Вместо проверки установленных программ по имени пакета «порты» проверяют наличие самой программы. Продолжая предыдущий пример, установка OpenOffice.org из «портов» означает, что сборка OOo будет производиться с учетом версии библиотеки libiconv, уже установленной в системе, при условии, что эта версия поддерживает все необходимые функциональные возможности. Такой подход делает «порты» гораздо более гибкими.

Применение «портов»

Сборка программного обеспечения из «портов» отнимает больше времени по сравнению с применением пакетов. Кроме того, системе «портов» необходимо устойчивое соединение с Интернетом. Тем не менее система «портов» может дать лучшие результаты, чем система пакетов. Давайте рассмотрим не просто произвольный «порт», а, пожалуй, самый крупный в операционной системе FreeBSD – OpenOffice.org, который находится в каталоге */usr/ports/editors/openoffice.org-2*. Этот программный продукт содержит текстовый процессор, электронную таблицу, инструмент создания презентаций и другие компоненты, что делает его серьезным конкурентом Microsoft Office. Каталог содержит:

```
Makefile        distinfo        files           pkg-descr       pkg-plist
```

Файл *Makefile* содержит базовые команды для сборки этого «порта». Заглянув в этот файл, мы видим всего несколько сотен строк. Немного для такого сложного программного продукта. Большую часть этого файла занимают настройки, используемые достаточно редко. Здесь почти нет информации о самом OOo и чуть больше о сборке программного обеспечения в системе FreeBSD. (Большинство *Makefile* «портов» FreeBSD расположены в каталоге */usr/ports/Mk*; их редактирование – это высший пилотаж, и прежде чем на него решиться, надо хорошо освоить утилиту make(1).)

Файл *distinfo* содержит контрольные суммы для различных файлов, загружаемых в ходе установки «порта», благодаря чему можно быть

уверенным, что файлы будут загружены без ошибок и никто не сможет изменить содержимое файла до того, как вы его получите.

Каталог *files* содержит все дополнительные файлы и исправления, необходимые для сборки этого «порта». Это очень массивный пакет офисных программ, в нем больше сотни мегабайт исходного кода и всего 18 заплаток. Большинство этих заплаток просто обеспечивают интеграцию в систему пакетов FreeBSD и фактически не требуются для сборки OpenOffice.org в операционной системе FreeBSD.

Файл *pkg-descr* содержит подробное описание программного продукта.

Наконец, в файле *pkg-plist* находится список всех файлов, устанавливаемых «портом» (опись). При установке из «порта» устанавливаются только файлы, перечисленные в этой описи.

Вместе эти файлы составляют набор инструментов и инструкций, необходимых для сборки программного продукта.

Установка «порта»

Имея представление об исходном коде, вы быстро заметите, что в «порте» нет исходного кода как такового. Конечно, есть заплатки на исходный код и сценарии для его обработки, но самого исходного кода нет! Возникает справедливый вопрос: как эта схема работает без исходного кода?

При активизации «порта» FreeBSD автоматически загружает нужный исходный код с соответствующего сайта в Интернете. После этого загруженный код проверяется на наличие ошибок целостности и извлекается в рабочий каталог. Затем на него накладываются заплатки, выполняется сборка программы, установка всего необходимого и регистрация установки в */var/db/pkg*. Если у «порта» есть неустановленные зависимости, то процесс make прервется на сборку этих зависимостей, а затем возобновится. Чтобы инициировать все это, войдите в каталог «порта» и введите такую команду:

```
# make install
```

После этого по экрану пробежит множество строк текста, и по завершении процесса установки вы вернетесь к приглашению командной строки.

Такой целостный процесс установки учитывает все изменения в зависимостях. Если «порту» необходима другая программа, то он просто «закроет глаза» на минимальные изменения в этой программе. Например, если изменился API или ABI требуемой программы, «порт» лишь проверит это, а сам пакет может оказаться неработоспособным.

Однако по мере приобретения опыта в работе с исходным кодом вы убедитесь, что целостный подход вовсе не обязательно применять всегда. Беспокоиться не о чем – система «портов» предоставляет возможность организовать процесс сборки «порта» по усмотрению пользователя,

поскольку `make install` на самом деле запускает серию подчиненных команд. Если указать одну из таких команд, make(1) запустит все предыдущие команды, а также ту команду, которую вы указали. Например, в случае команды `make extract` будут выполнены команды `make config`, `make fetch`, `make checksum`, `make depends` и `make extract`. Рассмотрим эти команды по порядку.

make config

Многие «порты» включают дополнительные параметры настройки. Команда `make config` позволяет указать параметры, которые должны быть учтены при установке программного обеспечения из «порта». Информация о выбранных параметрах сохраняется для использования при последующей сборке «порта». Эти параметры влияют на сборку «порта» – например, если указать, что программа должна собираться с поддержкой SNMP, скорее всего, это приведет к добавлению зависимости от программного обеспечения SNMP где-нибудь в дереве «портов».

make fetch

После настройки «порта» система отыскивает предопределенный список сайтов Интернета, откуда можно получить исходный код. Файл *Makefile* «порта» содержит адрес доверенного сайта, откуда можно загрузить файл. Если исходный код не найден на этом сайте, проверяются резервные сайты из списка, имеющегося в системе «портов». Первоначальный файл с исходным кодом называется *distfile* и сохраняется в каталоге */usr/ports/distfiles*.

Если для загрузки distfile требуется какая-то специальная программа, система «портов» выполнит установку этой программы как часть этого этапа.

make checksum

На следующем этапе `make checksum` подтверждает, что цифровая подпись файла distfile соответствует подписи из файла *distinfo* «порта». Это мера безопасности – если сервер FTP был взломан злоумышленником, а исходный код поврежден или заменен на троян, это обнаружится именно здесь. Процесс сборки будет остановлен с сообщением о несоответствии контрольной суммы. Совершенно неважно, был ли исходный код поврежден в процессе загрузки или неизвестный злоумышленник добавил зловредный код в distfile до его загрузки, – в любом случае нет смысла тратить время на сборку такого «порта» и уж тем более не стоит устанавливать его!

make extract

Получив distfiles «порта», их следует развернуть и извлечь. Большинство файлов исходного кода объединяются с помощью tar(1) в архив tarball, который затем сжимается с помощью gzip(1) либо bzip(1). Од-

«Как прострелить себе ногу», способ № 839: игнорировать контрольную сумму

Разработчики программного обеспечения, особенно бесплатного, любят слегка подправить свой код, не изменив при этом номер версии или имя файла исходного кода distfile. После этого «порт» FreeBSD может и не работать. Если вы уверены, что distfile не был подменен или поврежден, и хотите использовать его несмотря на предупреждение, замените эту команду на `make NO_CHECKSUM=YES install`.

Настоятельно советую не делать этого, пока автор не подтвердит, что все в порядке. Запросив подтверждение у автора программного продукта, вы сможете быть уверены, что не установите зловредный программный код, и лишний раз напомните автору о важности соблюдения порядка нумерации версий.

нако некоторые дистрибутивы сжимаются другими архиваторами (например Zip, compress(1) и т. д.). Эта команда создает *рабочий* подкаталог и извлекает в него содержимое архива. Если для извлечения файлов из distfile требуется какая-то отдельная программа, она будет установлена.

make patch

Эта команда накладывает на исходный код в *рабочем* каталоге все заплатки, имеющиеся в составе «порта». Если для наложения заплаток требуется какая-то другая программа, отличная от patch(1), она будет установлена.

make depends

На этапе `make depends` проверяется зависимость «порта» от любых других программ. Если такие программы есть, то проверяется, установлены ли они. Так, для оконного менеджера X необходим сервер X.org. Если программы, от которых зависит «порт», не установлены, система рекурсивно обойдет все зависимости и установит их.

make configure

Далее FreeBSD проверяет необходимость запуска сценария *configure*. Это отдельный этап, отличный от `make config`. Если в составе программного продукта поставляется собственный сценарий *configure*, он выполняется. Некоторые «порты» приостанавливают процедуру сборки на этом этапе, чтобы запросить дополнительную информацию, но в большинстве случаев сборка продолжается без остановки.

make build

На этом этапе происходит компиляция проверенного, извлеченного и исправленного программного кода.

make install

Наконец, на этапе `make install` устанавливается программа, а ее наличие регистрируется в */var/db/pkg*.

Интегрированные настройки «портов»

Многие программные пакеты обладают широчайшими возможностями настройки параметров сборки. Настроить эти параметры для каждого программного пакета в отдельности не так сложно, но универсального метода, определяющего их сразу для всех пакетов, нет. В одном случае может потребоваться редактировать *Makefile*, в другом эти параметры могут настраиваться с помощью ключей сценария *configure*. Изучение принципов внесения таких изменений требует времени и может оказаться скучным занятием. Система «портов» FreeBSD предлагает два способа настройки этих параметров в системе.

Самый новый и более предпочтительный метод основан на команде `make config`. Она выводит на экран диалоговое окно, похожее на то, что вы уже видели в процессе установки FreeBSD. Например, популярный программный продукт Snort, позволяющий обнаружить несанкционированное вторжение в систему, включает поддержку ведения протоколов в различных базах данных, интеграцию с динамическими правилами сетевой защиты и т. д. Если перейти в каталог */usr/ports/security/snort* и ввести команду `make config`, на экране появится меню, (рис. 11.1).

Для выбора параметра служит клавиша пробела, для перемещения – стрелки и клавиша табуляции. Чтобы завершить работу с диалогом, нажмите клавишу Enter, предварительно выбрав кнопку OK или Cancel (отмена). Выбранные значения будут записаны в файл */var/db/ports/<имя_порта>/options*, чтобы можно было использовать их при обнов-

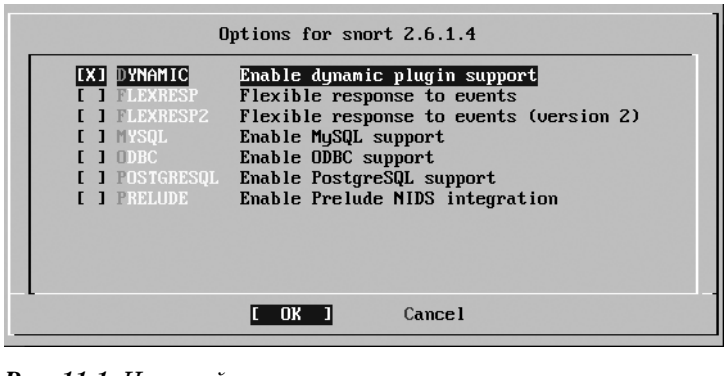

Рис. 11.1. *Настройка параметров «порта»*

лении или повторной сборке программы. Snort – отличный пример применения этого способа настройки, потому что у этой программы много часто используемых параметров. Многие администраторы протоколируют события Snort с помощью различных баз данных.

Однако этот удобный графический интерфейс выбора параметров появился в системе FreeBSD совсем недавно и работает не во всех «портах». Многие «порты» по-прежнему объявляют о дополнительных возможностях при первом запуске команды `make install`. Например, при запуске процедуры сборки OpenOffice.org на экран выводится следующее сообщение:

```
OPTIONS:

You can compile OOo with debug symbols with WITH_DEBUG=1
(Перевод: При компиляции OOo с параметром WITH_DEBUG=1 будет добавлена
отладочная информация)

If you set WITH_DEBUG=2, you add internal
OOo debug support.
(Перевод: При установке параметра WITH_DEBUG=2 будет добавлена поддержка
встроенного отладчика OOo.)

You can compile OOo without Mozilla connectivity with
make -DWITHOUT_MOZILLA
(Перевод: OOo можно скомпилировать без поддержки Mozilla, собрав пакет
с параметром -DWITHOUT_MOZILLA)
...
```

На самом деле список доступных параметров гораздо длиннее, но идею вы поняли. Если, увидев это объявление, вы желаете воспользоваться одной из предоставляемых особенностей, нажмите Ctrl-C, чтобы прервать сборку. После этого можно установить выбранные параметры в командной строке:

```
# make WITH_DEBUG=1 install
```

Данная команда изменяет ход сборки «порта», включая поддержку отладочной информации, что повысит качество отчетов об ошибках, которые вы сможете отправлять группе разработчиков проекта OOo. Это одно из больших преимуществ «портов» перед пакетами. Выполнить подобную настройку в пакетах невозможно.

Однако не все «порты» сами предлагают все доступные параметры настройки. Если вам действительно любопытно узнать о них, загляните в файл *Makefile* «порта». Давайте вместе посмотрим, что находится внутри *Makefile*.

Файлы Makefile «портов»

В начале файла *Makefile* можно найти множество строк, описывающих «порт»:

```
PORTNAME=       ❶apache
```

```
PORTVERSION=    ❷2.2.4
PORTREVISION=   ❸2
CATEGORIES=     ❹www
MASTER_SITES=   ❺${MASTER_SITE_APACHE_HTTPD} \
                ${MASTER_SITE_LOCAL:S/%SUBDIR%\//clement\/:aprmysql/}
DISTNAME=       ❻httpd-${PORTVERSION}
DISTFILES=      ❼${DISTNAME}.tar.bz2 \
                apr_dbd_mysql.rev-57.c:aprmysql
```

Это «порт» с именем apache ❶, соответствующий версии 2.2.4 ❷ этой программы. Это вторая версия ❸ «порта» для данной версии программы, поэтому в */var/db/pkg* он называется apache-2.2.4_2. В системе FreeBSD этот порт хранится в категории */var/ports/www* ❹. Переменная MASTER_SITES обычно содержит список интернет-сайтов, откуда можно загрузить программу. Дистрибутив веб-сервера Apache хранится на множестве зеркал, и во многих «портах» указывается их список. Чтобы не обновлять десятки и сотни «портов» всякий раз, когда появляется или исчезает какое-либо зеркало, система «портов» определяет внутреннюю переменную ❺, которая содержит список зеркал, чтобы обновлять его в одном месте. DISTNAME содержит базовую часть имени файла дистрибутива, распространяемого разработчиком, а DISTFILES – полное имя. Например, все файлы с исходным кодом Apache именуются в соответствии со схемой: *httpd-<что-то_еще>* ❻, а *httpd-<версия>.tar.bz2* ❼ – фактический файл с конкретным номером версии.

Далее в файле *Makefile* можно найти переменные с адресом электронной почты производителя, кратким описанием «порта» и т. д. За ними следуют такие строки:

```
❶ .if defined(WITH_EXCEPTION_HOOK)
CONFIGURE_ARGS+=            --enable-exception-hook
.endif
...
❷ .if defined (WITH_LDAP) || defined (WITH_LDAP_MODULES)
USE_OPENLDAP=               YES
CONFIGURE_ARGS+=            --with-ldap \
                            --with-ldap-lib="${LOCALBASE}/lib" \
                            --with-ldap-include="${LOCALBASE}/include"
.endif
...
```

Операторы .if defined – это настройки сборки «порта». Первая строка в данном примере – это переменная, которую надо установить, чтобы получить соответствующую функциональность. Я не знаю, что такое exception hook (ловушка исключений) ❶, и, честно говоря, меня это совершенно не волнует. Но если вы ближе знакомы с Apache, для вас этот параметр может иметь значение. FreeBSD позволяет вам легко включить эту функцию при сборке. Подобным образом, если требуется включить поддержку LDAP – при сборке Apache задайте в командной строке параметр WITH_LDAP=YES ❷, но тогда при сборке «порта» он потребует установить OpenLDAP как одну из зависимостей.

Утомительно? Да, но в результате вы узнаете, какие дополнительные параметры предлагает автор программного обеспечения, и научитесь их настраивать. Опытные системные администраторы, правда, могут настраивать свои любимые программы с закрытыми глазами. Но даже им не запомнить, как это делается для всех пакетов в коллекции «портов», которых больше 17 000.

Если для сборки «порта» вы используете команды, предшествующие команде `make install` (например, `make patch` и затем `make install`), вам придется включать дополнительные параметры на каждом таком этапе. В противном случае «порт» может быть собран неправильно. Например, если требуется собрать Apache с поддержкой LDAP, предварительно наложив собственную заплатку, выполните команду:

```
# make patch WITH_LDAP=YES
```

И затем команду:

```
# make install WITH_LDAP=YES
```

В противном случае можно наложить заплатку на программное обеспечение, подразумевающее поддержку LDAP, но в действительности собрать его без такой поддержки. В результате может нарушиться внутренняя целостность программы.

Самая сложная часть в сборке программы – решить, какие параметры применять. К сожалению, для этого вопроса нет простого ответа, поэтому для принятия решения лучше всего изучить руководство по этой программе или зайти на соответствующий веб-сайт. Мне самому не раз случалось установить какое-то программное обеспечение, а затем, прочтя документацию, удалить его и переустановить – уже с корректными параметрами.

Деинсталляция и повторная установка

Для установки «портов» характерна замечательная особенность: после того, как «порт» установлен, он воспринимается как пакет. Применив pkg_delete(1), его можно деинсталлировать, а с помощью pkg_info(1) – получить информацию о нем. Так как установка «портов» регистрируется в */var/db/pkg*, изучив файл *contents*, можно узнать, какие файлы входят в состав «порта».

Кроме того, «порт» можно деинсталлировать в каталоге «порта». Например, пусть FreeBSD содержит несколько версий одного «порта», скажем, веб-сервера Apache. Может потребоваться оценить различные версии, для чего каждую из них придется сначала установить, а потом удалить. Запуск `make deinstall` в каталоге «порта» удаляет программу из основной системы.

После деинсталляции «порта» скомпилированная программа и файлы с исходным кодом по-прежнему остаются в *рабочем* каталоге «порта». Команда `make reinstall` выполнит повторную установку деинсталлиро-

ванной программы. Удалять и переустанавливать программу можно столько раз, сколько вам потребуется.

Отслеживание состояния сборки «порта»

Как система «портов» отслеживает то, что уже сделано? Если вы запускаете сначала команду `make extract`, а затем `make install`, – как FreeBSD узнает, какие этапы уже пройдены? Для хранения информации о пройденных этапах система «портов» использует скрытые файлы. Увидеть эти файлы можно, запросив «длинный список» файлов в *рабочем* каталоге порта:

```
# cd /usr/ports/games/oneko/work
# ls -la
total 502
...
-rw-r--r--  1 root  wheel       0 Apr 22 14:59 .build_done.oneko._usr_X11R6
-rw-r--r--  1 root  wheel       0 Apr 22 14:59
.configure_done.oneko._usr_X11R6
-rw-r--r--  1 root  wheel       0 Apr 22 14:59 .extract_done.oneko._usr_X11R6
-rw-r--r--  1 root  wheel       0 Apr 22 14:59 .install_done.oneko._usr_X11R6
-rw-r--r--  1 root  wheel       0 Apr 22 14:59 .patch_done.oneko._usr_X11R6
...
```

Файлы, имена которых начинаются с точки, – это *скрытые* файлы; они не отображаются при обычном выводе списка файлов в каталоге. Система «портов» хранит информацию о пройденных этапах сборки в скрытых файлах. Эти файлы используются при сборке любого «порта». Видите файл *.install_done.oneko._usr_X11R6*? Это означает, что этап установки завершен.

В моей практике несколько раз бывало, что после многократного выполнения команд `make install/deinstall` «порт» отказывался устанавливаться повторно. Это происходило из-за скрытого файла, показывающего, что этап установки «порта» завершился. Удалите этот файл, и повторная установка будет выполнена.

Очистка с помощью make clean

«Порты» могут занимать много места. Некоторые программные пакеты, такие как X.org, могут съедать сотни мегабайт дискового пространства, а OpenOffice.org – целых девять гигабайт! В системе «портов» есть метод, позволяющий удалить ненужные файлы.

Установленной и сконфигурированной программе уже не нужен ее исходный код из каталога «порта». Его можно удалить с помощью `make clean`. Эта операция сотрет *рабочий* каталог текущего «порта» вместе со всеми его зависимостями, поэтому перед ее выполнением необходимо убедиться, что с установленной программой все в порядке. Очистить новый «порт» можно и автоматически по завершении установки. Для этого установку надо выполнять так: `make install clean`.

Также можно очистить исходные distfiles «порта», которые хранятся в */usr/ports/distfiles*. Команда `make distclean` удалит ненужные distfiles текущего «порта» вместе со всеми зависимостями.

Для очистки всего дерева «портов» запустите `make clean -DNOCLEANDEPENDS` прямо в */usr/ports*. Этот процесс займет какое-то время. Хотя есть более быстрые и эффективные способы удаления каждого *рабочего* каталога в дереве «портов», описанный метод напрямую поддерживается Проектом FreeBSD.

Сборка пакетов

С помощью «портов» вы можете собирать собственные пакеты для установки на других машинах FreeBSD, что сэкономит время и обеспечит идентичность программного обеспечения на всех машинах. Например, если требуется, чтобы на всех машинах, где есть Ooo, этот пакет обладал одними и теми же функциональными возможностями, можно собрать OOo один раз и затем установить его на остальных машинах.

Пакет создает команда `make package`. Она установит программу на локальной машине и создаст пакет в каталоге «порта». Затем надо скопировать этот пакет на другие системы и установить их с помощью pkg_add(1).

Если есть каталог */usr/ports/packages*, система «портов» создаст дерево пакетов в этом каталоге. То есть новые пакеты будут размещаться уже не в каталоге «портов», а в соответствующей категории, в каталоге */usr/ports/packages*.

Локальное хранилище пакетов

Помните переменную окружения `PACKAGESITE`? Задайте в ней путь к своему анонимному локальному серверу FTP (глава 17) или к разделяемому ресурсу NFS (глава 8) и разместите на нем свои пакеты. После этого можно выполнять `pkg_add -r` на других машинах, что позволит им автоматически забирать и устанавливать самодельные пакеты.

Изменение пути установки

Если у вас десятки, а то и сотни систем FreeBSD с идентичной конфигурацией, то для установки «портов» и пакетов может оказаться неудобным используемый по умолчанию путь */usr/local*. Если серверов много, каталог */usr/local* обычно резервируют для программ, уникальных для отдельной машины, а те программные пакеты, что используются всеми, устанавливают в другом месте, например в */opt* или */usr/pkg*. Указать требуемый каталог установки можно с помощью переменной окружения `PREFIX`:

```
# make PREFIX=/usr/pkg install clean
```

Устанавливаемые «порты» разместят все свои программы в заданном каталоге. Например, программы, которые обычно устанавливаются в каталог */usr/local/bin*, будут размещены в каталоге */usr/pkg/bin*.

Установка параметров make по умолчанию

Если вы устали без конца задавать одни и те же параметры при сборке «портов», можете перечислить их в */etc/make.conf*, и тогда они будут автоматически применяться при установке каждого «порта». Если какие-то из параметров не используются тем или иным «портом», их задание никак на него не повлияет.

Предположим, если в качестве стандартной базы данных используется MySQL, вы хотите, чтобы любое программное обеспечение, способное поддерживать работу с MySQL, собиралось с такой поддержкой. Для этого нужно определить в файле */etc/make.conf* переменную `WITH_MYSQL`:

```
WITH_MYSQL=YES
```

Теперь все устанавливаемые вами «порты», допускающие поддержку MySQL, автоматически будут собираться с такой поддержкой.

Локальное хранилище для файлов distfile

Если у вас несколько машин с «портами», то можно создать локальное хранилище для файлов distfile. Нужные файлы distfile можно загрузить из Интернета только раз, на одной из машин, а все остальные машины смогут получить доступ к этим файлам через локальную сеть, что ускорит сборку и уменьшит объем внешнего трафика. Для этого нужно указать в переменной `MASTER_SITE_OVERRIDE` местоположение централизованного хранилища файлов distfile. После этого, если потребуется собрать какой-либо «порт», система сначала проверит наличие требуемого файла дистрибутива в этом хранилище. Это может быть анонимный локальный сервер FTP или, например, разделяемый ресурс NFS.

Обеспечение безопасности при работе с «портами» и пакетами

За безопасностью «портов» уследить очень непросто. Каждый разработчик программного обеспечения самостоятельно решает проблемы, связанные с безопасностью. Обновление «порта» может означать, что в предыдущей его версии были обнаружены проблемы безопасности, но это может быть и просто новая версия программы. Как узнать, какие «порты» могут оказаться небезопасными?

В состав FreeBSD входит программа portaudit(1) – инструмент, который сверяет установленные версии «портов» с базой данных уязвимостей. Выявив в «порте» проблему с безопасностью, разработчики FreeBSD добавляют в базу данных portaudit описание этой проблемы. В результате при следующем запуске программа portaudit(1) предупредит вас о проблеме. Выполните установку программы portaudit:

```
# cd /usr/ports/ports-mgmt/portaudit
# make all install clean
```

По умолчанию программа portaudit(1) интегрируется с periodic(1), чтобы запускаться ежедневно. Во время работы она загружает последнюю версию базы данных уязвимостей с сайта *http://www.freebsd.org* и сравнивает со списком «портов» в системе. Результаты проверки отправляются по электронной почте пользователю root с отчетом periodic и URL-адресами, по которым можно найти описание выявленных проблем и пути их устранения. Вот пример одного из таких сообщений:

```
Affected package: mozilla-1.7.13_2,2
Type of problem: mozilla -- multiple vulnerabilities.
Reference: <http://www.FreeBSD.org/ports/portaudit/e6296105-449b-11db-ba89-
000c6ec775d9.html>

(Первод: Проблемный пакет: mozilla-1.7.13_2,2
Тип проблемы: mozilla -- множественные уязвимости.
Справка: <http://www.FreeBSD.org/ports/portaudit/e6296105-449b-11db-ba89-
000c6ec775d9.html>)
```

Кроме того, система «портов» запускает portaudit(1) при каждой попытке установить «порт». Если для «порта» обнаружены проблемы, связанные с безопасностью, его установка будет прервана. Если вы все же решили установить небезопасное программное обеспечение, то обойти запрет можно, введя команду установки «порта» `make install DISABLE_VULNERABILITIES=YES`.

Ежедневный запуск программы portaudit(1) не сделает вашу систему безопасной. Даже самая безопасная версия серверной программы не гарантирует безопасности при неправильных настройках, как самый прочный в мире сейф с надписью «ДЕНЬГИ» и незапертой дверцей. Для проверки безопасности я советую использовать дополнительные программные инструменты, такие как Nessus.

В этой главе вы узнали, как установить в операционной системе FreeBSD программное обеспечение, а в следующей мы рассмотрим вопросы его администрирования.

12

Расширенное управление программным обеспечением

В предыдущей главе были рассмотрены простые случаи установки и запуска программного обеспечения в операционной системе FreeBSD, однако FreeBSD способна решать и более сложные задачи, помогая системным администраторам полнее удовлетворять потребности пользователей. Обладая информацией о том, как в действительности работает система, вы сможете принимать оптимальные решения. Например, наличие нескольких процессоров в системе, многоядерные процессоры и процессоры с поддержкой технологии HyperThreading позволяют увеличить производительность системы, но ее прирост не всегда настолько велик, как можно было бы ожидать. Знание степени влияния многопроцессорной обработки при различных нагрузках поможет вам понять, где можно повысить производительность системы, а где – нет. Многопроцессорная обработка подразумевает использование многопоточных библиотек и специализированных планировщиков, поэтому здесь мы рассмотрим и эти составляющие.

Для запуска программ во время начальной загрузки и их корректного останова при выключении системы необходимо уметь редактировать и создавать сценарии запуска и останова. Хотя одни программы прекрасно прекращают работу при останове операционной системы, другим программам (например, базам данных) необходима более мягкая остановка работы. Хотя на выручку могут прийти различные трюки, «чистый» запуск и останов сетевых сервисов – это замечательный навык, который нужно приобрести и применять, поэтому здесь мы подробнее ознакомимся со сценариями запуска и останова FreeBSD.

В обычных условиях необязательно знать, как во FreeBSD организованы связывание и поддержка разделяемых (совместно используемых) библиотек, тем не менее здесь будут обсуждаться вопросы их конфигу-

рирования и управления ими. Почему? Потому что обычные условия, как ни странно, довольно редки в компьютерной среде.

Наконец, здесь будет рассмотрена возможность запуска программного обеспечения, созданного для других операционных систем. Мы узнаем, как это делается, сконцентрировав основное внимание на популярном пакете обеспечения совместимости с операционной системой Linux, позволяющем запускать немодифицированное программное обеспечение Linux в среде FreeBSD. Кроме того, будут рассмотрены вопросы запуска программного обеспечения, созданного для других аппаратных платформ.

Использование нескольких процессоров – SMP

Компьютеры с несколькими процессорами существуют несколько десятков лет, но сейчас наблюдается взрывообразный рост их популярности. FreeBSD обеспечивает поддержку нескольких процессоров, начиная с версии 3, однако многопроцессорные и многоядерные архитектуры еще не столь широко распространены. В современных системах используется принцип *симметричной многопроцессорной обработки* (Symmetric MultiProcessing, SMP), который подразумевает систему с несколькими идентичными процессорами. (Да, в некоторых многопроцессорных системах не требуется, чтобы процессоры были идентичными. Возможно, как раз такая система имеется в вашем компьютере – в виде нескольких видеокарт, у каждой из которых есть свой специальный микропроцессор, отвечающий за вывод графических изображений.)

По сравнению с одним процессором SMP-система обладает многими преимуществами – и она не просто «более мощная». Если рассуждать на микроуровне, процессор за раз может выполнить только одну операцию. Процессы в компьютере конкурируют за время процессора. Если процессор выполняет запрос к базе данных, он не примет пакет, который пытается доставить карта Ethernet. Каждую долю секунды процессор *переключает контекст* и работает с другим процессом, назначенным ядром. Такое переключение происходит довольно часто и достаточно быстро, поэтому кажется, что одновременно выполняются несколько операций – подобно тому, как картинка на телеэкране движется за счет очень быстрой смены отдельных кадров.

В качестве программного обеспечения рабочего стола я использую WindowMaker, для отображения содержимого архивов почтовой рассылки – Firefox, а OpenOffice.org принимает текст, который я ввожу с клавиатуры. В систему постоянно прибывают сетевые пакеты, экран отображает текст, MP3-плеер передает мелодии Blue Oyster Cult в мои наушники – все это происходит одновременно. В действительности, «одновременно» это происходит только для моего слабого мозга. На самом деле компьютер просто очень быстро переключается с одной

задачи на другую. В одну миллисекунду он посылает очередной звуковой фрагмент в мои наушники, в другую – обновляет изображение текста на экране.

При наличии нескольких процессоров компьютер может выполнять несколько операций одновременно. Это замечательно, однако такая способность чрезвычайно увеличивает сложность системы.

Допущения о работе ядра

Для понимания многопроцессорной обработки и проблем, связанных с ней, необходимо углубиться в работу ядра. Все операционные системы, в которых реализуется SMP, сталкиваются с одними и теми же проблемами. Следовательно, теория, представленная здесь, подходит для различных платформ. Однако FreeBSD в какой-то мере отличается от других операционных систем, поскольку опирается на 30-летнее наследие UNIX, а дальнейшая разработка FreeBSD не прекращается ни на месяц. Следующее описание представляет собой грубое упрощение. Конструкция ядра – мудреная тема. Почти невозможно описать ее точно на уровне, понятном для непрограммистов. Вот объяснение работы ядра в самом общем виде.

FreeBSD использует процессор по принципу *квантов времени* (*time slices*). *Квант времени* – это временной интервал, в течение которого процессор занимается одной задачей. Один процесс может использовать процессор в течение всего кванта либо до того момента, когда задача будет выполнена. После этого может быть запущен другой процесс. Для выделения квантов времени и их распределения между программами ядро применяет систему приоритетов. Если в системе появляется процесс, приоритет которого выше приоритета текущего процесса, ядро прерывает обработку первого процесса, или *вытесняет* его. Обычно такой режим называют *вытесняющей многозадачностью* (*preemptive multitasking*).

Хотя ядро запущено, это еще не процесс. Ядро запускает процессы. Процесс имеет определенные структуры данных, задаваемые ядром, и ядро управляет ими так, как считает нужным. Ядро можно представить как особый процесс, сильно отличающийся от других процессов. Он не может быть прерван другими программами – нельзя набрать `pkill kernel` и перезагрузить систему.

Ядро сталкивается с особыми проблемами, неведомыми остальным частям системы. Представьте, что имеется некоторая программа, которая выполняет передачу данных через сеть. Ядро принимает данные от программы и помещает их в область памяти, доступную для сетевой карты. Если компьютер занимается только одной какой-то задачей, то с той областью памяти или с сетевой картой ничего не случится, пока ядро не вернется к решению этой задачи. Однако при наличии нескольких процессоров компьютер одновременно может решать сразу несколько задач. Что если разные процессоры одновременно станут пере-

давать данные сетевой карте? Сетевая карта попадет в положение человека, которому в одно ухо кричит начальник, а в другое – мама, и что бы он ни сделал, никому не нравится. А что если один процессор попытается выделить область памяти для решения сетевых задач, а другой ту же самую область памяти – для нужд файловой системы? Ядро просто запутается, и полученные результаты вас точно не удовлетворят.

В старых ядрах разных UNIX, в том числе FreeBSD, трудности с SMP преодолеваются так: ядро объявляется невытесняемым – его работа не может быть прервана. За счет этого управление ядром становится простым и ясным: если какой-либо модуль ядра занимает участок памяти, он может рассчитывать на данный участок при выполнении следующей команды. Никакой другой модуль ядра не может захватить эту память. Когда компьютеры могли выполнять всего одну операцию в каждый конкретный момент времени, можно было делать некоторые допущения. Но стоит начать выполнять одновременно несколько операций – и все допущения идут прахом.

SMP: первая попытка

Первая реализация поддержки SMP в FreeBSD была довольно проста. Процессы распределялись между процессорами так, чтобы нагрузка была приблизительно равномерной. Для ядра существовала специальная «блокировка» (lock). Для выполнения кода ядра процессор должен был захватить «блокировку», проверив перед этим, доступна ли она. Если «блокировка» была доступна, он захватывал ее и запускал ядро. Если же «блокировка» была недоступна, процессор знал, что ядро выполняется другим процессором, и переключался на обработку чего-то другого. Эта «блокировка» называлась *Big Giant Lock* (*BGL*). В такой системе ядро знало, что его данные не затронет ни один процесс. По существу, данная схема гарантировала, что ядро будет запущено только на одном процессоре, как это всегда и было.

Такая стратегия была довольно эффективной в случае двух процессоров. На двухпроцессорной машине можно было запустить базу данных среднего уровня и веб-сервер и быть уверенным, что подсистема процессоров не станет причиной снижения производительности. Если один процессор был занят обслуживанием веб-страниц, то другой мог обрабатывать запросы к базам данных. Однако на машине с восемью процессорами вас поджидали трудности. Значительную часть времени система ждала, пока освободится BGL!

Такая упрощенная реализация SMP неэффективна и немасштабируема. В руководствах по SMP этот метод упоминается редко, поскольку он довольно неуклюж. Тем не менее он лучше методов организации SMP, предлагаемых другими системами. Например, по умолчанию в двухпроцессорной системе Windows 2000 один процессор выделяется для пользовательского интерфейса, а другой – для всего остального. Это решение тоже редко упоминается в руководствах, хотя благодаря

ему интерфейс становится быстрым, и мышь не замирает при увеличении нагрузки на систему.[1]

Современная реализация SMP

Новый метод симметричной многопроцессорной обработки данных был реализован в версии FreeBSD 5.0 и продолжает совершенствоваться по сей день. В операционной системе FreeBSD блокировка BGL была раздроблена на множество маленьких блокировок, и теперь каждая подсистема ядра использует для решения своих задач минимальную возможную блокировку.

Изначально блокировки были реализованы в таких базовых элементах инфраструктуры ядра, как планировщик (часть ядра, которая занимается распределением квантов времени между задачами), сетевой стек, стек дисковых операций ввода-вывода и т. д. Это сразу положительно сказалось на производительности, потому что в то время, как один процессор занимался планированием задач, другой мог заниматься обработкой сетевого трафика. Затем блокировки были сдвинуты еще глубже – в различные компоненты ядра. Для каждой части сетевого стека была создана своя блокировка, для каждой части подсистемы ввода-вывода – своя и т. д., что позволило ядру одновременно выполнять несколько операций. Такие отдельные подпроцессы ядра называются *потоками* (*threads*). Каждый тип блокировки предъявляет свои собственные требования. Вам встретится упоминание самых разных типов блокировок, таких как мьютексы, sx-блокировки, rw-блокировки, спин-блокировки и семафоры. У каждого из этих типов есть как преимущества, так и недостатки, и задействовать их в программном коде ядра нужно особенно аккуратно.

Проблема дробления блокировок на самом деле *гораздо* сложнее, чем может показаться. Если раздробить блокировки чересчур тонко, то основным занятием ядра станет не обработка данных, а обслуживание блокировок. Если же раздробить блокировки недостаточно тонко, то система будет простаивать в ожидании освобождения блокировок. Степень дробления, достаточная для обеспечения высокой эффективности в двухпроцессорной системе, может оказаться совершенно неприемлемой в тридцатидвухпроцессорной системе. Поиск оптимального решения занял годы и будет продолжаться в течение всей жизни FreeBSD.

Проблемы SMP: взаимоблокировка, клинч и нарушение порядка приобретения блокировок

Все блокировки ядра применяются по сложным правилам и взаимодействуют друг с другом миллионом способов. Правила защищают от непреднамеренного взаимовлияния блокировок. Предположим, что

[1] И поэтому в Windows 2000 вы можете гонять указатель мыши по экрану зависшего компьютера до бесконечности. – *Прим. науч. ред.*

потоку ядра A потребовались ресурсы Y и Z; потоку ядра B также потребовались ресурсы Y и Z, но при этом поток B должен сначала захватить ресурс Z, а потом Y. Если поток A захватит ресурс Y, а поток B в это же время захватит ресурс Z, то поток A остановится в ожидании освобождения ресурса Z, а поток B – в ожидании освобождения ресурса Y. Такой клинч дестабилизирует систему и, скорее всего, приведет к ее остановке. Правильная установка блокировок поможет избежать этой проблемы.

Возможно, вам приходилось увидеть на экране сообщение о *неверном порядке приобретения блокировок* – Lock Order Reversal (LOR), которое означает, что порядок приобретения блокировок был нарушен. Хотя это предупреждение не всегда свидетельствует о неминуемой гибели, тем не менее его не следует оставлять без внимания.

Параметр настройки ядра `WITNESS` позволяет активизировать функцию слежения за порядком приобретения блокировок. По умолчанию, в версии FreeBSD-current (глава 13) параметр `WITNESS` включен, и если вы отправите отчет о проблемах в вашей системе, команда разработчиков может попросить вас активизировать этот параметр. `WITNESS` вынуждает ядро инспектировать каждую операцию приобретения блокировки на предмет нарушения порядка следования, что снижает производительность системы. Тем не менее использование параметра `WITNESS`, чтение сообщений и реакция на них – это неоценимый способ помочь в улучшении FreeBSD.

Реакция на нарушение порядка приобретения блокировок

При получении одного из сообщений Lock Order Reversal, скопируйте его целиком. Эти сообщения дополнительно протоколируются в файле */var/log/messages*. Получив сообщение, проверьте страницу с сообщениями Lock Order Reversal по адресу *http://sources.zabbadoz.net/freebsd/lor.html*. Здесь содержится список всех ранее встреченных сообщений о нарушении порядка приобретения блокировок. Сравните первые несколько строк из этого списка со своим сообщением LOR. Если ваше сообщение присутствует в списке, сопутствующий комментарий позволит вам понять, что делать. Многие сообщения LOR, представленные на этой странице, не представляют опасности, а некоторые уже исправлены в более поздних версиях FreeBSD.

Если ваше сообщение LOR отсутствует в списке на сайте, просмотрите архивы почтовых рассылок. Если ваше сообщение LOR встретится в почтовой рассылке, прочитайте сообщение и выполните рекомендуемые действия. Не стоит посылать сообщение «и у меня тоже» в почтовую рассылку, если разработчик явно не выразил желание получать уведомления о сообщении LOR данного типа.

Если вам встретилось совершенно новое сообщение LOR, поздравляю! Обнаружение нового сообщения LOR такое же редкое событие, как открытие нового вида насекомых, – вы не сможете дать этому сообщению

свое имя, но это очень поможет проекту FreeBSD. Отошлите отчет о проблеме, как описано в главе 21. Укажите полную информацию о своей системе, особенно о том, какие задачи решались во время появления сообщения.

Процессоры и SMP

Существует три вида многопроцессорных систем: системы с несколькими процессорами, многоядерные процессоры и процессоры с поддержкой технологии HyperThreading. Понимать различия между ними совершенно необходимо, так как поведение системы и приложений напрямую зависит от различий в типах процессоров.

Несколько процессоров – это отдельные чипы на материнской плате (то есть то, что мы привыкли называть процессорами), способные сотрудничать друг с другом. Все процессоры в таких системах независимы, каждый из них обладает полным комплектом особенностей (такими как регистры, кэш и т. д.) и каждый управляется своим собственным аппаратным окружением.

Многоядерный процессор содержит несколько вычислительных устройств (ядер) в одном чипе. Популярность многоядерных процессоров продолжает расти, потому что многоядерные процессоры менее энергоемки и меньше нагреваются, чем сопоставимые многопроцессорные системы. Ядра могут совместно использовать такие встроенные аппаратные компоненты, как регистры и кэш. Каждое ядро в среднем работает медленнее, чем отдельный процессор, но наличие нескольких ядер позволяет параллельно выполнять несколько операций. Многоядерные процессоры более производительны в случае многопоточных приложений или в системах с несколькими процессами, а приложения, выполняющиеся как монолитный процесс, показывают лучшую производительность на одноядерном процессоре. На компьютере, особенно на сервере, может быть несколько многоядерных процессоров.

HyperThreading – это технология компании Intel, позволяющая одноядерному процессору разбить себя на два виртуальных процессора. Однако виртуальный процессор нельзя назвать полноценным процессором, поскольку второй виртуальный процессор задействуется, только когда первый (основной) простаивает в ожидании какого-либо события. Строго говоря, SMP – это *симметричная* многопроцессорная обработка, то есть все процессоры должны быть идентичны. Виртуальный процессор не обладает такой же производительностью, как реальный, поэтому их нельзя рассматривать как идентичные. Теоретически возможно написать планировщик задач, который будет использовать преимущества этой технологии, но для большинства типов решаемых задач в этом практически нет никакой пользы. Ни одна из UNIX-подобных операционных систем не обладает планировщиком, который бы полностью поддерживал все возможности технологии HyperThreading. Кроме того, эта технология таит в себе различные проблемы, связанные с без-

опасностью. Задача, исполняемая на одном виртуальном процессоре, может перехватывать данные, например ключи шифрования, принадлежащие задаче, исполняемой на другом виртуальном процессоре. По этой причине в операционной системе FreeBSD поддержка HyperThreading отключена по умолчанию. Если вам будет интересно попробовать эту технологию в действии на процессорах Intel, установите параметр sysctl `machdep.hyperthreading_allowed` в значение 1 и оцените производительность приложений. Для систем, где отсутствуют внешние пользователи, описанная выше проблема безопасности, вероятно, не имеет большого значения. (Если у вас реализованы виртуальные системы в клетках, это означает, что у вас *имеются* внешние пользователи.)

Применение SMP

Применяя SMP, помните, что несколько процессоров не обязательно ускоряют выполнение операций. Один процессор может выполнять определенное количество операций в секунду; наличие второго процессора означает, что компьютер может выполнять в два раза больше операций в секунду, однако эти операции не обязательно выполняются быстрее.

Проведем аналогию процессоров с полосами движения на дороге. Если на дороге одна полоса, за один раз финишную черту может пересечь одна машина. Если на дороге четыре полосы, эту черту могут одновременно пересечь четыре машины. Хотя дорога с четырьмя полосами вовсе не обязательно позволит этим машинам быстрее добраться до пункта назначения, многие из них придут к финишу в одно время. Если вы не чувствуете разницу, представьте, что произойдет, если кто-нибудь перекроет вам путь на дороге с одной полосой. Пропускная способность процессорной подсистемы очень важна.

Один процессор может выполнять только одну операцию в каждый конкретный момент времени, один процесс может выполняться только на одном процессоре в каждый конкретный момент времени. Большинство процессов не могут выполнять свою работу сразу на нескольких процессорах. Исключение составляют многопоточные программы, как будет показано ниже, в этой же главе. Некоторые программы преодолевают это ограничение, запуская сразу несколько процессов, что позволяет операционной системе распределять их между процессорами. Такая методика уже много лет применяется в Apache. Многопоточные программы написаны специально для выполнения на нескольких процессорах и используют другую методику. Многие многопоточные программы просто создают несколько потоков обработки данных и распределяют исполнение потоков между несколькими процессорами, что представляет собой простую и достаточно эффективную реализацию параллельных вычислений. Другие программы вообще не используют преимущества многопроцессорных систем. Если один из процессоров занят на все сто процентов, а другие практически простаивают, следовательно, запущена программа, не поддерживающая

многопроцессорную обработку данных. Более подробно о вопросах производительности мы поговорим в главе 19.

SMP и make(1)

Утилита make(1), которая выполняет сборку программ, может запускать сразу несколько процессов. Если программа написана аккуратно, ее сборка может выполняться сразу несколькими процессами. Такой подход не поможет увеличить скорость сборки маленьких программ, но при сборке крупных программных продуктов, таких как сама операционная система FreeBSD (глава 13) или OpenOffice.org, наличие нескольких процессоров может существенно ускорить процесс сборки. Чтобы указать системе, сколько процессов следует запускать одновременно, нужно передать утилите make(1) ключ `-j` и число процессов. Оптимальным выбором будет число процессоров или ядер плюс один. Например, в двухпроцессорной системе, где каждый процессор имеет по два ядра, можно было бы запустить пять процессов сборки программы.

```
# make -j5 all install clean
```

Не все программы можно собирать с ключом `-j`. Если дела пойдут плохо, уберите ключ `-j` и повторите попытку.

В главе 19 представлены различные способы оценки и измерения производительности.

Планировщики

Планировщик (*scheduler*) – это часть ядра, которая определяет, какие задачи и процессы могут получить процессор и в какой последовательности. Это очень важный компонент системы, от которого в значительной степени зависит ее производительность. Правильно выбранный планировщик обеспечивает устойчивую производительность системы, в то время как неправильно выбранный планировщик приводит к существенному снижению производительности, несмотря на то, что процессоры простаивают. Планировщик не может произвольным образом назначать приоритеты: если серверу базы данных дать больший квант времени во время интенсивного выполнения дисковых операций ввода-вывода, это не повысит производительность базы данных, если она ожидает завершения записи на диск. Планировщики должны учитывать типы процессоров, объем памяти, степень нагрузки и другие параметры. На первый взгляд, достаточно просто написать планировщик для конкретного типа нагрузки, но создать планировщик, который работал бы на сервере базы данных, веб-сервере, настольном компьютере и ноутбуке одинаково хорошо, очень *сложно*.

В операционной системе FreeBSD есть два основных планировщика – 4BSD и ULE. Планировщик 4BSD появился давно, еще в 4.4BSD. Он хорошо зарекомендовал себя за последние несколько десятков лет под различными видами нагрузок. Планировщик показывает достаточ-

ную зрелость, однако он плохо подходит для четырехпроцессорных и крупных систем. В операционной системе FreeBSD 7.0 по умолчанию используется планировщик 4BSD, в конфигурации ядра ему соответствует параметр `SCHED_4BSD`.

Планировщик ULE был специально разработан для многопроцессорных систем, но он показывает неплохие результаты и на однопроцессорных системах. ULE показал существенный прирост производительности в многопроцессорных системах, но у него нет такой долгой истории тестирования и отладки, как у 4BSD. Скорее всего, в недалеком будущем планировщик ULE станет планировщиком по умолчанию (к моменту написания этих строк еще не стал). Активизировать планировщик ULE можно с помощью параметра ядра `SCHED_ULE`.

Вообще говоря, планировщик ULE обеспечивает более высокую производительность в многопроцессорных системах, где работают такие сложные приложения, как серверы базы данных. Группа разработки FreeBSD использует тесты производительности на основе MySQL и PostgreSQL, и по мнению разработчиков уровень производительности должен совпадать или даже превышать уровень производительности текущей версии Linux[1], при использовании одних и тех же тестов на одном и том же аппаратном обеспечении.

Какой планировщик выбрать?

Я рекомендую ULE для всех типов систем. Если с этим планировщиком работа системы ухудшится, сообщите о проблеме и, если нужно, вернитесь к планировщику 4BSD. Ядро может использовать только один планировщик.

Сценарии запуска и останова

Сценарии запуска и останова находятся в каталоге */etc/rc* и называются *rc-сценариями*. Такой сценарий управляет загрузкой системы в многопользовательском режиме и процессом ее останова. Основные сценарии запуска располагаются в каталоге */etc/rc.d*, сценарии из других каталогов управляют дополнительным программным обеспечением. Сценарии запуска устанавливаются из «портов» и пакетов, но если вы устанавли-

[1] Группы разработки ядра FreeBSD и Linux ведут весьма корректную конкурентную борьбу в этой области, каждая из них пытается превзойти успехи конкурента. В конечном результате это приводит к улучшению обеих операционных систем. Иногда вперед вырывается FreeBSD, иногда Linux. От такой конкуренции выигрывают обе операционные системы. По данным группы разработчиков FreeBSD, отвечающей за производительность, эта операционная система лидирует гораздо чаще.

ваете собственное программное обеспечение, то вам потребуется самостоятельно создать сценарии запуска. Этот раздел предполагает базовое понимание сценариев на языке командного интерпретатора. Если вы никогда их не видели и не применяли, изучите примеры очень внимательно. Создавать командные сценарии нетрудно, и лучший способ научиться этому состоит в том, чтобы изучать примеры и приспосабливать их под свои нужды. Кроме того, для изменения процесса запуска или останова необходимо понимать, как работают сценарии запуска.

В ходе начальной загрузки операционная система FreeBSD проверяет каталог */usr/local/etc/rc.d* на наличие дополнительных сценариев командного интерпретатора, чтобы использовать их в процессе запуска/останова. (С помощью параметра `local_startup` в файле *rc.conf* можно определить дополнительные каталоги, но сейчас мы будем полагать, что в системе имеется единственный каталог, используемый по умолчанию.) В процессе запуска производится поиск исполняемых сценариев командного интерпретатора и предполагается, что все найденные сценарии являются сценариями запуска. Система выполняет такие сценарии с аргументом `start`. При останове система FreeBSD запускает те же сценарии с аргументом `stop`. Предполагается, что сценарии принимают эти аргументы и выполняют соответствующие действия.

Порядок запуска rc-сценариев

Самое интересное, что rc-сценарии сами определяют порядок запуска. Вместо определения порядка запуска по именам файлов сценариев, как это делается во многих других UNIX-подобных операционных системах, каждый сценарий запуска сам определяет, какие ресурсы необходимы, чтобы он мог быть запущен. С помощью этой информации система запуска и определяет порядок запуска сценариев. Делается это с помощью утилиты rcorder(8) во время загрузки и останова, но вы можете попробовать запустить ее вручную в любое время, чтобы посмотреть, что из этого получится. Просто передайте в качестве аргументов команде rcorder пути к каталогам, где находятся сценарии запуска.

```
# rcorder /etc/rc.d/* /usr/local/etc/rc.d/*
rcorder: Circular dependency on provision `mountcritremote' in file `archdep'.
(Перевод: Циклическая зависимость от `mountcritremote' в файле `archdep')
rcorder: requirement `usbd' in file `/usr/local/etc/rc.d/hald' has no
providers.
(Перевод: `usbd', требуемый в файле`/usr/local/etc/rc.d/hald', отсутствует)
dumpon
initrandom
geli
gbde
encswap
...
```

В данном случае утилита rcorder(8) отсортировала все сценарии в каталогах */etc/rc.d* и */usr/local/etc/rc.d* в порядке, в каком они будут запус-

каться во время загрузки системы. В первую очередь мы видим предупреждения: дело в том, что некоторые сценарии имеют циклические ссылки. Наличие таких конфликтов необязательно означает что-то ужасное, потому что не каждый сценарий всегда запускается во всех системах. В данном примере сценарий mountcritremote требует, чтобы перед ним был запущен сценарий archdep, но сценарий archdep, в свою очередь, должен запускаться после того, как отработает сценарий mountcritremote. В этом конфликте нет ничего страшного, потому что несмотря на него моя система все-таки загружается. Эта система не монтирует удаленные файловые системы, поэтому сценарий mountcritremote фактически ничего не делает, выдавая бессмысленное сообщение об ошибке. Точно так же сценарий hald требует, чтобы перед ним был запущен несуществующий сценарий usbd. В версии FreeBSD 7.0 отсутствует демон usbd(8), но он имеется в более старых версиях операционной системы, которые поддерживаются деревом «портов». Система запуска достаточно интеллектуальна, чтобы перескочить эти ошибки.

Упорядочивание сценариев запуска производится на основе маркеров, расположенных внутри самих сценариев.

Типичный сценарий запуска

Система запуска постоянно развивается. Поэтому, прежде чем приступать к созданию собственного сценария, загляните сначала в один из простых сценариев в каталоге */etc/rc.d*, чтобы убедиться в том, что мои слова все еще соответствуют действительности. Типичный сценарий запуска должен выглядеть примерно так, как показано ниже, хотя вы можете заметить незначительные отличия:

```
#!/bin/sh

❶ # PROVIDE: rpcbind
❷ # REQUIRE: NETWORKING ntpdate syslogd named

❸ . /etc/rc.subr

❹ name="rpcbind"
❺ rcvar=`set_rcvar`
❻ command="/usr/sbin/${name}"

❼ load_rc_config $name
❽ run_rc_command "$1"
```

Метка `PROVIDE` ❶ сообщает утилите rcorder(8) официальное название сценария. Если какой-то другой сценарий потребует предварительно запустить rpcbind(8), этот сценарий будет перечислен в списке зависимостей. Подобным образом метка `REQUIRE` ❷ перечисляет сценарии, которые должны быть запущены перед этим сценарием. Данный сценарий должен запускаться после сценариев NETWORKING, ntpdate, syslogd и named.

Чтобы использовать механизм самоупорядочения сценариев запуска, необходимо подключить к rc-сценарию инфраструктуру */etc/rc.subr* ❸.

Ваш сценарий должен определить имя команды ❹, как указано в */etc/rc.conf*.

Выражение `rcvar` ❺ – это наследие тех времен, когда FreeBSD и OpenBSD использовали единую систему сценариев запуска, но система запуска по-прежнему предполагает наличие этого выражения. Вам также необходимо указать точную команду, которая должна быть запущена ❻, – в принципе, в системе может быть несколько одноименных команд, если они находятся в разных каталогах. Далее выполняется загрузка конфигурации для этой команды из */etc/rc.conf* ❼. И, наконец, можно выполнить фактическую команду ❽.

Хотя все это может испугать, в действительности здесь нет ничего сложного. Для начала просто создайте копию одного из существующих сценариев. Впишите имя вашей команды и соответствующим образом измените путь. Выясните, какие сценарии должны запускаться перед ним: может быть, для запуска сценария требуется подключение к сети? Должны ли быть прежде запущены какие-то демоны или нужно, чтобы ваша программа запускалась перед какими-то демонами? Если это пока неизвестно, сделайте так, чтобы последним запускался ваш сценарий, скопировав содержимое метки `REQUIRE` из сценария, который запускается последним. Ознакомившись с содержимым других сценариев запуска, реализующих подобные функции, вы узнаете практически все, что нужно для создания сценария запуска.

С помощью этого простого сценария вы сможете разрешать или запрещать запуск своей программы, а также настраивать ее, добавляя параметры настройки в */etc/rc.conf*. Допустим, если вы дали своему демону имя tracker, ваш сценарий будет отыскивать переменные `tracker_enable` и `tracker_flags` в */etc/rc.conf* и использовать их каждый раз при выполнении сценария.

Специальные значения в метке PROVIDE

Система запуска предусматривает несколько специальных значений, с помощью которых в метке `PROVIDE` можно определить ключевые моменты в процессе загрузки. Эти значения позволяют упростить создание сценариев запуска.

Значение `FILESYSTEMS` гарантирует, что к моменту запуска будут смонтированы все локальные файловые системы, перечисленные в */etc/fstab*.

Значение `NETWORKING` гарантирует, что будут сконфигурированы все сетевые функции, включая настройку IP-адресов на сетевых интерфейсах, настройку фильтра пакетов PF и т. д.

Значение `SERVER` гарантирует готовность системы к запуску таких основных серверов, как named(8), и программ поддержки NFS. К этому моменту удаленные файловые системы еще не смонтированы.

Значение `DAEMON` гарантирует, что к моменту запуска будут смонтированы все локальные и удаленные файловые системы, включая NFS

и CIFS, и что будут запущены расширенные сетевые службы, такие как DNS.

Значение `LOGIN` соответствует состоянию, когда все сетевые службы системы уже запущены и FreeBSD приступает к запуску служб поддержки входа пользователя через консоль, FTP, SSH и других.

Задав одно из этих значений в метке `REQUIRE` в своем сценарии запуска, вы сможете в общих чертах определить, когда должна запускаться ваша программа.

Применение сценариев для управления запущенными программами

Те из вас, кто имел дело со сценариями командного интерпретатора, могли заметить, что приведенный выше сценарий фактически не включает ничего, что отвечало бы за запуск, перезапуск, остановку и получение информации о состоянии программы. Все это реализует rcNG.[1] Нужно перезапустить демона? Просто выполните команду:

```
# /usr/local/etc/rc.d/customscript restart
```

а все остальное сделает rcNG.

Читатели, обладающие некоторым опытом работы с UNIX, могут подумать: «Зачем тогда вообще создавать сценарий, если я просто могу использовать командную строку?» Если вы знакомы с программой, вы наверняка знаете, какую команду следует выполнить, чтобы запустить или остановить ее. Все так, но это не самый лучший выход, потому что тогда системный администратор рабочего сервера должен будет запускать и останавливать каждую программу строго определенным способом. При использовании инфраструктуры rcNG вы уверены, что в любое время программа будет запущена именно таким способом, как во время загрузки системы. Подобным образом, каждый раз программа будет останавливаться точно так же, как во время останова системы. Можно выучить все аргументы командной строки для каждого демона в системе и гордиться этим, но осознать необходимость единообразия запуска и останова и всегда применять обеспечивающие его системные инструменты – значит понимать разницу между *знанием* системы UNIX и *знакомством* с ней.

Сценарии запуска/останова сторонних производителей

Представьте, что вы устанавливаете сложный программный продукт, разработчик которого не поддерживает систему запуска FreeBSD. Для

---

[1] rcNG – это название всей подсистемы запуска и останова служб в NetBSD и FreeBSD. Возможно, вы найдете больше подробностей в системном руководстве (см. apropos rcng). В Solaris подсистема с подобной, но несколько более элегантно реализованной и мощной функциональностью, называется SMF. – *Прим. науч. ред.*

нас это не проблема. В большинстве случаев разработчики поставляют сценарии, принимающие единственный аргумент, например `stop` или `start`. Не забывайте, что во время загрузки FreeBSD запускает rc-сценарии без аргумента `start`, а во время останова – без аргумента `stop`. Добавив операторы `PROVIDE` и `REQUIRE` в виде комментариев, вы сможете организовать запуск сценария в соответствующее время на этапе загрузки и останова системы. Учет особенностей системы запуска в управлении сценариями не является обязательным требованием.

Отладка своих сценариев запуска

Локальные сценарии, например те, что устанавливаются системой «портов», запускаются с помощью */etc/rc.d/localpkg*. Если ваш собственный сценарий вызывает проблемы, можно попробовать запустить сценарий localpkg в режиме отладки и посмотреть, как ваш сценарий взаимодействует с системой запуска. Лучший способ заключается в использовании режима отладки.

```
# /bin/sh -x /etc/rc.d/localpkg start
```

При этом будет выполнена попытка запустить все локальные демоны на сервере, в рабочей системе это не всегда желательно. Попробуйте сначала выполнить отладку на тестовой системе. Кроме того, следует помнить, что ключ включения режима отладки `-x` не передается дочерним сценариям – в режиме отладки будет работать только сценарий */etc/rc.d/localpkg*.

Управление разделяемыми библиотеками

Разделяемые (совместно используемые, shared) библиотеки – это фрагмент компилированного кода, предоставляющий свои функции другим фрагментам компилированного кода. Разделяемые библиотеки предоставляют часто вызываемые функции другим программам и спроектированы так, чтобы как можно больше программ могли многократно их использовать. Например, многие программы должны *хешировать* данные, или шифровать без возможности расшифровки. Однако если каждая программа будет включать в себя модули для хеширования, писать такие программы станет труднее, а поддерживать – менее приятно. Более того, у программ могут возникнуть трудности с взаимодействием, если хеширование в них будет реализовано по-разному, и авторам программ придется изучать принципы хеширования, чтобы использовать эту возможность. Если же программа, которой необходимо хеширование, может обратиться к соответствующей функции совместно используемой библиотеки (в данном примере libcrypt), проблемы с взаимодействием и поддержкой устраняются. Такой подход снижает средний размер программ, высвобождая значительный объем системной памяти, и уменьшает их сложность.

Версии и файлы совместно используемых библиотек

Совместно используемые библиотеки имеют интуитивно понятные имена, номера версий и размещаются в соответствующих файлах. Имя библиотеки обычно (но не всегда) совпадает с именем соответствующего файла. Например, версия 2 разделяемой библиотеки libpthread хранится в виде файла */lib/libpthread.so.2*, а версия 10 библиотеки легковесной службы разрешения имен lwres – в файле */usr/lib/liblwres.so.10*.

Со временем изменения в библиотеке становятся столь значительными, что она делается несовместимой с более ранними версиями, в этом случае номер версии увеличивается на единицу. Например, библиотека libpthread.so.2 превратилась в libpthread.so.3. Группа разработки FreeBSD никак не заботится о номерах версий, за исключением момента начала цикла разработки очередной версии (глава 13). Кроме того, для каждой библиотеки существует символическая ссылка, имя которой совпадает с именем библиотеки, но без номера версии, которая указывает на файл самой последней версии библиотеки. Например, вы можете найти файл */usr/lib/libwres.so*, который в действительности является символической ссылкой, указывающей на файл */usr/lib/libwres.so.10*. Это существенно упрощает компиляцию приложений, поскольку программа будет искать файл библиотеки с обобщенным именем, а не конкретную версию.

В операционную систему FreeBSD 7 был добавлен *механизм контроля версий имен* (*symbol versioning*), который позволяет разделяемым библиотекам поддерживать несколько программных интерфейсов. При наличии контроля версий имен разделяемая библиотека предоставляет каждой программе требуемую ей версию библиотеки. Если, например, имеется программа, которой требуется библиотека версии 2, то версия 3 библиотеки также будет поддерживать необходимые функции.

Поддержка механизма контроля версий имен операционной системой FreeBSD еще не означает, что его поддерживают все программные продукты, присутствующие в коллекции «портов». Помните о возможных проблемах, связанных с версиями библиотек.

Подключение разделяемых библиотек к программам

Так как же программа получает доступ к нужной разделяемой библиотеке? Доступ к разделяемым библиотекам в операционной системе FreeBSD обеспечивают утилиты ldconfig(8) и rtld(1), а также несколько дополнительных инструментов, позволяющих влиять на порядок управления разделяемыми библиотеками.

Программа rtld(1), пожалуй, самая простая для понимания. Всякий раз, когда запускается какая-нибудь программа, rtld(8) отыскивает для нее нужные разделяемые библиотеки. Поиск ведется в каталогах с библиотеками, и если библиотеки найдены, выполняется связывание библиотек с программой. Вообще-то rtld(1) позволяет сделать не так

много, но она – важное связующее звено, обеспечивающее доступ к разделяемым библиотекам. Все современные библиотеки используют стандартную разделяемую библиотеку ld-elf.so, предназначенную для двоичных файлов в формате ELF. (ELF – это формат двоичных файлов, используемый в современных версиях FreeBSD; мы рассмотрим его ниже в этой главе.) Данная библиотека настолько важна для нормальной работы системы, что при обновлении FreeBSD предыдущая версия этой библиотеки не удаляется – на тот случай, если что-то пойдет не так.

Список каталогов с библиотеками: ldconfig(8)

Чтобы всякий раз при запуске какой-то программы, требующей динамического связывания с библиотеками, просеивать весь жесткий диск в поисках чего-то, напоминающего разделяемую библиотеку, система поддерживает с помощью ldconfig(8) список каталогов с разделяемыми библиотеками. (В старых версиях FreeBSD был реализован кэш библиотек, а в современных хранится только список каталогов, где производится поиск разделяемых библиотек.) Если программа не может обнаружить разделяемые библиотеки, которые, по вашему мнению, точно присутствуют в системе, это означает, что ldconfig(8) не знает о существовании каталога, где размещаются эти разделяемые библиотеки.[1] Чтобы увидеть список библиотек, которые могут быть найдены с помощью ldconfig(8), выполните команду `ldconfig -r`.

```
# ldconfig -r
/var/run/ld-elf.so.hints:
        search directories: /lib:/usr/lib:/usr/lib/compat:/usr/local/lib:/usr/
local/lib/nss
        0:-lcrypt.3 => /lib/libcrypt.so.3
        1:-lkvm.3 => /lib/libkvm.so.3
        2:-lm.4 => /lib/libm.so.4
        3:-lmd.3 => /lib/libmd.so.3
...
```

При запуске с ключом `-r` программа ldconfig(8) выведет список всех разделяемых библиотек, хранящихся в каталогах разделяемых библиотек. В этом списке сначала выводится имя каталога, в котором производится поиск, а затем имена библиотек, расположенных в этом каталоге. На моем ноутбуке этот список содержит 433 разделяемых библиотеки.

Если запуск какой-либо программы завершается ошибкой с сообщением о том, что невозможно найти разделяемую библиотеку, это означает, что нужная библиотека отсутствует в данном списке. В этом случае вам нужно установить нужную библиотеку в каталог с разделяемыми библиотеками или добавить каталог с библиотекой в список катало-

[1] Возможно, вам лишь *кажется*, что требуемые библиотеки присутствуют в системе, а *на самом деле* это совершенно другие библиотеки. Никогда не исключайте возможность собственной ошибки, пока окончательно не идентифицировали проблему!

гов, в которых ведется поиск. Можно было бы просто скопировать все разделяемые библиотеки в каталог */usr/lib*, но такое решение серьезно осложнит обслуживание системы, например, как в случае с картотекой, где все папки хранятся под одной литерой «Б» (бумаги). Для долгоживущих систем лучшим решением является добавление каталогов в список разделяемых библиотек.

Добавление каталогов с библиотеками в список поиска

Если был создан новый каталог с разделяемыми библиотеками, его следует добавить в список, который используется программой ldconfig(8) для поиска библиотек. Взгляните на следующие записи в файле */etc/defaults/rc.conf* с параметрами настройки ldconfig(8):

```
ldconfig_paths="/usr/lib/compat /usr/local/lib /usr/local/lib/compat/pkg"
ldconfig_local_dirs="/usr/local/libdata/ldconfig"
```

В переменной `ldconfig_paths` хранится список каталогов с библиотеками. В свежеустановленной системе FreeBSD каталог */usr/local/lib* отсутствует в списке, но в большинстве систем он появляется там вскоре после установки. Точно так же в списке появляется каталог */usr/lib/compat*, где находятся библиотеки обеспечения совместимости с предыдущими версиями FreeBSD. В каталоге */usr/local/lib/compat/pkg* размещаются старые версии библиотек, устанавливаемых пакетами. По умолчанию программа ldconfig(8) сначала просматривает каталоги */lib* и */usr/lib*, а затем каталоги из списка, которые являются типичными каталогами для размещения библиотек.

Вместо того чтобы бездумно сваливать все в каталог */usr/local/lib*, «порты» и пакеты включают свои разделяемые библиотеки в список поиска с помощью переменной `ldconfig_local_dirs`. Каждый пакет может установить файл в один из этих каталогов. Файл называется по имени пакета и просто содержит список каталогов с библиотеками, устанавливаемыми пакетом. Программа ldconfig отыскивает файлы в этих каталогах, извлекает из файлов пути к каталогам и рассматривает их как дополнительные пути к библиотекам. Например, «порт» nss (*/usr/ports/security/nss*) устанавливает разделяемые библиотеки в каталог */usr/local/lib/nss*. Кроме того, «порт» устанавливает файл */usr/local/libdata/ldconfig/nss*, который содержит единственную строку – путь к этому каталогу. Сценарий запуска ldconfig добавляет пути к каталогам, указанные в этих файлах, в список мест, где могут находиться разделяемые библиотеки.

ldconfig(8) и необычные библиотеки

С разделяемыми библиотеками связаны два крайних случая, которые вы должны понимать, и масса других случаев, о которых вам не придется беспокоиться, – это библиотеки для исполняемых файлов различных двоичных форматов и библиотеки для других архитектур.

/usr/local/lib или отдельный каталог с библиотеками для каждого «порта»?

Разве каталог */usr/local/lib* не предназначен специально для библиотек, устанавливаемых «портами» и пакетами? Почему бы просто не помещать все разделяемые библиотеки в этот каталог? Большинство «портов» именно так и поступает, но иногда наличие отдельного каталога может существенно упростить сопровождение системы. Например, на своем ноутбуке я установил Python 2.4, и каталог */usr/local/lib/python24* содержит 587 файлов! Если записать все эти файлы в каталог */usr/local/lib*, это привело бы к перемешиванию с библиотеками, не имеющими отношения к Python, и осложнило бы поиск файлов, установленных «портами».

Чтобы добавить свой каталог с разделяемыми библиотеками в список поиска, следует либо добавить путь к каталогу в переменную `ldconfig_paths` в файле /etc/rc.conf, либо создать файл со списком каталогов в */usr/local/libdata/ldconfig*. Оба варианта дают один результат. Как только каталог будет добавлен в список поиска, библиотеки из этого каталога сразу же станут доступны программам.

Операционная система FreeBSD поддерживает два различных формата двоичных исполняемых файлов, *a.out* и *ELF*. Системный администратор не обязан хорошо разбираться в этих двоичных форматах, но вы должны знать, что ELF – это современный формат, ставший стандартом FreeBSD, начиная с версии 3.0 в 1998 году. Старые версии FreeBSD использовали формат a.out. Скомпилированные программы одного типа не могут использовать библиотеки другого типа. И хотя двоичные файлы в формате a.out практически исчезли, затраты на поддержку этого формата настолько незначительны, что ее не стали удалять из операционной системы. Программа ldconfig(8) поддерживает отдельные списки каталогов для форматов a.out и ELF, что можно заметить в выводе сценария */etc/rc.d/ldconfig*. Для библиотек формата a.out в файле *rc.conf* присутствуют отдельные параметры настройки ldconfig(8).

Другой крайний случай – запуск 32-битовых приложений в 64-битовой версии FreeBSD. Чаще всего такая ситуация встречается, когда возникает необходимость использовать в архитектуре amd64 программы из более ранних версий FreeBSD. 64-битовые программы не могут использовать 32-битовые библиотеки, поэтому ldconfig(8) имеет отдельные списки каталогов с библиотеками для разных архитектур. Для этого случая в файле *rc.conf* также присутствуют отдельные параметры настройки ldconfig(8). Не смешивайте 32-битовые и 64-битовые библиотеки!

LD_LIBRARY_PATH

Механизм встраивания разделяемых библиотек в конфигурацию системы замечательно работает с точки зрения системного администратора, однако он не подходит для скромного пользователя без привилегий root.[1] Кроме того, если у обычного пользователя есть свой набор библиотек, системный администратор может не разрешить всеобщий доступ к ним. Скорее всего, системный администратор постарается не оставить ни малейшего шанса для связывания системных программ с личными библиотеками пользователя.

Всякий раз, когда запускается программа rtld(1), она проверяет значение переменной окружения `LD_LIBRARY_PATH`. Если в переменной указан список каталогов, программа проверяет их на наличие разделяемых библиотек. Все библиотеки, присутствующие в этих каталогах, могут использоваться программами. В переменной `LD_LIBRARY_PATH` можно указать любое число каталогов. Например, если бы мне потребовалось протестировать библиотеки в каталогах */home/mwlucas/lib* и */tmp/testlibs*, перед следующим запуском программы я мог бы присвоить переменной следующее значение:

```
# setenv LD_LIBRARY_PATH /home/mwlucas/lib:/tmp/testlibs
```

Значение переменной можно устанавливать автоматически при входе в систему, добавив соответствующую команду в файл *.cshrc* или *.login*.

LD_LIBRARY_PATH и безопасность

Этот подход небезопасен – если в `LD_LIBRARY_PATH` будет задано чрезмерно доступное местоположение библиотек, ваша программа может быть связана с чем угодно. Кроме того, `LD_LIBRARY_PATH` подменяет список каталогов с библиотеками, поэтому любой, у кого есть возможность помещать файлы в каталог с библиотеками, сможет использовать ваши программы в неблаговидных целях. В связи с этим setuid- и setgid-программы игнорируют переменную `LD_LIBRARY_PATH`.

Какие библиотеки нужны программе

Наконец, возникает вопрос: какие библиотеки нужны программе? Эту информацию можно получить с помощью ldd(1). Например, узнать, что требуется программе Emacs, можно посредством такой команды:

[1] Хотя большинство читателей этой книги наверняка являются системными администраторами, тем не менее вы можете рекомендовать своим пользователям приобрести эту книгу и прочитать данный раздел. Они наверняка откажутся, но хотя бы замолчат и оставят вас в покое.

```
# ldd /usr/local/bin/emacs
/usr/local/bin/emacs:
        libXaw3d.so.8 => /usr/local/lib/libXaw3d.so.8 (0x281bb000)
        libXmu.so.6 => /usr/local/lib/libXmu.so.6 (0x2820e000)
        libXt.so.6 => /usr/local/lib/libXt.so.6 (0x28222000)
        libSM.so.6 => /usr/local/lib/libSM.so.6 (0x2826d000)
...
```

Этот вывод содержит имена разделяемых библиотек, необходимых Emacs, и указывает расположение файлов, содержащих эти библиотеки. Если программа не может отыскать необходимую библиотеку, ldd(1) сообщит об этом. Сама программа объявляет имя только первой не найденной библиотеки, а ldd(1) выведет полный список, благодаря чему можно найти все отсутствующие библиотеки с помощью механизма поиска.

Вооружившись утилитами ldconfig(8) и ldd(8), вы будете полностью готовы к управлению разделяемыми библиотеками в системе FreeBSD.

Потоки, потоки и еще раз потоки

Слово поток, или нить (thread), встречается в разных контекстах. Некоторые процессоры поддерживают технологию HyperThreading. Некоторые процессы запускают несколько потоков исполнения (нитей). Операционная система FreeBSD содержит три отдельные библиотеки, позволяющие реализовать многопоточную модель исполнения. Некоторые части ядра работают как отдельные потоки исполнения (нити). Мои штаны состоят из огромного множества нитей (хотя в некоторых предметах одежды этих нитей так мало, что в них, по мнению моей жены, неприлично появляться на публике).[1] Что же это за потоки и что подразумевается под этим словом?

В большинстве случаев под словом поток подразумевается легковесный процесс. Не забывайте: *процесс* – это задача в системе, запущенная программа. Процессы в системе имеют собственные числовые идентификаторы (ID), могут запускаться, останавливаться и обычно могут управляться пользователем. *Поток* – это часть процесса, но потоки управляются самим процессом и недоступны пользователю. В каждый конкретный момент времени процесс может выполнять только одно действие, а отдельные потоки исполнения могут действовать независимо друг от друга. В многопроцессорных системах одновременно могут исполняться потоки, принадлежащие одному процессу.

---

[1] В английском «нить» и «поток» обозначаются одним и тем же словом. В русском языке это разные слова. Поэтому, хотя данное предложение можно рассматривать как забавное отвлечение, тем не менее оно плохо вписывается в контекст, так как слово «нить» довольно редко используется в русскоязычной литературе для обозначения потоков исполнения. – *Прим. перев.*

Любая многопоточная программа использует *библиотеку реализации многопоточной модели исполнения* (*threading library*), позволяющую приложению использовать потоки в данной операционной системе, взаимодействуя с ядром. Библиотеки реализуют многопоточную модель разными способами, поэтому применение конкретных библиотек может влиять на производительность приложения.

Подобным образом, *поток ядра* – это подпроцесс в ядре. В операционной системе FreeBSD имеются потоки ядра, которые обслуживают операции ввода-вывода, поддерживают работу с сетью и т. д. Каждый поток имеет собственные функции, задачи и механизмы блокировок. В многопоточной модели ядра не используются библиотеки из пространства пользователя.

HyperThreading – это маркетинговый термин, не имеющий никакого отношения к потокам исполнения в системе. Хотя вам необходимо понимать, что такое HyperThreading и какое влияние эта технология оказывает на систему (об этом рассказывалось в предыдущем разделе), тем не менее она не является частью многопоточной модели исполнения.

Библиотеки реализации многопоточной модели в пространстве пользователя

Операционная система FreeBSD содержит три различные библиотеки реализации многопоточной модели исполнения: libc_r, libkse и libthr.

Самой первой на свет появилась библиотека lib_r. Ее многопоточная модель была целиком реализована в пространстве пользователя. Она эмулировала работу потоков в пределах одного процесса. Никакой масштабируемости, никаких преимуществ от наличия нескольких процессоров в системе. Библиотека libc_r уже практически не поддерживается, она плохо работает и является старейшей библиотекой реализации многопоточной модели исполнения в операционной системе FreeBSD. Библиотека libc_r по-прежнему существует в дереве исходных кодов FreeBSD, но по умолчанию она не устанавливается. Использовать библиотеку libc_r не рекомендуется, но при определенных обстоятельствах, когда она может потребоваться, ее можно установить из исходных кодов:

```
# cd /usr/src/lib/libc_r
# make obj all install clean
```

В версии FreeBSD 6.0 по умолчанию использовалась библиотека libkse, представляющая собой результат амбициозной попытки реализовать потоки M:N, где M:N означает, что в любой момент времени в системе имеется M процессоров и пул из N потоков в пространстве пользователя. Теоретически, это самая мощная модель многопоточного исполнения, но она отличается высокой сложностью и с трудом поддается реализации. Очень немногие операционные системы поддерживают

потоки *M*:*N*, и хотя реализация в операционной системе вполне работоспособна, она не отличается оптимальностью. Библиотека libkse попрежнему распространяется в составе FreeBSD, но уже не используется по умолчанию.

Библиотека libthr – это новая библиотека реализации многопоточной модели исполнения, используемая операционной системой FreeBSD по умолчанию. В библиотеке libthr применяется более простая модель потоков, чем в libkse, но она обеспечивает более высокую производительность, частично благодаря своей простоте.

Вам также могут встретиться упоминания о библиотеке libpthread(3). Название *pthread* – это сокращение от *POSIX threads* (потоки исполнения стандарта POSIX). Таким образом, так может называться любая библиотека реализации многопоточной модели исполнения, соответствующая требованиям стандарта POSIX. В операционной системе FreeBSD название libpthread – это всего лишь псевдоним системной библиотеки, используемой по умолчанию.

В большинстве случаев библиотека по умолчанию прекрасно справляется со своими обязанностями. Но возможно, что при запуске какой-либо программы в библиотеке или программе будет выявлена ошибка. Замена библиотек реализации многопоточной модели исполнения может помочь избавиться от ошибки. Кроме того, с нестандартной библиотекой программа может работать устойчивее. Механизм переназначения разделяемых библиотек позволяет заставить конкретную программу использовать библиотеку, отличную от той, что используется остальной системой.

Переназначение разделяемых библиотек

Для некоторых программных компонентов предпочтительнее использовать конкретные библиотеки, отличные от тех, что используются остальной системой. Например, стандартная библиотека языка C – libc. Вы можете завести отдельную копию libc со специальными функциями, предоставляемыми только определенной программе, и заставить эту программу работать со специальной версией libc, при этом все другие программы будут использовать стандартную библиотеку libc. Операционная система FreeBSD позволяет подменять любую разделяемую библиотеку для любого приложения. Хотя это и кажется странным, тем не менее такая возможность полезна в особых случаях. Программа rtld(1) может обмануть клиентскую программу, если такое поведение ей предписывается файлом */etc/libmap.conf*.

Предположим, что сервер базы данных показывает лучшую производительность при использовании конкретной библиотеки реализации многопоточной модели исполнения, не являющейся библиотекой по умолчанию. Программа ldd(1) может сообщить, что программа предполагает обращаться к разделяемой библиотеке libpthread (она же –

libthr), но у вас может появиться желание использовать библиотеку libkse. В этом случае вам не требуется подменять библиотеку, с которой работает остальная часть системы. Когда программа сообщает: «Мне нужна библиотека libpthread», – нужно, чтобы утилита rtld(1) ответила: «Она здесь» и подключила бы библиотеку libkse. Такую подмену можно настроить и для всей системы в целом, и для отдельной программы с определенным именем, и даже для программ, расположенных в определенном каталоге.

Содержимое файла */etc/libmap.conf* поделено на две колонки. Первая колонка содержит имя разделяемой библиотеки, которую запрашивает программа, а вторая – имя разделяемой библиотеки, которая будет использована фактически. Все изменения вступают в силу при следующем запуске программы – не требуется ни перезагружать систему, ни перезапускать демоны. Например, ниже системе предписывается, чтобы программам, запрашивающим библиотеку libpthread, предлагалась библиотека libkse. Такие подмены глобального характера должны быть описаны в начале файла *libmap.conf*:

```
libpthread.so.2        libkse.so.2
libpthread.so          libkse.so
```

«Могу ли я получить библиотеку libpthread.so.2?» «Да, конечно, вот вам libkse.so.2.»

Глобальное переназначение библиотек – это слишком смелый шаг, который заставит системных администраторов заговорить о вас; переназначение библиотек для отдельных программ не так впечатляет, зато оно способно решить проблем больше, чем принести. Для этого достаточно просто указать имя программы и полный путь к ней в квадратных скобках, но в этом случае переназначение будет работать только при вызове программы по полному пути к ней. Если определить только имя программы, переназначение будет работать всякий раз, когда запускается какая-либо программа с данным именем. Например, ниже производится переназначение для csup(8), в результате которого при запуске программы с указанием полного пути к ней вместо библиотеки libpthread она будет использовать библиотеку libkse:

```
[/usr/bin/csup]
libpthread.so.2        libkse.so.2
libpthread.so          libkse.so
```

Как убедиться, что переназначение работает? Конечно, с помощью команды ldd(1):

```
# ldd /usr/bin/csup
/usr/bin/csup:
        libcrypto.so.5 => /lib/libcrypto.so.5 (0x2808f000)
        libz.so.3 => /lib/libz.so.3 (0x281b8000)
        ❶libpthread.so.2 => ❷/usr/lib/libkse.so.2 (0x281c9000)
        libc.so.7 => /lib/libc.so.7 (0x281db000)
```

Здесь видно, что когда программа */usr/bin/csup* запросит libpthread.so.2 ❶, rtld(1) свяжет ее с библиотекой libkse.so.2 ❷. Однако в этом примере мы указали полный путь к программе. Попробуем вызвать ldd для csup без указания полного пути к ней:

```
# cd /usr/bin
# ldd csup
csup:
        libcrypto.so.5 => /lib/libcrypto.so.5 (0x2808f000)
        libz.so.3 => /lib/libz.so.3 (0x281b8000)
        libpthread.so.2 => /lib/libpthread.so.2 (0x281c9000)
        libc.so.7 => /lib/libc.so.7 (0x281ef000)
```

Если перейти в каталог */usr/bin* и передать утилите ldd программу csup без полного пути к ней, то rtld не получит полный путь к исполняемому файлу csup(1). Поскольку в */etc/libmap.conf* указано, что подстановка выполняется для полного имени */usr/bin/csup*, то при обращении по одному имени csup в ответ на запрос библиотеки libpthread.so.2 будет предложена библиотека libpthread.so.2.

Если нужно, чтобы программа использовала альтернативную библиотеку независимо от того, вызывается она по полному имени или по базовому, то в квадратных скобках нужно указать только имя программы:

```
[csup]
libpthread.so.2        libkse.so.2
libpthread.so          libkse.so
```

Подобным же образом можно определить подстановку библиотек для всех программ в каталоге, указав в квадратных скобках имя каталога с завершающим символом слэша. В следующем примере определяется подстановка альтернативной библиотеки для всех программ в каталоге:

```
[/home/oracle/bin/]
libpthread.so.2        libkse.so.2
libpthread.so          libkse.so
```

С помощью файла *libmap.conf* можно выполнять подстановку разделяемых библиотек произвольным образом. Обычно этот прием применяется для подстановки библиотек реализации многопоточной модели исполнения (и иногда для программ, запускаемых в режиме совместимости с Linux), тем не менее он позволит вам «подсовывать» своим программам любые библиотеки. Разумеется, настроить библиотеки можно и с помощью переменной окружения `LD_LIBRARY_PATH`, но это касается только тех пользователей, кто может задать значение этой переменной.

Запуск программного обеспечения из чужой ОС

Традиционно в операционной системе применяется программное обеспечение, написанное специально для нее, и программы могут выполняться только на той платформе, для которой они разработаны. Многие неплохо заработали, изменяя программы, написанные для одной

платформы, для работы в другой системе. Такой процесс называется *переносом* (*porting*), или «портированием». Как администратор FreeBSD, вы можете по-разному применять программы, написанные для другой платформы. Самый эффективный способ состоит в том, чтобы заново скомпилировать исходный код, сделав эти программы «родными» для FreeBSD. Если это невозможно, «неродную» программу можно запустить под управлением эмулятора, такого как Wune, или реализовав машинный двоичный интерфейс прикладных программ (Application Binary Interface, ABI) «родной» платформы.

Перекомпиляция

Многие программы в коллекции «портов» на самом деле представляют собой «порты» программ, первоначально разработанных для других платформ (именно поэтому она и называется коллекцией *«портов»*). Программы, написанные для Linux, Solaris и других разновидностей UNIX, зачастую можно собрать из исходного кода (перекомпилировать). Чтобы эти программы выполнялись во FreeBSD без сучка и задоринки, требуется внести в их исходный код минимальные изменения, а порой можно и вовсе обойтись и без таковых. Просто взяв исходный код и собрав его на машине FreeBSD, можно запускать эти программы во FreeBSD, словно «родные».

Перекомпиляция дает лучшие результаты, если платформы похожи. Например, FreeBSD и Linux предоставляют много идентичных системных функций: они собраны на стандартных функциях C, используют сходные инструменты сборки и т. д. Однако с годами различные платформы UNIX отдалились друг от друга. В каждой версии UNIX реализованы новые функции, которым, в свою очередь, нужны новые библиотеки и функции. Если той или иной программе необходимы особые функции, ее не удастся собрать на других платформах. Отчасти стандарт POSIX был введен для смягчения этой проблемы. POSIX – это стандарт, описывающий минимально приемлемую систему UNIX и UNIX-подобные операционные системы. Программы, в которых применяются только POSIX-совместимые системные вызовы и библиотеки, можно непосредственно переносить в любую другую операционную систему, реализующую POSIX. Большинство производителей UNIX разрабатывают программное обеспечение в соответствии с POSIX. Самое трудное – добиться того, чтобы разработчики следовали стандарту POSIX. Многие разработчики, создающие программы с открытым кодом, заботятся только о том, чтобы их программы запускались на их любимой платформе. Например, есть довольно много программ для Linux, не совместимых с POSIX. В свою очередь, код, полностью соответствующий POSIX, не может вобрать в себя преимущества специальных функций, предлагаемых той или иной операционной системой.

Например, во FreeBSD есть сверхэффективный системный вызов чтения данных kqueue(2). Вместо него другие системы применяют select(2)

и poll(2). Вопрос, которым задаются разработчики приложений, состоит в том, что именно использовать: kqueue(2) – при этом программа молниеносно выполняется во FreeBSD и не работает в других системах, или select(2) и poll(2) – программа выполняется медленнее, но сможет работать и на других платформах. Конечно, разработчик мог бы поднапрячься и реализовать поддержку как kqueue(2), так и select(2) и poll(2), но, к сожалению, задача осчастливить пользователей не входит в круг интересов разработчика.

Проект FreeBSD выбирает средний путь. Если программу можно скомпилировать и запустить должным образом во FreeBSD, команда программистов, разрабатывающих «порты», реализует такую возможность. Если программе требуются незначительные заплатки, члены команды создают их и высылают разработчику этой программы. Большинство разработчиков программ охотно принимают заплатки, позволяющие поддерживать другую операционную систему. У них может не быть той или иной ОС для тестирования, или эта система может быть им неизвестна, однако если подходящая с виду заплатка пришла из уважаемого источника, скорее всего, она не будет отвергнута.

Эмуляция

Если для поддержки FreeBSD программа потребовала бы значительной переделки либо исходный код просто недоступен, возможен другой вариант: эмуляция. *Программа эмуляции* преобразует системные вызовы одной операционной системы в системные вызовы локальной операционной системы. Программы, запускаемые в эмуляторе, работают, словно в «родной» системе. Однако преобразование системных вызовов означает дополнительные системные издержки, снижающие скорость выполнения программ в эмуляторе.

FreeBSD поддерживает множество разнообразных эмуляторов, большая часть которых представлена в коллекции «портов» (каталог */usr/ports/emulators*). В большинстве случаев эмуляторы полезны для обучения и развлечений. Если у вас есть неутолимое желание вновь запустить старую игру Commodore 64, можно установить */usr/ports/emulators/frodo*. (Также можно попробовать установить дисковод C64 и заставить его работать во FreeBSD. При этом вы узнаете о дисках больше, чем когда-либо.) Есть эмуляторы Nintendo GameCube (*/usr/ports/emulators/gcube*), PDP-11 (*/usr/ports/emulators/sim*) и т. д. Но поскольку эмуляторы практически не нужны на сервере, подробно они здесь не рассматриваются.

Реализация ABI

В дополнение к перекомпиляции и эмуляции есть последний вариант, благодаря которому система FreeBSD снискала известность: реализация *машинного интерфейса прикладных программ* (*Application Binary Interface, ABI*). ABI – это часть ядра, предоставляющая программам

различные сервисы – от доступа к звуковым картам и чтения файлов до вывода изображения на экран и запуска других программ – все, что может понадобиться программам. С точки зрения программ, ABI – это операционная система. Полностью реализовав ABI той или иной операционной системы в своей системе, вы сможете выполнять «неродные» программы так, как они выполняются на «родной» платформе.

Хотя реализация ABI часто трактуется как эмуляция, это не так. Предоставляя ABI, система FreeBSD не эмулирует системные вызовы, а реализует их естественным образом. В этом случае отсутствует программное обеспечение, которое транслировало бы системные вызовы «чужой» операционной системы в их эквиваленты FreeBSD или вызовы библиотечных функций в эквивалентные вызовы аналогичных библиотек FreeBSD. Ко всему прочему, было бы неверным сказать, что «FreeBSD реализует Linux» или «FreeBSD реализует Solaris». На самом деле, после создания этой технологии никому не удалось в двух словах описать, что же сделала команда FreeBSD. Даже сегодня неясно, как объяснить это вкратце. Вы можете сказать, что FreeBSD реализует интерфейс системных вызовов Linux и включает поддержку перенаправления вызовов в соответствующий интерфейс системных вызовов. Чаще всего применяют термин *режим* (*mode*), например «режим Linux» или «режим System V».

Трудности с эмуляцией ABI происходят из-за наложения имен. В большинстве операционных систем есть системные вызовы с общими именами, например read, write и т. д. Системный вызов read(2) во FreeBSD работает совершенно иначе по сравнению с системным вызовом read() в системе Windows. Когда программа вызывает read, FreeBSD должна отличать, какой это вызов – «родной» или «неродной». Системным вызовам можно дать различные имена, но тогда это будет нарушением стандарта POSIX. Либо надо предоставлять различные интерфейсы ABI и предписывать той или иной программе, какой ABI использовать. Именно такая схема *идентификации* реализована во FreeBSD.

Проверка типа исполняемого файла

Как правило, запуск программ в операционных системах выполняет специальная системная функция. Когда ядро отправляет программу этому модулю исполнения, он запускает программу.

Однако несколько десятилетий тому назад в BSD (в то время – UNIX) в систему запуска программ была добавлена специальная проверка. Если программа начинается с `#!/bin/sh`, то она передается не модулю исполнения, а командному интерпретатору. В BSD эта идея была логически продолжена: модуль исполнения взаимодействует с различными типами исполняемых файлов. Согласно своему типу, каждая программа перенаправляется соответствующему ABI. Таким образом, в системе BSD может быть несколько интерфейсов ABI, следовательно, она может поддерживать программы из различных операционных систем.

Отличительная черта такого механизма перенаправления – отсутствие издержек. Раз система так или иначе справляется с запуском программы, то почему бы ей не выбирать и ABI? В конце концов, исполняемые файлы, созданные для различных операционных систем, можно идентифицировать по характерным признакам; механизм идентификации и перенаправления делает этот процесс прозрачным для конечного пользователя. Идентификация двоичных исполняемых файлов называется *брендингом* (*branding*), или *типизацией*. Двоичные исполняемые файлы FreeBSD несут в себе опознавательный признак (клеймо) *FreeBSD*, а файлы из других операционных систем различаются по признакам принадлежности к соответствующей операционной системе.

Поддерживаемые ABI

Благодаря возможностям перенаправления программ разным ABI система FreeBSD способна запускать программы Linux и SVR4, как будто они были скомпилированы в ней самой. В ранних версиях FreeBSD имелась и возможность запускать исполняемые файлы из операционных систем OSF/1 и SCO, но потребность в поддержке этих платформ существенно снизилась.[1]

SVR4, или System V Release 4, была последним основным «выпуском» UNIX от AT&T. Она встречается в ранних версиях Solaris и SCO UNIX. (По некоторым отзывам, в режиме SVR4 системы FreeBSD некоторые программы SCO выполняются быстрее, чем в самой SCO UNIX.) В настоящее время программное обеспечение для SVR4 уже практически не применяется, но вы должны знать, что его все еще можно использовать.

Режим Linux, также известный как *Linuxulator*, позволяет запускать программы Linux во FreeBSD. Этот ABI был протестирован наиболее тщательно, потому что исходный код Linux доступен, а ABI хорошо документирован. На самом деле режим Linux работает так хорошо, что на него опираются многие программы из коллекции «портов». Я спокойно использую версии для Linux таких программ, как Macromedia Flash и Adobe Acrobat Reader, и даже коммерческое программное обеспечение, такое как PGP Command Line, прекрасно работает в режиме Linux. Основное внимание мы уделим именно режиму Linux как наиболее ценному для среднего пользователя.

Библиотеки программ из других систем

Хотя реализация ABI решает одну основную задачу, но программам требуется не только ABI. Большинство программ не смогут работать в отсутствие поддержки разделяемых библиотек. Какой бы ABI ни

[1] Операционная система OSF/1 привязана к уже не существующему аппаратному обеспечению (потрясающий процессор Alpha), а SCO UNIX, похоже, не перенесла позора.

применялся, обеспечить доступ к пространству пользователя данной платформы необходимо.

Для работы в режиме SVR4 потребуется компакт-диск Sun Solaris 2.5.1. Ранние версии операционной системы Solaris являются коммерческими продуктами, поэтому проект FreeBSD не может просто так взять и передать их вам. Это вполне преодолимо, но делает использование режима SVR4 более трудоемким и дорогостоящим. Если у вас имеется этот компакт-диск, загляните в */usr/ports/emulators/svr4_base*.

Библиотеки для режима Linux – самые доступные. Поскольку препятствия к доступу незначительны, мы рассмотрим совместимость с Linux более подробно. Изучив режим Linux, вы сможете применить знания для реализации любого ABI.

Режим Linux

Чтобы установить Linuxulator, выберите программы, которым требуется режим Linux, и установите их. Система «портов» самостоятельно определит, что для работы программе требуется режим Linux, и установит соответствующее программное обеспечение Linux. Например, для просмотра документов в формате PDF я использую Adobe Acrobat Reader (*/usr/ports/print/acroread7*). Попытка установить эту программу автоматически вызовет установку режима Linux.

В ходе установки «порта» будет загружено и установлено в каталог */usr/compat/linux* множество системных файлов Linux. Вместо библиотек FreeBSD программы для Linux будут использовать библиотеки из этого каталога.

Кроме того, будет загружен модуль ядра, обеспечивающий работу режима Linux. Чтобы этот модуль автоматически загружался при загрузке FreeBSD, добавьте в файл *rc.conf* следующую строку:

```
linux_enable="YES"
```

Вот и все! Теперь вы можете запускать программы для Linux без дополнительной настройки. Осталось лишь ликвидировать незначительные шероховатости режима Linux, которые, к сожалению, имеют место.

Библиотеки Linuxulator

Поддержку Linux ABI обеспечивает модуль ядра *linux.ko*, для работы Linuxulator тоже требуется несколько библиотек Linux. В каталоге */usr/compat/linux* вы увидите примерно такой список каталогов:

```
# ls
bin    etc     lib     media   mnt     opt     proc    sbin    selinux srv
sys    usr     var
```

Похоже на содержимое корневого каталога FreeBSD, не правда ли? Если присмотреться внимательнее, можно заметить, что содержимое

Что такое linux_base?

Вам нередко будут встречаться упоминания чего-то под названием *linux_base*. Раньше поддержка режима Linux устанавливалась из «порта» с именем linux_base, и лишь после этого устанавливалось программное обеспечение, работающее в этом режиме. Современная операционная система FreeBSD делает это автоматически, незаметно для пользователя, что позволяет устанавливать необходимое программное обеспечение, не заботясь о предъявляемых требованиях. Различные «порты» linux_base, которые можно увидеть в каталоге */usr/ports/emulators*, предназначены для разработчиков и проверки совместимости, но не для того, чтобы вы их устанавливали.

/usr/compat/linux в общих чертах схоже с содержимым основной системы FreeBSD. Здесь можно найти те же программы, что и в базовой системе FreeBSD.

Приверженец Linux сразу отметит, что содержимое «порта» linux_base лишь отчасти повторяет типичную систему Linux, потому что каждый «порт» linux_base устанавливает лишь то, что ему нужно для работы. При установке новых «портов» система расширяет набор библиотек Linux, если это требуется.

По возможности программы режима Linux пытаются «остаться» в */usr/compat/linux*, что отчасти напоминает незапертую клетку (глава 9). Когда система выполняет программу Linux, вызывающую другие программы, Linux ABI сначала ищет их в */usr/compat/linux*. Если в этом каталоге программа не найдена, то ABI ищет ее в основной системе FreeBSD. Например, представим программу Linux, которая вызывает ping(8). Сначала ABI будет искать программу ping в */usr/compat/linux*. Обнаружив, что ее там нет, ABI обратится к основной системе FreeBSD, найдет */sbin/ping* и использует ее. Данная возможность позволяет сократить число библиотек Linux, устанавливаемых режимом Linux.

Другой пример. Предположим, программа Linux хочет вызвать sh(1). Linux ABI сначала проверит */usr/compat/linux* и найдет */usr/compat/linux/bin/sh*, а затем запустит ее вместо «родной» программы */bin/sh* системы FreeBSD.

Тестирование режима Linux

Теперь, когда вы получили некоторое представление о том, что устанавливается для поддержки режима Linux, можно протестировать его работу. Запустите командную оболочку Linux и спросите ее о типе операционной системы:

```
# /usr/compat/linux/bin/sh
sh-3.00$ ❶uname -a
❷Linux stretchlimo.blackhelicopters.org ❸2.4.2 ❹FreeBSD 7.0-CURRENT #7: Wed
Apr 23 21:00:47 EDT 2008 i686 i686 i386 GNU/Linux
sh-3.00$
```

На запрос типа операционной системы ❶ командный интерпретатор отвечает, что это операционная система Linux ❷, основанная на ядре Linux 2.4.2 ❸ с названием FreeBSD ❹. Круто, да?

Однако не нужно забывать, что режим Linux – это не полное окружение Linux. В режиме Linux невозможно выполнить кросс-компиляцию программ. Вы сможете решать только самые общие задачи.

Идентификация и типизация

Типизировать исполняемые файлы гораздо проще, чем клеймить быков, но не менее опасно. Большинство двоичных исполняемых файлов в современных UNIX-подобных операционных системах имеют формат ELF, который предусматривает область для комментариев. В эту область и ставится клеймо, указывающее тип программы. Операционная система FreeBSD назначает каждой программе свой ABI, нанося на нее клеймо. Если двоичный файл не имеет клейма, предполагается, что это исполняемый файл FreeBSD. FreeBSD распознает три клейма: FreeBSD, Linux и SVR4.

Посмотрим и изменим клеймо с помощью утилиты brandelf(1):

```
# brandelf /bin/sh
File '/bin/sh' is of brand 'FreeBSD' (9).
```

Ничего удивительного. Это двоичный исполняемый файл из операционной системы FreeBSD, поэтому он будет запускаться под управлением FreeBSD ABI. Попробуем посмотреть клеймо программы Linux:

```
# brandelf /usr/compat/linux/bin/sh
File '/usr/compat/linux/bin/sh' is of brand 'Linux' (3).
```

Если имеется «неродная» программа, которая не желает запускаться, следует проверить ее клеймо. Если клеймо отсутствует или оно неправильное, то, возможно, вы обнаружили причину проблемы: FreeBSD пытается запустить программу под управлением «родного» FreeBSD ABI. Измените клеймо вручную с помощью команды `brandelf -t`. Например, типизация программы Linux выполняется следующим образом:

```
# brandelf -t Linux /usr/local/bin/имя_программы
```

При следующем запуске программы FreeBSD выполнит ее под управлением Linux ABI с применением библиотек Linux, и тогда программа должна заработать.

linprocfs

В операционной системе Linux используется файловая система процессов, или *procfs*. В операционной системе FreeBSD несколько лет тому назад эта файловая система была ликвидирована из соображений безопасности, но некоторые программы Linux, работающие в режиме Linux, требуют присутствия этой файловой системы. Применять программное обеспечение Linux, требующее наличия procfs, рискованно. В операционной системе FreeBSD файловая система procfs Linux доступна как linprocfs(5).

Чтобы разрешить использование linprocfs(5), нужно после установки режима Linux добавить в файл */etc/fstab* следующую строку:

```
linproc        /compat/linux/proc      linprocfs rw       0       0
```

FreeBSD загружает модули ядра, обеспечивающие поддержку файловой системы, по требованию, поэтому для активизации linprocfs(5) достаточно просто ввести команду `mount /compat/linux/proc`.

Отладка режима Linux с помощью truss(1)

Режим Linux – это не Linux, что только осложняет поиск причин, вызывающих ошибки в работе программ. Многие программы и без того выводят малопонятные сообщения об ошибках, а режим Linux может усилить неясность. Лучшим инструментом для отладки режима Linux из тех, которые мне когда-либо встречались, является truss(1) – трассировщик системных вызовов FreeBSD. Некоторые специалисты говорят, что применять truss(1) для этих целей – все равно что поставить 12-цилиндровый двигатель от грузовика «Бульдог» на фольксваген «Жук», но после тщательного и вдумчивого изучения я решил, что не согласен с ними. Это работает. Познакомившись с truss(1) поближе, вы удивитесь тому, что раньше могли обходиться без этой утилиты.

Утилита truss(1) точно идентифицирует, к каким системным вызовам обращается программа и какие результаты она получает. Не забывайте, что системные вызовы – это программный интерфейс ядра. Когда программа пытается выполнить некоторую операцию в сети, открыть файл или даже выделить память, она производит системный вызов. Это делает утилиту truss(1) превосходным инструментом поиска причин, вызывающих ошибки в работе программы. Программы производят большое число системных вызовов, а это означает, что truss(1) генерирует огромный объем данных, которые лучше обрабатывать с помощью script(1).

Например, как-то раз я установил в операционную систему FreeBSD версию PGP Command Line для Linux.[1] Согласно справочному руко-

[1] Прочтите книгу «PGP & GPG» (No Scratch Press, 2006), которую написал ваш покорный слуга. Чтобы написать всего лишь пару глав, мне нужно было либо запустить PGP в режиме Linux, либо устанавливать полноценную операционную систему Linux. Думаю, мой выбор для вас очевиден.

водству, при запуске с ключом -h программа выводит инструкции по пользованию программой. У меня имеется полное руководство, тем не менее при запуске в режиме Linux программа сообщает:

```
# pgp -h
/home/mwlucas:unknown (3078: could not create directory, Permission denied)
(Перевод: /home/mwlucas:unknown (3078: невозможно создать каталог, доступ
запрещен))
```

Хорошая новость: программа запускается! Плохая новость: она где-то «спотыкается». Какой каталог она пытается создать? Это можно узнать с помощью truss(1). Запустите сеанс script(1), запустите программу и выйдите из script(1).

Созданный файл будет заполнен сотнями или тысячами строк – как отыскать среди них причину ошибки? Ищите часть сообщения об ошибке или строку ERR. В данном случае я искал строку directory и нашел ее недалеко от конца полученного файла:

```
...
linux_open("/home/mwlucas/.pgp/randseed.rnd",0x8002,00) ERR#2 'No such file
or directory'
linux_open("/home/mwlucas/.pgp/randseed.rnd",0x80c1,0600) ERR#2 'No such
file or directory'
linux_open("/home/mwlucas/.pgp/randseed.rnd",0x8002,00) ERR#2 'No such file
or directory'
...
```

Ага! Программа не может открыть файл в каталоге */home/mwlucas/.pgp*, потому что этот каталог отсутствует. Как только я создал каталог *.pgp*, программа стала не только запускаться с ключом -h, но прекрасно справлялась и с более сложными заданиями.

В дополнение к truss(1) можно было бы использовать утилиты ktrace(1) и linux_kdump (*/usr/ports/devel/linux_kdump*) для анализа

Коммерческое программное обеспечение для Linux и режим Linux

Не забывайте, что производители коммерческого программного обеспечения для Linux не заботятся о поддержке режима Linux в операционной системе FreeBSD. Если вы работаете в организации, которая заключает с клиентами соглашения на обслуживание, и в случае появления проблем вам грозит денежный штраф – семь раз подумайте, прежде чем использовать режим Linux. Главное преимущество коммерческого программного обеспечения состоит в том, что у него имеется производитель, которому можно пожаловаться, если что-то пойдет не так, но режим Linux отменяет это преимущество.

полученных результатов. Я рекомендую попробовать оба инструмента и выбрать наиболее удобный для себя.

Запуск программного обеспечения для чужой архитектуры

Все большую популярность приобретают 64-битовые архитектуры, особенно с появлением i386-совместимой 64-битовой архитектуры AMD. Способность работать под управлением 32- и 64-битовых операционных систем является, пожалуй, огромным преимуществом аппаратного обеспечения с коммерческой точки зрения. Однако программное обеспечение должно учитывать особенности 64-битовых платформ. Несмотря на то что мир свободного программного обеспечения уже много лет работает с 64-битовыми платформами благодаря Sun Solaris, есть много программ, созданных для 32-битовых операционных систем. Если вы работаете с операционной системой на платформе amd64, рано или поздно вам встретится программное обеспечение, доступное только для аппаратного обеспечения i386. Что можно сделать в этом случае?

Вот хорошая новость для вас: при наличии параметра `COMPAT_IA32` в ядре (уже входит в состав GENERIC) FreeBSD/amd64 способна исполнять любое программное обеспечение FreeBSD/i386. Единственное, чего нельзя, – это использовать разделяемые библиотеки FreeBSD/amd64 для программ FreeBSD/i386. То есть чтобы запустить сложную 32-битовую программу на 64-битовом компьютере, вам придется обеспечить доступность 32-битовых библиотек. Такая возможность поддерживается операционной системой – в файле *rc.conf* есть параметры настройки ldconfig(8), такие как `ldconfig32_paths` и `ldconfig_local32_dirs`. В 64-битовой версии операционной системы эти параметры определяют местоположение 32-битовых библиотек.

Еще удивительнее то, что FreeBSD/amd64 может запускать 32-битовые программы для Linux! Так как 32-битовое программное обеспечение для Linux более широко распространено, чем 64-битовое, в режиме Linux операционной системы FreeBSD можно работать с 32-битовым программным обеспечением. Для этого достаточно включить в настройках ядра параметр `COMPAT_LINUX32`. Никакая дополнительная или необычная настройка не требуется – режим Linux в архитектуре amd64 работает точно так же, как и в архитектуре i386. В настоящее время FreeBSD не поддерживает работу с 64-битовым программным обеспечением для Linux, но это и не требуется, так как все, что доступно для 64-битовой версии Linux, доступно и для 64-битовой версии FreeBSD.

Тему управления программным обеспечением можно изучать бесконечно, однако теперь вы знаете достаточно, чтобы двигаться дальше. Давайте теперь рассмотрим вопросы обновления FreeBSD.

13

Обновление FreeBSD

Обновление (upgrade) сетевого сервера – дело хлопотное. Я могу справиться с персональным компьютером, зачудившим после обновления, но когда от одной системы зависит целая компания или сотни клиентов, даже мысль о том, чтобы ее тронуть, кажется мне кошмаром. Если даже самого опытного системного администратора поставить перед выбором – обновление системы или пытки каленым железом, он сядет и задумается. Несмотря на то что в некоторых версиях UNIX процедуры обновления просты, для их выполнения требуется несколько часов и известная доля удачи.

С другой стороны, процедура обновления – одно из крупнейших преимуществ FreeBSD. Например, у меня есть несколько серверов с установленными разными версиями FreeBSD, на которые было наложено множество заплаток. Очень немногие администраторы Windows обновляют сервер с Windows 2000 до Windows 2003. (Между прочим, они не зря получают зарплату.) Я списываю системы FreeBSD, только если они настолько устарели, что риск аппаратных сбоев лишает меня сна. На одном из моих серверов изначально была установлена FreeBSD 2.2.5. Затем она была успешно обновлена до FreeBSD 3 и, наконец, до FreeBSD 4. К моменту выхода FreeBSD 4.8 жесткий диск повел себя странно, так что новую систему мне пришлось установить на новой машине. Этот компьютер проработал под управлением FreeBSD 5, а затем FreeBSD 6 до самой своей смерти.[1] Только раз я испытал неудобство – при переходе от одной основной версии к другой, то есть от FreeBSD 5 к FreeBSD 6. Это заняло у меня пару часов. И попробуйте сделать то же самое с другими операционными системами.

---

[1] «В серверной никто не услышит крик блока питания».

Версии FreeBSD

Почему обновлять FreeBSD относительно просто? Все дело в методе разработки FreeBSD. Это постоянно развивающаяся операционная система. Если после обеда вы загружаете определенную версию FreeBSD, она немного отличается от утренней версии. Разработчики по всему миру непрерывно вносят изменения и улучшения, поэтому традиционная система нумерации версий, принятая для менее открытого программного обеспечения, здесь неприменима. В любой момент можно получить несколько различных версий FreeBSD: «выпуск» (release), ветку с исправленными ошибками (errata branch), -current, -stable и моментальную копию.

«Выпуск»

На рабочий сервер определенно стоит установить выпущенную версию FreeBSD и затем наложить заплатки до текущей ветки с исправленными ошибками.

«Выпуск» FreeBSD имеет стандартный номер версии, подобно любому другому программному обеспечению: 5.5, 6.3, 7.0. «Выпуск» – это просто копия самой стабильной версии FreeBSD на тот или иной момент времени. Три или четыре раза в год группа разработчиков, ответственная за «выпуски» (Release Engineering team), просит разработчиков приостановить внесение существенных изменений и сконцентрироваться на устранении выявленных неполадок. После этого группа Release Engineering отбирает несколько вариантов кода и предлагает их для всеобщего тестирования, а тщательно протестированный программный код получает номер «выпуска». Далее разработчики снова возвращаются к своим обычным проектам.[1]

Ветка с исправленными ошибками

Ветка с исправленными ошибками (*errata branch*) – это определенный «выпуск» FreeBSD плюс заплатки безопасности и исправления ошибок для данного «выпуска». Несмотря на все усилия разработчиков FreeBSD обеспечить отсутствие ошибок в каждом «выпуске», эта цель недостижима. Бывает, что неизвестный злоумышленник обнаруживает новую уязвимость в системе безопасности спустя неделю после выхода очередной версии FreeBSD. Тогда группа обеспечения безопасности выпускает исправления для тех, кто стремится обеспечить максимальную стабильность и безопасность своих систем.

У каждого «выпуска» имеется собственная ветка с исправленными ошибками. Например, FreeBSD 7.0-errata отличается от FreeBSD 7.1-errata, а переход от одной к другой так же труден, как и от FreeBSD 7.0

[1] Что бы вы ни думали, «недовольные пользователи» – не обычный проект для разработчиков FreeBSD. Это так, приложение.

к FreeBSD 7.1. Изменения в API и ABI настолько велики, что полностью, абсолютно не допускают такой переход. Приложения, работающие в основном «выпуске», точно так же будут работать и в любой версии ветки с исправленными ошибками для этого «выпуска». Для обеспечения максимальной стабильности следует оставаться в ветке с исправленными ошибками, предназначенной для установленной версии FreeBSD.

К моменту написания этих строк проект FreeBSD обеспечивал поддержку веток исправлений до двух лет со дня выхода основного «выпуска», но такое положение вещей может измениться. Загляните на страницу *http://www.freebsd.org/security* или в почтовую рассылку *FreeBSD-announce@FreeBSD.org*, где можно ознакомиться со списком обновлений и примечаниями, касающимися времени окончания поддержки той или иной версии. Безусловно, обладая доступом к исходному коду, вы можете поддерживать устаревший «выпуск» сколько хотите. Но не ждите, что коллектив разработчиков FreeBSD будет работать на вас до бесконечности!

FreeBSD–current

FreeBSD-current (текущая) – это передовая, самая последняя версия FreeBSD. Она содержит код, который первый раз представляется публике. Хотя разработчики и имеют тестовые серверы и высылают заплатки для ознакомления перед их наложением, тем не менее эти исправления доходят далеко не до всех пользователей FreeBSD-current. Версия FreeBSD-current получает первые экспертные оценки и время от времени подвергается радикальным изменениям, которые добавляют забот опытным системным администраторам.

Версия FreeBSD-current доступна для разработчиков, тестеров и заинтересованных сторон, но не предназначена для всеобщего использования. Ответы на вопросы пользователей о -current очень скудны, поскольку разработчикам просто некогда помогать в настройке веб-броузера – их внимания требуют тысячи более существенных замечаний. Пользователи должны справиться сами или терпеливо ждать, пока кто-нибудь не устранит эти неполадки.

Хуже того, настройки по умолчанию в версии -current включают массу отладочного программного кода, предусматривают вывод особых предупреждений и активируют прочие функции, относящиеся к отладке. Все это делает версию -current более медленной, чем любая другая версия FreeBSD. Вы можете отключить механизмы отладки, но в этом случае не получите грамотный отчет об ошибке, когда появится какая-нибудь проблема. Значит, у вас будут лишние сложности. За дополнительной информацией об отладке в версии -current обращайтесь к файлу */usr/src/UPDATING*.

Если вы не можете читать код C и командного интерпретатора, отлаживать ОС, терпеть непредсказуемое поведение функций и ждать, пока

кто-нибудь устранит возникшие неполадки, то версия -current не для вас. Храбрецам, желающим испробовать -current, всегда рады. Дорога открыта каждому, кто готов посвятить немало времени изучению и отладке FreeBSD или желает получить урок смирения. Это скорее не наставление «так делать нельзя», а утверждение «все в ваших руках». Никто не запрещает вам работать с версией -current, но рассчитывать придется только на себя. Версия -current – не всегда передовая, но здесь порой небезопасно. В общем, вы предупреждены.

Желающие опробовать -current *должны* подписаться на почтовые рассылки *FreeBSD-current@FreeBSD.org* и *cvs-src@FreeBSD.org*. Это рассылки с большим трафиком – несколько сотен предупреждений, извещений и комментариев в день. Если вы читаете эту книгу, скорее всего, вам пока рано писать сообщения в эту рассылку – просто читайте и учитесь. Если кто-то вдруг обнаружит, что новейшая заплатка для файловой системы превращает жесткие диски в зомби Ктулху, то информация об этом появится именно здесь.

Замораживание кода –current

Каждые год-полтора для FreeBSD-current проводится месячник *замораживания кода* (*code freeze*), когда не допускаются несущественные изменения и устраняются все оставшиеся неполадки. Цель – стабилизировать последнюю версию FreeBSD и ликвидировать шероховатости. По завершении этого процесса (или вскоре после него) -current становится новым «выпуском» *.0* системы FreeBSD.

После одного или двух «выпусков» из новой версии -current ответвляется новая, основная версия -stable. Например, в свое время FreeBSD 6.0 была -current, так же как и FreeBSD 7.0.

После появления «выпуска» .0 работа продолжается в двух направлениях: над версиями FreeBSD-current и FreeBSD-stable.

FreeBSD–stable

FreeBSD-stable (или просто -*stable*) – передний край для среднего пользователя. Эта версия содержит последний код, аттестованный экспертами. Версии FreeBSD-stable положено быть устойчивой и надежной; она не должна требовать от пользователя значительного внимания. Как только тот или иной код в -current тщательно протестирован, он может влиться в версию -stable. На версию -stable можно безопасно перейти почти в любое время; это своего рода FreeBSD-beta.

Со временем различия между -stable и -current увеличиваются, и в какой-то момент назревает необходимость ответвить от -current новую версию -stable. Предыдущая -stable будет активно поддерживаться несколько месяцев, пока не закрепится новая -stable. Одни пользователи захотят немедленно обновить систему до новой версии -stable, другие будут более осмотрительны. После выхода одной или двух новых версий -stable прежняя версия -stable признается устаревшей, а пользова-

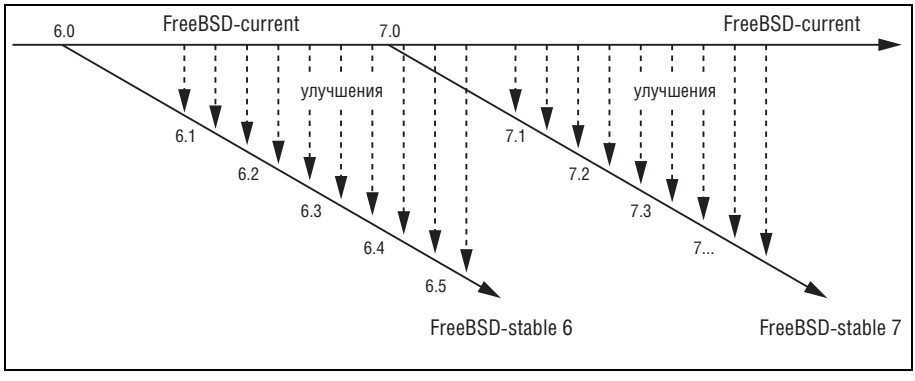

Рис. 13.1. *Направления разработки FreeBSD*

тели увидят приглашение обновить системы до новой -stable. В конце концов, изменения в устаревшей версии -stable будут сводиться лишь к исправлению критических ошибок, и наконец ее развитие будет остановлено полностью. Весь этот процесс можно увидеть на рис. 13.1.

Время от времени на версию -stable «наводят лоск» и подвергают ее испытаниям; разработчики останавливают перенос изменений из -current в -stable и все внимание обращают на тестирование. Когда каждый разработчик удовлетворится качеством версии, подготавливается новый «выпуск». Например, четвертый «выпуск» FreeBSD 7 – это FreeBSD 7.3. FreeBSD 7.3 – это всего лишь веха на пути развития FreeBSD-stable 7.

Пользователям FreeBSD-stable следует подписаться на почтовую рассылку *FreeBSD-stable@FreeBSD.org*. Трафик в этой почтовой рассылке модерируется, большую его часть составляют вопросы и ответы, которые в действительности должны были бы направляться в рассылку *-question@*, тем не менее в эту рассылку направляются важные

Стабильность версии –stable

Слово *stable* (стабильная) относится к программному коду, составляющему основу самой операционной системы FreeBSD. Очередная версия -stable гарантирует не устойчивость системы, а только отсутствие существенных изменений в основном программном коде ОС. Прикладной и двоичный программные интерфейсы (API и ABI) наверняка не изменятся. Разработчики прилагают все усилия, чтобы сохранить устойчивость, но от ошибок никто не застрахован. Если вас волнует риск потери устойчивости – переходите на использование ветки с исправленными ошибками (errata).

сообщения от разработчиков, обычно озаглавленные HEADS UP. Внимательно относитесь к этим сообщениями; обычно они касаются изменений в системе, способных разрушить ваши планы на день, если заранее не знать о них.

Поглощение из текущей версии

Фраза *поглощение из текущей версии* (*merge from -current, MFC*) означает перенос функции или подсистемы из FreeBSD-current во FreeBSD-stable (или, что случается реже, в ветку с исправленными ошибками). Однако такому переносу подвергаются далеко не все новые функциональные возможности, потому что версия FreeBSD-current – это полигон, где обкатываются существенные изменения, многие из которых требуют отладки и тестирования, длящихся месяцами. Такие изменения не переносят в стабильную версию, потому что это плохо повлияет на пользователей версии -stable, в первую очередь ожидающих стабильности. Новые драйверы, исправления ошибок и некоторые улучшения могут быть перенесены, но существенные изменения, которые могут отрицательно сказаться на работе пользовательских приложений, не переносятся.

Моментальные копии

Каждый месяц, или что-то около того, группа разработчиков FreeBSD Release Engineering выпускает моментальные копии (snapshots) версий -current и -stable, выкладывая их на FTP-сайте. Моментальные копии – это просто вешки на пути развития; они не подвергаются какому-то специальному тестированию. Моментальная копия не предполагает такого внимания к качеству, как «выпуск», но может служить хорошей отправной точкой для тех, кто интересуется версиями -current и -stable. Качество моментальных копий почти не контролируется, многие разработчики даже не знают о выходе новой моментальной копии, пока не увидят ее на FTP-сервере. Вам могут встретиться баги. Вам могут встретиться ошибки. Испытания, через которые вы пройдете, заставят вашу мать поседеть, если, конечно, вы сами еще не довели бедняжку до этого.

FreeBSD и тестирование

Каждая версия и «выпуск» FreeBSD подвергается различным испытаниям. Отдельные разработчики проверяют качество своей работы на собственном аппаратном обеспечении и просят друг друга перепроверить их работу. Если продукт достаточно сложный, они могут использовать личное хранилище исходного кода, чтобы организовать доступ сообщества к своей работе, прежде чем отправить ее в -current. Компания Coverity пожертвовала группе разработчиков FreeBSD аналитическое программное обеспечение для постоянного автоматического тестирования и отладки, чтобы отыскивать ошибки еще до того, как они попадут к пользователям. Такие корпорации, как Yahoo!, Sentex и iX

Systems пожертвовали проекту FreeBSD высококачественную аппаратуру для нужд тестирования, предоставив сервер группе, занимающейся вопросами безопасности, и высокопроизводительный сетевой кластер для разработчиков ядра. Несколько наиболее ценных разработчиков FreeBSD сделали тестирование своей главной задачей в рамках проекта FreeBSD.

Однако проект, развивающийся исключительно за счет труда сотен добровольных разработчиков, не в состоянии приобрести все разновидности компьютеров, какие только производятся, так же как не может проводить испытания под всеми возможными видами нагрузки. Проект FreeBSD целиком опирается на пожертвования производителей аппаратного обеспечения, заинтересованных в том, чтобы их оборудование работало под управлением FreeBSD, на помощь компаний, стремящихся использовать FreeBSD на имеющихся у них аппаратных средствах, и на пользователей.

Наиболее существенная помощь поступает от пользователей, имеющих реальное оборудование и испытательные полигоны, подвергающиеся настоящим рабочим нагрузкам. К сожалению, большинство из этих пользователей выполняют тестирование, только когда вставляют компакт-диск с дистрибутивом в компьютер, устанавливают и запускают систему. В этот момент уже поздно пытаться принести какую-либо пользу выпуску. Любые сообщения об ошибках, обнаруженных пользователями, могут помочь в подготовке следующего выпуска, но в то же время обновление до ветки с исправленными ошибками для этого выпуска может ликвидировать вашу проблему. Решение вполне очевидно – тестировать FreeBSD в реальных условиях необходимо до выхода выпуска. Предложения на тестирование новых выпусков -stable появляются в почтовой рассылке *FreeBSD-stable@FreeBSD.org*. Тестируя версии -stable и -current, вы сможете получить большую отдачу от FreeBSD.

Какую версию следует использовать?

-current, -stable, -errata, моментальные копии – от разнообразия голова кругом. Такая система может показаться сложной, но она оправдывает себя и обеспечивает необходимый уровень качества. Пользователь может быть уверен, что ветка с исправленными ошибками будет настолько стабильна, насколько это вообще возможно, и прошла экспертную оценку и всестороннее тестирование. Тот же самый пользователь знает, что в новых версиях -stable и -current ему будут доступны новые привлекательные возможности, если он готов пойти на риск, который несет в себе каждая новая версия. Так какую же версию предпочесть? Ниже приводятся мои рекомендации на этот счет:

Предприятие

Если ваша система обеспечивает работу предприятия, устанавливайте версию -stable и следите за выходом исправлений.

Тестирование

Администраторам, желающим узнать, как изменения в FreeBSD влияют на операционную среду, следует применять -stable на испытательной системе.

Разработка

Если вы разработчик операционной системы, обладатель массы свободного времени и крепких нервов или круглый дурак, то версия -current – для вас. Когда -current уничтожит вашу коллекцию MP3, разберитесь в проблеме и предложите свою заплатку.

Хобби

Если вы увлекающийся человек, запускайте любую версию! Просто помните об ограничениях того варианта, который вы выбрали. Новичкам в UNIX лучше выбрать -release. Как только почувствуете себя увереннее, обновите систему до -stable. Если у вас нет занятия получше и вам наплевать на свои данные – добро пожаловать в ряды мазохистов, выбравших -current!

Методы обновления

Система FreeBSD предоставляет три основных метода обновления: с помощью программы sysinstall, двоичных обновлений и исходного кода.

Поддержка двоичного обновления осуществляется через службу FreeBSD Update. Она чем-то напоминает службы обновления Windows, Firefox и других коммерческих программных продуктов. С помощью службы обновления FreeBSD Update можно обновить систему до ветки с исправленными ошибками.

sysinstall – это программа установки FreeBSD. Она позволяет обновить систему до того или иного выпуска, в составе которого распространяется. Например, программа установки из FreeBSD 7.5 предназначена для обновления до версии FreeBSD 7.5. Программа sysinstall служит для обновления от одного «выпуска» или моментальной копии до другого.

Защитите свои данные!

Глава 4 называется «Прочтите это раньше, чем что-нибудь испортите!» по вполне серьезным причинам. Обновление системы может привести к уничтожению данных. Создавайте резервную копию системы перед выполнением любого обновления! Я обновляю свой ноутбук каждую неделю, просто из интереса (см. выше замечание о круглом дураке и версии -current). Но прежде, чем обновить систему, я копирую все необходимые данные на другую машину. Копируйте данные на ленту, в файлы, куда угодно, но не выполняйте обновление, если у вас нет свежей резервной копии.

Метод, основанный на применении исходного кода, позволяет собирать программы, составляющие операционную систему FreeBSD, и устанавливать их на жесткий диск. Например, при наличии исходного кода для FreeBSD 7.5 можно обновить операционную систему до этой версии. Этот метод требует больше усилий, но обеспечивает гораздо более высокую гибкость. Этот метод применяется при работе с версией -stable или -current.

Двоичные обновления

Многие операционные системы предлагают двоичное обновление, если у пользователей есть возможность загрузить новые двоичные файлы для своих операционных систем. FreeBSD предоставляет похожую программу FreeBSD Update, которая обновляет систему до ветки с исправленными ошибками. С помощью FreeBSD Update нельзя выполнить обновление до версии -stable или -current – только до ветки с исправленными ошибками. Например, если имеется версия FreeBSD 7.0, то FreeBSD Update позволит обновить ее до версии FreeBSD 7.0-errata, но не до версий 7.0-stable или 7.1-release.

Программа FreeBSD Update предназначена для применения в стандартных системах, основанных на ядре GENERIC. Если система обновлялась из исходного кода, то FreeBSD Update здесь не подойдет. Аналогично, программа FreeBSD Updates предоставляет заплатки для ядра только в версии GENERIC. Если вы собрали собственное ядро, вам придется вручную встраивать в него обновления.

Двоичное обновление выполняется с помощью программы freebsd-update(8) в соответствии с настройками из файла */etc/freebsd-update.conf*.

/etc/freebsd-update.conf

Программа freebsd-update(8) создавалась с целью максимально облегчить ее использование среднему пользователю, поэтому изменять ее конфигурацию не рекомендуется. Впрочем, у вас могут оказаться не совсем обычные требования, поэтому здесь мы рассмотрим наиболее важные параметры настройки, которые можно найти в этом файле:

`KeyPrint 800...`

Параметр `KeyPrint` содержит криптографическую сигнатуру службы обновления. Если злоумышленнику удастся обнаружить уязвимость в службе FreeBSD Update и подменить заплатки троянскими версиями, то Проект FreeBSD должен будет ликвидировать уязвимость и выпустить новые криптографические ключи. В этом случае объявление об уязвимости будет отправлено в почтовую рассылку (и станет крупной новостью в мире информационных технологий). Другими словами, нет причин изменять это значение в обычных условиях, и не так много причин изменять его в необычных условиях.

`ServerName update.freebsd.org`

С помощью параметра `ServerName` программа frebsd-update(8) определяет, откуда загружать обновления. Проект FreeBSD предоставляет инструменты, необходимые для создания собственных обновлений, но они плохо документированы. Настройка собственного сервера обновлений – это для самых смелых.

`Components src world kernel`

По умолчанию FreeBSD Updates предоставляет самые последние заплатки для исходного кода в каталоге */usr/src*, для пользовательских библиотек и программ («мир») и для ядра GENERIC. Однако вам могут не потребоваться все эти компоненты. Заплатки на ядро и пользовательские компоненты необходимы, но вам могут не потребоваться заплатки на исходный код, установленный на вашей машине. В этом случае следует удалить запись `src` и принимать обновления только для ядра и пользовательских компонентов. Кроме того, можно настроить прием обновлений только для определенных частей исходного кода, как сказано на странице руководства freebsd-update.conf(5).

`UpdateIfUnmodified /etc/ /var/`

Служба FreeBSD Updates предоставляет обновления конфигурационных файлов в каталоге */etc*. Однако если вы уже изменяли эти файлы, скорее всего вам не потребуется обновлять их. Точно так же, файлы в каталоге */var* используются администраторами для настройки системы, и может оказаться нежелательным, чтобы программа FreeBSD Update изменяла содержимое этих файлов. FreeBSD Update накладывает заплатки на каталоги, перечисленные в параметре `UpdateIfUnmodified`, только если их содержимое не изменялось.

`MailTo root`

Если был запланирован автоматический запуск FreeBSD Update (как описано ниже в этой главе), программа freebsd-update(8) отправит результаты на учетную запись, указанную в параметре `MailTo`.

`KeepModifiedMetadata yes`

Есть вероятность, что администратор системы изменил права доступа или принадлежность системных файлов или команд. Тогда может оказаться нежелательным, чтобы программа freebsd-update(8) сбрасывала эти права доступа в значения по умолчанию. Если параметр `KeepModifiedMetadata` установлен в значение `yes` (да), freebsd-update(8) не затронет измененные права доступа.

Запуск freebsd-update(8)

Обновление системы с помощью двоичных обновлений производится в два этапа: загрузка обновлений и применение.

Чтобы загрузить последние обновления, запустите команду `freebsd-update fetch`.

```
# freebsd-update fetch
```

Далее программа выполнит поиск источника загрузки обновлений, сравнит криптографические ключи для этих источников и загрузит обновления. В завершение на экране должно появиться следующее сообщение:

```
The following files will be updated as part of updating to 7.0-RELEASE-p2:
(Перевод: В ходе обновления до версии 7.0-RELEASE-p2 были обновлены следующие
файлы:)
/boot/kernel/kernel
/etc/rc.d/jail
/usr/bin/dig
...
```

Файлы обновлений сохраняются в каталоге */var/db/freebsd-update*. Для установки загруженных файлов запустите команду `freebsd-update install`:

```
# freebsd-update install
Installing updates... done.
```

Вот и все. После перезагрузки системы вы увидите, что запущена новая версия с исправленными ошибками!

Планирование автоматического запуска двоичных обновлений

Лучше всего загружать и применять обновления регулярно в определенное время, например ежемесячно в дни технического обслуживания системы. Программа FreeBSD Updates поддерживает такую возможность и позволяет избежать перегрузки серверов ежечасными запросами. Команда `freebsd-update cron` предписывает системе загрузить обновления в случайно выбранный момент следующего часа. Вставьте эту команду в */etc/crontab*, чтобы загрузить обновления в течение выбранного часа. Это поможет снизить нагрузку на серверы загрузки. По окончании обновления администратор получит сообщение по электронной почте и сможет наметить перезагрузку системы на удобное ему время.

Оптимизация и настройка FreeBSD Update

При работе с FreeBSD Update обычно возникают два наиболее распространенных вопроса, касающихся нестандартных сборок FreeBSD и распространения обновлений по локальным серверам.

Многие администраторы собирают собственные версии FreeBSD для внутреннего использования. Нередко это стандартная версия FreeBSD с усеченными возможностями, как, например, NanoBSD, речь о которой пойдет в главе 20. Но в некоторых компаниях вносят гораздо более серь-

езные изменения. Если администратор удалит какие-либо файлы из стандартной версии FreeBSD, freebsd-update(8) не сможет их обновить.

Также во многих компаниях имеются внутренние сервера, где хранятся обновления. Система FreeBSD Update предусматривает возможность работы с кэширующими прокси-серверами. Поскольку все файлы снабжены цифровыми подписями, которые проверяются в обязательном порядке, их можно передавать по обычному протоколу HTTP, благодаря чему прокси-серверы могут их кэшировать. Тем самым снижается объем интернет-трафика.

Если вам действительно необходим собственный сервер FreeBSD Update, то программный код этого сервера вы найдете в хранилище CVS, в каталоге *projects*. Если вы не поняли, что это значит, то вам определенно не стоит пока создавать свой сервер обновлений.

Обновление с помощью sysinstall

Программа установки FreeBSD, sysinstall(8) также может выполнять обновление системы от одного «выпуска» до другого или от одной ветки с исправленными ошибками одного «выпуска» до другого «выпуска». В ходе обновления программа sysinstall просто замещает старые двоичные файлы на диске новыми, из следующего «выпуска». Например, если вы используете версию 7.1-errata, то sysinstall позволит обновить ее до версии 7.2. Следует заметить, что такой способ обновления наиболее предпочтителен при обновлении между близкими версиями. Обновление от версии 7.1 до 7.2 с помощью sysinstall(8) скорее всего завершится успехом, от 7.1 до 7.5 – может вызвать некоторые проблемы, от 7.1 до 9.0 – наверняка породит существенные проблемы.

Все необходимые инструкции по обновлению содержатся в документе *Install* на установочном компакт-диске. Обязательно прочитайте эти инструкции, прежде чем приступать к обновлению. Кроме того, поищите на сайте *http://www.freebsd.org* последние исправления, которые могут иметь отношение к этому обновлению. Наконец, не забудьте создать резервную копию.

Следует также помнить, что очень немногие разработчики выполняют обновление с помощью sysinstall. Эта утилита предназначена только для конечных пользователей, не желающих самостоятельно заниматься сборкой программ из исходного кода. Это означает, что у метода обновления с помощью sysinstall чуть больше острых углов по сравнению с другими методами, например с обновлением из исходного кода.

Проще всего обновить систему с помощью sysinstall, загрузившись с установочного компакт-диска с версией системы, до которой выполняется обновление. Далее надо пройти следующие этапы:

1. Когда на экране появится графическое меню установки, выберите пункт Upgrade an Existing System (Обновление существующей системы).

2. Программа sysinstall откроет инструкции по обновлению. Если вы еще не ознакомились с ними, сделайте это сейчас.
3. После этого следует выбрать тип обновляемого дистрибутива. Здесь лучше всего указать тот же тип, который был выбран во время первоначальной установки. Например, если ранее была установлена система в варианте Developer (для разработчика), то следует обновлять комплект компонентов, составляющих этот вариант установки. Выбор другого варианта может вызвать проблемы. Например, если первоначально была установлена FreeBSD 7.0 с полным комплектом компонентов, а обновление производится до FreeBSD 7.1 в минимальном составе, то в системе появятся новые программы из версии 7.1, а документация останется старая, из версии 7.0.
4. Программа sysinstall спросит, устанавливать ли коллекцию «портов». Если для обновления коллекции «портов» вы пользуетесь программами portsnap(8) или csup(1) (эти инструменты обсуждаются ниже в данной главе), то устанавливать ее не нужно. Если вы не обновляете коллекцию «портов» отдельно, то установите ее новую версию.
5. Вернувшись в меню Distribution Sets (тип установки), выберите пункт Exit (выйти) и покиньте меню.
6. Если для обновления был выбран вариант с исходным кодом системы, то sysinstall(8) сообщит, что не может обновить исходный код. Это нормально и можно продолжать обновление.
7. Далее будет задан вопрос о каталоге, в котором следует сохранить резервную копию текущего каталога */etc*. Не забывайте, что */etc* содержит жизненно важную информацию о конфигурации системы. Здесь вполне подходит каталог */usr/tmp/etc*, предлагаемый по умолчанию, но я обычно сохраняю резервную копию в корневой файловой системе, в каталоге */oldetc*.
8. Наконец, sysinstall спросит, откуда следует взять исходные файлы для обновления. Можно указать компакт-диск, с которого была выполнена загрузка.
9. Далее sysinstall предоставит последнюю возможность отказаться от обновления, а затем перезапишет все указанные исполняемые файлы системы. Ядро будет заменено ядром GENERIC новой версии; кроме того, будут заменены многие файлы в */etc*.
10. Перезагрузите систему.
11. После загрузки внимательно просмотрите */etc* и убедитесь, что важнейшие системные файлы соответствуют предъявляемым к ним требованиям. Хотя файлы с паролями и списками групп, а также */etc/fstab* останутся невредимыми, стоит проверить */etc/rc.conf* и любые другие измененные вами файлы.
12. Перезагрузите систему еще раз. На этом обновление базовой системы можно считать завершенным. Обновлять все дополнительные программные компоненты сторонних производителей следует отдельно.

Обновление из исходного кода

Еще один способ обновления системы – сборка программ из исходного кода. FreeBSD – это самодостаточная система, в том смысле, что в ней есть все инструменты, необходимые для сборки FreeBSD. Если нужно собрать последнюю версию FreeBSD из исходного кода, то для начала необходимо получить его последнюю версию.

Модификации FreeBSD, выпускаемые различными разработчиками, становятся доступными на серверах FreeBSD по всему миру в течение часа (точнее, 66 минут). Основной (master) сервер FreeBSD, предназначенный для хранения исходного кода, отслеживает все изменения в этом коде и авторов этих изменений. Разработчики могут «зарегистрировать» (check in) новый исходный код, а пользователи – «захватить» (check out) последние версии с помощью *системы параллельных версий* (*Concurrent Versions System, CVS*). Система параллельных версий – подходящий инструмент для управления исходным кодом, но совершенно непригодный для его распространения. Система CVS предъявляет большие требования к системным ресурсам и полосе пропускания и, похоже, стремится вывести из строя жесткий диск сервера. Поскольку все ресурсы Проекта FreeBSD кем-то дарятся, их следует расходовать как можно бережнее. Так, в Проекте FreeBSD для распространения исходного кода вместо CVS применяется собственный протокол доступа к хранилищу CVS. Он намного быстрее, эффективнее, «легче» для серверов и, как правило, удобнее для поддержки миллионов пользователей, разбросанных по всему миру. Информация с основного CVS-хранилища исходного кода дублируется на серверах CVSup по всему миру, а пользователи подключаются к серверам и загружают исходный код с помощью программы *csup*.

CSUP, CVSUP, CVS и SUP?

В документации и на веб-сайтах по FreeBSD часто упоминается инструмент CVSup, применяемый для обновления исходного кода. Нужды среднего пользователя вместо CVSup обслуживает программа csup, поэтому вы можете просто игнорировать любые ссылки на программы CVSup или cvsup(1). Однако во многих документах, имеющих отношение к обновлениям, по-прежнему упоминают CVSup. Например, серверы зеркал называются *серверами cvsup*, примеры конфигурационных файлов располагаются в каталоге */usr/share/example/cvsup* и т. д. Название CVSup являет собой комбинацию из названий CVS и sup (Software Update Protocol – протокол обновления программного обеспечения). Иногда вам будут встречаться названия этих инструментов, но знать их необязательно.

Программа csup подключается к зеркалу FreeBSD CVSup, сравнивает свою локальную копию исходного кода с версией, доступной на сервере, и копирует все изменения на локальный жесткий диск. Звучит сложно, но на самом деле процесс очень прост.

Начнем с того, что выясним – установлен ли исходный код в системе, в каталоге */usr/src*. Содержимое этого каталога должно быть примерно следующим:

```
# ls /usr/src
COPYRIGHT          UPDATING        include          sbin
LOCKS              bin             kerberos5        secure
MAINTAINERS        contrib         lib              share
Makefile           crypto          libexec          sys
Makefile.inc1      etc             ports-supfile    tools
ObsoleteFiles.inc  games           release          usr.bin
README             gnu             rescue           usr.sbin
```

Это корневой каталог дерева исходного кода FreeBSD, который содержит весь исходный код, необходимый для сборки программ и ядра FreeBSD. Исходный код подробно рассматривается в главе 11. При желании можно просмотреть эти каталоги и узнать, что представляет собой исходный код.

Если этот каталог пуст, значит, исходный код не установлен. Не беда. Исходный код можно инсталлировать с установочного компакт-диска. Для этого надо выполнить следующие команды, войдя в систему с правами root:

```
# mount /dev/acd0c /cdrom
# cd /cdrom/src
# ./install.sh all
```

Если установочного компакт-диска нет, исходный код можно взять с FTP-сервера FreeBSD.

Какой бы метод установки исходного кода вы ни избрали, начните с исходного кода для уже установленной версии FreeBSD. Например, компакт-диски FreeBSD 7.0 содержат исходный код для FreeBSD 7.0. Этот исходный код может пригодиться при администрировании системы, но он непригоден для выполнения обновлений – если пересобрать исходный код из FreeBSD 7.0, вы просто переустановите FreeBSD 7.0. Программа csup сравнит исходный код на диске и исходный код, доступный в Интернете, а затем загрузит код, отражающий изменения между версиями. Далее csup «приложит» эти *отличия* (*diffs*) к исходному коду на диске и приведет его в соответствие с исходным кодом нужной версии. Это намного эффективнее, чем загружать все 450 Мбайт дерева исходного кода! Если между обновлениями вышел один или два «выпуска», csup загрузит только один или два мегабайта, необходимых для изменения исходного кода на диске.

Применяя csup для обновления дерева исходного кода, надо указать, что обновлять, из какого источника и каким образом.

Выбор supfile

Программа csup использует конфигурационный файл, или *supfile*, определяющий параметры обновления локального хранилища исходного кода. Примеры supfiles для обновления до различных версий можно найти в каталоге */usr/share/examples/cvsup*. После выбора или создания своего supfile, отвечающего вашим требованиям, можно постоянно использовать этот файл. Последняя версия каталога */usr/share/examples/cvsup* содержит следующие примеры:

cvs-supfile

Позволяет загружать все хранилище исходного кода FreeBSD. Большинство пользователей в этом не нуждается, но разработчики FreeBSD находят такую возможность удобной. Этот файл необходим лишь тем, кто подготавливает собственные «выпуски» системы, включая образы дискет, компакт-дисков и т. д. Чтобы загрузить все CVS-хранилище, вместо csup следует использовать инструмент CVSup.

doc-supfile

Позволяет извлечь всю документацию FreeBSD на всех доступных языках (включая FAQ, Справочник и сопутствующие статьи).

gnats-supfile

Предназначен для тех, кто хочет иметь локальную копию базы данных с сообщениями о неполадках во FreeBSD (Problem Report (PR) database). Большинству пользователей она не нужна.

ports-supfile

Позволяет обновить дерево «портов» до последней версии (глава 11).

stable-supfile

Позволяет обновить исходный код до последней версии -stable.

standard-supfile

Позволяет получить исходный код для последней используемой версии FreeBSD. Если была установлена моментальная копия FreeBSD 7-stable, с помощью этого файла будет выполнено обновление исходного кода до последней версии FreeBSD 7-stable. Если была установлена версия -current, обновление будет выполнено до последней версии -current. Если была установлена ветка с исправленными ошибками FreeBSD 7.1, будет выполнено обновление до последней ветки -errata этой версии. (Ранее файл *standard-supfile* применялся исключительно для обновления версии -current.)

www-supfile

Предназначен для загрузки последней версии веб-сайта FreeBSD.

Различные компоненты, которые можно обновить с помощью csup, называются коллекциями (collections). Например, есть коллекция исходного кода, коллекция документации (*doc-supfile*), коллекция «пор-

тов» (*port-supfile*) и т. д. Многие коллекции разбиты на подколлекции: дерево исходного кода включает в себя подколлекции для таких компонентов, как программы userland, компиляторы, ядро и т. д. При обновлении FreeBSD главный интерес представляет коллекция исходного кода.

Модификация supfile

Выбранный supfile нужно настроить в соответствии с предъявляемыми требованиями. Прежде всего, скопируйте образец supfile в */etc* и откройте его в удобном редакторе. Строки, начинающиеся со знака решетки (#), представляют собой комментарии. Во всех образцах supfiles комментариев больше, чем настоящих конфигурационных записей. Большинство supfiles содержат по крайней мере шесть записей, например:

```
❶ *default host=CHANGE_THIS.freebsd.org
❷ *default base=/var/db
❸ *default prefix=/usr
❹ *default release=cvs ❺tag=RELENG_6
❻ *default delete use-rel-suffix
❼ *default compress
❽ src-all
```

Прежде всего нужно выбрать зеркало с исходным кодом, или *сервер CVSup*. Полный список серверов можно найти на веб-сайте FreeBSD, в общем случае имена серверов имеют формат *cvsup<номер>.<код_страны>.freebsd.org*. Например, можно найти сервер *cvsup15.us.freebsd.org* (пятнадцатый сервер CVS на территории Соединенных Штатов), *cvsup2.si.freebsd.org* (второй сервер на территории Словении) и т. д. Выполните ping до каждого сервера, чтобы найти сервер с наименьшим временем отклика для вашего сегмента Интернета, или воспользуйтесь программой */usr/ports/sysutils/fastest_cvsup* для автоматического определения самого быстрого сервера CVSup. Его имя нужно вписать в строку `default host` (хост по умолчанию), заменив `CHANGE_THIS.freebsd.org` ❶.

Строка `default base` (база данных по умолчанию) ❷ определяет каталог, где будут сохраняться файлы состояния, включая список обновленных файлов. Это ускорит последующие обновления. По умолчанию эти файлы сохраняются в каталоге */var/db/sup*.

Строка `default prefix` (префикс по умолчанию) ❸ – это место, куда будет помещена выбранная коллекция. По умолчанию для этого служит подкаталог *src* в префиксном каталоге или, в данном случае, – */usr/src*. Для установки исходного кода в каталог, отличный от */usr/src*, необходимо изменить этот путь. Содержимым этого каталога управляет программа csup(1), и она может стереть все, что ей не принадлежит.

Строка `default release` («выпуск» по умолчанию) ❹ – определяет тип хранилища, с которым производится синхронизация, в данном случае – это CVS-хранилище. Метка `tag` ❺ – это ветка в хранилище, то есть

выбранная версия FreeBSD. В табл. 13.1 приведены примеры наиболее типичных меток.

Таблица 13.1. Типичные теги версий FreeBSD

Метка	Версия
RELENG_7	FreeBSD 7-stable
RELENG_6	FreeBSD 6-stable
RELENG_7_0	FreeBSD 7.0-release с исправлениями
.	FreeBSD 7-current

Если разработчик удалил некоторые файлы с исходным кодом из основного хранилища FreeBSD, программа csup тоже должна удалить эти файлы из вашей системы, чтобы обеспечить идентичность копий. Строка `default delete` (удалять по умолчанию) ❻ дает программе csup права на это. Точно так же параметр `use-rel-suffix` позволяет csup задействовать общий базовый каталог для нескольких версий исходного кода, не смешивая их.

Параметр `compress` (сжатие) ❼ позволяет сократить трафик ценой увеличения нагрузки на процессор. Используйте эту возможность.

В заключение небходимо сообщить программе csup, какая коллекция должна обновляться. Флаг `src-all` ❽ предписывает csup обновить все дерево исходного кода. В файле stable-supfile есть список закомментированных подколлекций, таких как `usr.bin` (содержимое каталога */usr/bin*), `contrib` (содержимое каталога */usr/src/contrib*), `sys` (ядро) и т. д. Теоретически можно обновить только один раздел дерева исходного кода, но в общем случае это далеко не лучшая идея. Установка программы */usr/bin* из FreeBSD-current в версию FreeBSD-stable чревата непредсказуемыми сбоями, а значит, не поддерживается.

В одном supfile можно указать несколько коллекций. Например, мне нужен доступ к коллекции исходного кода для последней версии FreeBSD 7-stable. Как ответственному за документацию FreeBSD мне нужна последняя коллекция документации. Наконец, мне необходимо последнее дерево «портов», чтобы можно было устанавливать самое свежее программное обеспечение. FreeBSD содержит отдельные примеры supfiles для каждой из этих коллекций, они находятся в каталоге */usr/share/examples/cvsup*. Но я не хочу запускать csup отдельно для каждой коллекции, поэтому просто перечислю их в одном файле supfile.

Полный пример supfile

Вот содержимое моего файла supfile, включая все изменения, необходимые для обновления моего исходного кода:

```
❶ *default host=cvsup15.us.freebsd.org
*default base=/var/db
*default prefix=/usr
```

```
❷ *default release=cvs tag=RELENG_7
*default delete use-rel-suffix
*default compress
src-all
❸ ports-all tag=.
❹ doc-all tag=.
```

Замечу, что я внес в файл примера очень мало изменений. Здесь я указал ближайший сервер CVSup ❶ и нужную мне версию FreeBSD ❷. Особый интерес вызывают последние две записи, в которых добавлены коллекции `ports-all` ❸ и `doc-all` ❹. Коллекции, не относящиеся к исходному коду, не имеют «выпусков» и вариантов, которые есть у коллекций исходного кода. Значит, если указать версию `RELENG_7` дерева «портов», сервер не поймет, чего от него хотят, и обновления не будут загружены. Если к названию коллекции дописать ключевое слово `tag=.`, csup загрузит последнюю версию этой коллекции.

Блокирование обновлений: файл refuse

Мне требуется полное дерево исходного кода и полное дерево документации, но меня интересуют только отдельные части дерева «портов». Если заглянуть в каталог */usr/ports*, можно увидеть подкаталоги для программного обеспечения на арабском, французском, немецком, венгерском, японском, корейском, польском, португальском, русском, украинском и вьетнамском языках. Безусловно, мне приятно, что данное программное обеспечение доступно и тем, кто говорит на этих языках, но вероятность того, что мне понадобятся эти программы, колеблется между «ничтожно мала» и «равна нулю». Чтобы предписать csup не обновлять эти каталоги, создайте файл */var/db/sup/refuse*, который выглядит примерно так:

```
ports/arabic
ports/french
ports/german
...
ports/vietnamese
```

В файле refuse не должно быть комментариев!

Файл refuse можно создать для любых компонентов FreeBSD, однако лучше не указывать в нем что-либо из */usr/src*. Если отказаться от обновления существенной системной программы, эта программа в какой-то момент будет несовместима с системой.

Система интерпретирует файл refuse на основе сопоставления подстрок. Строка `sys` блокирует обновление всех программ, содержащих слово `sys`, в том числе исходный код ядра в */usr/src/sys*. Аккуратно заполняйте строки файла refuse, чтобы заблокировать только то, что вы хотите заблокировать!

Если вы решили использовать файл refuse, постарайтесь удалить соответствующие файлы с жесткого диска. Лучше немного пострадать из-за

отсутствующих программных компонентов, чем безуспешно отыскивать, теряясь в догадках, источники проблем, связанные с существующим программным обеспечением, которое не обновлялось уже три года.

Кроме того, в применении файла refuse есть определенный риск. Обновление только части дерева исходного кода может вызвать ошибки в процессе сборки программ. Коллекция «портов» создавалась как единое целое, и отказ от обновления ее отдельных частей может вызывать проблемы. Несмотря на то что дерево документации хорошо структурировано, проблемы бывают даже здесь. Файл refuse может стать источником многих проблем, но только вы решаете – использовать его или нет.

Обновление исходного кода системы

После создания supfile зайдите в систему с учетной записью root и запустите csup, введя следующую команду:

```
# csup supfile
❶ Cannot connect to 2001:468:902:201:209:3dff:fe11:442c: Protocol not
supported
(Перевод: Невозможно соединиться с 2001:468:902:201:209:3dff:fe11:442c:
протокол не поддерживается)
Connected to 128.205.32.21
Updating collection src-all/cvs
 Edit src/Makefile
 Edit src/Makefile.inc1
...
```

Первое, что мы видим, – это неприятное предупреждение ❶ о том, что программа csup не смогла подключиться к длинной строке шестнадцатеричных чисел. Эта длинная строка – адрес протокола IPv6, который поддерживается операционной системой FreeBSD, но еще не поддерживается большинством интернет-провайдеров. После этого программа csup выводит имена всех файлов, подвергающихся изменениям, и в конце – сообщение `Finished successfully` (выполнено успешно).

Поздравляю! Теперь у вас самая свежая версия дерева исходного кода.

Получение полного дерева исходного кода с помощью csup

Программу csup можно запускать и без установленного дерева исходного кода. Программа сравнит все, что у вас есть (ничего), с тем, что требуется (все), и установит отсутствующие компоненты (полное дерево). Однако администраторы зеркал предпочитают, чтобы вы устанавливали исходный код с компакт-диска, если, конечно, он у вас имеется; они предоставляют серверы, трафик и стремятся обслужить как можно больше желающих. Полное несжатое дерево исходного кода занимает порядка 450 Мбайт дискового пространства, тогда как для обновления имеющегося дерева потребуется загрузить всего несколько мегабайт. Даже применяя сжатие, вы сильно понизите пропускную

способность. Для подобного поведения есть профессиональный термин *невежливость* (*rude*).

Сборка FreeBSD из исходного кода

После обновления исходного кода изучите файл */usr/src/UPDATING*. В этом файле в обратном хронологическом порядке приведены все предупреждения и замечания, относящиеся к изменениям в исходном коде, которые могут представлять интерес для тех, кто выполняет сборку программ из исходного кода. Здесь указаны действия, которые, возможно, придется выполнить перед сборкой системы из исходного кода, а также описаны все существенные изменения системной функциональности. Если вы хотите, чтобы система осталась работоспособной после обновления, четко следуйте инструкциям, изложенным в этом файле. Кроме того, стоит проверить новые конфигурационные файлы ядра GENERIC или NOTES на предмет новых параметров и интересных изменений в ядре.

Пообщавшись с членами сообщества FreeBSD какое-то время, можно услышать разные истории об особых методах сборки FreeBSD. Вы услышите самые невероятные доказательства превосходства какого-нибудь метода перед стандартным. Вы, конечно же, свободны в выборе того или иного метода обновления системы, но Проект FreeBSD поддерживает единственный метод, описанный в файле */usr/src/UPDATING*. Если, следуя какой-нибудь другой процедуре, вы потерпите неудачу, вас призовут к документированному методу. Процедура, описанная в этой книге, применяется начиная с версии FreeBSD-current и немного отличается от той, что использовалась в версии FreeBSD 5-current, тем не менее я настоятельно рекомендую дважды сравнить эти инструкции с теми, что приводятся в файле */usr/src/UPDATING*.

Настройка параметров сборки FreeBSD

Помните, как в главе 11 обсуждался файл */etc/make.conf*? Для сборки самой системы во FreeBSD используется отдельный файл с настройками параметров сборки. Параметры в файле */etc/make.conf* влияют на сборку любого программного обеспечения в системе, а параметры в файле */etc/src.conf* – только на сборку самой FreeBSD из исходного кода.

Сборка «мира»

В первую очередь необходимо собрать обновления в пространстве пользователя (userland):

```
# cd /usr/src
# make buildworld
```

Команда `make buildworld` прежде всего задействует исходный код, чтобы собрать инструменты, необходимые для сборки системного компилятора. После этого она инициирует сборку компилятора и сопутствующих библиотек. Наконец, с помощью новых инструментов, компилятора и библиотек будет собрано все базовое программное обеспечение FreeBSD. (Все это напоминает сборку автомобиля по инструкции, которая начинается так: «Спуститесь в шахту и накопайте железной руды».) Результаты работы `make buildworld` помещаются в */usr/obj*. Эта процедура может занять до нескольких часов, в зависимости от производительности аппаратного обеспечения. При этом можно продолжать работать и в ходе исполнения команды `make buildworld`, если ваше аппаратное обеспечение функционирует достаточно устойчиво; потребляя системные ресурсы, эта команда не требует внимания.

Убедитесь, что команда `make buildworld` выполнилась без ошибок! Если она завершилась с кучей сообщений вроде тех, что появляются при неудачном компилировании ядра, то процесс обновления следует остановить. В главе 2 рассказано, как получить помощь. Никогда не пытайтесь установить поврежденные или неполные обновления.

Сборка, установка и тестирование ядра

Лучший способ протестировать обновления – это собрать новое ядро GENERIC. Это позволит отгородиться от проблем, связанных с нестандартным ядром, и оценить возможные проблемы, касающиеся системы FreeBSD в целом. Самые активные из вас, безусловно, могут обновить свое ядро с нестандартной конфигурацией, но если что-то пойдет не так, нужно попробовать собрать ядро GENERIC. Не забудьте сравнить конфигурацию своего ядра с конфигурацией ядра GENERIC, чтобы охватить все изменения в вашей нестандартной конфигурации.

По умолчанию в процессе обновления собирается ядро GENERIC. Если вам нужно обновить нестандартное ядро, определите имя ядра с помощью переменной `KERNCONF`. Задать значение этой переменной окружения можно из командной строки, в файле */etc/make.conf* или в файле */etc/src.conf*.

Собрать ядро можно одним из двух способов. Команда `make buildkernel` соберет новое ядро, но не установит его. В этом случае установку ядра можно выполнить следующей за `make buildkernel` командой `make installkernel`. Команда `make kernel` выполнит обе эти команды подряд. Используйте способ, который больше соответствует вашим планам. Например, если мне нужно обновить систему во время воскресной профилактики, я могу выполнить команды `make buildworld` и `make buildkernel` в течение предшествующей недели, чтобы сэкономить несколько часов моего драгоценного выходного дня. Однако до назначенного дня профилактических работ я не хочу устанавливать новое ядро – если в пятницу обнаружатся какие-либо проблемы и потребуется перезагрузка, желательно загрузить старое, рабочее ядро, а не новое. В воскресенье

утром, когда я буду готов к выполнению обновления, я запущу команду `make installkernel`. С другой стороны, при обновлении системы у себя на ноутбуке я предпочитаю запускать команду `make kernel`. Итак, чтобы обновить свое нестандартное ядро, я должен выполнить команду:

```
# make KERNCONF=HUMVEE kernel
```

Повторяю: не нужно пытаться установить ядро, если его компиляция не была успешной. Если работа команды `make buildkernel` завершилась сообщением об ошибке, вам следует устранить источник проблемы и только потом продолжить сборку.

Установив новое ядро, выполните загрузку системы в однопользовательском режиме. Система должна перезапуститься без сбоев. Однако пользовательские программы могут работать не так, как ожидалось; многие из них зависят от интерфейсов, реализуемых ядром, которые могут измениться в результате обновления. Обычно об этом явно упоминается в файле */usr/src/UPDATING*. Если с новым ядром система работает без сбоев, можно продолжать. В противном случае подробно опишите возникшую проблему и загрузите старое ядро, чтобы восстановить работу служб на то время, пока вы будете заниматься решением проблемы.

Оптимизация за счет многопоточной сборки

Для повышения скорости сборки опытные системные администраторы наверняка используют ключ `-j` утилиты make. Этот ключ позволяет запустить несколько процессов сборки и воспользоваться преимуществами многопроцессорной системы. Если в вашей системе несколько процессоров или в процессоре несколько ядер, ключ `-j` поможет ускорить сборку FreeBSD. Наиболее разумно – задать число процессов сборки, на единицу большее числа имеющихся процессоров. Например, если у вас четырехъядерный процессор, есть смысл использовать пять процессов сборки, введя команду `make -j5 buildworld && make -j5 kernel`.

Официально Проект FreeBSD не поддерживает выполнение обновлений с применением ключа `-j`, даже несмотря на то, что многие разработчики используют его. Если процесс сборки терпит неудачу при запуске с ключом `-j`, то прежде чем жаловаться, попробуйте запустить сборку без этого ключа.

Подготовка к установке нового «мира»

Осторожно! Из этой точки нет возврата! Здесь легко можно решить проблемы, связанные с новым ядром, – достаточно лишь загрузить старое, хорошо зарекомендовавшее себя ядро, но как только будет установлен свежесобранный «мир», вернуться назад можно будет только с помощью резервной копии. Прежде чем двинуться дальше, проверьте наличие резервной копии или хотя бы признайте, что прямо сейчас вам предстоит сделать первый шаг туда, откуда не вернуться.

Если новое ядро не вызывает проблем, можно приступить к установке свежесобранной системы. Во-первых, убедитесь, что система может установить новые двоичные файлы. Каждая новая версия FreeBSD предполагает, что предыдущая версия поддерживает все учетные записи и группы, необходимые для новой версии. Если владельцем некоторой программы должен быть определенный пользователь и его учетная запись отсутствует в системе, то процесс обновления завершится ошибкой. Тут на сцену выходит программа mergemaster(8).

mergemaster сравнивает файлы, находящиеся в каталоге */etc*, с новыми файлами */usr/src/etc*, выявляет различия между ними и либо устанавливает новые файлы, либо откладывает их для оценки различий или даже предлагает объединить отличающиеся конфигурационные файлы в один. Это очень удобно сделать при выполнении обновлений. Вы запускаете программу mergemaster первый раз перед установкой нового «мира», чтобы убедиться, что система в состоянии установить новые двоичные файлы, а затем второй раз – после установки, чтобы синхронизировать содержимое каталога */etc* с остальной системой.

Начнем с запуска программы mergemaster(8) в режиме предварительной оценки – с ключом `-p`. В этом режиме, в частности, программа сравнит */etc/master.passwd* и */etc/group* и выявит все учетные записи и группы, которые должны присутствовать в системе, чтобы команда `installworld` выполнилась успешно.

```
# mergemaster -p
❶ *** Unable to find mtree database. Skipping auto-upgrade.
(Перевод: Невозможно отыскать базу данных mtree. Пропуск функции auto-upgrade.)

❷ *** Creating the temporary root environment in /var/tmp/temproot
*** /var/tmp/temproot ready for use
*** Creating and populating directory structure in /var/tmp/temproot
(Перевод: Создание временного корневого каталога в /var/tmp/temproot
/var/tmp/temproot готов к использованию
Создание и заполнение структуры каталогов в /var/tmp/temproot)
```

В этих начальных сообщениях с тремя символами «звездочки» mergemaster описывает свои действия. Первое сообщение гласит, что mergemaster не смогла обнаружить базу данных для выполнения автоматического обновления ❶.

Многие обновления программа mergemaster может выполнить автоматически, как это будет показано ниже в этой главе. mergemaster устанавливает временный каталог */etc* в */var/tmp/temproot* ❷, где сохраняются новые конфигурационные файлы, которые затем будут сравниваться с существующими. Далее следует первое сравнение.

```
*** Displaying differences between ❶./etc/master.passwd and installed version:
(Перевод: Различия между ./etc/master.passwd и существующей версией)
❷ --- /etc/master.passwd Fri Nov 3 11:52:21 2006
❸ +++ ./etc/master.passwd Mon Jun 6 16:19:56 2005
@@ -1,6 +1,6 @@
```

```
❹ -# $FreeBSD: src/etc/master.passwd,v 1.39 2005/06/06 20:19:56 brooks Exp $
❺ +# $FreeBSD: src/etc/master.passwd,v 1.40 2005/06/06 20:19:56 brooks Exp $
#
❻ -root:$1$GtDsdFlU$F5mTAagzalt7dHImUsNSL1:0:0::0:0:Laptop Admin:/root:/
bin/csh
❼ +root::0:0::0:0:Charlie &:/root:/bin/csh
```

Очень важный момент: программа mergemaster первым упоминает имя нового сравниваемого файла ❶. Затем следуют имена двух различных версий сравниваемых файлов, причем сначала приводится имя существующего файла ❷, а затем – имя обновленной версии файла ❸. Проницательный читатель может обратить внимание на странные даты файлов – если работает старая версия FreeBSD, тогда почему файл паролей в ней более новый? Эти даты соответствуют датам последнего изменения файлов. Мой файл паролей последний раз изменялся 3 ноября 2006 года, тогда как версия файла в */usr/src* не изменялась с 6 июня 2005 года. Просто пользователь моей системы вдруг обновил свой пароль, вот и все![1] Обратите внимание на плюсы и минусы в начале строк. Минус означает, что эта строка находится в имеющемся файле, а плюс – в версии файла из каталога */usr/src*.

Это наглядно иллюстрируют следующие две строки, которые выводит программа mergemaster. Первая строка ❻ помечена минусом – это текущая учетная запись пользователя root. Вторая строка ❼ – это учетная запись в обновленном файле. Вероятно, следует сохранить текущую запись – было бы нежелательно «обновить» ее, очистив пароль!

Немного ниже видим, что картина изменилась:

```
_pflogd:*:64:64::0:0:pflogd privsep user:/var/empty:/usr/sbin/nologin
❶ +_dhcp:*:65:65::0:0:dhcp programs:/var/empty:/usr/sbin/nologin
uucp:*:66:66::0:0:UUCP pseudo-user
```

Строка с учетной записью пользователя _dhcp ❶ начинается с плюса, и для нее нет соответствующей строки, начинающейся с минуса. В настоящее время в системе отсутствует пользователь _dhcp. Если в конфигурации по умолчанию появляется новый пользователь, это означает, что данный пользователь – владелец каких-либо программ или файлов в новой системе. Если не добавить такого пользователя, установка потерпит неудачу.

Еще чуть ниже пара строк:

```
nobody:*:65534:65534::0:0:Unprivileged user:/nonexistent:/usr/sbin/nologin
❶ -mwlucas:$1$zxU7ddkN$9GUEEVJHOr.owyAwUONFX1:1001:1001::0:0:Michael W
Lucas:/
home/mwlucas:/bin/tcsh
```

[1] То есть эта система уже давно не обновлялась? Нет, я нарочно изменил дату последнего редактирования файла */etc/master.passwd* для пущей наглядности примера. Не копайте эти примеры слишком глубоко, чтобы не волноваться попусту.

Строка, начинающаяся с минуса ❶, – это моя учетная запись. Не удивительно, что она отсутствует в базовом дистрибутиве FreeBSD. Безусловно, мне следует сохранить свою учетную запись, чтобы пережить процедуру обновления.

```
Use 'd' to delete the temporary ./etc/master.passwd
Use 'i' to install the temporary ./etc/master.passwd
Use 'm' to merge the temporary and installed versions
Use 'v' to view the diff results again

Default is to leave the temporary file to deal with by hand

How should I deal with this? [Leave it for later] m

(Перевод:'d' - удалить временный файл ./etc/master.passwd
'i' - установить временный файл ./etc/master.passwd
'm' - объединить временный файл и существующую версию
'v' - просмотреть различия еще раз

По умолчанию временный файл будет оставлен для обработки вручную

Что делать с новой версией? [Оставить на будущее])
```

Решения, решения... Удалить ли временный (новый) файл *master.passwd*? Исключено – нам необходима учетная запись пользователя _dhcp, чтобы установить обновленную версию FreeBSD. Установить новую версию *master.passwd*? Но тогда будут уничтожены существующие учетные записи. Если у вас имеются какие-нибудь сомнения, можно просмотреть различия еще раз. Единственный способ двинуться дальше, не уничтожив прежнюю конфигурацию и не нарушив функциональность новой системы, – это объединить два файла. Введите `m`.

Программа mergemaster разделит окно пополам с помощью утилиты sdiff(1). В левой половине выводится начало существующего файла, в правой – новой версии файла. На экране отображаются только отличающиеся участки. Далее введите `r` (right – справа) и `l` (left – слева), чтобы выбрать запись слева или справа, которая должна попасть в наш новый файл *master.passwd*.

```
# $FreeBSD: src/etc/master.passwd,v 1.39 2005 | # $FreeBSD: src/etc/
master.passwd,v 1.40 2005
```

Слева – версия 1.39 файла, справа – версия 1.40. Программа mergemaster использует эти номера версий (как и многие другие инструменты) для определения необходимости выполнять обновление, поэтому наш новый файл должен иметь обновленный номер версии. Введем `r`, чтобы использовать запись справа, после чего mergemaster выведет следующее отличие.

```
root:$1$GtDsdFlU$F5mTAagzalt7dHImUsNSL1:0:0:: | root::0:0::0:0:Charlie &:/
root:/bin/csh
```

Слева – существующая учетная запись пользователя root с измененным паролем, справа – учетная запись этого же пользователя с паро-

лем по умолчанию. Нам следует взять запись из существующего файла паролей, поэтому вводим `l`.

```
> _dhcp:*:65:65::0:0:dhcp programs:/var/empty:/
```

Это новая учетная запись, для нее нет соответствующей старой записи. Нам следует создать пользователя _dhcp, чтобы команда `installworld` выполнилась успешно, поэтому нажимаем клавишу `r`.

```
mwlucas:$1$zxU7ddkN$9GUEEVJHOr.owyAwU0NFX1:10 <
```

А это моя учетная запись. Мне хотелось бы сохранить возможность входить в систему под своей учетной записью, поэтому лучше я введу `l`.

После обхода всех различий mergemaster представит следующие варианты выбора:

```
Use 'i' to install merged file
Use 'r' to re-do the merge
Use 'v' to view the merged file
Default is to leave the temporary file to deal with by hand
*** How should I deal with the merged file? [Leave it for later]
(Перевод: 'i' - установить объединенный файл
'r' - повторить процедуру объединения
'v' - просмотреть объединенный файл
По умолчанию временный файл будет оставлен для обработки вручную
Что делать с объединенным файлом? [Оставить на будущее])
```

Никогда не помешает просмотреть содержимое объединенного файла, если вы не уверены в том, что не придется ничего исправлять. Просмотрите объединенный файл, введя `v`, и если все в порядке, установите его, введя `i`.

```
*** Merged version of ./etc/master.passwd installed successfully
*** Temp ./etc/group and installed have the same CVS Id, deleting
*** Comparison complete
Do you wish to delete what is left of /var/tmp/temproot? [no] y
(Перевод: *** Объединенная версия файла ./etc/master.passwd успешно установлена
*** Временный файл ./etc/group имеет то же самое значение Id в CVS, удален
*** Сравнение завершено
Желаете удалить то, что осталось в /var/tmp/temproot? [нет])
```

Закончив необходимые минимальные проверки, программа mergemaster выведет заключительное сообщение и предложит удалить временные файлы. К данному моменту все действия с временным корневым разделом закончены, поэтому его можно удалить. Кроме того, mergemaster предложит выполнить некоторые дополнительные действия, как результат выполненных изменений. Например, после создания нового файла паролей следует обновить базу данных паролей – программа mergemaster сможет сделать это, если вы ей позволите.

В заключение mergemaster сравнит содержимое */etc/make.conf* с файлом *make.conf* из */usr/src*. Сведения об изменившихся или удаленных параметрах будут выведены на экран для изучения.

Установка «мира»

Оставаясь в однопользовательском режиме, можно выполнить установку обновленной системы FreeBSD, выполнив команду `make installworld`. После этого по экрану побегут строки, большая часть из которых будет содержать слово *install*.

```
# cd /usr/src
# make installworld
```

Некоторые программы и файлы могут оказаться ненужными в системе. Чтобы увидеть эти устаревшие файлы, запустите команду `make check-old`.

```
# make check-old
>>> Checking for old files
/usr/sbin/pccardc
/usr/share/man/man8/pccardc.8.gz
...
```

В этом списке будут перечислены все компоненты системы, некогда установленные и ставшие ненужными. Оцените потребность в них и либо сохраните существующие, но не поддерживаемые программы, либо найдите альтернативы. Немного дальше в выводе вы увидите разделяемые библиотеки, также ставшие нужными:

```
>>> Checking for old libraries
/lib/libcrypto.so.4
/usr/lib/libssl.so.4
/usr/lib/libroken.so.8
/lib/libatm.so.3
/lib/libc.so.6
...
```

В заключение вашему вниманию будет представлен список ставших ненужными каталогов. Удалять каталоги приходится очень редко – я могу вспомнить только семь таких случаев с момента появления команды `make check-old`.

Если вы не используете какие-либо из устаревших программ и каталогов, удалите их командой `make delete-old`. Перед удалением каждого файла утилита make(1) запросит подтверждение.

Устаревшие разделяемые библиотеки

Устаревшие разделяемые библиотеки требуют особого внимания. Этими библиотеками пользуются многие программы сторонних производителей. Если удалить такую библиотеку, программа просто не запустится. Крайне нежелательно, например, удалить библиотеку, если ее использует приложение, предназначенное для решения чрезвычайно важных задач. Единственный способ восстановить работоспособность такой программы – повторно скомпилировать ее или заменить библиотеку. Разделяемые библиотеки рассматривались в главе 12. Если биб-

лиотека не требуется для работы ни одной программе, ее можно удалить. Однако проверять все программы, чтобы выяснить, не используют ли они ту или иную библиотеку, весьма нелегко.

Проверим, для примера, список устаревших библиотек, приведенный выше. Одна из них – libc.so.6. Заглянув в каталог */lib*, я увидел, что там присутствует библиотека libc.so.7. Скорее всего, мне не требуются сразу две библиотеки libc.so.6 и libc.so.7. Однако от библиотеки libc зависит слишком многое – это как главная городская магистраль. Если я удалю ее, может получиться так, что программы, установленные из «портов», просто перестанут работать. Присутствие этой устаревшей библиотеки в ближайшее время никак не повредит системе – оставив устаревшие библиотеки в дополнение к новым, вы обеспечите работоспособность системы и сможете не спеша выполнить переустановку дополнительного программного обеспечения. Порядок обновления «портов» мы рассмотрим ниже в этой главе.

В любом случае, если вы считаете, что ни одна из перечисленных библиотек не представляет никакого интереса, их можно удалить, но я настоятельно рекомендую перед удалением сохранить резервные копии всех библиотек. Для этого достаточно скопировать их в каталог *old-libs* где-нибудь в файловой системе. Обнаружив, что какое-то важное приложение не запускается, вы легко сможете восстановить его работоспособность. Чтобы работоспособность программ сохранилась, но при этом сами библиотеки хранились отдельно, можно записать устаревшие библиотеки в каталог */usr/lib/compat*. Другой вариант состоит в том, чтобы использовать утилиту libchk (*/usr/ports/sysutils/libchk*) для идентификации программ, использующих ту или иную библиотеку.

И снова mergemaster

Мы почти у цели! Мы уже обновили информацию об учетных записях и группах в каталоге */etc*, теперь нужно обновить остальные файлы. У программы mergemaster много специализированных функций, все они описаны на страницах руководства. Я могу порекомендовать две из них, наиболее полезные с моей точки зрения.

После добавления файла в базовую установку FreeBSD нет смысла сравнивать его с чем бы то ни было. С ключом `-i` программа mergemaster автоматически установит файлы в каталог */etc*. Завершив работу, mergemaster выведет список установленных файлов.

Есть еще один набор файлов, которые меня совсем не интересуют и которые я никогда не редактирую. Например, в операционной системе FreeBSD присутствуют десятки сценариев запуска, расположенные в каталоге */etc/rc.d*. Поскольку я не изменяю эти файлы, то могу просто установить более новые версии этих сценариев. Ключ `-U` предписывает программе mergemaster автоматически обновлять любые системные файлы, которые я не редактировал.

```
# mergemaster -iU
```

mergemaster просмотрит все файлы в */etc* и проверит их принадлежность к базовому дистрибутиву FreeBSD. Принцип действия программы mergemaster в этом случае точно такой же, как и в случае с подготовкой к установке, поэтому здесь мы не будем углубляться в подробности. Выполните перезагрузку – и вы получите обновленную систему! Теперь можно побеспокоиться об обновлении портов.

Наконец, ключ `-P` предписывает программе mergemaster сохранять замещаемые файлы, при этом нужно указать каталог для сохранения копий в файле */etc/mergemaster.rc*. Вообще я предпочитаю заранее создавать резервную копию всей системы, тем не менее для сохранения замещаемых файлов в локальной файловой системе можно использовать ключ `-P`, что позволит вернуть файлы на место быстрее, чем восстановление из резервной копии.

Определите нужные значения параметров в файле *.mergemasterrs* из своего домашнего каталога или в файле */etc/mergemaster.rc*. Некоторые очень удобные параметры из тех, что находятся в конфигурационном файле, невозможно задать в командной строке, например, я всегда изменяю содержимое файла */etc/motd*, поэтому нахожу параметр `IGNORE_MOTD=yes` весьма полезным.

Обновление в однопользовательском режиме

Согласно инструкциям, обновление некоторых компонентов системы следует выполнять в однопользовательском режиме. Многие считают это неприятностью, а кое-кто даже серьезным препятствием. Программы FreeBSD – это всего лишь файлы на диске, не так ли? Здравый смысл подсказывает, что достаточно просто скопировать их на диск, перезагрузить систему и все будет в порядке.

Это еще один пример ситуации, когда здравый смысл может нарушить ваши планы на месяц. В редких случаях разработчики FreeBSD вносят в систему некоторые низкоуровневые изменения, требующие установки в однопользовательском режиме. Могут возникать конфликтные ситуации, если особо важные программы не смогут работать, если были установлены в многопользовательском режиме. Если в такой редкой ситуации окажется файл */bin/sh*, вы попадете в мир кошмаров. В этом случае у вас останется единственный способ восстановить систему: вынуть жесткий диск из сервера, установить его на другую машину, скопировать с него все нужные данные, отформатировать диск и вновь установить на него систему. Или можно загрузиться с компакт-диска восстановления системы и молиться, чтобы ваших навыков хватило для решения этой задачи.[1]

Обновление в многопользовательском режиме может породить и другие проблемы, такие как трудноуловимая гонка за ресурсами, пробле-

[1] Это глас опыта. Не поступайте так в действительности.

мы с именованием и масса других неприятностей. Конечно, вы можете выполнить обновление в многопользовательском режиме, но тогда не жалуйтесь, если в системе обнаружатся какие-нибудь проблемы.

В многопользовательском режиме можно выполнить сборку новой системы. В многопользовательском режиме можно даже собрать и установить новое ядро. Но если речь заходит об установке приложений и библиотек, вы *обязаны* находиться в однопользовательском режиме, причем после запуска системы на обновленном ядре.

NFS и обновление

Вам нужно обновить много компьютеров? Вспомните о сетевой файловой системе (NFS), которая рассматривалась в главе 8. Соберите новую систему и все ваши ядра на центральном, самом быстром сервере, а затем экспортируйте каталоги */usr/src* и */usr/obj* этой системы всем остальным клиентам. Использование этих томов NFS для работы команд `make installkernel` и `make installworld` позволит сэкономить время при сборке программных компонентов на других машинах и гарантирует, что во всех системах FreeBSD будут установлены одни и те же двоичные файлы.

Уменьшение размера FreeBSD

Зачем нужен весь этот исходный код, если не для сборки своей версии операционной системы? FreeBSD предоставляет вам не только исходный код, но и целую серию параметров, позволяющих легко влиять на процесс сборки.

Эти параметры могут быть определены в */etc/make.conf* (глава 10) или в */etc/src.conf*. Настройки в */etc/src.conf* влияют только на сборку системного программного обеспечения FreeBSD из исходного кода, параметры в */etc/make.conf* – на сборку любого программного обеспечения. Полный список параметров для файла *src.conf* есть на странице руководства src.conf(5), а в табл. 13.2 я привел те из них, которые считаю наиболее полезными в определенных ситуациях. В главе 20 мы познакомимся с этими параметрами поближе.

Система сборки проверяет наличие определений всех этих переменных. Определение `WITHOUT_TOOLCHAIN=NO` сделает переменную неопределенной, но в любом другом случае она будет считаться определенной. (Да-да, вполне допустимо даже определение `WITHOUT_SENDMAIL=postfix`.)

Если вас интересует точное назначение этих параметров, ознакомьтесь с обзором параметров сборки Пауля-Хеннинга Кампа (Poul-Henning Kamp) по адресу *http://phk.freebsd.dk/misc/build_options*. Там вы увидите, какие параметры отвечают за удаление файлов, какие – за их из-

менение и какие препятствуют установке файлов в систему. Кроме того, прежде чем использовать эти параметры на рабочем сервере, рекомендую опробовать их на тестовой системе, поскольку массовое удаление компонентов может привести к непредсказуемым последствиям.

Большинство параметров, начинающихся с `WITHOUT_`, включает компоненты системы в список файлов, которые будут удаляться командой `make delete-old`. Если, к примеру, вы решите, что в системе не нужен сервер Sendmail, то при обновлении Sendmail не только не будет собираться, но и будет предложен для удаления из системы. Если вы не предполагаете сборку каких-либо компонентов системы, лучше совсем удалить их.

Таблица 13.2. Параметры системы сборки

Параметр	Назначение
`WITHOUT_BIND`	Система не будет выполнять сборку каких-либо компонентов, составляющих BIND, включая named, dig, nslookup и связанные с ними библиотеки
`WITHOUT_CVS`	Система не будет выполнять сборку CVS
`WITHOUT_CXX`	Не будет собираться компилятор C++
`WITHOUT_DICT`	Не будет собираться словарь
`WITHOUT_EXAMPLES`	Не будут устанавливаться примеры
`WITHOUT_GAMES`	В этой системе вам не поразвлечься!
`WITHOUT_GDB`	Не будет собираться отладчик
`WITHOUT_HTML`	Не будет собираться документация в формате HTML
`WITHOUT_INET6`	Будет отключена поддержка Ipv6
`WITHOUT_INFO`	Не будет собираться или устанавливаться документация в формате info(5)
`WITHOUT_IPFILTER`	Не будет собираться IP Filter
`WITHOUT_IPX`	Ни одна программа не будет поддерживать протокол IPX
`WITHOUT_KERBEROS`	Не будет ни собираться, ни устанавливаться, ни поддерживаться протокол Kerberos
`WITHOUT_LIBPTHREAD`	Не будет собираться libpthread (глава 12)
`WITHOUT_LIBTHR`	Не будет собираться libthr (глава 12)
`WITHOUT_LPR`	Не будет собираться система печати
`WITHOUT_MAN`	Не будут собираться или устанавливаться страницы руководства
`WITHOUT_NIS`	Не будет собираться или поддерживаться NIS(8)
`WITHOUT_OBJC`	Не будет поддерживаться Objective C
`WITHOUT_RCMDS`	Не будут собираться и устанавливаться rlogin, rcp, rwho и другие r-программы
`WITHOUT_SENDMAIL`	Не будет собираться Sendmail

Параметр	Назначение
WITHOUT_SHAREDOCS	Не будет устанавливаться документация для старых версий
WITHOUT_TCSH	Дайте, угадаю: вы из этих, сдвинутых на */bin/bash*?
WITHOUT_TOOLCHAIN	Не будут устанавливаться такие компоненты, как компиляторы, отладчики и прочее. Этот параметр полезен при создании встраиваемых систем. Если он вам понадобится, укажите его в командной строке на этапе `make installworld`, потому что при его наличии команда `make buildworld` завершится с ошибкой.

Обновление с помощью csup и make

Один из популярных методов обновления заключается в использовании программы csup в комплексе с дополнительной целью для утилиты make(1). Сам я не пользуюсь этим методом, но многие предпочитают именно его. При работе с командой csup вместо создания собственных файлов supfile можно задействовать готовые образцы и утилиту make(1). Прежде всего, надо установить несколько переменных в */etc/src.conf* или в */etc/make.conf*:

```
SUP_UPDATE=yes
```

Эта строка сообщает утилите make(1), что она будет выполнять обновление программного обеспечения.

```
SUP=/usr/bin/csup
```

Параметр `SUP` – это полный путь к программе csup (или CVSup, если вы по-прежнему используете эту программу).

```
SUPHOST=cvsup9.us.freebsd.org
```

В параметре `SUPHOST` следует указать одно из ближайших зеркал FreeBSD CVSup.

```
SUPFILE=/usr/share/examples/cvsup/stable-supfile
```

Параметр `SUPFILE` определяет имя конфигурационного файла для программы csup.

```
PORTSSUPFILE=/usr/share/examples/cvsup/ports-supfile
```

Параметр `PORTSSUPFILE` указывает, какой supfile следует использовать для обновления «портов». Это значение не следует определять, если вы не собираетесь обновлять коллекцию «портов» или используете portsnap(8), как описано ниже в этой главе.

```
DOCSUPFILE=/usr/share/examples/cvsup/doc-supfile
```

Название параметра позволяет предположить, что он определяет supfile для системной документации. Но это не так. Параметр `DOCSUPFILE` – это supfile для исходного кода коллекции документов, то есть документов

в формате SGML, из которых будут созданы Справочник (Handbook), Сборник вопросов и ответов (FAQ), а также ряд других книг и статей.

Задав эти значения, можно приступать к обновлению:

```
# make update && make buildworld && make buildkernel
```

Действительно ли такая команда проще, чем `csup /etc/supfile && make buildworld && make buildkernel`? Многие думают, что да, потому что она позволяет сэкономить время и силы на создании собственных файлов supfile. Но вы все же должны понимать, как создаются и что содержат файлы supfile. Если вам понравился этот метод, применяйте его.

Кросс-компиляция FreeBSD

Кросс-компиляция (cross-building) не означает, что вам придется переделывать свой сервер. Операционная система FreeBSD может работать на самых разных аппаратных платформах, таких как i386, amd64 и т. д. Вы можете собрать версию FreeBSD для любой из этих машин на той машине, которая у вас имеется. Например, у меня есть допотопная рабочая станция Sparc (любезно предоставленная Дэвидом О'Брайеном (David O'Brien)) эпохи 1990-х. Она вполне пригодна для эксплуатации, но сборка новой системы на ней заняла бы несколько дней. Однако я могу собрать новую версию FreeBSD для архитектуры sparc64 на моей быстрой и мощной машине amd64, экспортировать каталоги */usr/obj* и */usr/src* через NFS (глава 8), смонтировать их на рабочей станции Sparc и выполнить обычную установку.

Чтобы выполнить кросс-компиляцию FreeBSD для другой аппаратной платформы, следует задать параметр `TARGET` в командной строке.

```
# make TARGET=sparc64 buildworld && make TARGET=sparc64 buildkernel
```

Все допустимые значения этого параметра перечислены в файле */usr/src/Makefile.inc1*, в табл. 13.3 приведены наиболее часто используемые из них.

Таблица 13.3. Допустимые значения параметра TARGET

Значение	Аппаратная платформа
`amd64`	64-битовые платформы AMD и Intel
`arm`	Платформы ARM (встраиваемые системы)
`i386`	Классические архитектуры x86
`ia64`	Платформа Intel Itanium
`powerpc`	Платформа PowerPC (Mac начала 2000-х)
`sparc64`	Классическая 64-битовая платформа Sparc

Эту особенность мы используем в главе 20.

Сборка локального сервера CVSup

При обновлении из исходного кода каждый сервер должен подключаться к серверу FreeBSD CVSup и загружать последние версии файлов исходного кода. Если у вас много серверов, то обновление операционной системы на них – весьма трудоемкое занятие. Обеспечение выхода в Интернет через брандмауэр для всех этих серверов в крупных компаниях может быть нежелательным – сетевой администратор откроет по вашей просьбе порт TCP 5999 для одной машины, но не для всех. Все зеркала поддерживаются добровольцами, которые предоставляют для этого свои серверы и полосу пропускания, а также тратят время на поддержку серверов. Зачем загружать одно и то же снова и снова?

Предположим, вы входите на каждый свой сервер по очереди и обновляете исходный код системы. В промежутках между началом обновления исходного кода на разных машинах код на зеркале может немного измениться. Код на зеркалах обновится не сразу, а процесс обновления на ваших машинах уже идет. Если у вас несколько рабочих систем, то лучше всего, если они будут абсолютно идентичны. Даже если на них установлены версии –stable «выпуска» от 7.1 до 7.2, такое расхождение представляет собой потенциальный источник проблем. Его устранение сильно облегчит выявление неполадок. Надо исключить саму возможность появления мысли вроде «Та-ак. Сервер 86 загибается. Может, из-за того, что на серверах установлены различные версии FreeBSD?», иначе вы рискуете сойти с ума. Преодолеть такие трудности позволит запуск центрального сервера CVSup (он же cvsupd-сервер), то есть локального зеркала исходного кода FreeBSD.

Запустить cvsupd-сервер не так-то просто, но это намного проще, чем запустить веб-сервер и обеспечить его безопасность. «Порт» */usr/ports/net/cvsup-mirror* поддерживает все мудреные этапы конфигурирования зеркала. На самом деле этот «порт» – всего лишь сценарий, выполняющий настройку других программ. Сценарий предложит значения по умолчанию для параметров настройки cvsupd-сервера, и все.

Одна из зависимостей cvsup-mirror – это полный комплект программного обеспечения CVSup. Несмотря на то что программа csup как клиент заменила CVSup, сам cvsupd-сервер по-прежнему остается частью CVSup. По умолчанию при установке CVSup устанавливается и графический интерфейс. Прежде чем начать настраивать зеркало, настоятельно рекомендую установить CVSup без графического интерфейса. Для обеспечения работы сервера вам совершенно не нужен графический интерфейс и все необходимое для его работы программное обеспечение X Window. (Если у вас уже установлен сервер X Window, то можно не беспокоиться об установке графического интерфейса, потому что в таком случае вам не придется устанавливать полный комплект X Window только из-за одной программы.)

```
# cd /usr/ports/net/cvsup-without-gui
```

```
# make all install clean
# cd /usr/ports/net/cvsup-mirror
# make all install clean
```

После загрузки работа сценария приостановится и появится приглашение для ввода информации. Ответ по умолчанию указан в квадратных скобках. Нажатие клавиши Enter означает ввод ответа по умолчанию.

```
Master site for your updates [cvsup-master.freebsd.org]?
(Перевод: Основной сайт для обновлений)
❶cvsup5.us.freebsd.org
How many hours between updates of your files [1]? ❷1
(Перевод: Сколько часов должно проходить между обновлениями файлов?)
```

В первую очередь нужно указать зеркало, откуда будут загружаться обновления ❶. Не принимайте значение по умолчанию, вместо этого укажите достаточно близкий к вам общедоступный сервер. Обычные конечные пользователи не смогут получить доступ к основному серверу CVSup, который зарезервирован для обслуживания официальных зеркал FreeBSD. (Даже я, будучи одним из разработчиков, не имею доступа к основному серверу cvsup-master, разве только в экстраординарных обстоятельствах и по специальному запросу.) Затем нужно решить, как часто будет обновляться содержимое зеркала ❷. Зеркала, с которых вы будете загружать обновления, сами обновляются не чаще одного раза в час, поэтому нет смысла выбирать более короткий интервал. Обычно я устанавливаю значение 168, что соответствует обновлению репозитария один раз в неделю. Если я начну обновлять свои сервера чаще, чем раз в неделю, значит, я делаю что-то не так!

Процесс обновления запускается из cron (глава 15). В большинстве случаев я удаляю задание на выполнение обновления репозитария и обновляю его вручную. Для обновления группы серверов до одной и той же версии -stable я сначала обновляю исходный код на сервере cvsupd, а затем обновляю с этого сервера остальные все машины. Зачастую я обновляю код на тестовой машине, провожу многоэтапное всестороннее тестирование качества и затем обновляю программы на остальных машинах с тем же комплектом исходного кода. Такой подход обеспечивает хорошее качество кода и идентичность систем. Нет никаких требований к частоте обновления кода; в конце концов, это ваш код, поэтому исходите из своих потребностей!

```
Do you wish to mirror the main source repository [y]? y
Where would you like to put it [/home/ncvs]?

(Перевод: Зеркалировать репозитарий основного исходного кода?
Где бы вы хотели хранить его содержимое)
```

Репозитарий основного исходного кода содержит исходный код FreeBSD, а поддержка локального зеркала – основная цель данного примера. */home/ncvs* – это местоположение CVS-репозитария по умолчанию, используйте его, если у вас нет достаточно веских причин для назначе-

ния другого каталога. В обоих случаях значения по умолчанию вполне приемлемы.

```
Do you wish to mirror the installed World Wide Web data [y]? n
Do you wish to mirror the GNATS bug tracking database [y]? n
Do you wish to mirror the mailing list archive [y]? n

(Перевод: Зеркалировать установленные данные Сети?
Зеркалировать базу данных GNATS (для отслеживания ошибок)?
Зеркалировать архив почтовых рассылок?)
```

Ответы на эти три вопроса помогут вам заполнить большой жесткий диск, который вы приберегли на черный день. В состав `installed World Wide Web data` входят установленные версии Справочника, Сборника вопросов и ответов и прочее содержимое веб-сайта FreeBSD. В базе данных GNATS хранятся все отчеты о проблемах (Problem Reports, PR), начиная с появления Проекта в прошлом веке. А архив почтовых рассылок содержит копии всех сообщений во всех почтовых рассылках FreeBSD. Мне и текущие-то рассылки FreeBSD некогда отслеживать, где уж там читать архивы!

```
Unique unprivileged user ID for running the client [cvsupin]?
Unique unprivileged group ID for running the client [cvsupin]?
Unique unprivileged user ID for running the server [cvsup]?
Unique unprivileged group ID for running the server [cvsup]?

(Перевод: Уникальный ID непривилегированного пользователя для запуска клиента
Уникальный ID непривилегированной группы для запуска клиента
Уникальный ID непривилегированного пользователя для запуска сервера
Уникальный ID непривилегированной группы для запуска сервера)
```

Для работы сервера зеркала необходимы два уникальных числовых идентификатора (ID) пользователя и связанных с ними групп. Не используйте учетные записи nobody, nonroot и npgroup – эти пользователи являются владельцами других файлов в системе, запуск с их помощью зеркала cvsupd снизит уровень безопасности системы. Значения по умолчанию обычно вполне приемлемы, если, конечно, вы заранее не создали специальную учетную запись для Базиля ван Пупкинсона или еще кого-то.

```
Syslog facility for the server log [daemon]?
(Перевод: Источник syslog-сообщений сервера)
```

Подробнее о syslog мы поговорим в главе 19, а пока примем значение по умолчанию.

```
Maximum simultaneous client connections [8]?
(Перевод: Максимальное количество одновременных подключений клиентов)
```

Позднее вы без труда сможете изменить максимальное число одновременных подключений клиентов, а пока примем значение по умолчанию. Это число определяет количество машин, одновременно выполняющих обновление.

«Порт» предложит создать учетные записи пользователей и группы, выбранные ранее, и настроить syslogd. Позвольте ему сделать это.

```
Would you like me to set up your crontab for hourly updates [y]? y
(Перевод: Создать настройки в crontab для выполнения обновлений каждый час?)
```

Возможно, вам не требуется выполнять обновления в cvsupd ежечасно, если так – введите в ответ `n`.

Обновление зеркала будет производиться либо с помощью cron(8) (глава 15), либо вручную, запуском сценария */usr/local/etc/cvsup/update.sh*. Вы должны указать полный путь к этой команде. В первый раз обновление выполняется достаточно долго из-за необходимости загрузить весь репозитарий исходного кода, но последующие обновления пройдут значительно быстрее. Чтобы сервер CVSup автоматически запускался, позволяя клиентам подключаться к нему для обновления исходного кода, поместите в файл */etc/rc.conf* строку `cvsupd_enable="YES"`.

Управление доступом

Желание быть хорошим системным администратором и наличие личного репозитария не означает, что вы хотите сделать загрузку с установленного зеркала доступной любому Базилю ван Пупкинсону. Сервер cvsupd позволяет управлять доступом компьютеров к зеркалу.

Файл */usr/local/etc/cvsup/cvsupd.access* управляет списком хостов, которым разрешено подключаться к зеркалу. Строки, начинающиеся с символа «решетки» (#), являются комментариями, символ «плюс» (+) означает, что клиент может подключиться, а символ «минус» (-) – что такой возможности нет. Звездочка (*) означает, что клиент должен пройти аутентификацию, о которой говорится чуть ниже в этом разделе.

Каждое правило в *cvsupd.access* может ссылаться либо на имя хоста, либо на IP-адрес, но IP-адрес предпочтительнее. Также с IP-адресами можно применять сетевые маски. Например, в следующих строках разрешается доступ из сети 192.168.0.0/16 и запрещается доступ клиентов с любых других адресов:

```
+192.168.0.0/16
-0.0.0.0/0
```

Зеркала CVS и разработка

Если хотите поучаствовать в разработке FreeBSD, можете использовать зеркало CVS как локальный CVS-репозитарий. Добавьте свою учетную запись в группу cvsup – и вы получите привилегии, необходимые для выполнения операций в локальном CVS-репозитарии. Если вы не разрабочик, то это не для вас.

Управление доступом по IP-адресам хорошо подходит для статических сетей. Однако вам может потребоваться более гибкая система, где клиенты подключаются со случайных IP-адресов. В этом случае предпочтительнее применить схему аутентификации на основе паролей. К сожалению, клиент csup не поддерживает возможность аутентификации. Если вам требуется поддержка механизма аутентификации, установите */usr/ports/net/cvsup* и прочитайте страницу руководства cvsupd(8). Однако большинство администраторов считают более подходящим способ на основе IP-адресов.

Обновление коллекции «портов»

Вместе с самой системой FreeBSD постоянно обновляется и коллекция «портов». В ходе обновления системы обновляются и установленные «порты». Не забывайте, что дополнительное программное обеспечение может открывать уязвимости в системе безопасности, как и системное программное обеспечение, и потому лучшая защита от нежелательных вторжений – постоянное обновление программного обеспечения. Обновление установленного программного обеспечения выполняется в два этапа: первый – обновление коллекции «портов», второй – обновление установленного программного обеспечения.

Если обновление производится с помощью csup(1), добавьте в файл supfile строку `ports tag=.`, чтобы одновременно с системой обновлялась и коллекция «портов». Однако, обновляя ветку с исправленными ошибками (errata), скорее всего, вы обновляете свою систему с помощью службы FreeBSD Update. Или, может быть, вам не требуется обновлять систему целиком, а только коллекцию «портов». Тогда на помощь придет portsnap(8).

Программа portsnap(8) загружает и устанавливает сжатые моментальные копии и обновления коллекции «портов», помогая системным администраторам обновлять установленное программное обеспечение. Каждый час, или что-то около того, центральный сервер FreeBSD portsnap собирает все изменения в коллекции «портов», внесенные с момента предыдущей проверки. Эти изменения собираются в виде за-

portsnap или csup

Обновлять коллекцию «портов» следует с помощью одной программы – portsnap(8) или csup(1), – не смешивая эти два способа. Эти два инструмента несовместимы между собой. Для системы версии -stable или -current удобнее использовать csup, а для рабочих систем при двоичном обновлении лучше подходит portsnap. В любой ситуации можно использовать portsnap(8) или csup(1), но только какую-то одну из них!

платки. После запуска portsnap(8) загрузит все обновления, накопившиеся с момента последнего обновления дерева «портов», и применит эти изменения к локальной коллекции «портов», предоставив в ваше распоряжение самые свежие версии «портов».

Конфигурирование portsnap

Порядок выполнения обновлений, производимых программой portsnap(8), определяется параметрами настройки в файле */etc/portsnap.conf*. Очень немногие изменят конфигурацию portsnap(8), тем не менее ниже перечислены параметры, значения которых вам, скорее всего, захочется изменить. Можно заметить, что конфигурация portsnap(8) выглядит почти так же, как и конфигурация freebsd-update(8). Оба инструмента были написаны одним и тем же человеком, поэтому они используют сходные механизмы для решения сходных задач. Файл */etc/portsnap.conf* содержит ключевые слова и значения, которые им присваиваются.

`SERVERNAME=portsnap.freebsd.org`

Это имя сервера portsnap. В действительности под именем *portsnap.freebsd.org* скрывается несколько серверов. Возможно, в будущем Проект FreeBSD объявит о запуске региональных серверов для более равномерного распределения нагрузки, но пока это единственный доступный сервер portsnap.

`KEYPRINT=9b5...`

Параметр `KEYPRINT` – это криптографический ключ сервера portsnap. Сервер portsnap(8) подписывает обновления своей цифровой подписью, чтобы гарантировать их подлинность и целостность. Не изменяйте этот параметр.

`REFUSE arabic korean`

Параметр `REFUSE` позволяет определить категории программного обеспечения в «портах», которые не должны обновляться. Указанное в этом примере значение параметра предотвращает обновление категорий программного обеспечения поддержки арабского и корейского языков, как это делается в файле refuse, который используется программой csup(1). Так же как и в случае с программой csup(1), следует проявлять особую осторожность, включая категории в этот список, потому что коллекция «портов» – это единое целое, и при отказе от обновлений крупных частей дерева можно привести систему в противоречивое состояние.

portsnap(8)

При первом запуске программы portsnap(8) выполните команду `fetch extract`:

```
# portsnap fetch extract
```

Получив эту команду, portsnap(8) загрузит последнюю моментальную копию коллекции «портов» и извлечет ее содержимое в каталог */usr/ports*. Команду fetch extract следует использовать всего один раз, с целью приведения portsnap(8) в известное состояние.

При последующих обновлениях следует использовать команду `fetch update`:

```
# portsnap fetch update
```

По этой команде все изменения, произошедшие с момента последнего запуска portsnap(8), будут загружены и установлены в коллекцию «портов».

Если вы планируете регулярно запускать portsnap из cron (1), то вместо команды `fetch update` лучше использовать команду `cron update`. Это поможет равномерно распределять нагрузку на серверах FreeBSD portsnap. Вот пример записи в файле crontab пользователя root, которая запускает portsnap ежедневно в пять часов утра:

```
0   5   *   *   *    /usr/sbin/portsnap cron update
```

В действительности обновление в этом случае будет производиться в случайно выбранный момент времени, в интервале между пятью и шестью часами утра, что *гораздо* более эффективно, чем если бы все пользователи portsnap(8) пытались одновременно подключиться к серверу ровно в пять утра.

Фактически это все, что вам требуется знать для работы с программой portsnap(8), предоставляющей отличный способ получения самых последних версий приложений для FreeBSD без использования csup(1) или обновления всего дерева исходного кода. Единственное, что теперь осталось сделать, – это обновить программное обеспечение, чтобы привести его в соответствие с обновленной версией коллекции «портов».

Обновление установленных «портов»

Если дерево «портов» обновляется с помощью программы csup, то любое программное обеспечение, установленное из «портов», будет иметь самый последний номер версии. Но как быть с приложениями, установленными раньше? Операционная система следит за зависимостями между дополнительными пакетами, и обновление одной программы часто влечет за собой обновление десятков других программ. Справиться с таким обновлением вручную будет очень нелегко. Нельзя ли просто сказать: «Обнови-ка мне Apache», – и дать возможность системе FreeBSD самой разобраться с зависимостями? Для решения этой задачи у FreeBSD есть два инструмента – portupgrade(8) и portmaster(8).

Программа portupgrade – это оригинальный инструмент обновления «портов» FreeBSD. Она написана на языке Ruby и поддерживает работу с базами данных информации, получаемой из дерева «портов».

Программа portmaster – это сценарий командного интерпретатора, способный решать наиболее типичные задачи управления программным обеспечением без привлечения дополнительного программного обеспечения или баз данных, не охватывая при этом все возможные случаи. Мы поговорим о программе portmaster, хотя она и не столь всеобъемлюща, как программа portupgrade. Она намного проще, следовательно, при работе с ней у вас будет меньше шансов ошибиться.

Начальная установка portmaster

Загляните в */usr/ports/potrs-mgmt/portmaster*. Этот «порт» устанавливается обычной командой `make install clean`. После установки запустите команду `portmaster -L`, чтобы увидеть, что скажет программа об установленных «портах». Это может занять несколько минут, поэтому я рекомендую перенаправить вывод команды в файл и после проанализировать его в спокойной обстановке. (Можно также использовать команду `portmaster -l`; она работает быстрее и предоставляет информацию о зависимостях, но не отмечает устаревшие «порты».)

```
# portmaster -L > portmaster.out
```

Программа portmaster делит «порты» на категории в зависимости от взаимоотношений между ними, учитывая программное обеспечение, которое требуется для нормальной работы каждого пакета. Например, я пишу эту книгу в OpenOffice.org. Для корректной работы OOo требуются система X Window System, различные программы с графическим интерфейсом, библиотеки системы безопасности, разнообразные шрифты и кухонная мойка с утилизатором отходов мощностью от 5.5 лошадиных сил. Ни одна из этих программ не является частью базовой системы FreeBSD, и если их не установить, OOo не сможет работать. Поэтому эти программы перечислены в OOo как зависимости. В свою очередь, все эти программы определяют OOo как зависимую программу. Такая структура взаимоотношений позволяет программе portmaster строить дерево зависимостей между «портами», и при обращении к «портам» portmaster использует терминологию деревьев.

Дерево «портов» начинается с *корня* (*root ports*), «порты», находящиеся в корне, сами не имеют зависимостей и от них не зависят другие программы. Например, сама программа portmaster не зависит от какого-либо другого программного обеспечения, кроме себя самой, и ни одна программа в системе не требует наличия portmaster. К корню дерева «портов» обычно относят еще несколько программ, таких как bash, sudo и zip.

«Порты», составляющие *ствол* дерева (*trunk ports*), не имеют зависимостей, но они необходимы для других программ. В эту категорию попадают многие разделяемые библиотеки для работы с графикой, шифрованием и математическими функциями, а также различные языки программирования и сценариев. Например, если вы работаете с язы-

ком Perl, то базовая установка Perl наверняка может рассматриваться как «порт» из ствола дерева.

Выше на дереве находятся *ветви* (*branch ports*). «Порты» в ветвях сами зависят от других программ и от них зависят другие программы. Типичными представителями «портов» в ветвях дерева являются Java, X Window System и механизмы отображения веб-броузера.

«Порты» в *листьях* (*leaf ports*) – это концы ветвей дерева, они зависят от других программ, но ни одна программа не зависит от них. Типичные представители листьев дерева «портов» – это текстовые редакторы, крупногабаритные пакеты офисных приложений, веб-броузеры, программы обмена сообщениями в Интернете и т. д.

При обновлении «портов» изменение в стволе или в ветви дерева «портов» может затронуть его листья. Неудивительно, если при обновлении графической библиотеки до версии, несовместимой с предыдущей, перестанут работать все программы, использующие эту библиотеку.

Идентификация ненужного программного обеспечения

Часто многие из нас устанавливают программное обеспечение только на время, но, наигравшись с ним, забывают удалить его. При обновлении есть смысл удалить эти пакеты, чтобы не тратить время на их модернизацию. Это как раз тот самый момент, когда сведения о «портах» в корне и листьях дерева оказываются особенно ценными.

Список «портов», который выводит portmaster, начинается с корня дерева. Пройдите по списку и отыщите программы, которые вам больше не нужны. Если вы не уверены, что сможете отыскать пакет, запустите команду `whereis` *`имя_пакета`*, чтобы определить, где находится «порт». Ознакомьтесь с описанием пакета, чтобы освежить память, и, если вам не удалось распознать программу или вы вспомнили, что она вам не нужна, можете деинсталлировать ее. Если это окажется что-то важное, всегда можно выполнить повторную установку. Проделайте то же самое для «портов»-листьев в конце списка. К чему обновлять то, что не нужно?

При вызове с ключом `-e` программа portmaster деинсталлирует «порт» и все его зависимости, если они не требуются другим «портам». Например, после деинсталляции OpenOffice.org у меня появится масса ненужных «портов». Команда `portmaster -e openoffice.org-2.2.0` удалит OOo и все «порты», необходимые для OOo, но не нужные другим программам.

Идентификация и обновление программного обеспечения

Кандидаты на обновление в выводе portmaster(8) выглядят примерно так:

```
===>>> linux_base-fc-4_8
       ===>>> New version available: linux_base-fc-4_9
```

На моем ноутбуке в настоящий момент установлен пакет linux_base версии 4_8, и программа обнаружила, что доступна версия 4_9. Стоит ли обновляться при таком незначительном изменении в номере версии? Может, стоит, а может, и нет. Всегда можно заглянуть в хронологию FreeBSD CVSWeb данного пакета и посмотреть, что изменилось. Если в этой версии были исправлены критические уязвимости, то почти наверняка следует обновить программу, а если изменения незначительные, нет смысла возиться с обновлением. В любом случае решать вам.

Если вы решили выполнить обновление, просто передайте программе portmaster имя «порта», который требуется обновить. Можно передать только имя «порта», полный путь к базе данных пакетов для этого «порта» или использовать другие способы, с которыми можно ознакомиться на странице руководства portmaster(8).

```
# portmaster linux_base-fc-4_8
```

Сначала portmaster отыщет все зависимости, которые необходимо обновить в рамках данного обновления, и запустит задания по обновлению этих «портов». Также будет запущена загрузка distfiles для обновляемого «порта» и его зависимостей. В первую очередь portmaster выполнит начальные этапы процесса сборки, требующие взаимодействия с пользователем. Программа постарается получить от вас всю необходимую информацию в самом начале, чтобы потом вы могли отойти с уверенностью в том, что процесс обновления не остановится на полпути, ожидая, пока вы нажмете клавишу Enter.

Всегда есть риск, что процесс обновления завершится с ошибкой. Если установить новую версию невозможно без деинсталляции старой, в случае ошибки можно остаться и без старой, и без новой версии программы. Однако portmaster не просто деинсталлирует «порт»; программа создает резервную копию пакета старой версии. Эта резервная копия удаляется только в случае успешного завершения процесса обновления, но вы можете запретить удаление, использовав ключ `-b`. Резервные копии сохраняются в каталоге */usr/ports/packages/All*. К тому же, если обновляется целая серия пакетов, portmaster сохраняет резервные копии всех зависимых пакетов до полного окончания процесса обновления. Если попытка обновить OpenOffice.org с помощью portmaster потерпит неудачу, потребуется восстановить прежнюю версию OpenOffice.org вместе со всеми зависимостями. Программа portmaster приложит максимум усилий, чтобы не оставить вас с малопригодной смесью старого и нового программного обеспечения.

Я предпочитаю сохранять резервные копии до тех пор, пока не буду уверен, что новая версия программы работает именно так, как мне надо, но некоторые читатели поумнее меня не нуждаются в таких вот ремнях безопасности.

Принудительная пересборка

portmaster не в состоянии идентифицировать все причины, по которым «порт» следует обновить или переустановить. Например, на моем тестовом сервере присутствует последняя версия программы sudo(8). Если проверить список разделяемых библиотек, необходимых программе, можно заметить кое-что любопытное:

```
# ldd `which sudo`
/usr/local/bin/sudo:
❶       libutil.so.5=> /lib/libutil.so.5 (0x28093000)
        libpam.so.3 => /usr/lib/libpam.so.3 (0x2809f000)
        libopie.so.4 => /usr/lib/libopie.so.4 (0x280a6000)
❷       libc.so.6=> /lib/libc.so.6 (0x280af000)
        libmd.so.3 => /lib/libmd.so.3 (0x28193000)
```

Помните, выше в этой главе библиотеки libutil.so.5 ❶ и libc.so.6 ❷ были идентифицированы как устаревшие? Если бы я по неопытности стер их, программа sudo перестала бы работать. Теперь мне нужно отыскать все программы, требующие наличия этих библиотек. portmaster не понимает, что нужно обновить программу sudo, но я-то вижу, что ее необходимо переустановить. Я мог бы перейти в каталог «порта» и выполнить команду `make deinstall && make reinstall`, но есть способ гораздо проще:

```
# portmaster sudo
```

Эта команда вынудит portmaster пересобрать sudo(8). В процессе сборки sudo автоматически свяжется с более новой библиотекой, удалив одну зависимость от этого устаревшего файла. Если то же самое надо сделать с листом дерева «портов», то можно использовать ключ `-f`, который предписывает программе portmaster(8) пересобрать все зависимости, и неважно – требуется это или нет.

Пересборка восходящих зависимостей

Программа portmaster автоматически обновляет программы, *от которых зависит* программа, требующая обновления, но от нее можно также потребовать пересобрать все программы, *которые зависят от* обновляемой программы. Это запросто может вызвать пересборку большого числа «портов», особенно когда речь заходит о критически важных библиотеках. Используйте ключ `-r`, чтобы потребовать от portmaster проверить как нисходящие, так и восходящие зависимости.

> **Внимание! Полный вперед!**
>
> portmaster с ключом `-a` попытается обновить все, что сочтет нужным. С комбинацией ключей `-af` portmaster выполнит пересборку всех «портов».

Изменение зависимостей

Бывает, что один «порт» нужно заменить другим. Предположим, мне захотелось заменить Emacs 20 на Emacs 21. Программа portmaster может выполнить это самостоятельно, если задать ей ключ `-o`. В качестве аргументов этого ключа следует указать новый «порт» и старый, установленный «порт» именно в таком порядке:

```
# portmaster -o editors/emacs20 editors/emacs21
```

Это не совсем типичная ситуация, но после нескольких таких изменений, произошедших в течение последних лет, я рад, что эта возможность есть.

Пропуск «портов»

Иногда отказ от обновления «порта» действительно необходим, даже если при этом возникает рекурсия. Например, когда я выполняю сборку OpenOffice.org, мой ноутбук занят этим целый день. Мне бы не хотелось, чтобы обновление OpenOffice.org происходило в результате обновления другого компонента; я предпочитаю обновлять этот пакет самостоятельно.

portmaster проверяет каждый каталог в базе данных пакета на наличие файла *+IGNOREME*. Если этот файл присутствует, portmaster не будет обновлять такой пакет в ходе обновления других пакетов.

```
# cd /var/db/pkg/openoffice.org-2.0.20060818
# touch +IGNOREME
```

Теперь, даже если я начну обновлять библиотеку, от которой зависит OpenOffice.org, и прикажу программе portmaster обновить все программы, зависящие от этой библиотеки, portmaster не станет обновлять OOo. В результате подобного обновления зависимостей OOo может выйти из строя, но выбирая этот путь, я иду на риск сознательно.

Другие особенности portmaster

Настоятельно рекомендую прочитать страницу руководства portmaster(8), чтобы ознакомиться с функциями, предлагаемыми этой программой для решения ваших проблем. Вы можете возобновить прерванную сборку; посмотреть, что сделала бы программа portmaster, если бы вы предложили ей обновить систему; сохранить пакеты в вашем репозитарии пакетов и т. д. Если у вас возникли проблемы с обновлением «порта», portmaster наверняка сможет предложить решение.

Уменьшение размера дерева «портов»

С течением времени растет и дерево «портов». Отчасти это хорошо, так как в дереве появляются новые «порты». Отчасти это необходимое зло, так как в дереве сохраняются устаревшие версии программ, кото-

рые вы только что обновили. В остальном этот рост было бы желательно предотвратить, так как в системе остаются файлы, которые когда-то были необходимы, а теперь уже не нужны. Например, когда я обновил пакет до версии Fedora Core 4, файлы пакета Fedora Core 3 стали ненужными. Поиск и удаление ненужных файлов позволит сэкономить гигабайты дискового пространства, которое, между прочим, не бесконечно. Хотя portmaster предоставляет возможность удаления ставших ненужными файлов distfiles, тем не менее предварительно стоит выполнить глобальный поиск таких файлов.

Программа portmaster может отыскивать устаревшие файлы distfiles. Если запустить команду `portmaster --clean-distfiles`, portmaster(8) идентифицирует каждый устаревший файл distfile и предложит удалить его. Если запустить команду `portmaster --clean-distfiles-all`, portmaster(8) молча удалит устаревшие файлы distfiles.

Другая распространенная проблема – когда после установки «порта» не выполняется команда `make clean`. Нередко я хотел бы иметь простой доступ к исходному коду для ознакомления с программой, поэтому не чищу свои «порты». В конечном счете я забываю об этом, а ставшие ненужными файлы продолжают занимать дисковое пространство. Работая над этим разделом, я обнаружил, что забыл удалить временные файлы после последней сборки OpenOffice.org; неудивительно, что у меня на диске так мало свободного места! «Порты» операционной системы FreeBSD легко поддаются массовой чистке; достаточно зайти в каталог */usr/ports* и запустить команду `make clean NOCLEANDEPENDS=yes`. В результате будет выполнен рекурсивный обход дерева «портов» и для каждого из них будет исполнена команда `make clean`, которая удалит каждый рабочий каталог.

С помощью этих инструментов вы сможете своевременно обновлять свою систему. А теперь перейдем к основам администрирования FreeBSD и посмотрим, что еще можно сделать с операционной системой FreeBSD.

14

Ориентирование в Интернете: DNS

Система доменных имен (*Domain Name Service, DNS*) – одна из тех незаметных, закулисных программ, которым не уделяется и половины того внимания, которого они заслуживают. Многие пользователи никогда не слышали о DNS, однако именно эта система делает Интернет таким, каким мы его знаем. DNS, или *служба имен*, обеспечивает отображение между именами хостов и IP-адресами. Без DNS веб-броузеру и программам электронной почты не были бы понятны удобные имена вроде *www.cnn.com*, и пришлось бы набирать числовые IP-адреса, что значительно снизило бы популярность Интернета. Для большинства конечных пользователей нет DNS – нет Интернета, все пропало.

Система DNS необходима всем сервисам Интернета. Здесь мы обсудим работу DNS, способы ее проверки, конфигурирование системы FreeBSD для использования DNS и сборку собственного DNS-сервера.

Принцип действия DNS

DNS просто отображает IP-адреса в имена хостов, а имена хостов – в IP-адреса. Например, пользователю не надо знать о том, что *www.absolutefreebsd.com* на самом деле представляет собой последовательность чисел 198.22.63.8. Он просто вводит URL нужной страницы в веб-броузере, а DNS преобразует это имя в ее числовой адрес. Системный администратор должен уметь устанавливать, контролировать и проверять информацию DNS, а также понимать, как система выполняет эти операции.

Информация DNS может находиться в любом месте: в локальной системе, на локальном DNS-сервере, на удаленном сервере имен. В системах UNIX для предоставления такой информации используется *распознаватель* (*resolver*) – программа, которая знает обо всех источниках данной информации и взаимодействует с ними. Когда программе требуется узнать IP-адрес хоста или имя хоста по IP-адресу, она обраща-

ется к распознавателю. Распознаватель консультируется с соответствующими источниками информации и возвращает результат программе, которой он необходим. Далее в этой главе будет рассмотрена настройка распознавателя.

Как правило, DNS-запрос, получаемый от программы, распознаватель перенаправляет *серверу имен* – компьютеру, на котором работает программа, собирающая информацию DNS с других компьютеров в Интернете. Получив DNS-запрос, сервер имен проверяет в своем локальном кэше, запрашивалась ли эта информация в последнее время. Если информация присутствует в кэше, сервер имен возвращает тот же самый ответ, который получил предыдущий пользователь. Серверы имен получают много идентичных DNS-запросов; например, сервер имен одного из моих интернет-провайдеров в течение часа получает несколько сотен запросов IP-адреса для *www.cnn.com*. Добавьте сюда запросы к Yahoo!, eBay и MSN, и станет ясно, что кэш довольно эффективен.

Представьте, что мы ищем такую информацию, как IP-адрес хоста *www.absolutefreebsd.com*, но на локальном сервере нет информации для ответа, потому что к этому хосту давно никто не обращался. Тогда локальный сервер обращается к *корневому серверу* (*root server*), который хранит список серверов имен, отвечающих за все домены в Интернете. Путем *рекурсивного запроса* (*recursive query*) корневой сервер сообщает корневому серверу о подходящих серверах имен, которые, в свою очередь, могут перенаправить запрос другим серверам. В конце концов, запрос попадает к *авторитетному серверу имен* (*authoritative nameserver*), отвечающему за этот домен, – единственному надежному источнику информации о нужном домене. В результате локальный сервер имен получает ответ на свой вопрос.

Прямой и обратный DNS

Конечно, в сервере DNS нет ни сцепления, ни задней передачи. Прямой и обратный DNS – это типы отображения (map) при разрешении имен. Метод прямого преобразования отображает имена хостов в их IP-адреса; он поддерживает псевдонимы и прочие приятные мелочи. Метод обратного преобразования отображает IP-адреса в имена хостов; он не такой гибкий, как метод прямого преобразования. С одним именем хоста может быть связано несколько IP-адресов, но каждый IP-адрес может отображаться только в одно имя хоста.

Например, компания-провайдер может содержать несколько тысяч веб-сайтов на одном сервере. Прямое отображение преобразует все IP-адреса этих веб-сайтов к одному и тому же серверу. Обратное отображение дает единственный ответ – фактическое имя хоста для сервера, на котором находятся все эти веб-сайты.

Каждый клиент в Интернете надеется получать информацию о любом домене из авторитетного источника – сервера имен этого домена. В каждом домене должно присутствовать по крайней мере два сервера имен. Если один сервер имен выходит из строя, вся нагрузка ложится на другой, а если не действует ни один сервер имен, домен «исчезает» из Интернета. Тогда при попытке обратиться к вашему домену пользователи получают сообщение «домен не найден», почта не доходит, все в проигрыше. Иногда это происходит даже с большими компаниями, такими как Microsoft. Менеджер или заказчики *уведомят* вас о сбое, причем не самым приятным образом. Не забывайте о своей службе имен!

Основные инструменты DNS

В FreeBSD есть несколько инструментов для проверки информации DNS. Поскольку большинство серверов DNS работают по протоколу передачи дейтаграмм пользователя (User Datagram Protocol, UDP – глава 6), вы не можете применить telnet и непосредственно обратиться к серверу, как в случае с электронной почтой и веб-сервисом (о них речь пойдет позже). Доступ к информации DNS можно получить только с помощью host(1) и dig(1).

Команда host(1)

Для быстрого получения IP-адреса хоста применяйте команду host(1). Например, адрес веб-страницы моего издателя можно получить так:

```
# host www.nostarch.com
www.nostarch.com has address 72.32.92.4
```

Это, пожалуй, самый простой вид запроса к DNS: «Вот тебе имя хоста, верни его IP-адрес». Другие, на первый взгляд, довольно простые запросы могут привести к более сложному результату:

```
# host www.cnn.com
❶ www.cnn.com is an alias for cnn.com.
❷ cnn.com has address 64.236.24.20
cnn.com has address 64.236.24.28
...
❸ cnn.com mail is handled by 10 atlmail5.turner.com.
cnn.com mail is handled by 20 nycmail2.turner.com.
...
```

Этот пример показывает, что один и тот же хост может иметь несколько псевдонимов – в данном случае имя *www.cnn.com* в действительности является псевдонимом хоста *cnn.com* ❶.

На самом деле, у хоста *cnn.com* восемь отдельных IP-адресов ❷ (в примере они не показаны). Здесь мы также видим почтовые серверы, которые занимаются обработкой электронной почты для *cnn.com* ❸.

Введя в веб-броузере адрес *http://www.nostarch.com*, пользователь, фактически, направляет веб-броузер на машину с именем *www.nostarch.com*

за ее веб-страницей по умолчанию. Броузер отправляется по единственному IP-адресу, указанному для этой машины. Когда запрашивается адрес *http://www.cnn.com*, распознаватель выбирает один из IP-адресов этой машины и передает его броузеру. Подобным образом почтовые серверы отправляют электронную почту одному из хостов, идентифицированных как серверы обработки почты данного домена. Приложениям требуется не только IP-адрес, но IP-адрес имеет очень большое значение. Не найдя компьютер, на котором находится веб-страница, вы не получите эту страницу!

Углубляемся в детали

Команда host(1) довольно полезна, но она, безусловно, не предоставляет подробную информацию. Кроме того, вы не знаете, откуда взялись данные – из кэша на локальной машине, или же сервер имен добыл их на сервере, ответственном за этот домен. Программа dig(1) – это стандартная программа для нахождения подробной информации DNS. (Другой инструмент, nslookup(1), долго был популярен, но устарел и сейчас практически не используется.) У dig есть множество ключей, которые позволяют выявлять различные неполадки, связанные со службой имен. Здесь рассмотрены основные возможности этой программы.

Базовая форма ее вызова включает имя «dig» и имя хоста. Например, чтобы найти сведения о веб-сервере моего издателя, можно ввести такую команду:

```
# dig www.nostarch.com
❶ ; <<>> DiG 9.3.2-P1 <<>> www.nostarch.com
❷ ;; global options:  printcmd
❸ ;; Got answer:
;; ->>HEADER<<- opcode: QUERY, ❹status: NOERROR, id: 33643
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2
❺ ;; QUESTION SECTION:
;www.nostarch.com.              IN      A
❻ ;; ANSWER SECTION:
www.nostarch.com.       2550    IN      A       72.32.92.4
❼ ;; AUTHORITY SECTION:
nostarch.com.           2550    IN      NS      ns2.laughingsquid.net.
nostarch.com.           2550    IN      NS      ns1.laughingsquid.net.
❽ ;; ADDITIONAL SECTION:
ns1.laughingsquid.net.  171750  IN      A       72.3.155.219
ns2.laughingsquid.net.  171750  IN      A       72.32.93.189
;; Query time: 58 msec
;; SERVER: 198.22.63.8#53(198.22.63.8)
;; WHEN: Mon Nov 20 18:43:22 2006
;; MSG SIZE rcvd: 135
```

Как много строк! Но обычно с помощью программы dig(1) ищут источник проблем, а в этом случае лучше узнать больше, чем меньше. Проанализируем информацию.

Прежде всего, строки, начинающиеся точкой с запятой, являются комментариями, которые не имеют прямого отношения к ответу, но могут содержать немаловажную информацию. Например, первая строка сообщает версию программы dig ❶. Во второй строке представлены ключи, которые применяет dig ❷. Поскольку в командной строке ключи не были указаны, dig выполняется с ключами по умолчанию.

Далее следует первый фрагмент интересующей нас информации: откуда программа dig получила ответ ❸. Элемент `status` ❹ содержит важное слово `NOERROR`, подтверждающее, что запрос был выполнен без ошибок. Если слова `NOERROR` нет, то что-то не так. Типичные ошибки: `NXDOMAIN` (домен не существует) или `SERVFAIL` (сервер имен знает о существовании этого домена, но не имеет никакой информации о нем).

Раздел QUESTION

В разделе `QUESTION` ❺ приводится фактический запрос, выполненный программой dig. В данном примере раздел запроса выглядит так:

```
;www.nostarch.com.              IN      A
```

Первая строка начинается точкой с запятой, то есть перед нами комментарий. Это текст запроса к службе DNS в формальном представлении. Сначала dig подтверждает, что была запрошена информация об имени *www.nostarch.com*, но что означает `IN A`? DNS может обслуживать различные системы имен, а не только IP-адреса и имена хостов в стиле Интернета. Комбинация символов `IN` указывает, что это система имен Интернета – все имена доменов принадлежат к классу `IN`. Завершается запрос описанием его типа – в данном случае запрашивался адрес (`A`). У нас есть имя хоста, и мы хотим получить его адрес. Тип `PTR` (pointer – ссылка) мог бы означать запрос на обратное разрешение имени в DNS (когда имеется IP-адрес и по нему требуется получить имя хоста). При работе с dig вам будут встречаться и другие типы запросов.

Раздел ANSWER

Раздел `ANSWER` ❻ содержит полученную информацию.

```
www.nostarch.com.       2550  IN        A       72.32.92.4
```

Мы запросили информацию для хоста *www.nostarch.com*, поэтому первым указано имя. Далее следует число `2550` – это *время жизни* (*Time-To-Live, TTL*) данной информации. Наш локальный сервер имен может удерживать эту информацию в кэше в течение последующих 2250 секунд (примерно 42 минуты). По истечении этого времени локальный сервер имен удалит запись из кэша. Если кто-то еще раз запросит эту же информацию, локальный сервер DNS отправится за новым ответом. Пара символов `IN` означает данные Интернета, а `A` – что это адрес. В конце указан IP-адрес хоста *www.nostarch.com* – 72.32.92.4.

Раздел AUTHORITY

Для получения IP-адреса *www.nostarch.com* необходимо пройти по цепочке авторитетных серверов имен. В разделе `AUTHORITY` ❼ перечислены серверы имен, ответственные за домен, – в нашем случае *ns1.laughingsquid.net* и *ns2.laughingsquid.net*. Информация о *www.nostarch.com* будет находиться в кэше в течение 2550 секунд. И снова это адреса Интернета (`IN`).

Раздел ADDITIONAL

Наконец, в разделе `ADDITIONAL` ❽ dig перечисляет IP-адреса всех хостов, представленных вместе с интересующим нас хостом. Программа dig определила авторитетные серверы имен для домена *nostarch.com*, но нам могут потребоваться IP-адреса этих серверов, чтобы в будущем выполнять DNS-запросы на получение информации о домене *nostarch.com*. Информация об авторитетных серверах имен получена, но обратите внимание на значение TTL – оно намного больше, чем то же значение в оригинальном запросе. Этот ответ будет храниться в кэше 171 750 секунд (48 часов). По истечении этого времени ваш локальный сервер имен удалит эти записи из кэша и снова обратится к корневому серверу имен за информацией об авторитетных серверах DNS для домена *nostarch.com*.

Запустите dig несколько раз с разными доменами и проанализируйте ее вывод.

Поиск имен хостов с помощью dig

Предположим, вы знаете IP-адрес и хотите выяснить связанное с ним имя хоста (как, например, по телефонному номеру определить его владельца). Это типичная задача в Интернете – скажем, пользователь с какого-то IP-адреса то и дело обращается к вашему веб-серверу, а вы хотите узнать, кто он такой и что ему нужно. Найти имя позволит *обратный поиск* (*reverse lookup*) с помощью ключа `-x` команды dig. Поскольку большая часть вывода при этом практически совпадает с результатам прямого поиска, рассмотрим только существенные различия.

```
# dig -x 72.32.92.4
...
;; QUESTION SECTION:
;❶4.92.32.72.in-addr.arpa.     IN         ❷PTR

;; ANSWER SECTION:
4.92.32.72.in-addr.arpa. 3600  IN          PTR    ❸squid14.laughingsquid.net.
```

Одно из наиболее очевидных отличий: сам запрос ❶ выглядит несколько необычно. Что за строка *4.92.32.72.in-addr.arpa*? При обратном поиске компоненты IP-адреса упорядочиваются в обратном порядке и помещаются в домен *in-addr.arpa*.

Кроме того, мы хотим получить запись типа `PTR` ❷, а не `A`. Тип `PTR` означает, что требуется определить имя хоста по известному IP-адресу.

Мы знаем, что имени *www.nostarch.com* соответствует IP-адрес 72.32.92.4, но обратный поиск позволяет получить более корректное имя этого хоста – *squid14.laughingsquid.net* ❸. Во многом это напоминает пример с телефонными номерами, когда одним номером может пользоваться целая семья, но зарегистрирован он на кого-то одного.

В общем случае результаты прямого и обратного поиска должны соответствовать друг другу, но поскольку один IP-адрес может соответствовать нескольким хостам сразу, запись типа `A` может не соответствовать записи типа `PTR`. Имя *www.nostarch.com* отображается в IP-адрес 72.32.92.4, но адрес 72.32.92.4 отображается в имя *squid14.laughingsquid.net*. Вполне очевидно, что запись типа `A` для имени *squid14.laughingsquid.net* должна указывать на IP-адрес 72.32.92.4. Если имя хоста, полученное при обратном поиске, не имеет соответствующей записи при прямом, это означает, что сервер DNS настроен неправильно. Инструменты, опирающиеся на DNS, такие как TCP wrappers (при определенных настройках), будут отвергать запросы на соединения от таких систем. Это можно наблюдать в домашних сетях, построенных на основе технологии ADSL, где по IP-адресу кабельного модема можно получить имя его владельца. Ниже в этой главе мы познакомимся с инструментами, которые автоматически проверяют соответствие результатов прямого и обратного поиска в DNS.

Ключи dig

Для получения полного списка ключей обратитесь к странице руководства dig(1), а ниже я приведу самые полезные. Я наиболее часто использую ключи для обращения к какому-то конкретному серверу и запрещающие рекурсию.

Запрос конкретного сервера имен

По умолчанию dig запрашивает первый сервер имен, указанный в */etc/resolv.conf*. Однако при устранении неполадок может возникнуть потребность обратиться к какому-то конкретному серверу имен. Для этого при запуске dig следует указать в командной строке ключ `@`. Например, чтобы спросить у *dns1.wideopenwest.com*, что он знает о домене *nostarch.com*, введите такую команду:

```
# dig nostarch.com @dns1.wideopenwest.com
```

Вывод этой команды довольно похож на вывод, представленный в предыдущем примере, за исключением того, что вы увидите ссылки на сервер имен *dns1.wideopenwest.com*. При выявлении неполадок необходимо тщательно сравнить эту информацию с результатами, выдаваемыми локальным сервером имен. Если данные от двух разных серверов имен противоречивы, это может дать ключ к разгадке.

Запрет рекурсии

По умолчанию для получения ответа сервер имен выполняет рекурсивный запрос. Иногда это нежелательно, например, когда нужно узнать лишь, что имеется в кэше этого сервера и к какому другому серверу он обратился бы за получением ответа на запрос. Часто бывает полезно выполнить рекурсию самостоятельно, особенно при выявлении неполадок. Чтобы запретить рекурсию, в командной строке следует указать ключ `+norecurse`. Попробуйте использовать этот ключ, обращаясь к локальному серверу имен за информацией о домене, к которому перед этим не было никаких обращений. Если информация об этом домене имеется в кэше сервера имен, результаты будут очень похожи на те, что приводились в предыдущем примере. В противном случае вы получите примерно следующее:

```
# dig absolutefreebsd.com +norecurse
...
❶ ;; QUESTION SECTION:
;AbsoluteFreeBSD.com.          IN    A
❷ ;; AUTHORITY SECTION:
❸ com.              33195     IN    NS        A.GTLD-SERVERS.NET.
com.                33195     IN    NS        B.GTLD-SERVERS.NET.
com.                33195 IN NS C.GTLD-SERVERS.NET.
...
;; ADDITIONAL SECTION:
A.GTLD-SERVERS.NET.     98529   IN      A       192.5.6.30
❹ A.GTLD-SERVERS.NET.   129969  IN      AAAA    2001:503:a83e::2:30
B.GTLD-SERVERS.NET.     98529   IN      A       192.33.14.30
B.GTLD-SERVERS.NET.     129969  IN      AAAA    2001:503:231d::2:30
C.GTLD-SERVERS.NET.     98529   IN      A       192.26.92.30
...
```

Обратите внимание: сразу вслед за разделом `QUESTION` ❶ идет раздел `AUTHORITY` ❷. Здесь отсутствует раздел `ANSWER` (ответ). Ответа нет, потому что у сервера имен *нет* ответа. Вместо этого он отсылает к авторитетным серверам имен для домена *.com* ❸, которые находятся в домене *gtld-servers.net*. В разделе `ADDITIONAL` (дополнительная информация) приводятся IP-адреса этих серверов. Примечательно, что некоторые из этих серверов имеют адреса в формате IPv6 ❹, помимо стандартных адресов IPv4. Если система не поддерживает адреса IPv6, она не будет их использовать.

Отсылая к серверам имен домена *.com*, данный сервер как бы говорит: «Не знаю, спросите у этих парней». Вы попросили свой сервер имен не выполнять рекурсивный запрос, он так и поступил.

Чтобы запросить один из предложенных серверов имен домена *.com*, следует указать выбранный сервер в командной строке и опять использовать ключ `+norecurse`.

```
# dig www.absolutefreebsd.com @a.gtld-servers.net +norecurse
```

В результате вас отошлют к другой группе серверов имен. Следуя по такой цепочке, вы сможете увидеть, как в точности сервер имен выполняет запрос.

in-addr.arpa

У записей типа `PTR` есть существенный недостаток – информация в них указывается в обратном порядке. Сервер DNS проверяет компоненты имени справа налево. При поиске хоста с именем *www.absolutefreebsd.com* сервер имен сначала отыскивает сервер имен, обслуживающий домен *.com*. Затем в домене *.com* отыскивается сервер имен для домена *absolutefreebsd.com*, а затем уже *www.absolutefreebsd.com*. Самый крупный элемент находится справа. Основной сервер домена *.com* перенаправляет запрос серверу имен поддомена *absolutefreebsd*, то есть моему серверу, а он в свою очередь отыскивает адрес конкретной машины *www*.

В IP-адресах самый крупный элемент находится слева. Возьмем IP-адрес 72.32.92.4. Первое число можно сравнить с целым доменом *.com*. Все IP-адреса, принадлежащие блоку 72, распределяются Американским бюро регистрации номеров Интернета (American Registry for Internet Numbers, ARIN). В свою очередь ARIN передает право распоряжаться блоком адресов 72.32 конкретной компании-провайдеру. Эта компания передает блок адресов 72.32.92 клиенту, который присваивает адрес 72.32.92.4 конкретной машине. Концепция передачи права распоряжаться блоками адресов проста, но происходит слева направо – в противоположном направлении тому, как распределены имена хостов.

IP-адрес, записанный в прямом порядке, очень легко перепутать с IP-адресом, записанным в обратном порядке, поэтому в DNS используется специальный домен, который указывает, что IP-адрес, записан в обратном порядке. К перевернутым IP-адресам справа добавляется строка *in-addr.arpa*. (Причины такого расположения уходят корнями в прошлое; они довольно скучны и здесь рассматриваться не будут.) В результате наш адрес 72.32.92.4 превращается в 4.92.32.72.in-addr.arpa.

Тогда почему бы не оставить прямую запись IP-адреса и не использовать строку in-addr.arpa для обозначения обратного запроса DNS? Хороший вопрос! Дело в том, что предыдущий запрос достаточно прост, потому что поиск производится в весьма ограниченной области. Однако если вы владеете блоком адресов 72.32.92.0/24[1], возможно, вам придется запросить информацию целиком обо всей области, которой распоряжается компания-провайдер, то есть информацию о 92.32.72.in-addr.arpa. Точно так же можно запросить информацию о целом блоке 72.32/16, или 32.72.in-addr.arpa, и даже о блоке 72.in-addr.arpa. Каждый такой запрос подразумевает проверку значительного пространства – аналогично запуску `dig .com`. Вам, скорее всего, никогда не понадобится за-

[1] Если вас смущает запись /24, вернитесь к главе 6 и прочитайте ее еще раз.

пускать `dig .com`, однако это потребуется инженерам, обслуживающим магистрали Интернета. Именно они и пишут такие программы. Один из недостатков профессиональных инструментов заключается в том, что в первую очередь они адресованы профессионалам.

> **Быстрый и грубый поиск**
>
> Если вам срочно нужна приблизительная информация DNS, программа host(1) сама выполнит обратное преобразование. То же самое сделает программа dig с ключом `-x`. Пусть запись *in-addr.arpa* не смущает вас.

Настройка распознавателя

Распознаватель отвечает за доставку информации DNS клиентским программам. Настройка распознавателя – это важная часть администрирования системы, необходимая для работы даже самого сервера DNS, потому что компьютер ничего не знает о других серверах имен. Всем операциям, выполняемым в сети, требуется работоспособный распознаватель. Настройка распознавателя связана с получением ответов на следующие вопросы:

- Где сервер сможет получить информацию DNS?
- Какие имена используются для локальных доменов?
- К каким серверам имен будет выполняться обращение?

Ответы на эти вопросы определяются настройками в файлах */etc/nsswitch.conf* и */etc/resolv.conf*.

Источники информации DNS

Похоже, все должно быть просто; сервер получает информацию о своем хосте у сервера имен, так? Собственно, об этом и глава?

В реальном мире все не так просто, как кажется. Возможно, у вас есть небольшая домашняя сеть всего из трех машин, поэтому вы считаете запуск сервера DNS бессмысленным. А может, вы пользуетесь локальной сетью крупной компании, где полная замена DNS может занимать недели. Операционная система FreeBSD (и другие UNIX-подобные операционные системы) поддерживает возможность разрешения имен как с помощью DNS, так и с помощью файла */etc/hosts*, и вам решать, с какого источника информации начать.

Если системе FreeBSD требуется узнать адрес по имени хоста (или наоборот), по умолчанию в первую очередь проверяется файл hosts, а затем выполняются запросы к указанным серверам имен. Это означает, что вы можете переопределить информацию, получаемую от сервера имен для локальных машин, что бывает очень полезно для работы

Выбор службы имен

Файл */etc/nsswitch* нужен не только для поиска имен хостов или IP-адресов; это общий файл конфигурации выбора службы имен. В сетевой операционной системе много служб имен. Например, в главе 6 рассматривались имена сетевых служб и номера портов в файле */etc/services*, а также имена протоколов и их номера. Поиск имени протокола выполняет отдельная служба имен, которая может обрабатывать запросы разными способами. Для определения числовых идентификаторов пользователя UID и GID (глава 7) требуется служба имен иного рода. Порядок выполнения всех этих запросов и многое другое определяется в файле */etc/nsswitch.conf*. Но здесь мы рассматриваем только службу поиска имен хостов.

с хостами за брандмауэром или в крупных корпоративных сетях с необычными требованиями.[1] В некоторых ситуациях бывает предпочтительнее изменить такой порядок вещей и сначала попробовать обратиться к службе DNS, а затем уже к файлу hosts. В современных версиях FreeBSD этот порядок устанавливается в файле */etc/nsswitch.conf*. (В старых версиях для этих целей использовался файл */etc/host.conf*, упоминания о котором до сих пор можно встретить в документации.)

Вот пример настройки службы поиска имен хостов в файле */etc/nsswitch.conf*:

```
hosts: files dns
```

Распознаватель обращается к источникам информации в указанном здесь порядке. Если у вас есть дополнительные источники информации, например cached (глава 15), добавьте их сюда. Чтобы определить обратный порядок поиска имен – в первую очередь обращаться к DNS и только потом к файлу hosts, – поменяйте порядок их следования в файле конфигурации.

Второй источник информации будет задействован только в том случае, если первый источник не нашел нужную запись. Если информация в */etc/hosts* и DNS отличается, используется информация из первого источника.

Настройка локальных доменных имен

Работая на машинах в своей локальной сети, едва ли кто-то захочет вводить полные имена хостов. Если у вас 30 веб-серверов, то, набирая `ssh www19.mycompany.com`, можно состариться. Чтобы указать распозна-

[1] К сожалению, когда речь заходит о корпоративных сетях, странности становятся нормой.

вателю, какие домены проверять по умолчанию, можно перечислить их в первой строке файла */etc/resolv.conf*.

Задание локального домена

Ключевое слово `domain` говорит распознавателю, в каком домене по умолчанию искать хосты. Например, чтобы задать *absolutefreebsd.com* в качестве локального домена, начните файл */etc/resolv.conf* с такой строки:

```
domain absolutefreebsd.com
```

После того как задан локальный домен, любая команда, которой обычно требуется доменное имя, будет считать доменом *absolutefreebsd.com*. Если я выполняю команду `ping www`, распознаватель добавит имя *absolutefreebsd.com* к *www* и предпишет выполнить ping по адресу *www.absolutefreebsd.com*.

Задание списка доменов

В качестве альтернативы можно применить ключевое слово `search` и указать список доменов. Допустим, моей компании принадлежит несколько доменов, в которых мне нужно выполнить поиск. Значит, я могу начать файл */etc/resolv.conf* со следующей строки:

```
search absolutefreebsd.com blackhelicopters.org stenchmaster.org
```

Распознаватель будет проверять эти три домена в порядке их указания, пока не найдет совпадение. Например, если я выполняю `ping petulance`, распознаватель попытается найти *petulance.absolutefreebsd.com*. Если не удастся, он проверит *petulance.blackhelicopters.org* и затем *petulance.stenchmaster.com*. Если такого хоста нет ни в одном из этих доменов, команда не будет выполнена.

> **Домен по умолчанию**
>
> Если не указать ни одно из ключевых слов `domain` или `search`, распознаватель будет использовать доменное имя локальной машины.

Список серверов имен

Итак, распознаватель знает, в каких доменах искать по умолчанию, и надо сообщить ему, к каким серверам имен обращаться. Перечислите все серверы имен в отдельных строках файла */etc/resolv.conf* – в порядке предпочтения. Распознаватель будет обращаться к серверам имен в этом порядке. Список в файле */etc/resolv.conf* выглядит примерно так:

```
domain absolutefreebsd.com
nameserver 127.0.0.1
nameserver 192.168.8.3
nameserver 172.18.33.4
```

Обратите внимание: первая запись в списке – это IP-адрес 127.0.0.1, принадлежащий «обратной петле» (loopback). Данная запись нужна на машине, выполняющей функции сервера имен, поскольку она предписывает распознавателю обращаться к серверу имен локального хоста. Иногда (довольно редко) возникает желание не применять локальный сервер. Можно поступить и так, но в большинстве случаев это напрасная трата сетевой полосы пропускания.

Если в */etc/resolv.conf* есть записи `nameserver`, а также ключевые слова `domain` или `search`, ваш распознаватель сконфигурирован. Теперь нужно настроить эти источники информации, чтобы распознаватель мог их использовать.

Файл /etc/hosts как замена локального сервера DNS

В файле */etc/hosts* сопоставляются IP-адреса и имена отдельно взятых хостов. Несмотря на всю эффективность применения файла hosts, его содержимое имеет силу только на отдельной машине и должно настраиваться вручную. Динамические серверы имен в значительной степени вытеснили */etc/hosts*, однако файл hosts все еще полезен в маленьких сетях и за устройством преобразования сетевых адресов (Network Address Translation, NAT). Например, файл hosts хорошо подходит, если у вас только один или два сервера, а за управление вашей службой имен, доступной извне, отвечает кто-то другой. Если же у вас несколько серверов, каждый из которых должен поддерживаться отдельно, необходим полноценный сервер имен.

Когда-то давно в Интернете имелся единый файл hosts, содержавший все IP-адреса и имена хостов Сети. Системные администраторы оповещали об изменениях в своих системах центрального администратора, который выпускал обновленный файл hosts раз в несколько месяцев. Такой подход вполне оправдывал себя, пока в Интернете было всего четыре хоста. Позже, при нескольких сотнях интернет-хостов, его еще можно было применять. Но когда Интернет начал расти не по дням, а по часам, от этой схемы вскоре отказались.

Каждая строка в */etc/hosts* представляет один хост. Первая запись в каждой строке – это IP-адрес, а вторая – полностью определенное доменное имя (fully qualified domain name) хоста, например *mail.absolute-freebsd.com*. Далее можно перечислить все псевдонимы этого хоста.

Например, у небольшой компании может быть единственный сервер, который обрабатывает почту, предоставляет сервисы FTP, веб, DNS и выполняет множество других функций. Рабочая станция в данной сети может иметь такую запись в */etc/hosts*:

```
192.168.1.2 mail.mycompany.com          mail ftp www dns
```

С помощью этой записи в */etc/hosts* рабочая станция может найти этот сервер либо по полному доменному имени, либо по кратким псевдонимам.

Если обнаружилось, что вам требуется больше двух или трех строк в hosts или что поддерживать файлы hosts стало нелегко, – это сигнал к тому, что для обслуживания данных о хостах надо создать сервер имен. Сервер имен намного более масштабируем по сравнению с файлами hosts для каждой машины. Обслуживать сервер имен намного проще – стоит только его установить.

Создание сервера имен

Самое популярное программное обеспечение для создания сервера DNS – это *BIND* (*Berkeley Internet Name Daemon*). На самом деле BIND представляет собой набор инструментов, в состав которого входят такие программы, как host(1), dig(1) и собственно сервер DNS – named(8). BIND поддерживается Консорциумом программного обеспечения Интернета (Internet Software Consortium, *http://www.isc.org*) и выпускается под BSD-подобной лицензией. Хотя у BIND есть конкуренты, например djbdns (*/usr/ports/net/djbdns*), BIND считается эталонной реализацией службы имен. Большинство конкурентов пакета BIND были написаны много лет тому назад, когда BIND еще «славился» своей уязвимостью. Но с тех пор BIND был полностью переписан. Последние его версии гораздо безопаснее прежних, поэтому нет веских причин рассматривать его конкурентов. Не важно, какой демон DNS вы будете использовать, – концепции, применяемые в BIND, в целом подходят для других серверов имен. Одна из наиболее важных концепций – это взаимоотношения между первичными и вторичными серверами.

Первичные и вторичные серверы имен

Для обслуживания каждого домена требуется по крайней мере два сервера имен. Только один из них может быть первичным, все остальные – вторичные. *Первичный* (*master*) сервер имен – последняя авторитетная инстанция для домена. Информацию о домене надо изменять на первичном сервере имен, так как именно оттуда эту информацию берет *вторичный* (*slave*) сервер. Предполагается, что оба сервера, и первичный и вторичный, являются авторитетными, то есть окружающий их мир считает, что информация о домене, исходящая от этих серверов, будет верна на все сто процентов.

Один сервер имен может быть первичным для одних доменов и вторичным для других. Например, у домена *absolutefreebsd.com* два сервера имен – *bewilderbeast.blackhelicopters.org*[1] и *ribble.lodden.com*. Сервер

[1] Этот зверь напоминает антилопу гну (wildebeest), но понять его гораздо труднее.

bewilderbeast.blackhelicopters.org является первичным сервером имен для этого домена, а *tribble.lodden.com* – вторичным. Всякий раз, когда я обновляю записи DNS, bewilderbeast посылает серверу tribble извещение о том, что появились новые записи, после чего вторичный сервер обновляет их у себя. Если bewilderbeast подвергнется нападению и исчезнет из Интернета, информацию DNS о домене будет поставлять tribble. В свою очередь сервер tribble является первичным сервером имен для домена *lodden.com*, а bewilderbeast – вторичным для других доменов. Системные администраторы часто обмениваются услугами вторичных серверов имен между собой.

Расширяйте область действия своих серверов имен. Если у вас появится возможность обменяться услугами вторичных серверов имен с кем-то из другого сегмента Интернета, расположенного в другой стране, а еще лучше – на другом континенте, не упускайте такую возможность. Предложения об обмене услугами DNS можно найти на многих общественных веб-сайтах. Самое лучшее при обмене, если оказываемые друг другу услуги будут сопоставимы по объему – не стоит оказывать услуги вторичного сервера имен сотне доменов в обмен на аналогичную услугу для единственного домена вашей компании! Наконец, услуги вторичных серверов DNS, разбросанных по всему свету, недорого предлагают некоторые компании.

Конфигурационные файлы BIND

На первичном и вторичном серверах имен используются одни и те же конфигурационные файлы демона службы имен named(8). В конфигурацию FreeBSD по умолчанию уже входят настройки, необходимые для работы простейшего сервера имен, но чтобы запустить службу имен для собственного домена, вы должны понимать, как она настраивается. Основной каталог с конфигурационными файлами – */etc/namedb*, и здесь вы найдете несколько очень важных файлов.

named.root

Файл *named.root* должен присутствовать обязательно, и редактировать его не надо. В нем перечислены корневые серверы имен. Когда сервер имен получает запрос информации о сайте, которой нет в его кэше, он обращается к корневым серверам имен. Корневые серверы имен идентифицируются по IP-адресам, и все сообщество Интернета стремится изменять эти IP-адреса как можно реже. Поэтому данный файл изменяется нечасто – к моменту написания этих строк (конец 2007 года) последний раз *named.root* обновлялся в конце 2004 года.

localhost-forward.db, localhost-reverse.db

Это файлы прямой и обратной зон для хоста localhost на вашем компьютере. Что такое файлы зон? Читайте дальше, и узнаете.

named.conf

Основу конфигурации сервера DNS составляет конфигурационный файл демона named, *named.conf*. Если в файле *named.conf* есть ошибки, сервер имен будет неработоспособен. Именно здесь хранятся главные настройки DNS.

Настройка BIND с помощью файла named.conf

Содержимое файла *named.conf* напоминает код на языке C. Если вам не знаком язык C, не волнуйтесь. Правила очень просты, а примеры объясняют все, что вам нужно знать. Если строка начинается с двух символов слэша (//), то это комментарий. А любой текст между старыми добрыми знаками комментариев C (/* и */) – это многострочный комментарий. Все остальное в *named.conf* – это либо параметры, либо зоны. *Зона* – это причудливое название домена (строго говоря, они не идентичны, но для наших целей можно сделать такое допущение). *Параметры* управляют работой BIND.

Параметры

Если не принимать во внимание комментарии, файл *named.conf*, поставляемый по умолчанию, начинается со списка параметров. Основная их часть неясна и по умолчанию закомментирована. Для применения параметров их нужно разместить в соответствующем разделе, который ограничен словом *options* и фигурными скобками ({ и }). Сами параметры располагаются между скобками и отделяются друг от друга точкой с запятой. Вот очень простой раздел параметров из файла *named.conf*:

```
options {
        directory       "/etc/namedb";
        pid-file        "/var/run/named/pid";
        dump-file       "/var/dump/named_dump.db";
        statistics-file "/var/stats/named.stats";
        listen-on       { 127.0.0.1; 192.168.0.5; };
};
```

В этом примере значениями большинства параметров являются каталоги или файлы, и только параметр `listen-on` содержит два IP-адреса.

Сначала рассмотрим параметр `directory`. Он задает каталог конфигурации, в котором демон named ищет и хранит файлы DNS. Значение по умолчанию – очень неплохой выбор, особенно если учесть, что каталог */etc/namedb* в действительности таковым не является. (Странно звучит? Читайте дальше, и вскоре все прояснится.)

Параметр `pid-file` – это имя файла, в котором хранится числовой идентификатор основного процесса named. Содержимое этого файла используется различными инструментами администрирования сервера имен.

Параметр `dump-file` — это кэш ответов демона named. Все ответы, отправленные named, сохраняются в кэше на диске. Файл кэша часто используется при поиске неисправностей в DNS.

Если от named требуется, чтобы он сохранял статистику и другие сведения о запросах, эти данные сохраняются в файле `statistics-file`.

Параметр `listen-on` определяет, на каких IP-адресах named принимает запросы. Если за одной сетевой картой закреплены десятки IP-адресов, named можно привязать только к одному адресу. Если в системе применяются клетки, такое решение особенно ценно, поскольку предотвращает неправильную настройку клиентов.

BIND поддерживает намного больше параметров, но эти применяются, пожалуй, чаще остальных. Полный список параметров и описание их применения можно найти в документации BIND на сайте *http://www.isc.org*.

Зоны в named.conf

Не измененный *named.conf* описывает три зоны, или домена, которые сервер имен обслуживает по умолчанию: *корневую зону* (*root zone*), *локальную зону для IPv4* (*IPv4 localhost*) и *локальную зону для IPv6* (*IPv6 localhost*). Изменять зоны не надо, иначе наверняка будут проблемы. Мы только рассмотрим их назначение.

Корневая зона

Сервер имен обращается к корневой зоне, когда у него нет информации о запрашиваемом домене или хосте. Такие запросы перенаправляются корневому серверу имен. Вот запись в *named.conf* для корневой зоны:

```
❶ zone "." {
❷       type hint;
❸       file "named.root";
  };
```

Первый элемент ❶ говорит, для какого домена предназначена эта запись. Точка в кавычках означает, что запись соответствует всему Интернету.

Поле `type` ❷ определяет тип домена. Корневая зона — особая: только она имеет тип `hint`. Все остальные записи могут иметь тип либо `master`, либо `slave`.

Наконец, ключевое слово `file` ❸ сообщает демону named, в каком файле хранится информация для этого домена. Демон обращается к каталогу, указанному в параметре `directory`, находит файл с этим именем и назначает его содержимое данной зоне. Эти файлы будут рассмотрены позже.

Локальные зоны

Помните, в главе 6 говорилось, что каждый компьютер использует «обратную петлю» (loopback) с IP-адресом 127.0.0.1 для обращения к самому себе? Сервер имен должен содержать записи для этого IP-адреса. Без них каждый системный вызов, запрашивающий имя локального хоста, переходил бы в режим ожидания, существенно замедляя работу системы, что раздражало бы пользователей.

Ниже приводится конфигурация для локальной зоны IPv4. В файле *named.conf* она расположена сразу после корневой зоны:

```
❶ zone "0.0.127.IN-ADDR.ARPA" {
❷         type master;
❸         file "master/localhost.rev";
  };
```

Довольно похоже на блок `options` и корневую зону из предыдущего раздела, не правда ли?

Имя зоны ❶ представлено в кавычках после слова `zone`. Поскольку эта зона применяется для обратного DNS, мы видим `IN-ADDR.ARPA`. Если данный IP-адрес записать в обратном порядке, получится фактическая группа IP-адресов 127.0.0.

Поле `type` ❷ указывает, является ли данный сервер первичным или вторичным для этого домена. Каждый сервер имен является первичным для локальных зон.

Наконец, ключевое слово `file` ❸ сообщает серверу имен, в каком файле можно найти информацию об этом домене. Информация об этой зоне находится в подкаталоге каталога конфигурации named в файле */etc/namedb/master/localhost.rev*.

Версия локальной зоны для IPv6 мало чем отличается от этого блока. Если IPv6 у вас не используется, как и на большинстве других серверов в Интернете, ее можно просто закомментировать.

Настройка вторичного сервера имен

Настройка вторичного сервера имен – пожалуй, самая легкая задача при установке DNS. Запись похожа на записи для корневой и локальной зоны. Надо знать имя домена, для которого данный сервер имен будет вторичным, и IP-адрес первичного сервера имен. Ниже приведен пример записи настройки вторичной зоны, очень похожей на корневую и локальные зоны.

```
❶ zone "absolutefreebsd.com" {
❷         type slave;
❸         file "slave/AbsoluteFreeBSD.com.db";
❹         masters {
❺                 198.63.22.8;
          };
  };
```

Запись выглядит очень знакомой. Здесь есть имя домена ❶, тип зоны ❷ и имя файла ❸, в котором хранится информация о домене. Кроме первичного сервера, у которого можно получить информацию о совместно обслуживаемой зоне, не менее важно знать, за какие записи вы действительно отвечаете и какие из них можно получить у первичного сервера. Кроме того, сервер имен должен иметь право на запись в файлы вторичных зон, но не в файлы первичных зон. По традиции имена файлов вторичных зон образуются добавлением расширения *.db* к имени домена. Вопреки ожиданиям эти файлы не являются двоичными файлами базы данных. named(8) создает их, когда вторичный сервер загружает данные о домене с первичного сервера.

Далее следует запись о первичном сервере домена ❹ со списком его IP-адресов ❺. Вторичный сервер запрашивает информацию о домене с первичного сервера через постоянные интервалы времени. Первичный сервер имен должен быть представлен своими IP-адресами; в конце концов, DNS-сервера должен иметь возможность загружать свои записи еще до того, как он узнает IP-адрес чего-либо!

Настройка первичного сервера имен

Конфигурация *named.conf*, необходимая для первичного сервера, даже проще конфигурации для вторичной зоны:

```
❶ zone "absolutefreebsd.com" {
❷         type master;
❸         file "master/absolutefreebsd.com.db";
  };
```

Итак, еще раз: есть имя домена ❶, тип зоны ❷ и имя файла ❸. В отличие от файла описания домена для вторичного сервера имен, данный файл надо создать. Создание этого файла зоны рассматривается ниже в этой главе.

Хранилище файлов зон для первичного и вторичного серверов имен

Если вы управляете крупными серверами имен Интернета, то отвечаете за тысячи доменов и миллионы пользователей. Ошибетесь – и на вас обрушится гнев этих миллионов. Поэтому перед настройкой сотен зон обдумайте, как их разместить.

Всегда отделяйте файлы зон, для которых ваш сервер является первичным, от файлов зон, для которых сервер является вторичным. Конфигурация сервера имен по умолчанию уже включает два таких каталога, но многие начинающие администраторы DNS сваливают все в каталог конфигурации */etc/namedb* и потом горько жалеют об этом. Файлы в каталоге первичного сервера священны, и их резервные копии следует хранить в безопасном месте, желательно в репозитарии RCS (глава 4). Файлы из каталога вторичного сервера тоже не мусор,

но они не требуют такого же бережного отношения и обязательного резервного копирования.

Если вам предстоит обслуживать несколько тысяч доменов, можно задуматься о более глубокой иерархии каталогов с файлами первичных зон. В прошлом я использовал 36 подкаталогов в каталоге первичного сервера – по одному для каждой буквы и цифры.

Безусловно, вы можете организовать иерархию каталогов исходя из своих потребностей, главное – помните, что с этой иерархией вам придется жить. Слишком сложная или глубокая иерархия каталогов может доставить хлопот ничуть не меньше, чем слишком простая.

Файлы зон

Итак, у нас есть конфигурационный файл, который сообщает демону named(8), за какие домены тот отвечает и где расположены файлы с информацией об этих доменах. Но сами эти файлы еще надо создать!

Файлы зон имеют замысловатый синтаксис, поскольку BIND создавали программисты, больше заинтересованные в эффективности, чем в простоте использования. В отличие от некоторых других современных программ (таких как Sendmail), сконфигурировать файлы зон здесь не так уж сложно, хотя кое-что в них заставит пользователя чесать в затылке до появления лысины.

Чтобы изучить работу с файлами зон, ознакомьтесь с представленными примерами. И обнаружив, что скребете затылок, вспомните, что продираетесь сквозь болото первобытного Интернета. Если бы DNS был изобретен сегодня, файлы зон наверняка выглядели бы совсем иначе.[1]

Ниже представлен пример файла обратной зоны для хоста localhost с IP-адресом 127.0.0.1. Давайте исследуем этот файл. Прежде всего, вспомним, что все строки, начинающиеся с точки с запятой, представляют собой комментарии. Тщательно комментируйте файлы зон; впоследствии эта информация поможет вспомнить, почему настройка выполнялась так, а не иначе.

```
❶$TTL   3600
❷@ ❸IN ❹SOA ❺test.blackhelicopters.org.
❻root.test.blackhelicopters.org. (
              ❼20061203    ; Serial
               3600        ; Refresh
               900         ; Retry
               3600000     ; Expire
               3600 )      ; Minimum
```

[1] Прежде всего, если бы файлы зон были изобретены сегодня, они имели бы формат XML. И тогда могло бы потребоваться два промежуточных уровня редактирования и проверки – чтобы упростить внесение изменений, и три уровня синтаксического анализа и постобработки – чтобы увидеть результат.

```
        IN   NS    test.blackhelicopters.org.
1       IN   PTR   localhost.blackhelicopters.org.
```

Инструкция `$TTL` ❶ задает время жизни зоны (в секундах). Это значение определяет, как долго другие серверы будут кэшировать информацию об этой зоне. Данная инструкция позволяет задать любое время жизни зоны по своему усмотрению. 3600 секунд (1 час) – это не очень много. Хорошее среднее значение – 10 800 секунд (3 часа). Выбор TTL сродни черной магии; значение по умолчанию подойдет в большинстве случаев.

Следующая запись – *Start of Authority* (SOA, *начало авторитетности*). В ней представлено краткое описание зоны – каково ее поведение и как серверам следует себя с ней вести. У каждой зоны есть только одна запись SOA. В нее входит не информация о домене, а лишь сведения о продолжительности ее хранения.

Символ `@` ❷, с которого начинается SOA-запись, – это специальное сокращение для «зоны, указанной в *named.conf*». В данном случае *named.conf* сообщает, что в этом файле хранятся данные для зоны 0.0.127.in-addr.arpa. Когда named считывает *named.conf* и загружает его содержимое в память, он выполняет такую замену. Фактическое доменное имя было бы более понятным для новичков, однако это препятствовало бы возможности определять несколько зон в одном файле. Вместо символа `@` можно применять полное доменное имя, но почти никто так не поступает.

`IN` представляет тип данных ❸ – данные Интернета. `SOA` ❹ означает запись Start of Authority. Оба элемента присутствуют в каждом создаваемом файле зоны.

Следующая часть – имя машины ❺, на которой находится мастер-файл. Этот файл создан на *test.blackhelicopters.org*.

Далее следует адрес электронной почты ❻ человека, ответственного за эту зону. Сценарий make-localhost по умолчанию задает адрес учетной записи root на локальной машине. В нем отсутствует символ `@`, потому что он соответствует названию зоны, указанному в *named.conf*.[1] Если оставить `@` в адресе электронной почты, то адрес превратился бы в *root0.0.127.in-addr.arpa.test.blackhelicopters.org*, что не только плохо смотрится, но и является ошибкой. Первую точку в адресе электронной почты следует рассматривать как заменитель символа `@`.

В большинстве случаев на сервере имен нет почтового сервера. Согласно установившейся в Интернете практике, в поле электронного адреса поставьте имя пользователя учетной записи администратора и свое доменное имя, например *hostmaster@absolutefreebsd.com*. Предполагается, что у каждого домена есть электронный адрес администратора, предназначенный для ответа на вопросы, связанные с DNS.

---

[1] DNS был создан еще до того, как знак `@` стал широко применяться в электронных адресах. Такое перекрытие – порок электронной почты, а не BIND.

Обратите внимание на круглые скобки в начале и в конце записи SOA. Формально запись SOA должна располагаться в одной строке. Однако тогда ее было бы трудно читать. Поэтому в стандартных файлах зон эта запись разбита на несколько строк. Открывающая круглая скобка указывает на обрыв строки. Каждая строка до закрывающей круглой скобки считается частью записи SOA. Традиционно круглые скобки включают в себя набор таймеров и порядковый номер ❼.

Первое число – это *порядковый номер* (*serial number*), который обозначает версию файла зоны. Всякий раз, когда вы редактируете файл, увеличивайте и порядковый номер. Порядковый номер может быть любым, однако удобнее всего применять дату. Обычно она записывается в формате *YYYYMMDD* с двумя дополнительными цифрами в конце. Порядковый номер 20061203 в листинге означает 3 декабря 2006 года. Две дополнительные цифры сообщают, сколько раз файл изменялся в течение дня. Мне случалось обновлять информацию о домене до десяти раз в день.

Предположим, я создаю файл зоны 9 мая 2008 года и задаю порядковый номер 2008050901. Если я изменю файл зоны 8 июня, порядковый номер изменится на 2008060801. Если я изменю файл зоны второй раз в этот же день, порядковый номер будет 2008060802. Такая система допускает до 100 изменений в день – примерно одно изменение каждые 15 минут. Если окажется, что этого недостаточно, то надо переосмыслить рабочий процесс.

Порядковый номер важен, поскольку время от времени вторичный сервер связывается с первичным и смотрит, не обновилась ли зона. Если порядковый номер на первичном сервере больше, чем на вторичном, вторичный сервер решит, что файл зоны был обновлен, и загрузит новейшую информацию о домене.

Следующее число – значение интервала *обновления* (*refresh*) в секундах. Это число определяет, как часто вторичные серверы связываются с первичным и проверяют, не обновился ли мастер-файл. В этом примере файла *localhost.rev* вторичный сервер имен будет обращаться за обновленным мастер-файлом каждые 3600 секунд (1 час).

Если вторичный сервер не может сравнить свои данные с данными на первичном сервере, в ответ на запросы он выдает текущую информацию

Порядочная проблема

Если вторичные серверы имен не обновили свои файлы зон с первичного сервера имен, причиной могут быть неполадки с порядковым номером. Даже если вы ручаетесь, что увеличили порядковый номер, увеличьте еще раз и посмотрите, что получится. Возможно, все встанет на свои места.

из файла зоны – в конце концов, для этого и предназначен резервный сервер! Данный механизм будет подробно рассмотрен в разделе «Refresh, retry и expire на практике» этой главы.

Значение *retry* (повторная попытка) определяет, как часто вторичный сервер должен выполнять попытки связаться с первичным сервером в случае его недостижимости. В данном примере этот интервал составляет 900 секунд (15 минут). Если вторичный сервер имен не сможет обновить свои данные через час, он будет пытаться сделать это каждые 15 минут, пока первичный сервер имен не отзовется.

Значение *expire* (истечение срока действия) определяет, когда вторичный сервер должен уничтожить записи в кэше. По мнению администратора, в данных обстоятельствах наличие недействительной информации хуже ее отсутствия. В нашем примере период времени составляет 3 600 000 секунд (1000 часов, или чуть более 41 дня).

Последнее число – *минимальное время жизни* (*minimum time-to-live*). В старых реализациях BIND это значение определяло время жизни буквально всей информации. Сейчас оно означает только TTL для отрицательных ответов. Например, если вы ищете *givememymoneyback.absolutefreebsd.com*, ваш сервер имен узнает, что такого хоста не существует. В результате он будет «помнить» об этом факте в течение времени, равного минимальному времени жизни. Не забудьте поставить закрывающую круглую скобку после значения времени жизни, в противном случае named предположит, что остаток файла также представляет собой часть SOA-записи, и не только будет сбит с толку, но и поведет себя весьма агрессивно.

Теперь, получив полное представление о записи SOA, вы сможете указать действительную информацию о своем домене. В нашем примере информацию о фактических хостах зоны содержат следующие строки:

```
        IN    ❷NS     test.blackhelicopters.org.
❶1      IN    ❸PTR    localhost.blackhelicopters.org.
```

Каждая строка состоит из четырех частей: имя хоста или число, тип данных, тип сервера и фактические данные. Первое поле может быть пустым либо содержать имя хоста (например, *www*) или число (например, 12). Имя зоны автоматически присоединяется в начало либо в конец этой записи в зависимости от того, для чего предназначен этот файл – для обратного или прямого DNS, соответственно. Поскольку наш пример предназначен для обратного DNS, цифра 1 ❶ добавляется к 127.0.0. В результате получается IP-адрес 127.0.0.1.

Если в первом поле ничего нет, named так или иначе добавит имя зоны. В результате получится разумное значение по умолчанию. Например, первое поле строки NS, представленной в примере, ничем не заполнено, поэтому named считает, что это зона 0.0.127.in-addr.arpa или сеть, начинающаяся с 127.0.0, то есть домен, указанный для этого файла в *named.conf*.

Тип данных DNS – всегда `IN` (Интернет).

Третье поле (тип сервера) особенно интересно. NS-запись ❷ представляет сервер имен. В нашем примере единственным сервером имен для данного домена является *test.blackhelicopters.org*. Если вы размещаете *localhost.rev* на нескольких серверах имен, в них надо добавить дополнительные строки `NS`. С другой стороны, PTR-запись ❸ представляет отображение IP-адреса в имя хоста. Эта зона – для сети 127.0.0., а хост .1 в данной сети – это *localhost.blackhelicopters.org*.

Refresh, Retry и Expire на практике

Уже говорилось о том, что означают параметры Refresh, Retry и Expire, – теперь мы посмотрим, как они работают, на примере.

Допустим, у нас есть домен с периодом Refresh, равным 4 часам, периодом Retry, равным 1 часу, и периодом Expire, равным 48 часам. Это означает, что вторичный сервер имен связывается с первичным каждые четыре часа и проверяет наличие обновлений. Время от времени вы редактируете записи на первичном сервере имен, и в течение ближайших четырех часов они передаются на вторичный сервер. Пока что все хорошо. Теперь предположим, что первичный сервер имен взорвался, и осколки его жесткого диска разлетелись по всей округе. Что произойдет со вторичным сервером и с клиентами, которым необходима информация о домене?

С клиентами проще всего. Поскольку первичный сервер имен недостижим, удаленные серверы имен начнут пользоваться услугами вторичного сервера и клиенты получат необходимую информацию.

В следующий раз, когда вторичный сервер попытается заполучить обновленные записи, он не сможет связаться с первичным сервером. В этот момент он изменяет периодичность своих обращений за обновлениями. Он начнет проверять наличие обновлений каждый час, пока не будет восстановлена работоспособность первичного сервера.

Если вторичный сервер не сможет подтвердить свои данные в течение периода Expire, информация о домене будет признана устаревшей и ненужной. В этом случае вторичный сервер сотрет у себя всю информацию о домене и будет возвращать ошибку в ответ на любой запрос об этом домене. Домен исчезнет из Сети.

Действия в случае аварий DNS

Если с вашим первичным сервером имен и в самом деле произойдет катастрофа, у вас есть время, равное сумме значений Expire и Retry, чтобы заменить данный сервер или переконфигурировать вторичный сервер, сделав его первичным.

Пример реальной зоны

Файл локальной зоны отчасти надуман – он представляет только одну машину с одним IP-адресом. Однако он удобен, его можно найти на любом сервере имен, а данные, содержащиеся в нем, либо широко используются, либо совершенно необходимы. Теперь рассмотрим файл зоны, который более полно представляет домены, находящиеся на обслуживании, – файл зоны для домена *absolutefreebsd.com*.

```
$TTL ❶1h
@ IN ❷SOA blackhelicopters.org. hostmaster.blackhelicopters.org. (
                                2006121002      ; Serial
                                1d       ; Refresh
                                2h       ; Retry
                                1000h    ; Expire
                                2d )     ; Minimum
        IN   NS        ❸bewilderbeast.blackhelicopters.org.
        IN   NS         tribble.lodden.com.
        IN  ❹MX 20      mail.nobletechnology.net.
        IN   MX 10      bewilderbeast.blackhelicopters.org.
        IN  ❺A          198.22.63.8
www     IN  ❻CNAME      absolutefreebsd.com.
```

Данный файл похож на файл *localhost.rev*, обсуждавшийся выше, однако это файл реально существующей зоны для настоящего домена Интернета. Посмотрим, что в нем есть. Во-первых, есть время жизни ❶. Его значение равно одному часу – когда сервер имен получает информацию для этого домена, он хранит ее один час. SOA-запись ❷ содержит контактную информацию и значения времени Refresh, Retry и Expire, а также порядковый номер.

В файле зоны перечислены два сервера имен: *bewilderbeast.blackhelicopters.org* и *tribble.lodden.com* ❸. Согласно значениям времени в файле зоны, *tribble.lodden.com*, будет сравнивать свои записи с записями *bewilderbeast.blackhelicopters.org* каждый день. Если попытка сравнить записи завершится неудачей, то сравнение будет осуществляться каждые 2 часа. Если *tribble.lodden.com* не может сравнить свои записи с записями первичного сервера в течение 1000 часов, он перестает отвечать на запросы о домене *absolutefreebsd.com*. Наконец, удаленные серверы имен будут кэшировать ответы «нет такого хоста» в течение двух дней.

Обратите внимание: устаревшие версии серверов DNS могли принимать значения времени только в секундах. Теперь вместо числа `86400` для определения интервала длительностью в одни сутки можно использовать значение `1d`. Интервалы в секундах до сих пор можно встретить в старых файлах зон, но в наше время заниматься такими математическими вычислениями – совершенно лишнее.

Почтовый ретранслятор

Далее представлен новый тип записи, `MX` ❹ – почтовый ретранслятор (mail exchanger) домена. Хотя у домена есть только один первичный почтовый хост, у него может быть несколько резервных почтовых серверов. Тем не менее почта должна обязательно дойти до основного почтового хоста. В этих записях указываются предпочтительный почтовый и резервные серверы. Эта тема подробно обсуждается в главе 16.

Дополнительное поле в записи `MX` – это приоритет (preference), числа `10` и `20`. Серверы с меньшим приоритетом считаются более предпочтительными. В нашем случае сервер *bewilderbeast.blackhelicopters.org* с приоритетом 10 является предпочтительным почтовым сервером для *absolutefreebsd.com*. Если первичный сервер недоступен, *mail.noble-technology.net* выступит в качестве резервного сервера.

Когда-нибудь вы, возможно, захотите добавить еще один почтовый сервер к двум уже имеющимся или изменить предпочтительный сервер, поэтому между приоритетами необходимо предусмотреть определенный зазор. Если поступить иначе и нумеровать их по порядку (1, 2, 3 и т. д.), то такая схема окажется менее гибкой.

Записи о хостах

Наконец, у нас есть записи о каждом хосте в домене. Наиболее часто используются записи о хостах двух типов: `CNAME` и `A`. Мы уже говорили при рассмотрении программы dig(1), что `CNAME` – это *псевдоним* (точнее, ссылка на *каноническое имя*). Запись `A` ❺ указывает IP-адрес. В нашем примере программа dig(1) покажет, что хост *absolutefreebsd.com* – это псевдоним для *www.absolutefreebsd.com*. (Напомню, что в отсутствие явно указанного имени запись по умолчанию относится к домену, представленному этим файлом!) Хост *www.absolutefreebsd.com* имеет IP-адрес ❻ 198.22.63.8.

Почтовые ретрансляторы и серверы имен нельзя определять записями типа `CNAME`; для них следует указывать фактические имена хостов.

Точки, окончание и файлы зон

Демон named(8) предполагает, что все имена хостов в файле зоны являются частью этой зоны. В файле зоны для *absolutefreebsd.com* нельзя указать имя *www.absolutefreebsd.com*; named уже знает, что речь идет о домене *absolutefreebsd.com*, поэтому достаточно просто указать *www.*, а named(8) добавит имя зоны к имени каждого хоста, если явно не указать обратное. Это довольно удобно, за исключением случаев, когда хост не является частью домена. Например, ни один из серверов имен, обслуживающих домен *absolutefreebsd.com*, не входит в этот домен. Я ведь не хочу, чтобы мой первичный сервер имен отображался как *bewilderbeast.blackhelicopters.org.absolutefreebsd.com*.

Вы уже видели, что при создании зон в SOA-записях символ точки может подставляться вместо символа @ в адресах электронной почты. Однако точки также играют роль символов окончания для имен хостов. Если после имени хоста добавить символ точки, демон named(8) будет считать, что все имена хостов являются частью зоны, для которой предназначен этот файл. Как видно из предыдущих примеров, каждое полное имя хоста, расположенное после записи SOA, заканчивается точкой. Даже в записи CNAME имя хоста *www.absolutefreebsd.com* заканчивается точкой. Если бы точка отсутствовала, то этот псевдоним относился бы к хосту *www.absolutefreebsd.com.absolutefreebsd.com*. Я бы не хотел вводить такое имя в адресной строке веб-броузера, чтобы получить нужную мне веб-страницу! (Замечу, что наличие такой вещи, как фактическое имя хоста, фанаты DNS находят забавным. Вспомните об этом, прежде чем превратиться в такого фаната.)

Обратные зоны DNS

Для каждого блока IP-адресов, находящегося в вашем распоряжении, требуется файл обратной зоны DNS, где должны быть перечислены все имена хостов с соответствующими им IP-адресами. Каждой записи типа A должна соответствовать запись PTR. Это своего рода двойная бухгалтерия, может быть и полезная в учете финансов, но весьма утомительная для системных администраторов.

Я настоятельно рекомендую использовать сценарий mkrdns для автоматизации администрирования обратных зон DNS. Это позволит сэкономить время и уменьшит число ошибок. Сценарий mkrdns написан на языке Perl, поэтому он настолько прост, что для него даже не требуется отдельный «порт». Поищите сценарий mkrdns в какой-нибудь поисковой системе, загрузите его и положите в каталог */usr/local/bin*.

Чтобы сценарий mkrdns смог создать файл обратных зон, у вас уже должен иметься файл зоны. Этой зоне должна соответствовать корректная запись типа SOA, не обязательно содержащая информацию о хостах. mkrdns использует SOA-запись в качестве шаблона при создании файла обратной зоны. Сценарий запускают так:

```
# mkrdns /etc/namedb/named.conf
```

Когда сценарий закончит работу, вы увидите, что файлы обратных зон DNS автоматически обновились. Чтобы эти изменения вступили в силу, достаточно просто перезагрузить сервер имен. Как, мы еще не рассматривали вопрос перезагрузки сервера имен? Давайте сделаем это прямо сейчас.

Управление демоном named

При работе с сервером имен надо автоматически запускать демон named во время начальной загрузки. Для этого предназначен параметр named_enable файла *rc.conf*:

```
named_enable="YES"
```

Такая установка разрешает запуск сценария начальной загрузки */etc/rc.d/named*. С помощью этого сценария можно запускать и останавливать named(8) вручную, как рассказывалось в главе 3. Всегда используйте команды `/etc/rc.d/named stop` и `/etc/rc.d/named start` для останова и запуска демона named.

Однако когда named уже запущен, может появиться необходимость перезапустить его по каким-либо причинам, в том числе и для того, чтобы сервер имен проверил наличие обновлений в файлах зон. Для этих целей пакет BIND включает в себя инструмент удаленного управления сервером имен (Remote Name Daemon Control), rndc(8).

Настройка rndc

rndc взаимодействует с демоном named(8) через безопасное TCP-соединение, даже на локальном компьютере. Это означает, что у вас есть возможность управлять серверами имен на удаленных системах, не выполняя вход в них. Однако для создания безопасного соединения необходим ключ. Пакет BIND содержит сценарий rndc-confgen(8), который генерирует ключи и выполняет рутинные задачи по настройке rndc. Чтобы создать конфигурационный файл для rndc, достаточно запустить команду `rndc-confgen`:

```
# rndc-confgen > /etc/namedb/rndc.conf
```

Полученный файл включает в себя все необходимые параметры настройки rndc(8), а также ряд параметров настройки для включения в файл *named.conf*. Раздел с настройками для *named.conf* закомментирован символами решетки (#) и выглядит примерно так:

```
# Use with the following in named.conf, adjusting the allow list as needed:
# key "rndc-key" {
#     algorithm hmac-md5;
#     ❶secret "VO5IU1GnxQTlfTxALgciCw==";
# };
#
# controls {
#     ❷inet 127.0.0.1 port 953
#         ❸allow { 127.0.0.1; } keys { "rndc-key"; };
# };
# End of named.conf
```

Самая интересная часть – это ключ ❶, необходимый для обеспечения возможности управления сервером имен. Ключи в *rndc.conf* и *named.conf* должны совпадать, в противном случае rndc не сможет подключиться к серверу. Здесь также указываются IP-адрес и порт TCP, на котором named ожидает управляющие соединения ❷, далее выдается разрешение на управление с указанного IP-адреса при наличии указанного ключа ❸. Есть возможность выдать разрешение на управление

сервером имен нескольким клиентам и каждому клиенту выдать свой собственный ключ. За подробностями обращайтесь к страницам руководства rndc(8) и rndc.conf(5).

Скопируйте закомментированные строки из файла *rndc.conf* в файл *named.conf*, удалите символы решетки и перезапустите демон named командой `/etc/rc.d/named restart`. После этого вы сможете управлять сервером имен с помощью rndc.

Работа с rndc

Теперь, когда все препоны установки и настройки rndc позади, – что еще можно сказать об этой программе? Полный список ее возможностей приведен на странице руководства rndc(8). Наиболее типичные задачи, решаемые с ее помощью: перезапуск сервера имен, повторная загрузка зон, обновление зон, перенастройка named и проверка состояния демона.

Перезагрузка сервера имен командой `rndc reload` вынуждает named(8) повторно прочитать и обработать конфигурационные файлы, загрузить из текстовых файлов информацию о первичных доменах и проверить наличие обновлений для вторичных доменов. Эта операция обычно выполняется при изменении конфигурационных файлов и файлов зон.

В промежутках между перезагрузками и полным перезапуском сервера имен с помощью команды `rndc reconfig` можно заставить named проверить наличие новых зон.

Для *повторной загрузки зоны* без перезагрузки всего сервера имен в команде `rndc reload` следует указать имя домена. Обычно такая операция выполняется в случае, когда сервер имен испытывает серьезную нагрузку и повторная перезагрузка может потребовать слишком длительного времени или вызвать неприятности в сети.

Выполнение *операции обновления зоны* вынуждает вторичный сервер немедленно проверить наличие обновлений для указанной зоны на первичном сервере. Например, заставить вторичный сервер *absolutefreebsd.com* проверить наличие обновлений на первичном сервере можно с помощью команды `rndc refresh absolutefreebsd.com`.

Программа rndc не дает возможности перезапустить named. Для полного перезапуска демона следует применить команду `/etc/rc.d/named restart`.

С помощью команды `rndc status` можно получить разнообразную информацию о сервере имен, включая количество обслуживаемых им доменов, число клиентов, обратившихся к серверу, и т. д.

Проверка DNS

Ошибки в конфигурации DNS появляются в */var/log/messages*. В виде сообщений об ошибках появляются и уведомления о запуске, повтор-

ном запуске и перезагрузке named. Если сервер имен не обслуживает информацию о домене, проверьте файл протокола. Сообщения, содержащиеся в нем, обычно достаточно ясны; в них представлен номер строки, которая, возможно, послужила причиной ошибки.

Создав первую зону, выведите всю информацию о домене и проверьте свою работу. Ключевое слово `axfr` программы dig запрашивает список всех хостов в домене:

```
# dig имя_домена @первичный_сервер axfr
```

Изучите результаты. Все ли имена представлены, как ожидалось? Есть ли хосты с двойными доменными именами, например *www.absolutefreebsd.com.absolutefreebsd.com*? Если да, значит вы забыли точку. На месте ли все почтовые серверы и серверы имен? Если нет, внесите исправления.

Для дополнительной проверки можно применить dnswalk(1) (*/usr/ports/net/dnswalk*). Этот инструмент выявляет различные неполадки в конфигурации, однако не способен обнаружить концептуальные проблемы. Если для определенного хоста есть `CNAME`, но каноническое имя представляет собой значение `CNAME`, указывающее на первый хост, образуя циклическую ссылку, dnswalk отловит такую ошибку. Однако если предпочтительный почтовый ретранслятор задан как *mail.whitehouse.gov*, dnswalk это пропустит. Запустить dnswalk можно так:

```
# dnswalk absolutefreebsd.com.
```

Обратите внимание на завершающую точку.

Защита сервера имен

BIND – притягательная мишень для злоумышленников, поскольку предоставляет много информации о сети. Даже если атакующий не сможет получить контроль над вашей машиной, он проявит особый интерес к полученной информации, особенно если ваши хосты имеют наглядные имена, такие как «accounting» или «finance». Кроме того, уже многие годы named(8) работает с привилегиями root. Если кому-то удастся взломать named, он сможет полностью овладеть вашей машиной. Обе эти проблемы мы рассмотрим по отдельности.

Управление передачей информации о зонах

Пример с dig, в котором мы получили весь список хостов в домене, описывает *передачу зоны* (*zone transfer*). Такого рода запросы используются вторичными серверами имен для обновления своих записей о домене. Предполагаемый взломщик проявит особый интерес к полному списку хостов. Назначение сервера имен состоит в том, чтобы обслуживать имена, поэтому он не может полностью прикрыть доступ взломщиков к машине. Однако named можно настроить так, чтобы он

давал ответы только на особые запросы, а не выставлял напоказ свое содержимое. Таким образом, если кто-нибудь выдаст запрос о конкретном хосте, сервер имен ответит. Если же будет запрошен весь список хостов, запрос будет отклонен.

Чтобы ограничить передачу зоны узким кругом хостов, можно использовать параметр `allow-transfer`:

```
options {
    allow-transfer {
        127.0.0.1; 192.168.8.3;
    };
};
```

В данном примере хосты с адресами 127.0.0.1 (локальный компьютер) и 192.168.8.3 – единственные системы, которым позволено выполнять передачу зоны. Замените эти IP-адреса на адреса ваших вторичных серверов имен и своей рабочей станции – и значительная часть информации о вашей сети будет скрыта от посторонних глаз. В список можно добавить и настольные машины, чтобы они также могли выполнять передачу зоны при отладке DNS.

При необходимости named(8) позволяет еще больше ограничить возможность доступа, передавая определенным клиентам информацию только об отдельных зонах. За дополнительной информацией обращайтесь к документации BIND в каталоге */usr/src/contrib/bind/doc*.

Защита named(8)

А как насчет хакеров, атакующих сам демон named(8) с целью получить доступ к командной строке? Можно запустить named в клетке (глава 9), чтобы удачливый взломщик не смог получить доступ больше ни к чему. По умолчанию BIND работает в chroot-окружении. В данном случае использование chroot-окружения предпочтительнее, так как оно включает в себя только системные ресурсы, необходимые для обеспечения работы named, и ничего больше. chroot-окружение можно представить себе как непривилегированную упрощенную клетку. И хотя этот режим работы используется по умолчанию, вы должны знать, как он работает.

В этом окружении корневым каталогом для named(8) является */var/named*. Если заглянуть в него, можно увидеть подкаталоги *dev*, *etc* и *var*. Запущенный в chroot-окружении named(8) полагает, что */var/named* – это корневой каталог /, а все указанные выше каталоги – это */dev*, */etc* и */var*. Программа обладает доступом только к этим подкаталогам. Обратите внимание на отсутствие таких каталогов, как */bin* или */sbin*. Злоумышленник не сможет получить доступ к командной оболочке или другим программам просто потому, что эти программы отсутствуют!

Но, минутку... Если named(8) заперт в каталоге */var/named*, как тогда он читает файлы конфигурации в */etc/namedb*? Присмотритесь к */etc/namedb* – это всего лишь символическая ссылка на каталог */var/named/etc/namedb*. Каталог конфигурации носит имя */etc/namedb* так долго, что теперь ни в одной операционной системе не рискуют удалить эту ссылку.

Дополнительная информация о BIND

По мере расширения сети вам понадобится дополнительная информация о BIND. Некоторые расширенные возможности, такие как обзоры, будут весьма кстати при расширении сети. Хорошим источником является документация в */usr/src/contrib/bind9/*, особенно подкаталог *doc*. Стандартное руководство по BIND – книга Крикета Ли (Cricket Liu) и Пола Альбитца (Paul Albitz) «DNS and BIND»[1], 5-е издание (O'Reilly Media, 2006). Настоятельно рекомендую ее, так как она содержит массу дополнительных сведений о BIND.

[1] Крикет Ли и Пол Альбитц «DNS и BIND», 5-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2008.

15

Управление малыми системными сервисами

Даже серверу очень узкого, специализированного назначения (например, веб-серверу) необходимо множество меньших, «вспомогательных» сервисов для решения задач администрирования. В этой главе будут рассмотрены некоторые из этих сервисов, такие как сервер времени, DHCP, планирование заданий и т. д. Начнем с обеспечения безопасности удаленных соединений с вашим сервером FreeBSD с помощью SSH.

SSH

Одна из самых сильных сторон UNIX – простота удаленного администрирования. Неважно, где находится сервер – перед вами или в подвале запертой лаборатории посреди совершенно секретного военного оборудования, охраняемого свирепыми собаками и бешеными горностаями. Если к нему есть доступ по сети, то им можно управлять, как если бы он находился прямо перед вами.

Многие годы стандартным средством для доступа к удаленному серверу был telnet(1). telnet хорош. С его помощью можно соединиться с произвольным TCP-портом системы и непосредственно «разговаривать» с сервером по сети. (Далее в этой главе такая возможность будет применена для тестирования различных сервисов.) Однако telnet, как протоколу удаленного администрирования, сопутствует одна серьезнейшая проблема: в большинстве его реализаций все данные, посылаемые по этому протоколу, передаются в нешифрованном виде. Любой владелец анализатора пакетов (packet sniffer), прослушивающий ваше соединение, может заполучить ваши имя пользователя, пароль и любую другую информацию, которая передается в рамках сеанса связи. При использовании telnet даже самая лучшая схема выбора паролей в мире не сможет защитить ваши имя пользователя и пароль. Злоумышлен-

ники размещают анализаторы пакетов повсюду; я встречал их в небольших локальных сетях, в юридических фирмах, где работают с документами государственной важности, в домашних персональных компьютерах и в магистралях Интернета. Единственная защита от анализаторов пакетов – управлять соединениями так, чтобы злоумышленники не смогли извлечь секретную информацию. Именно здесь на помощь приходит SSH, или secure shell.

В значительной степени SSH работает так же, как и telnet: он предоставляет отлично конфигурируемое терминальное окно на удаленном хосте. Однако, в отличие от telnet, все данные, посылаемые по этому протоколу, шифруются. SSH обеспечивает не только защиту паролей от анализатора, но и шифрование набираемых команд и их вывода. Хотя у telnet есть несколько преимуществ над SSH (он требует меньше процессорного времени и проще в настройке), его преимущества нивелируются преимуществами SSH в обеспечении безопасности. Кроме того, SSH обладает целым рядом таких особенностей, как возможность создавать шифруемые туннели для любого протокола. Как и telnet, SSH может работать на всех современных версиях UNIX и даже в Microsoft Windows.

При использовании SSH шифрование и аутентификация удаленных соединений производятся с помощью открытых ключей, о чем уже рассказывалось в главе 9. Демон SSH предлагает клиентам использовать открытый ключ сервера и сохраняет свой закрытый ключ. Клиент и сервер используют ключ, чтобы договориться об организации шифруемого канала между ними. Поскольку для совершения этого действия необходимы оба ключа, и открытый, и закрытый, ваши данные оказываются в безопасности; даже если кто-то перехватит SSH-трафик, он сможет увидеть только зашифрованную абракадабру.

Чтобы использовать SSH на машине, работающей под управлением FreeBSD, должен быть запущен сервер SSH, а на рабочей станции – клиент SSH.

Сервер SSH: sshd(8)

Демон sshd(8) принимает запросы SSH, поступающие из сети, на порт TCP с номером 22. Чтобы разрешить запуск sshd во время загрузки, добавьте в файл */etc/rc.conf* следующую строку:

```
sshd_enable="YES"
```

Выполнив эту настройку, можно использовать сценарий */etc/rc.d/sshd* для запуска и останова SSH. Останов демона SSH не вызывает прекращение сеансов связи SSH, которые к этому моменту уже запущены; эта команда только лишь воспрепятствует установлению новых соединений.

В отличие от других протоколов, рассматриваемых в этой книге, SSH трудно тестировать вручную. Можно лишь убедиться, что демон sshd

запущен. Для этого надо с помощью telnet подключиться к TCP-порту, на котором предположительно работает SSH.

```
# telnet localhost 22
❶ Trying ::1...
❷ Trying 127.0.0.1...
❸ Connected to localhost.
Escape character is '^]'.
❹ SSH-2.0-OpenSSH_4.4p1 FreeBSD-20060930
```

Telnet сначала предпринимает попытку подключиться с использованием адреса IPv6 локального компьютера ❶, затем с использованием адреса IPv4 ❷ и преуспевает в этом. Попытка подключения увенчалась успехом, и мы видим, что демон, прослушивающий этот порт, сам себя называет как SSH версии 2, реализует OpenSSH 4.4p1 в системе FreeBSD, версии 20060930 ❸. (Протокол SSH имеет две версии, 1 и 2. *Всегда* используйте версию 2.) Всю эту информацию можно получить с помощью простой команды telnet, но это все, что sshd предлагает без ограничений. Если вы не в состоянии шифровать пакеты вручную, «на лету», то дальше вам не удастся продвинуться ни на шаг. Нажмите комбинацию клавиш Ctrl-], чтобы закрыть соединение, а затем Ctrl-C, чтобы прервать работу telnet и вернуться в командную строку.

Ключи SSH и их цифровые отпечатки

При первом запуске sshd(8) программа обнаружит отсутствие ключей шифрования и автоматически создаст их. Если система только что загрузилась, вам будет предложено понажимать клавиши на клавиатуре, чтобы добавить элемент случайности. В процессе инициализации sshd создаст три пары ключей.

SSH версии 1 использует файлы ключей */etc/ssh/ssh_host_key*, */etc/ssh/ssh_host_key.pub*, */etc/ssh/ssh_host_rsa_key* и */etc/ssh/ssh_host_rsa_key.pub*. SSH версии 1 не гарантирует полную безопасность, но он намного безопаснее, чем telnet. SSH версии 2 наряду с ключами RSA использует ключи DSA */etc/ssh/ssh_host_dsa_key* и */etc/ssh/ssh_host_dsa_key.pub*.

Файлы с расширением *.pub* содержат открытые ключи (public keys). Это те самые ключи, которые sshd передает клиентам, пытающимся установить соединение. Это дает клиентам возможность убедиться, что сервер, к которому они пытаются подключиться, действительно является тем сервером, за который он себя выдает. (В прошлом злоумышленникам удавалось обманывать пользователей, подставляя им поддельные машины, чтобы перехватить имена их учетных записей и пароли.) Загляните в один из файлов с открытыми ключами; он довольно велик. Если пользователю предложить убедиться в том, что сервер предлагает правильный ключ, то его длина помешает проверить все символы до единого даже самым одержимым пользователям.

К счастью, SSH позволяет сгенерировать *цифровой отпечаток ключа*, который намного короче соответствующего ему ключа. Обладая цифровым отпечатком, вы не сможете шифровать трафик или договориться о параметрах подключения, но вероятность того, что два независимых ключа будут иметь одинаковый отпечаток, крайне низка. Чтобы сгенерировать отпечаток открытого ключа, следует выполнить команду `ssh-keygen -lf` *`файл_ключа.pub`*.

```
# ssh-keygen -lf /etc/ssh/ssh_host_dsa_key.pub
1024 31:28:4b:6e:aa:23:63:2e:9a:6b:44:00:9f:fd:28:21
/etc/ssh/ssh_host_dsa_key.pub
```

Первое число, `1024`, показывает число битов в ключе. Число 1024 было стандартным в 2007 году, однако я полагаю, что в ближайшие несколько лет с ростом мощности компьютеров оно увеличится до 2048. Шестнадцатеричная строка, начинающаяся с `31` и заканчивающаяся на `21`, – это цифровой отпечаток открытого ключа. Строка достаточно длинная, тем не менее она намного короче самого ключа. Скопируйте этот отпечаток с оригинального сервера и поместите его куда-нибудь, где он будет доступен клиентам, – на веб-страницу или на лист бумаги. Используйте этот ключ, чтобы убедиться в подлинности сервера при первом подключении к нему.

Конфигурирование демона SSH

Даже при том, что sshd распространяется со вполне приемлемой конфигурацией, по мере изучения всех особенностей sshd(8) у вас может возникнуть желание подрегулировать некоторые настройки. Конфигурационный файл */etc/ssh/sshd_config* содержит все параметры настройки со значениями по умолчанию, закомментированные с помощью символа решетки (`#`). Если вам потребуется изменить значение выбранного параметра, раскомментируйте строку и измените значение.

Мы не будем рассматривать все имеющиеся параметры настройки sshd; для этого потребовалось бы написать отдельную книгу. Более того, OpenSSH продолжает развиваться настолько быстро, что такая книга устаревала бы, еще не достигнув книжной полки. Поэтому мы сконцентрируемся на изменениях, выполняемых администраторами наиболее часто.

После изменения конфигурации демона SSH его следует перезапустить командой `/etc/rc.d/sshd restart`.

VersionAddendum FreeBSD-20061110

Значение параметра `VersionAddendum` появляется в имени сервера при попытке подключиться к порту TCP демона sshd. Некоторые администраторы рекомендуют изменить это значение, чтобы скрыть версию операционной системы. Однако операционную систему легко идентифицировать с помощью определенных приемов, выполняя обмен паке-

тами с хостом, поэтому изменение данного параметра не стоит потраченного на это времени. (С другой стороны, стоит сменить имя параметра `VersionAddendum` на `DrunkenBadgerSoftware`, хотя бы ради смеха.)

Port 22

По умолчанию sshd(8) прослушивает порт TCP с номером 22. При желании вы можете указать другой, нестандартный порт. Если нужно, чтобы sshd прослушивал несколько портов (например, в дополнение к порту 22 еще и порт 443, чтобы обойти ошибки в настройке брандмауэра), можно добавить дополнительные определения параметра `Port` в отдельных строках:

```
Port 22
Port 443
```

Protocol 2

Любая современная версия SSH по умолчанию поддерживает только версию 2 протокола SSH. Протокол SSH версии 1 не обеспечивает должный уровень безопасности, а клиенты с поддержкой протокола SSH версии 2 теперь входят в состав наиболее распространенных систем. Однако демон sshd(8) до сих пор обладает поддержкой протокола версии 1, и при необходимости можно разрешить эту поддержку, хотя она и не обеспечивает достаточно высокий уровень безопасности. Перечислите в порядке предпочтений поддерживаемые версии протоколов, разделяя их запятыми.

ListenAddress 0.0.0.0

По умолчанию sshd(8) ожидает входящие запросы на всех IP-адресах, присвоенных сетевым интерфейсам машины. Если необходимо ограничить диапазон прослушиваемых адресов (например, на сервере клеток), сделать это можно так:

```
ListenAddress 192.168.33.8
```

Если требуется, чтобы sshd прослушивал несколько IP-адресов, определите несколько параметров `ListenAddress` в отдельных строках.

SyslogFacility AUTH и LogLevel INFO

Эти два параметра управляют способом протоколирования информации о соединениях. Подробнее о протоколировании рассказывается в главе 19.

LoginGraceTime 2m

Этот параметр определяет максимальный интервал времени, в течение которого пользователь может войти в систему после подключения. Если пользователь выполнит подключение, но в течение этого времени не зарегистрируется, sshd разорвет соединение.

PermitRootLogin no

Не разрешайте пользователям регистрироваться на сервере с учетной записью root. Они должны выполнять вход через SSH как обычные пользователи, а затем получать привилегии root с помощью su(1). Разрешив прямой вход в систему с учетной записью root, позднее вы не сможете определить, чьи действия стали источником проблем, кроме того, злоумышленникам будет проще замести следы.

MaxAuthTries 6

Это максимальное число попыток, которые может предпринять пользователь для ввода пароля. После этого числа неудачных попыток войти в систему соединение с пользователем будет разорвано.

AllowTcpForwarding yes

SSH дает пользователям возможность перенаправлять любой трафик на произвольные порты TCP/IP удаленной системы. Значение `no` запретит такую возможность. Однако если у пользователя есть доступ к командному интерпретатору, он сможет установить собственный механизм перенаправления данных (TCP port forwarder) и обойти это ограничение.

X11Forwarding yes

В UNIX-подобных системах для отображения графического интерфейса программ используется протокол X11 (или X). В X дисплей отделен от физической машины. Вы можете, например, запустить веб-броузер на одной машине, а результаты отображать на другой.

История безопасности X уходит корнями в далекое прошлое, поэтому многие администраторы отключают этот параметр без дополнительных размышлений. Однако запрет на перенаправление X через SSH не означает запрет на перенаправление вообще. Большинство пользователей в случае запрета перенаправления X через SSH просто перенаправляют X через нешифруемое соединение TCP/IP с помощью встроенных сетевых механизмов X или программных продуктов сторонних производителей, что в большинстве случаев гораздо хуже, чем разрешить перенаправление X через SSH. Если на вашем сервере установлены библиотеки X и клиентские программы, пользователь сможет перенаправлять трафик X любым из способов; поэтому лучше разрешить перенаправление через SSH. Если программное обеспечение X не установлено, то этот параметр ни на что не влияет.

MaxStartups 10

Это максимальное число одновременных попыток соединения. Если к системе одновременно попытаются подключиться больше пользователей, чем указано параметре `MaxStartups`, демон sshd(8) будет отвергать некоторые из попыток до того момента, когда другие пользователи

завершат вход в систему, либо истечет предельное время ожидания для других попыток, либо число попыток аутентификации достигнет предельного значения.

Banner /some/path

Баннер (banner) – это сообщение, которое выводится перед тем, как пользователю будет предоставлена возможность аутентифицироваться. Чаще всего этот параметр используется для вывода предупреждений о законности. По умолчанию баннер не используется.

Subsystem sftp /usr/libexec/sftp-server

SSh позволяет безопасно копировать файлы из одной системы в другую с помощью scp(1). Программа scp прекрасно справляется с возложенными на нее обязанностями, но не особо дружелюбна к пользователям. Сервер sftp обеспечивает FTP-подобный интерфейс для передачи файлов, что сокращает время обучения пользователя и при этом гарантирует высокий уровень безопасности.

Управление доступом пользователей через SSH

По умолчанию любой, кто обладает правом доступа к командному интерпретатору, сможет выполнить вход на сервер. Параметры настройки `AllowGroups`, `DenyGroups`, `AllowUsers` и `DenyUsers` позволяют вам определять пользователей и группы, которые будут или не будут получать доступ к вашей машине.

Если явно указать, кто из пользователей сможет войти в систему через SSH, любые другие пользователи не смогут этого сделать.

Например, параметр `AllowGroups` позволит ограничить круг пользователей, обладающих правом входа в систему через SSH, указанными группами, которые определены в файле */etc/group* (глава 7). Если этот параметр определен, а пользователь не принадлежит ни к одной из допустимых групп, он не сможет войти в систему. Группы в списке должны отделяться пробелами:

```
AllowGroups wheel webmaster dnsadmin
```

Если вы не хотите предоставлять доступ через SSH целой группе пользователей, можно с помощью параметра `AllowUsers` определить список допустимых пользователей. При использовании параметра `AllowUsers` автоматически запрещается доступ всем остальным пользователям, отсутствующим в списке.

Параметр `DenyGroups` противоположен параметру `AllowGroups`. Пользователи из перечисленных здесь системных групп не смогут войти в систему. Указанные группы должны быть их главными группами, то есть должны быть представлены в */etc/master.passwd*, а не просто в */etc/group*. Это ограничение делает параметр `DenyGroups` менее полезным, чем может показаться на первый взгляд; для достижения желаемого

эффекта будет недостаточно определить универсальную группу nossh и просто добавить в нее пользователей, эта группа должна быть главной для них. Явное перечисление допустимых групп обеспечивает более полезную политику безопасности.

Наконец, параметр `DenyUsers` содержит перечень пользователей, которым должно быть отказано в доступе. Данный параметр можно использовать, чтобы явно ограничить доступ пользователям из той группы, которой разрешен вход в систему.

Как эти параметры влияют на права доступа пользователя, входящего в несколько групп? Например, пользователь входит в группу, присутствующую в списке `AllowGroups`, и в группу из списка `DenyGroups`. Что произойдет в этом случае? Демон SSH проверяет значения этих параметров в следующем порядке: `DenyUsers`, `AllowUsers`, `DenyGroups` и `AllowGroups`. Учитывается первое совпадение. Предположим, что я член группы wheel, и в файле *sshd_config* присутствует следующий фрагмент:

```
DenyUsers: mwlucas
AllowGroups: wheel
```

Я не смогу войти в эту систему через SSH, потому что значение параметра `DenyUsers` проверяется до значения параметра `AllowGroups`.

Клиенты SSH

Безусловно, клиент SSH уже присутствует в FreeBSD, как и в большинстве других UNIX-подобных операционных систем. По возможности старайтесь пользоваться встроенным клиентом SSH, поскольку он входит в состав пакета OpenSSH, разработанного командой OpenBSD, и является не только самой популярной, но и лучшей реализацией. Если вы вынуждены использовать операционную систему Microsoft Windows, рекомендую программу PuTTY, которая может бесплатно использоваться как в коммерческих, так и в некоммерческих целях и является прекрасным эмулятором терминала.

Эта книга посвящена FreeBSD, поэтому все внимание будет уделено клиенту OpenSSH в этой операционной системе. Вы можете настроить клиента самыми разными способами, но в большинстве случаев настройки сводятся к запрету использования функций, предлагаемых сервером. Если вам действительно интересна тема настройки поведения клиента, прочитайте страницу руководства ssh_config(5).

Чтобы подключиться к другому хосту через SSH, следует ввести команду `ssh` *имя_хоста*. В ответ вы должны увидеть примерно следующее:

```
# ssh sardines.blackhelicopters.org
The authenticity of host 'sardines.blackhelicopters.org (192.168.1.1)' can't
be established.
DSA key fingerprint is a4:df:7c:7e:0e:27:e5:21:b4:f4:0e:2b:c9:10:5f:ea.
Are you sure you want to continue connecting (yes/no)? yes
(Перевод: Подлинность хоста 'sardines.blackhelicopters.org (192.168.1.1)'
```

```
не может быть установлена.
Отпечаток ключа DSA a4:df:7c:7e:0e:27:e5:21:b4:f4:0e:2b:c9:10:5f:ea.
Вы действительно желаете продолжить соединение (yes/no)?)
```

Клиент сразу же получает открытый ключ хоста, к которому производится подключение, и сверяется со своим внутренним списком ключей SSH. Если ключ, полученный от сервера, присутствует в списке клиента, клиент полагает, что соединение устанавливается с подлинным хостом. Если ключ хоста отсутствует в списке клиента, он выводит цифровой отпечаток ключа, чтобы вы могли одобрить или отвергнуть его.

Отпечаток, представленный клиентом, должен совпадать с отпечатком, сгенерированным на сервере. Если отпечатки не совпадают, значит вы подключились не к тому хосту и следует немедленно разорвать соединение. Если отпечатки совпадают, можно принять ключ и продолжить соединение. После того как вы подтвердите свое желание продолжить, ключ хоста будет сохранен в вашем домашнем каталоге в списке *.ssh/known_hosts*.

Если вы создаете новый сервер в локальной сети для личного пользования, возможно, вам не потребуется сравнивать отпечатки ключей вручную. Тем не менее все равно следует скопировать отпечаток ключа, потому что иногда вам может понадобиться подключиться из другого места и проверить ключ. Если к серверу будет подключаться много пользователей, то совсем не лишним будет поместить отпечаток на веб-страницу. Вы сами определяете, какой уровень безопасности вам требуется. Лично я считаю, что лучше излишек осторожности, чем ее недостаток.

Примите ключ хоста, и вам будет позволено выполнить вход в систему. Хотя использование закрытого ключа и секретной фразы предпочтительнее использования простого пароля, все же в SSH пароль защищен намного лучше, чем в telnet.

Копирование файлов через SSH

Клиент SSH хорош для организации доступа к командной строке, а как быть с перемещением файлов из одной системы в другую? SSH включает два инструмента перемещения файлов по сети – scp(1) и sftp(1).

Инструмент scp(1) (secure copy – безопасное копирование) идеально подходит для перемещения отдельных файлов. Команда scp принимает два аргумента: первый – текущее местоположение файла, и второй – место, где файл будет сохранен. Место, куда следует скопировать файл, определяется строкой в формате *<имя_пользователя>@<имя_хоста>:<имя_файла>*. Допустим, требуется скопировать файл *bookbackup.tgz* из локальной системы на удаленный сервер *bewilderbeast.blackhelicopters.org* под другим именем. Для этого вводим команду:

```
# scp bookbackup.tgz mwlucas@bewilderbeast.blackhelicopters.org:bookbackup-
january.tgz
```

Если файла копируется с тем же именем, то имя файла во втором аргументе можно опустить:

```
# scp bookbackup.tgz mwlucas@bewilderbeast.blackhelicopters.org:
```

Кроме того, scp(1) позволяет копировать файлы из удаленной системы в локальную:

```
# scp mwlucas@bewilderbeast.blackhelicopters.org:bookbackup-january.tgz
bookbackup.tgz
```

Если копия файла в локальной системе должна иметь то же самое имя, имя копии в команде можно заменить символом точки:

```
# scp mwlucas@bewilderbeast.blackhelicopters.org:bookbackup.tgz .
```

Наконец, если имена пользователя в удаленной системе и локальной системах совпадают, то можно опустить имя пользователя и символ `@`. Например, я могу скопировать свой файл на сервер с помощью команды:

```
# scp bookbackup.tgz bewilderbeast.blackhelicopters.org:
```

Несмотря на кажущуюся сложность, эта команда очень удобна для быстрого перемещения отдельных файлов по сети.

Если вы предпочитаете интерактивные системы или когда заранее неизвестно точное имя файла, который требуется скопировать из удаленной системы, на помощь придет sftp(1). Команда sftp(1) принимает единственный аргумент, состоящий из имени пользователя и имени сервера, в формате, который используется командой scp для обозначения удаленного сервера:

```
# sftp mwlucas@bewilderbeast,blackhelicopters.org
Connecting to bewilderbeast...
Password:
sftp> ls
```

Интерфейс sftp(1) очень напоминает стандартный клиент FTP; он поддерживает типичные команды FTP, такие как `ls` (list – список), `cd` (change directory – изменить каталог), `get` (download a file – загрузить файл) и `put` (upload a file – выгрузить файл на сервер). Одно из существенных отличий состоит в том, что sftp(1) не требует выбирать тип передаваемых файлов – ASCII или двоичный; файлы просто передаются в том виде, в каком они есть.

OpenSSH защищает все

Многие сетевые программы включают поддержку SSH для обеспечения взаимодействий. Кроме того, OpenSSH может создавать туннели между портами TCP разных машин. Благодаря OpenSSH можно обеспечить передачу любых данных по сети только в зашифрованном виде.

С помощью команд scp и sftp вы сможете полностью устранить из своей сети пароли в открытом текстовом виде.

Время в сети

Если база данных начинает вводить время, отстающее на три часа, или входящие электронные письма датируются завтрашним днем, вы узнаете об этом очень быстро. Время играет очень *важную*[1] роль. Для управления системным временем есть три инструмента: часовой пояс, tzsetup(8); инструмент коррекции времени в сети ntpdate(8) и программа непрерывной коррекции времени ntpd(8). Для начала рассмотрим порядок изменения часового пояса, а потом перейдем к средствам сетевого протокола синхронизации времени (Network Time Protocol) .

Задание часового пояса

Часовой пояс нетрудно установить с помощью tzsetup(8) – программы, управляемой с помощью меню. Она выполнит соответствующие изменения в системе. В крупных компаниях можно по умолчанию задавать среднее время по Гринвичу, а в небольших организациях – местное время. Введите команду `tzsetup`, укажите географическое расположение вашей компании и выберите соответствующий часовой пояс.

Сетевой протокол синхронизации времени

Сетевой протокол синхронизации времени (*Network Time Protocol, NTP*) – это способ синхронизации времени по сети. Вы можете заставить локальный компьютер синхронизировать время с атомными часами в исследовательской лаборатории или с часами вашего основного сервера. Компьютеры, предоставляющие услугу синхронизации времени, называются *серверами времени* и относятся к двум основным типам – уровня 1 и уровня 2 (Tier 1 и Tier 2).

Серверы времени уровня 1 напрямую подсоединяются к высокоточным устройствам, ведущим хронометраж, таким как атомные часы. Им положено быть точными до абсурда. Если вам нужна такая точность, то приобретите атомные часы. Если вас устаивает задержка, присущая передаче со скоростью света, то подойдут радиочасы, которые можно найти в недорогих устройствах GPS.

Серверы NTP уровня 2 подпитываются серверами уровня 1, предоставляя сервисы времени во всеобщее пользование. Точность времени, сообщаемого этими сервисами, составляет доли секунды. Большинству приложений этого более чем достаточно. Дополнительные поиски могут привести даже к серверам времени уровня 3, подпитываемым серверами времени уровня 2.

---

[1] Разумеется, самое важное время – это момент «пора домой».

Отличным источником информации о серверах времени может служить список по адресу *http://www.pool.ntp.org*. Эта группа собрала серверы NTP в круговые пулы через DNS, что позволяет упростить настройку NTP. Эти серверы времени сначала включаются в глобальный список, потом в списки по континентам и, наконец, в списки по странам. Например, если вы проживаете в Канаде, недолгий поиск на этом сайте приведет вас к серверам *0.ca.pool.ntp.org*, *1.ca.pool.ntp.org* и *2.ca.pool.ntp.org*. Мы будем использовать эти серверы в примерах ниже, однако для настройки собственной службы времени вам следует выбрать серверы времени для своей страны.

Конфигурирование ntpd(8)

ntpd(8) периодически сверяет системное время с серверами времени из определенного списка. Опросив их, ntpd вычисляет разумное среднее значение и постепенно подгоняет системное время до соответствия среднему. Если значения на каком-то сервере времени существенно отличаются от остальных, то они не учитываются. Таким образом, вы получаете самое точное системное время, которое только возможно. При этом от того или иного сервера не требуется слишком много, а аппаратные средства находятся под контролем. Конфигурация демона ntpd(8) находится в файле */etc/ntpd.conf*. Вот пример его содержимого:

```
server 1.ca.pool.ntp.org
server 2.ca.pool.ntp.org
server 3.ca.pool.ntp.org
```

Данная система будет получать значения точного времени от трех серверов. Если в списке указать всего один сервер, демон ntpd(8) будет зависеть от часов этого единственного сервера и не сможет опознать появление проблем на сервере времени. Использование двух серверов гарантирует, что ваша система не будет знать точное время; если вы еще помните, при использовании NTP вычисляется среднее из значений, полученных от серверов, но значения, отстоящие далеко друг от друга, просто отбрасываются. Как механизм NTP сможет определить, какое время неверное, если у него будет всего два значения? Оптимальное число используемых серверов – три; если на одном из них появятся проблемы с точностью, ntpd обнаружит, что значение времени, поставляемое этим сервером, лишено смысла по сравнению с другими двумя серверами. (Это своего рода «власть большинства»: тот, чье мнение отличается от мнения большинства, лишается права голоса.)

Мгновенная коррекция времени

ntpd(8) прекрасно справляется с обеспечением достаточно высокой точности хода системных часов в течение длительного времени, но он корректирует локальные часы лишь в небольших пределах. Если же время в системе отличается от фактического на несколько часов или дней (что иногда случается при установке системы или после длитель-

ного отсутствия напряжения), вам наверняка потребуется установить точное время, прежде чем запускать любые приложения, работа которых в значительной мере зависит от точного времени. Демон ntpd(8) предоставляет такую возможность в виде команды `ntpd -q`.

Для однократной принудительной коррекции системных часов используйте команду `ntpd -q`. Она произведет подключение к серверам NTP, получит корректное время, установит системные часы и завершится.

```
# ntpd -q
ntpd: time set -76.976809s
```

В этой системе время отстало примерно на 77 секунд, но теперь синхронизировано с серверами NTP.

Не изменяйте время на рабочих серверах в произвольные моменты времени. В программах, работа которых зависит от точного времени, например в приложениях, управляемых базой данных, возникнут проблемы, если время вдруг скакнет вперед или назад.

Если у вас действительно очень хорошее аппаратное обеспечение с высокоточными часами, то применения команды `ntpd -q` на этапе загрузки вполне достаточно, чтобы решить все проблемы, связанные со временем. Однако такое аппаратное обеспечение есть далеко не у всех. Большинству приходится обходиться аппаратным обеспечением с весьма посредственными часами. В этом случае лучший способ обеспечить точность хода системных часов заключается в запуске демона ntpd(8), который постоянно будет мягко корректировать время.

Запуск ntpd(8) на этапе загрузки

Чтобы выполнить однократную синхронизацию времени во время загрузки и обеспечить последующую непрерывную коррекцию часов, следует определить следующие параметры в файле */etc/rc.conf*:

```
ntpd_enable="YES"
ntpd_sync_on_start="YES"
```

Распространение информации о времени

ntpd потребляет не слишком большой объем сетевого трафика, тем не менее, если каждый сервер в вашей сети станет обращаться к общедоступным серверам времени, возможен неоправданный расход сетевых ресурсов – как ваших, так и серверов-доноров. Это также может привести к небольшим расхождениям в показаниях часов на компьютерах в вашей сети.

Чтобы этого не случилось, рекомендую настроить единый авторитетный сервер времени в вашей сети. Этот сервер должен синхронизироваться с группой серверов NTP в Интернете. А все остальные серверы в вашей сети должны синхронизировать время с этим локальным сервером NTP. При таком подходе все часы в сети будут достаточно точно синхронизированы между собой. Любые ошибки в показаниях часов

будут происходить либо сразу во всей сети (что может служить признаком появления проблем на сервере времени), либо на одном сервере (что свидетельствует о возникшей проблеме на данном конкретном сервере). Вам не придется просматривать все файлы протоколов, пытаясь определить, не вызвано ли отклонение показаний ваших системных часов проблемами на каком-либо общедоступном сервере времени. Такую политику лучше проводить в жизнь с применением правил брандмауэра; позволяя только одному локальному серверу времени взаимодействовать с внешними серверами NTP и оставляя лишь один возможный источник проблем со временем.

Выбор службы имен и кэширование

Любая UNIX-подобная операционная система выполняет неисчислимое множество проверок с помощью различных служб имен. Мы уже говорили о службе доменных имен (Domain Name System, DNS), которая отображает имена хостов в IP-адреса (глава 14), но есть еще служба поиска учетных записей, служба сопоставления номера порта TCP/IP с именем сетевой службы, служба сопоставления имени и номера IP-протокола и т. д. С помощью службы nsswitch (name service switching – выбор службы имен) вы можете указать, как FreeBSD должна выполнять все эти запросы и какие источники информации при этом должны использоваться. Кроме того, в состав операционной системы входит демон службы кэширования имен nscd(8), который хранит результаты обращений к службе имен, производившихся ранее, уменьшая объем сетевого трафика и время выполнения последующих запросов.

/etc/nsswitch.conf

Система выбора службы имен, управляющая тем, как FreeBSD отыскивает сетевую информацию, настраивается с помощью файла */etc/nsswitch.conf*. В этом файле для каждой службы имен имеется отдельная запись, которая включает тип службы и используемые источники информации. В одном из предыдущих примеров (глава 14) мы уже встречались с выбором службы имен. Помните запись, управляющую порядком поиска хостов?

```
hosts: files dns
```

Она означает следующее: «Искать IP-адреса сначала в локальных файлах, а затем использовать службу DNS». Остальные источники информации работают похожим образом. Как и многие другие UNIX-подобные операционные системы, FreeBSD поддерживает возможность выбора служб имен, перечисленных в табл. 15.1, при определении источника информации.

Большинство из них вам не придется изменять, если вы не хотите нарушить нормальную работу системы. Если вы используете Kerberos или у вас имеется домен NIS, например, может потребоваться, чтобы

компьютер, работающий под управлением FreeBSD, использовал их для получения сведений о группах и пользователях, в противном случае перенастройка механизма поиска информации об учетных записях только осложнит, а то и нарушит работу системы!

Таблица 15.1. Службы имен, поддерживаемые системой

Служба	Функция
groups	Проверка принадлежности к группе (*/etc/group*)
hosts	Проверка имени хоста и IP-адреса (DNS)
networks	Поиск информации о сетях (*/etc/networks*)
passwd	Поиск информации об учетных записях (*/etc/passwd*)
shells	Проверка наличия командного интерпретатора (*/etc/shells*)
services	Поиск служб TCP и UDP (*/etc/services*)
rpc	Удаленный вызов процедур (*/etc/rpc*)
proto	Протоколы в сетях TCP/IP (*/etc/protocols*)

Для каждой службы имен необходимо определить один или больше источников информации. Многие из этих служб имен очень просты и по умолчанию имеют единственный авторитетный источник информации – файл. Более сложные, такие как служба hosts, могут иметь несколько источников информации. Некоторые службы сложны лишь благодаря большому числу источников информации и многообразию способов получения этой информации. Поскольку в этой книге не рассматриваются технология Kerberos, сетевая информационная служба NIS и другие пользовательские системы управления уровня предприятия, мы не обсуждаем возможность изменения источников информации для служб passwd, groups и shells. Если вы работаете в такой среде, обращайтесь за дополнительной информацией к странице руководства nsswitch.conf(5).

Службы, которые мы будем рассматривать, могут иметь три источника информации: файлы, dns и кэш. *Файлы* – это стандартные текстовые файлы, содержащие информацию для службы. Например, названия сетевых протоколов традиционно хранятся в файле */etc/protocols*, имена сетевых служб – в файле */etc/services*, сведения об учетных записях – в файле */etc/passwd* и в файлах, сопутствующих ему. Источник *dns* означает, что требуемая информация доступна на сервере DNS (обычное дело для службы hosts). Наконец, *кэш* означает, что информацию можно получить у локального демона службы кэширования имен.

Перечислите все необходимые источники информации в желаемом порядке. Наша запись hosts предписывает службе имен сначала выполнить поиск в локальном файле, а затем обратиться к серверу DNS. Если нужно, чтобы сначала производилось обращение к локальному кэшу, используйте такую запись:

```
hosts: cache files dns
```

Теперь, когда вы знаете, как определяются источники информации для служб имен, давайте посмотрим, как активизировать демон кэширования.

Кэширование имен с помощью nscd(8)

При анализе протоколов моего сервера DNS очень быстро выяснилось, что большая часть запросов исходит от прокси-сервера и сервера управления сетью. В конце концов, этого и следовало ожидать, потому что прокси-сервер каждую секунду запрашивает десятки веб-страниц для пользователей, а система управления опрашивает все жизненно важные системы в моей сети каждые несколько минут. Хотя эти два сервера и не производят существенную нагрузку, тем не менее нет никаких причин продолжать запрашивать одни и те же данные, которые изменяются чрезвычайно редко. Например, рабочая станция управления сетью отыскивает IP-адреса всех маршрутизаторов и коммутаторов каждые 60 секунд. Эти адреса практически не изменяются, поэтому такое решение не только выглядит неэлегантным, но и раздражает, особенно когда мне приходится просматривать файлы протоколов моего сервера DNS. Точно так же, если сервер интегрирован в крупную сеть, вам может потребоваться аутентификация в домене Kerberos или каталоге LDAP, что может привести систему к необходимости выполнять по сети проверку тысяч учетных записей в минуту. Даже файл */etc/services* может содержать несколько тысяч записей, а синтаксический анализ файла в любом случае занимает какое-то время.

Демон кэширования nscd(8) способен обслуживать все источники информации, которые могут быть определены в *nsswitch.conf*. Каждому типу поискового запроса соответствует отдельная строка в файле конфигурации nscd, */etc/nscd.conf*. По умолчанию nscd кэширует все шесть служб имен. Каждая запись содержит ключевое слово и либо название кэша и значение, либо просто значение. Например, запись для службы hosts по умолчанию выглядит так:

```
enable-cache hosts yes
```

Ключевое слово – `enable-cache`, название кэша – `hosts`, а значение – `yes`. При таких настройках nscd(8) будет сохранять информацию об именах хостов. Если теперь в файле */etc/nsswitch.conf* в качестве первого источника информации для службы hosts указать демон nscd, то все запросы службы имен в первую очередь будут направляться локальному демону nscd. С такими настройками моему серверу DNS направляется лишь часть запросов от прокси-серверов и рабочей станции управления сетью.

nscd(8) и синхронизация

У демона nscd много параметров, которые редко могут пригодиться. Два параметра, наиболее полезных с моей точки зрения, связаны с синхронизацией. Длительностью периода хранения информации в кэше

nscd(8) управляют параметры `negative-time-to-live` и `positive-time-to-live`.

Параметр `positive-time-to-live` определяет, как долго nscd(8) будет хранить ответы в кэше. Предположим, что демон nscd(8) кэширует информацию об именах хостов. Определив значение параметра `positive-time-to-live`, вы тем самым укажете демону nscd(8), сколько секунд должен храниться каждый ответ, прежде чем он будет удален из кэша. Следующая строка предписывает nscd(8) хранить информацию об именах хостов 600 секунд (10 минут):

```
positive-time-to-live hosts 600
```

Через 10 минут nscd(8) сотрет все устаревшие ответы и позволит службе имен искать новый ответ. При получении этого нового ответа демон nscd(8) будет хранить его в своем кэше следующие 10 минут. По умолчанию параметр `positive-time-to-live` имеет значение 3600 секунд (1 час).

nscd(8) может хранить не только положительные, но и отрицательные ответы. Запросив IP-адрес хоста *nonexistent.absolutefreebsd.com*, вы узнаете, что такого хоста не существует. Демон nscd(8) сохранит и этот отрицательный ответ. Управлять длительностью хранения отрицательных ответов можно с помощью параметра `negative-time-to-live`, для определения которого используется тот же синтаксис, что и для параметра `positive-time-to-live`. По умолчанию параметр `negative-time-to-live` имеет значение 60 секунд, для любого вида информации.

Очистка кэша

Время от времени в кэше будет сохраняться неверная информация. Например, моя система управления сетью проверяет работоспособность всех жизненно важных сетевых устройств каждые несколько минут. Изменяя IP-адрес маршрутизатора, я вовсе не хочу, чтобы станция управления начала предупреждать меня об отсутствии маршрутизатора; мне требуется, чтобы она, не дожидаясь истечения ближайшего часа, обратилась за получением нового ответа. Пользователь может очистить свой кэш, вызвав nscd с ключом `-i`, а пользователь root может выполнить очистку всего кэша с помощью ключа `-I`.

Например, я могу очистить свой кэш хостов с помощью команды:

```
# nscd -i hosts
```

Если не выполнить очистку кэша, моя система обновит информацию в кэше только по истечении интервалов времени, заданных параметрами `time-to-live`.

Запуск nscd(8) на этапе загрузки

Чтобы автоматически запускать nscd(8) во время загрузки, необходимо добавить в файл */etc/rc.conf* следующую строку:

```
nscd_enable="YES"
```

inetd

Демон inetd(8) обслуживает подключения к тем демонам, услуги которых запрашиваются сравнительно редко. В большинстве систем нет устойчивого потока входящих запросов FTP – тогда зачем нам постоянно работающий демон FTP? Вместо этого за прослушивание порта FTP отвечает inetd. Когда поступает запрос FTP, inetd(8) запускает сервер FTP и передает запрос ему. Из других программ, опирающихся на inetd, можно назвать telnet, tftp и POP3.

Кроме того, inetd обслуживает функции, настолько скромные и редко применяемые, что их легче реализовать в inetd, нежели выделять для них отдельные программы. К таким функциям относятся discard (выполняет дамп данных, поступивших в «черную дыру» */dev/null*), chargen (генерирует поток символов) и т. д. В наше время большинство этих служб не нужны и отключены по умолчанию, но при необходимости их можно активизировать.

inetd и безопасность (8)

Некоторые системные администраторы представляют себе inetd(8) как отдельную службу с единой конфигурацией параметров безопасности. Другие утверждают, что inetd(8) не может похвастаться высоким уровнем безопасности. И те и другие ошибаются. Собственно демон inetd(8) безопасен, но он принимает на себя часть вины тех программ, которым он передает запросы. Некоторые службы, работающие благодаря inetd (такие как ftp и telnet), небезопасны от природы, у других было трудное детство и они заслужили дурную репутацию (например, qpopper). Поступайте с демоном inetd(8) как с любой другой сетевой службой: запускайте его только при необходимости и проверяйте, что с его помощью запускаются только надежные и безопасные программы!

/etc/inetd.conf

Рассмотрим файл */etc/inetd.conf*. У большинства демонов есть отдельные конфигурации для IPv4 и IPv6. Таким образом, если вы не работаете с IPv6, все записи IPv6 можно игнорировать. Обратимся к одной из строк этого файла, задающей конфигурацию сервера FTP:

```
❶ftp ❷stream ❸tcp ❹nowait ❺root ❻/usr/libexec/ftpd ❼ftpd -l
```

Первое поле ❶ – это имя сервиса. Оно должно соответствовать имени в */etc/services*. По имени сервиса демон inetd определяет, какой порт TCP или UDP открыть. Если необходимо изменить порт TCP для сервера FTP, следует изменить номер порта FTP в файле */etc/services*. (Можно было бы также изменить первое поле, подставив в него имя

службы, которая по умолчанию ассоциируется с желаемым номером порта, но я считаю, что от этого пострадает наглядность конфигурационного файла.)

Второе поле ❷ – тип сокета. Все соединения TCP имеют тип `stream`, а соединения UDP – тип `dgram`. Есть и другие возможные значения, однако их применение либо означает такое требование в документации той или иной программы, либо наверняка является ошибкой.

Третье поле ❸ – протокол. Это может быть `tcp` (IPv4 TCP), `udp` (IPv4 UDP), tcp6 (IPv6 TCP) или `udp6` (IPv6 UDP). Если ваш сервер принимает обе разновидности соединений – IPv4 и IPv6, применяйте значение `udp46` или `tcp46`.

Следующее поле ❹ указывает, должен inetd ждать, пока определенный сервер примет соединение, или запустить соответствующую программу и откланяться. Как правило, программы TCP используют `nowait`, а программам UDP необходимо значение `wait`. (Из этого правила есть исключения, но они встречаются крайне редко.) Для каждого входящего соединения inetd(8) запускает новый экземпляр сетевого демона. Если сервис применяет `nowait`, то можно задать максимальное количество соединений в секунду, которые разрешено обслуживать серверу. Для этого сразу после `nowait` надо добавить косую черту (слэш) и соответствующее число, например `nowait/5`. Один из способов, которым злоумышленники (обычно – детки со скриптами) пытаются поставить интернет-сервер на колени, как раз заключается в том, чтобы послать запросов на соединение больше, чем сможет обработать сервер, а ограничение числа входящих соединений поможет остановить такие атаки. С другой стороны, это означает, что злоумышленник сможет воспрепятствовать другим пользоваться службой. Внимательно выбирайте противоядие!

Следующее поле ❺ сообщает, от чьего имени запускается демон. Сервер FTP ftpd(8) работает с привилегиями пользователя root, так как он должен обслуживать запросы от многих системных пользователей, хотя другие серверы могут запускаться от имени специальных, выделенных пользователей.

Шестое поле ❻ – полный путь к программе, которую inetd запускает, получив запрос на соединение. Если это сервис, включенный в состав inetd, будет указано internal.

Последнее поле ❼ задает команду, запускающую внешнюю программу, и необходимые ключи командной строки.

Конфигурирование серверов в inetd

Файлу */etc/inetd.conf*, похоже, нужно много информации, однако для добавления той или иной программы достаточно скопировать существующую строку и незначительно ее изменить. Например, рассмотрим реализацию простейшего сетевого сервиса Quote of the Day (qotd, ци-

тата дня). Когда вы подключаетесь к порту qotd, сервер возвращает случайную цитату и разрывает соединение. В состав коллекции игр FreeBSD входит генератор случайных цитат fortune(1). Такой генератор – это все, что нужно для реализации сетевой программы на базе inetd. Мы должны указать номер порта, сетевой протокол, имя пользователя, путь и командную строку.

Номер порта

В */etc/services* для службы qotd определен порт 17.

Сетевой протокол

Поскольку сервис qotd подразумевает обращение к сетевому порту и получение информации в ответ, это TCP-сервис. Не забывайте, что протокол не ориентирован на установление соединений – он не предполагает возврат ответа. Следовательно, в конфигурации inetd нужно указать протокол `tcp`, что в свою очередь означает необходимость выбора режима `nowait` в четвертом поле записи.

Пользователь

В идеальном случае следовало бы создать отдельную непривилегированную учетную запись для запуска службы qotd, как это делается для таких служб, как pop или proxy. В данном примере мы будем использовать непривилегированную учетную запись nobody, но если вы предполагаете реализовать эту службу на рабочем сервере, обязательно создайте непривилегированную учетную запись qotd.

Путь

Программа fortune находится в каталоге */usr/games/fortune*.

Запуск команды

Программа fortune(6) не требует дополнительных аргументов командной строки, но при необходимости вы можете их использовать. Поклонники законов Мерфи могут использовать команду `fortune murphy`, фанаты сериала «Звездный путь» («Star trek») могут получать цитаты командой `fortune startrek`. (Последняя команда генерирует цитаты только из самого первого сериала «Звездный путь», а не из последующих продолжений.) Я использую команду `fortune -o`, и любой, кто соединяется с моим сервером, получает то, что заслужил.

Пример конфигурации inetd.conf

Собрав все сведения вместе, получим следующую строку для файла */etc/inetd.conf*:

```
qotd stream tcp nowait nobody /usr/games/fortune fortune
```

Этот пример тривиален, однако запуск других служб из inetd(8) выполняется ничуть не сложнее.

Запуск inetd(8)

Для начала необходимо разрешить запуск демона inetd(8) на этапе загрузки. Для этого следует добавить в */etc/rc.conf* следующую строку:

```
inetd_enable=YES
```

Выполнив эту настройку, можно запустить inetd вручную с помощью команды `/etc/rc.d/inetd start`. Теперь, когда inetd запущен, попробуйте с помощью команды telnet подключиться к порту 17 и проверить работу службы qotd:

```
# telnet localhost 17
❶ Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
❷ It is difficult to produce a television documentary that is both
incisive and probing when every twelve minutes one is interrupted by
twelve dancing rabbits singing about toilet paper.
-- Rod Serling
(Перевод: Сложно снять телевизионный документальный фильм, который был бы
острым и познавательным, когда каждые двенадцать минут он прерывается
рекламой с двенадцатью танцующими кроликами, поющими о туалетной бумаге.
-- Род Серлинг)
Connection closed by foreign host.
```

Служба работает! Здесь мы сначала получили привычную информацию о соединении TCP/IP ❶, а затем случайную фразу, сгенерированную программой fortune ❷. (Кроме того, теперь вы знаете, почему я не пишу сценарии для телевидения.)

Изменение поведения inetd

Поведение inetd определяется набором ключей, используемых при его запуске. По умолчанию ключи активизируют TCP Wrappers в соответствии с настройками в */etc/hosts.allow* (глава 9). Некоторые ключи представлены в табл. 15.2.

Таблица 15.2. Ключи inetd

Ключ	Описание
`-l`	Протоколирует все успешные соединения.
`-c`	Устанавливает максимальное количество подключений в секунду, которые может обслуживать любой сервис. По умолчанию количество не ограничено. Заметим, что «не ограничено» не означает «бесконечно много»: количество подключений определяется возможностями аппаратных средств.
`-C`	Устанавливает максимальное количество подключений к сервису в минуту, выполняемых с одного удаленного IP-адреса. По умолчанию количество не ограничено, но введение такого ограничения не позволит отдельным лицам монополизировать всю полосу пропускания.

Ключ	Описание
-R	Устанавливает максимальное количество запусков одного сервиса в минуту. Значение по умолчанию – 256. Если задать -R 0, будет разрешено неограниченное количество подключений к каждому сервису.
-a	Задает IP-адрес, к которому привязан inetd, обычно прослушивающий все доступные IP-адреса.
-w	Применяет TCP Wrappers для внешних служб в соответствии с настройками в *hosts.allow* (глава 9).
-W	Применяет TCP Wrappers для внутренних служб в соответствии с настройками в *hosts.allow* (глава 9).

Рассмотрим пример. Если вы хотите применять TCP Wrappers, разрешить только два соединения в секунду с одного хоста и неограниченное количество обращений к службе в минуту, а также прослушивать только один IP-адрес 192.168.1.2, укажите следующие ключи в файле */etc/rc.conf*:

```
inetd_flags="-Ww -c 2 -R 0 -a 192.168.1.2"
```

Возможности inetd(8) позволяют превратить в сетевую службу практически любое программное обеспечение.

DHCP

Протокол динамической конфигурации хоста (*Dynamic Host Configuration Protocol, DHCP*) – это стандартный способ присвоения IP-адресов клиентским компьютерам. Изначально службы DHCP не интегрированы в систему FreeBSD, но они часто требуются для обеспечения работы бездисковых рабочих станций и в случае использования метода установки PXE (Preboot Execution Environment – предварительная загрузка среды исполнения). Далее рассматриваются основы настройки DHCP, без расширенных возможностей, чтобы дать вам лишь объем знаний, достаточный для настройки компьютеров в офисе, бездисковых клиентов или глобальной сети, связывающей несколько офисов.

Коллекция «портов» FreeBSD включает демон ISC dhcpd, разработанный той же группой программистов-создателей BIND (глава 14). «Порт» можно найти в каталоге */usr/ports/net/isc-dhcp3-server*. Установите этот «порт», следуя инструкциям из главы 11. Вам будет предложено меню с начальными параметрами настройки, в том числе многих особенностей и возможностей демона ISC dhcpd, таких как интеграция с LDAP. Значения по умолчанию вполне пригодны для эксплуатации в большинстве случаев и включают множество параметров обеспечения безопасности. В набор устанавливаемых компонентов входят dhcpd(8), конфигурационный файл */usr/local/etc/dhcpd.conf* и обширные страницы руководства.

Принцип действия DHCP

В сетях со сложной структурой DHCP может оказаться ужасно сложным, но в офисной сети Ethernet он довольно прост. Каждый клиент DHCP отправляет в локальную сеть широковещательный запрос на получение информации о сетевых настройках. Если сервер DHCP находится в этой локальной сети, он получает запрос непосредственно. Если сервер DHCP находится в другом сегменте сети, маршрутизатор этого сегмента должен знать, на какой IP-адрес передавать DHCP-запросы. После этого сервер отправляет клиенту информацию с настройками и запоминает, какому клиенту какой IP-адрес был присвоен. Конфигурация, переданная клиенту, называется *договором о найме* (*lease*). Подобно договорам, по которым вы снимаете жилье или берете напрокат автомобиль, договор о найме DHCP имеет свой срок действия и должен время от времени возобновляться.

В запросе клиент может затребовать определенные характеристики, например, клиенты Microsoft запрашивают IP-адрес сервера WINS, а бездисковые системы – местоположение ядра. При необходимости вы можете управлять всеми этими характеристиками.

Каждый клиент идентифицируется по MAC-адресу сетевой карты, используемой для подключения к сети. Информация о MAC-адресах, IP-адресах, а также о договорах найма сохраняется в файле */var/db/dhcpd/dhcpd.leases*. С помощью этого файла можно определить, какому хосту какой IP-адрес был присвоен. Если хост отключается от сети, а потом появляется снова, dhcpd(8) стремится присвоить ему тот же самый IP-адрес, если он в этот промежуток времени не был присвоен другому клиенту.

Управление dhcpd(8)

Чтобы разрешить запуск dhcpd(8) во время загрузки, следует добавить в */etc/rc.conf* следующую строку:

```
dhcpd_enable="YES"
```

Добавив эту строку, вы сможете использовать сценарий */usr/local/etc/rc.d/isc-dhcpd* для запуска и останова демона.

Конфигурирование dhcpd(8)

Файл */usr/local/etc/dhcpd.conf* содержит все возможные параметры настройки демона dhcpd. Описание всех возможностей dhcpd(8) может занять целую книгу, поэтому мы остановимся лишь на основных функциях, необходимых для работы в сети малого офиса, а также на параметрах, которые будут использоваться в примерах ниже. По умолчанию файл *dhcpd.conf* содержит множество подробных комментариев и примеров, но и электронное руководство является достаточно полным. Мы будем исходить из предположения, что в нашей сети есть только один сервер DHCP, отвечающий на все DHCP-запросы. (В прин-

ципе можно реализовать службу DHCP на базе кластера, чтобы повысить отказоустойчивость, но эта тема далеко выходит за рамки нашей книги.) Начнем рассмотрение *dhcpd.conf* с нескольких общих правил, описывающих конфигурацию клиентов:

```
❶ option domain-name "absolutefreebsd.com";
❷ option domain-name-servers 192.168.1.11, 192.168.1.12;
❸ default-lease-time 3600;
❹ max-lease-time 14400;
❺ authoritative;
❻ ddns-update-style none;
❼ log-facility local7;
```

Каждый клиент DHCP регистрирует свое имя хоста на сервере, а сам сервер должен знать имя локального домена. (В принципе есть возможность организовать назначение клиентам и имен хостов.) Имя домена устанавливается с помощью параметра `domain-name` ❶. Вы можете присвоить своим клиентам DHCP любое имя домена, которое не обязательно должно совпадать с именем домена сервера. Можно указать несколько имен доменов, разделив их пробелами, но не все операционные системы способны распознавать дополнительные домены.

Каждому клиенту сетей TCP/IP необходим один или два сервера DNS. Укажите их с помощью параметра `domain-name-servers` ❷. Серверы имен в списке следует разделять запятыми.

Обычная продолжительность договора о найме (в секундах) определяется параметром `default-lease-time` ❸. По истечении этого времени клиент запросит у сервера новый договор DHCP. Если сервер DHCP окажется недостижим для клиента, он будет продолжать использовать старый договор в течение времени (в минутах), определяемого параметром `max-lease-time` ❹.

Если ваш сервер DHCP единственный в сети, известите dhcpd(8), что его слово в настройке клиента является решающим, добавив ключевое слово `authoritative` ❺.

DHCP может интегрироваться с динамическим DNS, который не рассматривается в этой книге. Наиболее типичный пример использования динамического DNS – служба Microsoft Active Directory. Если вы работаете со службой AD, то скорее всего используете сервер Microsoft DHCP Server, а не ISC dhcpd. При отсутствии AD вам едва ли потребуется динамический DNS. Укажите серверу, что ему не нужно обновлять глобальный DNS, указав параметр `ddns-update-style none` ❻.

Наконец, можно указать, куда демон dhcpd(8) должен отправлять сообщения для протокола ❼. Дополнительную информации о протоколировании вы найдете в главе 19.

Для каждой подсети в вашей сети должно присутствовать определение `subnet`, где указывается идентификационная информация для клиентов DHCP в этой подсети. Пример определения подсети для единой сети офиса:

```
❶ subnet 192.168.1.0 netmask 255.255.255.0 {
❷   range 192.168.1.50 192.168.1.100;
❸   option routers 192.168.1.1;
❹   option netbios-name-servers 192.168.1.3, 192.168.1.5;
}
```

Объявление каждой подсети начинается определением адреса сети и сетевой маски ❶. В данном случае определяется IP-сеть 192.168.1.0 с сетевой маской 255.255.255.0, или диапазон IP-адресов от 192.168.1.1 по 192.168.1.255. Информация в скобках, следующих за объявлением подсети, относится ко всем хостам в этой конкретной подсети.

Ключевое слово `range` ❷ определяет диапазон IP-адресов, которые демон dhcpd(8) может присваивать клиентам. В этом примере клиентам доступен 51 IP-адрес. Если к серверу одновременно попробуют обратиться 52 клиента DHCP, последний из них не получит IP-адрес.

Маршрут по умолчанию определяется с помощью параметра `routers` ❸. Примечательно, что вам не нужно определять дополнительные маршруты с помощью dhcpd(8), – маршрутизатор локальной сети должен обеспечивать доступность адресатов сам. Если в локальной сети присутствует несколько шлюзов, ваш шлюз сам будет обмениваться с клиентами пакетами ICMP для перенаправления их на нужный маршрут. (Ничего страшного, если вам неясно, о чем речь. Когда понадобится, вы быстро начнете понимать то, о чем я говорю, а если такая необходимость никогда не возникнет, значит вы впустую потратили несколько секунд на чтение этих слов.)

Вы можете проявить снисходительность к клиентам Microsoft и определить сервер WINS ❹. Обычно в каждом сегменте сети присутствует свой сервер WINS. Если у вас только одна сеть или только один комплект серверов WINS, можно разместить параметр `netbios-name-servers` за пределами определения подсети.

Если у вас несколько подсетей, создайте описание для каждой из них.

Вот и все! Отредактируйте *dhcpd.conf*, запустите dhcpd(8) и загляните в файл протокола, чтобы увидеть, как клиенты будут получать адреса.

Печать и серверы печати

Тема вывода информации на принтер в UNIX-подобных операционных системах заставит начинающих сисадминов зарыдать; в утешение им бывалые примутся рассуждать о старых добрых временах, когда все принтеры были устройствами TTY, и о молодом поколении, не понимающем своего счастья.[1] Обычно принтер подключается непосредственно к компьютеру, через параллельный порт или USB, но иногда принтеры могут подключаться к сетевому серверу печати (print server).

[1] Мы, старые сисадмины, правы – вы еще не осознали, какой мощью обладаете.

Если у вас принтер подключен непосредственно к компьютеру под управлением FreeBSD, я могу предложить установить универсальную систему печати UNIX (Common UNIX Printing System, CUPS) (*/usr/ports/print/cups*). Этот комплект программного обеспечения способен управлять многими популярными бытовыми и профессиональными принтерами, от небольших струйных и до огромных лазерных. Я не буду вдаваться в детали CUPS, поскольку во многом они зависят от выбранной модели принтера. Подробнее о CUPS можно узнать на сайте *http://www.cups.org*.

Кроме того, CUPS можно использовать для организации доступа к удаленным принтерам, но это довольно редкая ситуация. Я рекомендую использовать CUPS для доступа к удаленным принтерам, если их несколько, и вам может потребоваться переключаться между ними. С другой стороны, обычно мне хочется, чтобы все серверы использовали принтер, который находится рядом с моим столом. Это легко можно осуществить простой настройкой lpd(8).

Не сложнее организовать и доступ к удаленному принтеру, управляемому другим компьютером. Для этого нужно запустить локальный демон печати и указать ему, где находится принтер. Удаленный сервер печати должен уметь общаться по протоколу lpd через порт TCP с номером 515. Проверьте, так ли это, с помощью команды telnet; если вам удалось получить ответ, значит сервер понимает протокол lpd. Большинство UNIX-подобных серверов печати понимают этот протокол, но если вы работаете в окружении Microsoft, попросите администратора Windows установить пакет Print Services for Unix.

Чтобы активировать lpd(8), добавьте в */etc/rc.conf* следующую строку:

```
lpd_enable="YES"
```

Теперь вы сможете вручную запускать и останавливать демон lpd(8) с помощью команд `/etc/rc.d/lpd start` и `stop`.

/etc/printcap

Файл */etc/printcap* управляет настройками принтера с помощью lpd(8). У принтеров есть десятки параметров, от стоимости печати одной страницы до настройки ручной подачи бумаги.

Каждому принтеру, известному системе, должна соответствовать отдельная запись в файле */etc/printcap*, то есть в базе данных возможностей принтеров. Этот файл имеет неудобоваримый, по современным меркам, формат и выглядит малопонятным для всех, кому ранее не приходилось работать с файлом termcap(5). К счастью, для организации доступа к серверу печати вам не требуется понимать формат файла printcap(5), достаточно следовать следующим инструкциям.

Чтобы подключиться к принтеру, соединенному с сервером печати, вам следует узнать имя хоста сервера печати или его IP-адрес и имя принтера. Создайте запись в файле */etc/printcap*, следуя приведенному

ниже шаблону. Особое внимание уделите двоеточиям и символам обратного слэша (\), поскольку они имеют жизненно важное значение.

```
❶ lp|имя_принтера:\
❷       :sh=:\
❸       :rm=имя_сервера_печати:\
❹       :sd=/var/spool/output/lpd/имя_принтера:\
❺       :lf=/var/log/lpd-errs:\
❻       :rp=имя_принтера:
```

В первой строке определяется имя принтера ❶. Для каждого принтера можно указать любое количество имен, разделив их символом вертикальной черты (|). Принтер по умолчанию в UNIX-подобных системах получает имя lp, поэтому обязательно укажите это имя в списке имен предпочтительного принтера. Другое имя должно совпадать с именем, которое используется сервером печати для обозначения этого принтера (например, «3rdFloorPrinter»). Будьте внимательны: нередко серверы печати от Microsoft присваивают несколько имен одному и тому же принтеру, чтобы реализовать различные процедуры печати. Если в вашей сети наблюдается подобная ситуация, выбирайте имя PostScript.

По умолчанию система будет предварять каждое задание печати страницей с названием задания, номером, именем хоста и другой информацией. Если ситуация, когда весь офис пользуется единственным общим принтером, не характерна для вас, то подобная возможность чревата напрасным расходом бумаги. Запись `:sh:\` ❷ подавляет вывод такой страницы.

Параметр `rm` (remote machine – удаленная машина) ❸ определяет имя хоста сервера печати. Вы должны иметь доступ к серверу по имени, указанному здесь.

Для каждого принтера следует определить отдельный временный каталог ❹, где локальный демон печати сможет хранить документы перед отправкой серверу печати. Этот каталог должен принадлежать пользователю root и группе daemon.

В отличие от временных каталогов, которые должны быть отдельными для каждого принтера, файл протокола может быть общим ❺.

Наконец, нужно указать имя удаленного принтера ❻. Файл */etc/printcap* обязательно должен заканчиваться пустой строкой. Следует также отметить, что последнюю строку записи не нужно заканчивать символом обратного слэша (\).

После подготовки файла */etc/printcap* следует перезапустить lpd(8), и только после этого вы сможете увидеть очередь печати с помощью lpd(1) и следить за появлением ошибок в файле */var/log/lpd-errs*.

TFTP

Завершая обсуждение малых сетевых служб, рассмотрим, пожалуй, самый маленький сетевой сервис из тех, что используются по сей день, –

тривиальный протокол передачи файлов (Trivial File Transfer Protocol, TFTP). *TFTP* позволяет передавать файлы с одной машины на другую без аутентификации. Несмотря на меньшую гибкость по сравнению с такими протоколами копирования файлов, как SCP или FTP, производители встраиваемых устройств (например, Cisco) по-прежнему применяют TFTP для загрузки конфигурации системы и обновлений. Мы рассмотрим здесь TFTP по той простой причине, что бездисковые клиенты используют TFTP для загрузки ядра операционной системы и получения начальной информации о конфигурации. Демон tftpd(8) запускается из inetd(8) и прослушивает порт TCP с номером 69.

Настройка tftpd(8) производится в четыре этапа: выбор корневого каталога сервера, создание файлов для сервера, выбор владельца файлов и запуск серверного процесса.

> **TFTP и безопасность**
>
> TFTP не подходит для использования в Интернете, потому что любой желающий сможет читать и записывать файлы на сервер TFTP! Используйте TFTP исключительно за брандмауэром или хотя бы защищайте его с помощью TCP Wrappers (глава 9).

Корневой каталог

По умолчанию tftpd(8) использует каталог */tftproot*. Этот каталог пригоден только для нескольких редко изменяемых файлов, потому что корневой раздел обычно и служит для хранения файлов, которые почти не изменяются. Кроме того, едва ли кому понравится, если система рухнет только потому, что в результате выгрузки файлов на сервер корневой раздел окажется переполнен! Обычно я помещаю корневой каталог tftpd(8) в */var/tftproot* и создаю символическую ссылку */tftproot* на него:

```
# mkdir /var/tftroot
# ln -s /var/tftproot /tftroot
```

Теперь можно создать файлы для доступа через TFTP.

tftpd и файлы

Пользователи могут читать и записывать файлы через TFTP. Если пользователям требуется возможность читать файл через tftpd(8), он должен быть доступен всем для чтения:

```
# chmod +r /var/tftproot/имяфайла
```

Также tftpd(8) позволяет записывать файл, только если он уже существует и доступен всем для записи. Не забывайте, что программы и обычные файлы имеют различные права доступа. Для файла про-

граммы, помимо прав на чтение и на запись, должно быть установлено право на исполнение, поэтому права доступа для программ и обычных файлов вы должны устанавливать по-разному. Заранее создать файл для последующей выгрузки на сервер через TFTP можно с помощью утилиты touch(1).

```
# chmod 666 /var/tftproot/имяфайла
# chmod 777 /var/tftproot/имяпрограммы
```

Разумеется, это означает, что любой, кто знает имя файла, может перезаписать его. Наиболее важные файлы должны быть доступны только для чтения.[1] Это также означает, что вам не придется беспокоиться по поводу того, что кто-то сможет выгрузить слишком большой файл, способный переполнить ваш жесткий диск.

Принадлежность файла

Файлы на сервере TFTP должны принадлежать пользователю с минимальными возможными привилегиями. Если вы редко используете сервер TFTP, можно воспользоваться учетной записью nobody. Например, если сервер TFTP нужен только для нечастых обновлений встроенного устройства, то можно назначить владельцем файлов пользователя nobody и останавливать tftpd(8), когда в нем нет необходимости. Но если сервер TFTP работает постоянно, лучше создать отдельную непривилегированную учетную запись tftp и сделать этого пользователя владельцем файлов. При этом пользователь tftp не должен быть владельцем каталога *tftproot*, в действительности он должен иметь совсем другой домашний каталог. Он нужен лишь для того, чтобы играть роль владельца файлов, доступных пользователям.

Конфигурирование tftpd(8)

Конфигурирование tftpd(8) выполняется исключительно с помощью параметров командной строки, которых не так много. Полный список параметров вы найдете на странице руководства tftpd(8), а здесь описаны только самые полезные из них.

Если специально для запуска tftpd(8) была создана отдельная учетная запись, вы сможете указать имя пользователя с помощью ключа `-u`. Если имя пользователя не указано, tftpd будет запущен с привилегиями пользователя nobody.

Рекомендую протоколировать все запросы, поступающие демону TFTP. Протоколирование включается с помощью ключа `-l`. Для нужд

[1] Разве что вы захотите попробовать установить чей-либо конфигурационный файл в качестве новой подсистемы ввода-вывода (IOS) на своем маршрутизаторе Cisco. Тогда, дозвонившись в службу техподдержки, попросите их включить диктофон, прежде чем излагать суть проблемы, – думаю, они захотят поделиться услышанным со своими коллегами!

протоколирования tftpd(8) использует источник (facility) LOG_FTP, который необходимо активировать в файле *syslog.conf* (глава 19).

Демон tftpd поддерживает возможность работы в chroot-окружении (глава 9), для чего нужно задать ключ -s. Это позволит ограничить дисковое пространство, доступное демону tftpd(8), выбранным каталогом. Не следует делать доступными для чтения пользователям tftp такие файлы, как */boot/kernel/kernel*, – это общий принцип! Всегда запускайте tftpd(8) в chroot-окружении.

Ключ -c позволяет поместить в chroot-окружение клиентов с определенными IP-адресами. В этом случае для всех клиентов, которым будет позволено пользоваться tftpd, необходимо создать по отдельному каталогу. Предположим, что доступ к TFTP получит только один хост (маршрутизатор) с IP-адресом 192.168.1.1. Для этого можно было бы создать каталог */var/tftproot/192.168.1.1* и использовать ключ -c. При этом также следует использовать ключ -s, чтобы определить базовый каталог */var/tftproot*. Это идеальный вариант, когда необходимо обеспечить доступ к TFTP одному или двум хостам и запретить доступ к серверу TFTP всем остальным.

Также можно разрешить клиентам создавать новые файлы на сервере TFTP. Такая практика не может быть признана хорошей, потому что это позволит пользователям переполнить жесткий диск, но если есть такая необходимость, можно использовать ключ -w.

Предположим, требуется протоколировать все запросы к tftpd, ограничить доступную область каталогом */var/tftproot*, запускать сервер с привилегиями пользователя tftpd и помещать клиентов в chroot-окружение. Тогда команда запуска tftpd может выглядеть примерно так:

```
tftpd -l -u tftpd -c -s /var/tftpboot
```

Введите эту строку в файл *inetd.conf*, как описывалось ранее, перезапустите initd(8) – и дело в шляпе!

Планирование заданий

Планировщик заданий операционной системы FreeBSD cron(8) позволяет администратору заставить систему выполнять произвольные команды через определенные интервалы времени. Если вы хотите создавать по ночам резервные копии базы данных или четыре раза в день перезапускать сервер имен, в этом вам поможет cron. Конфигурационные файлы cron называются *crontab* и администрируются с помощью утилиты crontab(1). У каждого пользователя есть свой файл crontab в каталоге */var/cron/tabs*, а глобальный файл crontab хранится как */etc/crontab*.

Файлы crontab пользователей и глобальный /etc/crontab

Назначение глобального файла */etc/crontab* иное, чем у отдельных пользовательских файлов crontab. С помощью */etc/crontab* пользователь root

может указать, какую команду следует запустить и от имени какого пользователя. Например, в */etc/crontab* пользователь root может сказать: «Это задание должно запускаться по вторникам в 22:00 с привилегиями root, а это задание – в 7:00 с привилегиями пользователя www». Другие пользователи могут запускать задания только от своего имени. Разумеется, root может редактировать файлы crontab пользователей.

Кроме того, любой пользователь в системе может просматривать содержимое файла */etc/crontab*. Если вы не хотите, чтобы пользователи знали о наличии запланированного задания, поместите его в пользовательский crontab. Например, если сервер базы данных работает с правами непривилегированного пользователя, используйте crontab этого пользователя для выполнения заданий по обслуживанию базы данных.

Файл */etc/crontab* считается системным файлом FreeBSD. Будьте внимательны и не перезапишите этот файл при обновлении системы. Один из способов упростить обновление */etc/crontab* состоит в том, чтобы добавлять свои записи в конец файла, отделяя их несколькими строками знаков «#».

Наконец, если файл */etc/crontab* можно редактировать в текстовом редакторе, то пользовательские файлы crontab следует редактировать с помощью команды `crontab -e`.

cron и окружение

Задания из crontab запускаются под управлением командного интерпретатора, и программам для нормальной работы могут потребоваться некоторые переменные окружения. Определения переменных окружения можно добавить прямо в строку команды, запускаемой из cron. cron не наследует никакие переменные окружения – все окружение программ должно быть определено явно. Вот, например, список переменных окружения из */etc/crontab* в системе FreeBSD 7:

```
SHELL=/bin/sh
PATH=/etc:/bin:/sbin:/usr/bin:/usr/sbin
HOME=/var/log
```

Да, не густо... В пользовательские файлы crontab вы можете добавлять любые переменные окружения, но при изменении */etc/crontab* лучше придерживаться здорового консерватизма. Если для работы задания необходима какая-то нестандартная переменная окружения, то для запуска такого задания лучше применять пользовательский crontab, а не глобальный */etc/crontab*, потому что этот файл в основном используется для обслуживания системы.

Формат файла crontab

Строки файла crontab, следующие за информацией об окружении, разбиты на шесть колонок. Первые пять колонок представляют время запуска команды: минута, час, число месяца, месяц года и день недели

соответственно. Звездочка (*) в любой колонке означает «*любой(ая)*» (*every one*), а число означает «*именно в это время*». Счет минут, часов и дней недели начинается с 0, а чисел месяца и месяцев – с 1. Кроме того, благодаря старым разногласиям между AT&T и BSD, воскресенье можно обозначтаь двумя числами – 0 или 7. После колонок времени следует команда, которая должна быть запущена в указанное время.

В файле */etc/crontab* имеется одна дополнительная колонка: пользователь, с привилегиями которого должна быть запущена команда. Она находится между колонками времени и командой. Ниже приводится несколько примеров из */etc/crontab*.

Примеры содержимого crontab

Предположим, мы имеем дело с файлом crontab непривилегированного пользователя и собираемся запланировать запуск программы обслуживания. В файле */etc/crontab* имеются заголовки колонок, а здесь будет показано содержимое пользовательского файла crontab. (Чтобы эти примеры можно было использовать в */etc/crontab*, перед командой нужно добавить имя пользователя.) Ниже представлен пример задания, которое запускает сценарий */usr/local/bin/maintenance.sh* каждый час на 55-й минуте:

```
55  *  *  *  *   /usr/local/bin/maintenance.sh
```

Звездочки говорят о том, что cron должен запускать это задание каждый час каждого числа каждого месяца и каждый день недели. Число 55 предписывает cron запускать задание на 55-й минуте.

Чтобы то же самое задание запускать каждый день в 13:55, строка должна выглядеть так:

```
55  13  *  *  *   /usr/local/bin/maintenance.sh
```

Здесь число 13 представляет час в сутках, а 55 – количество минут после часа.

Многие допускают при использовании cron типичную ошибку – указывают старший компонент времени, но пропускают младший. Допустим, вы хотите, чтобы задание запускалось ежедневно в 8 часов утра:

```
*  8  *  *  *   /usr/local/bin/maintenance.sh
```

Это неправильно. Да, задание будет выполняться в 8 часов утра. Но точно так же оно будет выполняться в 8:01, 8:02, 8:03 и т. д., до 9:00. Если задание выполняется больше одной минуты, такая строка быстро поставит систему на колени. Правильный способ указать 8:00 и только 8 часов утра:

```
0  8  *  *  *   /usr/local/bin/maintenance.sh
```

Чтобы указать диапазон времени, скажем, для ежечасного запуска этой программы между 8:00 и 18:00 с понедельника по пятницу, нужна примерно такая строка:

```
55  8-18  *  *  1-5   /usr/local/bin/maintenance.sh
```

Указывая точные моменты времени, разделяйте их запятыми:

```
55  8,10,12,14,16  *  *  *   /usr/local/bin/maintenance.sh
```

Более того, можно указать равные интервалы времени, или *шаги* (*steps*). Например, чтобы программа запускалась каждые пять минут, введите такую строку:

```
*/5  *  *  *  *   /usr/local/bin/maintenance.sh
```

Шаги и диапазоны можно объединять. Например, если задание должно выполняться каждые пять минут, но не раньше, чем через минуту после предыдущего задания, то строка может выглядеть так:

```
1-56/5  *  *  *  *   /usr/local/bin/maintenance.sh
```

Дни выполнения задания можно определять в двух полях, указывая число месяца и день недели. Если указать и число месяца, и день недели, то задание будет запущено при выполнении *любого* из условий. Например, так программе cron указывают «запускать это задание 1-го и 15-го числа плюс каждый понедельник»:

```
55  13  *  1,15  1   /usr/local/bin/maintenance.sh
```

Если для задания необходимо нестандартное окружение, определите его в командной строке, подобно запуску команды в среде интерпретатора. Например, если программе необходима переменная окружения `LD_LIBRARY_PATH`, ее можно установить так:

```
55  *  *  *  *   LD_LIBRARY_PATH=/usr/local/mylibs ; /usr/local/bin/
maintenance.sh
```

Кроме того, с помощью символа `@` cron поддерживает такие варианты планирования, как «ежегодно» или «ежедневно». Однако большинство из этих вариантов лучше не использовать, так как они допускают неоднозначную интерпретацию. Компьютер-то четко знает, что они означают, но люди могут ошибаться! Тем не менее одним из полезных вариантов является `@reboot`, который означает «всякий раз, когда производится загрузка системы». Он позволяет непривилегированным пользователям запускать задания во время загрузки системы. Чтобы использовать такую возможность, нужно вместо пяти колонок с определением времени вставить ключевое слово `@reboot`:

```
@reboot   /usr/local/bin/maintenance.sh
```

Программа cron(8) позволяет планировать автоматический запуск заданий, освобождая пользователя от выполнения рутинных операций по обслуживанию.

Теперь, когда вы получили представление о малых сервисах, предоставляемых операционной системой FreeBSD, можно перейти к рассмотрению более тяжеловесных служб, таких как электронная почта.

16

Спам, черви и вирусы (плюс электронная почта)

Электронная почта – одна из самых убойных услуг, предоставляемых Интернетом, что бы ни говорили соседи-маркетологи. Электронная почта превратилась в излюбленный инструмент доставки непрошенной рекламы, вирусов, червей и прочей зловредной гадости только потому, что электронный почтовый ящик в Интернете есть у каждого. В этой главе рассматриваются вопросы обработки потока электронной почты при взаимодействии «сервер-сервер», «клиент-сервер» и «сервер-клиент». В современном Интернете в каждой из этих трех ситуаций используется немного отличающийся протокол.

FreeBSD прекрасно справляется с обязанностями почтового сервера, способного обслуживать миллионы сообщений в день. Сам Проект FreeBSD использует несколько серверов электронной почты, которые служат для организации рассылок и являются одними из самых нагруженных почтовых серверов в Интернете. Система FreeBSD на приличных аппаратных средствах может получать и передавать свыше 40 000 сообщений в час. Это примерно 11 сообщений в секунду – вместе с графоманскими текстами, неподъемными графическими файлами и раздутыми HTML-страницами, присоединенными к сообщениям. Однако чтобы добиться такой производительности, необходимо понимать, как работает электронная почта.

Обзор электронной почты

Большинство сообщений, передаваемых по электронной почте, создают пользователи на своих настольных компьютерах. Чаще всего это PC под управлением Windows или Mac с Outlook, Eudora, Thunderbird или одной из подобных программ, однако почту можно посылать почти в любой операционной системе.

Клиент посылает сообщения серверу электронной почты. Почти у каждой компании или интернет-провайдера есть по крайней мере одна выделенная почтовая система. Почтовый сервер выполняет основную «санитарную» проверку сообщений, посылаемых клиентом, а затем пытается найти сервер, который возьмет на себя ответственность за доставку адресату. Ваш почтовый сервер, отыскав сервер места назначения, передает ему сообщение электронной почты. Когда получатель проверяет наличие почты, клиентская программа соединяется с почтовым сервером, запрашивает все сообщения и загружает их на настольный компьютер. Если получатель отвечает, весь процесс повторяется в обратную сторону.

Поиск почтового сервера домена

Почтовые серверы обмениваются электронной почтой между собой, но как они отыскивают друг друга? Например, моя личная электронная почта поступает из домена *blackhelicopters.org*. Если я отправлю электронное послание пользователю из домена *freebsd.org* – как мой почтовый сервер отыщет почтовый сервер для домена *freebsd.org*?

В записи DNS для каждого домена почтовые серверы перечислены в виде записей «MX». Мы вкратце рассмотрели записи типа MX в главе 14. (В случае отсутствия записей типа MX почта возвращается к записям типа A, что менее гибко.) Запись DNS для домена, принимающего электронную почту, включает одну или более записей типа MX, каждая из которых содержит число, обозначающее приоритет. Вот пример поиска почтовых серверов для домена *freebsd.org* с помощью запроса MX:

```
# dig freebsd.org mx
...
;; QUESTION SECTION:
;freebsd.org. IN MX

;; ANSWER SECTION:
freebsd.org. 381 IN MX 10 mx1.freebsd.org.
```

В домене *freebsd.org* имеется единственный сервер MX с приоритетом 10. Любая входящая почта, предназначенная для этого домена, будет идти через *mx1.freebsd.org*. Сравните эту запись с записью крупной компании, например *cnn.com*:

```
# dig cnn.com mx
...
;; QUESTION SECTION:
;cnn.com. IN MX

;; ANSWER SECTION:
❶ cnn.com. 3600 IN MX 30 lonmail1.turner.com.
❷ cnn.com. 3600 IN MX 40 hkgmail1.turner.com.
❸ cnn.com. 3600 IN MX 10 atlmail3.turner.com.
❹ cnn.com. 3600 IN MX 10 atlmail5.turner.com.
```

```
❺ cnn.com. 3600 IN MX 20 nycmail1.turner.com.
❻ cnn.com. 3600 IN MX 20 nycmail2.turner.com.
```

У CNN имеется шесть почтовых ретрансляторов! Два сервера имеют приоритет 10. Это самое маленькое значение, поэтому при передаче почты сначала будет произведена попытка отправить ее на сервер *atlmail3.turner.com* ❸ или *atlmail5.turner.com* ❹. Если один из них окажется недоступен, сервер-отправитель попробует связаться с другим. Если оба сервера с приоритетом 10 окажутся недоступны, будет выполнена попытка обратиться к серверу с приоритетом 20, *nycmail1.turner.com* ❺ или *nycmail2.turner.com* ❻, в случайном порядке. Если окажутся недоступны все серверы с приоритетами 10 и 20, сервер-отправитель попробует использовать сервер с приоритетом 30 ❶, а затем с приоритетом 40 ❷. Хотя бы один из этих серверов должен работать и принимать почту.

Наличие единственной записи типа MX становится все более популярной мерой защиты от спама. Если в вашем домене сервер DNS настроен правильно, отправитель будет удерживать почту в очереди несколько дней. Часто резервные почтовые ретрансляторы не знают, какие адреса электронной почты являются допустимыми, что делает их основными кандидатами на получение спама. Многие инструменты рассылки спама используют резервные записи MX как дополнительные пути проникновения в домен. Выше я говорил, что в каждом домене должно быть несколько хостов MX, но с каждым днем становится все труднее и труднее иметь их, не увеличивая объем времени, которое тратится на борьбу со спамом.

Недоставленная почта

Если почтовый сервер обнаруживает почтовые ретрансляторы для требуемого домена, но ни один из них не принимает электронную почту, почтовый сервер помещает сообщение в очередь. Каждые пятнадцать минут, или что-то около того, сервер снова и снова будет пытаться выполнить доставку сообщения. Стандарты Интернета допускают доставку почтового сообщения до пяти суток.[1] Если сообщение не будет доставлено в этот срок, оно возвращается отправителю.

Если почтовому серверу не удалось отыскать в домене запись типа MX или какую-либо запись с именем хоста, совпадающим с именем домена, сообщение немедленно возвращается отправителю. После этого пользователь спросит вас, почему его совершенно нормальное письмо не дошло до адресата, а вы сможете указать ему, что *yahoo.com* пишется не через *Q*.

---

[1] Не забудьте сообщить продавцам из вашей конторы, что электронная почта на вполне законных основаниях может доставляться в течение пяти дней. То-то они позеленеют.

Две совершенно непохожие друг на друга ошибки отправитель электронной почты обрабатывает по-разному. Это одна из причин, почему было бы желательно иметь вторичный сервер имен, расположенный вдали от первичного, – чтобы в случае выхода из строя первичного сервера имен и почтового ретранслятора почтовые сообщения смогли, наконец, попасть к адресату, когда работа сервиса будет восстановлена.

POSTMASTER@EXAMPLE.COM

Копии всех отвергнутых электронных писем отсылаются на учетную запись postmaster вашего домена. Наличие этого электронного адреса является *обязательным*, и все сайты Интернета, выполняющие обмен электронной почтой, *должны* иметь эту учетную запись. Обеспечьте доставку электронной почты, которая приходит на учетную запись postmaster, вашему администратору электронной почты. Стандарты требуют, чтобы эту почту читал человек, а не автоответчик. Если в дополнение к FreeBSD у вас установлена коммерческая почтовая система, убедитесь, что она принимает электронную почту по адресу *postmaster*@!

Протокол SMTP

Передача электронной почты осуществляется по протоколу SMTP (Simple Mail Transfer Protocol – простой протокол передачи почты), который работает через порт TCP с номером 25. По умолчанию электронная почта не шифруется и целостность ее не проверяется. Это очень старый протокол, появившийся в те дни, когда о безопасности не особенно беспокоились. В действительности почту довольно легко послать вручную, не прибегая к клиентской программе. Этот прием можно использовать для устранения сложных неполадок или для того, чтобы похвастаться перед знакомыми. (Только не пытайтесь подобными трюками произвести впечатление на девушку, поверьте моему опыту.)

Определить, способен ли хост получать почту, можно с помощью telnet, выполнив попытку подключиться к порту SMTP:

```
# telnet имяхоста 25
```

Если вы получили ответ, значит в данной системе имеется запущенный почтовый сервер. Если запрос на соединение был отвергнут, следовательно, этот сервер не принимает электронную почту. Попробуем соединиться с системой FreeBSD и узнать, способна ли она принимать электронную почту:

```
# telnet bewilderbeast.blackhelicopters.org 25
Trying 198.22.63.8...
Connected to bewilderbeast.blackhelicopters.org.
Escape character is '^]'.
```

```
220 bewilderbeast.blackhelicopters.org ESMTP Sendmail 8.13.8/8.13.8; Thu, 7
Jun 2008 22:06:25 -0400 (EDT)
```

Такой ответ не только доказывает, что в системе запущен почтовый сервер, он еще демонстрирует точную версию программного обеспечения сервера (Sendmail 8.13.8), а также локальную дату, время и часовой пояс. Единственная непонятная часть ответа – это число `220`. Каждый ответ почтового сервера включает в себя код ответа и понятный для человека текст ответа. Программы, которые общаются с почтовым сервером, читают только код ответа, а остальная часть предназначена для слабого человеческого мозга. Описания всех кодов ответов SMTP можно найти на некоторых сайтах в Интернете.

Попробуем теперь пообщаться с почтовым сервером. Начнем беседу с команды `helo` и имени хоста, с которого выполняется соединение:

```
helo pesty.blackhelicopters.org
```

Сервер должен дать примерно такой ответ:

```
250 bewilderbeast.blackhelicopters.org Hello d149-67-12-
190.col.wideopenwest.com [67.149.190.12], pleased to meet you
```

Полученный ответ включает в себя числовой код (`250`) и имя хоста, с которым установлено соединение (`bewilderbeast.blackhelicopters.org`). Приветствие `Hello` означает, что сервер готов общаться с вами, и он смог идентифицировать компьютер, с которого вы установили соединение. Мой ноутбук имеет сетевое имя *pesty.blackhelicopters.org*, но обратный поиск в DNS по моему IP-адресу показывает, что в действительности я подключен к Интернету через кабельный модем Wide Open West. Имя хоста *pesty* является действительным только за моим домашним брандмауэром.

Теперь с помощью команды `mail from:` сообщим почтовому серверу, от кого исходит сообщение, указав адрес электронной почты автора:

```
mail from: <mwlucas@AbsoluteFreeBSD.com>
250 2.1.0 mwlucas@AbsoluteFreeBSD.com... Sender ok
```

Если сервер готов продолжать общаться с вами, он вернет код `250` и сообщит, что вы можете продолжать. В противном случае, если сервер не пожелает продолжить общение, он закроет соединение. Теперь можно указать имя получателя с помощью команды `rcpt to:`.

```
rcpt to: <mwlucas@blackhelicopters.org>
250 2.1.5 mwlucas@blackhelicopters.org... Recipient ok
```

Если вы лишь проверяете возможность общения с почтовым сервером, укажите другой адрес, а не мой.

Теперь почтовый сервер знает отправителя и получателя. Это наиболее типичное место, в котором отклоняется передача сообщения, как говорится в разделе «Управление ретрансляцией» далее в этой главе. Теперь все готово к отправке вашего сообщения. Выполните команду `data`:

```
data
354 Enter mail, end with "." on a line by itself
```

Теперь можно набирать любое сообщение. Согласно инструкциям, которые следуют за кодом 354, после текста сообщения надо ввести одну точку в отдельной строке. В следующем примере отправляются слова «Тестовое сообщение»:

```
Тестовое сообщение
.
```

После того как точка набрана в отдельной строке, почтовый сервер должен вернуть код 250, подтверждая прием сообщения и сообщая числовой идентификатор (ID) электронного письма:

```
250 2.0.0 l5827nBE004533 Message accepted for delivery
```

Введите команду quit, чтобы закончить сеанс связи.

Такой метод можно применить как во благо, так и во вред. Как администратор вы можете проверять почтовую конфигурацию без возни с клиентской программой, которая может скрыть результаты теста. Однако такой метод позволяет легко подделать сообщения, просто придумав какой угодно свой собственный обратный адрес mail from. Правда, причин, которыми можно было бы оправдать такую подделку[1], не так уж много.

Управление ретрансляцией

Вообще говоря, почтовый сервер принимает почту, предназначенную для его локальных доменов, или почту, отправляемую с них. Если почтовый сервер домена *absolutefreebsd.com* получает сообщение для пользователя с адресом в домене *absolutefreebsd.com*, он принимает это сообщение. Если сообщение от пользователя с адресом в *absolutefreebsd.com* предназначено для другого домена и если имеют место другие процедуры контроля доступа, то сервер принимает это сообщение и отправляет его получателю. Если кто-либо, совершенно не связанный с *absolutefreebsd.com*, пытается использовать данный почтовый сервер как ретранслятор (relay) для перенаправления писем на сторону, сервер отклонит такой запрос.

Пользователи, рассылающие незатребованные коммерческие предложения (спам), постоянно ищут почтовые серверы, позволяющие передавать через них электронную почту. Если на вашем сервере такая

[1] Я занимаюсь подделкой электронной почты, чтобы продемонстрировать начинающим сисадминам, насколько она ненадежна. Хотите превратить своих младших коллег, специалистов по Windows, в сверхпараноидальных админов? Отправьте им письмо от имени контроллера домена с темой «Ежедневный отчет. Хайку» и текстом «Три вещи неизбежны: смерть, налоги и потеря данных. Угадайте, что произошло?» И они навсегда запомнят: а) что сообщение электронной почты можно подделать, и б) вас в лицо.

ретрансляция возможна, он становится потенциальным источником «макулатурной» электронной почты (junk email). Обязательно контролируйте прохождение почты через свою систему. Разрешая неограниченный поток писем через свои серверы, вы попадете в различные черные списки. Если не управлять ретрансляцией почты, можно потерять связь с 30% или 40% хостов Интернета.

Итак, что представляет собой упомянутый выше «контроль доступа»? Одна из самых распространенных составляющих – ограничение IP-адресов, с которых через вашу систему можно посылать почту на любые адреса. Если отправку писем через данный сервер вы разрешите только пользователям локальной или корпоративной сети, посторонние лица уже не смогут использовать ваш сервер для передачи «макулатурной» почты.

Однако если вы оказываете почтовые услуги мобильным пользователям, ситуация сильно усложняется. Вам необходимо обеспечить этих пользователей возможностью отправлять и получать электронную почту без перенастройки их ноутбуков и при этом предотвратить возможность незаконного использования вашего сервера для передачи макулатурной почты.

Борьба со спамом

Любой администратор хотел бы, чтобы доброкачественная электронная почта быстро достигла адресата, точно так же любой администратор хотел бы оградить пользователей от макулатурной почты. Под макулатурной почтой понимается реклама, вирусы, черви и все то, что вы обычно выбрасываете, не читая. Вполне возможно, что макулатурную почту начнет рассылать обычный пользователь вашего сервиса, поэтому в идеале необходимо выработать такие условия пользования сервисом, которые не только запрещали бы подобное поведение, но и предусматривали бы высокие штрафы с целью компенсации убытков, которые вы понесете в случае включения вашего сервиса в черные списки. Для борьбы с макулатурной почтой широко используются два метода: *черный список* и *серый список*.

Черные списки содержат перечень хостов, рассылающих макулатурную почту. Будучи общедоступной службой, учредители черных списков позволяют администраторам почтовых серверов проверять каждое входящее сообщение на попадание в черный список. Если сообщение приходит с IP-адреса, включенного в черный список, сервер отвергает его. При использовании черного списка его администратор должен вызывать у вас доверие, чтобы вы могли добавлять и удалять спамеров, а также проводить действенную нелицеприятную политику по борьбе со спамом.

Серый список – это, пожалуй, самая эффективная методика борьбы со спамом, разработанная в последние несколько лет, которая в действительности не использует насильственных методов. Спам, черви, вирусы

часто рассылаются с некоторыми нарушениями протокола SMTP – почтовые серверы обычно более близко придерживаются стандарта. Электронная почта не является средой передачи сообщений в реальном масштабе времени; предполагается, что почтовые серверы должны выполнить несколько попыток доставить сообщение, прежде чем вернуть его отправителю. Ботнеты, выполняющие отправку макулатурной почты, не придерживаются этого правила – они производят отправку каждому отдельному получателю всего один раз. Таким образом, если ваш почтовый сервер будет принимать сообщения от новых серверов только со второй попытки, он эффективно сможет отделять настоящие почтовые серверы от ботнетов.

В состав операционной системы FreeBSD входит почтовый сервер Sendmail(8), который поддерживает работу как с черными, так и с серыми списками.

Sendmail

Много лет UNIX-подобные операционные системы традиционно включают Sendmail – агент передачи почты (Mail Transfer Agent, MTA), или просто почтовый сервер. Это огромная, малопонятная и бестолковая программа, наводящая ужас на начинающих администраторов. Да и многие опытные администраторы UNIX находят ее огромной, малопонятной, бестолковой и ужасной. Программа Sendmail даже нарушает один из важнейших принципов UNIX, утверждающий, что множество небольших инструментов можно объединять по своему усмотрению. Sendmail – огромная монолитная программа, способная решать множество задач. В то время, когда создавалась программа Sendmail, даже сама идея возможности конфигурирования программы без полной ее перекомпиляции была революционной. И тогда было неважно, что сам конфигурационный файл напоминал результат мышиной возни на клавиатуре. Взгляните на */etc/mail/sendmail.cf* – и вы получите представление о базовом конфигурационном файле sendmail(8).

Сегодня уже нет необходимости редактировать файл *sendmail.cf* вручную. Процедура конфигурирования sendmail стала более простой и управляемой, в результате программа sendmail(8) была интегрирована в систему FreeBSD таким способом, чтобы сделать настройку электронной почты настолько несложной и безболезненной, насколько это возможно.

В течение тех лет, когда управление сервером Sendmail было не таким простым делом, были разработаны альтернативные программные продукты, более удобные в использовании и имевшие более высокий уровень безопасности. Наиболее популярными из этих продуктов стали Postfix (*/usr/ports/mail/postfix*) и Qmail (*http://cr.yp.to/qmail.html*). Оба – замечательные инструменты. Проект FreeBSD использует Postfix для обработки массивных списков почтовой рассылки, самых больших в мире. FreeBSD позволяет легко заменить интегрированный

в систему сервер Sendmail любым из этих пакетов или другим почтовым сервером по вашему выбору.

mailwrapper (8)

В течение многих лет программа Sendmail была единственным почтовым сервером, доступным в UNIX-подобных операционных системах. Вот и в каталоге */usr/sbin/sendmail* присутствует масса дополнительного программного обеспечения, от которого можно было бы ожидать, что оно поведет себя так же, как Sendmail. Хуже всего, что Sendmail ведет себя по-разному, запускаясь под разными именами. Например, программа mailq(1) в действительности – тот же самый файл, что и программа sendmail(8), но, поскольку у него другое имя, программа ведет себя совершенно иначе. Многие другие программы должны точно эмулировать Sendmail, вплоть до различий в поведении при запуске под разными именами, а это не так же просто, как стереть двоичный файл Sendmail и заменить его другим.

В результате администратор, не знакомый с UNIX-подобными системами, может просто не понимать, что собой представляет */usr/sbin/sendmail* в действительности. Если кому-то раньше приходилось ради эксперимента устанавливать различные почтовые серверы, ему придется заняться исследованиями и обладать известной долей удачи, чтобы идентифицировать так называемый Sendmail.

Во FreeBSD эту проблему обходят с помощью отдельной программы mailwrapper(8). Эта программа направляет почтовые запросы требуемой программе почтового сервера. Вы можете найти файл */usr/bin/sendmail*, который в действительности является программой mailwrapper(8), маскирующейся под Sendmail. Эта программа переадресует запросы, отправляемые Sendmail, другим выбранным программам, установленным в другом каталоге.

Файл */etc/mail/mailer.conf* содержит список имен программ с полными именами файлов этих программ. Например, ниже приводится содержимое файла */etc/mail/mailer.conf* по умолчанию для перенаправления всех запросов программе sendmail(8):

```
sendmail    /usr/libexec/sendmail/sendmail
send-mail   /usr/libexec/sendmail/sendmail
mailq       /usr/libexec/sendmail/sendmail
newaliases  /usr/libexec/sendmail/sendmail
hoststat    /usr/libexec/sendmail/sendmail
purgestat   /usr/libexec/sendmail/Sendmail
```

Каждая из шести «программ», перечисленных в левой колонке, – это фактически имена других программ, которые могли бы быть использованы вместо Sendmail. У альтернативных почтовых серверов, таких как Postfix и Qmail, в действительности имеются отдельные программы под этими именами. Если вы собираетесь использовать альтернативный почтовый сервер, отредактируйте файл *mailer.conf*, чтобы

определить правильные пути к файлам соответствующих программ. Если устанавливать альтернативный почтовый сервер из коллекции «портов», «порт» выведет точные инструкции по обновлению файла *mailer.conf*. Следуйте этим инструкциям, чтобы обеспечить работу своего нового MTA. Если альтернативный почтовый сервер устанавливается не из коллекции «портов», вы должны самостоятельно определить, какие изменения следует внести в *mailer.conf*.

Прием и передача

В системе FreeBSD (как и во всех других UNIX-подобных операционных системах) Sendmail служит для решения самых разных задач. Одни машины принимают почту из Интернета и либо отправляют ее дальше на другие машины, либо доставляют локальным пользователям. Другие машины не принимают электронную почту, а только генерируют ее. Каждую из этих задач программа Sendmail решает немного по-своему.

Демон передачи Sendmail обслуживает отправку электронной почты с локальной машины. В большинстве случаев такая электронная почта просто направляется почтовому ретранслятору. Например, все серверы в моей сети отправляют ежедневные отчеты о состоянии на фактический сервер электронной почты, откуда я их получаю. Сервер гораздо больше доверяет сообщениям локального происхождения, чем сообщениям, прибывающим из Интернета, поэтому процесс Sendmail обслуживает эти сообщения по гораздо менее строгим правилам и выполняет намного меньше проверок. Это процесс Sendmail прослушивает только локальный IP-адрес 127.0.0.1, поэтому другие хосты в сети не могут воспользоваться им.

Демон приема Sendmail принимает электронную почту с других хостов в сети. Этот процесс Sendmail требует особой заботы и внимания. Именно этот процесс стараются скомпрометировать злоумышленники, и именно он является потенциальным звеном в цепи передачи макулатурной почты. Большую часть своей энергии мы потратим на настройку демона, выполняющего прием почты.

Эти две конфигурации являются взаимоисключающими. Развернув в системе полноценный почтовый интернет-сервер, вы откажетесь от более слабого представителя Sendmail, занимающегося обработкой локальной почты, как большинство из нас не пойдет спать из дома в сарай. Система FreeBSD автоматически запрещает запуск демона передачи Sendmail, если активизирован демон приема почты.

Активизация демонов передачи и приема почты производится в файле */etc/rc.conf* по отдельности. Полное описание всех конфигурационных параметров Sendmail вы найдете в странице руководства rc.sendmail(8), а здесь я приведу четыре основных:

```
sendmail_enable="NO"
sendmail_submit_enable="YES"
```

```
sendmail_outbound_enable="YES"
sendmail_msp_queue_enable="YES"
```

Параметр `sendmail_enable` активизирует демон приема почты. Если вам нужно только отправлять почту с локального хоста и не нужно принимать ее, включите параметр `sendmail_submit_enable`. Параметры `sendmail_outbound_enable` и `sendmail_msp_queue_enable` отвечают за активизацию функций передачи и ретрансляции электронной почты, отправляемой с локального хоста.

Большая часть конфигурационных параметров Sendmail, о которых рассказано ниже, относится к демону приема. Процесс передачи Sendmail требует незначительного числа настроек. При описании конфигурационных параметров, применяемых к демону передачи Sendmail, это будет подчеркнуто особо. В противном случае будет подразумеваться, что конфигурационные параметры предназначены для настройки демона приема. Не то чтобы эти параметры препятствовали работе демона передачи, скорее, они просто не будут иметь смысла для него. Например, зачем нужна функция управления доступом на базе имен хостов или IP-адресов или функция защиты от спама демону, к которому можно обратиться только с локального хоста?

Протоколирование сообщений Sendmail

Все свои действия, ошибки и повторные запуски sendmail(8) протоколирует в файле */var/log/maillog*. Проверяйте его, чтобы иметь представление о деятельности сервера электронной почты, особенно после внесения изменений в его конфигурацию. Когда мне приходится работать с системой электронной почты, я предпочитаю открывать файл протокола в отдельном окне терминала и просматривать записи по мере их поступления с помощью команды `tail -F`:

```
# tail -F /var/log/maillog
```

Всякий раз, когда Sendmail отправляет, принимает или отвергает сообщение электронной почты, он записывает соответствующее сообщение в файл протокола. Команда `tail -F` показывает изменение этого файла в реальном масштабе времени. Чтобы остановить отображение содержимого файла протокола и вернуться в командную строку, нужно нажать комбинацию клавиш Ctrl-C.

Конфигурирование Sendmail

Все удобочитаемые и неудобочитаемые конфигурационные файлы Sendmail находятся в каталоге */etc/mail*. В большинстве случаев, когда сервер занимается приемом входящей почты, предметом беспокойства для вас будут всего четыре файла: */etc/mail/access*, */etc/mail/aliases*, */etc/mail/mailertable* и */etc/mail/relay-domains*.

Файл *access* позволяет управлять доступом к почтовому серверу на уровне доменов и отдельных хостов.

Файл *aliases* содержит карту перенаправлений электронной почты для локального хоста.

Файл *mailertable* позволяет переопределять записи типа MX для данного почтового сервера. Файл *mailertable* особенно удобен при использовании системы FreeBSD для обеспечения дополнительной степени защиты в проприетарных системах с недостаточным уровнем безопасности, таких как Microsoft Exchange.

Файл *relay-domains* содержит перечень имен доменов и адресов, для которых ваш сервер будет выполнять функции ретранслятора электронной почты. В основном, в этот список будут включаться клиенты, использующие вашу машину в качестве почтового сервера, однако в этот список могут попасть и другие домены, для которых ваш сервер будет играть роль резервного ретранслятора почты.

Для каждого из этих файлов (за исключением *relay-domains*) имеется соответствующая база данных, файл с расширением *.db*. Подобно файлам */etc/passwd.db* и */etc/spwd.db*, они создаются из текстовых файлов для обеспечения программ более быстрым способом доступа к данным. После изменения текстового файла вам придется обновлять и соответствующий файл базы данных, о чем подробнее рассказано при рассмотрении каждого конкретного файла.

Файл access

Файл */etc/mail/access* позволяет явно определить, кто сможет передавать почту через вашу систему. Этот файл обычно используется, чтобы ограничить прием почты от нежелательных корреспондентов. По умолчанию Sendmail отвергает любые почтовые сообщения, не предназначенные для локальной системы. Этот файл позволит переопределить другие правила Sendmail, в частности – черные списки. База данных с правилами доступа в идеале должна использоваться для описания редких исключений из других правил, описываемых параметрами настройки Sendmail, хотя есть и задачи, которые можно решить только с помощью базы данных access.

Файл */etc/mail/access* состоит из двух колонок. В левой колонке указываются имя домена, имя хоста, IP-адрес или блок IP-адресов, откуда приходит электронная почта. В правой колонке указываются инструкции, определяющие действия с электронной почтой. Это может быть одна из инструкций `RELAY`, `OK`, `REJECT` или `DISCARD`. Кроме того, здесь же с помощью инструкции `ERROR` можно определить текст своего сообщения об ошибке для определенных хостов или доменов. Умудренные опытом администраторы Sendmail могут также указывать свои коды ошибок, но для этого требуется более глубокое знание протокола SMTP, чем может дать эта книга.

Например, ниже разрешается прием почты от локального хоста с IP-адресом 127.0.0.1. Это всегда желательное правило, на случай если кто-то по ошибке занесет локальный хост в черный список базы данных о спамерах. Правило RELAY предписывает Sendmail перенаправлять электронную почту локального происхождения в любом направлении. Перечень допустимых IP-адресов локальной сети также поможет преодолеть ограничения, если вдруг ваша компания попадет в черный список, которым вы пользуетесь. Безусловно, вы прекратите пользоваться черным списком, в котором перечислена ваша компания, но когда вас занесут в черный список, данное правило позволит безболезненно дожить до того момента, когда вы это обнаружите. Подобно */etc/login.access*, в файле */etc/mail/access* нельзя указывать диапазоны IP-адресов в современной нотации, но можно указывать часть IP-адреса, соответствующую диапазону адресов. Ниже я хотел включить диапазон адресов 192.168.0.0/23, но из-за недостатка, указанного выше, мне пришлось включить два блока IP-адресов 192.168.0 и 192.168.1:

```
127.0.0.1    RELAY
192.168.0    RELAY
192.168.1    RELAY
```

Инструкция OK – более строгая, она означает, что демон Sendmail должен только принимать, но не ретранслировать электронную почту от указанных хостов. Допустим, у вас имеется компания-партнер, постоянно попадающая в черный список только потому, что ее интернет-провайдер также обслуживает известного спамера. В этом случае с помощью инструкции OK можно определить правило, по которому будет приниматься почта от этой компании:

```
theircompany.com    OK
```

Еще более строгой инструкцией является инструкция REJECT, которая предписывает категорически отвергать любую почту, поступающую из указанного источника. Предположим, некто с IP-адресом 192.168.8.83 передает вам огромные объемы электронной почты, пытаясь вызвать переполнение жесткого диска на вашем почтовом сервере. В этом случае вы можете отказаться от приема почты с помощью инструкции REJECT:

```
192.168.8.83    REJECT
```

Вы можете отвергнуть почту, послав в ответ собственное сообщение об ошибке и указав текст сообщения правее инструкции ERROR. Обычно, отвергая прием почты, Sendmail генерирует код сообщения 550. Ваше сообщение об ошибке увидит конечный пользователь, поэтому проявляйте определенную долю такта при создании своих сообщений. Текст сообщения должен окружаться кавычками, чтобы Sendmail не удалил пробелы и не изменил его каким-либо другим способом.

```
suckycompany.com ERROR: 550 "Ваша мама весьма забавно одевает вас"
```

Наконец, некоторые хосты могут быть настолько недружелюбными к вам, что вы даже видеть не хотите почту, поступающую от них. И даже

не желаете утруждать себя отклонением сообщений от них. В этом случае используйте инструкцию DISCARD:

```
absolutefreebsd.com   DISCARD
```

Файл aliases

Файл */etc/mail/aliases* содержит пути перенаправления электронной почты для конкретных учетных записей или пользователей. Каждая строка в этом файле начинается с псевдонима, а после него через двоеточие указывается список действительных пользователей системы, которым должна быть перенаправлена почта. Почтовые псевдонимы учитываются как в конфигурации приема, так и в конфигурации передачи почты.

Пересылка почты от одного пользователя другому

Многие администраторы предпочитают пересылать почту, отправленную пользователю root, на действительную учетную запись. Следующий пример демонстрирует, как организовать пересылку почты пользователя root на другую учетную запись:

```
root:   mwlucas@AbsoluteFreeBSD.com
```

В действительности многие адреса электронной почты не имеют учетных записей, связанных с ними. Например, обязательный адрес *postmaster@* обычно не связан с действительной учетной записью. Пересылать такую электронную почту на действительную учетную запись можно с помощью псевдонима:

```
postmaster:   root
```

Согласно этим двум примерам почта для postmaster будет пересылаться на адрес root, а с адреса root – мне, то есть я буду получать всю почту, которая поступает на адреса *root@* и *postmaster@* на этой машине.

Файл *aliases* уже содержит несколько стандартных адресов интернет-сервисов, а также псевдонимы для всех учетных записей сервисов FreeBSD, используемых по умолчанию. Все они по умолчанию выполняют переадресацию на учетную запись root. Определив действительный адрес электронной почты для псевдонима root, вы автоматически будете получать все системные электронные сообщения.

Псевдонимы как списки рассылки

В определении псевдонима можно также перечислить несколько пользователей, создавая таким способом небольшие локальные списки рассылки электронной почты. Данный прием не подходит для организации динамических списков, где пользователи часто подписываются и отписываются, но он вполне подходит для быстрой организации простых списков:

```
sales:   mwlucas, bpollock, sales@nostarch.com
```

Пересылка почты в файлы

С помощью файла *aliases* можно реализовать еще более интересную возможность – пересылку сообщений в другие пункты назначения, не являющиеся адресами электронной почты. Если указать имя файла, Sendmail будет дописывать сообщения в этот файл. Можно организовать протоколирование всей почты, поступающей на электронный адрес, например так:

```
mwlucas:   /var/log/mwlucas-mail, mwlucas
```

Пересылка почты программам

Кроме того, можно пересылать почту программам для автоматической обработки. Для этого просто укажите полный путь к программе после символа вертикальной черты (|). Например, если у вас есть сценарий для обработки входящей почты, поступающей на определенный адрес, организовать пересылку почты можно с помощью такого определения псевдонима:

```
orders:   |/usr/local/bin/process-orders.pl
```

Включения

Наконец, в файл *aliases* можно включать содержимое других файлов. Это позволит доверенным пользователям изменять свои псевдонимы самостоятельно. Включаемый файл должен содержать список электронных адресов, по одному адресу в строке.

```
clientlist:   include:/usr/home/salesdude/clientlist.txt
```

Файл mailertable

Файл *mailertable* позволяет выборочно переопределять записи MX. Такая возможность была важна, когда сети использовались для подключения к почтовым системам вроде UUCP, но и сегодня она позволяет превратить FreeBSD в своего рода брандмауэр или бастион. В ряде случаев я применяю Microsoft Exchange – продукт, который нельзя напрямую соединять с Интернетом из соображений безопасности. Задействовав машину FreeBSD в качестве общедоступного ретранслятора почты и файл *mailertable* на ней, я смог организовать защиту системы Exchange без покупки дорогого брандмауэра электронной почты.

У меня нет системы Exchange в домене *absolutefreebsd.com*, но для примера предположим, что она есть. Мои хосты называются *freebsd.absolutefreebsd.com* и *exchange.absolutefreebsd.com*. Моя запись DNS могла бы выглядеть так:

```
absolutefreebsd.com   IN   MX   10   freebsd.absolutefreebsd.com
```

Теперь все могут посылать электронную почту на машину с операционной системой FreeBSD.

На этой машине я добавил бы в файл *mailertable* такую строку:

```
.absolutefreebsd.com    smtp:[exchange.absolutefreebsd.com]
```

Это позволило бы машине с операционной системой FreeBSD игнорировать запись MX для домена, указанного в левой колонке, и пересылать всю почту для этого домена на хост *exchange.absolutefreebsd.com*.

Файл relay-domains

Sendmail упрощает создание резервных ретрансляторов почты для других доменов. Как и в случае со вторичными DNS, лучший способ обрести резервный ретранслятор состоит в том, чтобы найти близкую по размеру компанию для обмена услугами. Нет большого смысла тратить время на организацию резервного ретранслятора, особенно для организаций, сильно отличающихся от вашей, лучше найти примерно такую же организацию с близким уровнем подготовки технического персонала, чтобы ваш партнер не вызывал у вас дополнительных проблем.

Файл */etc/mail/relay-domains* содержит список всех доменов и/или IP-адресов, для которых ваш сервер будет играть роль ретранслятора. Имена доменов и адреса следует указывать по одному в строке. Например, ниже приводится запись, которая образует сервис резервного ретранслятора почты для домена *absolutefreebsd.com*:

```
absolutefreebsd.com
```

Трудно, да? Не забудьте указать сервер в DNS-записи домена, чтобы резервный MX-сервис заработал.

Активизация изменений

Для файлов *access*, *aliases* и *mailertable* имеются соответствующие файлы базы данных (*.db*). Подобно файлу паролей, эти файлы обеспечивают простой и быстрый способ получения ссылки на содержимое конфигурационных файлов Sendmail и других программ. Изменив один из этих файлов, следует обновить и соответствующий ему файл базы данных. Современные версии Sendmail обеспечивают управление этими файлами базы данных с помощью утилиты make(1). Чтобы обновить файлы *access* и *mailertable*, следует перейти в каталог */etc/mail* и ввести команду `make maps`:

```
# cd /etc/mail
# make maps
```

Утилита make(1) сравнит время последнего изменения каждого файла с временем последнего изменения соответствующего файла базы данных и выполнит пересборку базы данных, если текстовый файл окажется новее.

Чтобы перестроить базу данных *aliases*, нужно запустить команду newaliases(8) либо make aliases:

```
# cd /etc/mail
# make aliases
```

Почему для перестройки базы данных *aliases* используется отдельная команда? Дело в том, что aliases стала первой базой данных в Sendmail, поэтому для ее обновления была написана специальная команда. С появлением новых файлов базы данных был выработан более универсальный метод.

Изменения в базе данных вступают в силу немедленно, сразу после ее пересборки. Однако другие изменения требуют перезапуска всей почтовой системы. Изменения в файле *mailertable* относятся именно к таким изменениям. Почтовую систему можно перезапустить разными способами, но я рекомендую использовать для этого систему начального запуска FreeBSD, чтобы обеспечить нормальное поведение почтового сервера после перезагрузки:

```
# /etc/rc.d/sendmail restart
```

Это обеспечит вступление в силу всех изменений, включая изменения в текстовых файлах.

Самые наблюдательные могут заметить, что */etc/rc.d/sendmail* в действительности является интерфейсом к сценарию командного интерпретатора */etc/rc.sendmail*. Файл */etc/rc.sendmail* был создан еще до появления системы запуска rc.d и остается для обеспечения обратной совместимости.

Виртуальные домены

По умолчанию Sendmail отвергает все сообщения, не предназначенные для локального хоста. Чтобы в такой конфигурации почта работала, необходимо совпадение имени хоста почтового сервера с именем домена, то есть такая схема будет работать, если, например, почтовый сервер для домена *absolutefreebsd.com* будет работать на машине с именем *absolutefreebsd.com*. Но так бывает редко. Например, у меня почтовый сервер домена *absolutefreebsd.com* запущен на машине с именем *bewilderbeast.blackhelicopters.org*. *Виртуальные домены* позволяют организовать получение почты для другого домена. Одна машина, работающая под управлением FreeBSD, может обслуживать сотни и тысячи виртуальных доменов.

Конфигурирование виртуального домена производится в три этапа. Первый этап – определить почтовому серверу домен, за который он будет нести ответственность. Затем сконфигурировать пересылку электронной почты пользователей в этом домене на соответствующие учетные записи. И, наконец, опубликовать MX-запись для этого домена, которая будет указывать на ваш почтовый сервер.

Файл /etc/mail/local-host-names

Файл */etc/mail/local-host-names* определяет имена хостов и доменов, почту для которых будет принимать данная машина. Добавив хост или домен в этот файл, вы тем самым сообщаете Sendmail, что локальная машина является конечной точкой на пути электронной почты, связанной с этим хостом или доменом. Этот файл содержит список хостов и доменов, по одному в каждой строке, например так:

```
blackhelicopters.org
absolutefreebsd.com
absoluteopenbsd.com
pgpandgpg.com
ciscoroutersforthedesperate.com
...
```

Не перечисляйте здесь домены, почту для которых вам не требуется принимать! Sendmail воспринимает этот файл как последнее слово для любого домена, перечисленного здесь, и если почта не сможет быть доставлена локально, отправителю будет возвращено сообщение об ошибке. Перечислив здесь лишние удаленные домены, вы никому не доставите неудобств – кроме своих пользователей.

Карта пользователей

Теперь, чтобы Sendmail могла принимать электронную почту домена, нужно сообщить системе, куда доставлять почту этого домена. По умолчанию Sendmail пытается отыскать учетную запись с тем же именем, что и адрес электронной почты. Моя машина принимает почту для доменов *blackhelicopters.org* и *absolutefreebsd.com*. По умолчанию Sendmail будет доставлять почту, направленную по адресу *mwlucas@absolutefreebsd.com*, локальному пользователю mwlucas, а почту, направленную по адресу *mwlucas@blackhelicopters.org*, также локальному пользователю mwlucas. Теоретически вполне возможно, что это два совершенно разных человека. Пользователем *mwlucas@blackhelicopters.org* мог бы быть Mark W. Lucas, не имеющий никакого отношения к вашему покорному слуге.[1] Мне не нужна его электронная почта, как и ему не нужна моя. Вы не можете запретить использование одинаковых имен пользователей в разных доменах. Решение состоит в том, чтобы обеспечить Sendmail возможностью отображения имен учетных записей в виртуальной таблице пользователей, */etc/mail/virtusertable*. Ниже приводится пример нескольких записей:

```
stenchmaster@blackhelicopters.org   chris
postmaster@blackhelicopters.org     mwlucas
mwlucas@blackhelicopters.org        mwlucas
```

[1] В реальной жизни эти два адреса, конечно же, относятся к одному и тому же человеку, но саму идею, я думаю, вы поняли.

```
silence@blackhelicopters.org        error:nouser spambait address
@blackhelicopters.org               error:nouser no such user
```

Первая запись отображает конкретный адрес электронной почты слева на конкретную учетную запись справа. Письма, отправленные на адрес *stenchmaster@blackhelicopters.org*, будут доставляться на локальную учетную запись `chris`. Следующие две записи тоже отображают конкретные электронные адреса в домене на учетную запись пользователя в локальной системе. Четвертая запись более интересна, поскольку возвращает отправителю сообщение об ошибке. Мне нужно, чтобы система возвращала сообщение «нет такого пользователя» любому, кто отправит письмо на адрес *silence@blackhelicopters.org*. Я использую этот адрес при заполнении форм на сайтах, от которых я ожидаю получить спам. Я активизирую этот адрес на несколько минут, чтобы получить письмо, необходимое для активизации учетной записи, а затем снова блокирую этот адрес. Наконец, последняя запись – это универсальное правило, которое применяется ко всему домену в целом. Указав адрес *@blackhelicopters.org*, я предписываю Sendmail применять это правило ко всем адресам электронной почты, для которых отсутствуют конкретные правила. В ответ на сообщения, отправленные на адреса, отсутствующие в этом списке, будет посылаться сообщение об ошибке «no such user» (нет такого пользователя).

Когда-то многие системные администраторы использовали такие универсальные адреса электронной почты собственных доменов, создавая адреса «на лету». «Хотите пообщаться со мной? Вот мой личный адрес электронной почты в домене». Однако теперь, когда спамеры могут выполнить подбор по словарю, наличие такого правила отображения

Подбор по словарю

Подбор по словарю (dictionary attack) – это одна из разновидностей атак, когда злоумышленник пытается перебирать возможные имена пользователей, одно за другим. Спамер может попытаться выполнить подбор по словарю, пробуя посылать спам на все возможные адреса в домене. Быстро рассылая сообщения по адресам в последовательности: *aardvark@yourdomain.com*, *aardwolf@yourdomain.com*, *aaron@yourdomain.com* и далее по словарю, спамер надеется, что какая-то часть макулатурной почты успеет достигнуть действительных пользователей до того, как его отключат. Файл */usr/share/dict/words* содержит 235 882 слова; если ваш сервер будет принимать сообщения по адресам, соответствующим каждому из этих слов, и пересылать их на вашу учетную запись, вам не останется ничего иного, как уничтожить все сообщения разом, уповая на то, что во время атаки никто не послал вам ничего важного.

универсального адреса на личную учетную запись представляет собой отличный способ получить огромный объем спама.

Создайте аналогичные записи для других виртуальных доменов. Если нужно, чтобы адреса *mwlucas@absolutefreebsd.com* и *mwlucas@blackhelicopters.org* соответствовали разным учетным записям, просто создайте отдельные записи в файле *virtusertable*:

```
mwlucas@absolutefreebsd.com   michael
mwlucas@blackhelicopters.org  mwlucas
```

Я рекомендую перечислять имена доменов в алфавитном порядке, чтобы уменьшить вероятность ошибки, когда у вас будут тысячи доменов на одной машине.

Подобно другим большим конфигурационным файлам Sendmail, доступ к информации в файле *virtusertable* осуществляется через файл базы данных, который необходимо обновлять после каждого изменения. Изменения в *virtusertable* не возымеют эффект, пока вы не перейдете в каталог */etc/mail* и не запустите команду `make maps`, как поступали для обновления других файлов базы данных Sendmail.

Теперь достаточно опубликовать MX-запись для вашего домена, указав в ней ваш почтовый сервер, и виртуальные домены начнут действовать.

Изменение sendmail.cf

Помните, я говорил, как ужасен конфигурационный файл */etc/mail/sendmail.cf*? Да, вам придется изменять его. Однако не нужно редактировать *sendmail.cf* напрямую. На самом деле, у Sendmail есть инструменты для сборки этого конфигурационного файла из других конфигурационных файлов. Сегодня звучит странновато, но во время появления Sendmail это считалось передовым достижением. (Думаю, молодежь скажет: «Старая школа – это отпад».)

Операционная система FreeBSD включает два файла с расширением *.cf* – *sendmail.cf* и *submit.cf*. Файл *sendmail.cf* содержит настройки Sendmail, которые определяют порядок пересылки почты другим системам и ее получения процессом приема почты, если таковой запущен. Файл *submit.cf* используется только в конфигурации передачи почты и только для обслуживания очереди неотправленных сообщений. В действительности файл *submit.cf* намного проще своего собрата, если у вас хватит стойкости и желания читать его. Эти два файла создаются автоматически, за счет применения правил из двух отдельных файлов, *freebsd.mc* и *freebsd.submit.mc*.

Файлы с расширением *.mc* – это конфигурационные файлы процессора макроязыка m4. Команда m4(1) читает инструкции из этих файлов и определения из */usr/share/sendmail/cf* и на их основе создает файлы *.cf*. Желательное поведение Sendmail определяется в файлах *.mc*, а затем на их основе с помощью команды m4 создаются соответствующие

конфигурационные файлы Sendmail. Например, рассмотрим файл *freebsd.mc*.

Хотя файл начинается с символа решетки (#), обозначающего комментарии, тем не менее после начала конфигурационных определений символ решетки больше не встречается – вместо него, если нужно добавить комментарий, который будет игнорироваться утилитой m4, используется строка символов dnl. Некоторые из строк в этом файле выглядят достаточно очевидными:

```
...
FEATURE(access_db, ❶`hash -o -T<TMPF> /etc/mail/access❷')
...
FEATURE(mailertable, `hash -o /etc/mail/mailertable')
FEATURE(virtusertable, `hash -o /etc/mail/virtusertable')
...
```

Мы только что говорили о файлах *access*, *mailertable* и *virtusertable*! Эти инструкции предписывают утилите m4(1) включить эти особенности с помощью хешей, созданных из указанных имен файлов. Пока все это не очень пугает.

Однако взгляните на эти инструкции внимательнее. Здесь определены имена особенностей и за ними указаны имена конфигурационных файлов в апострофах. Первый апостроф – это на самом деле обратная одиночная кавычка ❶, а второй – обычная одиночная кавычка ❷. Апострофы следует указывать именно в таком виде и в таком порядке, в противном случае m4(1) не сможет корректно интерпретировать файл.

Здесь также можно встретить ряд конфигурационных параметров, закомментированных с помощью строки dnl:

```
dnl Dialup users should uncomment and define this appropriately
dnl define(`SMART_HOST', `your.isp.mail.server')
```

Если вы хотите использовать интеллектуальный хост, то удалите символы dnl в начале второй строки и впишите имя хоста своего почтового сервера. Подробнее об интеллектуальных хостах мы поговорим ниже в этой главе.

Изменяя записи в файле *.mc* и пересобирая конфигурационный файл, вы можете управлять работой Sendmail. Мы задействуем эту возможность для настройки системы. Ниже я продемонстрирую два простых примера, а полученные знания мы используем в последующих разделах.

Нестандартные файлы .mc

Файлы с инструкциями (*Makefile*) для сборки конфигурационных файлов Sendmail по умолчанию имеют имена, составленные из имени хоста системы с расширением *.mc*. Например, на моем ноутбуке *pesty.blackhelicopters.org* имеются два файла *.mc*: */etc/mail/pesty.blackhelicopters.org.mc* и */etc/mail/pesty.blackhelicopters.org.submit.mc*. Если

такие файлы у вас отсутствуют, make(1) создаст их, скопировав содержимое файлов *sendmail.mc* и *submit.mc*.

Я не попадал в ситуации, когда приходилось бы редактировать файл *submit.mc*. Не исключая появления таких ситуаций, я полагаю, что желание отредактировать этот файл, скорее всего, лишнее. Практически всегда вам придется иметь дело только с файлом *<имя_хоста>.mc*.

Для начала скопируем файл *freebsd.mc* в *<имя_хоста>.mc* и затем внесем в него все необходимые изменения. Первым сконфигурируем интеллектуальный хост.

Интеллектуальные хосты

Интеллектуальный хост (*smart host*) – это локальный почтовый сервер, который знает, как отправить почту за пределы локальной сети. Я не желаю, чтобы мой ноутбук контактировал с удаленными почтовыми серверами напрямую; предпочитаю вручать свою почту моему выделенному почтовому серверу, который возьмет на себя всю грязную работу по поиску MX-записей, идентификации серверов, а также передаче и повторной передаче моей электронной почты, пока удаленный сервер не снизойдет до того, чтобы принять ее. Ноутбук должен быть свободен для моих более важных дел, как то: слушать MP3-файлы и писать всю эту чепуху.

Вы можете настроить интеллектуальный хост в своем файле *<имя_хоста>.mc*. Помните пример интеллектуального хоста из предыдущего раздела? Просто укажите имя своего интеллектуального хоста и пересоберите файл *sendmail.cf*. В качестве интеллектуального хоста у себя в сети я использую перегруженный *bewilderbeast.blackhelicopters.org*, поэтому отредактирую свой файл */etc/mail/pesty.blackhelicopters.org.mc* так:

```
dnl Dialup users should uncomment and define this appropriately
define(`SMART_HOST', `bewilderbeast.blackhelicopters.org')
```

Здесь я удалил символы `dnl` в начале второй строки. Тем самым я раскомментировал определение параметра `SMART_HOST`, сделав его доступным для m4(1). Я благоразумно не тронул символы апострофов, заключив в них имя хоста своего почтового ретранслятора. Теперь мне нужно пересобрать *sendmail.cf* с помощью команды `make all` и установить его командой `make install`:

```
# cd /etc/mail
# make all
/usr/bin/m4 -D_CF_DIR_=/usr/share/sendmail/cf/ /usr/share/sendmail/cf/m4/cf.m4
pesty.blackhelicopters.org.mc > pesty.blackhelicopters.org.cf
# make install
install -m 444 pesty.blackhelicopters.org.cf /etc/mail/sendmail.cf
install -m 444 pesty.blackhelicopters.org.submit.cf /etc/mail/submit.cf
```

Здесь видно, что m4(1) использует файл *<имя_хоста>.mc* для сборки файла *<имя_хоста>.cf*, который затем устанавливается как *send-*

mail.cf. Мои изменения вступят в силу сразу после перезапуска sendmail(8). Теперь мой ноутбук будет отправлять всю электронную почту серверу *bewilderbeast.blackhelicopters.org*, который и доставит ее удаленным серверам.

Если вместо файла *<имя_хоста>.mc* используется другой файл, укажите полный путь к этому файлу в переменной `SENDMAIL_MC` в файле */etc/make.conf*.

Блокирование источников спама

Наверное, проще всего уменьшить поток макулатурной почты с помощью *черного списка*. Черный список – это список IP-адресов, откуда поступает макулатурная почта. Есть множество черных списков, созданных по разным критериям, с разной степенью детализации и уровнем серьезности. Прежде чем выбрать какой-то определенный черный список, попробуйте описать для себя предъявляемые требования. Отыскать черные списки антиспама можно с помощью любой поисковой системы.

Прежде чем использовать черный список, выясните размер организации, которая его сопровождает. Список с названием «Fred's Anti-Spam Blacklist» может обслуживаться одним человеком, чья энергия, направленная на поддержку списка, подвержена приливам и отливам. Небольшие организации, занимающиеся ведением черных списков, могут иметь неоправданные предубеждения против некоторых видов деятельности, поэтому следует очень тщательно оценить стандарты такой организации, прежде чем использовать ее черный список. Кроме того, обязательно посмотрите, как организация добавляет и удаляет записи своего черного списка. Учитывают ли они мнение общественности? Если да, то как они проверяют это мнение? Как люди выбираются из черного списка? Помещает ли эта организация в свой черный список большие диапазоны IP-адресов из-за некорректного поведения одной или двух небольших сетей в этом диапазоне? Если да, то использование такого черного списка будет препятствовать получению электронной почты от хостов с IP-адресами из этих диапазонов. Это может никак не сказаться на вас лично, но вы должны учесть все факторы, прежде чем взяться за обслуживание электронной почты других людей.

К моменту написания этих строк моим любимым черным списком был Spamhaus (*http://www.spamhaus.org*). Для почтовых серверов с небольшим трафиком компания Spamhaus, созданная в 1999 году, предоставляет услугу бесплатно, а для крупномасштабных сайтов – по невысокой цене. Черный список ZEN этой компании включает IP-адреса известных производителей макулатурной почты, оконечных устройств (разного рода модемы), которые никогда не должны отправлять электронную почту непосредственно, а также адреса в ботнетах. Любые из этих адресов могут оказаться источниками спама, вирусов и прочей гадости, поэтому, отклоняя электронную почту, поступающую с этих

адресов, вы сможете значительно снизить объем получаемой макулатурной почты. Если вы предоставляете услугу интеллектуального хоста клиентам, использующим SASL для доступа к ретранслятору (как будет обсуждаться в следующем разделе), список ZEN может заблокировать таких пользователей. В этом случае можно порекомендовать использовать список SBL вместо ZEN. В примерах этого раздела мы будем использовать услугу Spamhaus ZEN, но вы можете использовать любой другой черный список.

По умолчанию *sendmail.mc* включает пример записи, определяющей черный список:

```
dnl Unc
```

Черный список по умолчанию, расположенный на сайте *http://www.mail-abuse.org*, – это платная услуга для любых пользователей, даже для тех, у кого почтовый трафик очень невелик, но в рамках этой услуги предоставляется шаблон настройки черного списка. Проверьте свой черный список на наличие необходимой информации об имени хоста и затем добавьте правило, напоминающее следующее (в одной строке):

```
FEATURE(❶`dnsbl', ❷`zen.spamhaus.org', ❸``550 Mail from ¨
❹$``&{client_addr} ¨ refused - see ❺http://www.spamhaus.org/zen/¨')
```

Здесь программе Sendmail предписывается использовать черный список на базе информации DNS ❶ и выполнять проверку записей в домене *zen.spamhaus.org* ❷. Кроме того, мы вставили в правило свое сообщение об ошибке ❸, включив в него IP-адрес клиента ❹ и указав источник дополнительной информации ❺.

Пересоберите свой файл *sendmail.cf*, перезапустите Sendmail, чтобы задействовать черный список, и посмотрите в файле протокола на записи для входящих сообщений. Сообщения, заблокированные в результате использования черного списка, в */var/log/maillog* выглядят примерно так:

```
Jun 16 12:10:13 bewilderbeast sm-mta[40174]: ruleset=check_relay, arg1=82-
46-225-100.cable.ubr04.dund.blueyonder.co.uk, arg2=127.0.0.4, relay=82-46-
225-100.cable.ubr04.dund.blueyonder.co.uk [82.46.225.100], reject=550 5.7.1
Mail from 82.46.225.100 refused - see http://www.spamhaus.org/zen/
```

Это сообщение макулатурной почты, на которую вам не придется больше тратить свое время. Впрочем, черные списки – далеко не идеальный инструмент, хотя они и помогут вам существенно уменьшить поток макулатурной почты. Следующий раздел посвящен другому мощному инструменту борьбы со спамом – серым спискам.

Серые списки

Черные списки обеспечивают одностороннюю защиту против известных источников спама. Белые списки описывают известные источники доброкачественной электронной почты. Ключевое слово в обоих

случаях – «известные». Защититься от неизвестного источника угрозы гораздо сложнее, чем от известного. Механизм *серых списков* (greylists) способен обеспечить вполне приемлемый уровень защиты и позволит отклонять почту, поступающую из известных источников спама, и принимать ее из известных доброкачественных источников, помогая при этом сортировать почту из любых других источников по этим двум категориям.

Большинство источников макулатурной почты стремятся доставить максимум сообщений максимальному числу адресатов. Для этого они используют «урезанную» версию протокола SMTP. В частности, большинство ботнетов выполняют всего одну попытку доставить почту. Настоящие почтовые серверы будут производить такие попытки в течение пяти суток, а ботнеты сразу же сдаются, как только им сообщают о необходимости выполнить повторную попытку. Механизм серых списков учитывает эту особенность, отличающую ботнеты и настоящие почтовые серверы, отвергая почту от первых и принимая от вторых.

При первой попытке почтовый сервер с настроенным механизмом серых списков, принимая соединение от ранее неизвестного почтового сервера, возвращает код ошибки, который означает: «Временный сбой, пожалуйста, попробуйте повторить попытку позднее», одновременно фиксируя IP-адрес сервера-отправителя и электронный адрес получателя почты. Если удаленный почтовый сервер повторит попытку доставить ту же самую почту еще раз через заданный интервал времени, локальный почтовый сервер примет сообщение. После этого сервер, оснащенный механизмом серых списков, добавляет отправителя и получателя в список доброкачественных операций и некоторое время принимает все почтовые отправления, соответствующие этому описанию, без задержки. Типичный источник спама, предприняв всего одну попытку или две, в течение короткого промежутка времени, прекращает дальнейшие попытки.

Безусловно, спамеры пытаются подстроиться под складывающиеся условия, и некоторые ботнеты пытаются переслать почту через несколько часов. Хорошо, если за это время учредители черных списков успеют идентифицировать новые источники спама и добавить их IP-адреса в свои черные списки. У вас есть шанс воспользоваться преимуществами механизма серых списков.

Единственный недостаток серых списков – они задерживают доставку легитимной почты. Это так, хотя и с некоторыми оговорками. Администратор почтового сервера может добавлять известные ему источники доброкачественной почты в *белый список*, то есть список адресов источников электронной почты, которые никогда не должны проверяться механизмом серых списков. Так можно обеспечить беспрепятственное прохождение почты от крупных клиентов и важных деловых партнеров.

В разных агентах передачи почты (MTA) можно обнаружить множество различий в реализации серых списков. По моему мнению, наиболее мощным и удобным является реализация spamd(8) из OpenBSD, разработанная Бобом Беком (Bob Beck). Эта система серых списков применяет пакетный фильтр PF и обладает множеством способов[1] уменьшения потока макулатурной почты, но она очень сложна в настройке. Если на вашем почтовом сервере задействован пакетный фильтр, вам следует рассмотреть возможность применения spamd(8). Ниже приведен пример с milter-greylist (*/usr/port/mail/milter-greylist*) – более простой в настройке программой, дающей при этом почти такие же хорошие (хотя и менее забавные) результаты, как у spamd.

Конфигурирование milter-greylist

После установки milter-greylist загляните в конфигурационный файл */usr/local/etc/mail/greylist.conf*. Это типичный для UNIX конфигурационный файл, содержащий серию определений параметров и их значений, где комментарии обозначены символами решетки (#). В этом конфигурационном файле можно задать предельные времена ожидания, IP-адреса соседних почтовых ретрансляторов, «белые» адреса отправителей и получателей и множество других характеристик. Здесь мы рассмотрим только те из них, которые необходимы для средней системы. Более подробную информацию о механизме серых списков вы найдете на странице руководства greylist.conf(5). milter-greylist – очень серьезный программный продукт и наверняка пригодится вам в отдельных ситуациях.

Базовая настройка программы

В начале конфигурационного файла расположен набор базовых параметров настройки программы. Вам не следует изменять их.

```
pidfile "/var/run/milter-greylist.pid"
socket "/var/milter-greylist/milter-greylist.sock"
dumpfile "/var/milter-greylist/greylist.db"
user "mailnull"
```

Параметр `pidfile` – это имя файла, в котором хранится числовой идентификатор процесса (PID) запущенного демона milter-sendmail(8). Параметр `socket` – это сокет UNIX, посредством которого демон Sendmail

[1] Мой любимый прикол с применением PF/spamd: ограничить скорость приема входящего спама, выделив спамеру полосу пропускания, как у модема со скоростью 900 бод. Одна машина с настроенной связкой PF/spamd способна удерживать хосты ботнета в занятом состоянии до нескольких дней, не затрачивая при этом свои системные ресурсы. Самое приятное – пытаясь передать почту вам, спамбот не отправит тысячи других макулатурных сообщений. Это одна из немногих компьютерных фишек, заставляющих меня улыбнуться даже спустя годы после того, как я о них узнал.

взаимодействует с демоном milter-sendmail. Параметр `dumpfile` – это база данных для идентификации доброкачественной почты. Обнаружив новый источник доброкачественной почты, milter-greylist добавляет его в эту базу данных. Наконец, параметр `user` определяет непривилегированную учетную запись, с правами которой работает демон milter-sendmail.

Соседние почтовые ретрансляторы

Если в сети несколько почтовых ретрансляторов, могут наблюдаться попытки доставить легитимную почту через все ретрансляторы по порядку. Это типичное поведение легитимных ретрансляторов почты, поэтому механизм серых списков должен учитывать данную особенность. Параметр `peer` позволяет указать адреса других почтовых ретрансляторов, также использующих механизм серых списков, с целью обеспечить синхронизацию баз данных между ними. Допустим, у меня имеется основной почтовый сервер и три резервных почтовых ретранслятора с адресами 192.168.1.1, 172.16.18.3 и 10.19.84.3 – я мог бы перечислить их в конфигурационном файле как соседние почтовые ретрансляторы:

```
peer 192.168.1.1
peer 172.16.18.3
peer 10.19.84.3
```

Как только я добавлю соответствующие записи в конфигурацию других почтовых ретрансляторов, они автоматически начнут обмениваться между собой данными серого списка.

Списки адресов

Можно определять отдельные списки IP-адресов (ключевое слово `list`) и имен (ключевое слово `addr`). Адреса в списке нужно заключить в фигурные скобки, разделяя пробелами. Например, можно создать список с названием `my network`, который будет содержать IP-адреса компьютеров вашей организации:

```
list "my network" addr {198.22.63.0/24 192.168.0.0/16}
```

Как и во многих других файлах, где могут содержаться длинные строки, символ обратного слэша (\) указывает, что запись продолжается в следующей строке. Ниже приведено начало списка основных почтовых серверов, для работы с которыми не должен применяться механизм серых списков:

```
list "broken mta" addr { \
        12.5.136.141/32 \ # Southwest Airlines (unique sender)
        12.5.136.142/32 \ # Southwest Airlines
        12.5.136.143/32 \ # Southwest Airlines
...
```

Для каждого сервера в списке есть комментарий, облегчающий его идентификацию, – это позволяет списку бесконечно удлиняться, оставаясь удобочитаемым.

Списки доменов

Список доменов похож на список адресов. Каждый домен должен быть фактическим доменным именем почтового сервера, выполняющего входящее соединение, а не доменным именем адреса электронной почты конечного пользователя. Например, стремясь обеспечить беспрепятственное обслуживание электронной почты от моего издателя, я создал список, который включает его домен. Электронная почта от моего издателя может приходить с сервера *nostarch.com* или с одного из компьютеров провайдера, к чьим услугам он прибегает, из домена *laughingsquid.net*. Список доменов задается с помощью ключевого слова `domain`:

```
list "good domains" domain { nostarch.com laughingsquid.net}
```

Списки пользователей

С помощью параметра `rcpt` можно создавать списки своих пользователей, чтобы применять для них особые правила обработки почты:

```
list "spam lovers" rcpt { \
    sales@absolutefreebsd.com \
    cstrzelc@stenchmaster.com \
}
```

Списки управления доступом

milter-greylist обслуживает входящие соединения в соответствии с правилами, определенными в *списках управления доступом* (*access control lists, ACL*). Списки управления доступом позволяют применять механизм серых списков с той или иной степенью точности по вашему выбору. milter-greylist проверяет входящие сообщения на соответствие правилам списков управления доступом и применяет первое подошедшее правило. Чтобы определить поведение сервера для сообщений, которые не совпали ни с одним из предыдущих правил, можно использовать специальное определение `acl default`, что очень похоже на то, как производится прием или отклонение пакетов по умолчанию в пакетном фильтре. Например, следующие правила разрешают прием почты из локальной сети от известных почтовых серверов, для работы с которыми не должен использоваться механизм серых списков, а также от моего издателя. Некоторые из моих пользователей не нуждаются в обслуживании механизмом серых списков, поэтому они специально помещены в белый список, но вся остальная электронная почта проверяется механизмом серых списков:

```
acl whitelist list "my network"
acl whitelist list "good domains"
acl whitelist list "broken mta"
```

```
acl whitelist list "spam lovers"
acl greylist default
```

Первая запись разрешает прием электронной почты, приходящей от компьютеров с IP-адресами, включенными в список `my network`. Вторая разрешает прием электронной почты, исходящей из доменов, перечисленных в списке `good domains`. Третья разрешает прием электронной почты, приходящей от компьютеров с IP-адресами, включенными в список `broken mta`. Четвертая запись определяет пользователей, которые не будут защищены механизмом серых списков, то есть будут получать всю почту без исключений. Наконец, вся остальная почта по умолчанию будет подвергаться дополнительной проверке, то есть отправителям придется подтвердить, что они являются нормальными почтовыми серверами, прежде чем предлагаемая ими почта будет принята.

Интервалы времени в серых списках

По умолчанию milter-greylist требует повторить передачу через три минуты. Некоторые спамботы пытаются повторить передачу макулатурной почты в течение нескольких минут, но очень немногие готовы ждать 30 минут. Поведение спамботов меняется по мере роста мощности компьютеров, и я не удивлюсь, если со временем они смогут повторять передачу через все более и более длительные интервалы. Так или иначе, вам придется настроить период временного отказа. Делается это с помощью ключевого слова `delay` в списках управления доступом. Для обозначения дней, часов и минут можно использовать сокращения `d`, `h` и `m`, соответственно. Ниже я задал интервал времени ожидания, равный двум часам, что было безумно долго для 2007 года, но, к сожалению, достаточно разумно для последующей пары лет:

```
acl greylist default delay 2h
```

Как только сервер-отправитель пройдет проверку, milter-greylist сохранит информацию о нем до следующей проверки сервера, которая состоится через трое суток. В течение этого времени вся входящая почта от этого сервера будет приниматься без задержки. Возможно, вам потребуется изменить этот интервал, это зависит от вашего окружения. Величина этого интервала задается с помощью ключевого слова `autowhite`. Следующая настройка предписывает milter-greylist пропускать почту от отправителей, благополучно прошедших проверку, в течение 30 суток.

```
acl greylist default autowhite 30d
```

Как правило, временные интервалы, устанавливаемые для механизма серых списков, сильно зависят от вашего окружения. Может оказаться, что для успешной борьбы со спамом вам вполне хватит пятиминутной задержки, а другим для отсева макулатурной почты может потребоваться увеличить интервал задержки до нескольких часов.

Подключение milter-greylist к Sendmail

milter-greylist исполняется за пределами Sendmail. Как же организовать взаимодействие между Sendmail и milter-greylist? Ключом к успеху является интерфейс подключения Sendmail, позволяющий внешним программам подключаться к Sendmail для обеспечения дополнительной функциональности.

При установке milter-greylist из «портов» вы увидите инструкции по подключению milter-greylist к Sendmail. Обычно для этого придется добавить несколько строк в конец файла *<имя_хоста>.mc* и пересобрать *sendmail.cf*. Например, во время написания этих строк «порт» milter-greylist предлагал добавить в файл *.mc* следующее:

```
dnl j,{if_addr},{cert_subject},i,{auth_authen} are already enabled by default
define(`confMILTER_MACROS_HELO', confMILTER_MACROS_HELO``, {verify}'')
define(`confMILTER_MACROS_ENVRCPT', confMILTER_MACROS_ENVRCPT``, {greylist}'')
INPUT_MAIL_FILTER(`greylist', `S=local:/var/milter-greylist/miltergreylist.
sock, F=T, T=R:30s')
```

Записи, начинающиеся с ключевого слова `define`, присваивают значения макроопределениям внутри Sendmail, а инструкция `INPUT_MAIL_FILTER` предписывает Sendmail пропускать входящую почту через фильтр, доступный через указанный сокет домена UNIX.

Я добавил эти строки в файл *sendmail.mc* и запустил команду `make all restart`. Sendmail тут же начал общаться с запущенным ранее процессом milter-greylist, и спустя 15 минут широкий и мутный от макулатурной почты поток превратился в прозрачный ручеек легитимной почты.[1]

Sendmail и аутентификация SASL

Больше всего проблем доставляют пользователи, которые могут пользоваться услугами электронной почты из любой точки мира. Почтовый сервер должен принимать почту из любого места, если она предназначена для такого пользователя. Президент вашей компании, отправившись в Антарктиду на важную встречу с крупным клиентом, ожидает, что почтовый сервер беспрепятственно примет почту от него, несмотря на всю вашу хитрую борьбу со спамом. Сделать это позволяет протокол *простой авторизации и уровня безопасности* (*Simple Authentication and Security Layer, SASL*). Протокол SASL требует, чтобы пользователь прошел аутентификацию на почтовом сервере, прежде чем он примет почту. Это означает, что некоторые, «законные», пользователи смогут пользоваться услугами почтового сервера независимо от своего местонахождения, а другие смогут отправлять почту только с известных, добропорядочных IP-адресов.

---

[1] Пропуская только не-спам, я не собираюсь отвечать абсолютно на все письма. Кажется, я дожил до состояния, когда уже *не хочется* получать информацию.

Для включения поддержки SASL в Sendmail понадобится пересобрать Sendmail из исходных текстов. Чтобы избежать мучений с программой Sendmail, входящей в базовую установку системы, FreeBSD включает «порт» Sendmail в */usr/ports/mail/sendmail*. С помощью этого «порта» можно собрать Sendmail с дополнительными параметрами и использовать эту программу вместо той, что входит в базовую установку системы. Использование «порта» означает, что вы легко можете пересобрать Sendmail с нужными параметрами, избежав процедуры наложения заплаток на базовую систему Sendmail (и их последующего обновления). Заглянув в файл *Makefile* «порта», можно увидеть множество дополнительных параметров Sendmail, начиная от поддержки баз данных для хранения внутренней информации и заканчивая интеграцией с LDAP. Перейдите в каталог «порта» и запустите команду:

```
# make SENDMAIL_WITH_SASL2=YES all install clean
```

Она соберет программное обеспечение поддержки SASL версии 2 и Sendmail с поддержкой SASL, после чего установит программы в каталог */usr/local*, как все «порты».

saslauthd(8)

Демон saslauthd(8) выполняет аутентификацию пользователя по запросу других приложений. В данном случае нам нужно, чтобы Sendmail аутентифицировала пользователя в соответствии с базой данных паролей FreeBSD. Аутентифицированный пользователь сможет передать почту, неаутентифицированный – нет. Чтобы демон saslauthd(8) смог обрабатывать запросы Sendmail на аутентификацию, его следует активировать в файле */etc/rc.conf*:

```
saslauthd_enable="YES"
```

После этого нужно либо перезагрузить систему, либо запустить saslauthd(8) командой `/usr/local/etc/rc.d/saslauthd start`.

mailer.conf и обновленная программа Sendmail

С этого момента новая программа Sendmail становится одним из дополнительных пакетов. Ее нужно активизировать в файле *mailer.conf*, как это делается для Postfix или Qmail. Поскольку в ходе установки «порта» двоичный файл Sendmail был установлен как */usr/local/sbin/sendmail*, новый файл *mailer.conf* должен выглядеть так:

```
sendmail /usr/local/sbin/sendmail
send-mail /usr/local/sbin/sendmail
mailq /usr/local/sbin/sendmail
newaliases /usr/local/sbin/sendmail
hoststat /usr/local/sbin/sendmail
purgestat /usr/local/sbin/Sendmail
```

Сборка sendmail.cf

Хотя новая версия программы Sendmail скомпилирована с поддержкой SASL, все же придется сконфигурировать sendmail(8), чтобы она могла принимать результаты аутентификации SASL. Кроме того, во избежание передачи имен пользователей и паролей в явном виде, следует использовать SSL. Добавьте в файл */etc/mail/<имя_хоста>.mc* такие строки:

```
TRUST_AUTH_MECH(`GSSAPI DIGEST-MD5 CRAM-MD5 LOGIN')dnl
define(`confAUTH_MECHANISMS', `GSSAPI DIGEST-MD5 CRAM-MD5 LOGIN')dnl
define(`CERT_DIR', ❶`/usr/local/etc/certs')dnl
define(`confCACERT_PATH', `CERT_DIR')dnl
define(`confCACERT', `CERT_DIR/hostname.pem')dnl
define(`confSERVER_CERT', `CERT_DIR/hostname.pem')dnl
define(`confSERVER_KEY', `CERT_DIR/hostname-key.pem')dnl
define(`confCLIENT_CERT', `CERT_DIR/hostname.pem')dnl
define(`confCLIENT_KEY', `CERT_DIR/hostname-key.pem')dnl
define(`confAUTH_OPTIONS', `A p y')dnl
DAEMON_OPTIONS(`Port=smtp, Name=MTA')dnl
DAEMON_OPTIONS(`Port=smtps, Name=TLSMTA, M=s')dnl
```

Эти настройки предписывают Sendmail аутентифицировать пользователей, прежде чем принимать от них почту.

Кроме всего прочего, здесь указывается местоположение сертификата, который может потребоваться изменить, чтобы он соответствовал общесистемному сертификату. Например, если вы используете Dovecot, как это предлагается в следующем разделе, то сможете упростить конфигурацию, поместив свои сертификаты в каталог */etc/ssl/cert*. В конце концов, одним и тем же сертификатом могут пользоваться сразу несколько сервисов.

Внеся необходимые изменения в конфигурационные файлы, выполните команду `make all install` в каталоге */etc/mail*, чтобы собрать и установить новый файл *sendmail.cf*.

Проверка SASL

Перезапустите Sendmail командой `/etc/rc.d/sendmail restart` – и ваша инсталляция Sendmail должна быть готова к приему почты через аутентификацию SASL. Простейший способ проверить – настроить почтовый клиент на использование SASL и попытаться отправить почту на хост, расположенный за пределами вашей сети. Специально для таких проверок я держу почтовый ящик на Yahoo!; если мне удастся отправить почту на Yahoo!, почти наверняка я смогу отправить ее и в любое другое место.

В большинстве клиентов электронной почты параметр, отвечающий за включение SASL, выглядит как флажок примерно с такой подписью: «Сервер исходящей почты требует аутентификации». Настройте клиент на использование той же аутентификационной информации, что

и при получении входящей почты. После этого клиент сможет отправлять почту без проблем, а попытки других клиентов, у которых аутентификация не настроена, будут отвергаться.

Стоп, у вас ведь еще нет имени пользователя и пароля для получения входящей почты! Теперь, зная, как Sendmail отправляет и получает почту, перейдем к изучению принципа доставки почты от сервера к клиенту.

IMAP и POP3

Несмотря на то что вы можете войти непосредственно в систему FreeBSD и прочитать свою почту, ваши клиенты и конечные пользователи наверняка предпочтут пользоваться удобными клиентскими программами чтения электронной почты, такими как Thunderbird или Eudora. Операционная система FreeBSD поддерживает работу с подобными клиентскими программами через различные «порты» и пакеты. Для передачи электронной почты клиенту служат два стандартных протокола – IMAP и POP3.

Протокол интерактивного доступа к электронной почте (*Internet Message Access Protocol, IMAP*) позволяет пользователям настольных компьютеров синхронизировать свою почту с центральным почтовым сервером. Синхронизация почтовых ящиков на сервере IMAP может выполняться с нескольких клиентских машин: так, имея настольный компьютер и ноутбук, вы сможете получить всю свою электронную почту на обе машины. Протокол IMAP очень популярен в корпоративных сетях, где клиентам разрешается хранить электронную почту на сервере. Но интернет-провайдерам я не рекомендую использовать IMAP для организации службы электронной почты. При использовании IMAP клиенты ожидают, что провайдер будет хранить их почту вечно, и разочаровываются, даже негодуют, когда сбои в работе аппаратного обеспечения демонстрируют ошибочность этих ожиданий. Возложите ответственность за сохранность электронной почты на самих клиентов. Тем не менее протокол IMAP отлично подходит для использования в офисе, поэтому ниже в этой главе мы рассмотрим использование IMAP через SSL.

Почтовый протокол (*Post Office Protocol, POP*) версии 3 – это очень простой протокол, позволяющий пользователям загружать электронную почту на настольный компьютер. Протокол POP3 очень популярен у интернет-провайдеров и на небольших предприятиях из-за его простоты. Большинство IMAP-серверов кроме IMAP могут предоставлять и услуги POP3, поэтому для удовлетворения всех наших потребностей достаточно будет одного демона. Мы рассмотрим работу POP3 через SSL, что обеспечивает защиту имени пользователя и пароля, а также содержимое самого почтового отправления.

Самые простые серверы IMAP, такие как imap-wu, позволяют лишь синхронизировать почтовые клиенты со стандартным почтовым буфе-

ром UNIX в каталоге */var/mail*. Другие, такие как Cyrus IMAP, поддерживают возможность работы с десятками тысяч пользователей. Сервер Dovecot IMAP занимает промежуточное положение. Он обладает богатыми возможностями, достаточными для большинства применений, и при этом настолько прост, что для его обслуживания не требуется нанимать дополнительный персонал. Кроме того, автор сервера Dovecot предлагает награду в тысячу евро тому, кто первым «укажет брешь в защите Dovecot, которую можно использовать удаленно». Многие уже потратили массу времени в попытках получить награду, но это еще никому не удалось.

Установка Dovecot

Найти сервер Dovecot можно в каталоге */usr/ports/mail/dovecot*. Dovecot может взаимодействовать с серверами LDAP, различными типами баз данных и системами аутентификации; кроме того, он поддерживает множество других конфигурационных параметров. Однако для создания типичного почтового сервера, обслуживающего пользователей системы FreeBSD, нам достаточно установить Dovecot в стандартной конфигурации. Перейдите в каталог «порта» и запустите команду `make all install clean`.

Вместе с сервером Dovecot устанавливаются целый пакет документации (в каталог */usr/local/share/doc/dovecot*) и примеры конфигурационных файлов (в каталог */usr/local/etc*). Подобно серверу Apache, у сервера Dovecot масса конфигурационных параметров, которые лучше не трогать без особой нужды. Все свое внимание мы сосредоточим на том, чтобы организовать службы POP3 и IMAP через SSL. Можно реализовать услуги электронной почты без SSL или какой-либо другой криптографической защиты, но в современном Интернете это будет не самый лучший выбор.

Конфигурирование сервера Dovecot

Скопируйте пример конфигурационного файла */usr/local/etc/dovecot-example.conf* в */usr/local/etc/dovecot.conf* и откройте его в текстовом редакторе. Верите или нет, но чтобы обеспечить безопасность функционирования сервера Dovecot в системе FreeBSD, требуется внести не так уж много изменений.

По умолчанию Dovecot предлагает реализацию служб IMAP и POP3 без шифрования. Это общепринятый стандарт в современном Интернете, но все современные почтовые клиенты поддерживают возможность передачи аутентификационной информации и сообщений через SSL-соединение. Запись `protocols` определяет протоколы, предлагаемые сервером. Измените оба имени протоколов IMAP и POP3 на имена версий этих протоколов с поддержкой SSL, добавив символ `s` в конец имени каждого из них:

```
protocols = imaps pop3s
```

Теперь с помощью параметров `ssl_cert_file` и `ssl_key_file` следует определить место, где Dovecot будет хранить сертификаты SSL. В состав программного обеспечения Dovecot входит сценарий, создающий соответствующие самозаверенные сертификаты, впрочем, вы можете использовать заверенные сертификаты, как описано в главе 9.

```
ssl_cert_file = /etc/ssl/certs/dovecot.pem
ssl_key_file = /etc/ssl/private/dovecot.pem
```

Если изменить эти значения по умолчанию, придется отредактировать сценарий создания самозаверенных сертификатов, чтобы указать правильные каталоги сохранения. Фанаты FreeBSD определенно могут вам порекомендовать изменить сценарий так, чтобы он сохранял сертификаты в каталоге */usr/local/etc/ssl*, но тогда после каждого обновлении Dovecot вам придется снова изменять сценарий. Местоположение файлов SSL не так уж важно, при условии, что вы документируете параметры настройки системы.

Создание сертификата SSL в Dovecot

Можно создавать самозаверенные сертификаты, как описано в главе 9, но у Dovecot есть сценарий командного интерпретатора, создающий необходимые сертификаты с учетом потребностей сервера Dovecot. Сценарий и конфигурационный файл находятся в каталоге */usr/local/share/dovecot*. Откройте в текстовом редакторе конфигурационный файл *dovecot-openssl.cnf*. Параметры настройки должны выглядеть очень знакомыми по главе 9, но поскольку мы ушли вперед почти на полкниги, еще раз вкратце рассмотрим их здесь.

В начале файла находятся такие настройки, как тип сертификата и степень его сложности. Оставьте этим параметрам значения по умолчанию. Если потребуется, вы сможете изменить их позднее.

Параметр `C` – это двухбуквенный код страны. Я живу в Соединенных Штатах, поэтому присваиваю значение `C=US`.

Параметр `ST` – это название штата или области, а `L` – название населенного пункта. Я живу в городе Детройт, штат Мичиган, поэтому присваиваю следующие значения:

```
ST=Michigan
L=Detroit
```

Параметр `O` – это название организации. Если вы работаете на какую-то компанию, укажите здесь ее название. Если нет – вставьте свое имя.

Параметр `OU` – это подразделение организации или отдел, отвечающий за поддержку данной системы. Обычно здесь я указываю `Email Team` (группа поддержки электронной почты).

Параметр `common name` – это имя хоста сервера, возвращаемое при обратном поиске в DNS, которое клиенты будут использовать для подключения к серверу. Это имя в большинстве случаев совпадает с именем хоста сервера. Например, имя моего хоста – *bewilderbeast.blackhelicopters.org*.

Параметр `emailAddress` – это адрес электронной почты человека, отвечающего за работу сервера. Это может быть и групповой адрес, например *helpdesk@mycompany.com*.

Это все, что требуется серверу Dovecot для создания собственного самозаверенного сертификата. Теперь достаточно просто запустить сценарий *mkcert.sh*:

```
# /usr/local/share/dovecot/mkcert.sh
```

Вот и все! Ваш самозаверенный сертификат готов.

Запуск сервера Dovecot

Активизируйте Dovecot в файле */etc/rc.conf*, добавив параметр `dovecot_enable="YES"`, а затем запустите его командой `/usr/local/etc/rc.d/dovecot start`. Файл */var/log/maillog* покажет вам, что сервер запустился и инициализировал механизм поддержки SSL.

Теперь у вас имеется работающий сервер Dovecot! Настройте клиент и попробуйте обратиться к серверу. Не забудьте указать клиенту, что соединение должно устанавливаться через SSL! Любые возникшие ошибки будут отражены в файле */var/log/maillog*.

Осталось самое сложное – отделить ошибки на стороне сервера от ошибок на стороне клиента. У каждого клиента электронной почты есть свои недостатки и достоинства. Отделить ошибки, связанные с клиентом, от ошибок сервера бывает весьма непросто, особенно в случае незнакомого программного обеспечения. При решении проблем, связанных с клиентами, важно уметь проверять почтовую учетную запись вообще без использования клиента.

Проверка POP3S

Мы настроили службу POP3 на использование SSL, а это означает, что мы не можем использовать команду telnet, чтобы установить прямое соединение с сервером. Однако в главе 9 мы видели, что команду openssl(1) можно использовать для соединения с удаленными портами способом, напоминающим telnet. Эту команду можно использовать для доступа к интерфейсу SSL POP3 почтового сервера. Сервис POP3S прослушивает порт 995. Ниже приводится пример сеанса связи с интерфейсом POP3S моего почтового сервера:

```
# openssl s_client -connect bewilderbeast.blackhelicopters.org:995
CONNECTED(00000003)
```

```
...
+OK Dovecot ready.
```

Теперь, когда соединение установлено и Dovecot готов к общению, идентифицируйте себя с помощью команды `user`.

```
user mwlucas
+OK
```

С помощью команды `pass` введите свой пароль. Пароль будет отображаться на экране в открытом текстовом виде, поэтому позаботьтесь, чтобы никто не подглядывал!

```
pass n0tmyr3alpassw0rd!
+OK Logged in.
```

Соединение установлено! Теперь проверим наличие почтовых сообщений.

```
list
+OK 2 messages:
1 1391
2 4258
```

Сообщение 1 имеет размер 1391 байт, а сообщение 2 – 4258 байтов. Чтобы просмотреть сообщение, нужно ввести команду `retr` с номером сообщения.

Если вам удалось выполнить все эти команды, можно быть уверенным в работоспособности POP3S. Настройка произвольного клиента для работы с POP3S выходит за рамки этой книги.

Проверка IMAPS

Проверка работоспособности IMAPS мало чем отличается от проверки POP3S. Проверив IMAPS вручную, можно сразу же отсечь все проблемы, связанные с настройкой клиентских программ. Протокол IMAPS намного сложнее, чем POP3, и некоторые из его команд, нужные вам, выглядят очень неуклюже. (Мой технический редактор признал, что я храбрее его, поскольку отважился выполнить проверку IMAP вручную. Ого!) Сервис IMAPS прослушивает порт 993, но команда openssl(1) практически идентична той, что использовалась для проверки POP3S:

```
# openssl s_client -connect bewilderbeast.blackhelicopters.org:993
CONNECTED(00000003)
...
* OK Dovecot ready.
```

Обратите внимание: сервис IMAPS использует символ звездочки (`*`) вместо плюса (`+`), применяемого в сервисе POP3S. Все команды IMAP начинаются с числа. Например, для идентификации и аутентификации следует ввести команду `01 LOGIN` с именем пользователя и паролем:

```
01 LOGIN mwlucas n0tmyr3alpassw0rd!
01 OK Logged in.
```

Соединение с сервером IMAP установлено. Теперь с помощью команды `02 LIST` можно выяснить, какие каталоги имеются в учетной записи:

```
02 LIST "" *
* LIST (\NoInferiors \UnMarked) "/" "Junk E-mail"
* LIST (\NoInferiors \UnMarked) "/" "mwlucas"
* LIST (\NoInferiors \UnMarked) "/" "INBOX"
02 OK List completed.
```

Пока все идет хорошо. Давайте посмотрим, что находится в каталоге `INBOX`, с помощью команды `03 SELECT`:

```
03 SELECT INBOX
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft \*)] Flags
permitted.
❶ * 2 EXISTS
* 0 RECENT
* OK [UIDVALIDITY 1185130816] UIDs valid
* OK [UIDNEXT 3] Predicted next UID
```

В моей папке по умолчанию имеются 2 сообщения ❶. Здесь видно, что следующему сообщению будет присвоен номер 3, поэтому текущие сообщения имеют номера 1 и 2. Посмотрим, можно ли получить первое сообщение, введя следующую команду:

```
06 UID fetch 1:1 (UID RFC822.SIZE FLAGS BODY.PEEK[])
```

В результате первое сообщение, находящееся в папке по умолчанию, должно появиться на экране. (Если к этому моменту вы уже задействовали IMAP, скорее всего сообщение будет иметь номер больше 1.) Наконец, с помощью команды `07 LOGOUT`, отсоединитесь от сервера.

Если вам удалось выполнить все эти команды, можно быть уверенным в работоспособности IMAP через SSL. Проверьте настройки клиента.

Программы Sendmail и Dovecot можно изучать и изучать, но уже сейчас вы сможете реализовать базовую почтовую службу для корпоративных клиентов и клиентов. А теперь перейдем к изучению возможности реализации веб-сервера в операционной системе FreeBSD.

17

Веб и FTP-сервисы

Интернет возник в 1970-х, но по-настоящему популярным стал в середине 1990-х с появлением веб-страниц. Корпорация Netscape взяла веб-броузер Mosaic и превратила его в коммерческий продукт. Это стало началом не прекращающейся и поныне революции в отображении информации и обмене ею. Многие компании появлялись на свет и исчезали, однако эра взаимодействия «человек–человек» (person-to-person) началась с появлением веб-броузера Netscape. Такие технологии, как peer-to-peer, расширили Интернет еще больше, однако Сеть (Web) – это первое, что ассоциируется со словом «Интернет».

О производительности веб-сервера FreeBSD ходят легенды. Многие годы некоторые подразделения Microsoft отдавали предпочтение FreeBSD, а не собственной платформе Windows, Microsoft даже выпустила набор инструментов «.NET for FreeBSD» в виде условно бесплатного программного обеспечения. Система FreeBSD работает в Yahoo!, а также в великом множестве других крупных веб-компаний, предъявляющих высокие требования к платформе. Компания Netcraft в своих обзорах самых надежных услуг по организации веб-узлов постоянно свидетельствует, что примерно треть из них предоставляется на базе операционной системы FreeBSD.

Вы можете самостоятельно создать собственный высоконадежный веб- и FTP-сервер на базе FreeBSD.

Как работает веб-сервер

Алгоритм работы веб-сервера довольно прямолинеен: веб-броузер открывает соединение с веб-сервером и запрашивает страницу, веб-сервер выдает ее и закрывает соединение. Это довольно просто. Дело значительно усложняется, когда вы начинаете применять серверные модули, динамические страницы и т. д.

В Сети применяется протокол передачи гипертекста (Hypertext Transfer Protocol, HTTP) – очень простой протокол, как и POP3. За последние несколько лет в HTTP добавлены функции, сделавшие его более сложным, однако базовые операции HTTP настолько просты, что их можно выполнять вручную. Давайте попробуем. С помощью telnet(1) подключимся к серверному порту 80 и наберем команду `GET /`.

```
# telnet www.blackhelicopters.org 80
Trying 198.22.63.8...
Connected to www.blackhelicopters.org.
Escape character is '^]'.
GET http://www.blackhelicopters.org/
<HTML>
<body bgcolor="black">

<center><font color="white">Nothing to see here.<br>This is not the
site you're looking for.</br>

</html>
Connection closed by foreign host.
```

Если вы когда-нибудь видели HTML-страницу, то вывод этой команды покажется вам очень знакомым. Если нет, то когда в следующий раз загрузите веб-страницу, попробуйте в меню своего броузера выбрать пункт View \ Source (Вид \ Исходный код). Тогда вы увидите, что привлекательная страница, появляющаяся в броузере, создается средствами HTML. Если страницы не загружаются с веб-сервера, в нем могут быть неполадки.

Веб-сервер Apache

Заглянув в каталог */usr/ports/www*, можно найти несколько различных «портов» веб-серверов. Здесь есть dhttpd, thttpd, bozohttpd, XS-HTTPD и многие другие веб-серверы, каждый из которых обладает своими особенностями. Например, tclhttpd – это веб-сервер, полностью реализованный на языке Tcl. Если вам нужен специализированный веб-сервер, вполне вероятно, что вы найдете его здесь. Мы сосредоточимся на наиболее популярном веб-сервере Apache.

Операционная система FreeBSD включает несколько «портов» различных версий Apache, но большая их часть – это устаревшие версии. Если у вас имеется приложение, которому требуется Apache версии 1.3 с определенным набором модулей, FreeBSD предоставит его. Однако для вновь создаваемого веб-сервера лучше выбрать последнюю доступную версию. На момент написания этих строк последней была версия Apache 2.2. Сервер Apache обладает очень широкими функциональными возможностями, такими как поддержка LDAP и баз данных SQL, однако мы сосредоточимся только на основных его функциях. Apache 2.2 легко устанавливается как из «порта» (*/usr/ports/www/apache22*), так и из пакета.

Конфигурационные файлы Apache

Основные конфигурационные файлы Apache 2.2 расположены в */usr/local/etc/apache22*. И хотя за эти годы конфигурация Apache претерпела значительные изменения, в нынешнем виде она представляет неплохой компромисс между сложностью и модульностью. Ключевыми конфигурационными файлами являются *httpd.conf*, *magic* и *mime.types*.

mime.types

Файл *mime.types* содержит список всех стандартных типов файлов и их идентифицирующие характеристики. Когда веб-сервер отправляет файл клиенту, он должен идентифицировать тип файла, чтобы тот мог выполнить соответствующие действия. Вам наверняка не захочется, чтобы веб-броузер интерпретировал потоковое видео как HTML! Соответствия, хранимые в *mime.types*, предоставляют серверу Apache информацию, необходимую для правильного указания этих типов. Файл *mime.types* содержит практически все возможные типы файлов. Поэтому, даже если в документации к какой-либо программе говорится о необходимости добавить какую-либо информацию в файл *mime.types*, прежде чем вносить какие-либо изменения, убедитесь, что эта информация действительно отсутствует. Скорее всего, вам никогда не придется редактировать этот файл.

magic

Поскольку файл *mime.types* не может учитывать все возможные типы файлов, модуль *mime_magic*, встроенный в Apache, применяет файл *magic* для идентификации неизвестных файлов. На протяжении своей карьеры системного администратора я неоднократно собирался что-то изменить в файле *magic* и каждый раз оказывался неправ.

httpd.conf

Файл *httpd.conf* – это место, где происходит все самое интересное. Этот файл снабжен подробными комментариями и содержит несколько сотен строк, поэтому подробно здесь он не рассматривается. Если вы действительно хотите детально изучить Apache, то сможете отыскать несколько толстых книг на эту тему. Мы рассмотрим лишь те конфигурационные параметры, которые вам придется изменять, а также несколько наиболее популярных параметров. В любом случае, не изменяйте ничего в этом файле, если точно не представляете себе, что делаете.

Подкаталоги *Includes*, *extra* и *envvars.d* обеспечивают дополнительную функциональность сервера Apache, позволяя отделить базовую функциональность сервера от часто изменяющихся дополнений. Например, раскомментировав соответствующую строку в файле *httpd.conf*, можно активировать любую функцию из каталога *extra*. Мы узнаем, как это делается и как использовать эту архитектуру для повышения уровня безопасности.

Базовая конфигурация Apache

Сервер Apache поддерживает множество различных особенностей в своей сложной конфигурации, тем не менее настройка и запуск сервера выполняются достаточно просто. Прежде всего следует определить параметр `apache_22_enable="YES"` в файле */etc/rc.conf*. После этого настройка вашего первого веб-сайта заключается в том, чтобы внести небольшое число изменений в файл *httpd.conf*.

Путь к корневому каталогу сервера

Параметр `ServerRoot` определяет каталог, который содержит все файлы веб-сайта и серверных программ. Когда в файле *httpd.conf* указывается ссылка на файл, Apache автоматически присоединяет к ней спереди значение параметра `ServerRoot`, при условии, что строка пути к файлу не начинается с символа слэша (/).

```
ServerRoot "/usr/local"
```

Например, ссылка на каталог *libexec/apache22*, с учетом значения по умолчанию параметра `ServerRoot` в действительности указывает на каталог */usr/local/libexec/apache22*. С другой стороны, элемент */var/log/httpd.log* останется без изменений. Я не рекомендую изменять значение этого параметра просто потому, что большая часть конфигурации по умолчанию сервера Apache опирается на значение параметра `ServerRoot`, равное */usr/local*.

Listen

Параметр `Listen` определяет, к каким TCP-портам или IP-адресам будет привязан Apache. По умолчанию используется порт 80 (стандартный порт для HTTP) на всех IP-адресах, имеющихся на локальной машине:

```
Listen 80
```

Можно указать отдельные IP-адреса:

```
Listen 192.168.8.44
```

Объединив эти две возможности, можно указать серверу прослушивать нестандартный порт на единственном IP-адресе:

```
Listen 192.168.8.44:8080
```

Добавляя дополнительные инструкции `Listen`, можно обеспечить доступность Apache на любом количестве портов или IP-адресов системы.

Пользователь и группа

Эти параметры задают непривилегированного пользователя и группу, под которыми запускается Apache. В системе FreeBSD предусмотрены пользователь www и группа www. Это специальные учетные записи без привилегий, предназначенные для веб-сервера. Хотя иногда встречаются рекомендации запускать Apache от имени root, не поступайте так

ни по каким причинам. Если злоумышленник взломает ваш веб-сервер, в качестве дополнительной награды он получит права root в вашей системе!

```
User www
Group www
```

Почтовый адрес администратора

Веб-сервер должен знать адрес электронной почты системного администратора. Apache автоматически вставляет этот адрес в различные сообщения об ошибках.

```
ServerAdmin webmaster@blackhelicopters.org
```

Спамботы без труда могут извлечь этот адрес, поэтому предусмотрите хотя бы базовый уровень защиты этого адреса от спама.

Имя сервера

Это имя веб-сайта. Оно должно представлять собой реальное имя хоста и иметь соответствующую запись в DNS. В противном случае запустить Apache не удастся. Однако во время тестирования вместо записи DNS можно применять запись в */etc/hosts*. Здесь также можно указать IP-адрес.

```
ServerName www.absolutefreebsd.com
```

Путь к корневому каталогу документов

Корневой каталог документов – это место, предназначенное для размещения HTML-файлов, из которых фактически состоит веб-сайт. Сайт по умолчанию содержит простое сообщение о том, что веб-сервер работает. Это, конечно, хорошо, но такое сообщение не привлечет много посетителей. Если в этот каталог поместить свои документы HTML, при следующем же обновлении коллекция «портов» заметит изменения и сообщит об этом. В идеале следует выбрать другой каталог для своих документов. Я предпочитаю размещать свои веб-сайты в каталоге */var/www*, так как это позволяет сохранять содержимое каталога */usr* неизменным.

```
DocumentRoot "/var/www/mywebsitename"
```

Протоколы сервера Apache

Сервер Apache обладает довольно сложной системой протоколирования и позволяет выбирать желаемую степень детализации сообщений, которые будут появляться в файлах протоколов. Вы можете выработать свои собственные форматы записей для файла протокола, однако я настоятельно рекомендую использовать один из форматов, принятых по умолчанию. Многие инструменты сторонних производителей позволяют анализировать протоколы сервера Apache, но если вы измените формат записей, эти инструменты вам не помогут.

Основные протоколы

Самый простой протокол – это протокол ошибок. Если у вас на одном и том же сервере работает несколько веб-сайтов, рекомендую включить имя сайта в имя файла протокола с ошибками:

```
ErrorLog /var/log/httpd-имясайта-error.log
```

Наибольший интерес для большинства веб-администраторов представляет протокол со списком запрашиваемых файлов. Для протоколирования обращений к сайту необходимо определить формат протокола и объявить местоположение файла протокола в этом формате. Сервер Apache 2.2 включает три формата:

- Формат *common* подразумевает протоколирование IP-адреса клиента, времени запроса и имени запрашиваемого файла.
- Формат *combined* включает все то же самое, что и формат *common*, а также сайт, с которого клиент перешел на данный сайт, и информацию о веб-броузерах клиентов.
- Формат *combinedio* включает все то же самое, что и формат *combined*, а также фактическое число байтов, переданных по сети. В это число включаются заголовки и прочая метаинформация, передаваемая вместе с документом.

Выбрав один из вышеуказанных (или определив свой) форматов протокола, его можно указать в инструкции `CustomLog`:

```
CustomLog /var/log/httpd-имясайта-access.log combinedio
```

Для каждого сайта можно определить несколько значений параметра `CustomLog` и использовать каждый из протоколов для разных целей. Допустим, вы применяете модуль Apache, предоставляющий особые функции, действие которых необходимо протоколировать. В то же время вы не хотите мешать работе программ, выполняющих анализ веб-протокола, которые используются вашими клиентами. В таком случае инструкция `CustomLog` позволит выполнять для одного и того же сайта запись в несколько протоколов.

Ротация протоколов

Есть хорошее эмпирическое правило, согласно которому каждые 10 000 запросов увеличивают файл протокола на 1 Мбайт. Эта величина может показаться несущественной для маленького сайта, но даже маленький веб-сайт может получить огромное число запросов в течение нескольких месяцев. Мой (совершенно убогий) личный веб-сайт получает примерно 60 000 запросов в год.[1] Нагруженный сайт может получать значительное число запросов в минуту. Если не производить

[1] Это сказано не для того, чтобы увеличить посещаемость моей домашней страницы, честно. Если вы почувствовали желание посетить мой веб-сайт, лучше прогуляйтесь и подышите свежим воздухом.

ротацию протоколов, вскоре окажется невозможным отыскать в них нужную информацию.

Сервер Apache реализует работу с протоколами не так изящно, как большинство других серверных программ, главным образом потому, что основная цель Apache – обслуживать тысячи и тысячи пользователей одновременно. Если для ротации файлов протоколов вы просто воспользуетесь услугами newsyslog(8), как это делается в случае других программ, Apache начнет повреждать свои собственные протоколы. Тем не менее, Apache поддерживает работу с программами ротации протоколов. Я рекомендую обслуживать протоколы с помощью программы rotatelogs(8), которая входит в состав Apache. Для этого в операторах определения протоколов вместо имени файла следует указать вызов rotatelogs(8) через конвейер (pipeed call). Например:

```
ErrorLog "|/usr/local/sbin/rotatelogs /var/log/httpd-имясайта-error-log 86400"
```

Вместо имени файла здесь присутствует вызов программы rotatelogs(8), которой передается базовое имя файла. Завершающее число – это количество секунд между ротациями файла протокола. 86 400 секунд – это одни сутки.

Подобным образом можно применять программу rotatelogs(8) для протокола обращений, не забыв при этом указать формат протокола в конце строки определения. Чтобы избежать лишнего ввода с клавиатуры, можно использовать путь из параметра `ServerRoot`.

```
CustomLog "|sbin/rotatelogs /var/log/httpd-имясайта-access-log 86400" combined
```

В следующих примерах будет рассматриваться только оператор `ErrorLog`, потому что эти записи короче и их проще вводить вашему ленивому автору. Но все они точно так же могут применяться и к протоколу обращений.

Ротацию протоколов можно производить и на основе их размеров, что может вызвать выполнение через разные интервалы времени, но сделает вашу жизнь проще. Ниже показано, как выполнять ротацию протокола ошибок через каждые 5 Мбайт:

```
ErrorLog "|sbin/rotatelogs /var/log/httpd-имясайта-error-log 5M"
```

По умолчанию rotatelogs(8) присваивает каждому файлу протокола имя, включающее время начала протокола, исчисляемое в секундах от начала эпохи.[1] Хотя вы и сами без труда переведете эпохальное время в более понятное человеку представление с помощью команды date(1), но можете и заставить rotatelogs сразу использовать дружелюбный формат значений даты и времени в именах файлов протоколов. Так можно обеспечить включение времени создания файла в имя файла протокола:

[1] В данном случае под «началом эпохи» понимается «начало эпохи UNIX» – 1 января 1970 года; системное время в UNIX всегда измеряется в секундах от 00:00 этой даты. – *Прим. науч. ред.*

```
ErrorLog "|sbin/rotatelogs /var/log/httpd-имясайта-error-log.%Y-%m-%d-
%H_%M_%S 86400"
```

Если ваши особые требования к протоколированию не может удовлетворить rotatelogs(8), обратите внимание на cronlog (*/usr/ports/sysutils/cronolog*) или httplog (*/usr/ports/sysutils/httplog*). Правда, мне это никогда не требовалось.

Модули Apache

Сервер Apache состоит из модулей, почти так же, как и ядро FreeBSD. Apache может обслуживать такие разные расширения, как Microsoft FrontPage Extensions, языки сценариев (включая PHP) и встроенный Perl. В версии Apache 2.2 базовые функции содержатся в модулях. Вы можете запретить использование некоторых из них, но это нарушит стандартное поведение веб-сервера и вызовет массу неприятностей. Вы также можете обнаружить модули для сжатия страниц перед отправкой, позволяющие значительно снизить трафик. Каждый модуль представляет собой «порт» в */usr/ports/www*. Имена файлов с «портами» модулей начинаются с префикса *mod_*, например mod_gzip.

Загрузка и выгрузка модулей Apache производятся с помощью конфигурационного файла. Операторы загрузки модулей выглядят так:

```
LoadModule authn_file_module libexec/apache22/mod_authn_file.so
LoadModule authn_dbm_module libexec/apache22/mod_authn_dbm.so
LoadModule authn_anon_module libexec/apache22/mod_authn_anon.so
...
```

Каждая запись состоит из оператора `LoadModule`, имени модуля и файла, в котором находится этот модуль. Пути к файлам указаны относительно значения параметра `ServerRoot`, то есть данные файлы находятся в каталоге */usr/local/libexec/apache22*.

Модули Apache и модули ядра

Хотя модули ядра не имеют никакого отношения к модулям Apache, тем не менее операционная система FreeBSD включает модули ядра, предназначенные для оптимизации веб-серверов. Если клиент отправляет длинный запрос HTTP, ожидание прибытия всего запроса может занять много времени (с точки зрения компьютера). *Фильтр приема HTTP* (*HTTP accept filter*) буферизует входящий HTTP-трафик в ядре, пока не будет получен весь запрос. Благодаря такой буферизации веб-сервер не тратит время на ожидание получения всего запроса – он получает запрос целиком и тут же может заняться его обработкой. Чтобы активизировать фильтр приема HTTP, необходимо добавить параметр `apache22_http_accept_enable="YES"` в файл */etc/rc.conf*.

Вот несколько популярных модулей Apache, доступных в коллекции «портов». Кто-то наверняка считает популярными другие модули – что ж, это его право.[1] Большинство этих модулей можно найти в */usr/ports/www*.

mod_bandwidth Позволяет управлять полосой пропускания сайта, что очень удобно для организации виртуальных серверов, когда клиентам выделяется определенный объем трафика в месяц.

mod_dtcl Встраивает в Apache интерпретатор языка Tcl, что обеспечивает быструю обработку приложений на языке Tcl.

mod_fastcgi Ускоряет работу CGI-сценариев.

mod_gzip Улучшает время отклика веб-сайтов и уменьшает объем передаваемого трафика за счет сжатия данных. Рекомендую установить этот модуль.

mod_mp3 Превращает Apache в сервер потокового MP3.

mod_perl2 Встраивает интерпретатор Perl в сервер Apache lkz быстрой обработки приложений на языке Perl.

mod_python Встраивает интерпретатор Python в сервер Apache для быстрой обработки приложений на языке Python.

mod_ruby Встраивает интерпретатор Ruby в сервер Apache для быстрой обработки приложений на языке Ruby.

mod_webapp-apache2 Подключает Apache к Tomcat – серверу Java-приложений.

php5 Обеспечивает поддержку популярного языка веб-сценариев PHP. (Этот модуль находится в каталоге */usr/ports/lang/php5*.) При установке PHP вам будет предложено заодно установить и MySQL, еще один популярный инструмент.

Вы сможете отыскать и другие модули Apache, разбросанные по всей коллекции «портов». Обнаружив у себя проблемы, связанные с Apache, поищите в Сети; есть шанс, что необходимое вам программное обеспечение уже существует.

После установки многие из этих модулей добавляют свои параметры настройки в *httpd.conf*. Если вы используете систему управления версиями для сохранения файла *httpd.conf* (как и должно быть!), значит перед установкой «порта» необходимо захватить *httpd.conf* (check out). Если «порт» не добавляет никаких настроек, прочитайте документацию к этому модулю; возможно, на то есть веские причины. Не исключено, что прежде чем приступать к конфигурированию Apache, вам придется принять определенные решения относительно желаемого поведения программного обеспечения.

---

[1] Да, я позволяю людям не соглашаться со мной. Даже если они неправы.

Каталоги и права доступа

В Apache есть много отличных функций, однако не стоит включать их всегда и везде – неряшливая настройка может чрезмерно увеличить объем информации, передаваемой с вашего веб-сайта, и даже привести к взлому сервера. Права на доступ к функциям Apache предоставляются по каталогам. Конфигурация немного похожа на XML: есть метка `Directory` в угловых скобках, список прав доступа и параметров, а затем закрывающая запись `Directory` с обратным слэшем. Любые значения или параметры между открывающим и закрывающим операторами `Directory` относятся к этому каталогу:

```
<Directory /path/to/files>
    ...параметры и настройки...
</Directory>
```

По умолчанию Apache применяет весьма ограничивающие права и значения параметров. Например, в файле *httpd.conf* в начале раздела, описывающего каталоги, можно увидеть следующую запись:

```
<Directory />
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>
```

Apache разрешает пользователям загружать на сервер свои конфигурационные файлы, которые изменяют параметры настройки сервера, параметры защиты паролями, типы MIME и т. д. Строка `AllowOverride None` означает, что пользователи не смогут установить те или иные параметры в указанном каталоге, если это явно не разрешено. Инструкции `Order` и `Deny` означают, что ни один системный каталог не доступен из Сети. Это *стратегия отказа в доступе по умолчанию*. Если иное не оговорено, каждый каталог на сервере имеет такие же права доступа.

Теперь, когда по умолчанию запрещено все, можно выдать явные разрешения на использование необходимых функциональных возможностей. Ниже приводятся некоторые полезные настройки.

Управление доступом на основе IP-адресов

Параметры `Allow` и `Deny` задают IP-адреса и имена хостов, с которых разрешен доступ к содержимому каталога. Адрес клиента сравнивается со списками `Allow` и `Deny` в порядке, указанном в инструкции `Order`. Далее, в зависимости от результата сравнения, Apache разрешает или запрещает доступ к файлам каталога. Если порядок `Order` определен как `deny,allow`, то по умолчанию доступ открыт, если нет запрещающей инструкции `Deny`. Если же порядок определен как `allow,deny`, то по умолчанию доступ запрещен, если нет разрешающей инструкции `Allow`. Применяется последнее совпавшее правило.

```
Order allow,deny
Allow from all
```

В этом примере для каталога *DocumentRoot* определяется стратегия запрета доступа по умолчанию, а затем добавляется правило с инструкцией `Allow`, которое разрешает всем хостам доступ к сайту. Как и в случае с TCP Wrappers, здесь можно обозначить все хосты с помощью ключевого слова `all`. Можно заблокировать отдельные сайты, добавив инструкцию `Deny`:

```
Order allow,deny
    Deny from 192.168.0.0/16
    Allow from all
```

Здесь был заблокирован доступ к сайту с определенных IP-адресов. Точно так же здесь можно было бы использовать имена хостов, если при обеспечении безопасности веб-сервера вы готовы положиться на результаты обратного поиска в DNS:

```
Order allow,deny
    Deny from *.absolutefreebsd.com
    Allow from all
```

В этом случае все, что я должен сделать, чтобы получить доступ к сайту, – это замаскироваться под машину, которая при обратном поиске DNS не будет определяться как принадлежащая домену *absolutefreebsd.com*. Изменить мой обратный DNS намного проще, чем IP-адрес.

И наоборот, вы с легкостью сможете ограничить доступ к внутреннему веб-сайту, разрешив его посещение только с IP-адресов, принадлежащих компании. Для этого достаточно определить примерно такие правила:

```
Order deny,allow
    Allow from 192.168.0.0/16
    Deny from all
```

Подобные примеры вы встретите во всех конфигурационных файлах Apache.

Параметры каталогов

Параметры представляют основные серверные функции, которые можно включать или выключать для отдельных каталогов. Они позволяют веб-разработчику выполнять программы на сервере, включать и выключать защиту с помощью паролей для тех или иных каталогов и управлять поддержкой языков. Параметры предоставляют веб-разработчику не только мощные возможности, но и могут вызвать дополнительные вопросы. Включение только необходимых параметров позволит впоследствии сэкономить время на устранении проблем.

Параметры для каталогов можно указывать с помощью ключевого слова `Options`. Например, чтобы задать параметры `ExecCGI` и `MultiViews` для каталога */var/www/mysite/cgi-bin*, надо выбрать такую конфигурацию:

```
<Directory /var/www/mysite/cgi-bin>
    Options ExecCGI, MultiViews
</Directory>
```

Теперь посмотрим, какие параметры поддерживает Apache.

`None`

Параметр `None` запрещает все остальные параметры. В примере файла *httpd.conf* параметр `Option` имеет значение `None`. Если вы создаете конфигурацию сервера на основе примера по умолчанию *http.conf*, вам придется явно разрешить все параметры, которые понадобятся для работы.

`All`

Параметр `All` включен в сервере Apache по умолчанию. Если не указаны никакие параметры, будут доступны почти все функции Apache. Если пользователь выгрузит на сервер сценарий для защиты своего каталога с помощью паролей, это сработает. Пользователь может выгрузить на сервер CGI-сценарий, который пробьет брешь в локальной системе и запустит командный интерпретатор от имени root на TCP-порте, открыв всем желающим двери в вашу систему. Параметр `All` разрешает все остальные параметры Apache, *за исключением* `MultiViews` (как показано далее).

`ExecCGI`

Веб-сервер Apache сможет запускать CGI-сценарии, находящиеся в этом каталоге.

`FollowSymLinks`

Для обращения к другим файлам на сервере можно применять *символические ссылки*, или псевдонимы, описываемые на странице руководства ln(1). Пользователь может создать символическую ссылку чуть ли не на каждый файл на сервере, причем этот файл будет видимым, конечно, если это позволяют права доступа к файлу.

`SymLinksIfOwnerMatch`

Сервер будет применять символические ссылки, если владелец ссылки также владеет файлом, на который указывает ссылка. Другими словами, пользователь может применять символические ссылки, указывающие на его документы.

`Includes`

Вставки на стороне сервера (*Server Side Includes*, *SSI*-файлы HTML, содержащие инструкции для командного процессора) и CGI-сценарии будут работать в каталогах с установленным параметром `Includes`, однако они могут быть опасны, если их писали неосторожно. В конце концов, вы позволяете посетителям вашего веб-сайта запускать команду, которая использована в этом HTML-файле. Коварные взломщики, проявив изобретательность, могут с помощью данной команды выполнить действия, которые не предусматривал веб-

дизайнер. Поищите в Сети материалы, посвященные SSI и безопасности, и вы найдете себе занятие на очень долгое время. Если вы не знаете, как безопасно применять SSI, не включайте этот параметр!

`IncludesNOEXEC`

Разрешает SSI, но запрещает функции `#exec` и `include` в серверных вставках. Без функции `#exec` код HTML не может просто запустить любую команду – все команды SSI должны быть написаны в тщательно ограниченном диапазоне параметров. По сути, этот параметр разрешает простые вставки SSI и сценарии CGI, но исключает наиболее типичные бреши безопасности. Однако исключение наиболее типичных брешей системы безопасности вовсе не означает полную безопасность; вы лишь немного прибавили работы злоумышленнику.

`Indexes`

Если в каталоге нет файла *index.html*, параметр `Indexes` позволит серверу вернуть хорошо отформатированный список содержимого каталога. Можно считать это брешью в системе безопасности – в зависимости от содержимого каталога. Например, если кто-либо просматривает содержимое каталога моей домашней веб-страницы, это не страшно. Но если открыт доступ к каталогу с исходными текстами моих программ, дело принимает другой оборот. (В моем случае неприятность заключается лишь в том, что эти исходные тексты содержат ошибки, но чьи-то тексты программ могут быть на самом деле дорогими.)

`MultiViews`

Этот параметр позволяет серверу обслуживать HTML-документы, представленные на нескольких языках. Например, веб-разработчик может создать документ HTML, содержащий текст на английском, китайском и испанском языках. Если параметр `MultiViews` включен, Apache посылает клиенту страницу на языке, который задан в веб-броузере.

Параметры настройки для пользователей

Одна из интересных возможностей, которые предоставляет Apache, заключается в том, что пользователи могут выгружать на сервер собственные конфигурационные файлы, а сервер будет читать их и использовать. Ключевое слово `AllowOverride` позволяет администратору веб-сервера Apache определить, какие конфигурационные параметры заданного каталога могут или не могут изменяться пользователями. Это дает веб-разработчикам возможность взять на себя большую часть хлопот по конфигурированию, а также устанавливать небезопасные CGI-сценарии в произвольные каталоги.

Свои изменения в конфигурации пользователи помещают в файл с именем *.htaccess*, в требуемом каталоге. Если вы запускаете корпоративный веб-сервер, и ваш веб-разработчик так или иначе получает все,

что ему нужно, нет никаких причин запрещать ему вносить изменения. Но если вы запускаете общедоступный веб-сервер или веб-сервер компании-провайдера и хотите запретить определенным группам клиентов выполнять CGI-сценарии, то надо выключить параметр `ExecCGI` и запретить те или иные подмены, позволяющие использовать CGI.

Параметр `AllowOverride` размещается в отдельной строке внутри определения `Directory` вместе с допустимыми подменами. Вот набор разрешенных подмен, приемлемый для большинства веб-сайтов:

```
AllowOverride FileInfo AuthConfig Limit Indexes
```

Для параметра `AllowOverride` определены следующие ключевые слова, разрешающие пользователям подменять что угодно путем записи в файл *.htaccess*.

`AuthConfig`

`AuthConfig` позволяет защищать каталоги с помощью паролей и обеспечивает довольно высокую степень безопасности. Ее применение имеет смысл при администрировании групп серверов, где учетную запись может получить любой дурак с кредитной картой.

`FileInfo`

`FileInfo` разрешает пользователям вставлять собственную информацию MIME для файлов в каталоге. Как правило, лучше добавлять такую информацию в серверный файл *mime.types*, но некоторым пользователям может показаться, что такая возможность им необходима.

`Indexes`

`Indexes` разрешает управлять индексами, включая задание документа по умолчанию, настройку представления пиктограмм в индексах, созданных сервером, и т. д.

`Limit`

Пользователи, которые могут изменять параметр `Limit`, смогут использовать ключевые слова `Allow`, `Deny` и `Order` для управления доступом к своим каталогам на базе IP-адресов и имен хостов. Этот параметр также вполне безопасен.

`None`

`None` означает, что пользователь ничего не может изменять в настройках сервера. Это хороший выбор по умолчанию, однако он налагает излишние ограничения на работу большинства приложений.

`Options`

Наконец, можно разрешить пользователям настраивать параметры своих каталогов, как обсуждалось в предыдущем разделе. Это удобно, если вы доверяете вашим веб-разработчикам либо не беспокоитесь о том, что кто-то сможет выгрузить небезопасную программу, подвергающую сервер риску.

Прочие настройки каталогов

Веб-сервер Apache позволяет настраивать любые параметры для каждого каталога, однако здесь мы рассмотрим только наиболее важные из них. В документации к Apache вы найдете больше, намного больше параметров.

Индексные документы

Параметр `DirectoryIndex` задает имя документа, выдаваемого по умолчанию. Когда клиент обращается к каталогу, а не к имени файла, Apache ищет в этом каталоге файлы с указанными именами согласно порядку их представления в данном операторе. Измените это значение, если вы пользуетесь таким инструментом, как PHP (имена файлов имеют расширение *.php*) или редакторами веб-страниц операционной системы Windows, которые дают HTML-файлам расширения *.htm*, или, что еще хуже, когда начальная страница хранится в файле *default.htm*.

```
DirectoryIndex index.php index.htm index.html
```

Псевдонимы

Параметр `Alias` задает псевдонимы для каталогов на вашем веб-сайте – подобно символическим ссылкам. С помощью параметра `Alias` вы можете объединить различные каталоги в единый сайт, не прибегая к `FollowSymlinks`. Это особенно удобно при работе с программами и веб-приложениями сторонних разработчиков.

```
Alias /icons/ "/usr/local/www/icons"
```

В этом примере подразумевается, что пользователь, заглянувший по адресу *http://www.absolutefreebsd.com/icons*, на самом деле будет просматривать содержимое каталога */usr/local/www/icons/*, хотя значение параметра `DocumentRoot` не имеет никакого отношения к этому каталогу.

Вероятно, вам также придется добавить определение параметра `Directory`, чтобы указать права доступа к каталогу, на который ссылается псевдоним.

Нестандартные страницы с сообщениями об ошибках

Дополнительная возможность обслуживания веб-страниц – веб-сервер может отображать для клиента сообщения об ошибках. Сервер Apache содержит стандартные страницы с сообщениями об ошибках, однако вы можете создать собственные страницы и отправлять клиентов к ним. Делается это с помощью ключевого слова `ErrorDocument`, вслед за которым указывается код ошибки и имя файла страницы с описанием ошибки:

```
ErrorDocument 404 /missing.html
```

404 – это код ошибки «Страница не найдена». Запросив отсутствующую страницу, пользователь получит вместо нее файл *missing.html*.

Защита с помощью пароля и Apache

Ограничить доступ к страницам веб-сайта на основании имен пользователей и паролей – вполне обычное желание. Сервер Apache может выполнять аутентификацию пользователей с применением любых схем, начиная от интеграции с LDAP и Kerberos и заканчивая базами данных в виде простых текстовых файлов. Мы рассмотрим два стандартных способа аутентификации: файлы паролей Apache и аутентификация средствами Radius.

Файлы паролей

Файлы паролей обычно применяются в случаях, когда нужно предоставить доступ к какому-либо каталогу небольшому числу пользователей. Этот метод также популярен при использовании виртуальных хостов, где пользователям может потребоваться управлять доступом к своим сайтам. Не размещайте файл паролей в каталоге веб-сайта, в противном случае пользователи смогут загрузить его и попытаться взломать пароли. Храните файл паролей вообще за пределами любого веб-сайта. Если пользователь, управляющий своим веб-сайтом, одновременно имеет учетную запись в системе, файл паролей можно поместить в домашний каталог пользователя.

Для начала создайте пустой файл паролей с помощью утилиты touch(1), а затем используйте программу htpasswd(1), чтобы добавлять имена пользователей и изменять их пароли. Синтаксис этой команды очень прост:

```
# htpasswd файл_паролей имя_пользователя
```

Например, добавить пользователя с именем mwlucas в файл паролей *webpasswords* можно с помощью команды `htpasswd webpasswords mwlucas`. Изменить пароль пользователя можно той же самой командой. Все записи в этом файле хранятся в виде отдельных строк, по одной на каждого пользователя, где указаны имя пользователя и хеш пароля:

```
mwlucas:iJf2e7KIgS5i6
```

Чтобы удалить пользователя из базы данных, достаточно удалить соответствующую строку из файла паролей. Можно также использовать команду htpasswd(1) с ключом `-D`, но, думаю, гораздо проще удалить строку с помощью редактора vi(1), чем запоминать все это.

Теперь нужно сообщить веб-серверу Apache о необходимости применить защиту паролями к каталогу. В принципе, можно настроить защиту паролем прямо в файле *httpd.conf*, но наиболее часто настройка аутентификации производится с помощью файла *.htaccess*. Такой способ настройки аутентификации не требует полной перезагрузки всего веб-сервера. Обычно конечным пользователям изредка требуется изменить свои настройки аутентификации, а файл паролей обеспечивает простой способ это сделать. Определите параметр `AllowOverride AuthConfig` для

каталога, защищенного паролем (или для всего сайта), затем создайте в защищенном каталоге файл *.htaccess*, содержимое которого должно выглядеть примерно так:

```
AuthName "Employees Only"
AuthType basic
AuthUserFile /home/mwlucas/sitepasswords
require valid-user
```

`AuthName` — это текст, который появится в диалоге ввода пароля. Вы можете поместить сюда любой текст в кавычках, и веб-броузер отобразит его.

`AuthType` определяет тип аутентификации клиента. В случае обычного имени пользователя и пароля используйте тип `basic`.

`AuthUserFile` сообщает серверу Apache местоположение базы данных с информацией о пользователях и тип этой базы данных. В данном случае указан файл паролей, созданный с помощью htpasswd(1).

Инструкция `require valid-user` сообщает серверу Apache, что он должен потребовать ввод имени пользователя и пароля и разрешить доступ только тем пользователям, которые ввели верную аутентификационную информацию.

Аутентификация пользователей средствами сервера Radius

Сервер Radius обеспечивает очень неплохой способ аутентификации, по сравнению со службами каталогов сторонних производителей, такими как Active Directory и LDAP. Есть много разнообразных серверов Radius, от свободно распространяемого сервера OpenRADIUS и до сервера Internet Authentication Service, связанного с серверами Windows. Вся прелесть применения Radius в том, что вам не придется мучаться с LDAP или Kerberos, так как группы управления каталогами предприятия могут отказать в разрешении на использование нестандартных систем обеспечения доступа к каталогам. В большинстве случаев вы можете просто собрать сервер Radius и позволить выполнять аутентификацию ему, а не службе каталогов предприятия или чему-то еще. В Интернете можно найти несколько различных серверов Radius для Apache, я предпочитаю mod_auth_xradius (*/usr/ports/www/mod_auth_xradius*). Он прекрасно работает с современными версиями Apache и довольно прост в настройке.

В первую очередь нужно сконфигурировать сервер Radius. Если вы работаете на предприятии или в компании-провайдере, возможно, у вас уже имеется сервер Radius. Есть много различных версий сервера Radius от разных производителей, у каждой из них есть свои достоинства и недостатки. Несколько серверов входят в состав коллекции «портов» операционной системы FreeBSD. Самое замечательное, что Apache не нуждается в каких-то необычных возможностях Radius; подойдет любой имеющийся сервер. Если у вас используется Microsoft Active

Directory, приобретите Internet Authentication Services. Это очень небольшой сервер, и, вполне возможно, вам удастся убедить администратора AD установить его для вас. Ваши коллеги будут счастливы, что ваше веб-приложение обеспечивает способ аутентификации, интегрированный с их учетными записями на настольных компьютерах, ведь так делается далеко не везде.

Теперь нужно настроить загрузку модуля в Apache. Перейдите в конец списка инструкций `LoadModule` и добавьте следующую строку:

```
LoadModule auth_xradius_module libexec/apache22/mod_auth_xradius.so
```

Затем нужно настроить кэш сервера Radius. Сервер Apache хранит в кэше список пользователей, благополучно прошедших аутентификацию. Без этого кэша Apache будет обращаться к серверу Radius перед отправкой каждого объекта на каждой веб-странице. Единственный щелчок мыши может породить десятки запросов, что, как правило, нежелательно.

```
AuthXRadiusCacheTimeout 300
AuthXRadiusCache dbm "/tmp/auth_xradius_cache"
```

Параметр `AuthXRadiusCacheTimeout` определяет время хранения кэшированных объектов (в секундах). Параметр `AuthXRadiusCache` определяет, где будет храниться кэш. В данном случае я использую файл типа `dbm` (база данных хешей) в каталоге */tmp*. Если пользователи, которым вы не доверяете, могут входить в вашу систему, получая при этом доступ к командной строке, перенесите кэш в каталог, недоступный пользователям, однако в случае выделенного сервера выбор каталога */tmp* достаточно безопасен.

Теперь нужно настроить режим аутентификации Radius для требуемого каталога. Вы можете сделать это с помощью файла *.htaccess*, как в случае с именами пользователей и паролями, но на этот раз я рекомендую поместить настройки непосредственно в файл *httpd.conf*. Сайты, использующие аутентификацию Radius, обычно поддерживаются системным администратором, а не пользователями. Поместите настройки непосредственно в инструкцию `Directory`. Например, я использую программу мониторинга Nagios. Вот конфигурация защиты каталога *nagios* паролем с помощью Radius:

```
<Directory /usr/local/www/nagios>
        AllowOverride None
        ...
        AuthName "Nagios"
        AuthBasicProvider "xradius"
        AuthType basic
        AuthXRadiusAddServer "radius.absolutefreebsd.com" "RadiusSecret"
        AuthXRadiusTimeout 2
        AuthXRadiusRetries 2
        require valid-user
</Directory>
```

На первый взгляд, довольно непривычно, но если ознакомиться с настройками не спеша, все станет понятно. Инструкция `AllowOverride` запрещает переопределять параметры настройки с помощью файлов *.htaccess*. Это нормально, так как настройки каталога выполняются непосредственно в конфигурационном файле веб-сервера.

Параметр `AuthName` определяет текст, который появится в диалоге ввода пароля.

Параметр `AuthBasicProvider` указывает на источник аутентификационной информации. Файлы паролей появились в Apache давным-давно, поэтому Apache предполагает их использование по умолчанию, но в случае применения других систем аутентификации необходимо явно указать это.

Ключевое слово `basic` в параметре типа аутентификации `AuthType` указывает, что используется базовая аутентификация HTTP.

Параметр `AuthXRadiusAddServer` указывает, где можно найти сервер Radius, и содержит ключ доступа к нему. Имя сервера и ключ должны быть указаны в кавычках.

Сервер Radius откликается быстрее чем за секунду. Параметр `AuthXRadiusTimeout` позволяет задать серверу Apache временную задержку перед повторной попыткой, а количество попыток определяется параметром `AuthXRadiusRetries`.

Наконец, необходимо указать, что для получения доступа к каталогу пользователь обязан предоставить верное имя пользователя и пароль.

Хотя этот пример показывает использование сервера Radius, в частности – модуля XRadius, вы можете применять продемонстрированные здесь принципы для работы с любым другим модулем аутентификации. Прочтите документацию модуля и выполните необходимую настройку. Параметры другого модуля могут отличаться от параметров Radius, но скорее всего они будут очень похожи.

Группы и файл .htaccess

Одна из удобных особенностей аутентификации на основе имен пользователей и паролей – возможность предоставлять доступ группам пользователей. На крупных веб-сайтах аутентифицированным пользователям предоставляется доступ к большей части информационного наполнения, но лишь немногим из них положен доступ к административной части. Реализовать такую возможность можно с помощью групп пользователей. Файл групп сервера Apache очень напоминает файл */etc/group*, где вслед за двоеточием, стоящим после имени группы, перечислены имена пользователей, разделенные запятыми:

```
administrators: mwlucas, gedonner
```

Затем необходимо сообщить Apache о существовании файла групп и добавить инструкцию `require-group` в файл *.htaccess* или непосредственно в файл *httpd.conf*:

```
authgroupfile /usr/local/etc/apache22/users/webgroup
require-group administrators
```

Теперь ваши администраторы будут использовать единый пароль как для смешанной, так и для административной части сайта, причем доступ к административной части смогут получить только администраторы. Этот способ позволяет как угодно разделить веб-сайт, создав много групп.

Включение других конфигурационных файлов

Одна из особенностей, которая упрощает сопровождение конфигурации сервера Apache, состоит в возможности подключать другие конфигурационные файлы. В старых версиях Apache файлы *httpd.conf* содержали тысячи строк, не нужных большинству пользователей. Теперь эти настройки сгруппированы по разделам и вынесены в отдельные конфигурационные файлы. В конце *httpd.conf* можно встретить примерно такие строки:

```
...
# Fancy directory listings
❶ #Include etc/apache22/extra/httpd-autoindex.conf
# Language settings
#Include etc/apache22/extra/httpd-languages.conf
# User home directories
#Include etc/apache22/extra/httpd-userdir.conf
...
```

Раскомментировав какую-либо строку, вы тем самым задействуете функциональность, настройки которой находятся в этом подключаемом файле. Например, чтобы включить функцию улучшенного представления содержимого каталога, раскомментируйте строку ❶, ссылающуюся на этот конфигурационный файл. Конечно, предварительно следует ознакомиться с содержимым файла, чтобы убедиться, что данная функция настроена должным образом.

В конфигурационном файле веб-сервера Apache по умолчанию предусмотрено наличие двух каталогов, предназначенных для размещения подключаемых конфигураций: *extra* и *Includes*. Каталог *Includes* отведен для вас. Любой файл с расширением *.conf*, находящийся в каталоге *Includes*, автоматически попадает в глобальную конфигурацию сервера Apache. Мы используем эту особенность при создании виртуальных хостов ниже в этой главе. Файлы в каталоге *extra* поставляются вместе с сервером Apache и определяют настройки специальных функций, которые не всегда нужны, но достаточно популярны, чтобы включить их в дистрибутив Apache. Вот эти функции:

MPM (модуль многопроцессорной обработки) Определяет, как Apache должен обслуживать рабочие процессы. Все настройки в этом файле имеют значения по умолчанию.

Multi-Language Error Messages (представление сообщений об ошибках на нескольких языках) По умолчанию сервер Apache выводит сообщения об ошибках на английском языке. Если вам требуется обеспечить поддержку нескольких языков, включите этот модуль. Броузер и сервер сами договорятся о языке, на котором должны выводиться сообщения об ошибках.

Fancy Directory Listings (улучшенное представление содержимого каталога) По умолчанию функция `autoindex` сервера Apache генерирует не очень красивый список содержимого в каталогах. Данный модуль обеспечивает улучшенное представление содержимого каталога.

Language Settings (настройки языка) С помощью этой функции можно присвоить серверу Apache другой язык по умолчанию и определить различные кодировки символов.

User Home Directories (домашние каталоги пользователей) Традиционно домашние каталоги пользователей на веб-сервере доступны в виде веб-страниц *http://<имя_сервера>/~<имя_пользователя>*. Этот модуль включает данную функциональность. Посмотреть, как она работает, можно на моей домашней странице *http://www.blackhelicopters.org/~mwlucas*.

Status (состояние) Сервер Apache может генерировать веб-страницы с информацией о состоянии веб-сервера, о его конфигурации и прочих характеристиках.

Virtual hosts (виртуальные хосты) С помощью виртуальных хостов можно на одном веб-сервере запустить несколько веб-сайтов. Подробнее о виртуальных хостах мы поговорим ниже в этой главе.

Manual (руководство) Сервер Apache распространяется с руководством по установленной версии. Подключив этот модуль, вы тем самым сделаете доступным возможность просмотра руководства на веб-сайте.

WebDAV DAV позволяет создавать область совместного использования файлов, куда пользователи смогут выгружать и откуда они смогут загружать документы. Прежде чем активизировать эту возможность, необходимо изучить проблему безопасности веб-сайта и ознакомиться с документацией WebDAV.

Defaults (настройки по умолчанию) У Apache много параметров со значениями по умолчанию, определенных непосредственно в программном коде. Если вам требуется переопределить эти настройки, загрузите конфигурацию по умолчанию и выполните необходимые настройки в ней.

SSL Применение SSL – достаточно важная тема, для обсуждения которой выделен целый раздел ниже в этой главе.

Для каждой из этих функций имеется свой конфигурационный файл в каталоге *extra*. Для более полного представления ознакомьтесь с их содержимым.

Виртуальный хостинг

Виртуальный хостинг подразумевает поддержку нескольких веб-сайтов одним веб-сервером. Сервер сконфигурирован для обслуживания веб-запросов к каждому из этих доменов и возвращает пользователям соответствующие страницы. Многим компаниям нужен очень маленький веб-сайт, содержащий лишь несколько информационных страниц и, возможно, один-два CGI-сценария. Именно здесь замечательно подойдут виртуальные хосты. Мои серверы FreeBSD без особых усилий обслуживали тысячи маленьких доменов, при этом системная нагрузка не превышала 0,2. Учитывая, что каждый владелец такого сайта платит $19,95 в месяц за обслуживание пары десятков посетителей в день, можно быстро заработать хорошие деньги на недорогих аппаратных средствах.

Типичный камень преткновения в понимании виртуальных хостов – распространенное заблуждение, что волшебное слово *www* в URL указывает на веб-сайт. На самом деле, вводя, например, URL *http://www.freebsd.com*, вы предписываете веб-броузеру найти машину с именем *www.freebsd.com*, а затем подключиться к порту 80 и посмотреть, что веб-сервер сможет предложить. Сочетание символов *www* было придумано системными администраторами как сокращение от «сервер с кусочком Сети (*w*orld *w*ide *w*eb) на нем». Если взглянуть немного повнимательнее, можно встретить веб-сайты, размещенные на нескольких машинах с разными именами. В случае виртуального хостинга несколько имен хостов относятся к одной и той же машине. Сервер должен различать запросы к различным доменам, а затем предоставлять соответствующие страницы. Чтобы включить функцию обслуживания виртуальных хостов, нужно раскомментировать соответствующую строку с инструкцией `Include` в файле *httpd.conf*.

Конфигурирование виртуальных хостов

Поскольку единственный сервер Apache в состоянии обслуживать тысячи виртуальных хостов, я предлагаю хранить конфигурацию каждого такого хоста в отдельном файле в каталоге *Includes*. Если имя файла имеет расширение *.conf*, Apache автоматически подключает конфигурацию при последующих перезагрузках. Кроме того, рекомендую присвоить файлу имя, содержащее имя домена, который он будет обслуживать. В результате у вас могут получиться такие имена файлов, как *www.customer1.com.conf*, *www.customer2.com.conf* и т. д. По сравнению с монолитной конфигурацией, когда все параметры настройки сосредоточены в одном файле, такой подход позволяет упростить поиск неполадок на сайте.

Сервер Apache поддерживает два способа определения виртуальных хостов: по имени и по IP-адресу. При определении виртуальных хостов *по имени* предполагается, что клиент обращается к веб-сайту по его имени. Все броузеры, начиная с Netscape 3 и IE 4, поддерживают

такую возможность. Определение виртуальных хостов по имени будет правильным выбором практически в любом случае. Определение виртуальных хостов *по IP-адресу* означает, что каждому отдельному IP-адресу соответствует свой виртуальный хост. Оно необходимо только в том случае, когда для работы виртуального хоста требуется сертификат SSL. Оба способа можно применять на одном и том же сервере.

Ниже приведен пример определения виртуального сервера по имени для домена *absolutefreebsd.com*. Его конфигурационный файл: */usr/local/etc/apache22/Includes/absolutefreebsd.com.conf*.

```
<VirtualHost *:80>
❶     ServerAdmin webmaster@absolutefreebsd.com
❷     DocumentRoot /var/www/absolutefreebsd.com
❸     ServerName absolutefreebsd.com
❹     ServerAlias www.absolutefreebsd.com
❺     ErrorLog /var/log/http/absolutefreebsd.com-error_log
❻     CustomLog /var/log/http/absolutefreebsd.com-access_log combined
</VirtualHost>
```

Первое, что мы замечаем, – это теги `<VirtualHost>` и `</VirtualHost>`. Все, что находится между этими тегами, является определением единственного виртуального хоста. Запись `*:80` говорит о том, что запросы к этому веб-сайту могут поступать на порт 80 по любому IP-адресу.

Как и в случае с основным сервером, для виртуального хоста необходимо определить параметры `ServerAdmin` ❶ и `DocumentRoot` ❷; благодаря последнему Apache сможет найти документы, составляющие информационное наполнение веб-сайта. Кроме того, для веб-сервера нужно определить параметры `ErrorLog` ❺ и `CustomLog` ❻. (Можно завести единый протокол ошибок для всех виртуальных веб-сайтов, но потом в нем будет сложнее искать нужную информацию.)

Пожалуй, самое интересное здесь – это параметры `ServerName` и `ServerAlias`. Имя сервера ❸ – *absolutefreebsd.com*. Параметр `ServerAlias` ❹ определяет второе имя, *www.absolutefreebsd.com*. В результате Apache будет выдавать одно и то же содержимое для любого из этих имен. Если кто-то поленился ввести часть *www* в имени сайта, это не значит, что ему следует отказать в получении страницы.

Виртуальный хост на базе IP-адреса задается практически так же, как на базе имени, за исключением содержимого тега `<VirtualHost>`. В этом случае вместо комбинации `*:80`, указывающей, что данный хост будет принимать запросы на любой из IP-адресов, относящихся к машине, следует определить единственный IP-адрес:

```
<VirtualHost 192.168.1.5:80>
      ServerAdmin webmaster@AbsoluteFreeBSD.com
      ...
```

Этот виртуальный хост будет принимать запросы, поступающие на IP-адрес 192.168.1.5. Создавать виртуальный хост на базе IP-адреса необ-

ходимо, только если сайт использует протокол HTTPS, для которого должен использоваться единственный, уникальный IP-адрес. Читайте раздел «Веб-сайты HTTPS» ниже в этой главе.

Определение виртуального хоста должно содержать все необходимое для обеспечения работы веб-сайта. Например, сервер Nagios, для которого ранее в этой главе я настраивал аутентификацию через сервер Radius, фактически является виртуальным хостом. Поэтому конфигурация Radius должна находиться в конфигурационном файле виртуального хоста, а не в *httpd.conf*.

> **Хост по умолчанию**
>
> Если вы используете виртуальные хосты, самый первый хост в конфигурации служит веб-сайтом по умолчанию. Обязательно проверьте, чтобы значения параметров `DocumentRoot` и `ErrorLog` этого хоста совпадали со значениями сайта по умолчанию, определенными в *httpd.conf*. Кроме того, рекомендую удалить пример определения виртуального хоста на базе имени из файла *extra/httpd-vhosts.conf*.

Тонкая настройка виртуальных хостов

Обеспечив работу виртуальных хостов с минимальной конфигурацией, можно провести дополнительную настройку. Здесь мы обсудим параметры виртуальных хостов, которые подходят для виртуальных хостов обоих типов.

Номера портов

Различные сайты можно обслуживать на разных портах TCP/IP. Вы могли видеть это раньше – когда имя хоста в URL завершалось двоеточием и числом. Например, если Apache прослушивает порты 80 и 8080, на каждом из них можно разместить различные виртуальные хосты, если добавить номера портов к IP-адресам в директиве `VirtualHost`.

Ниже приведен пример конфигурации, создающей два различных сайта на разных портах. На деле мне пришлось бы дополнить эту конфигурацию инструкциями протоколирования и информацией об администраторе, а в примере я ее упростил:

```
<VirtualHost *:80>
    DocumentRoot /var/www/www.absolutefreebsd.com
    ServerName www.absolutefreebsd.com
</VirtualHost>
<VirtualHost *:8080>
    DocumentRoot /var/www/data.absolutefreebsd.com
    ServerName data.absolutefreebsd.com
</VirtualHost>
```

Вы наверняка заметили, что содержимое обоих сайтов находится в одном и том же каталоге, с тем же успехом можно было бы настроить оба сайта на прослушивание единственного порта с номером 80. Это совершенно не важно, если помогает решить вашу задачу.

Options и AllowOverride

По умолчанию виртуальные хосты наследуют значения `Options` и `AllowOverride` из корневого каталога Apache. Значения, заданные по умолчанию, являются весьма ограничивающими, запрещая доступ откуда угодно. Однако вы можете переопределить поведение по умолчанию, поместив инструкции `Options` и `AllowOverride` в определение виртуального хоста. Здесь можно использовать любые параметры, допустимые в *httpd.conf*. Это позволяет по-разному настраивать отдельные сайты. Например, на одном виртуальном сайте можно разрешить вставки на стороне сервера, запретив их на другом. В следующем определении виртуального хоста значение `Options` переопределяет настройки сервера по умолчанию и назначает для каталога `DocumentRoot` специальные привилегии.

```
<VirtualHost *:80>
    DocumentRoot /var/www/absolutefreebsd.com
    ServerName absolutefreebsd.com
    Options IncludesNOEXEC
</VirtualHost>
<Directory /var/www/absolutefreebsd.com>
    AllowOverride AuthConfig
</Directory>
```

Все это делает виртуальный хост почти таким же гибким, как выделенный сервер, разумеется, пока ваши клиенты не пожелают установить собственные модули Apache. В этом случае вы можете продать им сервер в клетке и позволить устанавливать любые модули.

Веб-сайты HTTPS

Многие интернет-магазины, защищенные паролями, применяют так называемые *безопасные веб-сайты*. Обычно под этим понимается шифрование трафика между сервером и клиентом с помощью SSL. Хотя безопасность таких сайтов не в полной мере соответствует их названию, функции SSL жизненно важны почти для любого веб-сервера. Функциональность SSL интегрирована в Apache в виде модуля *extra/httpd-ssl.conf*. Для включения поддержки SSL надо раскомментировать соответствующую строку в файле *httpd.conf*.

Каждому серверу SSL нужен сертификат. Генерирование запроса на сертификат и создание самозаверенного сертификата рассматривалось в главе 9. Для личных нужд вполне подойдет самозаверенный сертификат, но в случае предоставления услуг другим необходимо приобрести сертификат, заверенный центром сертификации. Если попытаться

использовать самозаверенный сертификат в приложении, с которым работают клиенты, они будут получать страшные предупреждения о недостаточно высоком уровне безопасности.

Полный сертификат состоит из двух частей – файла сертификата (*hostname.crt*) и секретного ключа (*hostname.key*). Разместите оба файла в каталоге за пределами содержимого веб-сервера, чтобы никто не смог загрузить его с веб-сервера. Затем надо разрешить чтение файла только непривилегированному пользователю, с правами которого работает веб-сервер, и сделать их недоступными для обычных пользователей.

```
# chmod 600 hostname.crt
# chmod 600 hostname.key
# chown www:www hostname.crt
# chown www:www hostname.key
```

Теперь, когда в системе появился сертификат, надо известить об этом Apache. В прошлом необходимость настраивать SSL доводила опытных системных администраторов до слез; сегодня для этого нужно добавить всего четыре строки в конфигурацию виртуального хоста:

```
<VirtualHost 192.168.1.5:443>
    ServerName secure.absolutefreebsd.com
    SSLEngine on
    SSLCertificateFile etc/apache22/ssl.crt/hostname.crt
    SSLCertificateKeyFile etc/apache22/ssl.key/hostname.key
    ...
```

Прежде всего, отметим, что виртуальный хост создан на базе IP-адреса и прослушивает порт 443. Стандартные сайты HTTPS работают с портом TCP 443.

Параметр `ServerName` чрезвычайно важен для сайтов HTTPS. Значение параметра `ServerName` должно точно соответствовать результатам обратного поиска в DNS по IP-адресу и имени в сертификате SSL. Если

Безопасные веб–серверы и SSL

Я рекомендую избегать термина «безопасный веб-сервер», если имеется в виду SSL. Шифрование трафика между сервером и клиентом защищает лишь от одной из разновидностей сетевых атак. Злоумышленники по-прежнему могут взломать либо сервер, либо клиентскую машину. Для безопасности веб-серверу требуется хороший дизайн веб-страниц, регулярное обслуживание и образованный системный администратор, которого не дергают по пустякам, позволяя ему спокойно заниматься своим делом. Хотя последний пункт этих требований практически невыполним, он меркнет рядом с таким условием безопасности веб-клиента, как образованный пользователь!

между тремя этими именами нет точного соответствия, пользователь может увидеть предупреждение системы безопасности. Параметр `ServerAlias` не имеет большого значения для веб-сайтов HTTPS.

Затем необходимо включить механизм SSL для сайта и указать ему полный путь к сертификату и секретному ключу. Указывайте полное имя хоста для файлов сертификата и секретного ключа.

Управление сервером Apache

Сервер Apache – это сложная программа, которой можно управлять по-разному. Хотя утилита apachectl(8) работает очень неплохо, рекомендую применять сценарий запуска, интегрированный в операционную систему FreeBSD. Этот сценарий запускает apachectl(8) с особыми настройками, необходимыми именно в вашем случае, и гарантирует, что при следующей загрузке системы сервер Apache будет работать точно так же, как до ее останова. Однако сервер Apache немного отличается от большинства других программ, поэтому в ваше распоряжение предоставляется несколько дополнительных параметров командной строки, помимо обычных `start` и `stop`. Обычно используются параметры `start`, `stop`, `restart`, `graceful`, `gracefulstop` и `configtest`.

Команда `/usr/local/etc/rc.d/apache22 start` запускает Apache со всеми сконфигурированными модулями. Никаких специальных команд для запуска веб-сервера с поддержкой SSL теперь нет. Если в конфигурации сервера обнаружатся ошибки, эта команда выведет предупреждение и прервет процедуру запуска сервиса.

Команда `/usr/local/etc/rc.d/apache22 stop` немедленно останавливает Apache и закрывает все соединения, не дожидаясь полного завершения запросов.

Команда `/usr/local/etc/rc.d/apache22 restart` проверяет конфигурацию сервера. Если в конфигурации сервера обнаружатся ошибки, эта команда выведет предупреждение и больше никаких действий не предпримет. В случае отсутствия ошибок она остановит и тут же запустит Apache.

Команда `/usr/local/etc/rc.d/apache22 graceful` выполнит постепенный перезапуск. Перед остановом сервера будет разрешено полностью завершить обмен для всех открытых соединений. Это может показаться лишним, но при обслуживании больших файлов или при наличии нескольких серверов позади стабилизатора нагрузки такой подход обретает особую важность. Подобно команде `restart`, эта команда выполняет проверку конфигурации, прежде чем остановить сервис, и не предпринимает никаких действий в случае выявления проблем в конфигурационных файлах.

Команда `/usr/local/etc/rc.d/apache22 gracefulstop` останавливает Apache, не прибегая к насильственному разрыву соединений. Открытые соеди-

нения остаются открытыми, пока запросы не будут полностью выполнены, и только после этого процесс останавливается.

Команда `/usr/local/etc/rc.d/apache22 configtest` проверяет конфигурацию Apache и выводит предупреждения в случае выявления ошибок. Это та самая функция, с помощью которой команды `restart` и `graceful` проверяют конфигурацию перед остановом текущего процесса.

Теперь, когда вы научились управлять сервером Apache, давайте посмотрим, как можно выгружать файлы на сервер и загружать их с сервера.

Передача файлов

Никому не нужен веб-сервер без веб-страниц. Мои веб-сайты, созданные на базе FreeBSD, выглядят отвратительно.[1] Профессиональные веб-дизайнеры обычно проектируют сайты на своих рабочих станциях, а затем выгружают их на сервер. Стандартным способом передачи файлов являются FTP и sftp/scp.

FTP, File Transfer Protocol (протокол передачи данных) – это классический протокол для переноса файлов с одного компьютера на другой по Интернету. Большинство пользователей предпочтут использовать FTP. Как и многие другие протоколы, FTP со временем не стал лучше. Он с трудом работает в современных средах. За долгие годы в спецификации накопилось столько исправлений, что протокол стал похож на ужасное творение Франкенштейна, которое падает, как только администратор сети отвернется хотя бы на секунду. Хотя система FreeBSD обеспечивает максимально легкое обслуживание FTP, требуется немного поработать, чтобы этот протокол продолжал с пыхтением двигаться вперед.

Безопасность FTP

Пароли и имена пользователей передаются по FTP открытым текстом, поэтому обладатель анализатора пакетов может перехватить эту информацию. Никто в локальной сети, за исключением сетевого администратора, не должен обладать анализатором пакетов, но если ваши пользователи находятся в удаленных сетях или за кабельными модемами, их пароли под угрозой.

Будучи системным администратором, никогда, *никогда*, ***никогда*** не передавайте по сети свой пароль открытым текстом! Вместо этого для переноса файлов между машинами используйте scp(1) или sftp(1).

---

[1] Замечу, что мои веб-сайты отвратительны вовсе не потому, что созданы на базе FreeBSD. Сделать их привлекательными не дают свойственные мне цинизм, сарказм и грубость.

Клиент FTP

FTP – достаточно сложный протокол. В отличие от POP3 и SMTP, его трудно протестировать с помощью команды telnet(1). Для работы с FTP нужен FTP-клиент. Для подключения к хосту достаточно набрать команду `ftp` и указать имя хоста.

```
# ftp sardines
Connected to sardines.blackhelicopters.org.
220 sardines.blackhelicopters.org FTP server (Version 6.00LS) ready.
Name (sardines:mwlucas):
```

По умолчанию клиент посылает локальное имя пользователя, однако при необходимости можно ввести другое имя. Далее будет запрошен пароль:

```
331 Password required for mwlucas.
Password:
230 User mwlucas logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Если все идет, как задумано, то в результате вы зарегистрируетесь на удаленном сервере с оболочкой, подобной командному интерпретатору. Выполнять команды вы не сможете, но сможете перемещаться и просматривать файлы с помощью стандартных команд UNIX, таких как `ls` и `cd`.

Для загрузки файла по FTP служит команда `get` с именем файла:

```
ftp> get .cshrc
local: .cshrc remote: .cshrc
229 Entering Extended Passive Mode (|||50451|)
150 Opening BINARY mode data connection for '.cshrc' (614 bytes).
100% |************************************|   614     272.17 KB/s   00:00 ETA
226 Transfer complete.
614 bytes received in 00:00 (193.48 KB/s)
ftp>
```

Как видите, клиент FTP открывает соединение для передачи файла. По мере загрузки файла по экрану движется строка из звездочек (*). Значение в поле ETA обновляется в соответствии с временем, оставшимся до окончания загрузки. По завершении передачи файла вы увидите извещение, размер загруженного файла и приглашение FTP на ввод команды.

Для выгрузки файла из локальной системы на сервер FTP служит команда `put`. Вывод этой команды практически такой же, как у команды `get`, поэтому я не буду повторять его здесь.

Для перемещения сразу нескольких файлов предназначены команды `mget` и `mput`. Например, если требуется загрузить все файлы с расширением *.html*, введите команду `mget *.html`. Единственная неприятность

состоит в том, что сервер будет запрашивать подтверждение перед передачей каждого файла. Включить или выключить запрос подтверждения можно с помощью команды `prompt`.

Наконец, по FTP можно просматривать содержимое текстовых файлов. Команда `less` отображает содержимое удаленного файла по одной странице, как и обычная команда less(1). Просматривать файлы и затем загружать их – нелепо, но зато с помощью `less` удобно просматривать содержимое *README* и индексных файлов.

Передача двоичных и ASCII-файлов

Различие между передачей двоичных и ASCII-файлов – постоянный источник путаницы из-за того, что по-разному обрабатываются символ возврата и символ разделителя строк. С давних пор в системах DOS и UNIX конец строки обозначался по-разному, что можно увидеть при передаче файлов между двумя системами. В Интернете можно найти много документов, детально описывающих это различие. Есть много статей, в которых сторонники двух систем жестко критикуют противоположные подходы. Вам же надо лишь знать, как действовать в сложившейся обстановке.

Можно передавать и двоичные, и текстовые файлы в двоичном режиме, но нельзя передавать двоичные файлы в режиме ASCII. В UNIX-подобных системах по умолчанию используется двоичный режим, а в операционной системе Windows – режим ASCII. Команда `bin` предписывает серверу FTP выполнять двоичную передачу, а команда `a` – передачу в режиме ASCII. Двоичный режим подходит для обоих случаев, поэтому применяйте именно его, и у вас не будет проблем.

Сервер FTP

Теперь, когда вы узнали, как пользоваться клиентом FTP в системе FreeBSD, давайте посмотрим, как можно реализовать сервис FTP. Первая ваша задача – определиться с тем, как запускать демон ftpd(8), из inetd(8) или в автономном режиме.

По умолчанию FreeBSD запускает демон FTP из inetd(8). Большинство систем получают не так много запросов FTP, поэтому inetd легко справляется с поступающими запросами. Если не требуется поддерживать много сессий FTP одновременно, можно использовать inetd(8). Достаточно раскомментировать строку с описанием службы FTP в файле */etc/inetd.conf* (глава 15) и перезапустить inetd. Любое изменение командной строки запуска ftpd(8) можно внести в файле */etc/inetd.conf*.

Однако если предполагается, что ваш сервер FTP будет обслуживать сотни и тысячи одновременных соединений, то запуск FTP из inetd приведет к дополнительной нагрузке на систему. Чтобы этого не было, настройте использование FTP в автономном (standalone) режиме, подразумевающем постоянное прослушивание сети и самостоятельное обслуживание запросов. Не забудьте убрать запуск ftpd(8) из inetd, а за-

тем добавьте строку `ftpd_enable="YES"` в файл */etc/rc.conf*. После этого можно будет запустить ftpd(8) с помощью сценария */etc/rc.d/ftpd* или перезагрузки.

Вы можете выполнить некоторые подстройки ftpd(8) с помощью ключей командной строки. Добавьте эти ключи в строку запуска ftpd(8) в файле */etc/inetd.conf* или в параметр `ftpd_flags` в файле */etc/rc.conf*. Теперь посмотрим, какие возможности предлагает демон ftpd(8), входящий в состав FreeBSD.

Протоколирование обращений к ftpd(8)

Демон ftpd(8) имеет два уровня протоколирования. Если задать ключ `-l` один раз (значение по умолчанию в */etc/inetd.conf*), ftpd(8) будет протоколировать все успешные и неуспешные попытки регистрации. Если данный ключ указать дважды, ftpd(8) будет протоколировать все операции FTP: загрузку и выгрузку файлов, создание и удаление каталогов и т. д.

Режим «только для чтения»

Иногда требуется запретить выгрузку файлов на сервер, замещение файлов или любые другие операции, приводящие к изменениям в файловой системе сервера. Этот режим прекрасно подходит для организации серверов, разрешающих выполнять только загрузку файлов, таких как сайты-зеркала. Для этого применяется ключ `-r`.

Режим «только для записи»

Возможно, вам потребуется организовать сервер, куда пользователи смогут выгружать файлы, но не смогут загружать их. Для этого применяется флаг `-o`.

Тайм-аут

По умолчанию, если сеанс связи с пользователем простаивает 15 минут (900 секунд), соединение разрывается. Определить новое значение тайм-аута можно с помощью ключа `-t`, которому следует передать значение в секундах.

Управление пользователями FTP

С работающим сервером FTP связаны две типичные проблемы: 1) он позволяет пользователям загружать произвольные системные файлы и 2) пароли FTP могут быть похищены. Стоит ли подвергать свою систему риску из-за того, что какой-то пользователь-разиня, подключившись к серверу из местного интернет-кафе, не уберег свой пароль, и злоумышленник смог загрузить ключевые системные файлы! Лучший способ управлять пользователями заключается в том, чтобы выбрать, кто может входить на сервер и какие каталоги будут доступны этим выбранным пользователям.

chroot для пользователей

Вы можете заблокировать пользователей FTP в своих домашних каталогах с помощью chroot(8). Для пользователя его домашний каталог будет выглядеть как корень файловой системы. Пользователи не смогут покинуть свои домашние каталоги или получить доступ к произвольным системным файлам. Это очень напоминает маленькую клетку. Вариант с chroot полезен для веб-серверов с большим количеством клиентов на одной машине, то есть для веб-серверов с большим числом виртуальных хостов. В конце концов, только пользователь должен видеть свой каталог и больше никто.

Чтобы посадить пользователя в клетку, добавьте его имя в файл */etc/ftpchroot*. Каждый пользователь должен указываться в отдельной строке. Всякий раз, когда пользователь подключается к системе по FTP, его учетная запись сравнивается с содержимым */etc/ftpchroot*. Если имя есть в этом файле, пользователь блокируется в своем домашнем каталоге с возможностью полностью контролировать его (если это позволяют установленные вами права доступа), создавать столько подкаталогов и хранить столько файлов, сколько позволяет отведенное ему дисковое пространство. Но пользователь не может покидать свой домашний каталог и исследовать систему.

В файле */etc/ftpchroot* можно также указывать группы, что будет приводить к блокированию в домашних каталогах всех пользователей, входящих в эту группу. Имена групп должны начинаться с символа `@` (например, `@customers`).

Допустим, в системе есть два веб-дизайнера, Гордон и Крис. Требуется разрешить им выгрузку файлов только в свои домашние каталоги. Есть также группа клиентов, которые поддерживают свои веб-сайты. Клиенты входят в группу webclients. Чтобы «посадить» всех этих пользователей в клетку, настройте */etc/ftpchroot* так:

```
gordon
chris
@webclients
```

Теперь все эти пользователи заперты в своих домашних каталогах.

Запрет доступа к FTP

Имя файла */etc/ftpusers* обманчиво. Файл содержит не список разрешенных пользователей, а список тех, кому *не* позволено входить в систему по FTP. По умолчанию в */etc/ftpusers* представлены разные системные учетные записи, такие как root и nobody. Ни один системный администратор не сможет зарегистрироваться как root!

В этот файл можно добавить группы, поместив перед их именами символ `@`. Обычно я запрещаю доступ к FTP членам группы wheel. Пользователи, применяющие пароль root, не должны передавать свои пароли открытым текстом! Все, кому я могу доверить доступ в систему с при-

вилегиями пользователя root, знают, как пользоваться SSH, scp(1) и sftp(1)!

Сообщения сервера FTP

При подключении сервер FTP может выводить два разных сообщения, текст которых хранится в файлах */etc/ftpwelcome* и */etc/ftpmotd*.

Когда клиент устанавливает соединение FTP впервые, ftpd(8) отображает содержимое файла */etc/ftpwelcome*. В него можно поместить правила пользования, предупреждения, сообщения о предоставляемых возможностях, непристойности, угрозы, то есть все что угодно. Пользователи увидят данную информацию еще до получения приглашения к регистрации. Это замечательное место для сообщения «Несанкционированное использование запрещено». Факт наличия этого сообщения, пусть и не такого забавного, как «Пользователи, совершившее незаконное вторжение, будут преследоваться по всей строгости моим злобным ротвейлером и 12-зарядным дробовиком», пригодится в суде, если кто-нибудь воспользуется вашими услугами в противозаконных целях.

Как только пользователь войдет в систему, отображается содержимое */etc/ftpmotd*. Обычно в нем содержатся соглашения по пользованию услугой.

Настройка анонимного сервера FTP

Анонимные FTP-сайты – это популярный способ предоставления файлов и документов всему Интернету. Однако такие сайты зачастую взламываются. Хотя ftpd(8) системы FreeBSD достаточно устойчив и безопасен, необходимо предпринять меры предосторожности и настроить сервер должным образом. Вот несколько рекомендаций:

- *Не объединяйте* службы анонимного и не анонимного FTP на одном сервере.
- Если возможно, разрешите только считывание файлов с сервера, применив ключ `-r`.
- Применяйте ключ `-S` для протоколирования в */var/log/ftpd* всех операций анонимного сервера FTP. Файл следует создать до запуска ftpd(8), поэтому сначала запустите команду `touch /var/log/ftpd`.
- Создайте непривилегированного пользователя с именем «ftp», с правами которого будет работать ftpd(8). Домашним каталогом этого пользователя будет корень дерева каталогов анонимного сервера FTP; все файлы, которые должны быть доступны всему миру, надо разместить в этом каталоге.
- Создайте каталог */home/ftp/pub* для папки *pub*, традиционно присутствующей на серверах FTP, и измените права доступа к домашнему каталогу пользователя ftp так, чтобы пользователи не могли ничего изменить в этом каталоге. При таких настройках пользователи смогут выгружать свои файлы только в каталог *pub*.

Предостережение по поводу анонимного FTP

Разрешение всем желающим загружать файлы на сервер кажется дружеским шагом. У вас есть достаточная полоса пропускания и вы хотите предоставлять общедоступный сервис. В идеальном мире (каким был Интернет 1980-х) это было бы замечательно.

Однако если разрешить всем желающим хранить данные в вашей системе, пользователи смогут использовать сервер FTP для хранения нелегальных программ, детской порнографии или переписки террористов. Чтобы все это было трудно обнаружить, злоумышленники могут создать скрытые каталоги или замаскировать данные. Даже если вы отслеживаете все барахло, загружаемое пользователями, вы можете не обратить внимание на файл *CookieRecipies.txt*, который фактически является учебным MPEG-фильмом Фронта освобождения хомяков, задача которого – засадить президента в гигантское колесо и заставить его бегать в нем всю оставшуюся жизнь. Вам будет очень нелегко объяснить наличие этого файла службе государственной безопасности.

chroot для sftp(1) и scp(1)

В заключение скажу несколько слов о scp(1) и sftp(1). Эти сервисы, предназначенные для передачи файлов через SSH (глава 15), обеспечивают возможность аутентификации, целостность и строгое соблюдение обязательств при передаче данных. sftp представляет собой защищенный вариант FTP, а scp – вариант rcp(1). Пользователи, выполняющие передачу файлов на сервер, могут использовать эти протоколы вместо FTP.

Одна из проблем, с которыми может столкнуться системный администратор, – заключение пользователей этих сервисов в chroot-окружение. Это здорово, что вы можете предоставить в распоряжение пользователей безопасное соединение с шифрованием данных, но вам попрежнему нежелательно разрешать пользователям загружать произвольные файлы из любого места в системе! Для работы с такими пользователями рекомендую использовать scponly (*/usr/ports/shells/scponly*). Пользователи, для которых в качестве командного интерпретатора указано */usr/local/bin/scponly*, смогут получить доступ к серверу только с помощью sftp и scp.

Теперь, когда вы научились обслуживать веб-сайты и FTP, потратим некоторое время на изучение дисковой подсистемы FreeBSD. Поздоровайтесь с GEOM!

18

GEOM и трюки с дисками

Операционная система FreeBSD обладает чрезвычайно гибкой системой управления дисками, которая называется GEOM. *GEOM* – это инфраструктура, позволяющая разработчикам ядра достаточно просто создавать модули, которые называются *классами GEOM*, для обеспечения функционирования дисков различных типов. Операционная система FreeBSD использует GEOM для обеспечения поддержки шифрования, журналирования, разнообразных программных RAID-массивов и экспортирования дисковых устройств по сети. Инструментальные средства GEOM обеспечивают гибкость, избыточность и простоту реализации как для разработчиков, так и для системных администраторов. В этой главе мы сначала рассмотрим способы разбиения дисков в операционной системе FreeBSD, а затем обратимся к некоторым классам GEOM.

На протяжении всей главы мы будем экспериментировать с дисковыми устройствами. Затрагивая разбиение или формат диска, вы всякий раз рискуете лишиться хранящихся на нем данных. Даже в процессе обучения, работая с неиспользуемой частью диска и не касаясь разделов с данными, помните, что риск все равно есть. *Создайте резервную копию всех своих данных*. Но лучше найти неиспользуемый диск и экспериментировать с ним! (С другой стороны, если вы станете тренироваться на диске, где хранятся ваши драгоценные данные, это заставит вас тщательнее сосредоточиться на выполняемых действиях.)

В частности, мы рассмотрим форматы дисков, используемые в системах i386 и amd64. На других платформах используются несколько отличающиеся системы управления дисками. На платформе SPARC метка диска находится в начале диска, а таблица участков отсутствует. На платформе ia64 используются участки, основанные на GPT, тогда как разбиение дисков ARM – самая простая часть работы по запуску FreeBSD на этой платформе.

Суть GEOM

GEOM – это универсальная платформа, позволяющая производить многоуровневую обработку устройств хранения данных. Различные модули GEOM выполняют представление носителей информации разными способами. Кроме того, эти модули являются наращиваемыми; вывод одного модуля может быть использован как ввод другого. Требуется зеркалировать жесткие диски? Запросто. А как насчет зеркалирования по сети? Нет проблем. А что если необходимо зашифровать диск при зеркалировании по сети? Разбудите меня, когда вам понадобится что-то посложнее, ребята. Замечу, что операционная система FreeBSD не обещает высокую производительность при использовании слишком большого числа уровней GEOM, но дело будет сделано.

Инфраструктура GEOM делит устройства хранения на потребителей и провайдеров. *Потребитель* (*consumer*) лежит ниже уровня модуля, тогда как *провайдер* (*provider*) предоставляет свои услуги следующему, более высокому уровню. Предположим, что мы используем класс GEOM для зеркалирования жестких дисков */dev/da0* и */dev/da1* на виртуальный диск */dev/mirror/mirror0*. Физические дисковые устройства потребляются виртуальным зеркальным диском, при этом комбинированное дисковое устройство */dev/mirror/mirror0* является провайдером, предоставляющим услуги следующему уровню.

С точки зрения FreeBSD все провайдеры одинаковы. Физический жесткий диск – это лишь один из провайдеров. Вполне возможно поместить файловую систему на провайдер или превратить */dev/mirror/mirror0* в потребителя модуля шифрования. В любом случае, чтобы использовать какую-то из расширенных возможностей системы управления дисками, вы должны знать чуть больше.

Дисковые устройства 102

Исторически, данные на диске могут иметь привязку к местоположению на жестком диске. Это местоположение может выражаться в терминах цилиндров, дорожек и секторов. Не забывайте, что каждый жесткий диск состоит из стопки пластин. На каждой пластине есть серия круговых *дорожек* (track), расположенных как годичные кольца в стволе дерева. Эти дорожки хранят данные в виде последовательности нулей и единиц. *Головка* (head) перемещается к конкретной дорожке на определенное расстояние от центра диска и считывает данные по мере вращения пластины под ней. Когда вы запрашиваете данные с конкретной дорожки, головка перемещается, пока нужная дорожка не окажется под ней, и захватывает данные, как это делает родитель, снимая ребенка с карусели.

Если теперь соединить воображаемой поверхностью дорожки на пластинах, расположенных друг над другом, вы получите *цилиндр*. Самые близкие к центру дорожки на всех пластинах образуют цилиндр 0.

Следующие за ними – цилиндр 1. Дорожки с номером 1603 на всех пластинах образуют цилиндр 1602. Многие операционные системы ожидают, что дисковые участки (slices) будут охватывать полные цилиндры, и приходят в ярость, если это не так.

Каждая дорожка делится на сегменты, которые называются *секторами* (sectors), или *блоками* (blocks). Каждый сектор может хранить определенный объем данных (вот уже много лет этот объем составляет 512 байтов) и имеет уникальный номер в пределах дорожки. Нумерация секторов начинается с 1.

Итак, секторы образуют дорожки, из которых образуются цилиндры, и все это составляет *геометрию* диска. Все не так уж сложно, и было бы совсем легко, если бы на это можно было положиться.

Долгие годы производители жестких дисков и операционных систем устанавливали и снимали ограничения. Это относится ко всем аспектам архитектуры компьютеров – возможно, некоторые из вас еще помнят времена, когда у персонального компьютера было не больше 640 Кбайт оперативной памяти. Может быть, среди вас есть даже те, кто считал эти 640 Кбайт невообразимо большим объемом и не знал, что делать с такой прорвой памяти.[1] Производители жестких дисков научились обходить эти ограничения, обманывая BIOS и/или операционную систему. Если вы, к примеру, производитель жестких дисков с 126 секторами на дорожке, а большинство операционных систем понимает только 63 сектора на дорожке, то у вас проблема. Самое простое решение состоит в том, чтобы научить жесткий диск лгать. Если диск заявит, что число секторов у него вдвое меньше фактического, а число пластин вдвое больше, результат вычислений от этого не изменится, и все операционные системы по-прежнему смогут идентифицировать отдельные блоки. Ложь ликвидирует проблему. Знакомясь с историей управления дисками, вы встретите самые разные типы подобных преобразований. Ведущий разработчик FreeBSD очень точно высказался по этому поводу: «x86 BIOS – просто дерьмо».

К тому времени, когда информация достигнет пользователя, весьма возможно, она претерпит не одно такое преобразование. Если у вас имеется аппаратный RAID – в виде локальной карты RAID или внешней системы SAN, информация о геометрии наверняка окажется поддельной. Сегодня важно понимать, как все это организовано логически, а не физически. Примите ложь, которую вам преподносит жесткий диск, даже если это явная ложь.

Многие операционные системы пытаются оптимизировать производительность, опираясь на информацию о цилиндрах. Однако из-за преобразования информации о геометрии эти схемы оптимизации работают не так, как предполагалось.

---

[1] Да знаю, знаю, что вас тогда и на свете-то не было! Пожалуйста, перечитайте сноску в разделе «Рождение FreeBSD» введения.

Деление дисков на участки

Вспомните, как в главе 8 говорилась, что операционная система FreeBSD делит диски двумя различными способами: на *участки* (*slices*) и на *разделы* (*partitions*).[1] Участки – это разделы базовой системы ввода-вывода (BIOS), единственный тип разделов, распознаваемый старыми версиями операционной системы Microsoft Windows. Разделы в операционной системе FreeBSD – это подразделы участка. FreeBSD позволяет конфигурировать участки и разделы независимо друг от друга. На одном диске (в архитектурах i386 и amd64) может присутствовать до четырех участков, а на каждом участке – до восьми разделов. Файловые системы FreeBSD располагаются на разделах внутри участков.

Перед первым участком находится *главная загрузочная запись* (*Master Boot Record, MBR*) – цилиндр 0, головка 0, сектор 1. В MBR хранится таблица участков, а также код начального запуска системы. Размер MBR ограничен 512 байтами – такого объема казалось вполне достаточно во времена, когда появилось это ограничение. Во время загрузки компьютер считывает MBR с загрузочного диска, чтобы идентифицировать участки диска, отыскивает участок с операционной системой и запускает код начального загрузчика на этом участке. Размещать операционную систему FreeBSD можно в любой части диска, расположенной после MBR.

Просмотр таблицы участков с помощью fdisk(8)

Прочитать и отредактировать таблицу участков в MBR позволяет утилита fdisk(8). Эта утилита всегда принимает один параметр – имя дискового устройства. Вот результат запуска fdisk(8) для моего тестового жесткого диска:

```
# fdisk /dev/da2
******* Working on device /dev/da2 *******
parameters extracted from in-core disklabel are:
❶ cylinders=1116 heads=255 sectors/track=63 (16065 blks/cyl)

❷ Figures below won't work with BIOS for partitions not in cyl 1
parameters to be used for BIOS calculations are:
cylinders=1116 heads=255 sectors/track=63 (16065 blks/cyl)

❸ Media sector size is 512
Warning: BIOS sector numbering starts with sector 1
Information from DOS bootblock is:
❹ The data for partition 1 is:
sysid 165 (0xa5),(FreeBSD/NetBSD/386BSD)
```

[1] Термин slice во FreeBSD означает то же самое, что во всех остальных системах называется словом partition. Это те самые штуки, которые бывают первичными и расширенными и которых не может быть больше четырех на одном жестком диске в архитектуре x86. – *Прим. науч. ред.*

```
❺   start 63, size 17928477 (8754 Meg), flag 80 (active)
❻     beg: cyl 0/ head 1/ sector 1;
      end: cyl 1023/ head 254/ sector 63
❼ The data for partition 2 is:
<UNUSED>
The data for partition 3 is:
<UNUSED>
The data for partition 4 is:
<UNUSED>
```

Этот диск SCSI заявляет, что имеет 1116 цилиндров, 255 головок и 63 сектора на дорожку ❶. Вспомните, *головка* – это механическое устройство, которое располагается над пластиной диска. 255 головок едва ли уместятся в корпус жесткого диска! Лично я предпочитаю, чтобы системы, которые мне лгут, брали на себя труд сделать эту ложь более или менее правдоподобной, однако у нас нет иного выбора, кроме как принять заявление жесткого диска и считать его фактом. Неважно, сколько головок, цилиндров и секторов в действительности имеет данный жесткий диск, главное, что общее число секторов равно 17 928 540 (1 116×255×63 = 17 928 540).

В прошлом, когда жесткие диски только начали подделывать информацию, из разных источников можно было получить противоречивые сведения о геометрии диска. Утилита fdisk(8) предоставляет полный объем сведений, которые ей удалось собрать о диске ❷, в надежде, что системный администратор сам разберется, где истина, а где ложь, и сумеет использовать диск. При работе с современными дисками такая потребность возникает довольно редко, а информация в двух первых разделах всегда должна совпадать.

Хотя все жесткие диски должны иметь размер сектора, равный 512 байтам ❸, тем не менее, fdisk(8) все равно проверяет это.

Далее следует информация обо всех четырех возможных участках на диске. Первый раздел ❹ помечен как раздел FreeBSD. Каждая операционная система присваивает участкам свой уникальный идентификационный номер. Число 165 сообщает, что это участок FreeBSD.

Здесь также видно, сколько миллионов секторов ❺ входит в состав этого участка. Участок начинается с сектора 63 (не забывайте про главную загрузочную запись, которая занимает первый цилиндр под первой головкой, или секторы с 1 по 62). Первый участок занимает 17 928 477 секторов, откуда следует, что последний сектор в этом участке имеет порядковый номер 17 928 539 (17 928 477 + 62 = 17 928 539). Диск имеет 17 928 540 секторов, но не забывайте про MBR. Этот диск полон.

Одна из неприятностей, связанных с MBR, состоит в том, что загрузочная запись может адресовать не более 1024 цилиндров на диск. (Это создает ограничение на размер жесткого диска – 504 Мбайт.) Но у этого диска гораздо больше цилиндров! Это означает, что вы не можете использовать информацию о количестве цилиндров ❻ в каждом из

участков для вычисления объема диска. Единственное число, которое имеет смысл, – это количество секторов.

В самом конце перечислены неиспользуемые участки с номерами 2, 3 и 4 ❼.

Резервное копирование таблицы участков

Прежде чем приступать к редактированию таблицы участков на диске, необходимо создать резервную копию старой таблицы. Это позволит восстановить диск в предыдущее состояние, если вы допустите ошибку. Чтобы создать резервную копию, можно запустить fdisk(8) с ключом `-p`:

```
# fdisk -p /dev/da0 > da0.slice.backup
```

Чтобы восстановить таблицу участков из резервной копии, необходимо передать утилите fdisk ключ `-f` и имя файла с резервной копией:

```
# fdisk -f da0.slice.backup /dev/da0
```

Если вы не касались жесткого диска с момента его последнего разделения на участки, возможно вам даже удастся восстановить свои данные. Однако в большинстве случаев восстановление таблицы участков не приводит к восстановлению данных на участках, и вам придется восстанавливать данные из резервной копии.

Изменение таблицы участков

Утилита fdisk(8) позволяет не только увидеть таблицу, но и отредактировать ее в интерактивном режиме или при запуске из сценария. Самый простой способ инициализировать новый диск FreeBSD состоит в том, чтобы потребовать от fdisk(8) содать на нем один большой участок, помеченный как участок FreeBSD. Такая ситуация настолько распространена, что для нее утилита fdisk(8) предусматривает специальный ключ `-I`. Ключ `-B` говорит о необходимости полностью пересоздать MBR, что всегда является лучшим выбором при первичной инициализации диска. Ниже выполняется инициализация диска SCSI */dev/da17*, на котором создается единственный раздел FreeBSD:

```
# fdisk -BI /dev/da17
```

Можно было бы также записать на диск свою таблицу участков в том же формате, который используется утилитой fdisk(8) для создания резервной копии, выполнив запись так, как если бы выполнялось восстановление таблицы из резервной копии. Но для большинства пользователей такой подход не рекомендуется. Когда вы используете резервную копию, fdisk(8) полагает, что вы знаете, что делаете, даже если это совсем не так. Утилита fdisk(8) включает интерактивный режим, в котором она автоматически обнаруживает большинство из возможных и малозаметных ошибок и предлагает варианты, не позво-

ляющие испортить жесткий диск. Я рекомендую всегда использовать интерактивный режим при редактировании таблицы участков. Перевод утилиты fdisk(8) в интерактивный режим производится с помощью ключа `-u`.

Вспомните, как в главе 8 мы использовали программу sysinstall для разбиения жесткого диска на участки с разделами и его форматирования. Используйте fdisk(8) только если программа sysinstall не работает по каким-либо причинам, или когда вам необходим более полный контроль над процессом деления диска.

Давайте с помощью fdisk(8) попробуем разделить наш жесткий диск на два участка примерно одинакового размера. Мы знаем, что весь диск содержит 17 928 477 секторов. Это означает, что каждый из участков будет содержать примерно 8 964 238 секторов.

```
# fdisk -u /dev/da17
******* Working on device /dev/da17 *******
```

Утилита выведет информацию о геометрии диска и затем спросит, действительно ли вы желаете изменить ее. Решение об изменении геометрии практически всегда можно рассматривать как ошибочное, и если вы не изучили свой конкретный диск досконально, лучше оставьте его в покое.

```
Do you want to change our idea of what BIOS thinks ? [n] n
(Перевод: Вы хотите изменить наше мнение о том, что думает BIOS?)
```

После этого на экране появятся сведения о конфигурации первого участка и будет предложено изменить ее:

```
The data for partition 1 is:
sysid 165 (0xa5),(FreeBSD/NetBSD/386BSD)
    start 63, size 17928477 (8754 Meg), flag 80 (active)
        beg: cyl 0/ head 1/ sector 1;
```

Диски и арифметика

Прежде чем приступать к работе с утилитой fdisk(8) или disklabel(8), выполните все арифметические вычисления на бумаге. Если в процессе работы с этими утилитами требуются дополнительные вычисления, также делайте их на бумаге. При работе с дисками приходится оперировать большими числами, и вам придется включить свой драгоценный мозг на полную мощность, чтобы гарантировать их корректность на этапе ввода. Несмотря на то, что операции сложения, вычитания и деления кажутся совсем несложными, все-таки большинство ошибок связано с вычислениями, доступными третьекласснику. Заранее выписывайте вручную все числа – и вам не помешает отсутствие способности решать в уме примеры для третьего класса.

```
        end: cyl 1023/ head 254/ sector 63
Do you want to change it? [n] y
```

В настоящее время участок вмещает в себя все секторы диска. Нам требуется уменьшить число секторов на участке 1, а освободившиеся секторы использовать для создания участка 2. Участок 1 начинается с сектора 63 и должен занимать примерно 8 964 238 секторов. Ниже мы просто передаем эти числа утилите fdisk(8). Чтобы принять значение по умолчанию, просто нажимайте клавишу Enter:

```
Supply a decimal value for "sysid (165=FreeBSD)" [165]
(Перевод: Укажите десятичное значение для "sysid (165=FreeBSD)")
Supply a decimal value for "start" [63]
(Перевод: Укажите десятичное значение для "start")
❶ Supply a decimal value for "size" [17928477] 8964301
(Перевод: Укажите десятичное значение для "size")
❷ fdisk: WARNING: partition does not end on a cylinder boundary
fdisk: WARNING: this may confuse the BIOS or some operating systems
Correct this automatically? [n] y
(Перевод: fdisk: ВНИМАНИЕ: конец раздела находится не на границе цилиндра
 fdisk: ВНИМАНИЕ: это может ввести в заблуждение BIOS и некоторые ОС
 Исправить размер раздела автоматически?)
❸ fdisk: WARNING: adjusting size of partition to 8964207
(Перевод: fdisk: ВНИМАНИЕ: новый размер раздела изменен до)
Explicitly specify beg/end address ? [n]
(Перевод: Будете явно указывать начальный/конечный секторы ?)
sysid 165 (0xa5),(FreeBSD/NetBSD/386BSD)
❹     start 63, size 8964207 (4377 Meg), flag 80 (active)
        beg: cyl 0/ head 1/ sector 1;
        end: cyl 557/ head 254/ sector 63
Are we happy with this entry? [n] y
(Перевод: Вас устраивают эти значения?)
```

Здесь мы позволили системе FreeBSD самостоятельно выбрать начальный сектор для размещения участка, но изменили число секторов на участке ❶. Утилита fdisk(8) произвела некоторые вычисления и заметила, что участок заканчивается не на границе цилиндра ❷. Разница по сравнению с размером, который вычислили мы, очень невелика. Несовпадение окончания участка с границей цилиндра может вызвать неполадки, особенно когда на компьютере установлено несколько операционных систем. Всегда позволяйте fdisk(8) исправить этот недостаток. После этого fdisk(8) показывает новый размер нового участка в секторах ❸ и выводит окончательную информацию о новом участке ❹, прежде чем запросить подтверждение на внесение изменений и перейти к следующему участку:

```
The data for partition 2 is:
<UNUSED>
Do you want to change it? [n] y
Supply a decimal value for "sysid (165=FreeBSD)" [0] 165
Supply a decimal value for "start" [0] 8964270
Supply a decimal value for "size" [0] 8964207
```

Этот участок ранее никогда не использовался, поэтому для него отсутствует информация по умолчанию. Нам необходимо ввести тип участка и номер начального сектора. Для участка 1 было выделено 8 964 207 секторов, MBR занимает 63 сектора, то есть уже использовано 8 964 269 секторов (8 964 207 + 62 = 8 964 269). Мы можем определить начало нового участка в секторе 8 964 270. Всего на диске имеется 17 928 477 секторов, таким образом, незанятыми остались 8 964 207 секторов (17 928 – 8 964 269 = 8 964 207). Давайте посмотрим, что об этих числах думает fdisk(8):

```
fdisk: WARNING: partition does not start on a head boundary
fdisk: WARNING: partition does not end on a cylinder boundary
fdisk: WARNING: this may confuse the BIOS or some operating systems
Correct this automatically? [n] y

(Перевод: fdisk: ВНИМАНИЕ: раздел начинается не на границе головки
 fdisk: ВНИМАНИЕ: конец раздела находится не на границе цилиндра
 fdisk: ВНИМАНИЕ: это может ввести в заблуждение BIOS и некоторые ОС
 Исправить размер раздела автоматически?)
```

Утилита fdisk(8) недолго думает над числами, но она знает, что делает. Позвольте ей внести свои коррективы.

```
fdisk: WARNING: adjusting start offset of partition to 8964333
Explicitly specify beg/end address ? [n]
sysid 165 (0xa5),(FreeBSD/NetBSD/386BSD)
    start 8964333, size 8964207 (4377 Meg), flag 0
        beg: cyl 558/ head 1/ sector 1;
        end: cyl 91/ head 254/ sector 63
Are we happy with this entry? [n] y
```

И снова мы не будем явно указывать номера начального и конечного секторов. fdisk(8) уже сделала всю тяжелую работу, поэтому оставим предлагаемые значения без изменений.

Далее будет предложено изменить оставшиеся два участка. У нас на диске не осталось свободного пространства, поэтому мы никак не сможем задействовать эти участки:

```
Partition 1 is marked active
Do you want to change the active partition? [n]
(Перевод: Раздел 1 помечен как активный
Хотите изменить активный раздел?)
```

Запомните, активный раздел – это участок с корневой файловой системой на нем. Начальный загрузчик передает управление загрузчику системы, который находится на участке, который вы укажете. Если это не загрузочный диск, выбор того или иного активного раздела не играет никакой роли.

```
We haven't changed the partition table yet. This is your last chance.
(Перевод: Таблица разделов еще не изменена. Это ваш последний шанс.)
...
Information from DOS bootblock is:
```

```
(Перевод: Информация из загрузочного блока DOS:)
1: sysid 165 (0xa5),(FreeBSD/NetBSD/386BSD)
    start 63, size 8964207 (4377 Meg), flag 80 (active)
        beg: cyl 0/ head 1/ sector 1;
        end: cyl 557/ head 254/ sector 63
2: sysid 165 (0xa5),(FreeBSD/NetBSD/386BSD)
    start 8964333, size 8964207 (4377 Meg), flag 0
        beg: cyl 558/ head 1/ sector 1;
        end: cyl 91/ head 254/ sector 63
3: <UNUSED>
4: <UNUSED>
Should we write new partition table? [n] y
(Перевод: Следует ли записать новую таблицу разделов?)
```

Это действительно ваш последний шанс предотвратить потерю оставшихся на диске данных. Просмотрите таблицу участков еще раз, и если все в порядке, нажмите клавишу Y, чтобы записать новую таблицу участков на жесткий диск.

Это все, что нужно сделать, чтобы получить новую таблицу участков на диске. Теперь создадим несколько разделов на этих участках.

Деление участков на разделы

Таблицы участков в стиле i386 недостаточно, чтобы удовлетворить требованиям, предъявляемым системой FreeBSD к разделам диска. Таблица участков поддерживает всего четыре раздела, тогда как для работы FreeBSD необходимы следующие разделы: корневой раздел, пространство для свопинга, */usr*, */var* и */tmp*. Кроме того, у системного администратора могут быть свои пожелания к делению диска на разделы. *Метка диска* (*disklabel*) – это специальный блок данных, расположенный в начале участка, который определяет позиции разделов внутри участка. Различные платформы имеют различные форматы и требования для меток дисков – на платформах i386 и amd64 используется формат *bsdlabels*, на платформе sparc64 – *sunlabels* и т. д. Метками дисков на всех платформах управляет утилита disklabel(8). Если вы работаете с незнакомой аппаратной архитектурой, обязательно прочитайте страницу руководства disklabel(8), прежде чем пытаться редактировать диск!

Чтение меток дисков

Чтобы просмотреть метку диска для определенного участка, запустите disklabel(8) с именем устройства участка в качестве аргумента. Например, */dev/ad0s1* – имя устройства для участка 1 на диске ad0:

```
# disklabel /dev/ad0s1
# /dev/ad0s1:
❶8 partitions:
# ❷size     ❸offset ❹fstype [❺fsize ❻bsize ❼bps/cpg]
a:  1048576        0  4.2BSD     2048   16384        8
```

```
  b:  2097152  1048576    swap
  c: 39179889        0  unused        0       0        # "raw" part, don't edit
  d: 10485760  3145728  4.2BSD     2048   16384    28552
  e:  2097152 13631488  4.2BSD     2048   16384    28552
  f: 23451249 15728640  4.2BSD     2048   16384    28552
```

Утилита disklabel показывает, что диск поддерживает до восьми разделов ❶, хотя сконфигурировано только шесть. Имена разделов определяются символами от a до h. Хотя диск и поддерживает «до восьми» разделов, это не означает, что на новом диске у вас действительно имеется восемь разделов. Раздел a традиционно используется как корневой раздел. Никто не помешает вам использовать его для хранения обычных данных, но сам я предпочитаю не рисковать и не вводить в заблуждение себя самого. Раздел b традиционно используется как пространство свопинга. Конечно, операционная система FreeBSD ответ для свопинга только разделы, помеченные как swap в файле */etc/fstab*, но меня вряд ли обрадует факт использования раздела b для хранения данных, всплывший при поиске неисправности в 3 часа ночи. Наконец, раздел c – это метка для всего участка. И хотя FreeBSD больше не требует наличия раздела c, это вовсе не означает, что все дополнительные инструменты, которыми я могу пользоваться, правильно воспримут раздел c с данными на нем! История этого раздела так глубоко уходит в прошлое, что я даже не мечтаю использовать его для хранения данных. Все это означает, что под хранение данных вы можете отвести до шести разделов (от d до h и, возможно, a).

Для каждого раздела указываются размер в секторах ❷ и смещение от начала участка ❸. Смещение – это расстояние от начала участка до начала раздела в секторах. Не забывайте, что нумерация секторов начинается с 0. Раздел c охватывает весь диск, который содержит 39 179 889 секторов, и имеет смещение 0. В этом примере раздел a имеет размер 1 048 576 секторов и смещение 0. Корневой раздел – первый на участке. Раздел b начинается с сектора 1 048 576 на участке, то есть следует сразу за разделом a, и занимает 2 097 152 сектора, откуда следует, что очередной раздел должен начинаться с сектора 3 145 728 (1 048 576 + 2 097 152 = 3 145 728). Проверьте смещение раздела d. Все значения вычисляются простым сложением.

Для каждого раздела нужно указывать тип файловой системы ❹. В операционной системе FreeBSD обычно используются следующие типы файловых систем: 4.2BSD (любые файловые системы FreeBSD, такие как UFS или UFS2), swap (для пространства свопинга) или unused (для раздела c и для пустого пространства). FreeBSD поддерживает и другие типы файловых систем, но довольно странно, когда в этой операционной системе посреди жесткого диска размещается файловая система FAT или CD, и уж точно вы никогда не увидите диск с разделом UNIX Sixth Edition.

Следующие два столбца указывают размеры фрагментов ❺ и блоков ❻ для файловых систем UFS и UFS2. В операционной системе FreeBSD

блоки по умолчанию имеют размер 16 Кбайт, а фрагменты – 2 Кбайт. Структура файловой системы UFS описана в главе 8. Значения по умолчанию прекрасно подходят для большинства современных систем. При наличии файловой системы специального назначения, с файлами только необычного размера, у вас может появиться потребность изменить размер блока. Например, если приложение использует миллионы файлов, не больше 6 Кбайт каждый, то есть смысл уменьшить размер блока до 8 Кбайт. Минимальный возможный размер блока – 4 Кбайт. Если приложение использует лишь несколько огромных файлов, возможно, есть смысл увеличить размер блока до 32 Кбайт или даже до 64 Кбайт. Однако должен предупредить: система FreeBSD предполагает, что размер фрагмента составляет точно одну восьмую часть размера блока. Использование другого отношения размеров блок/фрагмент отрицательно скажется на производительности.

В последней колонке ❼ указано число цилиндров в группе цилиндров этой файловой системы. Всегда, *всегда* позволяйте FreeBSD вычислять это значение. Учитывая объем лжи – простите, я имел ввиду количество «преобразований», которые производят жесткие диски и операционные системы при определении геометрии жесткого диска, – а также тот факт, что вам не удастся извлечь пользу из изменения информации о группе цилиндров, лучше оставить эти значения в покое.

Теперь, когда мы разобрались с тем, что такое метки дисков, попробуем создать свои собственные.

Резервное копирование и восстановление метки диска

Прежде чем выполнять какие-либо операции, необходимо создать резервную копию метки диска:

```
# disklabel /dev/da0s1 > da0s1.label.backup
```

При наличии этого файла вы сможете восстановить метку диска в ее прежнее состояние с помощью ключа `-R`:

```
# disklabel -R /dev/da0s1 da0s1.label.backup
```

Ап! – и ваша старая метка диска восстановлена!

Редактирование метки диска

Чтобы заново делить участок на разделы, необходимо изменить метку диска. Изменение метки диска влечет за собой уничтожение разделов на диске. Это означает, что все данные на этом диске будут утрачены. (Это не значит, что они полностью исчезли с диска, – их просто невозможно найти.) Я рекомендую редактировать метки дисков только на новых дисках, не содержащих данных, или на дисках, для которых были сделаны резервные копии. Вы не сможете редактировать метку диска для участка со смонтированными разделами.

Вот пример метки диска для моего тестового диска:

```
# disklabel /dev/da0s1
# /dev/da0s1:
8 partitions:
#       size  offset   fstype  [fsize bsize bps/cpg]
 c: 17767827       0   unused       0     0         # "raw" part, don't edit
 d: 17767827       0   4.2BSD    2048 16384  28552
```

Теперь нужно решить, как поделить этот диск. Согласно информации, полученной от fdisk(8), на этом участке есть 8 675 Мбайт свободного пространства. Мне требуется пространство свопинга размером 1 Гбайт и два раздела по 2 Гбайт, а четвертый раздел должен занять оставшееся пространство (примерно 3,5 Гбайт). На всех этих разделах будут использоваться стандартные размеры блоков и фрагментов. Теперь нужно запустить disklabel(8) с ключом `-e`, и мы получим копию метки диска в текстовом редакторе. Чтобы создать четыре нужных мне раздела, я отредактировал метку так:

```
# /dev/da0s1:
8 partitions:
#       size  offset  fstype  [fsize bsize bps/cpg]
 b: 1G             *    swap
 c: 17767827       0  unused       0     0         # "raw" part, don't edit
 d: 2G             *  4.2BSD    2048 16384 28552
 e: 2G             *  4.2BSD
 f: *              *  4.2BSD
```

Куда делись все числа – количество секторов, смещения? Утилита disklabel(8) распознает сокращения `K` (Кбайт), `M` (Мбайт) и `G` (Гбайт) и может самостоятельно выполнить необходимые вычисления, чтобы исходя из размера раздела получить количество секторов. Нам достаточно указать размер каждого раздела в требуемых единицах измерения. Обратите внимание на раздел `f`. Символ звездочки (`*`) сообщает, что disklabel(8) должна отдать этому разделу все остальное пространство. Кроме того, здесь можно использовать проценты, сказав, например: «Этот раздел должен занимать 50 процентов оставшегося свободного пространства».

Помимо этого disklabel(8) может вычислить смещение для каждого раздела. Символ звездочки (`*`) сообщает утилите disklabel(8), что она сама должна вычислить величину смещения.[1]

Наконец, обратите внимание на размеры блоков и фрагментов. Для раздела `d` они указаны явно, но лишь потому, что в первоначальной метке уже имелся раздел `d` с этими значениями, а я просто поленился

[1] Не забывайте, что все функциональные возможности BSD появились только потому, что они были необходимы авторам программ. Очевидно, авторы disklabel(8) не смогли в третьем классе усвоить арифметику, так что вы не одиноки в этом.

стереть их. Утилита disklabel(8) будет использовать значения по умолчанию, если явно не указано иное.

Сохраните плоды своего труда, закройте редактор и взгляните на новую метку диска:

```
#        size   offset  fstype  [fsize bsize bps/cpg]
 b:  2097152       16    swap
 c: 17767827        0  unused       0     0         # "raw" part, don't edit
 d:  4194304  2097168  4.2BSD    2048 16384 28552
 e:  4194304  6291472  4.2BSD       0     0     0
 f:  7282051 10485776  4.2BSD       0     0     0
```

Как видите, были вычислены размеры разделов в секторах и смещения, но как быть с размерами блоков и фрагментов для разделов `e` и `f`? Когда вы будете создавать файловые системы на этих разделах, утилита newfs(8) заполнит их значениями по умолчанию.

Копирование структуры деления диска на участки и разделы

Итак, вы научились выполнять резервное копирование таблиц участков и меток диска и восстанавливать их из резервных копий. А нельзя ли с их помощью воссоздать структуру участков и разделов на идентичных дисках? Да, это возможно! Но в первую очередь нужно убедиться, что диски действительно идентичны. Не все диски одного и того же размера имеют одинаковое число секторов. Если диски немного отличаются друг от друга, то вы сможете скопировать структуру разделов наименьшего диска на все остальные.

Сначала отредактируйте таблицы участков и разделов на первом диске в точном соответствии со своими потребностями. Создайте резервные копии меток диска и таблиц участков, а затем «восстановите» эти таблицы на других дисках. Например, на диске */dev/da0* только один участок. Давайте выполним копирование таблицы участков этого диска и метки этого участка, а затем восстановим их на диске */dev/da2*:

```
# fdisk -p /dev/da0 > da0.slice.table
# disklabel /dev/da0s1 > da0s1.disklabel
# fdisk -f da0.slice.table /dev/da2
# disklabel -R /dev/da2s1 da0s1.disklabel
```

Проверьте новую конфигурацию диска */dev/da2* с помощью утилит fdisk(8) и disklabel(8). Этот прием поможет создать серию дисков с идентичным делением, например, если понадобится выполнить сборку массива дисковой памяти с помощью классов GEOM.

Отсутствующие метки диска

Иногда встречаются участки, на которых вообще нет метки диска. У меня такая ситуация обычно возникает с тестовыми дисками, где я порой заново выполняю разбиение на участки и разделы для проверки

своих сумасбродных идей. Если на участке нет метки диска, ее можно создать с помощью ключа -w:

```
# disklabel -w /dev/da0
```

Эта новая метка диска будет иметь единственный раздел, охватывающий весь участок. Редактировать существующую метку гораздо проще, чем создавать ее на пустом месте.

Создание файловых систем

Создав участки и разделы, вы уже практически подготовили диск к использованию. Все, что остается сделать, – это создать файловые системы. Создать новую файловую систему UFS можно с помощью утилиты newfs(8). Используйте ключ -U, чтобы разрешить применение механизма Soft Updates. О некоторых других параметрах newfs(8) мы поговорим в оставшейся части этой главы, а пока просто создадим стандартную файловую систему UFS2:

```
# newfs -U /dev/da2s1d
/dev/da2s1d: 2048.0MB (4194304 sectors) block size 16384, fragment size 2048
        using 12 cylinder groups of 183.77MB, 11761 blks, 23552 inodes.
        with soft updates
super-block backups (for fsck -b #) at:
 160, 376512, 752864, 1129216, 1505568, 1881920, 2258272, 2634624, 3010976,
 3387328, 3763680, 4140032
```

В самом начале newfs(8) сообщает основные сведения о новой файловой системе, такие как ее размер, количество секторов и блоков, а также размеры блоков и фрагментов. Далее следует количество индексных дескрипторов (inodes). В ходе работы утилита newfs(8) выводит информацию о местоположении каждого суперблока – только затем, чтобы вы видели: что-то происходит. (Это было очень важно в эпоху медлительных компьютеров – процесс создания большой файловой системы мог длиться несколько минут, и утилита тактично намекала, что она не зависла и продолжает работать.)

Вы можете указать размеры блоков и фрагментов с помощью ключей -b и -f, соответственно. Если эти параметры не указаны в командной строке, newfs(8) проверит содержимое метки диска и возьмет эту информацию оттуда. Если эти значения не были указаны при создании метки диска, будут использованы значения по умолчанию. Лично я предпочитаю указывать эти значения в метке диска просто потому, что это дает отличную возможность дважды проверить себя. В любом случае newfs(8) обновит информацию о размерах блоков и фрагментов для последующего использования.

Теперь, когда вы научились создавать участки и разделы, а также форматировать диски, перейдем к рассмотрению более сложных тем, связанных с управлением дисками. Одна из таких тем – RAID.

RAID

Одно из важнейших требований, предъявляемых к любой серьезной системе хранения данных, – это наличие *RAID*, или *Redundant Array of Independent Disks* (*массив независимых дисковых накопителей с избыточностью*). Раньше символ *I* в аббревиатуре RAID означал *Inexpensive* (недорогой), но это определение относительно. Массив RAID емкостью в один петабайт стоит намного меньше отдельного диска емкостью в один петабайт, тем не менее, он очень дорог. Системы RAID распределяют данные между дисками для улучшения производительности и надежности. Существуют программные и аппаратные реализации RAID.

Сравнение программных и аппаратных RAID

FreeBSD поддерживает аппаратные и программные RAID. Аппаратным массивом RAID управляет контроллер SCSI, а хост-адаптеры, способные обслуживать RAID, называются *RAID-контроллерами*. В аппаратной реализации все вычисления, связанные с размещением данных на жестких дисках, выполняет сам RAID-контроллер. Большинство аппаратных RAID очень надежны, а аппаратный контроллер – бесспорно, лучший способ обслуживания RAID.

Одна из проблем, связанных с аппаратными RAID, состоит в том, что разные контроллеры хранят данные на дисках в разных форматах. Если RAID-контроллер выходит из строя, вы не сможете использовать контроллер другого производителя, так как более чем вероятно, что он не сможет распознать данные на ваших дисках. Если вы храните критически важные данные в аппаратном RAID-массиве, у вас должно быть либо соглашение на оказание технической поддержки, согласно которому в ваше распоряжение в случае поломки будет предоставлен идентичный контроллер, либо запасной контроллер того же производителя и той же модели, либо самая свежая резервная копия.

Программными RAID управляет операционная система. Именно ОС приходится управлять распределением данных по дискам. В сравнении с аппаратными RAID этот метод увеличивает нагрузку на систему, однако обходится дешевле аппаратного. Кроме того, вы не рискуете потерять данные, как в случае с аппаратным RAID, когда оказывается невозможным приобрести RAID-контроллер того же типа.

Применять аппаратный RAID намного легче, чем программный, поскольку обычно в первом случае достаточно следовать инструкциям из документации. Как правило, в аппаратных RAID есть BIOS, управляемая меню. BIOS позволяет назначать размеры виртуальных дисков, восстанавливать поврежденные диски и конфигурировать виртуальные диски. С другой стороны, применение программных RAID подразумевает, что системные администраторы отдают себе отчет в своих действиях. Если вы хотите получить больше информации об аппаратном RAID, прочитайте руководство по вашему RAID-контроллеру, по-

скольку здесь мы рассматриваем программные RAID. Многие особенности RAID, поддерживаемые операционной системой FreeBSD, реализованы и в аппаратных RAID.

Многие производители RAID поставляют программные инструменты управления своими контроллерами из операционной системы. Контролировать и настраивать массивы amr(4) можно с помощью megarc (*/usr/ports/sysutils/megarc*), массивы mfi(4) – с помощью MegaCli (*/usr/ports/sysutils/linux-megacli*), а массивы aac(4) arrays – с помощью aaccli (*/usr/ports/sysutils/aaccli*). Компания 3Ware поставляет программное обеспечение для управления контроллерами twe(4) и twa(4), а компания Areca – для управления контроллерами arcmsr(4).

GEOM RAID и размер диска

Все жесткие диски в RAID-системе должны иметь одинаковый размер, или хотя бы следует использовать одинаковый объем дискового пространства на каждом диске. Если RAID-массив состоит из 10 дисков по 500 Гбайт и один из них выходит из строя, то в случае установки нового диска размером 700 Гбайт на нем будет использовано только 500 Гбайт. Вам не удастся использовать эти оставшиеся 200 Гбайт без потери производительности всего массива.

Далее считаем, что все диски или участки, используемые для создания программного RAID, имеют одинаковые размеры. Если у вас несколько дисков, проверьте их размеры с помощью fdisk(8). Если размеры некоторых дисков отличаются, даже всего на пару сотен секторов, GEOM использует размер наименьшего диска в качестве ограничения.

Технически у вас не должно быть таблицы участков на томах, предназначенных для хранения данных. Вы можете использовать RAID с разбивкой на блоки или с зеркалированием поверх неформатированных дисков – для GEOM это не имеет никакого значения. В примерах мы не будем использовать таблицы участков.

Контроль четности и размер чередующейся области

Многие типы RAID используют контроль по четности для выявления ошибок. *Контроль по четности* (*parity*) основан на сравнении простых, очень простых контрольных сумм и дает возможность убедиться в безошибочном копировании данных. Система RAID использует контроль по четности для выявления ошибок, чтобы можно было определить истинное состояние области диска. За счет этого обеспечивается избыточность информации дискового массива. Для RAID-систем, использующих контроль по четности, в общем случае необходимо выделить некоторое дисковое пространство для хранения контрольной суммы, но большинство таких систем конфигурируют эту область самостоятельно.

Похожим образом на уровнях RAID, где происходит распределение данных между несколькими дисками, могут использоваться чередую-

щиеся области различных размеров. *Размер чередующейся области*[1] (*stripe size*) – это размер части файла, записываемой на один диск. Например, если файл имеет размер 512 Кбайт, а размер чередующейся области составляет 128 Кбайт, то этот файл окажется записанным в четыре чередующиеся области, разбросанные по дискам. Вам попадутся на глаза руководства, утверждающие, что по тем или иным причинам наиболее оптимальным является тот или иной размер области. Аргументы, которые там приводятся, могут даже быть истинными, но *для определенных видов рабочих нагрузок*. Если у вас появится желание поэкспериментировать с размерами чередующихся областей, проводите испытания на своем аппаратном окружении при типичной рабочей нагрузке. Вы можете получить некоторый прирост производительности, но, скорее всего, он будет не очень велик. В примерах мы будем использовать значение по умолчанию, равное 128 Кбайт, если явно не указано другое. Размер 128 Кбайт оптимален для большинства видов рабочих нагрузок.

Типы RAID

Есть разные типы RAID. Большинство аппаратных реализаций поддерживают RAID-0, RAID-1 и RAID-5. Но есть и другие типы. В частности, FreeBSD поддерживает удобный в некоторых случаях тип RAID-3. Можно также встретить RAID 0+1 и RAID-10, и очень редко – RAID 50 и RAID 6.

Тип RAID-0, или *разбивка на чередующиеся области* (*striping*), в действительности не обеспечивает избыточность и формально вообще не является RAID. Он требует наличия по крайней мере двух дисков, данные между которыми распределяются таким образом, что повышаются производительность и общая емкость массива. RAID-0 можно применять для объединения нескольких дисков по 500 Гбайт и создания одного многотерабайтного виртуального диска. Однако в случае выхода из строя одного диска будут уничтожены данные на всех дисках.[2] Для доступа к информации потребуется восстанавливать данные из резервной копии. RAID-0 полезен в тех случаях, когда нужна одна очень большая файловая система, не только не дающая выигрыша в надежности, но и более уязвимая, чем файловая система на единственном диске. Объем дискового массива RAID-0 равен сумме объемов жестких дисков, из которых он собран.

Тип RAID-1 называют *зеркалированием* (*mirroring*). Содержимое одного диска дублируется на другом. Это хороший метод, недорогой и пре-

---

[1] В литературе в качестве перевода термина stripe встречается также термин «полоса». – *Прим. науч. ред.*

[2] Данные не будут уничтожены и даже не перестанут быть доступными для чтения – они просто потеряют смысл, так как произвольные части многих файлов будут утрачены вместе с погибшим диском. – *Прим. науч. ред.*

доставляющий надежную схему хранения данных. Я применяю зеркалирование на всех своих серверах. Зеркалирование может обеспечить дополнительную защиту данных даже для сервера, созданного на базе настольного персонального компьютера. Объем дискового массива RAID-1 равен объему наименьшего из жестких дисков в массиве.

Тип RAID-3, или *разбивка на чередующиеся области с выделенным диском для хранения контрольных сумм* (*striping with a dedicated parity disk*), использует запасной жесткий диск для хранения контрольных сумм и обеспечения целостности данных, чередующихся по остальным жестким дискам. Это означает, что выход одного жесткого диска из строя не приведет к потере данных. Чтобы использовать дисковые массивы типа RAID-3, у вас должно быть нечетное число дисков, но не меньше трех. В некоторых случаях наличие выделенного диска для хранения контрольных сумм позволяет увеличить производительность. Однако в каждый конкретный момент времени массив RAID-3 может удовлетворить только один запрос на выполнение операции ввода-вывода. Если вы последовательно загружаете большие файлы, то стоит обдумать вариант с применением RAID-3. Объем дискового массива RAID-3 равен сумме объемов жестких дисков, из которых он собран, за исключением одного.

Тип RAID-5, или *разбивка на чередующиеся области с хранением контрольных сумм на всех дисках* (*striping with parity shared across all drives*), – это текущий промышленный стандарт обеспечения избыточности. Как и RAID-3, этот тип применяет контроль по четности, обеспечивающий избыточность данных; потеря одного диска не приведет к уничтожению данных. В отличие от RAID-3, пространство для хранения контрольных сумм распределяется между всеми жесткими дисками. Пропускная способность самая обычная, но массив RAID-5 может обслуживать сразу несколько операций ввода-вывода. Объем дискового массива RAID-5 равен сумме объемов жестких дисков, из которых он собран, за исключением одного. К сожалению, у FreeBSD 7 нет стабильной программной реализации RAID-5. Вам может встретиться упоминание о gvinum(8) – это «порт» менеджера томов Vinum, основанный на GEOM и включающий поддержку RAID-5, – но это программное обеспечение еще не гарантирует полную надежность. Другие реализации находятся на стадии разработки и в настоящее время недоступны.

Тип RAID 0+1 – *зеркалирование дисков, разбитых на чередующиеся области* (*mirror of striped disks*). Для организации дискового массива этого типа вам потребуется четное количество дисков, но не меньше четырех. Суть в том, чтобы сначала создать дисковый массив типа RAID 0, получив один большой виртуальный диск, а затем выполнить зеркалирование всего массива на идентичный набор жестких дисков. Объем дискового массива RAID 0+1 равен половине суммы объемов жестких дисков. Массив RAID 0+1 не обеспечивает отказоустойчивость, как RAID-10, и потому используется довольно редко. Мы не будем обсуждать этот тип.

Тип RAID-10 – *разбивка на чередующиеся области зеркалированных дисков* (*stripe of mirrored disks*). Для реализации такой схемы требуется как минимум четыре диска. Количество дисков должно быть четным, для полной надежности нужны два отдельных контроллера дисков. Диски объединяются в зеркальные пары, а затем данные записываются в чередующиеся области, распределенные по зеркалам. Поскольку при такой реализации контрольные суммы не вычисляются, она обладает самой высокой производительностью. Объем дискового массива RAID-10 равен половине суммы объемов жестких дисков.

Инфраструктура GEOM может предоставить программную реализацию всех этих типов RAID.

Универсальные команды GEOM

Большинство команд, имеющих отношение к GEOM (такие как gstripe(8), graid3(8) и прочие), поддерживают набор общих подкоманд для управления различными операциями GEOM. Все эти команды можно использовать через geom(8), но я считаю, что удобнее использовать их вместе с конкретными модулями, такими как gstripe(3). Например, подкоманда `status` показывает текущее состояние определенной подсистемы GEOM. Я могу запросить у GEOM информацию о состоянии всех моих устройств RAID-3, выполнив команду `geom raid3 status`. Но, на мой взгляд, гораздо удобнее использовать graid3(8) и вводить команду `graid3 status`. Во всех примерах этой главы используется вторая форма представления команд. Почти у каждого модуля GEOM имеются четыре команды: `load`, `unload`, `list` и `status`.

Команда `load` активизирует модуль ядра этого класса. Например, команда `gstripe load` загружает модуль ядра *geom_stripe.ko*. Аналогично, команда `unload` завершает работу выбранного модуля ядра.

Команда `status` выводит информацию о состоянии устройств данного типа. Вот информация о состоянии зеркалированных через GEOM дисков на моем почтовом сервере:[1]

```
# gmirror status
      Name    Status  Components
mirror/gm0  COMPLETE  ad1
                      ad2
```

Формат этого вывода зависит от используемого модуля GEOM в соответствии с типом устройства.

Наконец, команда `list` выводит список всех модулей *потребителей*, или устройств, находящихся ниже уровня модуля. Команда `list` часто

[1] «И после всех этих жалоб на диски SCSI на серверах вы используете диски IDE?» Да, но это мой личный почтовый сервер. Я получаю электронной почты больше, чем хотелось бы, и если диски сгорят, я не расстроюсь. Сомневаюсь, чтобы ваш начальник думал так же.

играет роль синонима «прочая информация о модуле, которая отсутствует в других трех командах» и используется при поиске неисправностей. Вот фрагмент вывода команды `list` для устройства graid3:

```
# graid3 list
❶ Geom name: MyRaid3
State: COMPLETE
Components: 3
❷ Flags: ROUND-ROBIN
GenID: 0
...
❸ Providers:
❹ 1. Name: raid3/MyRaid3
     Mediasize: 18210036736 (17G)
     Sectorsize: 1024
     Mode: r1w1e1
Consumers:
❺ 1. Name: da0
...
```

На данном сервере присутствует устройство graid3 с именем MyRaid3 ❶. Далее следует информация, свойственная большинству устройств, основанных на GEOM: устройство укомплектовано полностью (в том смысле, что все диски на месте, в рабочем состоянии и данные на них синхронизированы) и имеет трех потребителей. Далее следует отладочная информация модуля graid3 ❷.

Под заголовком `Providers` (провайдеры) ❸ выводится список устройств, предоставляемых этим классом GEOM. Этот сервер предоставляет устройство */dev/raid3/MyRaid3* ❹ типа RAID-3. Здесь можно видеть объем устройства, размер блоков и прочую информацию о диске.

Кроме того, здесь имеется раздел с заголовком `Consumers` (потребители) ❺, в котором перечислены все диски, составляющие устройство GEOM. Здесь можно найти информацию не только о самих дисках, но и о том, как эти диски взаимодействуют с модулем GEOM.

Данные команды составляют интерфейс к различным модулям GEOM, обеспечивающим программную виртуализацию дисков.

Чередование данных на дисках

Для настройки и управления чередованием данных на дисках в операционной системе FreeBSD предназначена команда gstripe(8). Она позволяет создать один виртуальный диск из двух или более дисков. Преимущество чередования данных: обеспечивается возможность распределения операций ввода-вывода между несколькими дисками в массиве, что теоретически увеличивает пропускную способность во столько же раз, сколько дисков входит в состав массива. (Практика редко подтверждает теорию!)

Например, у меня есть три жестких диска – */dev/da0*, */dev/da1* и */dev/da2*. На каждом диске имеется единственный участок с единственным разделом (`d`) на этом участке. Вы можете вручную конфигурировать чередование данных на дисках всякий раз, когда это понадобится, а я буду исходить из предположения, что это необходимо делать автоматически, во время загрузки.

Для начала нужно загрузить модуль ядра geom_stripe с помощью команды `gstripe load`. Можно также добавить в файл */boot/loader.conf* строку `geom_stripe_load="YES"`, чтобы загрузить механизм чередования на этапе загрузки.

Создание провайдера с чередованием данных

Теперь нужно сообщить дискам, что они объединены провайдером с функцией чередования. В приведенном ниже примере создается устройство чередования с именем MyStripe. Это устройство включает в себя три диска. Ключ `-s` задает размер области чередования, равный 131 072 байт, или 128 Кбайт. Ключ `-v` включает режим более подробного вывода результатов работы gstripe(8):

```
# gstripe label -v -s 131072 MyStripe /dev/da0 /dev/da1 /dev/da2
Metadata value stored on /dev/da0.
Metadata value stored on /dev/da1.
warning: /dev/da2: only 9105018368 bytes from 9186602496 bytes used.
Metadata value stored on /dev/da2.
Done.
```

Здесь видно, что команда gstripe(8) сохранила на дисках информацию о провайдере чередования. Страница руководства утверждает, что эта информация сохраняется в последнем секторе диска. Здесь также видно, что диск */dev/da2* немного больше других, но команда gstripe(8) достаточно интеллектуальна, чтобы исключить из использования дополнительное пространство на этом диске.

Теперь, когда создан провайдер чередования, загляните в каталог */dev/stripe*:

```
# ls /dev/stripe/
MyStripe
```

У нас появилось дисковое устройство */dev/stripe/MyStripe*. Этот виртуальный диск не требует наличия метки диска или таблицы участков, хотя при необходимости вы можете создать их. Замечу, что объединение нескольких дисков в один большой раздел с тем, чтобы потом снова разделить его, выглядит, мягко говоря, несколько странно. Вам нужна лишь файловая система и точка монтирования нового устройства. Не забудьте запустить newfs(8) с ключом `-U`, чтобы включить механизм Soft Updates.

```
# newfs -U /dev/stripe/MyStripe
/dev/stripe/MyStripe: 26042.8MB (53335720 sectors) block size 16384,
```

```
fragment size 2048
        using 142 cylinder groups of 183.77MB, 11761 blks, 23552 inodes.
        with soft updates
super-block backups (for fsck -b #) at:
 160, 376512, 752864, 1129216, 1505568, 1881920, 2258272, 2634624, 3010976,
3387328,
...
```

Взгляните внимательнее на этот вывод команды newfs(8). Каким-то образом newfs(8) удалось отыскать группы цилиндров на этом виртуальном диске. Утилита fdisk(8) сообщает, что обнаружила цилиндры, головки и секторы. Это лишний раз доказывает, что информация о цилиндрах, головках и секторах не много значит; только информация о секторах имеет хоть какой-то смысл.

Теперь, когда можно смонтировать и использовать диск с чередованием данных, укажите устройство из каталога */dev/stripe* в файле */etc/fstab* как любое другое дисковое устройство.

Останов gstripe

Если нужно остановить работу механизма чередования (например, когда требуется обновить что-то, связанное с избыточностью), следует сообщить дискам, что они больше не участвуют в чередовании. Для этого отмонтируйте устройство чередования и выполните команду `gstripe clear /dev/da0 /dev/da1 /dev/da2`, чтобы стереть с дисков конфигурацию чередования. После этого можно выгрузить модуль ядра.

Ежедневная проверка состояния

Если вы хотите, чтобы FreeBSD производила проверку состояния устройств gstripe(8) каждый день и включала результаты в ежедневный отчет periodic(8), просто добавьте строку `daily_status_gstripe_enable="YES"` в файл */etc/periodic.conf*.

Зеркалирование дисков

Конфигурирование и управление механизмом зеркалирования дисков во FreeBSD производится с помощью команды gmirror(8). Традиционно зеркало состоит из пары дисков, но вы можете создать зеркало с любым числом дисков. Зеркальные диски – одна из самых простых форм обеспечения отказоустойчивости: если из строя выходит один диск, данные остаются на зеркале. При создании зеркала в идеале следует выбирать физически отделенные друг от друга диски: на разных контроллерах, подключенные к разным кабелям и даже к разным серверам, если это возможно. В данном примере я использую два жестких диска SCSI – */dev/da0* и */dev/da1*.

Зеркальное копирование метаданных производится в конец каждого раздела с замещением любых данных, которые там могут находиться.

Я рекомендую зеркалировать только вновь созданные диски и разделы, так как любые данные в этих разделах обычно начинают записываться с начала раздела. В старых разделах файлы часто разбросаны по всему пространству.

Загрузите модуль ядра geom_mirror или выполните команду `gmirror load`. Чтобы активизировать поддержку зеркалирования на этапе загрузки, добавьте строку `geom_mirror_load="YES"` в файл */boot/loader.conf*.

Создание зеркала

Команда gmirror(8) очень напоминает команду gstripe(8): выберите диски для объединения в зеркало и с помощью gmirror(8) пометьте эти диски как зеркальную пару. Однако, в отличие от gstripe(8), зеркала обычно создаются для дисков, на которых уже хранятся какие-то данные. В этом примере я уже использую диск */dev/da7* для раздела */usr/ports*. (Точнее, я использую устройство */dev/da7s1d*, но это всего лишь раздел на диске.) Настраивать зеркалирование можно только на отмонтированных дисках, поэтому я сначала должен отмонтировать раздел и только потом попытаться создать зеркало. Сама операция включения диска в зеркало выполняется очень быстро.

```
# umount /usr/ports/
# gmirror label -v MyMirror /dev/da7
Metadata value stored on /dev/da7.
Done.
# mount /dev/da7s1d /usr/ports/
mount: /dev/da7s1d : No such file or directory
```

Ой! Что с моим разделом? Не волнуйтесь. Просто диск стал частью устройства-зеркала с именем MyMirror в каталоге */dev/mirror*:

```
# ls /dev/mirror/
MyMirror MyMirrors1 MyMirrors1c MyMirrors1d
# mount /dev/mirror/MyMirrors1d /usr/ports/
```

Проверка каталога */usr/ports* показала, что данные снова доступны.

Просмотреть состояние зеркала можно с помощью команды `status` утилиты gmirror:

```
# gmirror status
           Name    Status  Components
mirror/MyMirror  COMPLETE  da7
```

Информация на всех дисках зеркала полностью синхронизирована. Поскольку здесь только один диск, этого следовало ожидать! Добавлять диски в зеркало можно с помощью команды `insert`. Команда принимает два аргумента: имя зеркала и добавляемый диск:

```
# gmirror insert MyMirror /dev/da2
```

Проверьте состояние зеркала после добавления диска:

```
# gmirror status
           Name    Status  Components
mirror/MyMirror  DEGRADED  da7
                           da2 (11%)
```

Синхронизация данных на зеркале еще не закончилась, копирование данных на зеркальный диск может занять некоторое время. В зависимости от объема диска и скорости работы дисковой подсистемы это может занять от нескольких минут до нескольких часов. Вы в любой момент сможете проверить состояние зеркала, чтобы увидеть, как движется процесс синхронизации.

Восстановление зеркала

Основное назначение зеркала – избыточность. Когда один жесткий диск выходит из строя, желательно иметь возможность восстановить данные и/или произвести действия по обслуживанию. Способность продолжать работу диктуется аппаратными средствами. Если в вашей системе имеются диски SCSI или SATA, допускающие возможность горячей замены, вы можете восстановить зеркало прямо на ходу, но если вы используете диски ATA, то, скорее всего, вам придется остановить систему, чтобы заменить вышедший из строя жесткий диск.

Сначала удалите испорченный диск из зеркала с помощью команды `forget` утилиты gmirror(8). Несмотря на пугающее название, эта команда всего лишь предписывает утилите gmirror удалить все устройства, недоступные в настоящий момент, из конфигурации зеркала. Команда `forget` не стирает данные на зеркале! Затем просто используйте команду `insert` для добавления диска в конфигурацию, имя которой нужно указать первым:

```
# gmirror forget MyMirror
# gmirror insert MyMirror /dev/da8
```

Зеркалирование загрузочных дисков

Создать зеркало загрузочного диска не так сложно, как может показаться. Зеркало можно создать даже в процессе установки, с помощью командного интерпретатора восстановления (emergency shell), но мы пойдем более простым путем и сделаем это после установки.

Сначала нужно добавить загрузку модуля ядра geom_mirror в файл */boot/loader.conf*. Без этого FreeBSD не найдет загрузочный диск, так как это будет устройство, не опознанное ядром. Рекомендую после этого перезагрузить систему, чтобы модуль geom_mirror наверняка загрузился на этапе загрузки.

Следующая наша задача – помочь FreeBSD отыскать загрузочное устройство. По умолчанию FreeBSD устанавливается в корневой раздел устройства */dev/ad0s1a*. Однако после включения корневого диска в зеркало корневой раздел будет располагаться на другом устройстве,

например на */dev/mirror/MyBootDrives1a*. Остальные разделы и участки будут располагаться на похожих устройствах. Для корректной загрузки FreeBSD вы должны описать их в файле */etc/fstab*. Кое-кто пускается на разные хитрости, чтобы отредактировать */etc/fstab* в однопользовательском режиме, где недоступен даже редактор vi(1). Вы можете сделать это в sed(1) или ed(1) или даже смонтировать раздел */usr*, чтобы можно было использовать vi(1). Но почему бы не внести нужные изменения в многопользовательском режиме, сохранив их в отдельном файле?

Скопируйте файл */etc/fstab* в файл */etc/fstab-mirror*. Отредактируйте этот файл, чтобы назначить новые имена устройств вашим разделам. Допустим, предполагается создать устройство RootMirror0 для загрузочного диска. Вот содержимое небольшого файла */etc/fstab*:

```
/dev/ad0s1a                     /       ufs     rw      1       1
/dev/ad0s1e                     /usr    ufs     rw      2       2
/dev/ad0s1d                     /var    ufs     rw      2       2
```

После создания зеркала FreeBSD ожидает, чтобы файл */etc/fstab* содержал примерно следующее:

```
/dev/mirror/RootMirror0s1a      /       ufs     rw      1       1
/dev/mirror/RootMirror0s1e      /usr    ufs     rw      2       2
/dev/mirror/RootMirror0s1d      /var    ufs     rw      2       2
```

Раз вы уже знаете, как должен выглядеть необходимый файл */etc/fstab*, скорей создайте его, пока под рукой есть удобный редактор. Обязательно трижды проверьте свою работу – гораздо проще лишний раз проверить правильность таблицы файловой системы, чем потом возиться с загрузчиком или компакт-диском восстановления!

Теперь перезагрузитесь в однопользовательский режим. В этом режиме FreeBSD монтирует только корневой раздел в режиме «только для чтения». Обычно настроить ни один модуль GEOM для смонтированного диска невозможно, но установив параметр `kern.geom.debugflags` в значение `16`, вы сможете сделать это. Этот прием не поможет, если нужно включить корневой раздел в RAID любого типа. Находясь в однопользовательском режиме, скопируйте новый файл *fstab* на место, разрешите режим отладки GEOM и затем создайте зеркало:

```
# cp /etc/fstab /etc/fstab-nomirror
# cp /etc/fstab-mirror /etc/fstab
# sysctl kern.geom.debugflags=16
# gmirror label -v RootMirror0 /dev/ad0
```

После перезагрузки FreeBSD должна найти корневой раздел на новом зеркальном диске. Замечу, что сейчас зеркало состоит из единственного диска и поэтому совершенно неэффективно. Но как только вы вернетесь в многопользовательский режим, просто добавьте резервный диск с помощью соответствующей команды `insert`:

```
# gmirror insert RootMirror0 /dev/ad6
```

Многодисковые зеркала

Традиционно зеркала состоят из двух дисков, однако применение трех дисков в зеркале поможет существенно ускорить резервное копирование базы данных, не затрагивая избыточность. Поместите свою базу данных на трехдисковое зеркало. Если вам понадобилась «холодная» резервная копия, остановите базу данных, удалите диск из зеркала и запустите базу данных. На досуге создайте резервную копию с жесткого диска, извлеченного из зеркала. По окончании процедуры резервного копирования верните третий диск в состав зеркала и позвольте системе выполнить синхронизацию данных.

Поздравляю! Вы получили избыточный загрузочный диск.[1] Если выйдет из строя один из дисков, вам потребуется в BIOS настроить загрузку с другого жесткого диска, но это ничто по сравнению с потерей всех ваших данных.

Разборка зеркала

Если зеркало стало не нужным, выполните команду `gmirror stop`, передав ей имя зеркала. После этого можно выполнить команду `gmirror clear`, чтобы стереть метаданные на дисках, ранее входивших в состав зеркала:

```
# gmirror stop MyMirror
# gmirror clear /dev/da7 /dev/da2
```

После этого вы сможете использовать освободившиеся диски с другими модулями GEOM.

Ежедневный отчет о состоянии

Операционная система FreeBSD может включать результаты проверки состояния зеркалированных дисков в ежедневный отчет periodic(8). Для этого достаточно добавить строку `daily_status_gmirror_enable="YES"` в файл */etc/periodic.conf*.

---

[1] Меня часто спрашивают: «Почему это устройство называется RootMirror0, а не просто RootMirror?» Если в соответствии с правилами добавить имя участка в конец имени устройства, получится RootMirror1, что смущает меня гораздо больше, чем RootMirror0s1. Вы можете выбрать любое имя устройства, какое только заблагорассудится, – главное, чтобы оно не вводило в заблуждение ни вас, ни ваших коллег.

RAID-3

Хотя RAID-3 не особенно популярен в сообществе ИТ-специалистов, он демонстрирует высокую скорость при работе с такими приложениями, как системы автоматизированного проектирования, где обычно приходится работать с одним большим файлом. Чтобы настроить RAID-3, необходимо иметь нечетное число дисков, но не меньшее трех, например 5, 7, 9, 11 и т. д. Управление массивом RAID-3 выполняется с помощью команды graid3(8). Как и в других схемах RAID, в идеале диски желательно физически отделить друг от друга настолько, насколько это возможно.

Сначала с помощью команды `graid3 load` нужно загрузить модуль ядра поддержки RAID-3. Можно также использовать kldload(8) или добавить строку `geom_raid3_load="YES"` в файл */boot/loader.conf*.

Создание RAID-3

Порядок выполняемых действий здесь практически тот же самый: сначала выбираются диски для объединения в массив RAID-3, а затем с помощью graid3(8) эти диски помечаются как входящие в состав массива. В отличие от gmirror(8), при создании массива RAID-3 все данные на дисках будут уничтожены. В это примере для создания устройства RAID-3 с именем MyRaid3 я использовал диски */dev/da0*, */dev/da1* и */dev/da2*. На диске, указанном в командной строке последним, хранятся контрольные суммы.

```
# graid3 label -v -r MyRaid3 /dev/da0 /dev/da1 /dev/da2
Metadata value stored on /dev/da0.
Metadata value stored on /dev/da1.
warning: /dev/da2: only 9105018368 bytes from 9186602496 bytes used.
Metadata value stored on /dev/da2.
Done.
```

Выглядит очень похоже на создание зеркала или массива с чередованием записи, за исключением ключа `-r`. Этот ключ при выполнении команды `label` сообщает grade3(8) о необходимости при чтении использовать диск с контрольными суммами, что повышает скорость случайных операций ввода-вывода, но снижает выполнение последовательного ввода-вывода. Поскольку большинство дисковых операций можно назвать почти случайными, я предпочел использовать его.

Дисковое устройство появится в */dev/raid3*. Создайте файловую систему с помощью команды `newfs -U /dev/raid3/MyRaid3`, смонтируйте ее и приступайте к загрузке данных!

Восстановление RAID-3

RAID-3 предлагает некоторый уровень избыточности и отказоустойчивости, например, при выходе из строя одного из дисков это можно будет видеть в отчете о состоянии:

```
# graid3 status
         Name    Status  Components
raid3/MyRaid3  DEGRADED  da2
                         da0
```

Выглядит не очень хорошо. Что случилось с диском */dev/da1*? Мы поймем это позже[1], а пока перед нами стоит более важная задача – восстановить работу дискового массива. У меня есть несколько дисков «горячего» резерва в массиве SCSI, поэтому давайте включим один из них на место выбывшего, чтобы восстановить избыточность.

Сначала нужно удалить из конфигурации RAID-3 все ссылки на старый диск. При выводе списка устройств мы получим их номера:

```
# graid3 list
...
1. Name: da2
   Mediasize: 9186603008 (8.6G)
...
❶ Number: 2
❷ Type: PARITY
2. Name: da0
   Mediasize: 9105018880 (8.5G)
...
❸ Number: 0
❹ Type: DATA
...
```

Нам нужно получить тип и номер для всех дисков, оставшихся в массиве RAID-3. Диск */dev/da2* имеет номер 2 ❶ в массиве – это диск, хранящий контрольные суммы ❷. Диск */dev/da0* – это диск с номером 0 ❸, он хранит данные ❹. Отсюда следует, что был потерян диск с данными, но его любезно подменил диск с контрольными суммами. Если бы мы потеряли диск с контрольными суммами, наш массив RAID-3 стал бы работать примерно как массив дисков с чередованием записи, пока диск не был бы заменен.

В массиве из трех дисков присутствуют диски с номерами 0 и 2, значит диск, вышедший из строя, имел номер 1. Начнем с того, что удалим номер 1 из устройства RAID-3, освободив место для нового диска. (Не забывайте, что массив RAID-3 может содержать только нечетное число дисков, поэтому нельзя просто взять и добавить еще один диск.) А затем добавим новый диск с номером 1, устройство */dev/da3*:

```
# graid3 remove -n 1 MyRaid3
# graid3 insert -n 1 MyRaid3 da3
graid3: warning: da3: only 9105018368 bytes from 9186602496 bytes used.
```

[1] Расследование показало бы, что кто-то вынул диск из массива, чтобы спровоцировать отказ и потом написать об этом. К счастью, мой начальник не читает мои книги.

Выполнить операцию замены оказалось проще, чем казалось, хотя на диске */dev/da3* остался незадействованным некоторый объем дискового пространства, потому что новый диск оказался немного больше старого. Но помогло ли это? Проверим состояние.

```
# graid3 status
         Name    Status  Components
raid3/MyRaid3  DEGRADED  da2
                         da0
                         da3 (1%)
```

Похоже, помогло. Подождите немного и убедитесь, что синхронизация данных в массиве прошла благополучно.

Разборка RAID-3

Если надобность в устройстве RAID-3 отпала, отмонтируйте раздел и выполните команду `graid3 stop`, передав ей имя устройства зеркала. После этого можно стереть метаданные с дисков:

```
# graid3 stop MyRaid3
# graid3 clear /dev/da3 /dev/da2 /dev/da0
```

После этого диски можно использовать по любому другому назначению, например создать из них массив RAID-10.

RAID-10

Разбивка на чередующиеся области зеркалированных дисков, или RAID-10, пожалуй, самый быстрый способ получить высокопроизводительное устройство хранения больших объемов данных. Сначала диски объединяются в зеркальные пары, а затем производится запись данных на эти пары с чередованием. В массивах этого типа отсутствует выделенное устройство для хранения контрольных сумм, вместо этого избыточность обеспечивается зеркалированием, а запись с чередованием – производительность. Для создания такого массива требуется много дисков, но RAID-10 – единственный реальный способ обеспечить высокую пропускную способность.

Формально RAID-10 – это *вложенный* (*nested*) *RAID*. Фактически это RAID-0, работающий поверх RAID-1. Создание этого массива начинается с создания зеркал (RAID-1) а затем добавляется механизм чередования (RAID-0) записи данных на эти зеркала. Платформа GEOM упрощает возможность создания многоуровневых схем.

Обычно RAID-10 используется в высокопроизводительных системах с высоким уровнем доступности. Этот массив – слишком дорогое удовольствие, если использовать его для хранения веб-страниц клиентов, где вполне достаточно будет обычного зеркала, но, что самое странное, финансовый отдел с легкостью согласится выделить деньги на его организацию, когда под угрозой окажутся их данные. Для зеркальных

дисков очень важно, если это возможно, чтобы они были подключены к разным контроллерам и в разных стойках.

Создание RAID-10

Прежде чем приступить к работе, выпишите на лист бумаги размещение ваших дисков, чтобы потом не запутаться. В этом примере задействованы четыре жестких диска: */dev/da0*, */dev/da2*, */dev/da3* и */dev/da4*. Диски da0 и da2 объединяются в пару mirror1, а диски da3 и da4 – в пару mirror2. Если возможно, диски da0 и da3 не должны быть подключены к одному и тому же контроллеру SCSI, так же как и диски da2 и da4.

Создание зеркал выполняется командой gmirror(8):

```
# gmirror label -v mirror0 /dev/da0 /dev/da2
# gmirror label -v mirror1 /dev/da3 /dev/da4
```

С помощью команды `gmirror status` проверьте состояние всех созданных зеркал – убедитесь в том, что они созданы и выполнили синхронизацию.

Теперь с помощью gstripe(8) создайте том, включающий оба зеркала. Мы применим стандартный размер области чередования, а само устройство назовем raid10:

```
# gstripe label -v -s 131072 raid10 /dev/mirror/mirror0 /dev/mirror/mirror1
Metadata value stored on /dev/mirror/mirror0.
warning: /dev/mirror/mirror1: only 9105017856 bytes from 9186601984 bytes
used.
Metadata value stored on /dev/mirror/mirror1.
Done.
```

Создание закончено

Нет, правда! Это действительно все! В каталоге */dev/stripe* появилось устройство raid10. Выполните команду `newfs -U /dev/stripe/raid10`, чтобы создать файловую систему на устройстве, смонтируйте ее – и дисковый массив готов к работе!

Состояние RAID-10

Чтобы проверить состояние массива RAID-10, нужно проверить состояния каждого из зеркал и состояние самого устройства чередования:

```
# gmirror status
          Name    Status  Components
mirror/mirror0  COMPLETE  da0
                          da2
mirror/mirror1  COMPLETE  da3
                          da4
# gstripe status
         Name  Status  Components
stripe/raid10      UP  mirror/mirror0
                       mirror/mirror1
```

Другие типы вложенных дисковых массивов

Вероятно, вы уже подумали: «Я же могу создать RAID-13, настроив RAID-3 поверх зеркал!» Да, платформа GEOM позволит сделать это. Вы можете придумать свои уровни RAID. Можно, например, создать RAID-33 из трех устройств RAID-3, RAID-130, добавив функцию чередования к двум RAID-3, созданным поверх зеркал, RAID-303, и т. д. Однако все эти варианты требуют слишком большого количества дисков, не давая преимуществ, а многие из них в действительности представляют бо́льшую угрозу для данных, чем простой RAID-10. Вложенный RAID может нести в себе массу ошибочных состояний, и вполне возможно, что кто-то уже проверил идеи, которые пришли в голову вам. Поэтому прежде чем внедрять тот или иной тип вложенного RAID в рабочую систему, почитайте соответствующую литературу!

Очень важна работоспособность зеркал. Если одно из зеркал деградирует, нужно заменить вышедший из строя диск и пересобрать зеркало. Устройство чередования не требует обслуживания, как и обычный раздел, созданный командой gstrip(8). Полная потеря одного из зеркал приведет к потере всех данных. Не забывайте, как важно расположить зеркальные диски как можно дальше друг от друга.

Разборка RAID–10

Чтобы прекратить работу RAID-10, нужно сначала разобрать устройство чередования, а затем разобрать зеркала, так же как для любых других дисковых массивов RAID-0 и RAID-1.

Создание журналируемых файловых систем с помощью gjournal(8)

Народная мудрость утверждает, что лучший способ обеспечить целостность файловой системы заключается в использовании *журналируемой* файловой системы. Журналируемая файловая система предварительно записывает все изменения на диске в небольшой файл, или *журнал*, и лишь потом выполняет фактическую операцию записи в файловой системе. Это означает, что все изменения быстро и безопасно кэшируются на диске, чтобы уменьшить риск потери данных в случае неожиданно пропавшего напряжения. Восстановившись после такой неприятности, система прочитает содержимое журнала и не спеша запишет изменения в файловую систему. На этапе загрузки система проверяет файл журнала на наличие изменений, еще не записанных на диск, и вносит эти изменения, гарантируя обновление данных.

В течение многих лет операционная система FreeBSD опиралась на механизм Soft Updates, позволяющий реализовать многие функции журналирования без полного внедрения журналируемой файловой системы. Однако с ростом объемов жестких дисков все больше и больше стала ощущаться недостаточность функциональности Soft Updates. UFS и UFS2 известны как стабильные и надежные файловые системы. Реализация функций журналирования требует такого глубокого изменения файловой системы UFS2, что это поставило бы под угрозу с трудом заработанную добрую репутацию FreeBSD, даже если бы эти изменения вносил программист, обладающий редкой комбинацией солнечного оптимизма и дремучего фанатизма.

Эта задача была трудноосуществима, пока не появились наращиваемые модули GEOM. Если добавление функции журналирования в UFS2 – сложная задача, почему бы не создать между диском и UFS2 промежуточный уровень, реализующий журналирование? И здесь на сцену выходит gjournal(8). Для gjournal(8) требует лишь нескольких точек входа в реализацию файловой системы, чтобы узнать, куда записывать изменения, затем gjournal выполняет фактическую запись. Это устраняет необходимость запуска fsck(8) во время загрузки после краха системы.

gjournal(8) – это новая функциональная возможность, появившаяся в версии FreeBSD 7, и в действительности представляет собой довольно радикальный подход к проблеме журналирования. Многие с успехом используют gjournal(8), но в настоящий момент эта особенность попрежнему должна рассматриваться как экспериментальная. Я использую ее в многотерабайтной файловой системе, которая содержит миллионы файлов, созданных программой Netflow. Однако в сравнении со зрелой файловой системой UFS, gjournal(8) – это неопытный малыш. Несмотря на то что ошибки исправляются по мере их обнаружения, на вашу долю может выпасть счастье наткнуться на новую ошибку, которая поставит ваши данные под угрозу. Я вас предупредил, а вы думайте.

По умолчанию файл журнала занимает один гигабайт дискового пространства, что малопригодно для небольших файловых систем. Для файловых систем с объемом менее 20 Гбайт я обычно использую механизм Soft Updates, а журналирование применяю в более крупных файловых системах. Журналирование и Soft Updates – это взаимоисключающие механизмы: вы можете использовать оба механизма на одном и том же сервере, но никак не в одной и той же файловой системе.

Внедрение gjournal(8) в файловую систему приводит к уничтожению всех данных в этой файловой системе. (При определенных обстоятельствах этого можно избежать, но в большинстве случаев лучше не полагаться на это.) Если вам требуется включить функцию журналирования в существующую файловую систему, сначала создайте резервную копию и подготовьтесь к восстановлению из нее. Учитывая объемы современных дисков, нет ничего сложного в том, чтобы скопировать порядка 400 Мбайт или что-то около того из раздела */usr* на другой раздел диска,

настроить журналирование раздела */usr* в однопользовательском режиме и восстановить содержимое */usr*. Изначально на других разделах хранится еще меньше данных, следовательно, их проще будет преобразовать. Думаю, как только gjournal(8) утратит свой экспериментальный статус, поддержка этого механизма будет интегрирована в инсталлятор.

Конфигурирование gjournal(8)

Для работы gjournal(8) требуется файл журнала. Журнал может быть отдельным дисковым устройством или располагаться в разделе, содержащем журналируемую файловую систему. Если и журнал и файловая система располагаются на одном диске, с точки зрения производительности нет смысла делить его на два раздела. Однако если разместить журнал на отдельном диске, вы можете ощутить увеличение производительности. (Я не имею в виду два старомодных диска ATA, подключенных к одному контроллеру!) Для наших примеров будем исходить из предположения, что файл журнала располагается на том же разделе диска, что и файловая система.

Загрузите модуль ядра командой `gjournal load` или загрузите модуль *geom_journal.ko* вручную. Кроме того, модуль ядра можно загрузить во время старта системы, добавив соответствующую инструкцию в файл */boot/loader.conf*.

Теперь нужно пометить раздел как журналируемый. Ниже выполняется преобразование */dev/da0s1d*, единственного раздела на этом диске, в журналируемый раздел:

```
# gjournal label /dev/da0s1d
gjournal: File system exists on /dev/da0s1d and this operation would destroy
it. Use -f if you really want to do it.

(Перевод: gjournal: На /dev/da0s1d файловая система уже существует
и эта операция может разрушить ее. Если вы действительно хотите сделать это,
используйте ключ -f.)
```

Скажем так: слово *преобразование* – не совсем то слово, которое мне хотелось бы употребить. В действительности, мы уничтожаем существующую файловую систему и помещаем файл журнала в конец раздела.

```
# gjournal label -f /dev/da0s1d
```

gjournal подготовит устройство журналирования для этого раздела. Вместо того чтобы создать отдельный каталог для устройств журналирования в */dev*, эта команда просто добавляет расширение *.journal* к имени файла устройства раздела. Все остальные операции с файловой системой должны выполняться над этим устройством журналирования.

Журналирование и newfs(8)

Теперь создадим файловую систему на устройстве журналирования, но вместо ключа `-U`, разрешающего использование механизма Soft Updates, используем ключ `-J`, разрешающий журналирование.

```
# newfs -J /dev/da0s1d.journal
/dev/da0s1d.journal: 7651.7MB (15670656 sectors) block size 16384, fragment
size 2048
...
```

Обратите внимание на объем устройства с журналированием. Диск имеет объем 8,5 Гбайт, а устройство с журналированием – 7 651 Мбайт. Куда пропал целый гигабайт? Его занял файл журнала. Это вполне предсказуемо и нормально.

Монтирование журналируемых файловых систем

Большинство разделов UFS2 монтируются в синхронном режиме, однако журналируемые файловые системы должны монтироваться в асинхронном режиме. Как говорилось в главе 8, в случае монтирования в асинхронном режиме, когда данные сбрасываются на диск, система не ожидает, пока они фактически будут записаны на диск. Функция журналирования сама выполняет проверку и контролирует целостность. Укажите параметр `async` в поле параметров монтирования в файле */etc/fstab*:

```
/dev/da0s1d.journal     /usr    ufs     rw,async        2       2
```

Отдельное устройство журналирования

Основное преимущество применения отдельного устройства журналирования состоит в том, что оно может – *может* – помочь сохранить существующую файловую систему. gjournal(8) сохраняет свои метаданные в последнем секторе раздела. Если последний сектор уже используется файловой системой, то ее нельзя преобразовать и придется пересоздать с помощью newfs. Имя устройства журналирования передается команде `label` в виде второго аргумента. Ниже устройство */dev/da0s1d* помечается как журналируемая файловая система, журнал для которой размещается на устройстве */dev/da1s1e*. Если последний сектор раздела предназначенной для журналирования файловой системы уже используется, gjournal(8) выдаст предупреждение.

```
# gjournal label /dev/da0s1d da1s1e
```

Далее, чтобы создать журналируемую файловую систему на устройстве */dev/da0s1d*, вместо newfs используется команда tunefs(8), которая активизирует механизм журналирования и запрещает применение механизма Soft Updates для существующей файловой системы:

```
# tunefs -J enable -n disable /dev/ad0s1d
```

Теперь можно смонтировать новую журналируемую файловую систему на устройстве *.journal* с параметром `async`.

Отключение функции журналирования

Как и для модулей GEOM, обслуживающих дисковые массивы RAID, чтобы отключить функцию журналирования, нужно выполнить ко-

манду `gjournal stop` и `gjournal clear` для файла устройства, чтобы стереть метаданные gjournal на разделе:

```
# gjournal stop da0s1d.journal
# gjournal clear /dev/da0s1d
```

Затем можно выгрузить модуль ядра, чтобы завершить процедуру отключения функции журналирования в системе.

Шифрование файловых систем

Операционная система FreeBSD поддерживает два разных метода шифрования дисков – GBDE и GELI. Оба инструмента работают совершенно по-разному, поддерживают отличающиеся криптографические алгоритмы и предназначены для защиты от различных видов угроз. Разговоры о шифровании дисков ведутся постоянно, но очень редко можно услышать, от чего же все-таки защищает функция шифрования диска.

Механизм GBDE, или Geom-Based Disk Encryption (шифрование диска на базе GEOM), обладает специализированными возможностями для применения в системах с высокой степенью секретности, где защита пользователей так же важна, как и скрытие данных. В дополнение к ключу шифрования, предоставляемому пользователем, механизм GBDE использует ключи, сохраняемые в определенных секторах на жестком диске. Если какой-либо ключ окажется недоступен, расшифровать раздел не удастся. Почему это так важно? Если вычислительный центр с высокими требованиями к секретности (например, в посольстве) подвергнется нападению, оператору потребуется совсем немного времени, чтобы уничтожить ключи на жестком диске и тем самым исключить возможность восстановления данных. Если плохие парни, приставив пистолет к моей голове, потребуют: «Введи пароль – или...», я предпочел бы, чтобы файловая система ответила: «Пароль принят, но ключи уничтожены». Мне не нужны универсальные сообщения вроде: «Диск не может быть расшифрован». Если в первом случае я еще могу представлять ценность как заложник, то во второй меня либо убьют, либо придумают что-нибудь похуже.[1]

Механизм шифрования GELI более гибок, но он не сможет защитить меня от телесных повреждений, как GBDE. Если кто-то захватит мой ноутбук с конфиденциальными документами или пользователь, которому я не доверяю, пожелает исследовать пространство свопинга на наличие секретной информации, механизма GELI будет вполне достаточно. Но GELI даже не попытается защитить мою персону – только мои данные. Поскольку я не собираюсь устраиваться на работу, которая не предполагает высокую степень защищенности от огнестрельно-

[1] Просто для справки: если у вас острый предмет и явные намерения, вы можете получить мои пароли.

го оружия (при том, что я живу в Детройте), этот механизм подходит для меня как нельзя лучше. Кроме того, GELI использует криптографический драйвер устройства FreeBSD, а это означает, что при наличии на сервере аппаратных средств шифрования GELI задействует их.

Следует заметить, что в большинстве случаев применение механизма шифрования файловых систем не только излишне, но и увеличивает риск потери данных. Из-за неправильных настроек шифрования или из-за утраты ключей данных было потеряно гораздо больше, чем из-за краж. Когда я слышу: «Я зашифровал весь свой жесткий диск!», – мне легко представить будущее, когда тот же самый человек скажет: «Я потерял все, что было на жестком диске!» Я оказываюсь прав более чем часто. Задумайтесь, так ли вам *необходимо* шифрование диска.

В этом примере мы будем использовать механизм CELI для шифрования раздела на диске */dev/da0*, а ключи шифрования будут храниться на устройстве USB, монтируемом в каталог */media*. Более разумным может оказаться решение использовать в качестве шифруемого раздела файловую систему внутри файла (глава 8). Шифрование всего жесткого диска требуется очень немногим и при весьма специфических обстоятельствах, потому что это может увеличить подозрения. У меня есть опыт объяснений со службой безопасности аэропорта, сотрудники которой пытались выяснить «почему мой ноутбук выглядит так странно». На их взгляд, от приглашения к вводу «Вставьте ключ и введите пароль», появляющегося перед загрузкой, до решения «этот человек подозрительно выглядит, и его нужно полностью обыскать» – всего один шаг. Если вам действительно необходимо зашифровать некоторые документы, скорее всего они занимают всего несколько мегабайт, и тогда лучше использовать файловую систему в файле.

Конфигурирование ядра

Для работы механизма GELI необходим модуль ядра *geom_eli.ko*, которому, в свою очередь, требуется модуль *crypto.ko*. Загрузить эти модули можно на этапе загрузки системы, с помощью файла */boot/loader.conf* или добавить соответствующие устройства в конфигурацию ядра:

```
options GEOM_ELI
device crypto
```

Создание и использование ключа шифрования

Для работы с зашифрованным устройством механизм GELI позволяет использовать файл ключа и/или секретную фразу в качестве ключа. Мы будем использовать и то и другое. Чтобы создать файл ключа, с помощью утилиты dd(1) извлеките достаточно большой объем данных из устройства */dev/random* и запишите их в файл. Не забывайте, каталог */media* – это точка монтирования устройства USB. Если вы действительно хотите защитить свои данные, создайте ключ непосредственно на устройстве USB – не оставляйте его в своей файловой системе, откуда пред-

полагаемый злоумышленник смог бы восстановить его. (Даже в случае удаления файла на диске по-прежнему остаются данные, которые могут быть восстановлены квалифицированным злоумышленником.)

```
# dd if=/dev/random of=/media/da0.key bs=64 count=1
1+0 records in
1+0 records out
64 bytes transferred in 0.000149 secs (429497 bytes/sec)
```

64 байта данных составляют 512-битовый ключ. При желании вы можете увеличить размер ключа ценой увеличения нагрузки на процессор при обращении к зашифрованной файловой системе. Не забывайте, что секретная фраза также увеличивает сложность ключа.

Связать секретную фразу с ключом позволяет команда *geli_init*. Ключ `-s` определяет желаемый размер сектора на шифруемой файловой системе; обычно 4 096 байтов, или 4 Кбайт вполне достаточно для данного применения. Ключ `-K` определяет имя файла ключа. Кроме того, нужно указать устройство, которое будет шифроваться.

```
# geli init -s 4096 -K /media/da0.key /dev/da0
Enter new passphrase:
(Перевод: Введите секретную фразу)
Reenter new passphrase:
(Перевод: Повторите ввод секретной фразы)
```

Секретная фраза очень напоминает пароль, но, в отличие от последнего, может содержать пробелы и иметь любую длину. Если вы хотите обеспечить действенную защиту своих данных, рекомендую использовать фразу, которая состоит из нескольких слов, содержит не только алфавитно-цифровые символы и не является устойчивым словосочетанием на вашем родном языке.

Теперь, когда у нас есть ключ, подключим его к устройству, которое будет шифроваться.

```
# geli attach -k /media/da0.key /dev/da0
Enter passphrase:
```

Теперь geli(8) знает, что */dev/da0* – это шифруемый диск и что файл */media/da0.key* – это файл ключа шифрования. После ввода секретной фразы вы сможете получить доступ к расшифрованному содержимому зашифрованного диска через устройство */dev/da0.eli*. Безусловно, чтобы можно было помещать на этот диск какие-либо данные, вам понадобится файловая система.

Файловые системы на шифруемых устройствах

Прежде чем создать файловую систему на шифруемом устройстве, следует очистить диск от старых данных. На самом деле, newfs(8) не удаляет с диска никакие данные; эта утилита просто добавляет суперблок, который указывает местоположение индексных дескрипторов. Если этот диск ранее использовался, злоумышленник сможет просмотреть

оставшиеся на нем части файлов. Хуже того, он сможет просматривать остатки зашифрованных данных, помещенных туда механизмом GELI. Прежде чем поместить на диск файловую систему, лучше всего покрыть диск «дымовой завесой» из случайных данных, чтобы существенно осложнить злоумышленнику возможность идентифицировать блоки с данными. Снова воспользуемся утилитой dd(1):

```
# dd if=/dev/random of=/dev/da0.eli bs=1m
8684+0 records in
8683+1 records out
9105014784 bytes transferred in 1575.226434 secs (5780131 bytes/sec)
```

Как видите, этой древней системе потребовалось больше 25 минут, чтобы покрыть весь 8,5-гигабайтный диск высококачественными случайными данными. Требуемое время зависит от быстродействия системы и объема доступных случайных данных. Аппаратное устройство шифрования может существенно увеличить объем доступных случайных данных.

Теперь, когда диск забит мусором под завязку, можно установить на него файловую систему и подключить ее к системе. Не используйте механизм Soft Updates для работы с шифруемыми дисками. В любом случае шифрование диска снизит производительность системы.

```
# newfs /dev/da0.eli
# mount /dev/da0.eli /mnt
```

Теперь ваш шифруемый диск доступен в каталоге */mnt* и можно сохранять на нем конфиденциальные файлы.

Деактивизация шифруемых дисков

По окончании работы с шифруемым разделом нужно остановить механизм GELI и вернуть раздел в недоступное, то есть зашифрованное состояние. Для начала следует отмонтировать раздел, а затем командой `geli detach` отсоединить шифруемый диск от ключа.

В то время как другие модули GEOM обычно помещают метаданные в последний сектор раздела, механизм GELI этого не делает. Поэтому вам не придется стирать метаданные из раздела GELI перед использованием раздела по другому назначению. Вам придется запоминать, какие разделы являются шифруемыми, а какие просто пусты, особенно если в вашей системе много пустых дисков. Диск, зашифрованный целиком, не имеет даже метки диска, поэтому очень легко можно перепутать зашифрованный диск с пустым!

Шифрование пространства свопинга с помощью geli(8)

Если вы позволяете входить в систему пользователям, не вызывающим доверия, возможно, не лишено смысла зашифровать пространство свопинга. Я нахожу это полезным на серверах, куда может заходить

множество пользователей, чтобы работать со своими приложениями. Вместо того чтобы создавать ключ шифрования, который требует защиты секретной фразой, FreeBSD будет во время загрузки генерировать случайный ключ без секретной фразы и применять его для шифрования пространства свопинга. Это обеспечит шифрование данных в разделе свопинга, чтобы ни один злоумышленник не смог извлечь секретные даны из пространства свопинга. Хотя оба механизма, и GBDE и GELI, поддерживают шифрование пространства свопинга таким способом, мы по-прежнему будем использовать GELI.

В дополнение к загрузке модуля ядра geom_eli во время запуска системы вы должны немного изменить файл */etc/fstab*. Добавьте расширение *.eli* к имени устройства пространства свопинга и в результате FreeBSD будет автоматически шифровать его средствами механизма GELI. Например, запись в файле */etc/fstab*, описывающая раздел свопинга на моей машине, изменится с:

```
/dev/ad0s1b     none    swap    sw      0       0
```

на:

```
/dev/ad0s1b❶.eli  none    swap    sw      0       0
```

Небольшое изменение ❶ – это все, что необходимо для обеспечения шифрования пространства свопинга. Лично я считаю более полезным наращивать объем памяти на сервере с секретными данными, чтобы вообще отказаться от раздела свопинга, но для вас это может оказаться неприемлемо.

Экспорт дисковых устройств по сети

Вам требуется устройство для записи компакт-дисков на машине A, но оно установлено на машине B? Или, может быть, вам нужен доступ к разделу жесткого диска, или даже к файловой системе на диске? Нет проблем! Просто экспортируйте устройство по сети с помощью модуля geom_gate и смонтируйте его на другой машине.

Зачем возиться с экспортом устройства, когда достаточно просто войти на другую машину? Возможно, вам потребуется приложение, которое подразумевает использование диска, а не файла. Возможно, вам потребуется снять образ с отмонтированного раздела, а на машине с этим разделом недостаточно свободного места, чтобы сохранить этот образ. Возможно, вам потребуется использовть gmirror(8) по сети, чтобы в случае выхода из строя одной машины у вас сохранилась свежая копия диска. Или, может быть, просто потому, что это круто и заставит других системных администраторов смотреть на вас снизу вверх. geom_gate – это приложение типа клиент-сервер: система с экспортируемым устройством является сервером, а система, которая монтирует удалаенное устройство, – клиентом. Обеим системам, и клиенту и серверу, требуется загруженный модуль ядра *geom_gate.ko*.

Безопасность geom_gate

Соединение geom_gate между клиентом и сервером не шифруется, поэтому любой, кто сможет перехватить трафик, сможет увидеть транзакции. Замечу лишь, что ему будет очень сложно идентифицировать содержимое этого трафика! Если вы используете geom_gate в сетях общего доступа, желательно настроить VPN или хотя бы создать туннель SSH. Для своей работы geom_gate по умолчанию использует порт TCP с номером 3080. (Если вам требуется нечто действительно интересное, попробуйте расположить geom_gate поверх GELI!)

Управление доступом на стороне сервера производится с помощью файла экспорта, подобно NFS. Сервер экспортирует только устройства, указанные в файле *exports*, и только указанным клиентам.

Настройка сервера geom_gate

С помощью geom_gate можно экспортировать только CD-ROM, образы дисков и разделы. Вы не можете экспортировать накопители на магнитной ленте и такие устройства, как */dev/random*, */dev/io* или терминальные устройства. Если устройство нельзя смонтировать утилитой mount(8), оно недоступно для экспорта. Однако устройство не может быть смонтировано, пока оно экспортируется; это должно быть устройство, доступное для монтирования, но не смонтированное.

Чтобы выполнить экспорт устройства, укажите информацию о нем в файле экспорта модуля geom_gate, */etc/gg.exports*. Все записи в этом файле имеют следующий формат:

```
хост        права_доступа    устройство
```

Хост можно указать в виде IP-адреса (например, 192.168.1.2), в виде адреса сети (например, 192.168.1.0/24) или в виде имени хоста (например, *pesty.blackhelicopters.org*).

Для каждого экспортированного устройства нужно указать права доступа: «чтение-запись» (`RW`), «только для чтения» (`RO`) или «только для записи» (`WO`).

Наконец, нужно указать файл устройства, которое требуется экспортировать. Например, ниже мы разрешаем доступ только для чтения к приводу CD-ROM на сервере с любого хоста из моей частной сети, а также доступ для чтения и записи к дисковому устройству */dev/da0* с IP-адреса моего ноутбука:

```
192.168.1.0/24  RO      /dev/acd0
192.168.1.202   RW      /dev/da0
```

Теперь можно запустить ggated(8). К моменту написания этих строк команда ggated(8) еще не была интегрирована в систему начальной загрузки rc.d. К счастью, вы уже прочитали главу 12 и знаете, как напи-

сать сценарий rc.d, который может быть интегрирован во FreeBSD. Команда ggated(8) не требует аргументов:

```
# ggated
```

Теперь модуль geom_gate будет принимать запросы клиентов, прослушивая порт TCP 3080.

Настройка клиента geom_gate

Теперь, когда у нас настроен экспорт нескольких устройств, можно запустить клиент. Для настройки клиента не требуется конфигурационный файл, достаточно выполнить команду ggatec(8). При создании локального файла устройства, который соответствует устройству, экспортированному удаленным сервером, используется следующий синтаксис:

```
# ggatec create -o права_доступа имя_сервера имя_устройства
```

Предположим, что нам нужен доступ только для чтения к приводу компакт-дисков сервера. Имя сервера – *sardines.blackhelicopters.org*, устройство CD-ROM – */dev/acd0*. Это устройство присутствует в файле */etc/gg.exports* на стороне сервера.

```
# ggatec create -o ro sardines /dev/acd0
ggate0
```

Команда ggatec отвечает созданием локального файла устройства (*/dev/ggate0*), соответствующего удаленному файлу устройства (*/dev/acd0*). Если это действительно экспортированный привод компакт-дисков, я должен иметь возможность увидеть компакт-диск в этом приводе.

```
# mount -t cd9660 /dev/ggate0 /cdrom/
```

Простая команда `ls /cdrom` показывает, что компакт-диск на стороне сервера теперь смонтирован клиентом. Круто, да?

Следующий вопрос: как далеко можно зайти? Давайте экспортируем фактический жесткий диск и проверим, какие ошибки можно допустить при обращении к нему.

```
# ggatec create -o ro sardines /dev/da0
ggate1
```

Первое, что приходит в голову, – попробовать работать с этим устройством как с обычным жестким диском. Можно ли использовать такие утилиты, как fdisk(8), disklabel(8) и newfs(8), с этим устройством? Ответ: нет, нельзя. Утилита fdisk(8) не сможет выполнить запись в первый сектор диска. Мы не сможем использовать disklabel(8), потому что эта утилита работает с участками дисков, а не с целыми дисками, а мы экспортировали не */dev/da0s1*, а только */dev/da0*. Если экспортировать */dev/da0s1*, мы не сможем получить доступ к разделам, а только к целому участку. Следует быть очень внимательным к тому, что экс-

портируется, но даже в этом случае вы не сможете выполнять низкоуровневые операции. Однако если серверный диск поделен на разделы, которые содержат файловые системы, вы сможете монтировать их и получать доступ к данным.

```
# ggatec create -o ro sardines /dev/da0s1a
ggate3
# mount /dev/ggate3 /mnt
```

Теперь удаленный диск */dev/da0s1a* смонтирован в локальный каталог */mnt*.

Идентификация устройств geom_gate

Некоторое время поработав с geom_gate, легко можно забыть, какое устройство с какой машины было смонтировано. Имя устройства */dev/ggate* мало что напомнит о фактическом устройстве. У утилиты ggatec(8) есть команда `list`, которая выводит сведения об устройствах geom_gate, смонтированных локально. Чтобы получить более подробную информацию об удаленных устройствах, используйте ключ `-v`:

```
# ggatec list -v
       NAME: ggate0
       info: sardines:3080 /dev/acd0
     access: read-only
    timeout: 0
queue_count: 0
...
```

Хотя ggatec(1) выводит достаточно много сведений о каждом устройстве, самые важные для нас – имя удаленного сервера, имя удаленного устройства и тип доступа к этому устройству.

Остановка geom_gate

По окончании работы с экспортируемыми устройствами их нужно удалить с клиента. Сначала нужно размонтировать локальное устройство на стороне клиента, а потом вызвать функцию `destroy` команды ggatec(8). Предположим, что у нас имеется устройство */dev/ggate0*, смонтированное в каталоге */cdrom*:

```
# umount /cdrom
# ggatec destroy -u 0
```

Ключу `-u` передается номер устройства ggate, которое требуется удалить. Мы использовали `-u 0`, потому что останавливаем устройство ggate0.

На стороне сервера достаточно просто остановить ggated(8) командой pkill(1):

```
# pkill ggated
```

Все! Ваши сетевые устройства больше не экспортируются.

Ой!.. Восстановление geom_gate

Предположим, что вы оказались были неосторожны с geom_gate, когда остановили его. Например, я размонтировал */cdrom* перед уничтожением устройства */dev/ggate0*, но у меня осталось устройство */dev/ggate3*, смонтированное в каталоге */mnt*. Теперь у меня нет доступа к этому разделу, я не могу размонтировать его, а если я попытаюсь скопировать на него какие-либо данные, команда зависнет. Что делать? У утилиты ggatec(8) есть команда `rescue`, предназначенная именно для таких ситуаций. Перезапустите ggated(8) на стороне сервера, а потом запустите новый процесс ggatec(8) на стороне клиента. Синтаксис команды `rescue` очень похож на синтаксис команды `create`, за исключением того, что ей можно передать номер файла устройства ggate. Ниже мы реактивизируем устройство */dev/ggate3*, смонтированное в каталоге */mnt*:

```
# ggatedc rescue -o rw -u 3 sardines /dev/da0s1a
```

Все. Никто не узнает, что вы попали впросак!

Зеркалирование дисков по сети

Два сервера могут использовать единственный образ диска. Когда данные записываются на один диск, они автоматически копируются на другой. Эту особенность можно использовать для достижения определенного уровня высокой доступности при использовании аппаратных средств массового потребления. Используйте утилиту ggated(8), чтобы экспортировать диск, и gmirror(8) для его дублирования.

Не забывайте, что скорость передачи данных по сети намного ниже скорости обмена данными с диском, поэтому зеркалирование дисков по сети – не самое лучшее решение при интенсивных операциях ввода-вывода. Например, я ожидаю, что к тому моменту, когда вы будете читать эти строки, широкое распространение получат диски 3Gbps SATA-II. Это три гигабита в секунду, или в три раза больше скорости передачи в гигабитной сети! Если скорость обмена с диском действительно будет составлять 3 Гбит/сек, то, учитывая тот факт, что не все гигабитные сетевые устройства фактически работают на скорости в один гигабит, резервное копирование по сети сильно разочарует вас. Однако в случае приложений с небольшим объемом ввода-вывода зеркалирование по сети дисков с важными данными может оказаться полезным.

В приведенном ниже примере выполняется зеркалирование по сети файловой системы в файле размером 100 Мбайт. Этот объем достаточно велик, чтобы его хватило, скажем, для небольшой базы данных, но слишком мал, чтобы работать в моей скромной сети.

Настройка сервера резервной копии

Создайте файл размером 100 Мбайт на одном сервере и скопируйте его на другой сервер. Таким образом, у нас изначально будет один и тот же

файл с данными по обеим сторонам соединения, что поможет уменьшить время синхронизации. Проще всего создать файл с помощью утилиты truncate(1):[1]

```
# truncate -s100m /var/db/dbmirrorbackup.img
```

Ключ `-s` команды truncate(1) определяет размер создаваемого файла. Допустимы следующие сокращения: `k` – килобайты, `m` – мегабайты и `G` – гигабайты. (Есть еще `t` – для обозначения терабайтов, но у нас нет дисков такой емкости. Пока.)

Теперь сконфигурируем */etc/gg.exports*, чтобы позволить первичному серверу получить доступ к этому диску. Первичный сервер (в примере его IP-адрес 192.168.1.2) требует наличия привилегий доступа к диску «чтение-запись»:

```
192.168.1.2     RW      /var/db/dbmirrorbackup.img
```

Теперь запустим ggated(8) – и сервер резервной копии готов к работе.

Настройка первичного сервера

Настроить первичный сервер сложнее, но ненамного. Сначала нужно подключить устройство ggate к удаленному файлу:

```
# ggatec create backupserver /var/db/dbmirror.img
ggate0
```

Теперь удаленный файл с файловой системой доступен локально как устройство */dev/ggate0*. Затем загрузим gmirror(8) и пометим устройство ggate как зеркало. Здесь наше зеркальное устройство называется remotemirror0:

```
# gmirror load
# gmirror label remotemirror0 /dev/ggate0
```

Теперь у нас есть зеркальное устройство. Нам нужно создать локальный файл с файловой системой, точно так же, как был создан резервный файл с файловой системой:

```
# truncate -s100m /var/db/dbmirrorprimary.img
```

Подключим локальный файл к устройству памяти, чтобы получить файл устройства, и затем вставим это устройство в зеркало:

```
# mdconfig -a -t vnode -f /var/db/dbmirrorprimary.img
md0
# gmirror insert remotemirror0 /dev/md0
```

Вы увидите, насколько быстро произойдет синхронизация. Это потому, что там еще нет данных!

---

[1] Если проще всего создать файл с помощью truncate(1), тогда почему в главе 8 мы использовали dd(1)? Это было сделано для вашей же пользы, поскольку вам необходимо знать, как работать с dd(1).

```
# gmirror status
                  Name    Status  Components
mirror/remotemirror0  DEGRADED  ggate0
                                  md0 (33%)
```

По окончании синхронизации зеркала, что должно занять не больше пары минут для файловой системы такого размера, используйте newfs(8) и смонтируйте ее:

```
# newfs -U /dev/mirror/remotemirror0
# mount /dev/mirror/remotemirror0 /database
```

Теперь у нас имеется распределенное зеркало, смонтированное в каталоге `/database`.

В зависимости от степени загруженности системы и пропускной способности сети может оказаться полезным монтировать диск с параметром `noatime`, чтобы минимизировать число операций записи.

Аварийные ситуации с зеркалом и восстановление

Зеркальный диск нужен для восстановления данных после выхода из строя диска, а сетевой зеркальный диск – после выхода из строя сервера. В комбинации с зеркальными дисками вы можете использовать программу heartbeat (*/usr/ports/sysutils/heartbeat*), что позволит реализовать недорогое и высоконадежное решение, когда одна система автоматически принимает на себя функции другой системы в случае ее выхода из строя. При использовании heartbeat резервная система следит за основной системой и приступает к работе, как только основная система исчезает из сети.

В такой схеме основным сервером может быть ваша клиентская машина, которая выполняет все необходимые операции большую часть времени. Но в случае появления неполадок резервная машина возьмет на себя роль основной машины. Резервная машина может активизировать свою копию зеркала с помощью команд mdconfig(8) и mount(8):

```
# mdconfig -a -t vnode -f dbmirrorbackup.img
md0
# fsck -B /dev/md0
# mount /dev/md0 /database
```

После этого можно запустить программу сервера базы данных и использовать данные зеркального раздела.

Когда основная система вернется в строй, она снова подключит резервный файл как устройство ggate, пометит его как зеркало, вставит локальный раздел в зеркало и даст зеркалу возможность синхронизироваться из резервной копии. По окончании синхронизации heartbeat сможет переключить работу сервиса обратно на основную машину. Создать сценарий автоматической обработки аварийных ситуаций не так сложно, но для этого нужно учитывать множество деталей реализации и окружения, поэтому его создание оставляю вам.

Высоконадежная система, как в данном примере, станет еще надежнее, если применять базу данных, использующую механизм транзакций, такую как PostgreSQL, в противоположность MySQL.[1] Будет совсем неплохо, если вам удастся свести к минимуму объем «оперативных» данных на зеркальном разделе и уделить основное внимание периодическому архивированию данных на другом диске удаленного сервера. Высокая надежность требует внимательного отношения к проектированию приложения, и к этому следует относиться серьезно.

На этом мы закончили наш тур по наиболее популярным модулям платформы GEOM. В операционной системе FreeBSD вы найдете множество других интересных модулей, но обладая знаниями, полученными в этой главе, вы безболезненно сможете их использовать. А теперь перейдем к теме, близкой сердцу любого системного администратора: производительность.

[1] На самом деле, MySQL тоже поддерживает механизм транзакций начиная с версии 3.23.15, которая вышла еще в 2000 году. Наверное, с тех пор это событие так и осталось незамеченным многими... – *Прим. науч. ред.*

19

Производительность системы и ее мониторинг

«Медленно работает!» – не самое страшное из того, что когда-либо слышал системный администратор, тем не менее, он боится это услышать. Пользователь не знает, почему система тормозит, скорее всего, он даже не задумывается, в чем проблема. Он просто *чувствует*, что система работает медленно. Хотя обычно ничего не тестирует, не повторяет какие-то шаги, не получает ошибок. Жалоба на медлительность чревата многочасовым исследованием, когда вы, углубившись в систему, пытаетесь найти проблему, которая то ли есть, то ли нет. Есть слова и пострашнее, особенно после того, как вы потеряли на этом кучу времени: «Все равно медленно».

Неопытный системный администратор считает: чтобы ускорить медленные системы, надо просто приобрести быстродействующие аппаратные средства. Это лишь обмен «проблемы быстродействия» на дорогостоящие запчасти и еще более дорогостоящее время. Обновление аппаратных средств лишь замаскирует истинные неполадки вместо грамотного использования имеющейся аппаратуры, а порой и вовсе не решит проблему.

В состав FreeBSD входит много инструментов, предназначенных для исследования производительности системы. Они предоставляют информацию, необходимую для выявления истинных причин медленной работы. Как только вы поймете, где кроется проблема, решение отыщется гораздо легче. Возможно, действительно помогут более быстрые аппаратные средства или удастся перераспределить нагрузку в системе и улучшить общую производительность более дешевым способом. В любом случае первый шаг состоит в том, чтобы понять истинную причину неполадок.

Ресурсы компьютера

Проблемы быстродействия обычно появляются при запуске большего объема задач, чем может обслужить компьютер. Такое утверждение кажется очевидным, но задумайтесь о нем на минуту. Что оно означает?

В компьютере есть четыре основных ресурса: ввод-вывод, сетевая пропускная способность, память и процессор. Если возможности одного из ресурсов исчерпаны, то остальные ресурсы не удастся задействовать с максимальной эффективностью. Например, процессор может просто-напросто ожидать данных от диска или прибытия сетевого пакета. В таком случае более быстрый процессор не увеличит производительность системы, и вы будете разочарованы. Покупка нового сервера может ликвидировать проблему, но лишь за счет некоторого расширения узких мест. Возможно, в новой системе будет больше оперативной памяти, более быстрые диски, лучшая сетевая карта и более быстрый процессор. Но тем самым вы лишь задержите проявление проблемы, пока производительность не достигнет некоторого нового предела. Определив узкое место в системе и направив туда свои усилия, вы сможете выжать из существующего оборудования намного больше. В конце концов, зачем покупать новую систему, если проблема решается покупкой нескольких гигабайтов относительно недорогой памяти? (Конечно, если ваша цель – сделать эту «медленную» систему своей новой настольной машиной, совсем другое дело.)

Ввод-вывод – также одно из обычных узких мест. Шина PCI имеет вполне определенный предел пропускной способности, и хотя фактическая производительность дисков или сетевой карты далека от их пропускной способности, замедление работы системы может быть вызвано сильной нагрузкой на шину PCI за счет их интенсивного использования.

Типичная причина замедления работы системы – одновременный запуск нескольких больших программ. Например, однажды я запланировал ротацию протоколов крупной базы данных, подразумевавшую перемещение и сжатие файлов, размер которых составлял гигабайты. Такая ротация затевалась одновременно с автоматическими проверками,

Что есть норма?

В этой главе описано много *отклонений* от нормы. Как системному администратору вам положено знать, что для вашей системы есть норма. Это сродни искусству: можно затрудниться описать *норму*, но вы обязаны отличать *ненормальность*, увидев ее. Используйте эти инструменты регулярно, даже при нормальном поведении системы, – и когда производительность понизится, у вас могут появиться неплохие мысли по поводу решения проблемы. Будьте внимательны к аппаратному обеспечению!

которые система выполняла ежедневно. Поскольку задание требовало останова основной базы данных и приводило к простою системы, скорость была очень важна. Производительность обоих процессов невыносимо замедлилась. Перенос одного из заданий ускорил работу обоих.

В этой главе мы рассмотрим несколько инструментов FreeBSD, позволяющих исследовать производительность системы. Вооружившись информацией, мы обсудим, как устранять неполадки с производительностью. Для исследования каждого потенциального узкого места есть различные инструменты. Система FreeBSD непрерывно изменяется; в более новых системах могут появиться новые инструменты настройки производительности. Обратитесь к странице руководства tuning(7) вашей системы и поищите новые советы по производительности.

Проверка сети

Если производительность системы снижается из-за сети, то необходима более широкая полоса пропускания или более эффективное управление пропускной способностью. Это можно выразить примерно так: «Невозможно втиснуть десятитонную полосу пропускания в пятитонный канал». Если переполнение канала T1 вызывает общее замедление, перестаньте переполнять его!

Выявляя неполадки, начните с мониторинга системы и выясните, какую полосу пропускания она занимает. Выполните команды `netstat -m` и `netstat -s` и посмотрите наличие ошибок или мест, где ощущается нехватка памяти или буферов. Выявите наиболее активные источники трафика с помощью анализатора пакетов или Netflow, задействуйте MRTG или Cricket для отслеживания трафика на долгосрочной основе.

Другие источники проблем могут иметь более сложную природу. Начните с выяснения того, где эти проблемы находятся. Здесь поможет vmstat(8).

Выявление узких мест с помощью vmstat(8)

FreeBSD включает несколько программ, позволяющих выяснить производительность системы. В том числе vmstat(8), iostat(8) и systat(1). Мы обсудим программу vmstat, потому что я считаю ее наиболее полезной. Программа iostat(8) похожа на vmstat(8), а systet(8) выводит ту же информацию в более удобном виде.

Программа vmstat(8) предоставляет статистику использования виртуальной памяти в текущий момент. К выводу vmstat(8) надо привыкнуть, но он очень хорош тем, что отображает большой объем данных в маленьком пространстве. Наберите `vmstat` в командной строке и посмотрите, что получилось.

```
# vmstat
procs      memory      page                    disks  faults     cpu
```

```
 r b w     avm     fre  flt re pi po    fr  sr ad0   in   sy   cs us sy id
 3 0 0  580416 1388912  234 11  7  0   209   0   0  171 5796 3816  3 10 87
```

Экран разбит на шесть разделов: процесс (`procs`), память (`memory`), пейджинг (`page`), диски (`disks`), ошибки (`faults`) и процессор (`cpu`). Кратко рассмотрим каждый раздел, а затем углубимся в вопросы, имеющие наибольшее значение при исследовании производительности.

Процессы

Под заголовком `procs` есть три колонки.

`r`

Количество процессов, ожидающих времени процессора. Они готовы к запуску, но не могут получить доступ к процессору. Если их много, то узким местом системы является процессор.

`b`

Количество процессов, блокированных в ожидании системного ввода или вывода. Обычно это ожидание доступа к диску. Такие процессы будут запущены сразу после получения нужных им данных. Если это число велико, узким местом является диск.

`w`

Показывает процессы, которые были запущены, но полностью вытеснены. Если процессы начинают вытесняться регулярно, размер памяти не соответствует заданиям, выполняемым в системе.

Память

FreeBSD разбивает память на фрагменты, которые называются *страницами*. Все страницы, выделяемые процессам, имеют одинаковый размер. Размер страницы зависит от аппаратной платформы и операционной системы. Система обрабатывает страницы целиком, например, если требуется вытеснить память в пространство свопинга (область подкачки), она вытесняется целыми страницами. Раздел `memory` состоит из двух колонок.

`avm`

Среднее количество страниц виртуальной памяти, используемых системой. Если это значение чрезмерно велико, значит, система активно расходует виртуальную память.

`fre`

Количество страниц памяти, доступных для использования. Если это значение чрезмерно мало, налицо проблемы с памятью.

Пейджинг

Раздел `page` показывает, как работает система виртуальной памяти. Внутренние механизмы виртуальной памяти – тайная наука, которую я не буду здесь описывать.

`flt`

Количество ошибок из-за отсутствия страницы (page faults), когда нужной информации нет в памяти и приходится загружать ее с диска, из пространства свопинга.

`re`

Количество страниц, восстановленных из кэша и повторно использованных.

`pi`

Сокращение от *pages in*. Показывает, сколько страниц было перемещено из физической памяти в пространство свопинга.

`po`

Сокращение от *pages out*. Показывает, сколько страниц было перемещено из пространства свопинга в реальную память.

`fr`

Количество освобождаемых страниц в секунду.

`sr`

Количество просмотренных страниц в секунду.

Вытеснение памяти в пространство свопинга – это совсем неплохо, но постоянное перемещение страниц из области подкачки в реальную память свидетельствует о нехватке памяти. Большие значения в колонках `fr` и `flt` могут свидетельствовать о чрезмерном количестве короткоживущих процессов, например CGI-сценарии запускают множество других процессов или слишком часто планируется выполнение некоторых заданий в cron.

Диски

Раздел `disks` показывает все диски по именам устройств. Числа, показанные здесь, – это количество дисковых операций в секунду, ценная подсказка для тех, кто стремится выяснить, как хорошо диски справляются с нагрузкой. При любой возможности следует распределять дисковые операции между различными дисками, а диски размещать на разных шинах. Если один диск загружен больше остальных, а система ожидает доступа к диску, то наиболее часто используемые файлы надо переместить с одного диска на другой. Одна из постоянно встречающихся причин высокой нагрузки на диски – это аварийное завершение программ с формированием дампа памяти на диске, которые могут перезапускать себя. Например, CGI-сценарий, содержащий ошибку и аварийно завершившийся с формированием дампа памяти на диске, всякий раз, когда кто-то щелкает на ссылке, может существенно увеличивать нагрузку на жесткий диск.

Если дисков много, можно заметить, что не все они появляются в выводе vmstat. Программа vmstat(8) разрабатывалась для работы с экраном шириной 80 символов, поэтому она не может отобразить все диски,

если их много. Однако если у вас широкий экран и вы не против того, чтобы вывод программы в ширину превысил ограничение в 80 символов, воспользуйтесь ключом `-n`, чтобы указать желаемое количество дисков для отображения.

Ошибки

В ошибках (faults) нет ничего плохого, они представляют собой всего лишь полученные системные прерывания. Плохо, когда их слишком много, но прежде чем заняться этой проблемой, необходимо знать, какое значение для вашей системы является нормальным.

`in`

Количество системных прерываний (запросов IRQ), полученных системой за последние пять секунд.

`sy`

Количество системных вызовов за последние пять секунд.

`cs`

Количество переключений контекста (context switches) за последнюю секунду или среднее число переключений в секунду с момента последнего обновления. (Например, если показания vmstat обновляются каждые пять секунд, в этой колонке отображается среднее число переключений контекста в секунду за последние пять секунд.)

Процессор

Наконец, в разделе `cpu` показано, сколько времени система потратила на выполнение пользовательских (`us`) и системных задач (`sy`) и сколько времени система бездействовала (`id`). Программа top(1) представляет эту же информацию в более дружественном формате, но только для текущего времени, тогда как vmstat позволяет просматривать деятельность системы за длительный период.

Использование vmstat

Как же применить эту информацию? Прежде всего, проверьте первые три колонки и выясните, чего ждет система. Если она ожидает доступа к процессору (колонка `r`), значит, не хватает мощности процессора. Если система ждет завершения дисковых операций (колонка `b`), то узким местом являются диски. Если часто происходит вытеснение процессов (колонка `w`), то в системе не хватает памяти. Другие колонки помогут более детально исследовать причины нехватки этих трех видов ресурсов.

Непрерывное выполнение vmstat

Применяя vmstat, вы наверняка больше заинтересованы в получении информации за определенный период времени, нежели в изучении состояния системы в тот или иной момент. С помощью ключа `-w` и числа

можно запустить программу vmstat так, что она будет непрерывно обновлять показания на экране через заданное число секунд. Параметры использования виртуальной памяти пересчитываются внутренними счетчиками системы каждые пять секунд, хотя другие счетчики обновляются непрерывно:

```
# vmstat -w 5
procs      memory   page                 disks     faults      cpu
r b w     avm   fre flt re pi po  fr  sr ad0 ad4   in  sy   cs us sy id
1 1 0  184180 12876   8  0  0  0  30  11   0   0 1143 122  517  0  1 99
0 1 0  184180  8360   1  0  0  0 260   0 373   0 1513 419 2287  0  5 95
0 1 0  184180  7984   0  0  0  0 224   0 387   1 1528 428 2318  0  5 95
0 1 0  184180 14892   0  0  0  0  62 346 188   1 1328 221 1322  0  3 96
...
```

Через каждые пять секунд в конце будет появляться новая строка с данными. Теперь можно спокойно наблюдать, как меняется производительность системы, когда запускаются запланированные задания или какие-либо программы. Завершив наблюдение, нажмите комбинацию клавиш Ctrl-C. В этом примере видно, что иногда процессы ждут освобождения процессора (о чем говорит изредка появляющееся число 1 в столбце r) и постоянно – доступа к диску.

Если процессы изредка ожидают освобождения того или иного ресурса, то это еще не означает, что данный системный компонент нуждается в обновлении. Не обращайте на это внимания, если производительность системы приемлема. Однако если это не так, следует продолжить исследование. В этом случае я наблюдаю за результатами vmstat(8), так как они свидетельствуют о наличии проблем с производительностью. vmstat(8) наглядно показывает, что процессы постоянно ждут, пока появится возможность обратиться к диску. Давайте посмотрим, насколько заняты наши диски.

Дисковый ввод-вывод

Управление дисками мы рассматривали сначала в главе 8, а затем в главе 18. Как правило, быстродействие диска существенно ограничивает суммарную производительность вычислительной системы. Если программы постоянно ожидают завершения дисковых операций, прежде чем продолжить работу, их производительность начинает снижаться. Такая ситуация, называемая *блокированием на диске*, препятствует нормальной работе программ. Действенное решение состоит в том, чтобы установить более быстрые диски или больше дисков, либо перераспределить нагрузку на диски.

Операционная система FreeBSD предоставляет несколько инструментов для проверки дисковой активности, но я предпочитаю пользоваться утилитой gstat(8), поэтому продемонстрирую ее. Достаточно просто запустить команду gstat без аргументов, и она будет выводить обновленную информацию примерно каждую секунду:

```
L(q) ops/s  r/s  kBps  ms/r  w/s  kBps  ms/w  %busy Name
   0     0    0     0   0.0    0     0   0.0   0.0| ❶ad0s1b.eli
   1   213  211  1066   4.6    2     4   9.2  97.9| ❷ad0
   0     0    0     0   0.0    0     0   0.0   0.0| acd0
   1   213  211  1066   4.7    2     4   9.2  98.0| ❸ad0s1
   0     0    0     0   0.0    0     0   0.0   0.0| ad0s1a
   0     0    0     0   0.0    0     0   0.0   0.0| ad0s1b
   0     0    0     0   0.0    0     0   0.0   0.0| ad0s1c
   0     0    0     0   0.0    0     0   0.0   0.0| ad0s1d
   0     2    0     0   0.0    2     4   9.2   0.9| ad0s1e
   1   211  211  1066   4.7    0     0   0.0  98.2| ❹ad0s1f
```

Мы видим здесь разнообразную информацию о каждом диске, участке и разделе в виде отдельных строк для каждого устройства. Утилита gstat(8) показывает разнообразную полезную информацию, например, количество операций чтения в секунду (`r/s`), количество операций записи в секунду (`w/s`), скорость чтения и записи в килобайтах в секунду, а также число миллисекунд, затраченных на каждую операцию чтения и записи. Первое, на что следует обратить внимание, – колонка `%busy`, вторая справа.

Первая запись, раздел свопинга ❶, у нас простаивает. А диск ad0 ❷ загружен на 97 процентов. Неудивительно, что ощущается замедление работы системы! Чуть ниже видно, что наибольшая активность диска ad0 в действительности приходится на участок ad0s1 ❸. Большинство разделов простаивает, лишь небольшая активность наблюдается в разделе ad0s1e. А вот раздел ad0s1f ❹ *очень* перегружен, на него приходится 98,2 % активности и сотни операций чтения в секунду. Проверив содержимое файла */etc/fstab* или воспользовавшись командой mount(8), можно увидеть, что раздел ad0s1f является разделом */usr* данного сервера. Какие-то процессы читают много, очень много данных с диска... Но что это за процессы? Чтобы ответить на этот вопрос, необходимо воспользоваться другим инструментом.

Исследование процессора, памяти и операций ввода-вывода с помощью top(1)

Утилита top(1) предоставляет хороший обзор состояния системы, отображая информацию об использовании процессора, памяти и диска. Достаточно ввести команду `top`, чтобы получить полноэкранное отображение сведений о производительности системы. Информация обновляется каждые две секунды, что позволяет представить состояние системы практически в режиме реального времени.

```
❶last pid:  977;  ❷load averages:  1.14,  0.80,  0.18  ❸up 0+00:18:03  18:30:34
❹41 processes:  2 running,  39 sleeping
❺CPU states:  0.0% user,  0.0% nice,  2.1% system,  0.0% interrupt,  97.9% idle
❻Mem: 106M Active, 139M Inact, 130M Wired, 8164K Cache, 68M Buf, 1607M Free
❼Swap: 4096M Total, 4096M Free
```

```
PID USERNAME THR PRI NICE   SIZE    RES STATE  C  TIME  WCPU COMMAND
840 mwlucas    1  96    0   278M 24740K select 1  0:33 1.76% Xorg
873 mwlucas    1  96    0   113M 81916K CPU0   0  0:51 0.59% acroread
903 mwlucas    6  20    0   169M   103M kserel 0  0:31 0.00% soffice.bin
853 mwlucas    1  96    0  7924K  6608K select 1  0:04 0.00% wmaker
762 root       1  96    0  3208K   964K select 1  0:02 0.00% moused
859 mwlucas    1   4    0 18496K 10544K select 0  0:00 0.00% yank
...
```

Довольно плотно, не так ли? top(1) пытается втиснуть как можно больше данных в стандартное окно терминала 80Ч25 символов. Остановимся на этом выводе и объясним смысл каждой записи.

Значения PID

Каждый процесс в машине UNIX имеет уникальный *идентификатор процесса*, или *PID*. Когда бы ни стартовал процесс, ему назначается PID, значение которого на единицу превышает PID предыдущего процесса. Поле `last pid` – это идентификатор процесса, запущенного в системе последним. В примере в этом поле видим значение 977 ❶. Следующий запущенный процесс получит идентификатор 978, затем 979 и т. д. Наблюдение за этим числом позволит увидеть, как быстро изменяется система. Если число `last pid` растет слишком быстро, это может свидетельствовать о выходе из-под контроля[1] какого-то ветвящегося процесса[2] или об аварийном состоянии того или иного демона.

Средняя нагрузка

Поле *load average* (средняя нагрузка) ❷ – это не совсем понятное число. Его назначение – предоставить приблизительные данные о нагрузке на процессор системы. Средняя нагрузка равна среднему числу потоков, ожидающих выделения времени процессора. (В других операционных системах средняя нагрузка вычисляется другими методами.) Приемлемое значение средней нагрузки зависит от системы. Если оно чрезмерно, значит, надо исследовать работу системы. Многие машины класса Pentium сталкиваются с трудностями при средней нагрузке, равной 3, а некоторые современные системы быстро работают при средней нагрузке, равной 10.

---

[1] Некоторые пользователи действительно пытаются израсходовать ресурсы системы полностью, запуская слишком много программ и закладывая при этом так называемую *бомбу размножения* (*forkbomb*). Такие пользователи – как script kiddies («детки со скриптами», производители разрушительных вирусов), но не настолько образованные и без чувства самосохранения.

[2] Forkbomb – от названия функции fork(), используемой при запуске нового процесса, так что здесь речь не о разветвлении, а о том, что в системе могут появиться процессы, которые быстро размножаются (например, посредством рекурсивного вызова функции fork() или просто порождая множество новых процессов). – *Прим. науч. ред.*

top(1) выдает три значения средней нагрузки. Первое (`1.14` в нашем примере) – это средняя нагрузка за последнюю минуту. Второе (`0.80`) – за последние 5 минут, а третье (`0.18`) – за предыдущие 15 минут. Если средняя нагрузка за последние 15 минут высока, а за последнюю минуту – низка, то налицо высокий пик активности системы, который пришелся на этот 15-минутный интервал. С другой стороны, если 15-минутное значение невелико, а нагрузка за последнюю минуту высока, значит, что-то произошло за последние 60 секунд и, возможно, продолжается сейчас. Если велики все значения средней нагрузки, то такое состояние удерживалось все 15 минут.

Uptime

Последнее поле в первой строке – *uptime* ❸. Оно сообщает о том, как долго работает система. В нашем примере система работает 18 минут, а текущее время – 18:30:34. Предоставляю вам возможность вычислить, когда произошла загрузка системы.

Количество процессов

Во второй строке ❹ представлена информация о процессах, запущенных в системе в настоящее время. Работающие (running) процессы непосредственно выполняют работу; они отвечают на запросы пользователей, обрабатывают почту и выполняют все другие операции. Спящие (sleeping) процессы ожидают входных данных от того или иного источника. Это хорошие состояния процессов. Процессы в других состояниях обычно ожидают доступности ресурсов или в некотором роде зависли. Большое количество неспящих, неработающих процессов может свидетельствовать о неполадках. Команда ps(1) покажет состояние всех процессов.

Типы процессов

Строка `CPU states` ❺ сообщает, какая доля доступного времени процессора (в процентах) расходуется на обслуживание процессов различных типов. В этой строке представлены пять различных типов процессов: `user`, `nice`, `system`, `interrupt` и `idle`.

Пользовательские процессы (`user`) – это средние повседневные программы. Это могут быть демоны, запущенные от имени root, команды, запущенные обычными пользователями, или что-нибудь другое. Если процесс представлен в выводе команды `ps -ax`, это пользовательский процесс.

Процессы `nice` – это процессы, приоритеты которых может определять пользователь. Более подробно они рассмотрены в разделе «Изменение приоритетов с помощью nice» этой главы.

Значение `system` дает общее время процессора (в процентах), потраченное процессами ядра FreeBSD и пользовательскими процессами при

выполнении в пространстве ядра. В состав таких процессов входят обработчики виртуальной памяти, сетевые процессы, процессы записи на диск, отладка с параметрами INVARIANTS и WITNESS и т. д.

Значение interrupt показывает, сколько времени система затрачивает на обработку запросов прерывания (IRQ).

Наконец, поле бездействующих процессов (idle) показывает, сколько времени система ничего не делает. Если время бездействия процессора очень мало, то стоит задуматься о том, чтобы пересмотреть порядок планирования заданий, или о покупке более быстрого процессора.

top и SMP

При работе с многопроцессорной системой следует иметь в виду, что top(1) отображает средние показатели нагрузки по всем процессорам. Один процессор может быть полностью занят компиляцией, а другой может бездействовать, и в этом случае top(1) покажет, что система загружена на 50 %.

Память

Далее идет строка Mem ❻, представляющая действительную физическую память. Операционная система FreeBSD делит информацию об использовании памяти на несколько различных категорий.

Активная память (Active) – это общая емкость памяти, которая в данный момент задействована для выполнения пользовательских программ. Когда выполнение программы завершается, использованная ею память попадает в *неактивную память* (Inact). Данные, полученные с диска, помещаются в *кэш* (Cache). Если системе потребуется вновь запустить эту же программу, она будет взята из кэша, а не с диска.

Аналогично, запись Buf показывает емкость буфера памяти. Этот буфер содержит данные, недавно полученные с диска. Категория Buf фактически является подмножеством категорий активной памяти, неактивной памяти и кэша, а не отдельной, самостоятельной категорией.

Свободная память (Free) вообще не задействована. Это может быть память, к которой еще не было обращений, или память, освобожденная процессом. В этой системе присутствует 1 607 Мбайт свободной физической памяти. Если по истечении месяца на сервере еще остается свободная память, можно подумать о том, чтобы перенести часть памяти на машину, где ощущается ее нехватка. Результаты работы top(1) в этом примере были получены на моем ноутбуке, и ему необходим каждый бит памяти, которую он сможет получить.

В процессе поддержки пула доступной памяти операционная система FreeBSD постоянно тасует память между категориями активной, неак-

тивной памяти и кэшем. Память в кэше легко может перейти в категорию свободной памяти. Когда начинает ощущаться нехватка памяти в кэше, а потребность в свободной памяти остается высокой, FreeBSD выбирает страницы из пула неактивной памяти, проверяет, могут ли они использоваться в качестве свободной памяти, и перемещает их в пул свободной памяти. FreeBSD старается сохранить общее число страниц свободной памяти и кэша выше значения sysctl `vm.v_free_target`. (Размер страницы определяется параметром sysctl `hw.pagesize`, значение которого для платформ i386 и amd64 составляет 4 096 байт.)

Наличие свободной памяти в системе вовсе не означает, что в системе достаточно памяти. Если vmstat(8) показывает большой объем операций с пространством свопинга, это свидетельствует о нехватке памяти. В системе может работать программа, которая постоянно освобождает память. Кроме того, FreeBSD перемещает некоторые страницы из кэша в пул свободной памяти, чтобы поддержать заданный уровень свободной памяти.

Связанная память (`Wired`) – это память, применяемая для структур данных ядра, а также для особых системных вызовов, которым память нужна немедленно. Связанная память никогда не вытесняется в пространство свопинга.

Своп

Далее идет строка `Swap` ❼, содержащая данные о доступном пространстве свопинга (области подкачки) и о ее задействованном объеме. При свопинге (swapping) система использует дисковый накопитель как дополнительную память. Далее в этой главе пространство свопинга рассмотрено более подробно.

Список процессов

Наконец, top(1) выводит список процессов системы и их базовые характеристики. Формат таблицы подразумевает вывод как можно большего объема информации в максимально сжатой форме. Каждый процесс описан в отдельной строке.

PID

Первая колонка – числовой идентификатор процесса, или PID. Каждый процесс, запущенный в системе, имеет уникальный PID. Выполняя команду kill(1), вы указываете процесс по его PID. (Если PID процесса неизвестен, можно использовать команду pkill(1), чтобы указать процесс по его имени.)

USERNAME

Следующая колонка – имя пользователя, от имени которого запущен процесс. Если несколько процессов с одним пользовательским ID интенсивно потребляют время процессора или память, вы знаете, с кем поговорить.

PRI и NICE

Колонки PRI (priority – приоритет) и NICE взаимосвязаны. Они показывают, какой приоритет система назначает тому или иному процессу. О приоритетах и nice речь пойдет чуть далее.

SIZE

Это объем памяти, которую система выделила для этого процесса.

RES

В колонке RES (Resident Memory) показано, какая часть программы действительно находится в памяти (резидентна). Для программы может быть отведен огромный объем памяти, но задействована ею будет лишь малая его часть. Например, в то время, как я пишу эти строки, моей программе OpenOffice.org выделено 169 Мбайт, но используется только 103 Мбайт. (Интересное наблюдение: в память, используемую X.org, включается память видеокарты, которая не входит в состав физической памяти системы.)

STATE

В колонке STATE показано, что делает процесс в данный момент. Процессы могут пребывать в разных состояниях: ожидать ввода данных; спать, пока их кто-нибудь не разбудит; выполняться и т. д. Здесь можно видеть имя системного вызова, такого как select, pause или ttyin, в котором процесс ожидает наступления определенного события. В многопроцессорных системах для работающих процессов можно видеть, на каком процессоре он исполняется. В нашем примере процесс acroread исполняется на процессоре CPU0.

TIME

Колонка TIME сообщает, какой объем процессорного времени использовал процесс.

WCPU

Колонка WCPU (использование CPU) выдает взвешенные показатели использования CPU, а именно процентное отношение времени CPU, которое процесс использует согласно своему приоритету и значению nice.

COMMAND

Наконец, в колонке COMMAND представлено имя программы.

Вывод команды top(1) позволяет понять, где система тратит больше всего времени.

top(1) и ввод/вывод

Помимо стандартной статистики использования процессора, у top(1) есть режим вывода информации об операциях ввода-вывода, в котором отображаются процессы, наиболее активно использующие диск. Чтобы перейти в режим отображения информации об операциях вво-

да-вывода, нужно после запуска top(1) нажать клавишу M. В верхней части экрана по-прежнему выводится информация об использовании памяти, пространства свопинга и о состоянии процессора, но нижняя часть существенно изменилась.

```
  PID USERNAME VCSW IVCSW READ WRITE FAULT TOTAL PERCENT COMMAND
 3064 root       89     0   89     0     0    89 100.00% tcsh
  767 root        0     0    0     0     0     0   0.00% nfsd
 1082 mwlucas     2     1    0     0     0     0   0.00% sshd
 1092 root        0     0    0     0     0     0   0.00% tcsh
  904 root        0     0    0     0     0     0   0.00% sendmail
...
```

Колонка PID – это идентификатор процесса, а в колонке USERNAME выводится имя пользователя, от лица которого был запущен процесс.

Название колонки VCSW происходит от *voluntary context switches* (*добровольное переключение контекста*) и показывает, сколько раз процесс добровольно уступил систему другим процессам. Название колонки IVCSW означает *involuntary context switches* (*принудительное переключение контекста*) и показывает, сколько раз ядро сообщало процессу: «Ваше время истекло, пора дать поработать другим процессам».

Аналогично, в колонках READ и WRITE выводится число дисковых операций чтения и записи, произведенных системой. В колонке FAULT показано, как часто этот процесс перемещал страницы памяти из пространства свопинга, что по сути является одной из разновидностей дисковых операций. Сумма значений этих трех столбцов выводится в колонке TOTAL.

В колонке PERCENT показан процент дисковых операций, приходящийся на долю этого процесса. В отличие от gstat(8), команда top(1) выводит все значения в процентах от фактической дисковой активности, а не от максимальной возможной. Если в системе имеется единственный процесс, использующий диск, top(1) покажет, что этому процессу принадлежат все 100 % дисковых операций, даже если он всего лишь записал немного данных. Команда gstat(8) показывает уровень нагрузки на диск, а top(1) – какие процессы ответственны за эту нагрузку. Здесь мы видим, что все дисковые операции выполняются процессом с идентификатором 3064. Это процесс командного интерпретатора tcsh(1). Давайте понаблюдаем за «злодеем».

Дополнительные возможности команды top

Команда top может выводить информацию на экран разными способами. С ее помощью можно просматривать процессы, принадлежащие определенному пользователю, включая или исключая потоки ядра, и т. д. За дополнительной информацией обращайтесь к странице руководства.

Исследование процессов

В любой UNIX-подобной системе все процессы связаны между собой отношениями «родитель-потомок». Во время загрузки FreeBSD в результате запуска init(8) создается единственный процесс, которому присваивается идентификатор PID, равный 1. Этот процесс запускает другие процессы, такие как сценарий начальной загрузки */etc/rc* и программа getty(8), которая обслуживает вход в систему. Эти процессы являются дочерними по отношению к процессу с идентификатором 1. Когда пользователь выполняет вход в систему, программа getty(8) запускает новый командный интерпретатор, который становится дочерним процессом по отношению к getty. Команды, запускаемые пользователем, становятся либо дочерними процессами по отношению к командному интерпретатору, либо его частью. Просмотреть эти отношения «родитель-потомок» можно с помощью команды ps(1) с ключами `-ajx` (помимо других).

```
# ps -ajx
USER      PID PPID PGID SID JOBC STAT TT     TIME COMMAND
root        0    0    0   0    0 WLs  ??  0:00.01 [swapper]
root        1    0    1   1    0 ILs  ??  0:00.01 /sbin/init --
root        2    0    0   0    0 DL   ??  0:00.79 [g_event]
...
haldaemon 826    1  826 826    0 Ss   ??  0:02.80 /usr/local/sbin/hald
root      827  826  826 826    0 I    ??  0:00.06 hald-runner
...
```

В первой колонке выводится имя пользователя, владеющего процессом, далее следуют идентификаторы самого процесса (`PID`) и его «родителя» (`PPID`). Это самая полезная для нас информация, которую здесь можно видеть, но мы вкратце рассмотрим и другие колонки.

Колонка `PGID` – это числовой идентификатор группы процессов, который обычно наследуется от родительского процесса. Программа может запустить новую группу процессов, и эта новая группа получит идентификатор PGID, равный идентификатору процесса. Значение идентификатора группы процессов используется для обработки сигналов и управления заданиями. Идентификатор сеанса, или `SID`: сеанс – это коллекция из одной или более групп процессов (PGID), которые обычно запускаются одним тем же пользователем или демоном. Процессы не могут переходить из одного сеанса в другой. Колонка `JOBC` содержит счетчик механизма управления заданиями и свидетельствует о том, что процесс был запущен под управлением этого механизма (то есть в фоновом режиме).

Колонка `STAT` показывает состояние процесса, то есть чем именно занимался процесс в момент запуска команды ps(1). Информация о состоянии процесса очень полезна, так как сообщает, простаивает ли процесс, в ожидании чего он простаивает и т. д. Настоятельно рекомендую прочитать раздел страницы руководства ps(1), в котором рассказывается о состояниях процессов.

В колонке TT указывается управляющий терминал процесса. В этой колонке выводится только окончание имени терминала, например, v0 – для ttyv0 или p0 – для ttyp0. В примере показаны только процессы, не имеющие управляющего терминала, о чем свидетельствует значение ??, однако вскоре мы увидим другой пример, с информацией об управляющих терминалах.

Колонка TIME показывает, как много процессорного времени было использовано процессом в пространстве пользователя и в пространстве ядра.

В последней колонке COMMAND приводится имя команды, запущенной родительским процессом. Процессы, имена которых приводятся в квадратных скобках, – это потоки ядра, а не настоящие процессы.

Итак, как эта информация поможет в исследовании процессов? Наш пример вывода команды top(1) показал, что источником всей дисковой активности является процесс 3064. Запустим команду ps -ajx и посмотрим информацию об этом процессе:

```
    ...
    root     3035  3034  3035  2969  1 S+  p0  0:00.03  _su -m (tcsh)
❶   chris    2981  2980  2981  2981  0 Is  p1  0:00.03  -tcsh (tcsh)
❷   root     2989  2981  2989  2981  1 I   p1  0:00.01  su -m
❸   root     2990  2989  2990  2981  1 D   p1  0:00.05  _su -m (tcsh)
❹   root     3064  2990  3064  2981  1 DV+ p1  0:00.15  _su -m (tcsh)
    mwlucas  2996  2995  2996  2996  0 Is  p2  0:00.02  -tcsh (tcsh)
    ...
```

Владельцем интересующего нас процесса ❹ является пользователь root, и это экземпляр tcsh(1), о чем также сообщала команда top(1) в режиме отображения информации об операциях ввода-вывода. Однако в действительности этот командный интерпретатор был запущен под su(1), а это означает, что за проблемой производительности стоит реальный человек. Процесс 3064 – потомок процесса 2990 ❸, который является потомком процесса 2989 ❷, причем, владельцем обоих процессов является пользователь root. Однако процесс 2989 является потомком процесса 2981 ❶ – командного интерпретатора, запущенного реальным пользователем. Вы можете заметить и то, что все эти процессы принадлежат одному и тому же сеансу с идентификатором 2981. В колонке TT выводится значение p1, которое говорит о том, что пользователь вошел в систему через терминал */dev/ttyp1*, второй виртуальный[1] терминал на этой машине.

Это нормально, когда система испытывает короткие периоды полной занятости. Если никто больше не использует систему и никто не жалуется на падение производительности, почему бы не позволить этому пользователю запускать свои задания? Однако если этот процесс вызы-

[1] Обычно ttypN называют псевдотерминалом, а ttyvN – виртуальным терминалом. – *Прим. науч. ред.*

вает проблемы у других пользователей, мы можем понизить его приоритет, завершить работу задания, воспользовавшись привилегиями root, или зайти к этому пользователю в кабинет с бейсбольной битой.

Пейджинг и свопинг

В применении пространства свопинга (области подкачки) нет ничего плохого. FreeBSD использует область подкачки как виртуальную память. Жесткий диск намного медленнее оперативной памяти, однако область подкачки так или иначе работает, а многим программам вовсе не надо хранить все данные в оперативной памяти. Обычно программы тратят 80% своего времени на исполнение 20% своего кода. Большая часть остального кода – это код запуска и завершения программы, обработки ошибок и т. д. Вы можете поместить этот код в область подкачки без серьезного снижения производительности.

С помощью области подкачки выполняется кэширование данных. Начав использовать область подкачки, процесс будет продолжать его использовать, пока либо сам не завершит работу, либо не отзовет память из области подкачки.

Под использованием области подкачки подразумеваются операции *пейджинга* (*paging*) и *свопинга* (*swapping*). Когда происходит пейджинг – это нормально; свопинг – это не так хорошо, но лучше, чем аварийное завершение программы.

Пейджинг

Система FreeBSD выполняет операцию пейджинга, когда она перемещает часть запущенной программы в пространство свопинга. Пейджинг действительно может способствовать повышению производительности в тяжело нагруженной системе, поскольку незадействованные данные хранятся на диске, пока они не понадобятся, а всю оперативную память можно отдать работающему коду. Да и что случится, если вытеснить в область подкачки код запуска базы данных после того как она уже запущена?

Свопинг

Если в компьютере недостаточно физической памяти для хранения процесса, который не работает в данную микросекунду, система может переместить весь процесс в пространство свопинга. Затем, когда процессор запустит этот процесс в следующий раз, FreeBSD перенесет образ процесса из области подкачки в физическую память и запустит его, а в область подкачки, возможно, будут отправлены другие процессы.

Сложности со свопингом заключаются в том, что использование диска поставлено с ног на голову, а производительность значительно падает. Поскольку запросы обрабатываются дольше, в системе одновременно может скопиться много запросов. Протоколирование с целью изуче-

ния проблемы лишь усугубляет ситуацию, поскольку само протоколирование – это дополнительный системный процесс. Некоторые системы могут обслуживать определенный объем пространства свопинга, а в других ситуация резко ухудшается. Такое влияние на производительность иногда называют спиралью смерти (death spiral).

Когда перегружен процессор – производительность системы падает, когда узким местом являются диски – производительность системы падает. Нехватка памяти может вызвать крах системы. Если причиной замедления является свопинг, вам *необходимо* приобрести дополнительную память либо смириться с неудовлетворительной производительностью.

Команда vmstat(8), в частности, показывает число вытесненных в область подкачки процессов в каждый момент времени.

Настройка производительности

Система FreeBSD кэширует в памяти данные, к которым обращались совсем недавно, потому что одни и те же данные считываются с диска ужасно часто. Если информация кэшируется в оперативной памяти, получить ее можно очень быстро. Если системе требуется больше памяти, она выталкивает из кэша наиболее старые записи, освобождая место для новых данных.

Например, обсуждаемый вывод команды top(1) был получен на моем ноутбуке. Я запустил Firefox, чтобы посмотреть свои комиксы.[1] Индикатор диска помигал некоторое время, пока программа считывалась с диска. Затем я закрыл броузер, чтобы приступить к работе, но FreeBSD оставила Firefox в буферном кэше. Когда я снова запустил Firefox, он был вызван из кэша, а не с диска, поэтому запустился намного быстрее. Запусти я другой большой процесс, он «вытолкнул» бы веб-броузер из кэша, чтобы поместить в него новую программу.

Если ваша система работает как полагается, у вас есть как минимум несколько мегабайт свободной памяти. Параметры sysctl `vm.v_free_target` и `hw.pagesize` сообщат вам, как много свободной памяти должно быть в системе, по мнению FreeBSD. Если свободной памяти больше, чем произведение значений этих двух параметров, значит, ваша система не в полной мере задействует свой потенциал. Например, у меня на ноутбуке:

```
# sysctl vm.v_free_target
vm.v_free_target: 13745
# sysctl hw.pagesize
hw.pagesize: 4096
```

[1] Sluggy Freelance (*www.sluggy.com*), Help Desk (*www.ubersoft.net*) и User Friendly (*http://www.userfriendly.org*), если это кому-то интересно. Опасайтесь кролика.

Моя система полагает, что необходимо иметь, по крайней мере, 13 745×4 096 = 56 299 520 байтов, или порядка 54 Мбайт свободной памяти. Я мог бы потерять гигабайт памяти на моем ноутбуке не дрогнув, если бы не страдал от нехватки памяти.

Память

Если в системе достаточно много памяти занято кэшем или буфером, то она не испытывает нехватку памяти. Дополнительная память может принести пользу, но ее наличие не строго обязательно. Если в системе мало свободной памяти, но много активной и связанной памяти, значит, система испытывает недостаток памяти. Дополнительная оперативная память позволит вам использовать преимущества буферного кэша.

В случае нехватки свободного пространства и минимума места (или его отсутствия) в кэше или буфере необходимо выяснить, куда расходуется память. Вполне вероятно, что системе ее не хватает. Как только система начнет использовать пространство свопинга, недостаток памяти перестанет ощущаться как нечто гипотетическое.

Пространство свопинга

Пространство свопинга (swap space) помогает справиться с кратковременной нехваткой памяти. Например, разархивируемый огромный файл может легко заполнить всю физическую память, и системе придется задействовать виртуальную. Не стоит покупать дополнительную память, если использоваться она будет лишь время от времени, – можно обойтись свопингом.

Проще говоря, свопинг чем-то напоминает вино. Бокал-другой время от времени не повредит, даже полезно, но регулярные обильные возлияния – это бич.

Процессор

Процессор может выполнять довольно много операций в секунду. Но если вы запустили больше задач, чем процессор может обслужить, запросы выстраиваются в очередь. Они будут накапливаться, а система замедлит свою работу. Так, в двух словах, работает процессор.

Если производительность ниже некуда, а top(1) показывает, что процессор занят около 100% своего времени, пора что-то делать. Можно, конечно, купить новый, но есть и другие варианты. Например, можно изучить процессы, запущенные в системе, и выяснить, все ли они необходимы. Не установил ли младший сисадмин клиент SETI@Home (*/usr/ports/astro/setiathome*), чтобы искать инопланетян за счет вашего процессора? Не запущены ли программы, бывшие важными когда-то, но не теперь? Найдите и завершите ненужные процессы. Кроме того, убедитесь, что они не будут запущены при следующей загрузке системы.

Выполнив эти шаги, снова измерьте производительность системы. Если проблемы остались, попробуйте перепланировать процессы или изменить их приоритеты.

Перепланирование

Перепланировать легче, чем изменить приоритеты. Это относительно простой способ сбалансировать процессы системы, чтобы они не монополизировали ресурсы системы. Мы уже говорили в главе 15, что для планирования запуска системных заданий в различное время вы и ваши пользователи могут задействовать cron(1). Если тот или иной пользователь запускает большие задания в определенное время, подумайте о том, чтобы перенести их выполнение на ночное время. Зачастую такие задания, как ежемесячный поиск по базе данных счетов, можно запускать между 6 часами вечера и 6 часами утра, и это никому не помешает – финансовому отделу данные нужны к 8 часам утра в первый день месяца, чтобы закрыть счета за прошедший месяц. Запуск `make buildworld && make buildkernel` тоже можно назначить на час ночи.

Изменение приоритетов с помощью nice

Если перепланирование не помогло, остается изменить приоритеты, что немного сложнее. Изменяя приоритет, вы предписываете системе FreeBSD изменить значимость данного процесса. Допустим, имеется некоторая программа, которая может работать часами, но мы хотим, чтобы она занимала ресурсы только в те моменты, когда система не занята ничем другим. В этом случае надо попросить программу быть *любезной* (*nice*) и уступить дорогу другим программам.

Чем любезнее процесс, тем меньше процессорного времени он запрашивает. По умолчанию любезность равна 0, но ее значение может находиться в диапазоне от 20 (очень любезный процесс) до -20 (совсем нелюбезный). Может показаться, что такой подход ставит все с ног на голову – как же, ведь больший номер должен означать больший приоритет! Однако, тут дело в самом термине; не думаю, что этот показатель лучше было бы назвать «неуживчивостью» или «капризностью».[1]

На экране top(1) приоритеты процессов показаны в колонке `PRI`. FreeBSD вычисляет приоритеты процессов, учитывая различные факторы, в том числе любезность, и при любой возможности система старается запускать первыми процессы с высокими приоритетами. Любезность влияет на приоритеты, однако приоритеты нельзя изменять напрямую.

Если известно, что система работает на пределе своих возможностей, запускаемым программам можно назначить степень их любезности с помощью nice(1). Укажите желаемый уровень любезности с помощью

[1] Кроме того, определения «неуживчивый» и «капризный» уже заняты самими системными администраторами.

команды `nice -n` и числового значения перед самой командой. Например, запустим `make buildworld` с уровнем любезности 15:

```
# nice -n 15 make buildworld
```

Отрицательные значения любезности, такие как `nice -n -5`, может назначать только root. Например, если требуется как можно скорее скомпилировать и наложить на ядро важную заплатку, укажите отрицательное значение любезности:

```
# cd /usr/src
# nice -n -20 make kernel
```

Обычно при запуске программы не надо указывать уровень любезности, однако нередко возникает потребность изменить уровень любезности «на лету», когда обнаруживается, что процесс полностью оккупировал ресурсы системы. Для этого вводится команда renice(8) с идентификатором PID или владельца процесса. Чтобы изменить уровень любезности процесса, запустите `renice`, указав в качестве аргументов новое значение любезности и идентификатор процесса.

Например, одна из моих систем выполняет протоколирование хоста, на котором запущены несколько экземпляров softflowd(8), flow-capture, Nagios и другие критически важные системы обслуживания сети. Кроме того, в этой системе работают другие программные компоненты, такие как сервер CVSup, используемый внутренними хостами сети. Если я нахожу, что сервер CVSup начинает мешать Nagios, выполняющему мониторинг сети, или серверу syslogd(8), я должен что-то предпринять. Уменьшение приоритета cvsupd(8) замедлит работу клиентов, но это лучше, чем замедлить систему мониторинга сети. С помощью pgrep(1) я определяю PID процесса cvsupd(8):

```
# pgrep cvsupd
993
# renice 10 993
993: old priority 0, new priority 10
```

Бац! Теперь FreeBSD будет планировать процесс cvsupd на исполнение реже, чем другие процессы. Конечно, пользователи сервиса будут очень

nice и tcsh

У командного интерпретатора tcsh(1) есть встроенная команда nice. Эта встроенная команда использует синтаксис команды renice(8), отличный от nice(1). Полагаю, это сделано не затем, чтобы вызвать неудовольствие пользователей tcsh, но отыскать разумные доводы пока не могу. Если вы непременно хотите использовать внешнюю утилиту nice(1), вызывайте ее с указанием полного пути к исполняемому файлу: */usr/bin/nice*.

недовольны, но поскольку главный пользователь – это я, и я уже недоволен, значит, все сделано правильно.

Изменить уровень любезности всех процессов, запущенных тем или иным пользователем, позволяет ключ `-u`. Например, чтобы придать моим процессам большую значимость, чем у всех чужих процессов, я ввожу такую команду:

```
# renice -5 -u mwlucas
1001: old priority 0, new priority -5
```

Число `1001` – это мой пользовательский ID в этой системе. Снова отмечу, что у меня есть достаточные основания так поступить. И не только затем, чтобы показать свое могущество.[1] Точно так же, если некий пользователь полностью оккупировал дисковый ввод-вывод, я могу сделать его процессы очень, очень любезными, что, скорее всего, избавит других пользователей от проблем.

Исследование программного обеспечения

Мы долго обсуждали настройку производительности FreeBSD. И это вполне логично, так как данная книга посвящена операционной системе FreeBSD. Но не стоит забывать о другом программном обеспечении! Нередко справиться с проблемами производительности можно за счет тонкой настройки программ, вызывающих эти проблемы. Например, у меня есть комплект CGI-сценариев на языке Perl, которые интенсивно работают с дисками и жадно потребляют время процессора. В результате поиска в Google я выяснил, что эти сценарии будут работать лучше под управлением mod_perl2. Я изменил конфигурацию веб-сер-

Узкие места

В каждой системе есть узкие места, ограничивающие производительность. Если устранить одно узкое место (bottleneck – «бутылочное горло»), производительность будет расти, пока не достигнет следующего узкого места. Производительность системы ограничена скоростью самого медленного компонента. Так, нередко скорость веб-сервера определяется пропускной способностью сети, поскольку самый медленный компонент системы – это соединение с Интернетом. Если обновить имеющуюся аппаратуру T1 с пропускной способностью 1,5 Мбит/с до OC-48 с пропускной способностью 2,4 Гбит/с, система будет выдавать веб-страницы с такой скоростью, с какой ей это позволят другие компоненты. Требование ликвидировать узкие места часто означает «ликвидировать узкие места, возникающие при нормальной рабочей нагрузке».

[1] Собственный эгоизм – не повод применять `renice -20` к своим процессам. По крайней мере, мне так говорили.

вера и проблема была ликвидирована. Это намного, намного проще, чем изменять конфигурацию системы.

Теперь, когда вы научились видеть проблемы, возникающие в системе, пора научиться слышать то, что система пытается вам сказать.

Письма о состоянии

Заглянув в */etc/crontab*, можно понять, что FreeBSD выполняет задания по обслуживанию системы каждый день, неделю и месяц с помощью команды periodic(8). Эти задания производят базовую проверку системы и уведомляют администратора об изменениях, компонентах, требующих внимания, и потенциальных «дырах» в защите. Вывод запланированных заданий ежедневно посылается пользователю root в локальной системе. Для того чтобы определить, чем занимается система, проще всего прочитать эти письма; многие системные администраторы, очень занятые – как и вы, – постарались, чтобы эти сообщения были полезными и необходимыми. Таких сообщений может быть слишком много, но вы довольно скоро научитесь бегло просматривать отчеты, обращая внимание только на критичные или необычные изменения.

Конфигурация ежедневных, еженедельных и ежемесячных отчетов определяется в *periodic.conf* (глава 10).

Поскольку вы вряд ли хотите каждый день регистрироваться под именем root на всех своих серверах, пересылайте всю свою почту в централизованный почтовый ящик. Для этого надо отредактировать файл */etc/aliases* (глава 16).

Единственное место, где я рекомендую отключать эти задания, – встроенные системы, которые управляются и проверяются с помощью других инструментов, таких как система мониторинга сети. В таких системах желательно отключать выполнение periodic(8) в */etc/crontab*.

Несмотря на то что отчеты высылаются ежедневно, они не дают всей картины событий. Лучше это делают файлы протоколов.

Протоколирование с помощью syslogd

Система протоколирования FreeBSD весьма полезна. Любые UNIX-подобные операционные системы позволяют протоколировать практически все происходящее с любой степенью подробности. По умолчанию система протоколирования регистрирует события для основных ресурсов, но администраторы могут конфигурировать протоколирование, как им надо. Почти все программы интегрированы с системой протоколирования syslogd(8).

Протокол syslogd работает с сообщениями. Программы отправляют отдельные сообщения, а демон syslogd(8) получает их и обрабатывает. syslogd обслуживает сообщения в соответствии с тем или иным источником сообщений (facility) и уровнем важности. Обе характеристики указыва-

ются в сообщениях, направляемых программе syslogd. Для управления системными протоколами надо понимать, что такое источник и уровень.

Источники

Источник – это тег, указывающий происхождение записи в протоколе. Это произвольное обозначение, простая текстовая строка, предназначенная для различения программ. В большинстве случаев у каждой программы, требующей уникального протокола, есть уникальное обозначение. Многие программы и протоколы имеют обозначения, выделенные специально для них. Например, FTP – как раз такой типичный протокол, для которого у syslogd есть специальное обозначение. Кроме того, syslogd поддерживает множество общих обозначений, которые могут быть задействованы любой программой.

Ниже перечислены стандартные обозначения и типы информации, которую они предоставляют.

`auth`

Общедоступная информация, относящаяся к авторизации пользователей во время входа в систему или при вводе команды su(1).

`authpriv`

Секретная информация, относящаяся к авторизации пользователей. Доступ к ней есть только у root.

`console`

Сообщения, которые обычно выводятся на системную консоль.

`cron`

Сообщения планировщика заданий.

`daemon`

Вместилище для сообщений всех системных демонов, не имеющих других явных обозначений.

`ftp`

Сообщения серверов FTP и TFTP.

`kern`

Сообщения ядра.

`lpr`

Сообщения системы печати.

`mail`

Сообщения почтовой системы.

`mark`

Это обозначение помещается в протокол каждые 20 минут. Полезно при комбинировании с другим протоколом.

`news`

Сообщения демонов Usenet News.

`ntp`

Сообщения сетевого протокола синхронизации времени (Network Time Protocol).

`security`

Сообщения различных программ защиты, таких как pfctl(8).

`syslog`

Сообщения системы протоколирования о самой себе. Однако не выполняйте протоколирование, когда получаете сообщения от syslogd. В противном случае может произойти путаница.

`user`

Вместилище пользовательских сообщений. Если вы не указали источник протоколирования для пользовательских программ, они задействуют это обозначение.

`uucp`

Сообщения протокола UNIX-to-UNIX Copy Protocol. Этот протокол применялся в UNIX до эпохи Интернета. Вы вряд ли когда-нибудь встретитесь с ним.

`local0 - local7`

Зарезервировано для нужд администратора. Во многих программах можно задавать источник протоколирования. Выберите одно из этих обозначений, если есть такая возможность. Например, сервис поддержки пользователей может протоколироваться как `local0`.

Хотя у большинства программ имеются определенные значения по умолчанию, тем не менее, это ваша задача как сисадмина – определить, какие программы и с какими обозначениями должны протоколироваться.

Уровни

Уровень протоколирования представляет относительную важность сообщений. Хотя программы посылают все данные протоколирования демону syslogd, большинство систем записывает только важную информацию, отбрасывая незначительные сообщения. Конечно, информация, которую один администратор считает пустяковой, может быть жизненно важной для другого. Именно здесь на помощь приходят уровни.

Протокол syslog предлагает восемь уровней важности. Эти уровни позволяют извещать syslogd о том, что надо записывать, а что следует отбросить. Вот перечень уровней в порядке убывания важности:

`emerg`

Паника системы. Сообщения поступают на каждый терминал. По существу, система в аварийном состоянии. Вам даже не понадобится перезагружать систему – она сделает это сама.[1]

[1] Все это может казаться забавным, пока не случится именно с вами.

`alert`

Плохо, но все же не как `emerg`. Система может продолжать работать, однако этой ошибке следует немедленно уделить внимание.

`crit`

Критические ошибки, например проблемы с аппаратными средствами, вроде сбойных блоков на жестком диске, или серьезные трудности с программным обеспечением. Можете продолжить работу, если хватит смелости.

`err`

Хотя эти ошибки и не разрушают систему, их следует исправить.

`warning`

Различные предупреждения, которые не мешают записавшим их программам продолжить работу.

`notice`

Общая информация, не требующая действий с вашей стороны, например факт запуска или останова демона.

`info`

Общая программная информация, например об отдельных транзакциях почтового сервера.

`debug`

Этот уровень обычно предназначен для программистов и лишь изредка – для сисадминов, которые пытаются выяснить, почему программа выполняется так, а не иначе. Протоколы отладки могут содержать всю информацию, необходимую программисту для отладки кода. Здесь же может оказаться конфиденциальная информация о пользователях.

`none`

Это означает «ничего не протоколировать с помощью этого обозначения». Чаще всего применяется для исключения информации из групповых записей (wildcard entries). Эта возможность будет рассмотрена ниже.

Комбинируя уровни с приоритетами, вы сможете разделить сообщения на весьма узкие категории и обрабатывать исходя из своих потребностей.

Обработка сообщений с помощью syslogd(8)

Демон syslogd(8) получает сообщения из сети и сравнивает их с записями в */etc/syslog.conf*. В этом файле две колонки. В первой описывается сообщение протокола либо по источнику и уровню, либо по имени программы. Во второй описывается действие, которое будет выполнено демоном syslogd(8), если протокольное сообщение соответствует

описанию. Например, рассмотрим следующую запись из файла *syslog.conf* по умолчанию:

```
mail.info               /var/log/maillog
```

Эта запись предписывает демону syslogd(8) протоколировать в файле */var/log/maillog* сообщения от источника `mail` с уровнем от `info` и выше.

Групповые символы

При указании информационного источника можно применять групповые символы, или символы шаблона (wildcards). Например, следующая строка предписывает протоколирование абсолютно всех сообщений от источника `mail`:

```
mail.*                  /var/log/maillog
```

Для протоколирования всех сообщений из любых источников раскомментируйте запись `all.log` и создайте файл */var/log/all.log*:

```
*.*                     /var/log/all.log
```

Результат вы получите, но информации будет слишком много, чтобы ее можно было применить на практике. Для нахождения нужных фрагментов придется выполнять сложные команды grep(1). Кроме того, в протоколы попадет вся секретная информация.

Исключение информации

Для исключения информации из протокола применяйте уровень `none`. Например, ниже из протокола, куда включаются все сообщения, исключается информация `authpriv`. Записи комбинируются в одной строке с помощью точки с запятой:

```
*.*; authpriv.none      /var/log/most.log
```

Сравнение

Также в правилах */etc/syslog.conf* можно применять операторы сравнения < (меньше), = (равно) и > (больше). По умолчанию syslogd записывает все сообщения указанного и более высоких уровней, но можно задать нужный диапазон уровней информации. Предположим, что вам нужно протоколировать все сообщения уровня `info` и выше в основном протоколе, а остальные сообщения – в файле с отладочной информацией:

```
mail.info       /var/log/maillog
mail.=debug     /var/log/maillog.debug
```

Запись `mail.info` собирает все протокольные сообщения источника `mail`, соответствующие уровню `info` или выше. Вторая строка охватывает только сообщения уровня не выше `debug`. В качестве источника сообщений нельзя указать `mail.debug`, иначе в протокол отладки попадут все записи предыдущего протокола! Таким образом, чтобы отследить деятельность почтового сервера, необязательно просматривать отладочную

информацию. А чтобы получить отладочный вывод почтового сервера, не нужно пробираться сквозь дебри информации о передаче почты.

Локальные источники

Многие программы предусматривают возможность протоколирования своих действий с помощью syslogd. Довольно часто они позволяют выбрать нужное обозначение. Для таких случаев зарезервировано несколько источников `local`. Например, по умолчанию dhcpd(8) (глава 15) использует источник `local7`. Вот настройка для перехвата таких сообщений и их направления в отдельный файл:

```
local7.*        /var/log/dhcpd
```

Если свободные источники `local` исчерпаны, можно взять другие незадействованные системные источники. Например, мне как-то пришлось воспользоваться источником `uucp` на нагруженном сервере протокола в сети, где не было служб `uucp`.

Протоколирование по имени программы

Если в системе нет источников, то для протоколирования можно задействовать имя программы. Для записи с именем требуются по крайней мере две строки: имя программы с восклицательным знаком в начале и строка с информацией о протоколировании. Например, для протоколирования ppp(8) подойдет такая запись:

```
!ppp
*.*             /var/log/ppp.log
```

В первой строке указано имя программы, а во второй групповые символы предписывают демону syslogd(8) добавлять в файл абсолютно все сообщения.

Синтаксис `!`*имя_программы* воздействует на все строки, следующие за этим определением, поэтому оно должно стоять последним в файле *syslog.conf*.

Протоколирование пользовательских сессий

Для протоколирования пользовательских сессий укажите имена пользователей через запятую. Тогда поступающие протокольные сообщения будут отображаться на терминалах указанных пользователей. Для отображения сообщений на терминалах всех пользователей в качестве места назначения укажите звездочку (`*`). Например, в файле *syslog.conf* по умолчанию есть такая строка:

```
*.emerg         *
```

Здесь сказано, что любое сообщение уровня `emerg` (авария) появится на терминалах всех пользователей. Поскольку такие сообщения обычно появляются, когда пора попрощаться с пользователями, они в любом случае не помешают.

Отправка протокольных сообщений программам

Наконец, если нужно, чтобы протоколы обрабатывала другая программа, применяйте символ конвейеризации (|) для перенаправления сообщений этой программе:

```
mail.*          |/usr/local/bin/mailstats.pl
```

Хост для протоколирования

Обычно в сетях я предусматриваю отдельный хост для протоколирования, обслуживающий не только машины FreeBSD, но и маршрутизаторы Cisco, коммутаторы, машины с другими версиями UNIX и любые другие системы, умеющие общаться с syslogd. Наличие единого хоста, на котором ведется все протоколирование, упрощает обслуживание систем и экономит дисковое пространство. Каждое сообщение в протоколе включает имя хоста, поэтому позднее легко отобрать нужные.

Для отправления протокольных сообщений другому хосту применяется символ @. Например, передача всех сообщений, предназначенных syslogd, на хост протоколирования в моей сети задается такой строкой:

```
*.*             @loghost.blackhelicopters.org
```

Файл *syslog.conf* на хосте протоколирования определяет окончательное место назначения для сообщений, которые получает хост.

На хосте протоколирования можно раздельно протоколировать события для разных хостов, с которых прибывают сообщения. Символ «плюс» (+) возле имени хоста означает, что к этому хосту применяются указанные ниже правила:

```
+dhcpserver
local7.*        /var/log/dhcpd
+ns1
local7.*        /var/log/named
```

При подобной конфигурации рекомендую помещать универсальные правила в начало файла *syslog.conf*, а правила для отдельных хостов – ближе к концу.

Перекрытие при протоколировании

syslog при проверке настроек не следует правилам первого или последнего совпадения; напротив, протоколируются все совпадения. Это означает, что одно и то же сообщение может появляться в нескольких протоколах. Рассмотрим фрагмент конфигурации syslog. (Первая строка взята из *syslog.conf* по умолчанию, просто я ее укоротил, чтобы уместить на страницу книги.)

```
*.notice;authpriv.none          /var/log/messages
local7.*                        /var/log/dhcpd
```

Все сообщения уровня `notice` и выше будут протоколироваться в файле */var/log/messages*. У нас есть сервер DHCP, события которого будут

протоколироваться в файле */var/log/dhcpd*. Это означает, что все сообщения DHCP уровня `notice` и выше будут протоколироваться в обоих файлах, */var/log/messages* и */var/log/dhcpd*. Мне это не нравится; сообщения сервера DHCP должны помещаться только в файл */var/log/dhcpd*. Я могу принудительно исключить сообщения DHCP из */var/log/messages* с помощью источника `none`:

```
*.notice;authpriv.none;local7.none      /var/log/messages
```

Моя конфигурация для файла */var/log/messages* очень быстро разрастается из-за необходимости исключить каждый локальный источник – и это правильно.

Пробелы и символы табуляции

По традиции UNIX-подобные операционные системы требуют разделять колонки в *syslog.conf* символом табуляции, однако FreeBSD разрешает применять пробелы. Символы табуляции нужны, только если разные UNIX-системы совместно используют один *syslog.conf*.

Настройка syslogd

Операционная система FreeBSD запускает демон syslogd по умолчанию, сразу же после установки она может служить сервером протоколирования. Работу syslogd можно настроить с помощью ключей командной строки. Ключи можно указать как в командной строке, запускающей syslogd, так и в файле */etc/rc.conf*, в виде параметра `syslogd_flags`.

Допустимые отправители сообщений

Можно жестко задать хосты, от которых syslogd будет принимать сообщения. Это предотвратит прием случайных сообщений из Интернета. Отправка множества протоколируемых сообщений может быть чьей-то попыткой переполнить ваш жесткий диск, то есть подготовкой к атаке. Хотя чаще это результат неправильной настройки. Сервер протоколирования в любом случае должен быть защищен брандмауэром. С помощью ключа `-a` определите IP-адреса или сети, которые смогут отправлять вам свои сообщения, как показано в двух (взаимно исключающих) примерах:

```
syslogd_flags="-a 192.168.1.9"
syslogd_flags="-a 192.168.1.0/24"
```

Хотя для создания подобных ограничений syslogd(8) примет имена хостов и доменов, определенные в DNS, все же DNS совершенно не подходит для управления доступом.

Ключ -s, применяемый в FreeBSD по умолчанию, позволяет запретить прием любых сообщений от удаленных хостов. Если вместо него использовать ключ -ss, syslogd(8) прекратит и отправку протоколируемых сообщений удаленным хостам. Применение ключа -ss исключает syslogd(8) из списка сетевых процессов, отображаемого программами sockstat(1) и netstat(1). Даже при том, что этот наполовину открытый сокет UDP не представляет угрозы, некоторые чувствуют себя комфортнее, когда syslogd(8) отключен от сети полностью.

Подключение к единственному адресу

По умолчанию syslogd(8) подключается к порту UDP 514 на каждом IP-адресе, присутствующем в системе. Серверу клеток необходим syslogd, но машина в клетке может запускать демоны, которые привязаны к единственному адресу. С помощью ключа -b можно заставить syslogd(8) подключаться к единственному IP-адресу:

```
syslogd_flags="-b 192.168.1.1"
```

Дополнительные сокеты для протоколирования

syslogd(8) может принимать сообщения не только через сетевые сокеты, но и через сокеты домена[1] UNIX. Стандартное местоположение для них – каталог */var/run/log*. Но ни один процесс, исполняющийся в chroot-окружении, не сможет получить к нему доступ. Если вы хотите предоставить эту возможность процессам в chroot-окружении, то либо настройте их на передачу сообщений по сети, либо создайте дополнительный сокет, по которому будет осуществляться прием сообщений. Для этого служит ключ -l, вместе с которым следует указать полный путь к дополнительному сокету:

```
syslogd_flags="-l /var/named/var/run/log"
```

Программы named(8) и ntpd(8) из состава FreeBSD обычно работают в chroot-окружении. Сценарий */etc/rc.d/syslogd* достаточно умен, чтобы добавить дополнительные сокеты, если эти программы помещаются в chroot-окружение из *rc.conf*.

Подробное протоколирование

При протоколировании в подробном режиме (-v) вместе с каждым сообщением, помещаемым в локальный протокол, выводятся числовые значения источника и уровня важности. Вывод названий источника и уровня вместо их числовых значений задает ключ -vv:

```
syslogd_flags="-vv"
```

[1] Сокет домена UNIX – это программный интерфейс, который время от времени используется во FreeBSD (например, фильтрами спама). К доменам Интернета или доменам сетей Windows эти сокеты никакого отношения не имеют. – *Прим. науч. ред.*

Это ключи, которые я постоянно использую. Полный список ключей можно найти на странице руководства syslog(8).

Управление файлами протоколов

Файлы протоколов растут, и вам следует решить, до какого размера можно позволить им расти. Стандартный способ – *ротация протоколов* (*log rotation*). При ротации протоколов самые старые протоколы удаляются, текущий файл протокола закрывается и ему присваивается другое имя, а для следующих данных создается новый файл протокола. В состав FreeBSD входит базовый обработчик протоколов newsyslog(8), способный также сжимать файлы, перезапускать демоны и выполнять все рутинные операции по ротации протоколов. cron(1) запускает newsyslog ежечасно.

При запуске newsyslog(8) считывает */etc/newsyslog.conf* и проверяет каждый файл протокола, указанный в конфигурационном файле. Если встретится условие ротации, то выполняется ротация протокола и другие надлежащие действия. В файле */etc/newsyslog.conf* каждому протоколу отводится одна строка с семью полями. Например:

```
/var/log/ppp.log       root:network    640     3     100  *     JC
```

Давайте рассмотрим все поля по очереди.

Путь к файлу протокола

Первое поле каждой строки содержит полный путь к обрабатываемому файлу протокола (в нашем примере – `/var/log/ppp.log`).

Владелец и группа

Во втором поле (в нашем примере – `root:network`) указаны владелец и группа обрабатываемого файла, разделенные двоеточием. Это необязательное поле, отсутствующее во многих стандартных записях.

newsyslog(8) может изменять владельца и группу в старых файлах протокола. По умолчанию файлами протокола владеет root, и они принадлежат группе wheel. Обычно владельца этих файлов не меняют, но к этой возможности имеет смысл прибегнуть на многопользовательских машинах.

Можно изменить только владельца или только группу. В данных случаях необходимо ставить двоеточие, даже если перед или после этого символа ничего нет. Например, значение `:www` изменит группу на `www`, а значение `mwlucas:` сделает меня владельцем файла.

Права доступа

Третье поле (`640` в нашем примере) – это права доступа в стандартном для UNIX представлении из трех цифр.

Количество

Следующее поле – количество (count) старых файлов протокола, которые должна хранить newsyslog(8). newsyslog(8) нумерует архивированные файлы протокола от новых к старым, начиная с 0 для самого свежего файла протокола. Например, при значении по умолчанию, равном 5 для */var/log/messages*, вы найдете следующие файлы протокола:

```
messages
messages.0.bz
messages.1.bz
messages.2.bz
messages.3.bz
messages.4.bz
messages.5.bz
```

Те из вас, кто умеет считать, обнаружат наличие шести, а не пяти резервных копий плюс текущий файл протокола! Как правило, избыток файлов протоколов лучше их нехватки. Если в системе крайне мало дискового пространства, то удаление одного-двух файлов поможет вам немного сэкономить время.

Размер

Пятое поле (в нашем примере – 100) – это размер файла в килобайтах. При запуске программа newsyslog(8) сравнивает размер, указанный в этом поле, с размером файла. Если размер файла больше указанного, файл подвергается ротации. Если размер файла не должен влиять на проведение ротации, поставьте здесь звездочку (*).

Время

Пока что все было просто, да? Что ж, шестое поле, время ротации, меняет все дело. В поле времени возможно четыре значения: звездочка (*), число и дата в двух различных форматах.

Если вы не хотите проводить ротацию протоколов на основании размера файла, поместите здесь звездочку (*).

Если поставить простое число, newsyslog(8) выполнит ротацию протоколов по истечении заданного количества часов. Например, если требуется проводить ротацию каждые 24 часа и неважно, когда именно она происходит, укажите в этом поле значение 24.

Формат даты чуть более сложен.

Формат времени ISO 8601

Любая запись, начинающаяся с символа @, – это время в ограниченном формате ISO 8601. Это стандартный формат, применяемый программой newsyslog(8) в большинстве UNIX-подобных систем. Первоначально данный формат применялся в базовой программе newsyslog(8),

разработанной в MIT. Правда, этот стандарт немного бестолков, но его поддерживают все UNIX-подобные операционные системы.

Полная дата в формате ISO 8601 – это 14 цифр с символом `T` в середине. Первые четыре цифры означают год, следующие две – месяц, следующие две – день месяца. Символ `T` после даты играет роль точки в десятичной дроби, отделяя целые дни от доли одного дня. Следующие две цифры – это часы, следующие две – минуты, следующие две – секунды. Например, дата 2 марта 2008 года и время 21 час 15 минут и 8 секунд в формате ISO 8601 будут представлены так: `20080302T211508`.

Если даты в ограниченном формате ISO 8601 приведены полностью, то все достаточно просто, однако при неполном указании даты возникает путаница. Например, если указать только поля рядом с `T`, а остальные оставить пустыми, то незаполненные поля будут расценены как групповые символы. Например, `1T` соответствует первому числу каждого месяца. Значение `4T00` соответствует полуночи четвертого числа каждого месяца. Значение `T23` соответствует 23 часам, или 11 часам вечера каждого дня. При указании `@T23` протоколы будут подвергаться ротации каждый день в 11 часов вечера.

Как и в случае cron(1), необходимо точно указывать часы. Например, дата, `@7T`, означает ротацию протоколов каждый час в седьмой день месяца. В конце концов, такая запись соответствует целому дню! Значение `@7T01` означает ротацию протоколов в 1 час ночи 7 числа каждого месяца, что выглядит более подобающим. Зато не требуется указывать минуты и секунды, так как ротация протоколов производится всего один раз в час.

Формат времени, специфичный для FreeBSD

В формате ISO 8601 нехорошо то, что он не позволяет легко назначить еженедельное задание или указать последний день месяца. Именно по этой причине в систему FreeBSD включена поддержка другого формата времени, позволяющего легко указывать эти вполне обычные даты. Любое значение, которому предшествует знак доллара (`$`), представляет время в формате *месяц неделя день*, специфичном для FreeBSD.

В этом формате применяются три идентификатора: `M` (день месяца), `W` (день недели) и `H` (час дня). За каждым идентификатором следует число, означающее конкретное время. Часы задаются в диапазоне от `0` до `23`, дни недели – от `0` (воскресенье) до `6` (суббота). Нумерация дней ме-

Ротация по размеру и по времени

Ротацию можно проводить в заданное время или по достижении определенного размера протоколов, или в том и другом случае. Если указать оба варианта, то протокол будет подвергнут ротации при выполнении любого из условий.

сяца начинается с 1 и доходит до последнего числа месяца, которое может быть представлено идентификатором L. Например, для ротации протоколов пятого числа каждого месяца в полдень надо указать $M5H12. Для запуска анализатора файлов протоколов в 22 часа последнего дня месяца применяйте запись $MLH22.

Флаги

Поле флагов определяет, какие специальные действия нужно выполнить при ротации протокола. Зачастую здесь указывается, как программе newsyslog(8) следует сжать файл протокола, но с помощью флагов можно также сигнализировать процессам о том, что началась ротация их протокола.

Формат файла протокола и сжатие

Протокол может быть либо текстовым, либо двоичным файлом.

Двоичные файлы можно записывать только специальным образом. Каждый новый файл протокола программа newsyslog(8) начинает с сообщения «logfile turned over» (файл протокола подвергнут ротации), но добавление такого текста в двоичный файл может повредить его. Флаг B сообщает программе newsyslog(8), что это двоичный файл и заголовок не следует записывать.

Однако многие файлы протоколов представляют собой простой текст ASCII, и программа newsyslog(8) может и должна вставлять сообщение в начало файла, указывая на то, что протокол был родвергнут ротации. Кроме того, сжатие старых протоколов позволит сохранить много свободного места на диске. Флаг J предписывает программе newsyslog(8) сжимать файлы протоколов с помощью bzip(1), а флаг Z – с помощью gzip.

Флаги специальной обработки протоколов

При ротации и создании файлов протоколов программа newsyslog(8) может выполнять какие-то особенные действия. Ниже описаны наиболее используемые, а об остальных вы узнаете на странице руководства newsyslog.conf(5).

Если нужно, чтобы определенные файлы протокола не попадали в резервную копию, флаг D позволит установить флаг NODUMP для вновь создаваемых файлов протокола.

Возможно, у вас имеется множество похожих друг на друга файлов протоколов, для которых необходимо реализовать одинаковый алгоритм обработки. Флаг G сообщает программе newsyslog(8), что имя файла протокола в начале строки – это на самом деле шаблон имени файла в формате командного интерпретатора и что выполнять ротацию всех файлов протокола, соответствующих этому выражению, следует одинаково. Регулярные выражения подробно описаны на странице руководства regexp(3). Перед чтением рекомендую принять аспирин.

Вы можете потребовать от newsyslog(8) создать файл, если он отсутствует. Для этого предназначен флаг C. Программа syslogd не будет производить протоколирование при отсутствии файла.

Наконец, если ни один из флагов не требуется, используйте дефис (-) как символ-заполнитель. Хотя вам, так или иначе, потребуется один из флагов *newsyslog.conf*, чтобы указать путь к pid-файлу.

Путь к pid-файлу

Следующее поле – путь к pid-файлу (в данном примере не показано, но его можно найти, заглянув в */etc/newsyslog.conf*). pid-файл – это простой способ записи идентификатора процесса (PID) той или иной программы, причем эту запись могут легко просматривать другие программы. Если вы приводите полный путь к pid-файлу, то при ротации протоколов newsyslog(8) пошлет программе сигнал kill -HUP. По этому сигналу процесс должен закрыть файл протокола и перезапустить себя. Не каждый процесс создает свой pid-файл и не все программы требуют выполнения особых действий во время ротации их протокола.

Сигнал

Большинство программ выполняют ротацию протокола по сигналу SIGHUP, но некоторым программам нужно отдельно сигнализировать о ротации их протокола. Для таких программ в последнем поле, после pid-файла, можно указать точный номер сигнала.

Пример записи в newsyslog.conf

Собрав все это воедино, рассмотрим худший случай – пример «вы-*наверное*-шутите». Предположим, у вас есть протокол базы данных, ротацию которого требуется проводить в 11 часов вечера последнего дня каждого месяца. В документации базы данных сказано, что для ротации файлов требуется послать программе сигнал прерывания (SIGINT, или сигнал с номером 2). Вам нужно, чтобы архивированными протоколами владел пользователь dbadmin. Просмотр протоколов разрешен только этому пользователю, а хранить протоколы надо шесть месяцев. Более того, протоколы представляют собой двоичные файлы. Строка файла *newsyslog.conf* будет такой:

```
/var/log/database      dbadmin:  600  6  *  $MLH23  B  /var/run/db.pid  2
```

Этот пример – исключение; в большинстве случаев достаточно указать имя файла и условия ротации.

FreeBSD и SNMP

Письма с отчетами полезны, однако в них содержится только общая информация, а протоколы, относящиеся к продолжительным периодам времени, трудно читать. Чтобы узнать больше о работе сервера сей-

час и в долгосрочной перспективе, применяйте индустриальный стандарт сбора информации о хостах сети – *простой протокол управления сетью* (*Simple Network Management Protocol, SNMP*). Большинство поставщиков поддерживают его как протокол сбора информации со многих сетевых устройств. FreeBSD включает агент SNMP bsnmpd(8), который не только предоставляет стандартные функции SNMP, но и обеспечивает доступ к особенностям, характерным для FreeBSD.

bsnmpd (сокращенно от *Begemot SNMPD*) – это минимальный агент SNMP, предусматривающий возможность расширения. В действительности все многообразие функциональных возможностей обеспечивается с помощью внешних модулей. Операционная система FreeBSD включает в себя модули bsnmpd для обеспечения стандартных сетевых функций и функциональных особенностей, присущих FreeBSD, таких как PF и netgrapf(4). bsnmpd – это не универсальный инструмент на все случаи жизни, скорее – это фундамент, на котором любой сможет построить свою реализацию SNMP, наилучшим образом отвечающую его требованиям, но не больше того.

Основы SNMP

SNMP работает по стандартной модели клиент/сервер. Клиент (или *агент*) SNMP посылает по сети запрос серверу SNMP. Сервер SNMP, bsnmpd, собирает информацию в локальной системе и возвращает ее клиенту.

Агент SNMP может также посылать серверу SNMP запрос на выполнение изменений. Если система сконфигурирована надлежащим (или ненадлежащим, в зависимости от вашей точки зрения) образом, вы можете выдавать команды через SNMP. Такая конфигурация с помощью «записи» чаще всего применяется в маршрутизаторах, коммутаторах и других сетевых устройствах. Большинство UNIX-подобных операционных систем имеют систему управления командной строки и обычно не принимают команды через SNMP. Создание конфигурации системы или запуск команд через SNMP требует тщательной настройки и внимания к безопасности, и это отличная тема для целой книги. Ни один системный администратор из тех, кого я знаю, не использует SNMP для управления своими системами. Помня об этом, мы рассмотрим доступ к SNMP исключительно для чтения.

В дополнение к возможности сервера SNMP отвечать на запросы клиента SNMP клиент может передавать *ловушки* (*traps*) SNMP получателю ловушек, расположенному где-то в сети. Агент SNMP генерирует эти ловушки в ответ на определенные события сервера. Ловушки SNMP во многом напоминают сообщения syslogd(8), за исключением того, что они имеют весьма специфический формат, необходимый для работы SNMP. В настоящее время в операционной системе FreeBSD нет получателя ловушек SNMP, но если он вам потребуется, обратите внимание на snmptrapd(8) из катеогрии net-snmp (*/usr/ports/net-mgmt/net-snmp*).

SNMP MIB

SNMP управляет информацией через *базу управляющей информации* (*Management Information Base, MIB*) – древовидную структуру, содержащую иерархическую информацию в формате ASN.1. Пример дерева параметров MIB был представлен в главе 5, когда рассматривался интерфейс sysctl(8).

У каждого сервера SNMP есть список информации, которую он может извлечь из локального компьютера. Сервер представляет эту информацию в виде иерархического дерева. В каждом дереве SNMP MIB есть общие основные категории (сеть, физическая система, программы и т. д.), разбитые на более специфичные подразделы. Такие деревья представляют собой хорошо организованные каталоги, подкаталоги которых содержат более узкую информацию. Аналогично, MIB верхнего уровня содержат множество MIB более низких уровней.

На MIB ссылаются по имени и по номеру. Например, так выглядит параметр MIB одной из систем:

```
interfaces.ifTable.ifEntry.ifDescr.1 = STRING: "fxp0"
```

Первый элемент этого параметра MIB `interfaces` показывает, что запись относится к сетевым интерфейсам системы. Если бы в данной системе не было интерфейсов, первая категория даже не существовала бы. Поле `ifTable` – это таблица интерфейсов, или список всех интерфейсов системы. Поле `ifEntry` показывает один конкретный интерфейс, а `ifDescr` – описание этого интерфейса. Данная запись MIB означает: «Интерфейс с номером 1 на этой машине называется fxp0».

Записи MIB можно также выражать в виде номеров. Большинство инструментов SNMP отдают предпочтение числовым значениям MIB. Большинство из нас предпочтет пользоваться словами, но все же наши хилые мозги должны уметь работать и с тем и с другим. Броузер MIB может переводить записи из одного представления в другое, но вы можете также установить */usr/ports/net-mgmt/net-snmp* и использовать snmptranslate(1), а пока верьте мне на слово. Предыдущий пример можно преобразовать в такую форму:

```
.1.3.6.1.2.1.2.2.1.2.1
```

Если выразить этот параметр MIB словами, то в нем будет пять элементов, разделенных точками. Параметр MIB, выраженный в числах, состоит из 11 компонентов. Есть подозрение, что здесь не все в порядке, потому что обе записи должны означать одно и то же. В чем тут дело?

Числовое представление параметра MIB длиннее, поскольку в него входит часть по умолчанию – `.1.3.6.1.2.1`. Она означает *.iso.org.dod.internet.mgmt.mib-2*. Это стандартное подмножество MIB, применяемое в Интернете. В названии почти каждого (но не во всех) параметра MIB есть префиксная строка – вот почему никто ее больше не приводит.

Если есть настроение почудить, можно даже смешать слова и числа:

```
.1.org.6.1.mgmt.1.interfaces.ifTable.1.2.1
```

В подобном случае международное право позволяет вашим сотрудникам изгонять вас как нечистую силу, с помощью вил и горящих факелов. Выберите какой-то один способ представления параметров MIB и придерживайтесь его.

Определения MIB и броузеры MIB

Определения параметров MIB следуют очень строгому синтаксису, описанному в *файлах MIB*. У каждого агента SNMP есть собственные файлы MIB – bsnmpd держит их в каталоге */usr/share/snmp*. Это обычные текстовые файлы. Вы можете читать и интерпретировать их, не пользуясь ничем, кроме своей головы. Но я настоятельно рекомендую скопировать их на свою рабочую станцию и установить броузер MIB, чтобы упростить их изучение.

Броузеры MIB интерпретируют файлы MIB, представляя их в виде древовидной структуры с полным определением каждой части и описанием каждого отдельного параметра MIB. Вообще говоря, броузер позволяет вводить имена конкретных параметров MIB и отображает как числовое, так и словесное представление этого параметра, а также запрос агента SNMP для определения состояния этого параметра MIB.

На рабочей станции с установленной операционной системой FreeBSD (или хотя бы UNIX-подобной системой) можно использовать броузер MIB – mbrowse (*/usr/ports/net-mgmt/mbrowse*). Для систем Windows есть множество броузеров MIB, в том числе вполне приличный и бесплатный Getif (*http://www.wtcs.org*). Если вам для работы с SNMP не нужны программы с графическим интерфейсом, загляните в категорию net-snmp (*/usr/ports/net-mgmt/net-snmp*), где вы найдете богатый ассортимент инструментов командной строки, предназначенных для работы с SNMP.

Безопасность и SNMP

Многие эксперты по компьютерной безопасности утверждают, что аббревиатура SNMP происходит от «Security: Not My Problem!» (безопасность – это не моя проблема). Не слишком доброжелательно, зато точно. В общедоступных сетях SNMP следует использовать только за брандмауэром. Если вы вынуждены использовать SNMP в незащищенном Интернете, применяйте механизм фильтрации пакетов, препятствующий получению посторонних запросов к вашему сервису SNMP. Агент SNMP работает с портом UDP 161.

Базовый уровень безопасности в SNMP обеспечивается через *сообщества*. Оглянитесь кругом – и вы без труда найдете массу разнообразных объяснений, почему сообщества – это не то же самое, что пароли, но *фактически* сообщество – это пароль. У большинства агентов SNMP

есть два пароля по умолчанию: открытый (доступ только для чтения) и закрытый (доступ для чтения и для записи). Да, есть пароль по умолчанию, который обеспечивает доступ для чтения и для записи. Всякий раз, когда вы настраиваете агент SNMP, в любой ОС, на любом хосте ваша первая задача – запретить имена сообществ по умолчанию и заменить их другими, которые не приводились в документации в последние несколько десятилетий.

Реализация протокола SNMP поставляется в разных версиях. Версия 1 была первой попыткой, версия 2c представляет современный стандарт. Вам могут встретиться упоминания о версии 3, в которой применяются усовершенствованные механизмы шифрования для защиты данных, передаваемых в сети. Однако лишь немногие производители в действительности реализовали SNMPv3. В bsnmpd(8) системы FreeBSD используется SNMPv2c. Это означает, что любой желающий, вооружившись анализатором пакетов, сможет перехватить имя сообщества SNMP, поэтому ни у кого не вызывает сомнений необходимость использования SNMP исключительно в закрытых сетях. Посылка запросов SNMP через общедоступные сети – отличный способ привлечь незнакомцев, пытающихся запустить руку в вашу систему управления.

Конфигурирование bsnmpd

Прежде чем использовать SNMP для мониторинга системы, необходимо настроить демон SNMP. Конфигурация bsnmpd(8) находится в файле */etc/snmpd.config*. Помимо открытого и закрытого сообществ по умолчанию конфигурация по умолчанию не включает ни одну из особенностей FreeBSD, придающих bsnmpd(8) привлекательность.

Переменные bsnmpd

В конфигурации bsnmpd используются переменные, которым присваиваются значения с помощью операторов. Большая часть определений переменных находится в начале конфигурационного файла, как показано здесь:

```
location := "Room 200"
contact := "sysmeister@example.com"
system := 1     # FreeBSD
traphost := localhost
trapport := 162
```

Эти переменны определяют значения параметров MIB, которые должны быть установлены в любом агенте SNMP. Переменная `location` описывает физическое местоположение машины. Для каждой машины должен быть указан действительный адрес электронной почты (`contact`). Демон bsnmpd(8) может работать и в других операционных системах, отличных от FreeBSD, поэтому здесь присутствует переменная (`system`), определяющая тип операционной системы. Последние переменные определяют имя хоста ловушки и его номер порта.

Далее в файле можно определить имена сообществ SNMP:

```
# Измените эти имена!
read := "public"
# Раскомментируйте строку begemotSnmpdCommunityString.0.2 ниже,
# чтобы определить имя сообщества для обеспечения доступа на запись.
write := "geheim"
trap := "mytrap"
```

Строка `read` определяет имя сообщества данного агента SNMP, разрешающего доступ только для чтения. Конфигурационный файл по умолчанию предлагает изменить его. Не пренебрегайте этим советом. Строка `write` — это имя сообщества с правом доступа для чтения и записи, которое далее в конфигурационном файле запрещено по умолчанию. Вы можете также определить имя сообщества для передачи ловушек SNMP этим агентом.

С этой конфигурацией уже можно запустить bsnmpd(8), обеспечив передачу основных сведений SNMP к вашей сетевой системе управления. Чтобы запускать bsnmpd во время загрузки, просто добавьте в файл */etc/rc.conf* строку `bsnmpd_enable="YES"`. Однако при этом вы не сможете получить дополнительную функциональность, предоставляемую FreeBSD. Давайте посмотрим, как ими управлять.

Детализированная конфигурация bsnmpd

Демон bsnmpd(8) использует значения переменных, которые вы определите в начале файла, для присвоения значений различным параметрам MIB, расположенным в конфигурационном файле ниже. Например, в начале файла были определены переменные `read` и `public`. Ниже в конфигурационном файле вы обнаружите такую инструкцию:

```
begemotSnmpdCommunityString.0.1 = $(read)
```

Она устанавливает значение параметра MIB `begemotSnmpdCommunityString.0.1` равным значению переменной `read`.

Почему бы просто не установить эти значения напрямую? bsnmpd(8) задумывался как расширяемый и легко настраиваемый инструмент. Установить несколько переменных в начале файла гораздо проще, чем редактировать правила, расположенные в файле ниже.

Давайте вернемся к параметру MIB `begemotSnmpdCommunityString`, значение которого здесь устанавливается. Для чего он нам нужен? Поискав эту строку в броузере MIB, вы увидите, что этот параметр MIB определяет имя сообщества SNMP. Возможно, вы допускали это исходя из того, что ему присваивается значение переменной `read`, но теперь сомнений не осталось.

Точно так же вы могли бы отыскать следующую запись:

```
begemotSnmpdPortStatus.0.0.0.0.161 = 1
```

Поиск в броузере MIB показывает, что это выделенный IP-адрес и порт UDP, к которым привязывается bsnmpd(8) (в данном случае – все имеющиеся в системе IP-адреса и порт 161). Все конфигурационные параметры MIB определяются подобным образом.

Загрузка модулей bsnmpd

Большинство наиболее интересных функциональных возможностей bsnmpd настраивается с помощью модулей. Включите модули в конфигурационный файл, имя которого определяется параметром MIB `begemotSnmpdModulePath`, укажите класс модуля и полный путь к разделяемой библиотеке, которая содержит реализацию этой особенности. Например, в конфигурационном файле можно увидеть закомментированную строку, описывающую модуль PF для bsnmpd(8):

```
begemotSnmpdModulePath."pf" = "/usr/lib/snmp_pf.so"
```

Эта запись включает поддержку параметров MIB для PF. Благодаря этому модулю программное обеспечение системы управления сможет увидеть все, что происходит в PF, когда это будет разрешено, начиная от сброшенных пакетов и заканчивая размером таблицы состояний.

К моменту написания этих строк вместе с bsnmpd(8) для FreeBSD распространялись следующие модули, ориентированные на FreeBSD. Все они выключены по умолчанию, но вы можете включить их, раскомментировав соответствующие строки в конфигурационном файле.

`Netgraph`

Предоставляет доступ ко всем сетевым особенностям Netgraph, описанным в snmp_netgraph(3).

`PF`

Предоставляет доступ к пакетному фильтру PF.

`Hostres`

Реализует параметр MIB – Host Resources SNMP, snmp_hostres(3).

`bridge`

Предоставляет доступ к функциям сетевого моста, описанным в snmp_bridge(3).

Включив необходимые модули в конфигурационном файле, перезапустите bsnmpd(8). Если программа не запустилась, проверьте содержимое */var/log/messages* на наличие ошибок.

В комплексе с bsnmpd(8), syslogd(8), письмами о состоянии и разнообразными инструментами анализа производительности вы сможете превратить систему FreeBSD в хорошо управляемое сетевое устройство. Теперь, когда вы познакомились со всем, что может предложить система, захватите фонарь, так как далее мы собираемся заглянуть в самые темные уголки FreeBSD.

20

Передний край FreeBSD

Проведя в сообществе FreeBSD некоторое время, вы услышите о самых разных вещах, которые можно было бы сделать, если бы знать – как. Кто-то производит встроенные устройства с FreeBSD и продает их по всему свету, при этом покупатели даже не подозревают, что внутри небольшой коробки, управляющей кондиционером или радиоретрансляционной станцией, находится UNIX-подобная операционная система. Кто-то запускает FreeBSD на машинах без жестких дисков, поддерживая сотни и даже тысячи бездисковых станций с одного единственного сервера. Вы без труда найдете USB-устройства и самозагружаемые компакт-диски с полноценной системой FreeBSD, включая установленное программное обеспечение, какое только можно пожелать. Все это вовсе не сложно, если знать некоторые секреты.

В этой главе мы исследуем передний край FreeBSD – настоящую передовую, где пользователи воплощают свои уникальные идеи, не всегда встречающие поддержку в основном Проекте FreeBSD. Иногда можно получить поддержку по этим вопросам по обычным каналам, но вы должны быть готовы устранять неисправности и отлаживать все, о чем будет рассказано в этой главе.

/etc/ttys

Файл */etc/ttys* управляет тем, как и откуда пользователи могут входить в систему FreeBSD. При этом вы можете устанавливать ограничения для каждой отдельной учетной записи где-нибудь в другом месте, а в файле */etc/ttys* перечислить устройства, с которых можно выполнять вход в систему, и указать, как эти устройства должны использоваться. Большинство пользователей могут входить только с консоли и по соединению SSH, но вы можете настроить и другие устройства, с которых можно будет выполнять вход.

Не забывайте: *tty* – это сокращение от *teletype* (телетайп). Когда-то телетайп был основным интерфейсом вывода информации для UNIX-подобных систем. Любое устройство, с которого можно выполнить вход, считается телетайпом. Система FreeBSD поддерживает самые разные типы терминальных устройств, от старомодных последовательных терминалов до современных консолей с клавиатурой, мышью и устройством отображения, а также сессии SSH и telnet. Системы FreeBSD предлагают четыре стандартных устройства входа в систему, или *терминалы*: консоль, виртуальные терминалы, коммутируемые терминалы и псевдотерминалы.

Консоль – единственное устройство, доступное в однопользовательском режиме. В большинстве систем FreeBSD это либо видеоконсоль, которая состоит из монитора и клавиатуры, либо последовательная консоль, доступная из другой системы. Как только система попадает в многопользовательский режим, консоль обычно подключается к виртуальному терминалу. Устройство консоли – */dev/console*.

Виртуальный терминал подключен к физическому монитору и клавиатуре. Если вы не используете систему X Window, то можете организовать несколько терминалов на одном физическом терминале. Переключение между ними осуществляется нажатием клавиши Alt и одной из функциональных клавиш. В следующий раз, когда вы окажетесь за консолью, нажмите комбинацию клавиш Alt-F2. Вы должны увидеть новый экран входа в систему со строкой `ttyv1` после имени хоста. Это второй виртуальный терминал. Нажатие комбинации Alt-F1 вернет вас обратно, в основной виртуальный терминал. По умолчанию в системе FreeBSD восемь виртуальных терминалов, а девятый зарезервирован для X Window. Вы можете использовать восемь виртуальных текстовых терминалов, даже работая в X, если некоторые окружения рабочего стола предоставляют несколько виртуальных терминалов. Виртуальным терминалам соответствуют устройства */dev/ttyv*.

Коммутируемый терминал подключается к последовательной линии. Вы можете подключать модемы непосредственно к последовательным портам и позволять пользователям подключаться к своему серверу. В наши дни эта возможность используется редко, но аналогичная функциональность служит для поддержки входа в систему через последовательную консоль. Коммутируемым терминалам соответствуют устройства */dev/ttyd*.

Наконец, *псевдотерминалы* – это полностью программная реализация. Для подключения к серверу через SSH вам не требуется специальное аппаратное обеспечение, а программному обеспечению по-прежнему необходим файл устройства для обеспечения сеанса входа. Псевдотерминалам соответствуют устройства */dev/ttyp*.

Настройки доступа к терминальным устройствам находятся в файле */etc/ttys*. Большинство терминальных устройств не требуют настройки – в конце концов, вы получили в свое распоряжение псевдотерминал

лишь потому, что были аутентифицированы и авторизованы средствами SSH, telnet или какого-то другого сетевого протокола. В этом файле можно определять консоль для входа в систему и разрешать вход через последовательную консоль.

Формат /etc/ttys

Типичная запись в файле */etc/ttys* выглядит примерно так:

```
ttyv0   "/usr/libexec/getty Pc"         cons25  on  secure
```

Первое поле – это устройство консоли, в данном случае – это первый виртуальный терминал в системе, `ttyv0`.

Второе поле – это программа, которая запускается для обслуживания процедуры входа с этого терминала. В системе FreeBSD вход с любого устройства, за исключением псевдотерминала, обслуживает программа getty(8). При получении запроса на вход псевдотерминалом вход в систему обслуживается каким-либо демоном.

Третье поле – тип терминала. Возможно, вам приходилось слышать о терминале vt100 или даже о терминале Sun. В системе FreeBSD для монитора используется тип `cons25`, которому соответствует экран размером 25×80 символов. Для псевдотерминалов используется тип `network`; их особенности определяются демоном и клиентским программным обеспечением.

Четвертое поле определяет, доступен терминал для входа в систему или нет. Допустимы значения `on` – вход разрешен и `off` – вход запрещен. Псевдотерминалы активируются по запросу.

В этом примере есть еще ключевое слово `secure`, которое сообщает программе getty(8), что через эту консоль в систему может входить пользователь root.

Предоставление терминалов – это низкоуровневая задача, которую решает непосредственно процесс init(8). Изменения в */etc/ttys* не вступят в силу, пока процесс init(8) не будет перезапущен. Идентификатор этого процесса всегда равен 1.

```
# kill -1 1
```

Небезопасная консоль

При загрузке в однопользовательском режиме вы получаете доступ к командной строке с привилегиями root. Это годится для ноутбука или для сервера в корпоративном вычислительном центре, а как быть с машинами, расположенными в местах, не вызывающих доверия? Если сервер установлен в объединенном информационном центре, вы едва ли захотите, чтобы кто-то мог получить доступ к машине с привилегиями root. В таком случае можно сообщить системе FreeBSD, что физическая консоль небезопасна и при входе с нее в однопользовательский режим необходимо требовать ввод пароля root. Система будет вы-

полнять загрузку от включения питания до многопользовательского режима, не требуя ввести пароль, но потребует его, когда будет произведена попытка загрузиться в однопользовательском режиме.

Требование ввода пароля в однопользовательском режиме не защитит данные полностью, но существенно поднимет планку, которую нужно будет преодолеть. Отдельные сотрудники центра, работающие поздно ночью, когда их никто не видит, могли бы загрузить систему в однопользовательский режим и добавить учетную запись для себя всего за 15 минут или что-то около того. Разборка вашей машины, извлечение жестких дисков, монтирование их в другой машине, внесение изменений и возврат вашего сервера в работу отнимут гораздо больше времени, труда и, скорее всего, не пройдут незамеченными для сотрудников управления центром.

Найдите запись в файле */etc/ttys*, соответствующую консоли:

```
console none            unknown off secure
```

Как видите, консоль – это не полноценный терминал, для нее не запускается программа getty(8) и используется универсальный тип терминала `unknown`. Консоль предназначена исключительно для работы в однопользовательском режиме или когда она подключена к другому терминалу, так что тут все в порядке.

Чтобы консоль запрашивала пароль root при загрузке в однопользовательском режиме, измените слово `secure` на `insecure`.

Вход по последовательной линии

В главе 3 мы говорили о последовательных консолях. Помимо консоли, FreeBSD предоставляет возможность входа в систему через последовательный порт. Во многих UNIX-подобных операционных системах 1970–1980-х это был единственный способ получить приглашение к входу. Сегодня он практически забыт благодаря вездесущей Сети, тем не менее, возможность входа в систему по последовательной линии очень полезна в системах, где уже имеется последовательная консоль в системах, не поддерживающих видеоконсоль. Системы без стандартного дисплея и клавиатуры называются *автономными системами* (*headless systems*).

Чтобы разрешить вход в систему по последовательной линии, отыщите в файле */etc/ttys* запись, соответствующую последовательному порту. По умолчанию в конфигурации перечислены четыре коммутируемых терминала `ttyd0-ttyd3`. Эти записи соответствуют последовательным портам sio0–sio3, или COM1–COM4. Предположим, что последовательная консоль подключена к первому последовательному порту.

```
ttyd0 "/usr/libexec/getty std.9600" dialup ❶off secure
```

Настройка выполняется довольно просто. Порт отключен ❶. Замените слово `off` на `on`.

Последовательные линии медленнее, чем сети. Скорость передачи данных ограничена скоростью работы последовательного порта. Это обстоятельство может превратиться в проблему, если последовательная консоль подключена к последовательному порту. Во время сеанса работы на консоль выводятся отладочные сообщения, что занимает некоторую часть полосы пропускания. Если в ходе сеанса выводится слишком много протоколируемых сообщений, это может препятствовать нормальной работе. Но даже в этом случае возможность входа в систему по последовательной линии может оказаться неоценимой для автономных серверов и встроенных устройств, в чем мы убедимся ниже в этой главе.

FreeBSD на бездисковых станциях

С одной системой FreeBSD справиться совсем не сложно, но десятки и сотни практически идентичных систем могут стать тяжким бременем. Один из способов сделать сопровождение менее хлопотным заключается в использовании *бездисковых* систем. Бездисковые системы совсем необязательно не должны иметь дисков, просто они загружают операционную систему с сервера NFS, расположенного в сети.

Зачем нужны бездисковые системы на ферме серверов? С единственного сервера NFS можно загружать множество систем, что позволяет централизовать наложение заплат и уменьшить объем работ по поддержке. Это решение прекрасно подходит для групп терминалов, вычислительных кластеров и других ситуаций, когда у вас много идентичных систем. При таком подходе обновление операционной системы превращается в простую замену файлов на сервере NFS. Аналогично, когда в обновленной версии обнаруживаются ошибки, возврат к предыдущей версии заключается в не менее простом восстановлении файлов на сервере NFS. Единственное, что потребуется сделать на стороне клиента, – это перезагрузить систему. Так как клиенты имеют доступ к серверу только для чтения, никто из пользователей не сможет внести изменения в свои локальные системы так, чтобы они остались после перезагрузки. Если у вас всего пара локальных систем, выгоды от организации бездисковых систем не будут стоить потраченных усилий, но чем больше локальных систем, тем отчетливее видны преимущества.

Тест, тест, тест!

Ваша первая попытка организовать бездисковую систему во многом будет напоминать первую настройку брандмауэра: масса ошибок и неприятностей, доводящих до бешенства. Я настоятельно рекомендую тщательно тестировать каждый этап подготовки, что облегчит вам поиск и устранение проблем. Для каждого необходимого сервиса будут предложены инструкции по тестированию.

Для организации бездисковых систем, необходимы сервер NFS, сервер DHCP, сервер TFTP и оборудование, позволяющее выполнять загрузку по сети. Обратимся к каждой из этих составляющих и посмотрим, как они настраиваются.

Бездисковые клиенты

Машины, работающие как бездисковые станции, должны уметь отыскать в сети начальный загрузчик и операционную систему. Реализовать это можно с применением BOOTP и PXE. *BOOTP*, Internet Bootstrap Protocol (протокол начальной загрузки), – это старый, давно забытый стандарт. *PXE*, Intel's Preboot Execution Environment (предварительная загрузка среды выполнения), уже в течение нескольких лет поддерживается всеми современными машинами, поэтому мы обратим свое внимание именно на этот способ.

Включите бездисковую станцию и зайдите в настройки BIOS. Где-то среди этих настроек должны находиться параметры, определяющие порядок выбора загрузочных устройств. Если машина поддерживает способ загрузки, основанный на PXE, одним из таких устройств будет сеть. Включите этот параметр и установите его первым в списке.

На этом подготовка бездискового клиента закончена. Теперь перейдем к подготовке сервера.

Настройка сервера DHCP

Многие считают DHCP лишь способом получения IP-адреса, но этот протокол способен на большее. Можно настроить сервер DHCP таким образом, что он будет сообщать о местоположении сервера TFTP, NFS и других сетевых ресурсов. Бездисковые системы широко используют DHCP, и вы обнаружите, что мы задействуем некоторые возможности DHCP, не рассмотренные ранее. В настройке сервера DHCP, предназначенного для обслуживания бездисковых систем, нет ничего сложного, если вам заранее известны MAC-адреса бездисковых станций.

MAC-адрес

Чтобы назначить конфигурационную информацию клиенту DHCP, необходимо знать MAC-адрес сетевой карты клиента. Некоторые реализации BIOS предоставляют информацию о MAC-адресах интегрированных сетевых карт, а на некоторых серверах даже наклеены ярлыки с MAC-адресами. Но это слишком просто, а мы попробуем пройти более сложным путем.

Пытаясь загрузиться через сеть, машина выполняет DHCP-запрос на получение конфигурационной информации. У вас пока нет конфигурации для бездисковых станций, тем не менее, любой сервер DHCP регистрирует MAC-адреса клиентов. Например, когда я попытался загрузить бездисковую станцию, в протоколе сервера DHCP появились следующие записи:

```
Jul 27 10:15:49 sardines dhcpd: DHCPDISCOVER from 00:00:24:c1:cb:a4 via fxp0
Jul 27 10:15:49 sardines dhcpd: DHCPOFFER on 192.168.1.78 to
00:00:24:c1:cb:a4 via fxp0
```

MAC-адрес этого клиента – 00:00:24:c1:cb:a4, и ему был предложен IP-адрес 192.168.1.78. Владея этой информацией, мы можем создать конфигурацию DHCP, чтобы присвоить этому хосту статический IP-адрес и предоставить загрузочную информацию.

Конфигурация DHCP: определенная бездисковая станция

Основные службы DHCP мы настроили в главе 15. В следующем примере приводится конфигурация dhcpd(8) для бездискового клиента. Эти параметры указываются уже не в инструкции subnet, а в отдельной инструкции верхнего уровня.

```
❶ group diskless {
❷         next-server 192.168.1.1;
❸         filename "pxeboot";
❹         option root-path "192.168.1.1:/var/diskless/1/";
❺         host diskless1.blackhelicopters.org {
❻                 hardware ethernet 00:00:24:c1:cb:a4 ;
❼                 fixed-address 192.168.1.99 ;
        }
}
```

Здесь мы определили группу с именем `diskless` ❶. Это позволило нам задать значения параметров для целой группы и добавить в нее хосты. Каждый хост в группе будет получать одни и те же значения параметров.

Параметр `next-server` ❷ сообщает клиентам DHCP IP-адрес сервера TFTP, а параметр `filename` ❸ – имя файла с загрузчиком, который находится на сервере TFTP. В главе 3 говорилось, что загрузчик – это программа, которая отыскивает и загружает ядро. Наконец, параметр `option root-path` ❹ сообщает загрузчику, где находится корневой каталог для данной машины. Все эти параметры и настройки присваиваются всем клиентам в группе.

Затем выполняется привязка бездискового клиента к группе с помощью инструкции `host` с именем хоста ❺. Наш первый клиент называется `diskless1`. Данный клиент идентифицируется по MAC-адресу ❻ и ему присваивается статический IP-адрес ❼. Кроме того, ему передается стандартная конфигурация группы.

Аналогичным образом можно добавить все остальные хосты в сети.

Перезапустите dhcpd(8), чтобы конфигурация вступила в силу. Теперь перезагрузите бездисковую станцию. В протоколе DHCP должны появиться записи, свидетельствующие о том, что этому клиенту предложен его статический IP-адрес. Однако клиент DHCP не может продолжить загрузку без загрузчика, а это означает, что нам следует настроить сервер TFTP.

Конфигурация DHCP: ферма бездисковых станций

Допустим, у вас есть много идентичных бездисковых станций, таких как «тонкие клиенты». Тогда вполне разумно отказаться от необходимости заводить статические записи в конфигурации DHCP для каждого из них. Позволим этим хостам получать загрузочную информацию с сервера DHCP без указания статическиех IP-адресов. В этом случае клиенты будут получать адреса из пула DHCP.

Кроме того, можно выполнять идентификацию хостов, запрашивающих информацию DHCP из PXE, и присваивать им определенные группы адресов. Хост, загружающийся через PXE, идентифицирует себя на сервере DHCP как клиент типа `PXEclient`. Можно написать правила для клиентов данного типа и конфигурировать их соответствующим образом. Загляните в руководство DHCP и поищите информацию о параметрах `vendor-class-identifier` и `dhcp-client-identifier`.

tftpd и загрузчик

Настройка сервера TFTP рассматривалась в главе 15. Сервер TFTP должен иметь файл *pxeboot* для бездисковых клиентов. FreeBSD устанавливает *pxeboot* в каталог */boot*.

```
# cp /boot/pxeboot /tftpboot
# chmod +r /tftpboot/pxeboot
```

Попробуйте на своей рабочей станции загрузить *pxeboot* через TFTP. Если файл загрузился успешно, попробуйте перезагрузить бездисковую станцию и посмотрите, что из этого получится. На консоли должны появиться примерно такие сообщения:

```
Building the boot loader arguments
Relocating the loader and the BTX
Starting the BTX loader
```

Вы видели эти сообщения раньше, когда загружали FreeBSD с жесткого диска. На этом месте загрузка бездискового клиента должна остановиться, потому что он не может смонтировать пространство пользователя (userland), которого еще не существует. Это задача сервера NFS.

Сервер NFS и пространство пользователя для бездискового клиента

Многие руководства по организации бездисковых станций предлагают использовать корневую файловую систему сервера со всеми программами для обеспечения работы бездисковых клиентов. Сделать это довольно просто, но это крайне небезопасно. На сервере, обслуживающем бездисковые станции, скорее всего, есть программное обеспечение, которое не должно попасть в руки клиентов, и наверняка есть секретная информация, доступ к которой нельзя разрешать для целой группы рабочих станций. Более мудрым будет создать отдельную файловую систему со всем необходимым.

Реализовать отдельную файловую систему можно множеством способов, но, на мой взгляд, наиболее просто будет использовать немного измененную конструкцию jail(8) из главы 9. Сначала нужно создать каталог для бездисковых клиентов, а затем установить в этот каталог ядро и все необходимые программы и библиотеки. Ниже приводится пример установки программного обеспечения в каталог */var/diskless/1*:

```
# mkdir -p /var/diskless/1
# cd /usr/src
# make installworld DESTDIR=/var/diskless/1
# make installkernel DESTDIR=/var/diskless/1
# make distribution DESTDIR=/var/diskless/1
```

Эти команды установят в каталог */var/diskless/1* все необходимое для пользователя программное обеспечение.

Теперь сообщим серверу NFS об этом каталоге. Я предполагаю развернуть в этой сети несколько бездисковых систем, поэтому экспортирую этот каталог через NFS сразу целой подсети. Клиентам не нужен доступ на запись к корневому каталогу NFS, поэтому экспорт выполняется только для чтения. Делается это с помощью следующей строки в файле */etc/exports*:

```
/var/diskless/1 -ro -maproot=0 -alldirs -network 192.168.1.0 -mask
255.255.255.0
```

Перезапустите mountd(8) с помощью сценария */etc/rc.d/mountd*, чтобы сделать доступным этот разделяемый ресурс, и попробуйте смонтировать его со своей рабочей станции. Проверьте, содержит ли каталог все необходимые программы и убедитесь, что у вас отсутствует право на запись в эту файловую систему. После этого перезагрузите бездисковую станцию и посмотрите, что произойдет. Клиент должен отыскать ядро и загрузиться в ненастроенный многопользовательский режим. В зависимости от производительности сервера, клиента и пропускной способности сети это может занять некоторое время. Мой ноутбук из

Повышение безопасности NFS для бездисковых клиентов

Как только ферма бездисковых станций заработает, вернитесь на шаг назад и назначьте учетную запись root для NFS другому пользователю. Запустите команду `find /var/diskless/1 -user 0 -exec chown nfsroot`, которая отыщет все файлы, принадлежащие root, и определит им в качестве владельца пользователя nfsroot. После этого можно отредактировать файл */etc/exports*, чтобы отобразить пользователя root в пользователя nfsroot. Вы пока только учитесь, поэтому не слишком увлекайтесь.

сети загружается так же быстро, как с жесткого диска, тогда как устройство 266 MHz Soekris net4801 загружается несколько минут.

С этого момента можно дополнительно сконфигурировать набор приложений и библиотек, чтобы он точнее соответствовал потребностям вашего единственного клиента. Можно внести изменения в */etc*, например, отредактировать */etc/fstab* или скопировать файл паролей на место. Для единственного клиента этого было бы вполне достаточно, но инфраструктура FreeBSD проектировалась специально для поддержки десятков и даже сотен клиентов с одной и той же файловой системы. Посмотрим, как этого можно добиться.

Конфигурирование фермы бездисковых станций

Одно из преимуществ бездисковых систем состоит в том, что несколько машин могут разделять одну файловую систему. Однако даже для практически идентичных машин могут понадобиться немного отличающиеся конфигурационные файлы. У системы FreeBSD есть механизм, позволяющий организовать передачу персонализированных конфигурационных файлов поверх универсальной файловой системы за счет повторного монтирования разделов файловой системы в памяти (Memory Filesystem, MFS) и копирования в них нестандартных конфигурационных файлов.

Заданные по умолчанию настройки FreeBSD для бездисковых станций позволяют настраивать бездисковые системы в разных сетях и подсетях, что особенно ценно при работе в больших сетях. Если у вас всего несколько бездисковых систем, поначалу настройка их может показаться немного громоздкой. Однако некоторое время спустя вы обнаружите, что применяете их все шире и шире. Бездисковые системы – это удобное решение множества проблем.

Во время загрузки FreeBSD выясняет с помощью параметра sysctl `vfs.nfs.diskless_valid` – не запущена ли она как бездисковая система. Если этот параметр имеет значение `0`, значит, система загружена с жесткого диска, в противном случае – это бездисковая система. В бездисковых конфигурациях FreeBSD запускает сценарий */etc/rc.initdiskless*.

Клиент NFS хранит бездисковую конфигурацию в каталоге */conf*, то есть в каталоге *conf*, размещенном в корне файловой системы NFS. Мы называем этот каталог */conf*, хотя на самом деле в файловой системе сервера это будет каталог */var/diskless/1/conf*.

Вы должны создать несколько подкаталогов: как минимум */conf/base* и */conf/default*, а также, возможно, отдельные каталоги для подсетей и/или отдельных IP-адресов. Содержимое этих каталогов будет использоваться при создании файловых систем в памяти, которые будут монтироваться в корневой каталог, что позволит вносить изменения, характерные для отдельных хостов.

Проще всего объяснить этот механизм на примере. Внесем некоторые изменения и настроим каталог */etc* так, чтобы он монтировался как файловая система в памяти на бездисковых хостах.

Каталог /conf/base

Каталог */conf/base* содержит базовую информацию о разделах, которые требуется смонтировать. Все, что находится в этом каталоге, влияет на все бездисковые станции. Для каждого каталога, в который будет смонтирована файловая система в памяти, нужно создать соответствующий каталог в */conf/base*.

Активизация повторного монтирования каталогов на бездисковой станции

После создания каталога */conf/base/etc* надо сообщить операционной системе FreeBSD о необходимости перемонтировать каталог */etc* на стороне клиента. Для этого создайте файл с именем *diskless_remount*, содержащий имя каталога, в который нужно выполнить монтирование. Звучит довольно сложно, но все это означает, что файл содержит всего лишь одно слово:

```
/etc
```

Тем самым мы предписываем FreeBSD создать MFS для каталога */etc* и смонтировать ее соответствующим образом на стороне бездискового клиента.

Наполнение и подстройка повторно монтируемых файловых систем

По умолчанию, чтобы наполнить файловую систему в памяти, сценарий *rc.initdiskless* копирует содержимое серверного каталога. То есть MFS */etc* – это копия каталога */etc* в файловой системе NFS. Однако некоторые файлы не нужны в файловой системе MFS. Например, бездисковые системы не должны хранить у себя файлы протоколов, откуда следует, что им не нужен *newsyslog* или */etc/newsyslog.conf*. На бездисковых клиентах не требуется выполнять резервное копирование, следовательно файл */etc/dumpdates* также не нужен. Внимательно просмотрев содержимое каталога */etc*, можно выявить довольно много файлов, которые не требуется копировать в MFS */etc*. Уменьшение количества файлов в */etc* позволит сэкономить память, которая так нужна в небольших системах. Однако если удалить больше, чем нужно, система просто не будет загружаться, причем выбор файлов для удаления не так очевиден. Например, если удалить файл */etc/mtree*, машина зависнет в однопользовательском режиме, потому что не сможет наполнить раздел MFS */var*.

Поместите полные пути к нежелательным файлам и каталогам в файл */conf/base/etc.remove*. Например, следующие записи удалят каталоги */etc/gss* и */etc/bluetooth*, а также файл */etc/rc.firewall*:

```
/etc/gss
/etc/bluetooth
/etc/rc.firewall
```

Довольно просто, правда? А теперь вернем кое-что в нашу конфигурацию.

Каталог /conf/default

Каталог *default* содержит файлы, которые передаются бездисковым клиентам. Многие файлы будут идентичны для всех клиентов, например */etc/fstab*. Поместите эти файлы в подкаталог *etc* каталога */conf/default*. Например, чтобы передать файл */etc/fstab* всем клиентам, нужно сохранить его как */conf/default/etc/fstab*.

Файлы из каталога *default* замещают одноименные файлы в повторно смонтированной базовой файловой системе.

Каталоги для отдельных подсетей и клиентов

Очень немногие файлы являются уникальными для отдельных бездисковых хостов. К примеру, вашему бездисковому серверу DNS, в частности, нужен файл *named.conf*, или вашему бездисковому серверу NTP нужен уникальный файл *ntp.conf*. Кроме того, каждый бездисковый хост должен иметь свой уникальный ключ SSH.

Бездисковая система предоставляет возможность перезаписывать базовые конфигурационные файлы, опираясь на адрес подсети или отдельный IP-адрес. Для каждой подсети надо создать каталог с именем, совпадающим с широковещательным адресом подсети, и поместить в него нужные каталоги. Например, наша бездисковая станция имеет IP-адрес 192.168.1.99 и сетевую маску 255.255.255.0. Широковещательный адрес этой сети: 192.168.1.255. Любые файлы, которые требуется скопировать на все бездисковые клиенты в этой сети, должны располагаться в каталоге */conf/192.168.1.255/etc*.

Распространение rc.conf по бездисковым хостам

Большинство системных администраторов FreeBSD единственным местом хранения конфигурации системы считают */etc/rc.conf*, но не забывайте, что также можно использовать файл */etc/rc.conf.local*. Это особенно полезно для бездисковых систем. Если вы хотите создать ферму бездисковых станций, где на всех машинах работают одни и те же сервисы, то можете распространить один и тот же файл *rc.conf* по всем хостам. Все, что характерно для отдельных машин, помещайте в файл */etc/rc.conf.local* и передавайте этот файл только на соответствующие бездисковые станции. Это самый простой способ синхронизировать множество систем.

Точно так же можно организовать перезапись файлов на конкретных хостах с помощью каталога, имя которого совпадает с IP-адресом хоста. Чтобы передать нестандартный файл *rc.conf* в нашу тестовую бездисковую систему, и только в эту систему, его нужно сохранить как */conf/192.168.1.99/etc/rc.conf*.

Все файлы для отдельных хостов будут перезаписывать одноименные файлы, предназначенные для отдельных подсетей. Точно так же все файлы, предназначенные для отдельных подсетей, будут перезаписывать одноименные файлы из каталога *default*, файлы из которого, в свою очередь, перезапишут файлы базовой конфигурации.

Пакеты и файлы для бездисковых систем

От операционной системы без дополнительного программного обеспечения мало проку. Рано или поздно вам придется устанавливать пакеты на бездисковые системы. Кроме того, придется устанавливать на бездисковые системы и конфигурационные файлы.

Установка пакетов

Суть установки программного обеспечения для бездисковых клиентов заключается в том, что установка должна выполняться в каталог, который бездисковые клиенты будут использовать в качестве своей файловой системы. Установить пакеты в нестандартный каталог можно с помощью функции chroot утилиты pkg_add(8). Прежде чем начать установку, файлы программного обеспечения нужно скопировать в каталог бездискового клиента.

```
# cp /usr/ports/packages/All/jdk-1.5.0.11p5,1.tbz /var/diskless/1
# pkg_add -C /var/diskless/1 /jdk-1.5.0.11p5,1.tbz
```

Не забывайте, что функция chroot изменяет корневой каталог процесса. Прежде чем утилита pkg_add(8) выполнит хоть одно действие, она сделает каталог */var/diskless/1* своим корневым каталогом. После этого, с точки зрения pkg_add(8), пакет будет находиться в корневом каталоге.

Кроме того, на бездисковых клиентах можно использовать и «порты», но организовать это немного сложнее. Если вам требуется какой-то «порт», рекомендую сначала собрать из него пакет, а затем установить этот пакет командой `pkg_add -C` для функции chroot.

Конфигурационные файлы бездисковых систем

Вам придется решить, распространять ли по всем бездисковым клиентам идентичные конфигурационные файлы – или создать для каждого клиента отдельную конфигурацию. Действительно ли конфигурационные файлы всех клиентов уникальны и должны сопровождаться по отдельности? Ниже описывается, как можно настроить большинство

из наиболее общих конфигурационных файлов для обеспечения работы без диска.

/etc/rc.conf

Каждому хосту необходим файл *rc.conf* с базовыми настройками. Имя хоста и IP-адрес уже установлены с помощью DHCP, но еще нужно, как минимум, создать доступные для записи файловые системы */var* и */tmp*.

/etc/fstab

Каждой системе FreeBSD требуется таблица файловых систем. В конце концов, даже на бездисковых машинах имеются файловые системы! Вот пример содержимого файла */etc/fstab* для нашего случая бездисковой машины:

```
192.168.1.1:/var/diskless/1    /       nfs     ro          0 0
md                             /tmp    mfs     -s=30m,rw   0 0
md                             /var    mfs     -s=30m,rw   0 0
md                             /etc    mfs     -s=2m,rw    0 0
```

Если вы предусматриваете другие разделы (например, для домашних каталогов), их также нужно указать здесь. Можно также предусмотреть использование локальных жестких дисков для пространства свопинга и временного каталога. Однако если вы используете локальный диск на общедоступной машине, рекомендую на каждой загрузке создавать эти разделы заново с помощью newfs(8), чтобы данные, оставшиеся от предыдущего пользователя, не создали угрозу для следующего.

Ключи SSH

Если ваши бездисковые хосты предоставляют сервисы SSH, они должны иметь уникальные ключи SSH. Время от времени можно услышать, как кто-то использует одни и те же ключи для всей фермы бездисковых станций. И хотя при этом трафик шифруется и, следовательно, не может быть прочитан по пути его следования, тем не менее, с общими ключами невозможно отличить одну удаленную машину от другой.

Оставшись без собственных устройств, FreeBSD автоматически создает новые ключи SSH во время загрузки. Так как эти ключи записываются в файловую систему, находящуюся в памяти, они теряются при каждом завершении работы. Не забывайте, что клиентское программное обеспечение SSH кэширует ключи для всех удаленных хостов, к которым выполняется подключение, и отвергает соединение, если ключ изменился. Если ключи SSH на ваших бездисковых хостах будут постоянно изменяться, это быстро выведет вас из равновесия. Избежать этого достаточно просто, достаточно лишь зайти в каталог конфигурации для каждого клиента на стороне сервера и создать там ключи:

```
# cd /var/diskless/1/conf/192.168.1.99/etc
# mkdirs ssh
```

```
# cd ssh
# touch ssh_host_key
# ssh-keygen -t dsa -f ssh_host_dsa_key -N ''
# ssh-keygen -t rsa -f ssh_host_rsa_key -N ''
```

Если у вас много бездисковых клиентов, напишите сценарий, который будет решать эту задачу.

Так как постоянная файловая система доступна только для чтения, на бездисковой станции даже root не сможет причинить сколько-нибудь серьезный ущерб. Учитывая это, вы могли бы решить предоставить сервис SSH всем бездисковым станциям под учетной записью root, чтобы не создавать пароли для каждого администратора. Скопируйте */etc/ssh/shhd_config* в каталог */conf/default/etc/ssh* и установите параметр `PermitRootLogin` в значение `yes`, чтобы разрешить это. (Реализация подобного решения на сервере с его файловой системой, доступной для чтения/записи, несет жуткую угрозу безопасности, но бездисковая система – это редкое исключение из правил.)

Файлы паролей

На бездисковой системе вам наверняка потребуется файл паролей. В системе FreeBSD нет достаточно удобного способа создания файлов паролей, независимых от основной системы. Лучше всего создать файл паролей на тестовой машине или в клетке, а затем скопировать четыре ключевых файла паролей (*master.passwd*, *passwd*, *pwd.db* и *spwd.db*) из каталога */etc* в соответствующий каталог *conf*. (Для создания и установки файлов базы данных можно использовать утилиту pwd_mkdb(8) с ключом `-d`, но вам по-прежнему придется вручную выполнить тяжелую работу по созданию подходящего файла паролей.) Также можно самому запереться в chroot-окружении в корневом каталоге бездисковой станции на сервере и запустить команду adduser(8) оттуда.

syslogd.conf

Не забывайте, у вас нет жесткого диска. Если хотите использовать syslog для бездисковых систем, вам потребуется обеспечить хост протоколов.

Бездисковая установка

Одна из неприятностей наших дней, – небольшие серверы, которые не всегда поставляются в комплекте с приводами для компакт-дисков. И хотя для нормальной работы машине не нужен CD-ROM, его отсутствие усложняет установку системы. Загрузка такого сервера как бездисковой станции ликвидирует эту проблему. NFS может экспортировать образ установочного компакт-диска FreeBSD, что позволяет выполнить загрузку с образом компакт-диска в качестве корневого каталога. Вы загрузитесь прямо в sysinstall!

Наверняка найдутся и другие нужные файлы, но те, что уже рассмотрены, позволят вам запустить в работу бездисковые сетевые станции.

NanoBSD: создаем собственные устройства

Экстренное сообщение: «Компьютеры слишком дороги!»

Многие приобретают компьютер с единственной целью, например создать брандмауэр с функцией трансляции сетевых адресов. Это правда, что высокопроизводительный компьютер на процессоре i486, работающий под управлением FreeBSD, прекрасно справляется с обслуживанием сети небольшого офиса, но стоимость компьютеров превышает стоимость минимально необходимых аппаратных средств. Такие системы громоздки, требуют для работы много электроэнергии, а их жесткие диски еле дышат. Если вы работаете в коммерческой организации, ваш менеджер вряд ли захочет услышать: «Помните старый компьютер, который так тормозил, что даже секретарша от него отказалась? Я сделал из него веб-сервер». Операционная система FreeBSD легко устанавливается на устаревшие компьютеры, но по определенным причинам RAIC (Redundant Array of Inexpensive Crap – избыточный массив недорогого барахла)[1] никогда не упоминается в публикациях. Конечно, вы всегда можете, выцарапав бюджет, приобрести новую машину с полной гарантией, но каждый из нас предпочел бы купить что-то другое, а не компьютер, который будет простаивать 99 процентов времени. Все это составляет понятие «дороговизны». Этого достаточно, чтобы побудить самого ярого сторонника движения за программное обеспечение с открытым исходным кодом рассмотреть возможность приобретения недорогого аппаратного брандмауэра, обменяв гибкость и уверенность на низкое энергопотребление и недорогую тишину.

Вы можете взять все самое лучшее из двух миров, собрав собственное устройство на недорогой аппаратуре с низким энергопотреблением. Операционная система FreeBSD широко используется на рынке встраиваемых устройств. Компании, такие как NetApp и Juniper, собирают высокопроизводительные устройства, работающие под управлением FreeBSD. Инструменты сборки образов FreeBSD для встраиваемых устройств интегрированы непосредственно в дерево исходных текстов FreeBSD. Вам встретится множество способов, но наиболее широко используется метод, получивший название NanoBSD (*/usr/src/tools/tools/nanobsd*).

Небольшие, недорогие и компактные компьютеры производят множество компаний, но я отдаю предпочтение компании Soekris (*http://www.soekris.com*). Компьютеры Soekris размерами напоминают книгу

[1] Потому что тогда промышленность могла бы продавать вам избыточные массивы *дорогого* барахла (Redundant Array of *Expensive* Crap), но это уже другая история.

в мягкой обложке. У них нет вентиляторов, видеокарт, очень низкое энергопотребление, кроме того, они проектировались специально для работы под управлением открытого программного обеспечения. Процессор, память, последовательные порты и порты USB, а также сетевые интерфейсы впаяны непосредственно в материнскую плату. Вы можете купить одну материнскую плату или вместе с корпусом. С этими компьютерами можно использовать жесткий диск или карту mini-PCI, в качестве диска можно использовать flash-карты. Но самое замечательное, что они стоят всего пару сотен долларов – достаточно дешево, чтобы благополучно преодолеть сопротивление бухгалтерии без заполнения множества документов. При покупке вы не получите гарантию, но за стоимость одного маленького сервера в стойке вы сможете купить десять компьютеров Soekris. Если один выйдет из строя, его можно просто выкинуть, а на его место поставить другой. Замечу, что эти системы настолько просты, что мне пока не приходилось сталкиваться со случаями выхода из строя.[1]

В примерах далее мы будем использовать одну из таких систем, с flash-картой вместо жесткого диска. Все примеры в этом разделе были проверены на Soekris net4801 с микропроцессором на 266 МГц, ОЗУ 128 Мбайт и блоком питания AC/DC. Для среднего использования я рекомендую flash-карту объемом 128 или 256 Мбайт. Больший объем будет пустой тратой, если конечно, вы не собираетесь использовать во встраиваемой системе действительно большие программные продукты. Впрочем, и сейчас, когда я пишу эти строки, карты емкостью 128 Мбайт уже практически исчезли из продажи, поэтому вам, скорее всего, придется использовать самую маленькую flash-карту, какую только удастся отыскать. Есть аналогичные аппаратные средства других производителей, но для них вам придется немного изменить приведенные ниже инструкции. (Мы на переднем крае FreeBSD, помните?)

Для следующего примера я построил минимально возможный сервер DNS, включив в систему только программное обеспечение, необходимое для поддержки DNS.

Что такое NanoBSD?

NanoBSD – это сценарий командной оболочки, который собирает усеченную версию FreeBSD, пригодную для запуска на простейших устройствах. Термин *простейшее устройство* означает, что система собрана для решения одной или нескольких узкоспециализированных задач. Система NanoBSD превосходно справится с ролью сервера имен, брандмауэра или маршрутизатора. Однако из нее не получится хороший настольный компьютер, так как настольные системы предполагают возможность решения сложных задач, круг которых все шире.

[1] Как оказалось, на один из моих тестов – когда я пролил «Слурпи»,– он не был рассчитан. Зато искры весьма впечатлили.

Точно так же большинство многоцелевых серверов плохо подходят для работы под управлением NanoBSD. Если вы хотите превратить свой брандмауэр NanoBSD в DNS-сервер NanoBSD, придется создать новый образ NanoBSD и переустановить его с самого начала. Это не сложно, но непривычно для сервера.

Создать образ NanoBSD непросто, и вам, вероятно, понадобится несколько попыток, прежде чем вы соберете свой первый диск с NanoBSD. Более того, процесс настройки тоже достаточно непрост, чтобы его можно было разбить на пошаговые инструкции. Вполне возможно, вам придется прочитать этот раздел дважды.

Процесс создания образа NanoBSD делится на два основных этапа: сборка операционной системы и образа диска и последующая настройка образа под ваши нужды. Мы рассмотрим каждый этап создания NanoBSD – сначала сконцентрируемся на создании образа NanoBSD, который будет загружаться на вашей аппаратуре, а затем выполним доводку этого образа.

Аппаратное окружение и flash-диск

NanoBSD – это версия FreeBSD, настроенная под ваши требования и установленная в образ диска, который можно скопировать прямо на flash-диск. Прежде чем приступать к созданию образа диска, нужно внимательно изучить геометрию flash-карты. На flash-дисках нет ни цилиндров, ни головок, хотя они сообщают компьютеру об их наличии. Хуже того, различные компьютеры могут видеть разные геометрии одной и той же flash-карты. Многие инструменты, такие как diskinfo(8), нормально работают с flash-картой, но в зависимости от конкретной модели карты они могут давать неверную информацию о ней. Единственный способ надежно идентифицировать геометрию карты состоит в том, чтобы смонтировать ее в целевой системе и посмотреть, что сообщит о себе flash-карта.

В устройствах компании Soekris в качестве консоли по умолчанию используется последовательный порт. Однако в отличие от многих других встраиваемых систем в устройствах Soekris скорость обмена по последовательному порту установлена равной 19200 бит/с, а не 9600 бит/с. Подсоедините нуль-модемный кабель к последовательному порту устройства, другой конец кабеля подключите к последовательному порту компьютера и откройте сеанс программы tip(1), выбрав скорость обмена 19 200 бит/с.

```
# tip -19200 sio0
```

Теперь включите устройство Soekris. После вывода начальных сообщений вы увидите на экране примерно вот что:

```
comBIOS ver. 1.28 20050529 Copyright (C) 2000-2005 Soekris Engineering.

net4801
```

```
0128 Mbyte Memory                     CPU Geode 266 Mhz

❶Pri Sla KODAK ATA_FLASH              ❷LBA 984-4-32 63 Mbyte
Slot   Vend Dev  ClassRev Cmd  Stat CL LT HT  Base1    Base2  Int
-------------------------------------------------------------------
0:00:0 1078 0001 06000000 0107 0280 00 00 00 00000000 00000000
...
```

На экране много интересной информации об аппаратном обеспечении, но сейчас нас больше всего интересуют сведения о диске. Данный flash-диск виден как подчиненное устройство, подключенное к первичному контроллеру (primary slave) ❶. Кроме того, машина видит эту flash-карту емкостью 63 Мбайт как диск, имеющий 984 цилиндра, 4 головки и 32 сектора ❷. Теперь мы сможем подготовить образ диска NanoBSD с учетом этой геометрии. Выключите устройство Soekris и извлеките flash-карту. В следующий раз, когда вы включите это устройство, оно загрузит FreeBSD с flash-карты.

NanoBSD и бездисковые системы

Если у вас есть бездисковая платформа, загрузите разок бездисковую систему NanoBSD. Это позволит вам заглянуть в файл *dmesg.boot* и поможет создать отличную конфигурацию ядра. Кроме того, вы сможете узнать, какое имя устройства ваша система использует для обозначения flash-карты, что позднее позволит вам сэкономить время на настройке.

Инструменты NanoBSD

Вы найдете NanoBSD в */usr/src/tools/tools/nanobsd*. Базовый набор инструментов включает каталог с именем *Files*, сценарий командного интерпретатора *nanobsd.sh* и файл *FlashDevice.sub*.

Каталог *Files* содержит файлы, которые следует скопировать в образ NanoBSD. NanoBSD поставляется с несколькими сценариями, упрощающими сопровождение систем NanoBSD, позднее вы сможете добавить их в свой арсенал.

Файл *FlashDevice.sub* содержит описания геометрии различных flash-дисков. Этот файл далеко не полон из-за неисчислимого множества производителей flash-дисков, но его легко расширить.

Основные функции NanoBSD находятся в сценарии командного интерпретатора *nanobsd.sh*. Фактически это программа, реализующая процедуру сборки из исходного кода, описанную в главе 13.

В дополнение к этим файлам вам потребуется конфигурационный файл NanoBSD и, возможно, каталог *packages*.

Дополнение файла FlashDevice.sub

Есть вероятность, что ваша flash-карта отсутствует в файле *FlashDevice.sub*. Добавить его описание в файл очень просто, дело всего нескольких секунд. Все, что нужно сделать, это скопировать одну из имеющихся записей и исправить ее в соответствии с параметрами вашей flash-карты. Вам необходимо знать название диска и размер, а также его геометрию. Название и размер можно прочитать на наклейке. Геометрия устройства отображается на экране BIOS устройства Soekris, равно как и в устройствах других производителей. Запись для новой карты выглядит примерно так:

```
❶ samsung)
       #Source: mwlucas@freebsd.org
       case $a2 in
❷      128|128mb)
❸          NANO_MEDIASIZE=`expr 130154496 / 512`
❹          NANO_HEADS=8
❺          NANO_SECTS=32
       ;;
❻ *)
       echo "Unknown Samsung Corp Flash Capacity"
       exit 2
       ;;
  esac
  ;;
```

Сначала нужно указать производителя карты ❶, при этом запись должна стоять на своем месте в алфавитном порядке внутри файла.

Нам известен один объем flash-карты, что находится в наших руках. Это 128 Мбайт ❷. Однако нам требуется, чтобы сценарий мог принимать в качестве аргумента любое из двух обозначений емкости – `128` и `128mb`, поэтому указываем оба.

Значение `NANO_MEDIASIZE` ❸ вычисляется умножением числа цилиндров на число головок, на число секторов в дорожке и на 512. Вычислите это произведение и подставьте его в инструкцию expr(1). (Да, NanoBSD сразу же делит это значение на 512.)

Значения `NANO_HEADS` ❹ и `NANO_SECTS` ❺ – это число головок и число секторов, которые мы видели на экране BIOS.

У нас также имеется вариант выбора по умолчанию ❻, чтобы в случае использования, например, отсутствующей в списке карты Samsung емкостью 256 Мбайт сценарий создания образа NanoBSD прекратил работу, вместо того чтобы создать дефектный образ.

Добавьте в файл *FlashDevice.sub* запись вроде этой, и можно продолжить создание вашего образа NanoBSD.

Параметры настройки NanoBSD

Настройка сборки NanoBSD производится с помощью конфигурационного файла. Система FreeBSD по умолчанию не содержит конфигурационный файл для NanoBSD, на что есть веские причины: каждая инсталляция NanoBSD уникальна. В Интернете можно найти множество примеров конфигурации, а я представлю здесь свой собственный файл. Однако из-за возможных изменений в последующих версиях FreeBSD вам может потребоваться внести изменения в свою конфигурацию. Конфигурация NanoBSD, пригодная для FreeBSD 7.0, вероятно будет не совсем корректна в версии FreeBSD 7.5 и наверняка некорректна в версии FreeBSD 8.0. Вы можете создать несколько файлов конфигурации для разных типов сборок NanoBSD.

Ниже приведены некоторые полезные параметры конфигурации и описание их применения.

`NANO_NAME=full`

Имя, которое вы присваиваете сборке NanoBSD. Это имя NanoBSD использует для присвоения имен рабочим каталогам и файлу образа диска.

`NANO_SRC=/usr/src`

Местоположение дерева исходного кода, который NanoBSD использует для создания образов дисков. Если вам требуется указать другое местоположение, измените этот параметр. Вы можете собрать NanoBSD в любой версии FreeBSD, достаточно близкой к той, что вы используете.

`NANO_TOOLS=tools/tools/nanobsd`

Сценарии и файлы NanoBSD считаются частью исходного кода FreeBSD. Это означает, что при запуске csup файлы NanoBSD могут обновиться. Если вы внесете какие-либо изменения в файлы NanoBSD, csup(1) затрет эти изменения. При изменении базовых компонентов NanoBSD желательно скопировать каталог NanoBSD куда-нибудь за пределы дерева исходного кода (например, в */usr/src/tools/tools/local-nanobsd*). Отслеживание различий между вашими нестандартными инструментами NanoBSD и базовыми системными инструментами NanoBSD, полностью ложится на вас.

`NANO_PACKAGE_DIR=${NANO_SRC}/${NANO_TOOLS}/Pkg`

В этом каталоге NanoBSD ищет все пакеты, которые должны быть установлены в образ диска. По умолчанию это каталог */usr/src/tools/tools/nanobsd/Pkg*.

`NANO_PMAKE="make -j 3"`

Данный параметр определяет ключ `-j`, передаваемый утилите make(1) во время обработки целей `buildworld` и `buildkernel`. Если вам придется столкнуться с ошибками во время сборки NanoBSD в ходе

обработки целей `buildworld` или `buildkernel`, попробуйте установить значение `make -j 1`.

`CONF_BUILD=' '`

Любые параметры, определенные здесь, будут использованы на этапе `make buildworld`. Полный список параметров, допустимых в вашей версии FreeBSD, вы найдете на странице руководства src.conf(5).

`CONF_INSTALL=' '`

Любые параметры, определенные здесь, будут использованы на этапе `make installworld`. Полный список допустимых параметров вы также найдете на странице руководства src.conf(5).

`CONF_WORLD=' '`

Значение этого параметра используется как ключи в ходе всего процесса сборки.

`NANO_KERNEL=GENERIC`

Вы наверняка захотите собрать нестандартное ядро для своего образа NanoBSD. Укажите его здесь. Это должно быть имя файла в каталоге */usr/src/sys/i386/conf*.

`NANO_CUSTOMIZE=""`

По умолчанию, в ходе сборки образа NanoBSD не использует какие-либо сценарии настройки. О доводке образа диска мы поговорим ниже в этой главе. Укажите здесь любые сценарии настройки образа, если хотите их использовать.

`NANO_NEWFS="-b 4096 -f 512 -i 8192 -O1 -U"`

Создание файловой системы в образе диска производится с помощью newfs(8). Укажите здесь все нужные вам параметры этой команды. Не забывайте, что NanoBSD будет использовать этот диск в режиме доступа «только для чтения», поэтому здесь не следует использовать какие-либо нестандартные параметры.

`NANO_DRIVE=ad0`

Нужно указать имя файла устройства, соответствующего flash-диску. Чтобы узнать это, можно загрузить целевое устройство и посмотреть, под каким именем оно видит эту карту.

`NANO_MEDIASIZE=1000000`

Емкость вашего flash-диска в 512-байтовых секторах.

`NANO_IMAGES=2`

NanoBSD позволяет создать несколько образов операционной системы на одном flash-диске. Это может упростить обновление, но требует в два раза больше пространства. Поскольку в наши дни flash-диски достаточно емки и способны вместить практически любой образ FreeBSD, который вы можете собрать, я не вижу веских причин изменять это значение.

`NANO_CODESIZE=0`

Здесь можно задать объем дискового пространства, выделяемого для дисковых участков операционной системы. Если установить значение `0`, сценарий, выполняющий сборку, выделит максимальное возможное пространство.

`NANO_CONFSIZE=2048`

Дополнительно NanoBSD создает дисковый участок для хранения конфигурационных файлов, как показано ниже. Данный параметр определяет размер этого участка в 512-байтовых секторах. 2048 секторов эквивалентно одному мегабайту.

`NANO_DATASIZE=0`

При желании NanoBSD может создать четвертый дисковый участок для хранения данных. Это немного необычно, но вполне допустимо для сборки устройства NanoBSD, способного долго хранить данные. Если указать значение `0`, сценарий *nanobsd.ch* не будет создавать раздел для данных. Если указать отрицательное число, NanoBSD сделает раздел с данными настолько большим, насколько это возможно.

`NANO_RAM_ETCSIZE=10240`

Объем дискового пространства в 512-байтовых секторах, выделяемого для каталога */etc*. Значение `10240` соответствует пяти мегабайтам.

`NANO_RAM_TMPVARSIZE=10240`

Объем дискового пространства, выделяемого для каталогов */var* и */tmp*. 5 Мбайт – это достаточно много при условии, что система не предполагает ведение файлов протоколов.

`NANO_SECTS=63` и `NANO_HEADS=16`

Здесь определяется геометрия используемого flash-диска. Не устанавливайте этот параметр в конфигурационном файле! Настраивайте flash-карту в файле *FlashDevice.sub*.

`FlashDevice samsung 128`

Этот параметр предписывает NanoBSD взять описание геометрии диска из файла *FlashDevice.sub* и не использовать значения `NANO_SECTS` и `NANO_HEADS`. Данный пример записи означает, что будет использован flash-диск емкостью 128 Мбайт, произведенный компанией Samsung.

Пример конфигурации NanoBSD

Все эти параметры выглядят достаточно внушительно, но как их использовать на практике? Давайте создадим конфигурационный файл и рассмотрим его повнимательнее. Не забывайте, я собираюсь создать сервер DNS на базе устройства компании Soekris. Любые параметры, которые я не использую, получают значения по умолчанию.

```
# конфигурация для сборки сервера DNS на базе Soekris
❶ NANO_NAME=SoekrisDNS
NANO_IMAGES=2
❷ NANO_KERNEL=SOEKRIS
❸ NANO_DRIVE=ad1
NANO_PMAKE="make -j 3"
FlashDevice samsung 128
customize_cmd cust_comconsole
customize_cmd cust_allow_ssh_root
customize_cmd cust_install_files
❹ CONF_INSTALL='
WITHOUT_TOOLCHAIN=YES
'
CONF_WORLD='
COMCONSOLE_SPEED=19200
NO_MODULES=YES
WITHOUT_ACPI=YES
WITHOUT_ASSERT_DEBUG=YES
WITHOUT_ATM=YES
WITHOUT_AUDIT=YES
WITHOUT_AUTHPF=YES
❺ #WITHOUT_BIND=YES
WITHOUT_BLUETOOTH=YES
WITHOUT_CALENDAR=YES
WITHOUT_CPP=YES
WITHOUT_CVS=YES
WITHOUT_CXX=YES
WITHOUT_DICT=YES
WITHOUT_DYNAMICROOT=YES
WITHOUT_EXAMPLES=YES
WITHOUT_FORTH=YES
WITHOUT_FORTRAN=YES
WITHOUT_GAMES=YES
WITHOUT_GCOV=YES
WITHOUT_GDB=YES
WITHOUT_GPIB=YES
WITHOUT_GROFF=YES
WITHOUT_HTML=YES
WITHOUT_I4B=YES
WITHOUT_INET6=YES
WITHOUT_INFO=YES
WITHOUT_IPFILTER=YES
WITHOUT_IPX=YES
WITHOUT_KERBEROS=YES
WITHOUT_LIBPTHREAD=YES
WITHOUT_LIBTHR=YES
WITHOUT_LPR=YES
WITHOUT_MAILWRAPPER=YES
WITHOUT_MAN=YES
WITHOUT_NCP=YES
WITHOUT_NETCAT=YES
```

```
WITHOUT_NIS=YES
WITHOUT_NLS=YES
WITHOUT_NLS_CATALOGS=YES
WITHOUT_NS_CACHING=YES
WITHOUT_OBJC=YES
WITHOUT_PAM_SUPPORT=YES
WITHOUT_PF=YES
WITHOUT_PROFILE=YES
WITHOUT_RCMDS=YES
WITHOUT_RCS=YES
WITHOUT_RESCUE=YES
WITHOUT_SENDMAIL=YES
WITHOUT_SHAREDOCS=YES
WITHOUT_SPP=YES
WITHOUT_SYSCONS=YES
WITHOUT_USB=YES
#WITHOUT_WPA_SUPPLICANT_EAPOL=YES
WITHOUT_ZFS=YES
'
```

Конфигурация начинается с определения уникального имени образа диска с помощью параметра `NANO_NAME` ❶. Для этого устройства я выбрал нестандартную конфигурацию ядра ❷. Где я взял эту конфигурацию? Я выполнил бездисковую загрузку устройства Soekris, получил файл */var/run/dmesg.boot* для ядра GENERIC, затем поискал в Интернете другие конфигурации ядер для net4801, чтобы выяснить потенциально полезные параметры настройки, используемые другими людьми. Единственное неудобство, которое пришлось преодолеть при бездисковой загрузке, это отключение ACPI в загрузчике.

Благодаря бездисковой загрузке я узнал, что устройство net4801 видит flash-карту как привод */dev/ad1* ❸. Если бы я не установил значение этого параметра в моем конфигурационном файле, во время загрузки NanoBSD сообщила бы, что ядро не может отыскать корневой раздел на */dev/ad0*.

Далее перечислены различные сценарии `customize_` для настройки этого образа. О настройке мы поговорим ниже в этой главе. Все эти сценарии входят в набор инструментов NanoBSD по умолчанию, и я рекомендую использовать их при создании своего первого образа.

Далее следуют специальные параметры утилиты make(1), которые будут использованы в процессе установки. Мы рассмотрим эти параметры немного ниже, а пока вам достаточно знать, что параметр `WITHOUT_TOOLCHAIN` предписывает утилите make(1) не устанавливать компилятор, библиотеки и другие инструменты, необходимые для сборки программного обеспечения. Мы не предполагаем собирать программное обеспечение на встраиваемой системе!

Заметьте, как установлено значение параметра `CONF_INSTALL` ❹. Я поместил апостроф (') после знака равенства, значение параметра указано в отдельной строке и затем следует закрывающий апостроф ('). Это по-

зволяет наглядно отделять множественные значения параметра друг от друга и делает содержимое файла более удобочитаемым. Похожим образом определяется значение параметра CONF_WORLD. NanoBSD предполагает, что каждая переменная будет указана в отдельной строке, так как из них будет создан скрытый файл *make.conf*.

Каждый из параметров в параметре CONF_WORLD предписывает NanoBSD не собирать соответствующий ему раздел FreeBSD. Назначение большинства из них очевидно, например, WITHOUT_ACPI предписывает не собирать ACPI. NO_MODULES – не собирать модули ядра. Я не включаю в системы то, чего не хотел бы видеть в окончательном образе диска, например Sendmail, игры и фильтры пакетов.

Все эти параметры похоронены в недрах механизма сборки FreeBSD. Некоторым из них уже довольно много лет, другие совершенно новые. Некоторым из них суждено исчезнуть, и нет никакой гарантии, что какие-то из них останутся допустимы в будущих версиях. Список всех допустимых параметров сборки содержится в странице руководства src.conf установленной версии FreeBSD. Я включил в свой пример полный список параметров, чтобы вы видели, насколько агрессивно можно урезать возможности системы. Размеры FreeBSD существенно меньше размеров многих версий Linux, но это не означает, что ее нельзя сделать еще меньше! Однако некоторые параметры сборки, перечисленные в src.conf(5), не должны использоваться для NanoBSD. Например, WITHOUT_SETUID_LOGIN окажет отрицательное влияние на возможность входа в систему. Использование параметра WITHOUT_SYMVER приведет к удалению символов контроля версий, что сделает невозможным использование в вашем образе NanoBSD двоичных файлов, скомпилированных на других машинах FreeBSD. Прежде чем использовать тот или иной параметр, обязательно внимательно изучите его влияние.

Следует также заметить, что вам, возможно, придется отключать некоторые из этих параметров сборки для каких-то образов NanoBSD. Я собираю сервер DNS, поэтому мне нужен сервер BIND, и я закомментировал элемент WITHOUT_BIND ❺ в своем конфигурационном файле для данной конкретной сборки.

В параметре CONF_INSTALL я перечислил только те параметры, которые используются не в процессе сборки, а в процессе установки FreeBSD 7.0. Параметр WITHOUT_TOOLCHAIN предписывает команде `make installworld` не устанавливать компилятор, заголовочные файлы и сопутствующие инструменты в образ диска. Нам потребуется новый компилятор для сборки новой версии FreeBSD, но он не нужен в окончательном образе. В процессе установки будет пропущено все, что мы исключили из сборки, но для запуска `make buildworld` необходима свежая версия компилятора.

Это полный конфигурационный файл NanoBSD. Теперь попробуем с его помощью собрать свой образ диска.

Сборка NanoBSD

Запустите файл *nanobsd.sh* как сценарий командного интерпретатора, передав ему имя конфигурационного файла с помощью ключа `-c`:

```
# /bin/sh nanobsd.sh -c mynanoconfig.txt
```

В ходе сборки NanoBSD на экран выводятся вполне типичные сообщения:

```
## Clean and create object directory (/usr/obj/nanobsd.SoekrisDNS/)
## Construct build make.conf (/usr/obj/nanobsd.SoekrisDNS//make.conf)
## run buildworld
### log: /usr/obj/nanobsd.SoekrisDNS//_.bw
## build kernel (SOEKRIS)
...
## build diskimage
### log: /usr/obj/nanobsd.SoekrisDNS//_.di
# NanoBSD image completed
```

Если вы в конце получили сообщение `NanoBSD image completed`, следовательно, сборка прошла успешно. В противном случае причину ошибки на последнем этапе вы найдете в файле протокола. Вполне обычное дело, если набор параметров в конфигурационном файле, приводящий к нормальной сборке образа, будет получен не с первой попытки, особенно при агрессивном использовании параметров `WITHOUT_` с целью уменьшить размер FreeBSD.

Каталог сборки NanoBSD

Для сборки NanoBSD используется подкаталог каталога */usr/obj*, имя которого задано параметром `NANO_NAME`. В нашем конфигурационном файле указано, что выполняется сборка NanoBSD с именем SoekrisDNS, откуда следует, что каталогом сборки будет */usr/obj/nanobsd.SoekrisDNS*. Перейдите в этот каталог и посмотрите на файлы. Вы найдете несколько файлов со странными именами, несколько каталогов и образов дисков.

NanoBSD использует файлы, имена которых начинаются с `_.`, как маркеры, протоколы и временные каталоги. Несмотря на лаконизм сообщений, наблюдаемых в процессе сборки, в действительности NanoBSD не скрывает вывод используемых команд, просто он перенаправляется в другие файлы (табл. 20.1).

Таблица 20.1. Файлы, создаваемые в ходе сборки NanoBSD

Файл	Назначение
_.bk	Протокол команды `make buildkernel`
_.bw	Протокол команды `make buildworld`
_.cust.<X>	Установка настройки X
_.di	Создание образа диска командой fdisk
_.dl	Создание метки диска командой disklabel

Файл	Назначение
_.env	Окружение, использованное при сборке NanoBSD
_.etc	Протокол команды `make distribution`
_.fdisk	Конфигурация fdisk(8)
_.ik	Протокол команды `make installkernel`
_.iw	Протокол команды `make installworld`
_.w	Вновь созданный «мир»
disk.image	Образ диска для единственного дискового участка
disk.full	Образ полного диска

При выяснении причин ошибок во время сборки используйте файлы протоколов.

Поиск причин ошибок во время сборки NanoBSD

Сообщения об ошибках, которые выводятся в процессе сборки, не отличаются особой информативностью. Например, вот полный вывод неудачной сборки NanoBSD:

```
# /bin/sh nanobsd.sh -c soekris-dns.conf
## Clean and create object directory (/usr/obj/nanobsd.SoekrisDNS/)
## Construct build make.conf (/usr/obj/nanobsd.SoekrisDNS//make.conf)
## run buildworld
### log: ❶/usr/obj/nanobsd.SoekrisDNS//_.bw
```

Здесь видно, что попытка сборки потерпела неудачу, но не видно, почему.

Чтобы выяснить действительную причину, загляните в файл, имя которого указано в сообщении. В данном случае это файл протокола */usr/obj/nanobsd.SoekrisDNS/_.bw* ❶. В конце этого файла находятся последние сообщения, выведенные в ходе неудавшейся сборки. Здесь вы увидите сообщения об ошибках, типичные для процесса обновления.

Одно из обстоятельств, которые следует запомнить, это использование команды `make -j 3` по умолчанию. Эта команда может порождать малопонятные сообщения об ошибках в файле протокола, а сама ошибка, вызвавшая отказ, может располагаться несколькими строками выше.

Почему установка NanoBSD терпит неудачу, в то время как обычное обновление из того же исходного кода проходит благополучно? Основной причиной является несовместимость параметров сборки из дерева исходного кода. Мой файл конфигурации NanoBSD включает множество разнообразных параметров `WITHOUT_`. Вы могли заметить, что некоторые из них закомментированы. Эти параметры отрицательно повлияли на возможность сборки. Хотя для моего сервера DNS в действительности не требуется программное обеспечение, представляемое закомментированными мной параметрами, но отсутствие этих парам-

етров сделало успешную сборку NanoBSD невозможной. (Параметр WITHOUT_BIND закомментирован потому, что мне действительно необходим сервер BIND в данном образе.)

Еще одна возможная проблема при сборке образа NanoBSD – это когда сама сборка прошла успешно, но конструирование образа диска завершилось ошибкой:

```
/usr/obj/nanobsd.SoekrisDNS/_.mnt: write failed, filesystem is full
(Перевод: ошибка записи, файловая система заполнена)
```

Для размещения ваших файлов требуется больше места, чем доступно на диске. По умолчанию участок */conf* занимает 1 Мбайт. NanoBSD разбивает оставшееся пространство пополам. Если flash-карта имеет емкость 128 Мбайт, ни один из двух образов NanoBSD не должен занимать больше 63,5 Мбайт дискового пространства. Если в состав сборки FreeBSD включены какие-то дополнительные файлы и были установлены пакеты, что привело к увеличению размера образа, то его конструирование потерпит неудачу. Не забывайте, что производители flash-карт измеряют емкость своих устройств так же, как и производители жестких дисков, по основанию 10, а не по основанию 2. В действительности емкость карты на 128 Мбайт немного меньше этого числа. Попробуйте поместить на диск только один образ NanoBSD, уменьшить объем включаемого программного обеспечения или приобрести flash-карту большей емкости.

Законченная сборка

Итак, сборка закончена! Давайте заглянем в образ диска и посмотрим, что в нем находится:

```
# mdconfig -af _.disk.full
md0
```

Если запустить утилиту fdisk(8) и передать ей образ диска, можно увидеть три участка – два больших и один совсем маленький. Два больших участка – это образы NanoBSD. Смонтируйте один из них и посмотрите, что в нем находится.

```
# mount /dev/md0s1a /mnt
```

Изучив содержимое, нетрудно заметить, что система NanoBSD собрана полностью, но пока еще ничего стоящего не делает. Это ненастроенная инсталляция FreeBSD. Тем не менее, я рекомендую скопировать образ на flash-диск и выполнить пробную загрузку на устройстве, чтобы убедиться, что система работоспособна. Установку NanoBSD на flash-диск */dev/da0* выполняет команда:

```
# dd if=disk.full of=/dev/da0
```

Теперь извлеките flash-диск из компьютера, подключите его к вашему устройству Soekris, запустите последовательную консоль и нажмите кнопку включения питания.

Скорость обмена с последовательной консолью

Если попробовать загрузить устройство Soekris с типичной инсталляцией FreeBSD, у вас возникнут трудности. Дело в том, что в устройствах Soekris скорость обмена с последовательной консолью по умолчанию составляет 19 200 бит/с, тогда как FreeBSD по умолчанию использует более привычную скорость 9600 бит/с. Это означает, что настроив терминал на скорость обмена 19 200, вы сможете увидеть все сообщения времени загрузки устройства Soekris, но при запуске FreeBSD на экране будет сплошной мусор. Если в терминале настроить скорость обмена 9600, вы сможете наблюдать сообщения FreeBSD, но на начальном этапе вместо сообщений Soekris будет выводиться всякий мусор.

Изменение скорости обмена с консолью в системе FreeBSD будет одной из первых настроек, которые мы выполним. Однако при первой пробной загрузке образа FreeBSD оставьте в консоли скорость обмена 9600.

Настройка NanoBSD

Теперь, с начальной инсталляцией FreeBSD, способной загружаться на нашем устройстве, настроим ее под наши нужды. Теоретически настройку NanoBSD можно выполнить двумя способами: отредактировать полученный образ диска или изменить параметры сборки и создать требуемый образ. Первый метод проще, но второй – предпочтительнее. Если вы действительно собираетесь использовать NanoBSD для решения реальных задач, рано или поздно вам придется создать новый образ диска. Возможно, вы хотите наложить заплатку на брешь в системе безопасности или обнаружили приложение, которое обеспечит идеальную работу второй системы NanoBSD. Изменяя параметры сборки, вы существенно облегчите воспроизведение обновленного и работоспособного образа. Вносить изменения в процессе сборки можно тремя способами: с помощью сценариев настройки, копированием файлов и добавлением пакетов.

Сценарии настройки

Сценарии настройки – это сценарии командного интерпретатора, вызываемые из сценария *nanobsd.sh* или из конфигурационного файла. Сценарии настройки служат для изменения файлов, которые наверняка будут изменяться по мере развития операционной системы FreeBSD. Например, через каждые несколько выпусков FreeBSD обновляется версия SSH, включенная в состав системы. В такой ситуации нежелательно создавать свой конфигурационный файл для sshd(8), так как он может оказаться несовместимым с новой версией OpenSSH. Предпочтительнее использовать конфигурационный сценарий, который будет вносить необходимые изменения в файл *sshd_config*, входящий в состав системы, независимо от ее версии. NanoBSD включает сценарий настройки именно для этой цели.

```
cust_allow_ssh_root () (
```

```
        sed -i "" -e '/PermitRootLogin/s/.*/PermitRootLogin yes/' \
            ${NANO_WORLDDIR}/etc/ssh/sshd_config
}
```

Этот простой сценарий sed(1)[1] изменяет единственную строку в файле */etc/ssh/sshd_config*, чтобы разрешить пользователю root выполнять вход в систему NanoBSD через SSH. Этот сценарий окажется бесполезным, если параметр `PermitRootLogin` переименован или удален, но такое происходит крайне редко, если вообще происходит. Вы можете включить в сценарий настройки все, что пожелаете, даже сложнейший сценарий на языке Perl, который принимает значение `NANO_NAME` в качестве аргумента и создает нестандартный файл *rc.conf* для вашего образа.

В состав NanoBSD входят четыре сценария настройки:

cust_comconsole

Разрешает вход через последовательную консоль и через последовательные линии связи.

cust_allow_ssh_root

Разрешает пользователю root вход в систему через SSH.

cust_install_files

Копирует файлы из каталога *Files* в законченный образ.

cust_pkg

Устанавливает в законченный образ пакеты из каталога *Pkg*.

Как вы уже наверняка догадались, сценарии настройки управляют двумя другими способами тонкой настройки образа.

Добавление пакетов

Создайте каталог *Pkg* в каталоге NanoBSD, и сценарий настройки *cust_pkg* выполнит установку в образ всех пакетов, находящихся в этом каталоге. Это действительно так и есть.

При нехватке свободного пространства можно отказаться от установки пакета целиком. Отыщите двоичный исполняемый файл требуемой программы, с помощью ldd(1) определите нужные ей библиотеки и просто скопируйте эти файлы в свой образ.

Добавление файлов

Сценарий настройки *cust_install_files* копирует в образ файлы из дерева подкаталогов каталога *Files*. По умолчанию каталог *Files* содержит единственный подкаталог *root*. Содержимое этого подкаталога копируется в каталог */root* образа диска.

---

[1] Опыт работы с программами sed(1) и awk(1) – еще одна вещь, отличающая «реальных сисадминов» от новичков. Читайте сценарии командного интерпретатора – и вы достаточно хорошо изучите эти инструменты, чтобы прослыть экспертом.

Если вам регулярно приходится собирать образы NanoBSD, рекомендую изменить сценарий *cust_install_files*. NanoBSD включает в каталог *Files* несколько файлов, которые считаются системными. При обновлении дерева исходного кода csup(1) затрет любые изменения в любых файлах каталога *Files*. Чтобы не потерять свои изменения, скопируйте содержимое *cust_install_files* в свой сценарий, дайте ему подходящее имя, например *cust_install_dns_files* и разместите где-нибудь за пределами каталога *Files*.

Я собираю сервер DNS, поэтому мне нужно создать свои файлы *named.conf* и *rndc.conf*. Эти файлы должны находиться в каталоге */var/named/etc/namedb*, поэтому мне нужно создать каталог */usr/src/tools/tools/nanobsd/Files/var/named/etc/namedb* (ик!) и поместить туда мой нестандартный файл.

Точно так же мне потребуется файл *rc.conf*. Я создам файл *Files/etc/rc.conf*, где будет находиться моя конфигурация. Мне необходимы ключи SSH, которые я создам так же, как делал это для бездисковой станции. Похожим образом мне нужно будет создать файл *syslog.conf*, чтобы протоколировать события на сетевом сервере протоколов.

Наконец, надо избавиться от проблемы скорости обмена с последовательной консолью, наблюдаемой при работе с устройствами Soekris. Мы установим скорость обмена с последовательной консолью в конфигурационном файле с помощью переменной `COMCONSOLE_SPEED`, сообщив системе FreeBSD, что по умолчанию должна использоваться скорость 19 200. Если это не сделано, то самый простой способ ликвидировать данный недостаток – сообщить системе FreeBSD о необходимости использовать более высокую скорость обмена с консолью. Сделать это можно с помощью соответствующей записи в файле */boot/loader.conf*. Просто добавьте в файл *Files/boot/loader.conf* следующие строки:

```
comconsole_speed="19200"
console="comconsole,vidconsole"
```

Благодаря им NanoBSD установит скорость обмена по умолчанию с последовательной консолью равной 19 200 бит/с. Однако эти настройки не будут иметь силы, пока не запустится загрузчик. Действительно, лучше собирать систему NanoBSD, задав скорость обмена с консолью равной 19 200 в переменной `CONF_WORLD`.

Уточнение настроек

Одна из проблем, сопровождающих сборку образа NanoBSD, состоит в выборе правильных настроек, подходящих для ваших нужд. Вы добавляете новые файлы и все настройки, которые по вашему мнению могут потребоваться, копируете получившийся образ на flash-диск и загружаете свое устройство только для того, чтобы понять, что вы забыли что-то еще и требуется добавить в образ еще какой-нибудь файл.

Сборка образа NanoBSD требует некоторого времени, но самая продолжительная часть процесса – это сборка ядра и остального программного

обеспечения. Однако можно добавлять файлы в настраиваемый образ и производить сборку образа, минуя сборку ядра и программ. Для этого достаточно вызвать сценарий *nanobsd.sh* с ключом `-b`:

```
# /bin/sh nanobsd.sh -b -c myconfig.txt
```

С этим ключом сценарий пропустит этапы `buildworld` и `buildkernel` и соберет свежий образ диска с измененными файлами.

Работа с NanoBSD

Итак, вы получили образ NanoBSD, который загружается и включает все необходимое программное обеспечение. Образ скопирован на flash-диск и работает без ошибок. Но что если требуется что-то немного изменить или установить обновления?

Не волнуйтесь. В NanoBSD имеются специальные средства как раз для подобных ситуаций.

Незначительные изменения

Для обработки незначительных изменений в NanoBSD есть специальный дисковый участок cfg. Все файлы, расположенные в */cfg*, копируются в каталог */etc* во время загрузки NanoBSD. Это означает, что можно изменить, например, файл */etc/rc.conf*, не собирая новый образ. Чтобы понять, как использовать эту возможность, загляните в сценарий *change_password* в каталоге *Files/root*. Этот сценарий запускает команду passwd(1), чтобы позволить вам изменить пароль пользователя root, монтирует раздел */cfg* в режиме «чтения/записи», копирует туда файлы с паролями и демонтирует раздел */cfg*. Когда система будет загружаться в следующий раз, NanoBSD скопирует эти файлы в каталог */etc*.

Если вы вносите какие-либо изменения в работающую систему NanoBSD, рекомендую тут же внести соответствующие изменения в систему сборки NanoBSD. Благодаря этому при создании нового образа в нем будут учтены все последние изменения.

Обновление образов дисков

Итак, вы собрали образ NanoBSD и запустили его на устройстве Soekris, закрепив его на вершине ветряного генератора, так что теперь можете управлять его выходной мощностью с помощью собственных сценариев SNMP. В один прекрасный день вам придется выполнить обновление. Готовы ли вы вскарабкаться по лестнице на вершину ветряка, чтобы установить новую flash-карту? Конечно нет, большинство разработчиков FreeBSD слишком малоподвижны, чтобы согласиться на подобные мучения.[1]

[1] Однако стоит поместить на верхушку ветряка хорошую выпивку – и три из четырех разработчиков мигом окажутся на лестнице. Проблема в том, как расставить приоритеты для собственных наклонностей.

Не забывайте, что у системы NanoBSD есть два образа диска. В каждый момент времени работает только один из них. Имеющиеся сценарии обновления позволят вам обновить образ диска, который в настоящий момент времени не используется. После обновления можно будет перезагрузиться с обновленного образа. Если новый образ окажется неработоспособным, можно будет загрузиться с прежнего образа и восстановить работу устройства, пока вы будете искать неисправности.

Удаленное обновление инсталляции NanoBSD производится через SSH с помощью сценариев *updatep1* и *updatep2*. Если работающая система находится на первом дисковом участке, используйте сценарий *updatep2*. Если на втором – *updatep1*. Помимо доступа к командной строке, SSH предоставляет вам возможность передавать файлы и даже запускать команды на удаленной системе. В следующем примере выполнен вход в систему NanoBSD и выполняется обновление дискового участка 2 на сервере 192.168.1.5 из образа диска:

```
# ssh 192.168.1.5 ❶cat _.disk.image | sh /root/updatep2
```

Здесь мы подключились к удаленной системе и отправили на нее вывод команды cat(1) ❶, в данном случае – образ одного раздела. На удаленной системе мы запустили сценарий *updatep2*, который сначала путем некоторых базовых проверок убедится, что все в порядке, а затем скопирует образ диска на неиспользуемый участок. Кроме того, сценарий назначит обновленный раздел загрузочным по умолчанию. Если обновление завершилось неудачей, то есть система загружается, но приложения работают некорректно, с помощью последовательной консоли выполните загрузку с другого участка или с помощью boot0cfg(8) на работающей системе сообщите NanoBSD, что в следующий раз нужно загрузиться с другого участка.

Начав экспериментировать с NanoBSD, вы найдете ей массу применений. Помимо нестандартных устройств, с помощью NanoBSD я создаю пространство пользователя для клеток и бездисковых рабочих станций. Если вам нужна более функциональная версия FreeBSD, загружаемая со сменных носителей, обратите свое внимание на FreeSBIE.

Сменные носители с FreeSBIE

В сравнении с NanoBSD FreeSBIE использует иной подход к созданию многоцелевых инсталляций FreeBSD и преследует иные цели. Назначение FreeSBIE – создание загрузочных сменных носителей с работающей операционной системой, таких как CD, DVD и flash-диски большой емкости, тогда как назначение NanoBSD – удовлетворить потребности пользователей небольших устройств. Бо́льший объем дискового пространства придает FreeSBIE больше гибкости, которой так не хватает NanoBSD. Лично я предпочитаю создавать небольшие устройства с помощью NanoBSD, но многие применяют FreeSBIE для сборки загружаемых компакт-дисков, превращающих старые рабочие станции в серверы.

Работать с такими компакт-дисками не сложнее, чем загружаться с CD. Операционная система на компакт-диске должна идентифицировать аппаратное окружение, настроить установленное программное обеспечение и загрузить свежую работоспособную операционную систему. Если вам нужна подробная информация о FreeSBIE, зайдите на веб-сайт FreeSBIE по адресу *http://www.freesbie.org*. Далее мы сосредоточимся на сборке нашего собственного диска FreeSBIE.

Ядро и пользовательские приложения FreeSBIE собирает из исходного кода в каталоге */usr/src*, но также принимает и собранные пакеты из системы, в которой производится сборка. Это означает, что ваша система должна иметь достаточно непротиворечивый набор пакетов. Если ваша система построена с нарушениями и пакеты собраны с ошибками, вы не сможете использовать их для сборки загрузочного компакт-диска FreeSBIE.

Несмотря на то что в этом разделе постоянно упоминается загрузочный компакт-диск FreeSBIE, все, о чем будет сказано, в равной степени справедливо для дисков DVD и flash-устройств.

> **FreeSBIE и FreeBSD**
>
> FreeSBIE не интегрирована в FreeBSD. Хотя разработчики FreeSBIE упорно трудятся над поддержкой новых версий FreeBSD, они все еще отстают от основной разработки FreeBSD. В данной главе имеется в виду версия FreeSBIE 2, основанная на FreeBSD 6. Полагаю, поддержка FreeBSD 7 появится вскоре после выпуска FreeBSD 7. Собираясь начать работу с FreeSBIE, приготовьтесь к самостоятельным исследованиям и борьбе с неполадками.

Установка инструментов FreeSBIE

Самую свежую версию инструментов FreeSBIE можно получить из репозитария CVS по адресу *http://www.freesbie.org* или, если у вас есть обновленная версия коллекции «портов», их можно установить из «порта» */usr/ports/sysutils/freesbie*. Все инструменты FreeSBIE устанавливаются в каталог */usr/local/share/freesbie*.

В наборе инструментов FreeSBIE мало инструкций. Большая часть документации доступна только на веб-сайте FreeSBIE: *http://www.freesbie.org*.

Конфигурирование FreeSBIE

В каталоге */usr/local/share/freesbie/conf* вы найдете конфигурационный файл FreeSBIE *freesbie.defaults.conf*, используемый по умолчанию. Содержимое этого файла не должно редактироваться напрямую,

вместо этого нужно создать файл *freesbie.conf* в том же самом каталоге. Все определения в файле *freesbie.conf* имеют приоритет перед определениями в файле *freesbie.defaults.conf*.

Ниже приведен перечень параметров конфигурации FreeSBIE 2.0, значения которых обычно приходится изменять. Следующий пример показывает, как могло бы выглядеть содержимое файла *freesbie.conf*, не похожего на файл *freesbie.defaults.conf*. (Конфигурационный файл по умолчанию имеет специальный синтаксис, что позволяет переопределять значения параметров в файле *freesbie.conf*.)

Для первой сборки рекомендую вам изменять как можно меньше параметров. Было бы весьма заманчиво изменить некоторые параметры, но старайтесь все-таки придерживаться значений по умолчанию, поскольку они обеспечивают хорошие начальные результаты. Чрезмерное увлечение изменением параметров настройки может привести к появлению ошибок в процессе сборки. Как и во многих других случаях, я рекомендую изучать подводные камни FreeSBIE по одному.

`BASEDIR="/usr/local/freesbie-fs"`

Каталог, в который FreeSBIE установит свежесобранную систему, пакеты и нестандартные файлы.

`CLONEDIR="/usr/local/freesbie-clone"`

Этот каталог FreeSBIE использует как временное хранилище во время сборки образов.

`SRCDIR="/usr/src"`

Каталог с деревом исходного кода, используемого для сборки ядра и приложений.

`ISOPATH="/usr/obj/FreeSBIE.iso"`

В этот файл записывается полный ISO-образ.

`IMGPATH="/usr/obj/FreeSBIE.img"`

Полный путь к UFS-образу FreeSBIE, который можно скопировать на flash-диск.

`MAKEJ_WORLD="-j3"`

Этот параметр определяет аргумент `-j` утилиты make(1) для сборки приложений. Если сборка завершается с ошибками и вы хотите убедиться, что они не вызваны параллельной компиляцией сразу нескольких файлов, установите этот параметр равным `-j1`.

`MAKEJ_KERNEL="-j1"`

Этот параметр определяет, сколько потоков компиляции запустит утилитой make(1) при сборке ядра.

`MAKEOPT="-DNO_CLEAN"`

Если вам потребуется использовать какие-либо дополнительные параметры для сборки ядра и приложений, укажите их здесь.

`KERNELCONF="/usr/src/sys/i386/conf/CUSTOMFREESBIE"`

Полный путь к конфигурационному файлу ядра FreeSBIE. Я считаю, что конфигурация ядра, поставляемая вместе с FreeSBIE, хорошо подходит только в том случае, если вы используете ту же версию FreeBSD, что и разработчики FreeSBIE. Для версии FreeBSD 7.0 FreeSBIE использует следующие конфигурационные параметры, отсутствующие в конфигурации GENERIC:

```
options         GEOM_UZIP
options         GEOM_LABEL
options         VESA
options         SC_PIXEL_MODE
```

Кроме того, я удаляю поддержку всех контроллеров SCSI и RAID, так как мне действительно не требуется доступ к этим дискам. Если мне будет не нужен доступ к жестким дискам, я также удалю поддержку устройств atadisk, чтобы мой образ системы не смог отыскать жесткие диски IDE. Если вам потребуется использовать функции пакетного фильтра PF, добавьте соответствующие конфигурационные параметры ядра.

`MAKE_CONF="/etc/make.conf"`

Этот параметр определяет нестандартный файл *make.conf*, который должен использоваться для сборке FreeSBIE. Обратите внимание: в случае копирования пакетов из установленной системы любые специальные настройки в *make.conf* к этим пакетам применяться не будут. Вместо этого файла вы можете использовать файл *src.conf*.

`SRC_CONF="/etc/src.conf"`

Порядок сборки из исходных текстов можно настроить с помощью *src.conf*, как вы это делали при сборке обычной системы FreeBSD. Укажите в этом параметре полный путь к файлу *src.conf*, который следует использовать для сборки FreeSBIE.

`FILE_LIST="/home/mwlucas/freesbie-files.txt"`

Вы можете определить точный перечень файлов, устанавливаемых на диск FreeSBIE. В этом случае в образ будут установлены только файлы из списка. (Фактически это утверждение означает, что из `BASEDIR` в `CLONEDIR` будут скопированы только файлы, присутствующие в списке.) То есть вы должны составить на 100 процентов полный список файлов, обязанных находиться на загрузочном компакт-диске. По моему мнению, такая возможность более полезна для малых систем, подобных NanoBSD.

`PRUNE_LIST="/home/mwlucas/freesbie-prune.txt"`

Этот файл содержит список файлов и каталогов, которые включены в сборку, но должны быть удалены перед началом сборки образа компакт-диска. Все файлы, указанные в этом списке, будут удалены из каталога `CLONEDIR` до того, как начнется сборка образа из этого

каталога. Это может оказаться очень полезным для уменьшения размера компакт-диска.

`PKGFILE="/home/mwlucas/freesbie-packages"`

Этот файл содержит список пакетов, по одному пакету в строке, без указания информации о версиях. Все перечисленные пакеты будут установлены в образ диска. Если этот параметр не определен, список будет создан командой `make packageselect` и сохранен в файле */usr/local/share/freesbie/conf/packages*.

`EXTRA="customroot rootmfs etcmfs sound xautostart"`

В состав FreeSBIE входят разные подключаемые модули, реализующие различные функциональные возможности для загрузочного компакт-диска. Стандартные модули мы рассмотрим ниже.

`MINIMAL=YES`

Если определить параметр `MINIMAL`, FreeSBIE соберет минимально возможную версию FreeBSD. Результат такой сборки напоминает NanoBSD. Функция минимальной сборки надежно работает только с той же версией FreeBSD, которой пользуются разработчики FreeSBIE.

`NO_BUILDWORLD=YES`

Определяя этот параметр, вы тем самым сообщаете FreeSBIE о том, что не нужно собирать новые приложения пользователя. При этом в каталоге */usr/obj* у вас должны иметься свежесобранные версии приложений.

`NO_BUILDKERNEL=YES`

Этот параметр предписывает FreeSBIE не собирать новое ядро, а использовать ядро, собранное ранее и находящееся в каталоге */usr/obj*.

`MAKEOBJDIRPREFIX="/usr/freesbie/obj"`

Этот параметр можно использовать, чтобы определить отдельную область сборки FreeSBIE, но это не рекомендуется. Многие сценарии командного интерпретатора предполагают, что свежесобранные приложения и ядро находятся в каталоге */usr/obj*.

`NO_COMPRESSEDFS=YES`

Этот параметр предписывает FreeSBIE использовать несжатую файловую систему в ISO-образе.

Подключаемые модули FreeSBIE

В состав FreeSBIE входят дополнительные подключаемые модули (plug-ins), позволяющие более тщательно настроить образ диска. Сборки, создаваемые FreeSBIE по умолчанию, вполне пригодны к использованию, однако дополнительные модули помогут вам добавить или убрать некоторые функциональные возможности компакт-диска. Подключаемые модули устанавливают в образ загружаемого диска дополнительные конфигурационные сценарии или активизируют различные

функции автоматизации операций. Все сценарии модулей размещаются в каталоге */usr/local/share/freesbie/extra*. Файл *README* содержит описание всех модулей, имеющихся по умолчанию.

Перечислите все нужные вам модули в параметре `EXTRA` в файле *freesbie.conf*. Ниже я перечислил те модули, которые считаю наиболее полезными или интересными.

adduser

Модуль adduser добавляет в образ диска пользователя freesbie. Вы можете указать свое имя пользователя с помощью параметра `FREESBIE_ADDUSER` в файле *freesbie.conf*.

autologin

Модуль autologin служит для автоматического входа в систему во время загрузки под учетной записью пользователя freesbie.

comconsole

Модуль comconsole добавляет в сборку поддержку последовательной консоли в качестве второй консоли. Если в файле *freesbie.conf* установить параметр `SERIAL_ONLY=YES`, будет собран образ, использующий только последовательную консоль. Сделать это также можно, добавив в образ нестандартный файл *loader.conf* с помощью модуля customroot.

customroot

Модуль customroot позволяет добавлять на компакт-диск любые файлы. Эти файлы, расположенные в каталоге */usr/local/share/freesbie/extra/customroot*, будут скопированы в новый образ диска. Создайте дополнительные каталоги, чтобы поместить файлы в требуемое место на компакт-диске. Например, если требуется записать на компакт-диск свой файл */etc/rc.conf*, поместите его в */usr/local/share/freesbie/extra/customroot/etc/rc.conf*. Добавив свой файл с настройками */usr/local/share/freesbie/extra/customroot/var/named/etc/namedb/named.conf*, вы получите легко заменяемый вторичный сервер имен, который загружается с носителя, доступного только для чтения. Любые файлы, добавляемые этим модулем, будут перезаписаны поверх любых других файлов, присутствующих в исходной сборке FreeSBIE.

customscripts

В процессе сборки FreeSBIE, непосредственно перед созданием ISO-образа, этот модуль запускает все сценарии, находящиеся в каталоге */usr/local/freesbie/extra/customscripts*. С его помощью вы сможете автоматически добавлять пользователей или редактировать файлы.

etcmfs

Этот модуль предписывает системе использовать для каталога */etc* файловую систему в памяти.

l10n.sh

Этот модуль предоставляет пользователям компакт-диска возможность выбирать национальный набор символов. Этот модуль будет полезен для неанглоязычных пользователей.

mountdisks

Если этот модуль включен, FreeSBIE будет монтировать все разделы с файловыми системами UFS, FAT или NTFS, найденные на жестких дисках в системе, загруженной с компакт-диска.

pf

Этот модуль создает и активизирует во время загрузки простой набор правил PF «никого не впускать, всех выпускать».

rootmfs

Этот модуль создает корневую файловую систему в памяти. Хотя пользователь получает возможность изменять все, что находится в корневом каталоге, эти изменения исчезнут при перезагрузке.

sound

В ходе загрузки с компакт-диска будет предпринята попытка автоматически обнаружить звуковую карту и установить нужный драйвер.

swapfind

Если в системе, запущенной с компакт-диска, присутствуют какие-либо разделы свопинга, FreeSBIE отыщет и будет использовать их. Разумеется, в этом случае будет производиться запись информации на жесткий диск, что порой может оказаться нежелательным.

varmfs

В ходе загрузки с компакт-диска для каталога */var* будет использована файловая система в памяти. Это позволит вести локальные протоколы, но они будут уничтожены при перезагрузке.

xautostart

В ходе загрузки с компакт-диска будет запущена X Window System.

xconfig

В процессе загрузки будет автоматически выполняться конфигурирование X. Этот модуль желательно использовать в паре с модулем xautostart.

xconfigure-probe

FreeSBIE будет использовать альтернативный метод автоматической настройки X во время загрузки. (X предлагает несколько методов настройки дисплея, такое деление команда разработчиков FreeSBIE не смогла разрешить.)

Выбор пакетов

Вы можете создать текстовый файл со списком всех пакетов, которые требуется установить на загрузочный компакт-диск, или использовать команду `make packageselect` в каталоге */usr/local/share/freesbie*. Эта команда создаст список всех пакетов, установленных в системе, где выполняется сборка, и предложит выбрать пакеты для установки на компакт-диск. При выборе пакетов все зависимости разрешаются автоматически.

Сборка образа FreeSBIE

Чтобы собрать компакт-диск FreeSBIE с учетом вашей конфигурации, просто введите команды:

```
# cd /usr/local/share/freesbie
# make iso
```

FreeSBIE выполнит полную сборку системы FreeBSD, удалит ненужное и добавит желаемое, установит пакеты, выполнит сценарии настройки и сожмет полученное в образ компакт-диска.

Если в место образа ISO вам нужно создать образ flash-диска, используйте команду `make flash`. FreeSBIE попросит указать, какое устройство flash вы предполагаете использовать, инициализирует его, соберет приложения и установит их на указанное устройство.

Пересборка FreeSBIE

FreeSBIE, как и «порты», использует файлы, имена которых начинаются с точки, чтобы указать, какие этапы были выполнены. По умолчанию эти файлы помещаются в каталог */usr/obj/usr/local/share/freesbie*.

Если требуется пересобрать образ, не пересобирая при этом приложения, удалите файлы для пройденных этапов.

Это была последняя остановка в нашем исследовании потайных уголков FreeBSD. Теперь давайте посмотрим, что делать, когда дела идут очень, очень, *очень* плохо.

21

Аварии и паника системы (и системных администраторов)

Одна из замечательных особенностей FreeBSD – стабильность этой системы; «синий экран смерти» появляется лишь при активизации заставки. На самом деле, прошел почти год, прежде чем я осознал, что аварии на машине FreeBSD могут происходить не только из-за плохих аппаратных средств.

Во FreeBSD могут случаться аварии или *паника*, однако систему можно легко восстановить, поэтому впадать, извините, в панику не стоит. Во время паники можно удаленно подключиться к консоли и инициировать перезагрузку. Операционная система FreeBSD предоставит инструменты, с помощью которых можно точно определить, что случилось, и заодно обеспечит вас подробной отладочной информацией. Даже если непонятно, что делать с этой информацией, можно подготовить отчет о проблеме и обсудить этот вопрос с командой разработчиков FreeBSD.

Что вызывает панику?

Когда система впадает в панику? Паника – это выбор, который делает ядро при виде неразрешимых проблем. Если система оказывается в ситуации, с которой не может справиться, либо внутренние проверки на непротиворечивость говорят о сбое, система сигнализирует о неполадках. Рабочие версии FreeBSD чрезвычайно устойчивы, но все же это может случиться. Самый легкий способ вызвать панику – сделать что-нибудь ненормальное, например, вынуть жесткий диск, не допускающий горячую замену во время работы. Паника – это не необычное явление для версии -current; паника возникает нечасто, но замечу, что она вовсе не редкость.

FreeBSD – сложная система, и ни аристократическое происхождение, ни процесс разработки с открытым кодом не могут защитить ее от всех

ошибок. К счастью, наследие системы и процесс разработки предоставляют вам необходимые инструменты, способные собирать информацию для специалистов, занимающихся отладкой. Натыкаясь на непонятный код ошибки, вы вскоре поймете, что для кого-то в этой строке странных символов есть некий смысл.

Рекомендую перед запуском каждой системы в работу готовить ее так, чтобы она формировала дамп паники. FreeBSD формирует дамп паники по умолчанию, но если вы переконфигурируете свой сервер или жесткий диск разбит на разделы особым образом, вы должны убедиться, что дамп паники будет создаваться. Эта предосторожность может оказаться ненужной на большинстве ваших серверов, но подготовка займет совсем немного времени. Когда один из ваших серверов впадет в панику, вы рады будете иметь под рукой полную информации о панике.

Что представляет собой паника?

Когда система паникует, она останавливает выполнение всех программ и прослушивание сети. При использовании основного выпуска FreeBSD, или версии -stable, система автоматически перезагрузится после паники. Не все необъяснимые перезагрузки являются паникой – если у вас, например, электропитание подается со сбоями или некачественная память, аппаратный сбой может привести к перезагрузке без вывода сообщений в протокол или на консоль. Если система запаниковала, в протокол ошибок будут выведены сообщения, содержимое которых зависит от типа паники. Если система настроена так, чтобы вместо перезагрузки просто выводить сообщение о панике на экран, то на консоли появляется примерно следующее:

```
panic: _mtx_lock_sleep: recursed on non-recursive mutex ahc_lock @ /usr/src/
sys/dev/aic7xxx/aic7xxx_osm.h:203

cpuid = 0
KDB: enter: panic
[thread pid 12 tid 100002 ]
Stopped at kdb_enter+0x32: leave
db>
```

Единственное, что в этом сообщении кажется мне более или менее понятным, находится в первой строке. Я знаю, что mutex (мьютекс) – это тип блокировки в ядре (глава 12), а моя карта SCSI использует драйвер ahc. Это сообщение действительно имеет определенный смысл, так как я вызвал панику на этой машине с системой версии -current, запустив fdisk(8) для диска SCSI.[1]

[1] «Думаю, мне не придется сегодня работать над главой 18 – видимо, FreeBSD желает, чтобы я писал главу 21». Я оказался здесь, задавшись вопросом, как получить хорошую панику для этой главы.

Строка символов db> внизу – это приглашение отладчика к вводу. Нажмите Enter пару раз и вы увидите, что можете вводить команды. Это не команды UNIX, но с их помощью вы можете получить информацию о системе.

Ответные действия при панике

Если система впала в панику, прежде всего надо скопировать сообщения о неполадках. Поскольку в случае паники FreeBSD стандартные способы копирования данных на вашей машине недоступны – система не позволит войти в нее с помощью SSH и запустить даже такие простые команды, как script(1). Консоль может быть намертво заблокирована либо можно «застрять» в отладчике. Но в любом случае вы должны иметь сообщение об ошибке.

FreeBSD не всегда автоматически перезагружалась после паники, раньше в этом случае просто выводилось сообщение. Впервые увидев панику, я стал искать бумагу и карандаш. Нашел старый конверт и огрызок карандаша, которым можно было писать под определенным углом, и пополз между серверной стойкой и необработанной кирпичной стеной. Я удерживал равновесие, одной рукой не давая упасть шестидюймовому черно-белому монитору, а другой прижимая к стене конверт. Очевидно, в экстренных ситуациях у меня может отрастать третья рука, так как мне все же удалось записать сообщение о неполадках на конверте. Наконец, поцарапанный и согнутый в три погибели, я задним ходом выбрался из-за стойки и победоносно отправил это сообщение по электронной почте. Уж конечно, первоклассные разработчики FreeBSD посмотрят на всю эту абракадабру и точно скажут, что случилось.

Вскоре я понял, что у FreeBSD нет специалистов, способных решить мою проблему. Вместо этого я получил короткий ответ: «Можете прислать вывод обратной трассировки?» Когда я спросил, как это сделать, меня отправили читать справочное руководство (глава 1). К счастью, паника оказалась легко воспроизводима, единственное, что требовалось, – это воссоздать ошибку при входе клиента в систему. Остаток дня я провел в борьбе с последовательной консолью и дампами ядра.

Неприятность с сообщением о неполадках, записанным мной на конверте, заключалась в том, что оно предоставляло лишь крохотный отрывок всей истории аварии. По сути, сообщение было настолько непонятным, что напоминало наивное описание примет угнанного автомобиля: «красный, с царапиной на крыле». Если вы не сообщите марку автомобиля, его модель, номер двигателя и номерной знак, то полиция недалеко продвинется. Точно так же без дополнительной информации от аварийного ядра разработчики FreeBSD не смогут выявить причины неполадок.

Однако есть простой путь решить эту проблему: подготовить сервер к возможной панике до того, как она произойдет. Проведите надлежа-

щую настройку при установке сервера. Сейчас я конфигурирую все свои машины для надлежащей обработки аварийных ситуаций, прежде чем запустить их в работу. Таким образом, при возникновении аварии вы автоматически получите отладочную информацию. Эта идея может показаться оригинальной, безусловно, ей не придается особое значение в документации FreeBSD, однако есть смысл подготовиться к аварии. Если она никогда не случится – что ж, не на что и жаловаться. Если же система впадет в панику, вы готовы предоставить разработчикам FreeBSD полный отладочный дамп без проблем.

Подготовка

Анализировать панику можно разными способами; я предпочитаю записывать дамп памяти на диск для дальнейшей спокойной обработки. Вам необходимо пространство свопинга (область подкачки), размером немного превосходящее память, занимаемую ядром. Несмотря на то что вам может потребоваться полный дамп оперативной памяти, в большинстве случаев дамп ядра FreeBSD 7.0 занимает меньше половины гигабайта. Если вы используете нечто требующее выделения большого объема памяти в ядре, например экспериментальную поддержку файловой системы ZFS, размер дампа ядра увеличится. Если область подкачки недостаточно велика, надо либо переустановить систему, либо добавить жесткий диск с областью подкачки достаточного размера. (Помните, что я говорил об этом в главе 2? Вы еще не жалеете, что не слушали меня тогда?) Наличие в разделе */var* свободного пространства, достаточного для дампа ядра, полезно, но необязательно.

Получение аварийного дампа

Процесс получения аварийного дампа примерно таков: если система сконфигурирована правильно, то при аварии она сохранит копию оперативной памяти. Эта копия называется *дампом*. Система не может сохранить дамп в файле. С одной стороны, потому что при аварии ядро ничего не знает о файлах, с другой – файловая система может быть повреждена или попытка записи может повредить ее. Однако ядро знает о разделах, поэтому оно может записать дамп в раздел. Проще всего записать дамп в раздел свопинга, предназначенный для временного хранения содержимого памяти. По умолчанию FreeBSD сохраняет дамп в первом разделе свопинга, размещая его как можно ближе к концу раздела. Как только дамп будет сохранен, система перезагрузит компьютер.

Во время перезагрузки FreeBSD активизирует раздел свопинга и проверяет чистоту файловых систем. После паники почти наверняка разделы повреждены. Возможно, они журналируются или предусматривают запуск fsck(8) в фоновом режиме, но система не может гарантировать это. FreeBSD должна включить свопинг до запуска fsck(8), потому что этой программе может потребоваться пространство свопинга. Будем надеяться, что в системе достаточно памяти, чтобы программе

fsck(8) не потребовался свопинг, а если он необходим, то будем надеяться, что пространство свопинга достаточно велико, чтобы избежать перезаписи файла дампа, скрытого в конце раздела свопинга.

Как только FreeBSD обретет файловую систему, где можно будет сохранить дамп памяти, она запустит программу savecore(8), которая скопирует его в подходящую файловую систему, удалит дамп из пространства свопинга и продолжит перезагрузку. Теперь у вас есть файл с дампом ядра, который можно проанализировать.

Настройка вывода аварийного дампа

По умолчанию FreeBSD сохраняет дамп в первом доступном разделе свопинга и при каждой перезагрузке проверяет пространство свопинга на наличие дампа. При необходимости такое поведение системы можно изменить.

Очевидно, вам может потребоваться сохранять дамп в другом разделе свопинга, особенно после добавления нового жесткого диска специально для того, чтобы создать раздел свопинга, способный хранить дамп. Определите устройство дампа с помощью переменной `dumpdev` в файле */etc/rc.conf*:

```
dumpdev="/dev/ad4s1b"
```

Не забудьте перечислить в файле */etc/fstab* всех разделы свопинга.

Программа savecore(8) автоматически сохраняет дамп ядра в каталоге */var/crash*. Однако если раздел */var* имеет недостаточный размер для хранения дампа, укажите другой каталог в переменной `dumpdir` в файле *rc.conf*:

```
dumpdir="/usr/crash"
```

Последовательные консоли и паника

Хотя последовательная консоль не является абсолютно необходимой для выяснения причин паники, тем не менее она может оказаться бесценной, когда придется столкнуться с зависшей машиной. Способность фиксировать все происходящее с помощью программы script(1) делает последовательную консоль ценным инструментом. В случае удаленной машины, последовательная консоль превращается в насущную потребность. Если вы действительно хотите подготовиться к панике, снабдите все машины последовательными консолями или хотя бы сдвоенными консолями. Если это возможно, протоколируйте вывод последовательных консолей – это позволит вам получить сообщение о панике, даже если система не настроена на сохранение дампа.

Несмотря на то что savecore(8) обладает некоторыми дополнительными возможностями, такими как возможность сжатия файла, в современных системах они обычно не используются.

Теперь, имея отладочное ядро и устройство сохранения дампа, вы готовы к аварийным ситуациям, насколько это возможно.

Отладка ядра

В процессе сборки ядра FreeBSD в него включается большой объем отладочной информации. Вы найдете первичную отладочную информацию в файлах *символов* (*symbols*), обеспечивающих отображение между машинным и исходным кодом. Кроме того, это отображение содержит полный список номеров строк исходного кода, поэтому разработчик может точно выяснить, где произошли неполадки. Без этой информации разработчику пришлось бы соотносить дамп ядра и исходный код вручную. Примерно так собирают пазл из миллиона кусочков, когда под рукой нет картинки-образца и неизвестно, все ли кусочки в наличии. Это нелегкая задача. Она еще труднее, если разработчик, который должен предложить решение, является добровольцем. Символы имеют важное значение.

Наличие отладчика в ядре облегчает процесс отладки, если у вас есть доступ к консоли или последовательная консоль к удаленной машине. Отладку можно вести полностью в автономном режиме, но иногда наличие отладчика упрощает поиск причин паники.

Чтобы сохранить отладочную информацию и включить отладчик в ядро, добавьте в конфигурацию ядра следующие строки:

```
makeoptions    DEBUG=-g
options        KDB
options        KDB_TRACE
options        DDB
```

После добавления опции `DDB` в ядро FreeBSD больше не будет автоматически перезагружаться после паники. Что вам больше подходит – чтобы система сама перезагружалась после паники или ждала вашего вмешательства? В случае удаленной системы вы почти наверняка захотите сохранить дамп и автоматически перезагрузить машину после паники, но если вы находитесь рядом с консолью, то, возможно, захотите выполнить перезагрузку вручную.

Для автоматической перезагрузки используйте параметр ядра `KDB_UNATTENDED`:

```
options        KDB_UNATTENDED
```

Этот параметр предписывает FreeBSD автоматически перезагрузиться после паники, даже если в конфигурации ядра присутствует параметр `DDB`. После паники на экране появится сообщение, предлагающее нажать клавишу пробела, чтобы войти в отладчик; если этого не сделать, система перезагрузится через 15 секунд.

Если случилась паника: создание дампа вручную

Предположим, что отладчик уже присутствует в ядре, раздел свопинга в состоянии вместить дамп и машина сконфигурирована так, что не предусматривает автоматическую перезагрузку после паники. Однажды одна из ваших машин исчезла из сети. Вы взглянули на консоль, но видите там только какое-то непонятное сообщение, ниже которого светится приглашение к вводу db>. Или, может быть, вы нажали клавишу Enter на последовательной консоли только для того, чтобы увидеть приглашение к вводу db>. FreeBSD ждет от вас команды на перезагрузку после паники.

Пока машина находится в отладчике, у вас есть возможность получить кое-какую отладочную информацию. Если на экране имеется сообщение – скопируйте его. Если нет – дайте отладчику команду повторить его:

```
db> panic
panic: from debugger
cpuid = 0
Uptime: 2m38s
panic: _mtx_lock_sleep: recursed on non-recursive mutex ahc_lock @ /usr/src/
sys/cam/cam_periph.h:182
...
```

Отладчик повторно выведет для вас сообщение о панике.

Вы также можете получить дополнительную информацию о панике:

```
db> trace
Tracing pid 12 tid 100002 td 0xc2117c00
kdb_enter(c0a908ed,0,3e9,c0b9fe2c,0,...) at kdb_enter+0x32
panic(c0a8f7ce,c0a54565,c0a4ba56,b6,c228ca30,...) at panic+0x124
_mtx_lock_sleep(c228ca30,c2117c00,0,c0a4ba56,b6,...) at _mtx_lock_sleep+0x47
_mtx_lock_flags(c228ca30,0,c0a4ba56,b6,780,...) at _mtx_lock_flags+0xef
...
```

Сообщение может содержать несколько страниц. Если вы работаете на последовательной консоли, убедитесь, что зафиксировали всю информацию.

Перед перезагрузкой сделайте дамп оперативной памяти.

```
db> continue
Physical memory: 243 MB
Dumping 62 MB: 47 31 15
Dump complete
= 0xf
```

FreeBSD даст обратный отсчет, показывая, сколько памяти осталось сохранить в дампе, и по его окончании перезагрузит систему.

Если дамп не нужен, можно просто перезагрузить машину:

```
db> reset
```

Сервер перезагрузится и приступит к работе.

Если был установлен параметр KDB_UNATTENDED, машина перезагрузится без всех этих сложностей, как если бы вы ввели команду continue.

Применение аварийного дампа

Если вы разработчик ядра, то в этом месте перестанете меня слушать, полагаясь на свой опыт отладки. Однако если вы системный администратор, то, вероятно, недостаточно хорошо владеете языком C и не знакомы с внутренним устройством ядра, чтобы надеяться на его успешную отладку в сложных случаях. Поэтому сосредоточим внимание на извлечении информации, помогающей разработчику идентифицировать проблему.

После создания дампа можно найти файлы *vmcore.0* и *info.0* в каталоге */var/crash*. (Каждый последующий дамп получает следующий по порядку номер.) Файл *info.0* содержит базовую информацию о дампе, например раздел, откуда был извлечен дамп, дату создания и успешно ли он был создан. Файл *vmcore.0* – это образ памяти ядра в момент аварии. Это полный отчет, о чем думало ядро в момент возникновения паники.

Если аварийные ситуации будут повторяться часто, то аварийные дампы могут переполнить раздел */var*. При многократном повторении аварийной ситуации определенного типа вам нужен только один дамп этого типа. Дампы прежних версий FreeBSD бесполезны и могут быть удалены.

Получение обратной трассировки

Зайдите в каталог, где находится запаниковавшее ядро. Это может быть ваше рабочее ядро или, возможно, тестовое ядро. В каталоге ядра должны присутствовать два файла для каждого модуля, один файл с именем модуля, а другой с расширением *.symbols*. Файлы с расширением *.symbols* – это отладочные образы. Перейдя в каталог с ядром, запустите команду kgdb(1), передав ей имя файла *kernel.symbols* и полный путь к файлу с дампом памяти *vmcore*:

```
# cd /boot/kernel.panicked/
# kgdb kernel.symbols /var/crash/vmcore.0
```

kgdb(1) выведет краткую информацию об авторских правах на отладчик gdb(1) и о лицензии (kgdb(1) – это, действительно, модифицированный отладчик gdb (1), специально настроенный для отладки ядра). Затем вы увидите копию сообщения о панике и в конце получите от kgdb(1) приглашение к вводу. Здесь вы, наконец, сможете получить требуемую обратную трассировку.

```
(kgdb) backtrace
#0  doadump () at pcpu.h:195
#1  0xc048c629 in db_fncall (dummy1=-875263368, dummy2=0, dummy3=524358,
```

```
    dummy4=0xcbd489e4 "РиНА") at /usr/src/sys/ddb/db_command.c:486
#2  0xc048cb95 in db_command_loop () at /usr/src/sys/ddb/db_command.c:401
#3  0xc048e305 in db_trap (type=3, code=0) at /usr/src/sys/ddb/db_main.c:222
#4  0xc0772426 in kdb_trap (type=3, code=0, tf=0xcbd48b88)
    at /usr/src/sys/kern/subr_kdb.c:502
#5  0xc09faa2b in trap (frame=0xcbd48b88) at /usr/src/sys/i386/i386/
trap.c:620
#6  0xc09e053b in calltrap () at /usr/src/sys/i386/i386/exception.s:139
#7  0xc07725a2 in kdb_enter (msg=0xc0a908ed "panic") at cpufunc.h:60
#8  0xc074b954 in panic (
    fmt=0xc0a8f7ce "_mtx_lock_sleep: recursed on non-recursive mutex %s @
%s:%d\n") at /usr/src/sys/kern/kern_shutdown.c:547
#9  0xc07400b7 in _mtx_lock_sleep (m=0xc228ca30, tid=3255925760, opts=0,
    file=0xc0a51887 "/usr/src/sys/dev/aic7xxx/aic7xxx_osm.h", line=203)
    at /usr/src/sys/kern/kern_mutex.c:311
#10 0xc07402df in _mtx_lock_flags (m=0xc228ca30, opts=0,
    file=0xc0a51887 "/usr/src/sys/dev/aic7xxx/aic7xxx_osm.h", line=203)
    at /usr/src/sys/kern/kern_mutex.c:187
...
```

Это полная запись всего, что делало ядро, в обратном порядке. Строка с номером 0 показывает, что отладчик вызвал функцию `doadump`, которая выполняет запись дампа на диск. Вслед за этой строкой мы видим различные вызовы отладчика. Строка 8 содержит само сообщение о панике, а слово `panic` отмечает момент, когда сервер начал падать.

В случае паники ядро вызывает функцию `panic` и иногда функцию `trap`. Вам могут встретиться варианты этих двух слов, например `db_trap`, `kdb_trap`, `calltrap` и т. д., однако нас интересуют только простые вызовы `trap` или `panic`, без всяких украшений. Поищите эти функции в выводе kgdb(1). В предыдущем примере вызов функции `panic` присутствует в строке 8. Строки 3, 4, и 5 содержат функции, которые отладчик вызывает для точного выяснения произошедшего и выбора последующих действий.

Строка 9 показывает, что произошло непосредственно перед тем, как система решила удариться в панику в строке 8. В строке 9 мы видим:

```
#9 0xc07400b7 in _mtx_lock_sleep (m=0xc228ca30, tid=3255925760, opts=0,
   file=0xc0a51887 "/usr/src/sys/dev/aic7xxx/aic7xxx_osm.h", line=203)
   at /usr/src/sys/kern/kern_mutex.c:311
```

Шестнадцатеричные числа говорят не так уж много, но очень похоже на сообщение о панике. Система FreeBSD запуталась при выполнении кода, начинающегося со строки 203 файла */usr/src/sys/dev/aic7xxx/aic7xxx_osm.h*. Эта информация подсказывает разработчику, где искать причины неполадок. Однако мы можем заглянуть немного глубже. Используйте команду `up` и номер строки, которая вас заинтересовала:

```
(kgdb) up 9
#9 0xc07400b7 in _mtx_lock_sleep (m=0xc228ca30, tid=3255925760, opts=0,
   file=0xc0a51887 "/usr/src/sys/dev/aic7xxx/aic7xxx_osm.h", line=203)
```

```
    at /usr/src/sys/kern/kern_mutex.c:311
311                 KASSERT((m->lock_object.lo_flags & LO_RECURSABLE) != 0,
```

Здесь мы видим фактическую строку кода, который вызвал панику в системе. На данном этапе я должен отдать ядро на милость разработчиков. Включив эту информацию в отчет о проблеме, вы уменьшите объем переписки, однако вполне возможно, что разработчик просто скажет: «Ой!» – и отправит вам исправления без объяснения причин неполадки. Чтобы покинуть отладчик kgdb(1), введите команду `quit`:

```
(kgdb) quit
```

Если вы собираетесь перестраивать ядро, сохраните запаниковавшее ядро для дальнейших исследований. Вы можете исследовать дамп ядра только с ядром, которое запаниковало.

Все вышеупомянутое – довольно хорошее начало для создания отчета о проблеме. Вы можете точно видеть, где система остановилась и на какой строке кода это произошло. Заинтересовавшиеся разработчики наверняка ответят вам и сообщат, какие еще команды следовало бы ввести в командной строке kgdb, но как бы то ни было, вы находитесь на правильном пути решения проблемы и оказания помощи ребятам из FreeBSD.

vmcore и безопасность

В файле *vmcore* присутствует все содержимое памяти ядра на момент возникновения паники. Сюда может входить различная секретная информация. Злоумышленник может воспользоваться этими данными, чтобы проникнуть в вашу систему. В своем письме разработчик FreeBSD может попросить у вас копию файла *vmcore* и само ядро. Для этого у него могут быть вполне разумные причины: отладка пройдет легче, а количество писем с уточняющими вопросами значительно сократится. Тем не менее внимательно продумайте возможные последствия предоставления такой информации. Если вы не знаете специалиста, который попросил вас об этом, или не доверяете ему, то не следует посылать этот файл!

Однако если паника воспроизводима, можно произвести «холодную» перезагрузку системы, войти в однопользовательский режим и немедленно инициировать панику. Если в системе не была запущена ни одна программа, содержащая конфиденциальную информацию, и никто не набирал пароли в системе, то в дампе не окажется опасной информации. Таким образом, воспроизведение паники в однопользовательском режиме позволит сгенерировать «чистый», свободный от секретной информации файл дампа памяти. Загрузите систему в однопользовательском режиме и введите команды:

```
# mount -ar
# /etc/rc.d/dumpon start
# команда_которая_вызвала_панику
```

Первая команда монтирует файловые системы «только для чтения», поэтому после паники не понадобится запускать fsck(8). Вторая команда указывает системе, куда поместить дамп. Последней следует команда, вызвавшая панику в системе. Чтобы вызвать панику, может потребоваться не одна команда, но в большинстве случаев мы получим чистый дамп. Если для воспроизведения паники требуется загрузить конфиденциальную информацию, то эта информация будет присутствовать в дампе.

Предоставление отчета о проблеме

Вы можете возразить, что эту тему следовало бы обсудить в главе 1 «Как получить помощь». Название *отчет о проблеме* (*Problem Report, PR*) впечатляет, не правда ли? Своим отчетом о проблеме вы не только сообщаете о своей проблеме, но и доказываете, что это у *FreeBSD* проблемы. Да, я сказал, *доказываете*. Не то чтобы FreeBSD тут ни при чем, пока вина не доказана, но вы должны предоставить полный отчет о проблеме, чтобы доказать свое утверждение. Если в отчете нет достаточных доказательств, он будет закрыт с краткой пометкой: «не ошибка» или «бесполезный отчет». Лучший форум, где вы сможете позвать на помощь, – это поисковая система, или даже список рассылки *FreeBSD-questions@FreeBSD.org*.

В виде отчета вы также можете представить Проекту FreeBSD свои предложения по желаемым улучшениям. Ключевое слово здесь – *улучшения*, а не *желаемые*. Такой тип отчета о проблеме должен

Что не является отчетом о проблеме?

Любые вариации на тему «Я не знаю, как это получилось» не должны входить в отчет о проблеме. Сюда же: «FreeBSD работает совсем не так, как мне хотелось бы» или «Когда я делаю какую-нибудь глупость, происходит какая-нибудь неприятность». Если вы сломаете руку, больница примет вас независимо от того, какую глупость вы совершили, но команда FreeBSD намного разборчивее. Точно так же отчетом о проблеме не являются любые вариации на тему «Меня не устраивают версии программ, включенных в состав FreeBSD». Проект FreeBSD, например, не спешит добавить обновление компилятора до самой последней версии в ту же секунду, как только он появится, и обновление для данной конкретной версии компилятора может так *никогда* и не появиться. Подобные решения всегда основаны на веских причинах. Если вы открыли отчет о проблеме, только чтобы поплакаться на эту тему, в ответ вы ничего не получите, разве только комментарий «Устанавливайте из портов», и отчет тут же будет закрыт.

содержать фактический код с результатами его тестирования и другую необходимую информацию.

Наконец, отчет о проблеме подразумевает готовность к сотрудничеству. Отправляя свой отчет, вы тем самым показываете свою готовность трудиться вместе с разработчиками FreeBSD над решением проблемы. Это может означать наложение заплаток, использование различных команд или запуск команд отладки и отправку вывода разработчикам. Представляя отчет о проблеме, не следует ждать ответа «Исправлено, сделайте так-то и так-то» – это просто нереально. Если вы не готовы работать с командой разработчиков, не создавайте отчет. Конечно, наличие достаточного объема информации в отчете поможет решить проблему гораздо быстрее. Если вы сможете включить в отчет все, что необходимо разработчику, ответ придет намного быстрее.

Повторяйте за мной: «Свободное программное обеспечение. Бесплатная поддержка». Помните это, когда ваш драйвер карты RAID заставляет жесткие диски отбивать барабанную дробь.

И, наконец, было бы лучше, если при подаче отчета о проблеме у вас была установлена последняя версия FreeBSD. Может, кто-то и заглянет в отчет о проблеме для версии FreeBSD, вышедшей один или два выпуска тому назад, но никто не будет смотреть отчет для FreeBSD 2.2-stable.

Перед подачей отчета о проблеме

В идеале, вам никогда не придется заполнять отчет о проблеме. Правильно созданный отчет – это большой труд не только для вас, но и для разработчиков. В проекте FreeBSD имеется отдельный список рассылки, который содержит базу данных с примерами отчетов и может использоваться как руководство по их составлению. Отправляя письмо в почтовую рассылку FreeBSD, вы объявляете о своей беде тысячам людей; создавая отчет о проблеме, вы не только объявляете о своей беде тысячам людей, но и *требуете*, чтобы они отреагировали на это. Прежде чем отправлять отчет о проблеме, вы должны быть абсолютно уверены, что это нужно и вам, и проекту FreeBSD.

Для начала представьте себе, что ваша проблема не уникальна, и поищите ее решение в обычных ресурсах FreeBSD. Посмотрите FAQ и Руководство. Поищите в Интернете и в архивах почтовой рассылки тех, кто уже сталкивался с подобной проблемой. Загляните в систему отчетов, нет ли там открытого отчета по этой проблеме. Затем задайте вопрос в почтовой рассылке *FreeBSD-questions@FreeBSD.org*, возможно, кто-то уже оказывался в подобной ситуации. Может быть, решение проблемы уже на подходе, или вам все же придется открыть новый отчет? Вопросы о вашей проблеме, которые будут вам задавать, окажут неоценимую помощь в устранении этой проблемы и в создании отчета.

Прежде чем открыть отчет, соберите всю информацию, которая может быть полезна. Сюда входят:

- Полный протокол загрузки системы
- Версия системы
- Изменения в конфигурации ядра (если таковые имеются)
- Вывод отладчика

Поддается ли проблема воспроизведению? Чтобы исследовать проблему, разработчикам нужно воспроизвести эту ситуацию. Если ваш сервер начал напевать мелодии в 3 часа ночи – это проблема. Если это случилось всего один раз, и вы не можете воспроизвести ситуацию, то лучше держите рот на замке, и никто не заподозрит, что вы псих. Однако если это происходит всякий раз, когда вы запускаете определенную комбинацию команд на определенных аппаратных средствах, проблему можно будет проверить и исследовать – и либо она будет решена, либо какая-нибудь студия звукозаписи предложит вашему серверу подписать контракт.

Для работы с PR система FreeBSD использует GNATS – хорошо зарекомендовавшую себя систему отслеживания ошибок. И хотя есть более новые и блистательные системы, ни одна из них не удовлетворяет требованиям Проекта FreeBSD. Прежде чем отправить свой отчет о проблеме, загляните в существующую базу отчетов, чтобы увидеть, что включается в отчеты о проблеме. Найти базу данных отчетов можно по адресу *http://www.freebsd.org/support.html*. Посмотрите, есть ли в базе данных отчеты о проблемах, похожих на вашу? Есть ли какие-либо комментарии к этим отчетам? Как насчет вывода отладчика? Можете ли вы указать на другой отчет и сказать, что они могут быть связаны? Или, может быть, другой отчет точно соответствует вашей проблеме, и вы сможете использовать информацию из него?

Предположим, наступит день, когда вам действительно придется представить отчет о проблеме. Давайте посмотрим, что не надо туда записывать.

Плохие отчеты о проблемах

Чтобы лучше понять, каким должен быть хороший отчет, нужно посмотреть несколько отчетов, квалифицированных как плохие. Отыскать плохой отчет совсем не сложно, и тем не менее я отобрал один и немного изменил его, чтобы никого не поставить в затруднительное положение (мне повезло, автор отчета только что прочел эти строки!). Я натолкнулся на этот отчет о проблеме несколько дней назад:

> При загрузке FreeBSD 6,2-REL с образа ISO я не могу получить экран «Welcome to FreeBSD!» с пунктами меню. Меню загрузки зависает и при каждом обновлении экрана оно останавливается на 10. Какие бы клавиши я ни нажимал, загрузка не продолжается. Если я нажимаю сразу несколько клавиш, я в конечном итоге получаю панику. Если загрузка с того же самого образа ISO происходит в VMWare, появляется обратный отсчет времени, и я получаю возможность

установить его в раздел диска. Я также пробовал включать/отключать в BIOS поддержку клавиатуры USB, но без успеха.

Отчет включает в себя в себя номер модели материнской платы, клавиатуры и мыши. Инструкция по воспроизведению проблемы рекомендовала загрузить ISO-образ в аналогичной конфигурации аппаратного обеспечения.

Прежде всего, у читателя, безусловно, проблема с FreeBSD. (Могут ведь и у FreeBSD быть проблемы.) У меня нет никаких сомнений в том, что система действительно зависает при загрузке именно так, как описано. Но в отчете нет никаких доказательств и никакой диагностической информации. Воспроизведение этой ситуации не имеет никакого смысла; если бы каждый установочный компакт-диск с версией FreeBSD 6.2 вел себя подобным образом на таких распространенных аппаратных средствах, как эти, команда разработчиков никогда бы не приняла его к выпуску.

Описание марки и модели оборудования не является столь полезным, как можно было бы подумать. Изготовители иногда вносят изменения в чипсеты, не изменяя номер модели. Протокол загрузки содержит подробную информацию о фактическом аппаратном окружении, так что номер модели указывать не требуется.

Если бы я столкнулся с таким поведением, то сначала бы попытался записать второй компакт-диск. Возможно, первый диск был записан с ошибками. Если ситуация повторится, я бы загрузил ISO-образ FreeBSD 6.1. Если бы попытка загрузиться с этого образа также оказалась неудачной, я бы обратился к почтовой рассылке *-questions@* за получением «дальнейшей консультации перед подачей отчета о проблеме». Если бы мне удалось установить FreeBSD 6.1, а 6.2 не удалось, я включил бы в свой отчет о проблеме подробную информацию о ядре и протокол загрузки версии 6.1. Я также включил бы ссылку на архив почтовой рассылки, где обсуждалась моя проблема.

Как вы уже могли догадаться, никто не поддержал обсуждение этой проблемы. Ваша цель – создать отчет о проблеме настолько полный и убедительный, чтобы разработчики захотели работать с ним. Покажите, что вы готовы сотрудничать с ними.

В FAQ системы FreeBSD есть шутка Дэга Эрлинга С. Сморграва (Dag-Erling C. Smorgrav): «Сколько пользователей -current требуется, чтобы заменить электрическую лампочку»? Ответ – одна тысяча, сто и шестьдесят девять, при этом «три пользователя должны представить отчеты об этой проблеме, причем один из отчетов отнесен к категории doc и содержит единственное слово „темно“». Если проблема сводится к диагнозу «темно», то это плохой отчет о проблеме.

Хорошие отчеты о проблемах

Отправить свой отчет о проблеме можно с помощью веб-интерфейса, но я нахожу этот способ довольно неуклюжим. Если вам нравится графический интерфейс, можете попробовать gtk-send-pr (*/usr/ports/sysutils/gtk-send-pr*). Для создания отчетов я всегда использую send-pr(1). На мой взгляд, программа send-pr(1) позволяет сконцентрироваться на создании отчета и даже вернуться к нему спустя некоторое время, когда появится возможность заняться им. Работа над отчетом начинается с вывода шаблона формы в файл с помощью ключа `-P`:

```
# send-pr -P > problemreport.txt
```

Теперь можно открыть отчет об проблеме в текстовом редакторе. Вот образец шаблона:

```
To: FreeBSD-gnats-submit@freebsd.org
From: Michael W Lucas <mwlucas>
Reply-To: Michael W Lucas <mwlucas>
Cc:
X-send-pr-version: 3.113
X-GNATS-Notify:
>Submitter-Id:     current-users
>Originator:       Michael W Lucas
>Organization:     <organization of PR author (multiple lines)>
>Confidential:     no <FreeBSD PRs are public data>
>Synopsis:         <synopsis of the problem (one line)>
>Severity:         <[ non-critical | serious | critical ] (one line)>
>Priority:         <[ low | medium | high ] (one line)>
>Category:         <choose from the list of categories above (one line)>
>Class:            <[ sw-bug | doc-bug | change-request | update |
maintainerupdate ] (one line)>
>Release:     FreeBSD 7.0-CURRENT i386
>Environment:
System: FreeBSD pesty.blackhelicopters.org 7.0-CURRENT FreeBSD 7.0-CURRENT #0:
Wed May 21 13:29:50 EDT 2008 mwlucas@pesty.blackhelicopters.org:/usr/obj/usr/
src/sys/GENERIC i386
     <machine, os, target, libraries (multiple lines)>
>Description:
     <precise description of the problem (multiple lines)>
>How-To-Repeat:
     <code/input/activities to reproduce the problem (multiple lines)>
>Fix:
     <how to correct or work around the problem, if known (multiple lines)>
```

И веб-интерфейс, и текстовая форма имеют одни и те же поля; труднее заполнить форму правильно. Шаблон включает в себя несколько строк, начинающихся с `SEND-PR`, которые send-pr(1) удалит из формы в процессе отправки. Любой текст в угловых скобках (< и >) является комментарием, и send-pr(1) также удалит его. В угловых скобках шаблона перечислены допустимые варианты ответов. Выберите один из них.

Все, что говорится ниже, в основном применимо к отчетам, описывающим панику системы и представляющим собой наиболее сложный тип отчета о проблеме. Однако база данных отчетов FreeBSD используется также для отправки новых «портов» и новых особенностей. Порядок заполнения отчетов такого рода описывается в документации на вебсайте FreeBSD.

`To:` (Кому)

Это поле адреса электронной почты базы данных отчетов FreeBSD. Оставьте его без изменений.

`From`, `Reply-To:` (От кого)

Убедитесь, что строка `From` содержит действительный электронный адрес. Именно по этому адресу служба поддержки GNATS сообщит вам номер вашего отчета, а заинтересованные разработчики попытаются связаться с вами.

`cc:` (скрытая копия)

Возможно, вы хотите отправить копию отчета кому-то еще. Например, мой босс чувствует себя лучше, имея копии отчетов о проблемах, которые он считает критическими.

`X-send-pr-version, X-GNATS-Notify, Submitter-Id:`

Оставьте эти поля без изменений.

`Originator:` (Источник)

Это ваше имя. Обычно оно берется из системного окружения. Некоторые пользователи скрываются под псевдонимами, но я рекомендую указать здесь свое настоящее имя. Трудно подходить к серьезной проблеме с должным вниманием, если вы скрываетесь под именем «Mr. Ticklebottoms».

`Organization:` (Организация)

Это поле не используется, поэтому вы можете написать здесь все, что хотите.

`Confidential:` (Конфиденциально)

GNATS – это общедоступная база данных, и вы никогда не должны вносить конфиденциальную информацию в отчет о проблеме. Изменение значения в этом поле приведет к тому, что send-pr откажется посылать отчет. Если вы считаете, что обнаружили ошибку, относящуюся к безопасности, напишите письмо по адресу *security-officer@FreeBSD.org*. Однако не поступайте так только из-за того, что в отладочную информацию включен пароль root.

`Synopsis:` (Краткий обзор)

Это самая важная строка в отчете. Дайте краткое, однострочное описание проблемы. Разработчики именно по этой строке определяют, каким отчетам уделить свое внимание. Отчет с описанием «Моя система не работает!» будет закрыт или проигнорирован, а со-

общение «Паника при большой нагрузке, отладочный дамп прилагается» имеет больше шансов привлечь внимание опытных специалистов. Если у вас есть заплатка для устранения проблемы, поместите слово PATCH (заплатка) в начало описания, такому отчету почти гарантирован быстрый ответ. Учтите, они могут признать заплатку негодной, но они посмотрят ее.

Severity: (Сложность)

Это поле может иметь одно из трех значений: non-critical (некритично), serious (серьезно) или critical (критично). Выберите из них наиболее подходящее. Критичные проблемы затрагивают каждого пользователя FreeBSD. Серьезные проблемы затрагивают пользователей определенного драйвера устройства или определенного окружения. Все остальное является некритичным. Если вы заслужите репутацию пользователя, представляющего мелкие ошибки как критичные, очень скоро ваши отчеты будут игнорироваться. Здесь все построено на честности, и репутация значит намного больше, чем можно подумать.

Priority: (Приоритет)

Вы можете ввести одно из трех значений: low (низкий), medium (средний) или high (высокий). Поле приоритета долгое время использовалось неправильно, теперь потеряло свою значимость и не принимается в учет. Я рекомендую использовать значение low или medium.

Category (Категория)

Следует указать один из вариантов, перечисленных в области комментариев в верхней части формы отчета. Категории включают такие типы, как ports, www, docs, bin, kern и misc. Выберите категорию, к которой относится ваша проблема. Я обычно использую kern для отчетов о проблемах с ядром, bin для проблем, связанных с приложениями пространства пользователя, ports для описания проблем с «портами» и doc для проблем с документами.

Class: (Класс)

Это тип проблемы. Если вы столкнулись с аварийным отказом программы или системы, укажите класс sw-bug.

Release: (Выпуск)

send-pr(1) автоматически внесет тип вашей системы в поле Release. Если вы заполняете отчет не в той системе, где проявляются неполадки, укажите верное значение этого поля.

Environment: (Окружение)

Поместите сюда вывод команды uname -a. В это поле можно добавить дополнительную информацию о своей машине. Например, если машина – это веб-сервер, испытывающий тяжелые нагрузки, укажите это. Если у вас есть фрагмент конфигурационного файла, позволяющий воспроизвести панику, поместите его сюда. Раздел Environment

должен быть очень кратким. Возможно, для вас будет желательно скрыть имя машины по соображениям безопасности.

`Description:` **(Описание)**

Поле `Description` – это текстовый раздел, где в свободной форме можно подробно описать проблему. Не разглагольствуйте и не говорите бессвязно, а просто опишите, что случилось. Если можно получить дамп паники, сделайте это и разместите здесь вывод отладчика. Кроме того, разместите в этом поле конфигурацию ядра и подробный протокол загрузки.

`How-To-Repeat:` **(Как воспроизвести)**

Дайте здесь краткие инструкции о том, как воспроизвести проблему. Для некоторых отчетов вполне подойдет предложение: «Прочитайте архив рассылки FreeBSD-questions за неделю и посмотрите, как часто это встречается», особенно если речь идет об изменениях в документации. Более специфичные проблемы требуют большего количества информации.

`Fix:` **(Решение)**

Самая важная часть отчета представлена в поле `Fix`. Если у вас есть заплатка, решающая проблему, разместите ее здесь. Если вы знаете обходной путь, расскажите о нем. Дайте здесь любую информацию, способствующую решению проблемы. Иногда самое необычное предложение или условие является жизненно важным для нахождения решения. Если вы не представляете, как решить или обойти эту проблему, оставьте поле `Fix` пустым. Случайные мысли или предположения в этом поле не улучшат ваш отчет.

Пример отчета

Учитывая вышесказанное, давайте заполним образец отчета о панике системы, анализ которой я приводил в первой половине этой главы. Это – реальное паника, с которой я столкнулся при написании главы 18, поэтому мне действительно необходимо решить эту проблему. Сразу же после обновления системы до версии FreeBSD-current от 4 июля 2007 года стали появляться аварийные ситуации при обращении к внешнему массиву SCSI. Я планировал использовать этот массив для демонстрации модулей GEOM, и мне была необходима его правильная работа. Похоже, что сообщение о панике указывало на проблему с драйвером ahc0. Я указал свои имя и адрес электронной почты в верхней части, а затем перешел к более интересной части формы:

```
>Submitter-Id:    current-users
>Originator:      Michael W Lucas
>Organization:    none
>Confidential:    no
>Synopsis:        panic: _mtx_lock_sleep: in aic7xxx_osm.h (with backtrace)
>Severity:        serious
```

```
>Priority:       medium
>Category:       kern
>Class:          sw-bug
>Release:        FreeBSD 7.0-CURRENT i386
```

Поле `Submitter-Id:` я оставил без изменения. Автором сообщения являюсь я, и любое упоминание *организации* со ссылкой на меня было бы смешным. Этот отчет не является конфиденциальным.

Заполнить поле `Synopsis` оказалось сложнее. Я поместил сюда начальную часть строки сообщения о панике и место ее возникновения, а также упомянул о том, что у меня есть обратная трассировка. Это все, что удалось поместить в одну строку.

Эта ошибка затрагивает единственное устройство, но поскольку она приводит к панике, я присвоил ей уровень серьезности `serious`. Я присвоил проблеме средний приоритет просто потому, что не смог заставить себя дать низкий, к тому же это позволит разработчику не считать проблему критичной.

Это паника ядра, поэтому я отнес ее к категории `kern`. Я столкнулся с аварийным отказом ядра, поэтому указал класс `sw-bug`.

Поле `Environment` дает немного больше свободы. Я записал вывод команды `uname -a` и затем добавил примечание, объясняющее некоторые особенности моего окружения:

```
Это - стандартная система i386 с внешним SCSI массивом.
Протокол загрузки прилагается.
```

Поле `Description` позволяет сообщить разработчикам FreeBSD, что проблема заключается в следующем:

```
Система прекрасно работала на версии ядра от 22 апреля 2007. Я обновил
систему до версии -current от 4 июля 2007 и сразу же получил эту панику.
После дополнительного тестирования я обнаружил, что без проблем могу читать
данные с дисков SCSI, но при попытке записи на диск получаю панику.
Сообщение о панике и обратная трассировка прилагаются.
```

В поле `How-To-Repeat` я дал пошаговые инструкции о том, как воспроизвести эту ошибку:

```
/dev/da7 - мой тестируемый диск. Я могу запустить команду:
# fdisk /dev/da7
И многократно читать таблицу участков без каких-либо проблем. Но простая
операция записи, например:
# fdisk -BI /dev/da7
Приводит к панике системы. Это происходит не каждый раз, но если я запускаю
эту команду несколько раз подряд, то обязательно получаю панику. После пяти
успешных запусков команды "fdisk-BI /dev/da7" шестая приведет к аварийной
ситуации.
```

Это кратко описывает мою проблему, без крика о том, что у меня сжатые сроки, и мне необходимо написать главу 18, прежде чем толпа моих

фанатов[1] придет к моей двери с вилами и факелами, требуя эту книгу. Я дал четкие инструкции, как воспроизвести проблему. Кроме того, я включил подробный протокол загрузки и обратную трассировку ядра.

Отправка отчета

Чтобы можно было отправить отчет, в вашей системе должна иметься служба электронной почты. Если вы не можете отправлять сообщения по электронной почте с машины, то вам не следует использовать для этого программу send-pr(1). Сохраните отчет о проблеме и связанные с ним файлы на машине, которая может отправлять электронную почту, и отправьте отчет с помощью send-pr(1) с этой машины. Используйте ключ `-f`, чтобы указать программе send-pr(1) заполненный шаблон сообщения:

```
# send-pr -f prreport.txt
```

Отчет отправлен. Ничего сложного, если у вас имеется вся необходимая информация!

После отправки отчета

Спустя несколько часов после отправки отчета о проблеме я получил по электронной почте сообщение, что теперь являюсь подателем отчета kern/114489. Неважно, каким образом вы отправили свой отчет, вы получите подтверждение с номером отчета в течение нескольких часов. Любые ответы по обратному адресу будут автоматически присоединяться к этому отчету, пока вы не измените тему. Вы можете отправлять заплатки и ответы с любого компьютера, где есть возможность отправлять электронную почту.

GNATS также присоединит любые ответы разработчиков к истории вашего отчета о проблеме. Теперь, когда ваш отчет в базе данных FreeBSD, он будет отслеживаться постоянно. Это не гарантирует, что кто-то займется вашим вопросом или что ваша проблема будет решена, он просто зарегистрирован и доступен для всех, навсегда.

Если через несколько недель покажется, что ваш отчет забыт, пошлите дружелюбное сообщение в соответствующую почтовую рассылку, указав номер отчета, коротко описав суть проблемы и пояснив, почему она так важна. Поскольку FreeBSD поддерживается силами добровольцев, вполне возможно, что-то случилось со специалистом, который обрабатывает отчеты такого типа. Многие разработчики FreeBSD – профессиональные программисты, однако для многих из них FreeBSD – это хобби, которое отходит на задний план перед больными детьми или рабочими сроками. Если ничего другого не остается,

[1] Три фаната – это, наверное, еще не толпа, и Аманда – это скорее саркастический, чем слепо преданный фанат, но я пользуюсь тем, что у меня есть.

вы можете обратиться в компании, оказывающие платную поддержку; их список представлен на веб-сайте FreeBSD.

В течение нескольких дней я получил ответы от разработчиков, предложивших заплатки для решения моей проблемы. Первый раунд наложения заплаток привел к появлению ошибок в других местах, но за этим быстро последовали заплатки, исправляющие и эти проблемы. Глава 18 присутствует в книге – это доказывает, что моя проблема решена.

Неожиданно много отчетов о сложных проблемах быстро приводят к решению проблемы, если в них представлена надлежащая информация. Не забывайте, что разработчики FreeBSD поддерживают систему из любви к ней, а не по принуждению. Они хотят производить хороший код. Это более сильная мотивация по сравнению с денежным чеком. Если вы помогаете им производить хороший продукт, они будут рады работать с вами.

На конец 2007 года я представил несколько десятков отчетов о проблемах. Большая часть этих проблем были решены и/или зафиксированы и затем закрыты. Половина из них представляли собой описание тривиальных ошибок в документации из каталога */usr/src/contrib* – области, где проект FreeBSD снимает с себя ответственность за незначительные исправления. Если такой тупица, как я, может благополучно разрешить более 90% своих проблем, то сможет и любой другой. Однако имейте в виду, если вы будете передавать достаточно корректные заплатки, то обнаружите, что разработчики, с которыми вы взаимодействуете, начнут шептаться за вашей спиной. В конечном счете они устанут выполнять функции секретаря и предложат вам возможность самостоятельно отправлять свои заплатки. Если вы откажетесь, никто настаивать не будет. Но не волнуйтесь, быть разработчиком не так тяжело. Слухи о том, что ритуал посвящения в Проект FreeBSD проводится группой датчан с топорами за велосипедным гаражом, не соответствуют действительности. По большей части.

Как бы то ни было, продолжайте подавать хорошие отчеты о проблемах, это – единственный путь улучшения FreeBSD!

Послесловие

Если вы добрались до этого места, то уже знаете, как управлять системой FreeBSD и применять ее в качестве платформы для выполнения любых серверных задач. Возможно, вам потребуется изучить работу новых программ, но сейчас вы знаете об операционной системе достаточно, чтобы настроить ее по своему усмотрению. Поздравляю! FreeBSD – это замечательная, гибкая платформа. Она способна играть любую роль в вашей сети. В заключение я кратко расскажу о некоторых других аспектах FreeBSD.

На протяжении всей книги мы говорили об очевидных компонентах FreeBSD: о программах, о ядре, о функциональных возможностях и т. д. Единственное, о чем мы говорили не так много, – это сообщество, которое создает все это.

Сообщество

Сообщество FreeBSD состоит из специалистов по вычислительной технике, опытных программистов, пользователей, системных администраторов, технических писателей и всех интересующихся этой системой. У них разное общественное положение, уровень образования. Они разбросаны по всему миру. Лично я общался с пользователями FreeBSD со всех континентов и самых крупных островов на планете.[1] Национальность, раса, цвет кожи, вероисповедание здесь просто не имеют значения. Среди них есть специалисты по вычислительной технике. Некоторые работают в фирме-провайдере или на производстве. Есть медики, и даже работники пунктов проката видеокассет. Однажды я тесно сотрудничал с блестящим разработчиком, который оказался подростком. Довольно странно: часовой пояс *имеет* большое значение, но лишь потому, что от него зависит возможность общения разработчиков. Поскольку сообщество в основном взаимодействует по Сети, его члены представлены только своими словами и работой. Это люди,

[1] Да, включая человека, который взял свой ноутбук с FreeBSD в антарктический круиз. Этим путешествием они с женой отмечали 20-летие совместной жизни. Он не уточнил, бросила супруга ноутбук за борт или нет. Даже если и бросила, полагаю, по возвращении он просто купил себе другой, получше.

которые улучшают и развивают ОС FreeBSD, делая ее не просто набором нулей и единиц, приспособленным для обслуживания веб-сайтов.

Одна из интересных особенностей сообщества FreeBSD состоит в том, что оно выработало методы, позволяющие пережить смену лидера. У многих открытых проектов – один лидер или небольшая группа лидеров. Когда эти люди решают уйти, проект, как правило, перестает развиваться. Кто-то может продолжить развитие проекта или на его основе организовать свой проект, но первоначальное сообщество, сложившееся вокруг прежнего проекта, обычно распадается. Практически все, кто стоял у истоков Проекта FreeBSD, уже занимаются чем-то другим, но сообщество воспитало новых лидеров. Три поколения лидеров Проекта FreeBSD свидетельствуют о его устойчивости к переменам в руководстве, уникальной в мире открытого программного обеспечения. Нынешние лидеры Проекта FreeBSD активно готовят смену, обучая младших членов сообщества, в которых провидят лидеров 2010-х. Да и мы всегда рады старшим товарищам, и в CVS-репозитарии, и в баре.

Почему мы делаем это?

У каждого, кто вносит свой вклад в проект FreeBSD, на то есть свои причины. За улучшение кода лишь немногие разработчики получают плату – от организаций, зависящих от FreeBSD, либо от правительственных организаций, таких как DARPA. Для большинства разработчиков FreeBSD их занятие – хобби, поэтому они могут программировать лучше, чем на основной работе. Сколько у вас было проектов, качество которых пострадало из-за ограниченности сроков разработки? Сроки выхода версий FreeBSD анонсируются на месяцы и годы вперед, поэтому разработчики могут планировать свое участие и занятость.

Многие из нас не являются разработчиками программного обеспечения, но вносят свой вклад в FreeBSD по-другому. Одни пишут документацию, другие проектируют веб-сайты, третьи просто околачиваются в почтовых рассылках и отвечают на вопросы пользователей. Люди отдают развитию FreeBSD много своего времени. Почему? Смею вас заверить, что гонорар за эту книгу не компенсирует мне вечера, которые я мог бы провести с семьей. Я пишу послесловие в беседке на скале, возвышающейся над озером Эри (Lake Erie) в провинции Онтарио, Канада. Все остальные ушли гулять и ловить бабочек, надеюсь, не на эту же скалу. Почему я занимаюсь этим вместо того, чтобы присоединиться к другим?

Мы делаем это потому, что нам приятно приносить пользу остальной части человечества, и для того, чтобы вернуть долг тем, кто когда-то сделал это для нас.

Вы можете просто взять то, что предлагает FreeBSD, и использовать это по своему усмотрению. Со временем многие из вас захотят внести

свой вклад в деятельность сообщества. Вот почему сообщество растет – а рост сообщества означает, что развитие FreeBSD будет продолжаться.

Если вы тоже хотите получать от этого удовольствие, у нас найдется место и для вас.

Что вы можете сделать?

Для тех, кто по тем или иным мотивам захочет помочь FreeBSD, всегда найдется место в какой-нибудь части Проекта. С 1996 года, когда я начал работать с FreeBSD, в рассылках мне часто попадаются сообщения вроде «Рад был бы помочь, но я не умею программировать» (думаю, правда, что это я сам себе пишу). Обычно на это нечего ответить. В конце концов, если вы сами знаете, что не можете помочь, то не о чем и говорить. Но если вы решите, что можете помочь, – значит, и в самом деле сможете.

Никто не отрицает, что видные программисты – главные знаменитости FreeBSD. Среди них есть великие мастера, и мало кто из нас помышляет стать новым Робертом Уотсоном (Robert Watson) или Джорданом Хаббардом (Jordan Hubbard). Но даже ни аза не смысля в программировании, вы можете чем-то помочь. Поменяйте местами две фразы в своем сообщении и отправьте его еще раз.

Не спрашивайте о нуждах проекта. Вы не сможете удовлетворить их без крупного банковского счета, предназначенного для благотворительных целей. (В любом случае, если вы не знаете, куда потратить деньги, фонд FreeBSD с радостью избавит вас от этого геморроя.) Не говорите: «Было бы круто сделать в FreeBSD то-то и то-то», – если сами не сможете сделать это. Что вы можете? Любой крупной организации требуются самые разные специалисты, и какими бы талантами вы ни обладали, они могут пригодиться проекту FreeBSD.

Я знаю тех, кто создает веб-сайты, посвященные FreeBSD, и предоставляет такие ценные ресурсы, как *http://bsdforums.com* и *http://www.freshports.org*. И их уважают за это.

Я знаю пользователей FreeBSD, которые занимаются веб-дизайном и пишут документацию. Они помогают создавать информационное наполнение веб-сайтов. Займитесь этим – и со временем вы сможете стать одним из создателей (committers).

Я знаю пользователей FreeBSD, которые тратят свое время, отвечая на вопросы в почтовых рассылках. Займитесь этим – и со временем у вас появится желание обновить сборник FAQ. Передав достаточное число обновлений этого сборника, вы сможете войти в круг создателей.

Я пишу очень много, и у меня получается. Я написал несколько обновлений для FAQ, а теперь пишу эту книгу. Благодаря обновлениям FAQ я стал одним из создателей (должен заметить, что эта книга – куда более весомый вклад). При виде написанного мной кода младенцы плачут,

а старушки творят крестное знамение, но ребята из FreeBSD рады мне и считают меня своим товарищем, просто потому что я что-то делаю.

В чем *вы* сильны? Что вы любите делать? Задействуйте свое мастерство. Оно будет оценено по достоинству.

Если совсем ничего...

Если у вас и в самом деле нет полезных навыков или идей, просто перечитайте эту книгу. Почитайте документацию на сайте FreeBSD. Подпишитесь на рассылку *FreeBSD-questions@FreeBSD.org* и помогайте другим пользователям. Многие начинают с этого.

Избрав этот путь, направляйте людей к существующим информационным ресурсам. Когда кто-то задает вопрос, ответ на который есть в FAQ, предоставьте ему ссылку на основную страницу FAQ. Если данный вопрос задавался ранее, посоветуйте обратиться к архивам почтовых рассылок. Люди, наученные помогать самим себе, – это самое эффективное вложение вашего времени, причем не только в рамках помощи проекту FreeBSD, но и в гораздо более широком смысле. Народная мудрость гласит: научи кого-то рыбачить – и можешь продавать ему рыболовные крючки. Отвечая на вопросы из почтовых рассылок, вы увидите, в чем нуждается проект. Удовлетворить одну из таких потребностей вам наверняка окажется по силам.

Получение результата

Открою большой секрет успеха FreeBSD. Все компоненты появились так: кто-нибудь видел ту или иную потребность, которую он мог удовлетворить, и делал это. NetBSD и FreeBSD появились, когда группа пользователей 386BSD устала ждать очередного выпуска. Я не спрашивал разрешения на создание этой книги, прежде чем приступить к ней. Добрые люди из рассылки *bugbusters@FreeBSD.org* перепахивают базу отчетов о проблемах не забавы ради – просто для них это важное дело, на которое не жаль времени. (Если вы программист, загляните в базу отчетов и поищите проблему, решить которую окажется вам по силам, и это будет самый ценный ваш вклад.)

Как только у вас появилась идея – ищите в почтовой рассылке обсуждение этой темы. Не все заявленные проекты были реализованы. Если кто-то уже предлагал вашу идею, ознакомьтесь с ее обсуждением. Если все ее одобрили, наверное, вы получили бы такую же реакцию. Если никто не работает над этим проектом, приступайте к работе! Группа разработки FreeBSD будет просто счастлива, если первое ваше сообщение, которое они увидят, будет таким: «Привет, по такому-то адресу вы можете найти мои заплатки, реализующие такую-то и такую-то функциональную возможность в инсталляторе».

Избегайте вопросов вроде: «Почему кто-нибудь не сделает то-то?» Большинство подобных вопросов относится к одной из трех категорий: очевидные («Правда, было бы круто, если бы FreeBSD могла работать на мэйнфреймах?»), глупые («Почему в ядре нет параметра ПРИНЕСИМНЕХОЛОДНОГОПИВА?») или то и другое сразу («Почему система не поддерживает мой Sinclair ZX80?»). В любом из этих случаев вопрошающий вряд ли готов что-то сделать сам или просто помочь тем, кому такие задачи *по силам*. Подобные предложения только раздражают и впустую расходуют трафик.

Короче: работайте молча. Делайте то, что можете, и делайте это добросовестно – и вас оценят. Программисты могут помочь выбором идеи из Списка идей проекта FreeBSD (FreeBSD Project Ideas List) или разгребанием отчетов о проблемах в рассылке *bugbusters@FreeBSD.org*. Не программисты могут помочь, обнаружив дыру и сообщив о ней. Вы можете стать одним из лидеров проекта или «тем чуваком, который тусуется на *-questions@* и помогает народу с ACPI». Все это очень важно. Ваша помощь поможет Проекту FreeBSD процветать и развиваться. Просто будьте постоянно рядом – и вы сможете стать одним из новых лидеров FreeBSD.

Я предвкушаю нашу встречу в почтовых рассылках.

A

Некоторые полезные sysctl MIBS

Это приложение представляет собой словарь полезных sysctl MIB, а также некоторые sysctl MIB, не столь полезные, но уже давно обсуждаемые на форумах FreeBSD. Инструменты для управления MIB обсуждаются в главе 5. Когда MIB подробно рассматривается в том или ином месте книги, приводится ссылка на соответствующую главу. Безусловно, в вашей системе намного больше sysctl, однако с представленными здесь параметрами я сталкиваюсь многократно.

Помните, что необдуманное применение sysctl может легко причинить ущерб работающей системе. Так, если установить на использование ресурсов предел ниже необходимого уровня, это может привести к аварийному завершению работы процесса или системы. Перед настройкой sysctl убедитесь, что вы представляете последствия своих действий. Если вы не понимаете назначение того или иного sysctl, поэкспериментируйте с ним на тестовой системе.

Для каждого sysctl я представлю типичное значение из моей тестовой системы, чтобы показать, какими они должны быть.

В описании для каждого sysctl указывается, когда sysctl может быть изменен. Один sysctl можно изменить в любое время, другой – только при загрузке. Некоторые доступны только для чтения.

Для каждого sysctl указано, является он переменной, переключателем или настраиваемым параметром (tunable). *Переменные* имеют широкий диапазон значений. *Переключатели* имеют только два значения, `1` (описываемый сервис активизирован) и `0` (сервис отключен). *Настраиваемые параметры* устанавливаются загрузчиком. (Строго говоря, настраиваемый параметр отличается от sysctl, но, определяя значение настраиваемого параметра, вы тем самым определяете значение sysctl.)

`kern.osrelease: 7.0-RELEASE`

Только для чтения. Содержит номер версии FreeBSD, например 7.2-release, 8.0-current и т. д.

`kern.maxvnodes: 100000`

Переменная времени исполнения. Максимальное количество виртуальных файловых дескрипторов (vnodes, дескрипторов виртуальной файловой системы), которые могут быть одновременно открыты в системе.

`kern.maxproc: 6164`

Настраиваемый параметр времени загрузки. Максимальное количество процессов, которые система может запустить одновременно.

`kern.maxfiles: 12328`

Переменная времени исполнения и настраиваемый параметр времени загрузки. Максимальное количество файлов, которые система может открыть для чтения или записи одновременно.

`kern.argmax: 262144`

Только для чтения. Максимальное количество символов, которые можно использовать в одной командной строке. При необычных обстоятельствах этот предел может оказаться слишком маленьким. В подобных ситуациях используйте xargs(1).

`kern.securelevel: -1`

Переменная времени исполнения. Текущий уровень безопасности ядра. Подробности в главе 9.

`kern.hostname: bewilderbeast.blackhelicopters.org`

Настраиваемый параметр времени исполнения. Имя хоста, устанавливается в */etc/rc.conf*.

`kern.posix1version: 200112`

Только для чтения. Версия POSIX, которой соответствует ядро. Если требуется изменить это значение, не стесняйтесь, предлагайте проекту свой код поддержки других версий POSIX.

`kern.ngroups: 16`

Только для чтения. Максимальное количество групп, к которым может принадлежать пользователь.

`kern.boottime: {sec = 1187744126, usec = 946476} Tue Aug 21 20:55:26 2007`

Только для чтения. Время загрузки системы, причем в двух форматах – в секундах от начала эпохи и в привычном для человека.

`kern.domainname:`

Переменная времени исполнения. Имя домена YP/NIS, а не имя домена TCP/IP. Если вы не используете YP/NIS, значение переменной будет пустым.

`kern.osreldate: 700052`

Только для чтения. Это версия FreeBSD, без фактической даты. Различные «порты» используют это значение для конфигурирования самих себя.

`kern.bootfile: /boot/kernel/kernel`

Только для чтения. Файл ядра, с которым загрузилась система. Формально вы можете изменить это значение, но не делайте это. Лучше выполните перезагрузку на новое ядро. Единственный случай, когда этот параметр изменяется в работающей системе, – это установка нового ядра и сохранение старого с именем */boot/kernel.old*. Но это особый случай.

`kern.maxfilesperproc: 11095`

Переменная времени исполнения. Максимальное количество файлов, которые может открыть один процесс.

`kern.maxprocperuid: 5547`

Переменная времени исполнения. Максимальное количество процессов, которые одновременно может запустить один пользователь.

`kern.ipc.somaxconn: 128`

Переменная времени исполнения. Максимальное количество новых соединений, которые система может принять одновременно. На сильно нагруженном сервере значение этого параметра можно увеличить до 256 или даже до 512.

`kern.ipc.maxpipekva: 16777216`

Настраиваемый параметр времени загрузки. Максимальный объем памяти ядра, который может быть использован для программного канала.

`kern.ipc.pipekva: 212992`

Только для чтения. Текущий объем памяти, используемый каналами.

`kern.ipc.shmmax: 33554432`

Переменная времени исполнения. Максимальный размер сегмента разделяемой памяти System V. Возможно, вам придется настроить его для программ, использующих большие объемы разделяемой памяти, например для баз данных.

`kern.ipc.shmseg: 128`

Настраиваемый параметр времени загрузки. Максимальное число сегментов разделяемой памяти System V, которое может открыть один процесс.

`kern.ipc.shmall: 8192`

Переменная времени исполнения. Максимальное число страниц, доступных для разделяемой памяти System V.

`kern.ipc.shm_use_phys: 0`

Переключатель времени исполнения, настраиваемый параметр времени загрузки. Позволяет запретить свопинг сегментов разделяемой памяти. Это может повысить производительность при использовании больших объемов разделяемой памяти System V, но это исключит возможность пейджинга разделяемой памяти, что может привести к снижению производительности других программ в вашей системе.

`kern.ipc.numopensockets: 70`

Только для чтения. Число открытых сокетов всех видов, включая сетевые и локальные сокеты.

`kern.ipc.maxsockets: 12328`

Переменная времени исполнения, настраиваемый параметр времени загрузки. Общее количество сокетов, доступных в системе. Включает в себя сокеты домена UNIX, а также сетевые сокеты.

`kern.logsigexit: 1`

Переключатель времени исполнения. При аварийном завершении программа обычно посылает сигнал. Если установлен этот переключатель, имя программы и сигнал завершения будут протоколироваться в */var/log/messages*.

`kern.init_path: /sbin/init:/sbin/oinit:/sbin/init.bak:/rescue/init:/stand/sysinstall`

Только для чтения. init(8) – это программа, запускающая систему. Если вы повредили систему (например, в результате неудачного обновления), с помощью этого sysctl можно указать серию резервных каталогов, где система может попытаться отыскать копию программы init(8). Этот sysctl также доступен в загрузчике как `init_path`.

`kern.init_shutdown_timeout: 120`

Переменная времени исполнения. Число секунд, в течение которых пользовательские процессы должны завершиться самостоятельно во время процедуры останова системы. По истечении этого времени ядро прекращает работу независимо от того, какие процессы остались запущенными.

`kern.randompid: 0`

Переменная времени исполнения. При установке в `0` каждый следующий запущенный процесс получает идентификатор на единицу больше предыдущего. При установке большего значения значение следующего PID выбирается случайным образом. С ростом значения этого sysctl увеличивается и случайность следующего PID.

`kern.openfiles: 246`

Только для чтения. Количество файлов, открытых в системе на данный момент.

`kern.module_path: /boot/kernel;/boot/modules`

Переменная времени исполнения. Список каталогов, в которых kldload(8) проверяет наличие модулей ядра.

`kern.maxusers: 384`

Настраиваемый параметр времени загрузки. В течение многих лет параметр `kern.maxusers` позволял системному администратору в BSD и FreeBSD определить, сколько памяти выделить для выполнения некоторых системных задач. Современные системы FreeBSD автоматически выбирают значение `kern.maxusers`. Любая документация, предлагающая изменить `kern.maxusers`, почти наверняка устарела.

`kern.sync_on_panic: 0`

Переключатель времени исполнения. В случае паники FreeBSD (глава 21) размонтирует все диски без синхронизации, что приводит к образованию *грязных дисков*, о чем говорилось в главе 8. Если установить этот sysctl в значение `1`, FreeBSD попытается синхронизировать диски перед остановкой. Это крайне опасно, поскольку нет никакой возможности узнать состояние запаниковавшей системы. В некоторых случаях попытка синхронизации дисков после паники может повредить данные или файловую систему, в результате запаникует не только система, но и системный администратор.

`kern.coredump: 1`

Переключатель времени исполнения. По умолчанию при аварийном завершении программы FreeBSD записывает дамп памяти. Если установлено значение `0`, дамп памяти не создается. Это весьма полезно на встроенных и бездисковых системах.

`kern.nodump_coredump: 0`

Переключатель времени исполнения. Позволяет автоматически устанавливать флаг `nodump` для файлов core. Подробности в главе 4.

`kern.timecounter.choice: TSC(-100) i8254(0) dummy(-1000000)`

Только для чтения. Список аппаратных часов, доступных в системе. Большинство современных аппаратных средств имеют четыре общих варианта, в порядке убывания предпочтительности: `ACPI`, `i8254`, `TCK` и `dummy`. Числа дают относительную точность различных часов.

`kern.timecounter.hardware: i8254`

Переменная времени исполнения. Изменяя эту переменную, вы можете использовать различные аппаратные часы. Выбранный вариант записывается в параметр sysctl `kern.timecounter.choice`. По умолчанию FreeBSD использует наиболее предпочтительные часы из имеющихся.

`kern.timecounter.smp_tsc: 0`

Настраиваемый параметр времени загрузки. Значение `0` предотвращает использование аппаратных часов TSC. Эти часы неправильно работают в большинстве многопроцессорных систем.

`kern.sched.name: 4BSD`

Только для чтения. Имя планировщика в запущенном ядре.

`kern.sched.quantum: 100000`

Переменная времени исполнения. Максимальное количество микросекунд для выполнения процесса, когда другие процессы ожидают предоставления времени процессора при использовании планировщика 4BSD. Если вы предполагаете изменить это значение, то почти наверняка делаете что-то не так. Планировщик ULE не использует этот параметр sysctl.

`kern.sched.preemption: 1`

Только для чтения. Показывает, разрешена ли возможность вытеснения процессов в ядре. Возможность вытеснения позволяет более срочному потоку ядра прерывать работу менее срочного.

`kern.log_console_output: 1`

Переключатель времени исполнения, настраиваемый параметр времени загрузки. По умолчанию все сообщения, которые FreeBSD выводит на консоль, одновременно передаются демону syslogd. (Эти сообщения не предполагают ввод ответов с консоли, а просто уведомляют о том, что кто-то вошел в систему или вышел из нее.) Установка параметра в значение `0` отключает эту функцию.

`kern.smp.disabled: 0`

Переключатель времени загрузки. Чтобы отключить поддержку SMP в */boot/loader.conf*, установите значение `kern.smp.disabled` в 1.

`kern.smp.cpus: 2`

Только для чтения. Количество процессоров, работающих в системе.

`kern.filedelay: 30`

Переменная времени исполнения. Определяет, как часто система синхронизирует файловые данные между буферным кэшем виртуальных дескрипторов и диском. Увеличение частоты синхронизации увеличивает нагрузку на диск, в то же время уменьшение частоты синхронизации увеличивает риск потери данных. Это значение могут изменять только те системные администраторы, которые хорошо разбираются в буферизации.

`kern.dirdelay: 29`

Переменная времени исполнения. Определяет, как часто система синхронизирует данные о каталогах, хранящиеся в буферном кэше и на диске. Значение этого параметра должно быть несколько меньше, чем `kern.filedelay`. Опять же, только очень опытные системные администраторы могут изменять это значение.

`kern.metadelay: 28`

Переменная времени исполнения. Определяет, как часто система синхронизирует метаданные файловой системы между буферным

кэшем и диском. Значение этого параметра должно быть несколько меньше, чем `kern.dirdelay`. Опять же, этот параметр предназначен только для очень опытных системных администраторов! Эксперименты с синхронизацией – прекрасный способ получить богатый опыт потери данных.

`vm.v_free_min: 3258`

Переменная времени исполнения. Максимальное количество страниц кэша и свободной памяти, которые должны быть доступны, прежде чем процесс, ожидающий памяти, будет разбужен или запущен.

`vm.v_free_target: 13745`

Переменная времени исполнения. Минимальное количество страниц свободной памяти и кэша, которое пытается обеспечить или превысить менеджер виртуальной памяти.

`vm.v_free_reserved: 713`

Переменная времени исполнения. Если количество страниц свободной памяти станет меньше этого значения, менеджер виртуальной памяти начнет выполнять свопинг процессов.

`vm.v_inactive_target: 20617`

Переменная времени исполнения. FreeBSD старается обеспечить это количество страниц неактивной памяти, делая активные страницы неактивными.

`vm.v_cache_min: 13745`

`vm.v_cache_max: 27490`

Переменные времени исполнения. Минимальный и максимальный желаемый размер очереди кэша виртуальной памяти.

`vm.swap_enabled: 1`

Переключатель времени исполнения. Управляет использованием пространства свопинга. Если параметр имеет значение `0`, система не будет выполнять свопинг. Это очень полезно для бездисковых систем.

`vm.swap_idle_enabled: 0`

Переключатель времени исполнения.

`vm.swap_idle_threshold1: 2`

Переменная времени исполнения.

`vm.swap_idle_threshold2: 10`

Переменная времени исполнения. Установка параметра `swap_idle_enabled` предпишет менеджеру виртуальной памяти помещать бездействующие процессы в пространство свопинга быстрее других процессов. Параметр `threshold` предписывает системе, сколько секунд надо ждать, прежде чем отнести процесс к бездействующим. Значения по умолчанию приемлемы в большинстве случаев; про-

стое включение vm.swap_idle_enabled окажет влияние. Это может помочь некоторым системам остаться наплаву, когда у них очень мало памяти и в своп вытесняются процессы целиком.

vm.exec_map_entries: 16

Переключатель времени загрузки. Максимальное число процессов, которые можно запустить одновременно.

vfs.nfs.diskless_valid: 0

Только для чтения. Если установлено значение 0, эта система работает как обычная система с дисками. Если не 0 – система работает как бездисковая.

vfs.nfs.diskless_rootpath:

Только для чтения. Указывает путь к корневой файловой системе бездисковой машины.

vfs.nfs.diskless_rootaddr:

Только для чтения. Указывает IP-адрес корневой файловой системы бездисковой машины.

vfs.vmiodirenable: 1

Переключатель времени исполнения. Позволяет UFS использовать систему виртуальной памяти для кэширования результатов поиска в каталогах, что позволит увеличить производительность дисков.

vfs.usermount: 0

Переключатель времени исполнения. Если этот параметр установлен, пользователи смогут монтировать файловые системы в собственные точки монтирования. Это дает возможность непривилегированным пользователям монтировать дискеты и компакт-диски. Подробности в главе 8.

vfs.ffs.doasyncfree: 1

Переменная времени исполнения. Указывает UFS файловой системе асинхронно обновлять индексные дескрипторы и косвенные блоки после реорганизации дискового пространства. См. sysctl vfs.ffs.doreallocblks.

vfs.ffs.doreallocblks: 1

Переменная времени исполнения. По умолчанию FreeBSD реорганизует дисковое пространство так, чтобы файлы все время занимали непрерывные области. По сути, UFS постоянно дефрагментирует себя. Установите значение 0, чтобы отключить этот режим.

net.inet.ip.portrange.lowfirst: 1023

net.inet.ip.portrange.lowlast: 600

Переменные времени исполнения. Некоторым программам или протоколам требуется устанавливать исходящие соединения, используя порты с низкими номерами; это указывает, что они запу-

щены привилегированными процессами. Не забывайте, что исходящие соединения TCP/IP используют порты на локальной системе. Кроме того, помните, что порты с номерами от 0 до 1023 могут быть открыты только пользователем root. Если входящее соединение исходит из порта с низким номером, теоретически это может означать, что приложение запущено с правами root. (При уверенности, что машина, которая инициализировала соединение, не была взломана.) FreeBSD использует номера портов между значениями `net.inet.ip.portrange.lowfirst` (по умолчанию `1023`) и `net.inet.ip.portrange.lowlast` (по умолчанию `600`). Я знаю только один случай, когда изменение этих значений имело смысл. Микропрограммы IPMI на некоторых сетевых картах Intel всегда перехватывают пакеты UDP, отправленные в порты 623 и 664. Если у вас используются другие сетевые карты, не изменяйте эти значения.

`net.inet.ip.portrange.first: 49152`

`net.inet.ip.portrange.last: 65535`

Переменные времени исполнения. Когда FreeBSD выделяет случайный порт исходящего соединения для непривилегированного процесса, она использует номера портов между `net.inet.ip.portrange.first` (по умолчанию `49152`) и `net.inet.ip.portrange.last` (по умолчанию `65535`).

`net.inet.ip.portrange.hifirst: 49152`

`net.inet.ip.portrange.hilast: 65535`

Переменные времени исполнения. Некоторым программам для соединения требуется порт с высоким номером. По умолчанию этот диапазон портов пересекается с портами, указанными `net.inet.ip.portrange.first` и `net.inet.ip.portrange.last`, но вы можете разделить эти диапазоны, используя параметры sysctl `hifirst` и `hilast`.

`net.inet.ip.portrange.reservedhigh: 1023`

`net.inet.ip.portrange.reservedlow: 0`

Переменные времени исполнения. Диапазон портов, которые могут быть открыты только пользователем root.

`net.inet.ip.portrange.randomized: 1`

Переменная времени исполнения. По умолчанию FreeBSD выделяет случайные номера портов в диапазоне, выделенном для этого типа соединений.

`net.inet.ip.portrange.randomcps: 10`

Переменная времени исполнения. Максимальное число случайно выделенных портов в течение одной секунды. Если необходимо большее количество портов, то FreeBSD переходит на последовательное выделение портов, чтобы не исчерпать источник случайности в системе.

`net.inet.ip.forwarding: 0`

Переключатель времени исполнения. Может потребоваться, чтобы система FreeBSD действовала как шлюз, маршрутизатор или брандмауэр. Если этот параметр установлен, система сможет осуществлять перенаправление пакетов.

`net.inet.ip.redirect: 1`

Переключатель времени исполнения. Если этот параметр установлен, система действует как маршрутизатор или шлюз, посылая пакеты переадресации ICMP другим системам в сети. Этот параметр не оказывает эффекта, если система не выполняет маршрутизацию.

`net.inet.ip.ttl: 64`

Переменная времени исполнения. Максимальное количество переходов (hops), которые сможет выполнить любой протокол, отличный от ICMP, в сети.

`net.inet.ip.sourceroute: 0`

Переключатель времени исполнения. Включает и выключает перенаправление пакетов от источника. Маршрутизация от источника в целом считается плохой идеей для открытого Интернета.

`net.inet.ip.accept_sourceroute: 0`

Переключатель времени исполнения. Позволяет принимать пакеты, маршрутизируемые от их источника. Если вы не знаете, что такое маршрутизация от источника, то, поверьте, лучше обойтись без нее.

`net.inet.ip.fastforwarding: 0`

Переключатель времени исполнения. Данный параметр sysctl значительно увеличивает скорость прохождения пакетов через маршрутизаторы и шлюзы. Это достигается за счет устранения проверок на наличие тривиальных ошибок в пакетах, а также за счет обхода правил фильтрации пакетов.

`net.inet.ip.check_interface: 0`

Переключатель времени исполнения. Обеспечит базовую защиту вашего сервера от подделки исходящих IP-адресов в пакетах (spoofing). Это полезно только для маршрутизаторов, шлюзов и брандмауэров. В большинстве случаев такую защиту лучше организовать на уровне брандмауэра.

`net.inet.icmp.icmplim: 200`

Переменная времени исполнения. Максимальное число запросов ICMP в секунду, на которые ответит система. Для систем, которые обрабатывают огромное количество запросов ICMP (главным образом, те, на которых работают средства обеспечения безопасности и поддержки сети, такие как Nessus, nmap и Nagios), необходимо увеличить это значение. Установите значение `0`, чтобы вообще отключить возможность пакетов ICMP.

`net.inet.icmp.drop_redirect: 0`

Переключатель времени исполнения. Предписывает FreeBSD игнорировать пакеты переадресации ICMP. В большинстве случаев переадресация повышает производительность, но у меня были случаи, когда игнорирование переадресации помогло обеспечить нормальную работу в сети.

`net.inet.icmp.log_redirect: 0`

Переключатель времени исполнения. Включение этого параметра sysctl заставляет FreeBSD протоколировать любые переадресации пакетов ICMP на консоли. (Вам действительно не нужна консоль для чего-нибудь полезного, не так ли?)

`net.inet.icmp.bmcastecho: 0`

Переключатель времени исполнения. Позволяет системе отвечать на запросы по широковещательному адресу. Хотя это облегчало поиск неисправностей и было изначально необходимым условием соблюдения стандартов, ответы на широковещательные запросы вызывали проблемы с безопасностью. Поэтому по умолчанию эта возможность отключена на всех современных операционных системах.

`net.inet.tcp.rfc1323: 1`

`net.inet.tcp.rfc3042: 1`

`net.inet.tcp.rfc3390: 1`

Переключатели времени исполнения. Включают различные функции TCP, описанные в соответствующих RFC и достаточно безопасные. Если эти функции включены в вашей системе, некоторые другие очень-очень старые системы не смогут установить соединение с вашей системой, либо соединение окажется очень медленным.

`net.inet.tcp.sendspace: 32768`

`net.inet.tcp.recvspace: 65536`

Переменные времени исполнения. Начальный размер буфера по умолчанию для нового TCP/IP соединения. FreeBSD распределяет эти буферы динамически, как того требует пропускная способность удаленного хоста, тем не менее эти значения могут послужить неплохой отправной точкой. Во многих старых руководствах вам могут встретиться советы по изменению этих значений, но в наши дни их лучше не трогать.

`net.inet.tcp.log_in_vain: 0`

Переключатель времени исполнения. Разрешает протоколирование попыток подключения к любому порту TCP, который не прослушивает ни одна программа. В общедоступном Интернете это может породить огромный объем выводимой информации. Я рекомендую разрешить протоколирование на тестовой системе, только чтобы увидеть, сколько мусора летит из Интернета, и затем отключить его, чтобы снова можно было пользоваться консолью.

`net.inet.tcp.blackhole: 0`

Переключатель времени исполнения. По умолчанию TCP/IP возвращает код ошибки, когда пытается подключиться к закрытому порту. Это проявляется как ошибка «Connection reset by peer». Если присвоить этому параметру значение 1, то попытки подключения к порту TCP будут блокироваться, но сообщение об ошибке посылаться не будет. Это замедлит сканирование портов и может улучшить безопасность вашей системы. Однако это не может служить заменой фильтрации пакетов!

`net.inet.tcp.delayed_ack: 1`

Переключатель времени исполнения. Предписывает системе включать в пакеты данных информацию TCP ACK (подтверждение), а не посылать дополнительные пакеты, сигнализирующие о завершении соединения.

`net.inet.tcp.recvbuf_auto: 1`

Переключатель времени исполнения. Разрешает автоматическое изменение размеров приемного буфера

`net.inet.tcp.recvbuf_inc: 16384`

Переменная времени исполнения. Шаг приращения, при автоматическом увеличении приемного буфера.

`net.inet.tcp.recvbuf_max: 262144`

Переменная времени исполнения. Максимальный размер любого приемного буфера в системе.

`net.inet.tcp.slowstart_flightsize: 1`

Переменная времени исполнения. Определяет число пакетов, которые можно послать во время медленного старта транзакции TCP в глобальной сети.

`net.inet.tcp.local_slowstart_flightsize: 4`

Переменная времени исполнения. Определяет число пакетов, которые можно послать во время медленного старта транзакции TCP в локальной сети.

`net.inet.tcp.newreno: 1`

Переключатель времени исполнения. Включает и выключает восстановление соединений согласно RFC 2582. Эта функция известна как TCP NewReno Algorithm.

`net.inet.tcp.tso: 1`

Переключатель времени исполнения. Некоторые сетевые аппаратные средства могут обрабатывать сетевые запросы, разбитые на пакеты, размер которых определяется средой передачи. Например, при использовании некоторых сетевых карт FreeBSD может просто вручить сетевой карте большой блок данных и потребовать от нее разбить этот блок на пакеты. Такая возможность позволяет опти-

мизировать взаимодействие с сетевой картой и уменьшить нагрузку на процессор. Предписывает FreeBSD использовать аппаратную сегментацию, если таковая поддерживается.

`net.inet.tcp.sendbuf_auto: 1`

Переключатель времени исполнения. Позволяет автоматически изменять размер выходного буфера TCP-соединения.

`net.inet.tcp.sendbuf_inc: 8192`

Переменная времени исполнения. Определяет шаг автоматического увеличения или уменьшения размера выходного буфера TCP, когда это необходимо.

`net.inet.tcp.sendbuf_max: 262144`

Переменная времени исполнения. Максимальный размер одного выходного буфера TCP.

`net.inet.tcp.sack.enable: 1`

Переключатель времени исполнения. FreeBSD по умолчанию использует выборочное подтверждение ACK, но иногда, на старых хостах, эта особенность может приводить к появлению неприятностей.

`net.inet.tcp.do_tcpdrain: 1`

Переключатель времени исполнения. Предписывает системе выбрасывать пакеты из повторно собранной очереди, когда в системе ощущается нехватка mbufs. Теперь, когда FreeBSD автоматически распределяет mbufs из памяти ядра, этот параметр стал гораздо менее полезным, чем раньше.

`net.inet.tcp.always_keepalive: 1`

Переключатель времени исполнения. По умолчанию FreeBSD проверяет все соединения TCP с целью отыскать недействующие. Обычно, если соединение между двумя хостами разрывается, эти сообщения, подтверждающие активность соединения, предписывают системам закрыть его. Если этот режим отключен, FreeBSD не будет проверять наличие связи между хостами, система будет считать, что соединение активно, пока через него можно передавать данные.

`net.inet.udp.log_in_vain: 0`

Переключатель времени исполнения. Протоколирует попытки соединения с любыми портами UDP, которые не прослушивает ни одна программа.

`net.inet.udp.blackhole: 0`

Переключатель времени исполнения. Подобно TCP, UDP возвращает код ошибки при попытке подключиться к закрытому порту, который никто не прослушивает. Если присвоить этому параметру значение 1, попытки подключения к порту UDP будут блокироваться, но сообщение об ошибке посылаться не будет. Это замедлит сканирование портов.

`net.link.log_link_state_change: 1`

Переключатель времени исполнения. Если этот параметр установлен, FreeBSD будет протоколировать каждое включение и отключение сетевого интерфейса.

`debug.debugger_on_panic: 1`

Переключатель времени исполнения. Если установлено значение `1`, FreeBSD будет запускать отладчик при возникновении паники.

`debug.trace_on_panic: 0`

Переключатель времени исполнения. Если этот параметр установлен, FreeBSD автоматически будет выводить трассировку стека при входе в отладчик после возникновения паники.

`debug.witness.watch: 1`

Переменная времени исполнения, настраиваемый параметр времени загрузки. Если ядро скомпилировано с включенным механизмом WITNESS, FreeBSD будет выполнять дополнительную проверку всех блокировок ядра. Установка этого параметра в значение `0` отключает эти проверки. Как только механизм WITNESS будет отключен, включить его будет невозможно без перезагрузки. Обратите внимание: при отключении механизма WITNESS вы лишитесь его преимуществ, в частности, если система перейдет в состояние паники из-за блокировки во время тяжелой нагрузки, вы не получите сообщение о блокировке, которое подсказало бы разработчику FreeBSD, где искать проблему.

`debug.witness.skipspin: 1`

Переключатель времени загрузки. Предписывает механизму WITNESS не выполнять проверку спин-блокировок. Проверка спин-блокировок механизмом WITNESS замедляет работу практически любой машины. Пропуск спин-блокировок и использование WITNESS во всех других случаях делают работу механизма WITNESS просто медленной вместо невыносимо медленной. Этот параметр суть то же самое, что и параметр ядра `WITNESS_SKIPSPIN`.

`debug.minidump: 1`

Переключатель времени исполнения, настраиваемый параметр времени загрузки. По умолчанию во время паники FreeBSD делает только дампы памяти ядра. Если установите этот параметр в значение `0`, во время паники в дамп будет сбрасываться содержимое всей физической памяти.

`hw.ata.ata_dma: 1`

Переключатель времени загрузки. Управляет использованием DMA в устройствах IDE. Установите значение `0`, если ваши аппаратные средства вместо DMA используют PIO. Если у вас есть аппаратные средства PIO, то вы, вероятно, об этом знаете.

`hw.ata.atapi_dma: 1`

Переключатель времени загрузки. Управляет использованием DMA в устройствах ATAPI. Установите значение `0`, если ваши аппаратные средства вместо DMA используют PIO или при использовании DMA возникают ошибки. Опять же, если у вас есть аппаратные средства PIO, то вы, вероятно, об этом знаете.

`hw.ata.wc: 1`

Переключатель времени загрузки. Управляет возможностью кэширования записи в дисках IDE, которая реализована в некоторых моделях жестких дисков. При кэшировании записи во время записи на диск используется небольшой буфер на жестком диске, увеличивая производительность ценой надежности. Установите значение `0` для отключения кэширования записи данных и повышения надежности. Подробности приводятся в главе 8.

`hw.syscons.kbd_reboot: 1`

Переключатель времени исполнения. Если установлен этот параметр, нажатие старомодной комбинации клавиш Ctrl-Alt-Delete будет приводить к перезагрузке системы. Установите значение `0`, чтобы отключить эту возможность.

`compat.linux.osrelease: 2.4.2`

Переменная времени исполнения. Определяет версию Linux, поддерживаемую режимом совместимости Linux. Некоторые программы Linux при изменении этого параметра ведут себя иначе. Значение этого параметра можно изменять «на лету», но если в системе в этот момент будет запущена хотя бы одна программа Linux, это вызовет панику системы. Пусть лучше это делает программное обеспечение режима Linux.

`security.jail.set_hostname_allowed: 1`

Переменная времени исполнения. Определяет, может ли администратор клетки изменить имя хоста своей клетки. Подробности приводятся в главе 9.

`security.jail.socket_unixiproute_only: 1`

Переменная времени исполнения. Определяет, может ли владелец клетки использовать протоколы, отличные от TCP/IP. Подробности приводятся в главе 9.

`security.jail.sysvipc_allowed: 0`

Переменная времени исполнения. Определяет, может ли владелец клетки использовать вызовы System V IPC. Подробности приводятся в главе 9.

`security.jail.enforce_statfs: 2`

Переменная времени исполнения. Если установлено значение `0`, пользователи клетки могут просматривать все точки монтирования

и разделы системы как внутри, так и за пределами клетки. При установке в значение 1 доступны будут только точки монтирования, расположенные внутри клетки. Если установлено значение 2, доступна только корневая точка монтирования клетки.

`security.jail.allow_raw_sockets: 0`

Переменная времени исполнения. Если установлено значение 1, пользователь root клетки сможет создавать низкоуровневые сокеты. Подробности приводятся в главе 9.

`security.jail.chflags_allowed: 0`

Переключатель времени исполнения. Администратор клетки сможет использовать chflags(1). Подробности приводятся в главе 9.

`security.jail.list:`

Только для чтения. Это список всех активных клеток в системе. FreeBSD изменяет его, когда запускаются и останавливаются клетки, но вы не сможете изменить этот sysctl с помощью sysctl(8), чтобы активизировать новые клетки. Подробности приводятся в главе 9.

`security.jail.jailed: 0`

Только для чтения. Если установлено значение 1, программа, которая обращается к sysctl, работает внутри клетки. Подробности приводятся в главе 9.

`security.bsd.see_other_uids: 1`

Переключатель времени исполнения. По умолчанию пользователи могут видеть процессы других пользователей с помощью команды ps(1) и связанных с ними приложений. Установка этого параметра в 0 отключает эту возможность.

`security.bsd.see_other_gids: 1`

Переключатель времени исполнения. По умолчанию пользователи могут видеть процессы других групп с помощью команды ps(1) и связанных с ними приложений. Установка этого параметра в 0 отключает эту возможность.

`security.bsd.unprivileged_read_msgbuf: 1`

Переключатель времени исполнения. По умолчанию непривилегированные пользователи могут читать буфер сообщений системы с помощью dmesg(8), включая пользователей внутри клетки. Установка этого параметра в 0 отключает эту возможность, что, как правило, бывает желательно на сервере клеток.

`security.bsd.hardlink_check_uid: 0`

Переключатель времени исполнения. По умолчанию пользователи могут создавать жесткие ссылки (ln(1)) на любой файл. Если установить этот параметр в 1, пользователи не смогут создавать ссылки на файлы, принадлежащим другим пользователям.

`security.bsd.hardlink_check_gid: 0`

Переключатель времени исполнения. По умолчанию пользователи могут создавать жесткие ссылки на файлы, принадлежащие любой группе. Если установить этот параметр в 1, пользователи смогут создавать ссылки только на файлы, принадлежащие группе, которой принадлежат сами пользователи.

`security.bsd.overworked_admin: 3.1415925`

Только для чтения. Если вы действительно прочитали все это приложение, то, безусловно, очень хорошо потрудились. Закройте книгу. Выйдите на улицу. Подышите свежим воздухом час-другой. И вы почувствуете себя лучше. Однако если вы думаете, что этот параметр является действительным параметром MIB sysctl, вам следует отдохнуть в оффлайне по крайней мере пару дней.

Алфавитный указатель

Специальные символы и цифры

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

А

Б

В

Г

Д

Е

Ж

З

И

K

Л

М

Н

О

П

P

C

T

У

Ф

Х

Ц

Ч

Ш

Э

Я

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-126-6, название «FreeBSD. Подробное руководство, 2-е издание» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.