# ЗАЩИТНЫЕ МЕХАНИЗМЫ ОПЕРАЦИОННОЙ СИСТЕМЫ LINUX

## ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

Операционные системы на базе ядра Linux приобретают все большую популярность среди пользователей в различных сферах их деятельности. Надежность, простая архитектура, полная документированность и практически неограниченные возможности этих систем всегда привлекали компетентных специалистов. К сожалению, длительное время эти системы не представляли интереса для обычных пользователей, избалованных элементами графического интерфейса и роскошью прикладных программ корпорации Microsoft. Однако постепенное совершенствование графического интерфейса и программных приложений для обработки компьютерной информации во всех ее формах, в том числе совместимых с офисными приложениями Microsoft, неизбежно стирают пока еще имеющиеся различия в функциональных возможностях операционных систем Linux и Windows.

Операционным системам UNIX присущ ряд особенностей, которые выгодно отличают их от других универсальных операционных систем.

1. Системы UNIX эксплуатируются в компьютерном мире уже четыре десятилетия и успешно пережили не одно поколение ЭВМ, доказав возможность работы на различных аппаратных платформах. Хорошая надежность и документированность программного обеспечения позволила операционным системам этого семейства за длительный период эксплуатации доказать свою устойчивость к различным атакам, сбоям и прочим неприятностям.
2. ОС UNIX – очень экономная система по числу сущностей. Создается представление, что в операционных системах этого семейства нет ничего лишнего. В операционной системе определены только две категории пользователей, три вида процессов и семь видов объектов файловой системы. Пользователям дается три права на доступ к файлам: право чтения, записи и исполнения. Тем не менее этого минимума в большинстве жизненных ситуаций оказывается вполне достаточно для реализации надежного функционирования и защиты компьютерной информации.
3. Все объекты, с которыми приходится работать пользователю, именуются файлами. При этом файлами объявляются не только структурированные области памяти, но и аппаратные компоненты, типа блочных или символьных устройств, и объекты межпроцессного взаимодействия как именованные каналы. Этим достигается единый подход к работе с разнообразными объектами, включая однообразное применение политики безопасности.
4. Операционные системы UNIX разработаны программистами для программистов, и опытный пользователь с правами администратора системы имеет возможность реализовывать на своей ЭВМ практически любые компьютерные фантазии. Обращаясь к устройству как к файлу,

пользователь может выполнять действия, совершенно немыслимые для операционных систем Windows*. Так, имея полномочия администратора системы, можно записать информацию на машинный носитель, совершенно игнорируя логическое форматирование и файловую систему, прочитать и отредактировать содержимое оперативной памяти, изменить настройки выполняющегося процесса и т. д.

В 90-х годах большую популярность приобрела ОС семейства UNIX – Linux, которая получила название по имени своего уже легендарного создателя – финна Линуса Торвальдса. Причиной особой популярности этой системы является её принадлежность к свободно распространяемому программному обеспечению. Пользователи имеют право свободно копировать, адаптировать, распространять и применять это программное обеспечение. Исполняемые (бинарные) файлы должны поставляться со своими исходными кодами. Это вызвало интерес не только программистов и пользователей, но и многих государств, которые разрабатывают на базе ядра Linux защищенные версии операционных систем, предназначенные для обработки конфиденциальной информации. В дальнейшем многочисленные клоны на базе ядра Linux будем просто именовать ОС Linux.

Операционные системы Linux представляют большой интерес для специалистов по защите компьютерной информации и сотрудников правоохранительных органов, специализирующихся на выявлении, предупреждении и пресечении компьютерной преступности. Благодаря своим возможностям реализовать любые желания программиста, эти системы превратились в излюбленный инструмент хакеров. Компьютеры с ОС Linux часто используются для подготовки и проведения сетевых атак: сканирования сети, перехвата и перенаправления трафика, удаленных атак на отказ в обслуживании. С помощью этих систем часто осуществляются попытки атак Web-серверов, неправомерного удаленного доступа к базам данных, электронным платежным системам. Даже если бы ОС Linux не использовались в созидательной деятельности, они заслуживали бы изучения в среде специалистов по компьютерной безопасности в качестве программного средства совершения преступлений.

Компьютер, управляемый операционной системой Linux, незаменим при копировании доказательной информации, при анализе уязвимостей вычислительных сетей, при проведении сложных криминалистических исследований ЭВМ. На базе ОС Linux проектируются и успешно функционируют многие системы обнаружения и предупреждения компьютерных атак.

Никто из специалистов не отрицает того, что при своей простоте Linux-системы довольно сложны для администрирования. Отсутствие избыточных функций порой заставляет администратора искать нестандартные решения, например, если необходимо разграничить доступ к информации различного уровня конфиденциальности при большом числе пользователей и функциональных подразделений. Администратору необходимо в

совершенстве знать множество команд и уметь писать разнообразные сценарии. Неограниченные полномочия администратора делают цену его возможных ошибок при управлении операционной системой весьма большой. Многочисленные примеры этого читатели найдут в данном учебном пособии. Эта книга в первую очередь предназначена для будущих и действующих администраторов компьютерной безопасности.

Большинство алгоритмических и программных защитных решений, отшлифованных десятилетиями компьютерных войн, подтвердили свою полезность и необходимость. К сожалению, многие алгоритмы и программы, использованные в системах UNIX за десятки лет, превратились в священную корову, которую никто не осмеливается прогнать с дороги. Автор в пределах своей компетентности попытался обратить внимание своих читателей на эти моменты.

К настоящему времени издано немало замечательных книг по операционным системам UNIX/Linux, авторы которых прошли хорошую школу проектирования, программирования, администрирования или криминалистического (не криминального!) применения этих ОС. Однако на полках книжных магазинов гораздо больше посредственных изданий. Автор, не написавший для Linux ни одной полезной утилиты и не имеющий опыта практического администрирования этих систем, решился на создание собственной книги вовсе не из желания заработать на модной теме. Он не ставил перед собой задачу издать очередное практическое руководство по установке и настройке системы. Автор попытался реализовать стремление во всем разобраться самостоятельно, а также научить читателей думать, исследовать и проверять свои сомнения на практике.

Материалы учебного пособия отшлифованы более чем десятилетней практикой преподавания дисциплины в ведущих университетах г. Екатеринбурга студентам, обучающимся по специальностям «Компьютерная безопасность» и «Информационная безопасность телекоммуникационных сетей», а также специалистам-практикам, связанным с компьютерной безопасностью операционных систем.

Рассматривая вопросы компьютерной безопасности, трудно обойти вниманием интересы пользователей. И все же часто приходится выбирать между ценностью защищаемой информации и удобством пользователей. В связи с этим очень уместно замечание Д. Бэндла [3, с. 65] о том, что систему безопасности трудно сделать удобной и понятной для пользователя (несмотря на известное высказывание Б. Шнайера «сложность – проклятие для безопасности»). Все безопасное в смысле информационной защиты не очень удобно, так как удобство предполагает такие не сочетающиеся с безопасностью качества, как предсказуемость и элементарность. Защитные механизмы неизбежно конфликтуют с удобством и широкой функциональностью универсальных операционных систем. Поэтому в данной книге защитные механизмы рассматриваются без пользовательских удобств, на уровне привычного для администратора интерфейса командной строки.

Защитные механизмы операционной системы даны не от Бога. Они основаны на человеческой логике, осознанных и исправленных ошибках. Но людям свойственно одушевлять вещи, созданные ими или другими людьми. К разряду таких сущностей относятся компьютер, компьютерная программа и, конечно, операционная система, которая, по сути, также является большой и очень сложной компьютерной программой. Автор, хоть и стремится к логической строгости и точности, также не смог избежать выражений типа «предусмотрено системой», «система не позволяет» и т. д.

Один из очевидных недостатков данной книги – невозможность изложения учебного материала в строгой последовательности. Компоненты операционной системы настолько органично связаны друг с другом, что автору постоянно приходится делать ссылки вперед и назад по тексту. Правило «повторение – мать учения» не всегда может прикрыть использование вынужденных повторов. Но, кстати сказать, многими иными авторами такое неудобство вообще не принимается во внимание.

В рамках данного учебного пособия предполагается ограничиться рассмотрением возможностей операционной системы, инсталлированной на автономном персональном компьютере. Операционная система Linux является сетевой, но вопросам защиты информации в сетях Linux в настоящее время посвящают свое внимание множество авторов. В книге не нашли отражения защитные механизмы, реализованные при построении серверов, межсетевых экранов, а также такие программные решения, как Samba, Squid, Apache, т. к. их защитные механизмы представляют лишь набор записей в конфигурационных файлах. По мнению автора, сетевая защита не затрагивает глубинных уровней операционной системы, а реализуется на уровне сетевых сервисов или приложений. Когда защита определяется только размерами и сложностью конфигурационного файла, уже неясно, какие же защитные механизмы здесь использованы.

Учебное пособие включает теоретическую часть, лабораторный практикум, тестовые вопросы и варианты ответов для программированного контроля, а также иные приложения.

Теоретическая часть содержит пять глав, последовательно раскрывающих устройство и механизмы обеспечения компьютерной безопасности основных функциональных компонентов операционной системы. Рассмотрение сущностей операционной системы в рамках книги проводится в привычном логическом порядке: субъекты (пользователи), процессы, файловые объекты и внутреннее строение базовых файловых систем, сетевые возможности ОС.

В *первой главе* рассматриваются структуры и механизмы, обеспечивающие важнейшую функцию безопасности ОС – управление учетными записями и процедурой аутентификации пользователей. Производится оценка уязвимости и безопасности процедуры регистрации пользователей, а также защищенности их учетных записей. Излагаются принципы и порядок назначения общих, специальных и комбинированных прав доступа к

файлам и каталогам. В главу 1 введены практические задачи на разграничение доступа с примерами их решения.

*Вторая глава* посвящена рассмотрению разнообразных аспектов компьютерной безопасности, связанных с процессами. Приводится классификация процессов и дается их характеристика. Изучаются особенности построения команд и интерфейс командной строки. Рассматриваются возможные способы запуска программ в автоматическом и в интерактивном режимах, включая возможности скрытого запуска. Анализируются механизмы многозадачности и средства межпроцессного взаимодействия. Изучаются средства управления процессами. С учетом специфики UNIX-систем рассматриваются консольные атаки и терминальный режим. Надлежащее внимание уделяется аудиту событий безопасности.

Файловые системы Linux рассматриваются с нескольких сторон. В *третьей главе* изучаются действия над файлами различного типа: регулярными файлами, каталогами, символическими ссылками и специальными файлами устройств. Особое внимание уделяется различным видам копирования данных и записи компакт-дисков в режиме командной строки. Рассматриваются способы монтирования разделов дисковой памяти с различными файловыми системами.

*Четвертая глава* посвящена углубленному анализу файловых систем (ФС) **`ext*fs`**. Производится подробное описание архитектуры ФС и ее составных частей. Изложение теоретического материала подробно иллюстрируется примерами исследования структурных элементов ФС с использованием общесистемных и специальных утилит. Рассматриваются механизмы формирования и использования в криминалистических целях временных отметок файлов. Подробно анализируются алгоритмы «ручного» и автоматизированного восстановления логически уничтоженных файлов и каталогов.

Сравнительно краткое изложение сетевых возможностей Linux содержится в *главе 5*. Излагается порядок контроля и настройки сетевых интерфейсов, возможности утилит, позволяющих вести разведку сети и сетевое копирование, а также порядок настройки механизмов фильтрации сетевого трафика с помощью известной программы **`tcpdump`**.

Учебное пособие содержит объемный, хорошо продуманный и многократно апробированный *лабораторный практикум*, с которого и начиналась эта книга. В настоящей редакции книги практикум содержит 8 лабораторных работ, каждая из которых требует для выполнения от двух до четырех академических часов. Цикл лабораторных работ охватывает большую часть изложенного теоретического материала и направлен на его практическое закрепление. Учебные исследовательские задачи, содержащиеся в каждой из лабораторных работ, должны способствовать формированию у обучаемых творческого отношения к защите компьютерной информации, без чего невозможно решение сложных практических задач администрирования операционных систем. Лабораторные работы в доста-

точной степени познавательны, учат размышлять, исследовать и сомневаться и, как правило, вызывают значительный интерес обучаемых.

Каждая лабораторная работа содержит контрольные вопросы и предусматривает составление отчета и защиту. Лабораторный практикум в университетских условиях предполагает наличие специализированного компьютерного класса. В то же время большинство работ может быть выполнено в домашних условиях, в том числе при загрузке персонального компьютера операционной системой Linux с внешнего носителя (Live CD).

В *Приложения* вынесены краткие сведения об использовании и синтаксисе основных команд Linux, которые группируются по своему функциональному назначению.

Теоретический и практический материал главы 4 дополняется краткими сведениями об особенностях логической архитектуры файловой системы **`ext4fs`**. Многие структурные элементы этой ФС еще окончательно не сформировались, и автор не счел возможным излагать этот материал в основной части учебного пособия.

В Приложения вынесены сведения о возможностях и порядке использования новых файловых утилит, написанных студентами УрФУ под руководством автора.

Наконец, Приложения содержат полторы сотни разнообразных тестовых вопросов для программированного контроля знаний. Вопросы ранжируются по сложности. Каждому вопросу соответствует от пяти до семи ответов, причем правильными могут быть несколько ответов.

Автор выражает большую признательность Касько Александру Дмитриевичу за его глубокие и разносторонние познания, нелицеприятную, но конструктивную критику и активную помощь в редактировании учебного материала.

Особая благодарность Желтышевой Екатерине Дмитриевне, сумевшей в рамках дипломной работы не только досконально исследовать элементы файловой системы **`ext4fs`**, но и написать превосходную утилиту **`extview`**, незаменимую при исследовании всего семейства этих файловых систем.

# 1. ПОЛЬЗОВАТЕЛИ И ИХ ПРАВА

Компьютерные системы предназначены для обслуживания людей. Человек, реально управляющий компьютером либо считающий себя таковым, называется его пользователем. Различие между обычными пользователями, администраторами и программистами заключается не только в решаемых задачах, но и в степени их полномочий и компетентности. Так, администратор компьютерной системы одновременно является и пользователем системы защиты, и программистом командных сценариев. Большинство программистов, по сути дела, давно превратились в пользователей интегрированной среды программирования.

Плохо, если управление ответственными вычислительными процессами доверяется некомпетентным пользователям, но гораздо хуже передать его чужому человеку, который может оказаться злоумышленником. Решить библейскую задачу отделения «агнцев от козлищ» должна встроенная в ОС система управления доступом. Защищенная операционная система обслуживает только своих пользователей, которых она должна уметь правильно распознавать.

Человеку свойственно иметь имя. Впрочем, операционную систему не интересуют подлинные имена и фамилии людей, ей необходимо знать лишь учетные имена, по которым каждого из пользователей можно идентифицировать, т. е. отличать от других. Пользователь в системе имеет два идентификатора: символьный и числовой. Символьный идентификатор, именуемый учетным именем пользователя, используется в качестве идентификатора для входа в систему. Он предназначен собственно для самого пользователя и не должен начинаться с цифры, содержать заглавных и русских букв, а также символов типа `* # % ^`. Когда пользователь докажет системе, что он действительно тот, за кого себя выдает, система теряет интерес и к его учетному имени. После успешной аутентификации система помнит пользователя за каждым терминалом только по номеру и помечает этим номером каждый запущенный пользовательский процесс и созданный файл.

Система при регистрации присваивает каждому пользователю уникальный числовой идентификатор (**`UID`** – User ID). Этих номеров намного больше, чем требуется для подсчета пользователей системы. Первоначально для их нумерации в метаданных файлов было зарезервировано два байта, что было достаточно для создания $2^{16}$ = 65536 учетных записей. Впоследствии это число почему-то сочли недостаточным, и соответствующие поля индексного дескриптора в совокупности были увеличены до 4 байтов, что позволяет довести число пользователей одной операционной системы до невообразимой величины, равной 4294967296 (свыше 4 млрд. чел.).

Пользователь с **`UID`** = 0 по умолчанию имеет учетное имя **`root`** и является для системы администратором (суперпользователем). Система не позволяет регистрировать более одного суперпользователя, но это можно

сделать «в обход», путем непосредственного редактирования файла с учетными записями. Подобным образом, как будет показано ниже, можно создать произвольное число учетных записей с нулевым идентификатором, и каждый из таких зарегистрированных пользователей будет обладать правами администратора.

Как говорят, **root** – это не право, а возможность не считаться ни с какими правами. Команды, которые операционная система откажется выполнять от имени суперпользователя, можно в буквальном смысле сосчитать на пальцах одной руки. В этом усматривается определенная угроза компьютерной безопасности, связанная с человеческим фактором. Достаточно много ответственных действий в системе можно выполнить только от имени администратора. Но привычка постоянно работать в системе с полными правами является опасной как для администратора, так и для защищаемой им компьютерной информации. Администратор тоже человек, и ему свойственно ошибаться. Иногда незначительная ошибка при вводе команды может привести к фатальным последствиям, поскольку Linux не отличается нудностью и без необходимости не спрашивает пользователя: «Вы уверены в том, что хотите удалить этот каталог?». Можно привести такой пример:

```
rm –rf /home/john/
```

Этой командой администратор хотел удалить всего один пользовательский каталог

```
rm –rf /home /john/ ,
```

но в командной строке был ошибочно введен лишний пробел, в результате чего уничтожается вся «домашняя» директория с главной ценностью автоматизированной информационной системы – файлами пользователей. В прежних версиях ОС из-за подобной невнимательности можно было удалить целиком корневой каталог.

Обычно первая сотня (или несколько сотен) номеров резервируется системой для так называемых псевдопользователей – неперсонифицированных регистрационных имен: **daemon, bin, sys, nobody** и др., от имени которых выступают компоненты операционной системы. Некоторые из псевдопользователей обладают опасными привилегиями, и, чтобы обычные пользователи не могли их употребить во зло, учетные записи псевдопользователей не содержат паролей и не предусматривают сеанса с командным интерпретатором.

Программы, включенные в состав операционной системы для ее администрирования, являются одновременно и наиболее опасными. Для администраторов, которым часто приходится исполнять рутинные действия над большим количеством файлов, процессов и учетных записей, программисты предусмотрели атрибуты, которые избавляют администратора от на-

доедливых напоминаний операционной системы о потенциальной опасности команды. Это обстоятельство дополнительно повышает цену возможной ошибки администратора.

Права всемогущего суперпользователя традиционно являются объектом противоправных посягательств. Информационный нарушитель может спровоцировать администратора на неосторожное действие. Можно специально подготовить опасную программу и создать условия, чтобы ее запустил именно администратор. Постоянное присутствие суперпользователя в системе не является необходимым, и, если это возможно, администратор должен работать в системе с правами обычного пользователя, приобретая права суперпользователя только на время выполнения необходимых операций. Администратор, работающий в системе под учетной записью суперпользователя, может трансформироваться в любого зарегистрированного пользователя, но при определенных обстоятельствах наличие пользователей-двойников может оказаться нежелательным. Поэтому администратор должен создать и использовать собственную учетную запись с правами обычного пользователя.

## 1.1. Учетные записи пользователей и работа с ними

Подсистема разграничения доступа ОС Linux, наследующая экономные базовые принципы UNIX, не может зафиксировать права каждого из пользователей на каждый файловый объект. По отношению к каждому из файлов пространство пользователей делится на три неравные по численности категории: *одного* владельца файла, членов его *основной* группы и всех остальных зарегистрированных пользователей.

Пользователей в целях удобства администрирования можно объединять в группы, которым также присваиваются символьные имена и уникальные числовые идентификаторы **`GID`** (Group ID). Точнее: при правильном планировании вначале создаются группы, а затем в них включаются пользователи. Существование групп предусмотрено функционированием системных алгоритмов, и администратор должен это учитывать. Если иного не предусмотрено командой или настройками, при создании новой учетной записи пользователя автоматически создается новая группа, включающая этого пользователя и наследующая его имя. Настройками конфигурационных файлов могут предусматриваться группы, в которые пользователи включаются «по умолчанию», например **`users`**. Если этого не учитывать, зарегистрированные независимо друг от друга пользователи могут оказаться групповыми совладельцами файлов с конфиденциальной информацией.

Минимальная численность группы – один пользователь. Поэтому для учета **`GID`** в метаданных каждого файла также предусматриваются поля общим размером в 4 байта.

Для регистрации пользователей и создания групп требуются полномочия администратора. Новая группа создается командой

**groupadd [-g GID] <group_name>**

В этой команде в качестве аргумента минимально необходимо указать уникальное символьное имя новой группы, а номер ей по умолчанию присвоит система. **GID** = 0 присвоено группе администратора, а номера с 1 до 99 (эти значения устанавливаются в файле **/etc/login.defs**) обычно резервируются для псевдогрупп.

Группы совершенно равноправны, но по отношению к конкретному пользователю и его файлам группы могут быть основными и дополнительными. В командах **useradd** и **usermod**, которые упоминаются ниже, основная группа пользователя указывается после параметра **-g**, а дополнительные перечисляются после параметра **-G**. Основная группа для пользователя и его файлов единственна, а дополнительных групп может быть много. Пользователи, включенные в основную группу пользователя, обладают особыми правами на его файлы. Члены дополнительных групп, куда может входить пользователь, имеют общие права на его файлы. В то же время он сам по отношению к файлам пользователей, входящих в дополнительные группы, пользуется групповыми правами. В такой асимметрии заложена возможность разграничения доступа, когда одни пользователи должны иметь больше прав, чем другие.

Утилита **groupmod** имеет такие же параметры, но используется при необходимости модификации уже существующих групп. Впрочем, вся модификация сводится к изменению номера группы **GID.**

Информация о группах содержится в конфигурационном файле **/etc/group**. Часть этого файла приведена на рис. 1.1.

```
root::0:root
bin::1:root,bin,daemon
daemon::2:root,bin,daemon
sys::3:root,bin,adm
adm::4:root,adm,daemon
floppy::11:root
mail::12:mail
news::13:news
audio::17:
video::18:
cdrom::19:
rpc::32:
sshd::33:sshd
ftp::50:
nobody::98:nobody
users::100:
console::101:
alfa::102:
beta::103:ivanov,petrov
sigma::104:john,braun
```

Рис. 1.1. Содержимое файла **/etc/group**

Если открыть этот файл в любом текстовом редакторе либо воспользоваться командой **cat /etc/group**, можно увидеть ряд записей, каждая из которых состоит из трех-четырех полей, отделенных двоеточиями. Первое поле – имя группы, второе – признак группового пароля, который обычно не используется (пустое поле), третье поле – числовой идентификатор группы **GID**. Четвертое поле часто пустует; в него поименно, через запятую и без пробела записываются учетные имена пользователей, включенных в группу дополнительно. Несколько десятков записей выделены псевдогруппам **bin, daemon, sys, adm** и др.

Администратор может поменять пользователю основную группу и переместить его в другую, для чего используется команда

**usermod –g <new_group> <user_name>**

При этом члены его новой группы приобретут соответствующие групповые права на его файлы. Члены старой группы сохранят групповые права на уже созданные файлы пользователя, но не будут иметь никаких групповых прав на вновь создаваемые файлы. Приведенная команда **usermod** будет упоминаться далее по тексту.

Обычный пользователь для смены своей основной группы может воспользоваться командой **newgrp**, однако в этом случае программа запросит у него пароль на вход в новую группу. Если такого пароля не предусматривается («теневой» парольный файл **/etc/grshadow** существует, но обычно бывает пустым), то инициатива пользователя будет оставлена без внимания.

Для удаления пустой группы предусмотрена команда

**groupdel <group_name>**

Группу нельзя удалить, если в ней есть пользователи. Необходимо вначале с помощью уже упомянутой команды **usermod** поименно удалить каждого пользователя этой группы либо перевести их в другие группы (пример будет приведен ниже). Исключить пользователей из группы можно и путем непосредственного редактирования учетных записей в файлах **/etc/group**, **/etc/passwd**. Таким же путем можно удалить и любую группу, в том числе и не пустую.

Любой пользователь с помощью команды **id** может получить сведения об основных и дополнительных группах, причем не только своих, но и другого пользователя. Автору неизвестна утилита, которая отображала бы для произвольной группы поименный состав входящих в нее пользователей.

Для регистрации новых пользователей используются специальные утилиты и сценарии. В качестве аргументов командной строки они принимают следующий набор параметров: имя пользователя, имена групп – основной и дополнительных, их числовые идентификаторы, хэшированный пароль пользователя, срок годности пароля, домашний каталог пользовате-

ля, имя командного интерпретатора. Если какие-либо из этих параметров администратор явно не указывает, они подставляются системой из конфигурационных файлов. Пароль рекомендуется задавать отдельно, и в режиме командной строки для этого используется утилита **passwd**.

Для создания новых учетных записей пользователей предназначена утилита **useradd**, с достаточно большим числом параметров (указаны не все):

```
useradd [-u UID] [-g GID] [-G <add_group_name>]
[-d <dir_home>] [-m] [-e <date_del_user>] <user_name>
```

В этой команде кроме параметров, указанных выше, аргумент **-d** явно указывает полное имя домашнего каталога, с помощью параметра **-m** пользователю выделяется каталог, имя которого совпадает с именем пользователя, а параметр **-e** определяет дату удаления учетной записи. Все параметры, заключенные в [квадратные скобки], могут явно не указываться. В этом случае необходимые элементы учетной записи будут взяты из конфигурационных файлов.

Если для создания учетных записей используется именно утилита **useradd** и администратор не желает каждый раз явно указывать в командной строке все параметры, ему требуется отредактировать конфигурационный файл **/etc/default/useradd**. В числе редактируемых можно указать такие строки:

**GROUP=100** – номер, с которого начинается нумерация пользовательских групп;
**HOME=/home** – каталог, в котором будут создаваться подкаталоги пользователей;
**SHELL=/bin/bash** – полный путь к командной оболочке;
**SKEL=/etc/skel** – имя каталога, с которого делается «слепок» домашнего подкаталога нового пользователя.

Некоторые настройки, используемые по умолчанию, могут быть изменены путем редактирования файла **/etc/login.defs**. В частности, надо уделить внимание параметрам:

**PASS_MAX_DAYS** – максимальный срок «жизни» пароля в днях;
**PASS_MIN_DAYS** – минимальный срок «жизни» пароля;
**PASS_WARN_AGE** – срок в днях до окончания действия пароля, когда об этом уже следует предупредить пользователя;
**PASS_MAX_LEN** – минимальная длина пароля, вводимого пользователем с помощью команды **passwd**.

Подробнее об указанных параметрах будет сказано ниже.

Сценарий **adduser** кажется более удобным тем, что позволяет после ввода команды отвечать на запросы программы, вводя данные в интерак-

тивном режиме. При этом программа подсказывает значения, которые будут введены по умолчанию [в квадратных скобках]. Ввод параметра сопровождается нажатием **<Enter>**. Если администратор согласен с подсказанным параметром, он просто нажимает **<Enter>**. Но командный файл **adduser** присутствует только в дистрибутиве Slackware Linux и дистрибутивах на его базе.

В графической оболочке системы также имеются свои варианты программ с оконным интерфейсом, позволяющие создать новую учетную запись и сделать этот процесс более наглядным.

При необходимости администратор может напрямую редактировать файлы паролей и групп, используя для этого текстовые редакторы, например **vi** или **mcedit**. В качестве редактора по умолчанию используется **vi**, но его можно заменить на другой редактор, определив переменную окружения VISUAL в файле **/etc/profile** командой

```
export VISUAL=mcedit
```

В системе также присутствуют две утилиты **vipw** и **vigr**, специально предназначенные для внесения изменений в файлы паролей и групп. Утилита **vipw** работает непосредственно с файлом **/etc/passwd**, а при использовании ее в виде **vipw -s** – с файлом **/etc/shadow**. Утилита **vigr** работает с файлом **/etc/group**, а при использовании ключа **-s** – с файлом **/etc/gshadow**.

Текстовые файлы **group, passwd** и **shadow**, располагающиеся в каталоге **/etc**, представляют собой отдельные записи, состоящие из полей. Символы в этих полях являются данными для операционной системы. Поля отделяются друг от друга двоеточиями.

Учетные записи пользователей хранятся по частям в двух файлах. Первая часть учетной записи хранится в файле **/etc/passwd**, который доступен для чтения всем пользователям (рис. 1.2).

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
nobody:x:99:99:Nobody:/:/sbin/nologin
petrov:x:1001:101::/home/petrov:/bin/bash
ivanov:x:1002:101::/home/ivanov:/bin/bash
sidorov:!:1003:101::/home/sidorov:/bin/bash
```

Рис. 1.2. Фрагмент файла **/etc/passwd**

Каждая строка этого текстового файла состоит из 7 полей: регистрационное имя пользователя, признак пароля, идентификатор пользователя, идентификатор группы, дополнительная информация, «домашний» каталог пользователя, полный путь к файлу командного процессора.

В приведенном листинге значится один суперпользователь, 25 псевдопользователей и три обычных пользователя. Сами хэшированные пароли в файле **/etc/passwd** обычно не хранятся. Если в учетной записи пользователя вместо признака пароля стоит символ **:x:**, это указывает на то, что хэшированный пароль находится в другом файле – **/etc/shadow**. Однако в некоторых дистрибутивах Linux во втором поле записей файла **/etc/passwd** можно прочитать хэш-функцию пароля, которая является фальшивой и служит, вероятно, для отвлечения внимания начинающих хакеров.

В определенном смысле признак пароля представляет собой наиболее уязвимое место учетной записи. Удаление этого символа в большинстве версий Linux позволяет и обычным пользователям, и администратору входить в систему без пароля. Наличие в этом поле иных символов не предусмотрено.

Блокирование учетной записи производится командой

**usermod -L <user_name> ,**

при этом выполняется вставка символа «!» перед зашифрованным паролем в файле **/etc/shadow**, а по команде

**usermod -U <user_name>**

выполняется обратное действие.

Программа авторизации разрешает вход в систему пользователю с неполной учетной записью. Так, вставив в файл паролей запись из четырех полей:

**echo john::999:99: >>/etc/passwd** ,

администратор обеспечит беспрепятственный (т. е. «беспарольный») вход в систему пользователя **john**. По причине отсутствия у него домашнего каталога этот пользователь после входа окажется в корневом каталоге, а командную оболочку система ему выделит по умолчанию. Четыре поля в учетной записи минимально необходимы, и отсутствие любого из них воспринимается как отсутствие самой записи.

Еще одним уязвимым местом учетных записей пользователей является **UID** – уникальный идентификатор пользователя. Достаточно поменять в третьем поле **UID** обычного пользователя на ноль – и бесправный «юзер» становится еще одним администратором. Число таких администраторов может быть произвольным, главное, чтобы их имена различались между собой.

Путем непосредственного редактирования файлов **/etc/passwd** и **/etc/shadow** можно создать учетные записи с повторяющимися именами. Но авторизоваться в системе сможет только один пользователь, который поименован в файле **/etc/passwd** первым, т. к. выборка строк происходит от начала файла до первого совпадения.

Файл **/etc/shadow** (рис. 1.3) представляет собой продолжение таблицы учетных записей, каждая строка которой состоит из 9 полей, также разделенных двоеточиями:

- регистрационное имя пользователя или псевдопользователя,
- хэшированный пароль,
- число дней, прошедших с полуночи 01.01.1970 до дня последнего изменения пароля,
- минимальное число дней действия пароля со дня его последнего изменения,
- максимальное число дней действия пароля,
- число дней до даты, когда система начнет выдавать предупреждения о необходимости смены пароля,
- число дней со времени обязательной смены пароля до блокировки учетной записи,
- день блокировки учетной записи.

```
root:$1$gka0eOIt$3RXPSZVX9AMLVZ65gXmQA1:13766:0:::::
bin:*:9797:0:::::
daemon:*:9797:0:::::
adm:*:9797:0:::::
lp:*:9797:0:::::
sync:*:9797:0:::::
shutdown:*:9797:0:::::
halt:*:9797:0:::::
mail:*:9797:0:::::
news:*:9797:0:::::
uucp:*:9797:0:::::
operator:*:9797:0:::::
games:*:9797:0:::::
ftp:*:9797:0:::::
smmsp:*:9797:0:::::
mysql:*:9797:0:::::
rpc:*:9797:0:::::
gdm:*:9797:0:::::
pop:*:9797:0:::::
apache:*:9797:0:::::
messagebus:*:9797:0:::::
haldaemon:*:9797:0:::::
nobody:*:9797:0:::::
petrov:$1$omn/uYx4$LUIxQiJwjTRNswF.hcxUz1:14196:0:99999:7:::
```

Рис. 1.3. Содержимое файла **/etc/shadow**

Последнее, девятое, поле зарезервировано и не используется. В принципе можно обойтись всего первыми двумя полями, а пользователи с «пустым» паролем в этом файле вообще не нуждаются. Шесть временных отметок, связанных с паролем, можно установить командой **chage** (change aging – дословно: изменить возраст).

Файл **/etc/shadow** является секретным и недоступен для чтения и записи обычным пользователям. Право доступа к этому файлу имеет только администратор. Впрочем, на короткое время изменения своих паролей право записи в этот файл могут получить и обычные пользователи.

Первый пароль пользователю назначает администратор при создании учетной записи. Обновление паролей происходит в соответствии с выбранной политикой администрирования. В системе предусмотрена команда **passwd**, с помощью которой каждый пользователь может изменить свой пароль (если минимальный срок действия старого пароля был установлен и еще не истек). **Passwd** – это одна из утилит, которая может запускаться обычным пользователем, а выполняется всегда с правами администратора, поскольку ей приходится записывать новые данные в теневой файл **/etc/shadow**, чтение и запись которого разрешены только пользователю **root**. Такие исполняемые файлы имеют установленный бит **SUID**, речь о котором пойдет ниже.

Программа **passwd** запрашивает у пользователя старый пароль и, если он оказывается верным, требует ввести новый. На систему возлагается обязанность проверять вновь введенный пароль по словарю и по длине. Если введенный пароль окажется слишком простым или коротким, программа предупредит пользователя об опасности и запросит у него другой пароль. Обычный пользователь не сможет игнорировать такое предупреждение и вынужден будет подчиниться. Администратор, впрочем, может подтвердить свое намерение ввести простой пароль, и система вынуждена будет повиноваться своему хозяину.

Проверить парольные файлы **/etc/passwd** и **/etc/shadow** на предмет отсутствия ошибок позволяет утилита **pwck** (password check). Утилита способна выявить случайные ошибки, которые были пропущены командами **useradd** и **usermod**, а также допущены при непосредственном редактировании файлов. В частности, выявляются дублирующие строки, связанные с одним именем, пользовательские записи без установленного пароля, незавершенные записи и отдельные поля, наличие файлов, на которые дана ссылка, и др. Но преднамеренные и хитрые искажения парольных файлов этой утилитой не выявляются, и визуальная проверка файлов вовсе не исключается.

Для модификации существующей учетной записи используется команда **usermod**, которая имеет такой же синтаксис, что и команда **useradd**. Эта команда не создает, а модифицирует учетную запись, и после команды следует указывать только те параметры, которые подлежат

изменению. В случае необходимости удаления параметра, назначенного утилитой **useradd**, он удаляется указанием пустого символа, например командой

```
usermod –G “” petrov
```

пользователь **petrov** удаляется из всех дополнительных групп.

Команда **userdel** позволяет администратору удалить учетную запись. До удаления учетной записи пользователь должен завершить сеанс работы. Если пользователь не намерен выходить из сеанса, администратор может послать программе, обслуживающей его консоль, сигнал принудительного завершения. Как это делается, будет показано ниже.

Команда без опций

```
userdel <user_name>
```

удаляет учетную запись пользователя, сохраняя его каталоги и файлы. При задании опции **-r** учетная запись удаляется вместе с каталогами и файлами пользователя. Если пользовательские файлы представляют ценность для организации, а их использование не нарушает прав бывшего пользователя, то администратор может их сохранить. Следует помнить, что в большинстве версий Linux эти файлы сохраняют **UID** своего бывшего владельца. Но стоит системе по умолчанию присвоить новому пользователю идентификатор прежнего владельца – и файлы автоматически перейдут в его собственность. Поэтому администратор, удаляя учетную запись бывшего сотрудника, должен правильно распорядиться его файлами, изменяя их владельца либо помещая их в недоступный для пользователей каталог. Для того чтобы не случалось путаницы с принадлежностью файлов из старых резервных копий, администратору не следует присваивать новым пользователям **UID** прежних сотрудников, которые по той или иной причине покинули организацию. Из этих же соображений при создании новых учетных записей не рекомендуется присваивать **UID** по умолчанию, так как система будет повторно использовать освободившиеся номера.

Поскольку конфигурационные файлы **/etc/group** и **/etc/passwd** доступны для чтения любому пользователю, из них может быть извлечена информация об **UID**, **GID** и символьных именах групп любого пользователя, имеющего учетную запись. Однако проще получить такую информацию с помощью уже упомянутой команды **id <user_name>**. Закрывать эти файлы для чтения нельзя, так как ими пользуются многие утилиты, в том числе и **id**.

## 1.2. Процедура регистрации и ее безопасность

После окончания загрузки операционной системы процесс **init** запускает несколько экземпляров процесса **getty** (get tty – получить терми-

нал), обслуживающего физические консоли либо псевдотерминалы. О терминалах и терминальном режиме будет рассказано ниже. Действия этого процесса во многом определяются объявлением и значениями параметров в конфигурационном файле **/etc/login.defs**. Далее описываются действия, выполняемые программой по умолчанию.

Пользователь по приглашению процесса **getty** вводит свое регистрационное имя. **Getty** в свою очередь запускает дочерний процесс **login**, передавая ему имя учетной записи в качестве аргумента. Программа проверяет файл **/etc/passwd** на предмет наличия соответствующей учетной записи, но не информирует пользователя об ошибках. Сделано это, по-видимому, для того, чтобы не позволить злоумышленнику точно определить имена зарегистрированных пользователей и тем самым облегчить проникновение в систему. Если проверка идентификатора и пароля прошла неудачно, управление будет возвращено программе **getty**, которая вновь запросит регистрацию в консоли.

Поиск имен производится в первых полях учетных записей файла **/etc/passwd** до первого совпадения. Если введенное имя пользователя совпадает с учетным, а второе поле найденной записи окажется пустым, читаются идентификаторы **UID** и **GID** в третьем и четвертом полях, после чего аутентификация считается успешно завершенной. **Login** запускает процесс командной оболочки, указанный в последнем поле соответствующей строки файла **/etc/passwd**, и передает ей управление. Командный интерпретатор **/bin/bash** использует сценарии **/etc/profile**, **.bash_profile**, **.bash_login**, **.profile** в домашнем каталоге пользователя (если такие файлы существуют) и выводит предусмотренное настройками приветствие пользователю с приглашением для ввода команд. Обычно у носителя нулевого **UID** последним символом в строке приглашения является **#**, а для всех остальных пользователей выводится символ **$**. Если администратор считает этот признак демаскирующим, он может изменить строку приглашения командой

```
export PS1=...
```

либо переписать переменную окружения **PS1** в сценарии **/etc/profile** или в одном из конфигурационных файлов **~/.bash_profile**, **~/.bash_login**, **~/.profile**, **~/.bash_logout**.

При совпадении имен и наличии во втором поле установленного признака пароля **:x:** программа **login** запрашивает у пользователя пароль. Если учетное имя не совпадает, либо во втором поле присутствует иной символ, либо пароль оказался неверным, пользователь извещается о некорректной попытке входа в систему. После некоторой паузы программа **login** вновь предложит посетителю зарегистрироваться. Количество повторов, а также тайм-аут на ввод пароля устанавливаются в файле **/etc/login.defs**. При исчерпании числа повторов посетителю необхо-

димо перезагрузить операционную систему, однако на персональном компьютере с эмуляцией нескольких терминалов ничто не мешает ему комбинацией клавиш **<Alt+Fn>** переключиться в иную виртуальную консоль и продолжить попытки входа.

Вводимый пароль контрольными маркерами на экране не сопровождается, что заставляет пользователя быть внимательным и лишает случайного очевидца возможности узнать длину пароля. Последовательность символов пароля кратковременно фиксируется только в памяти процесса **login**, а клавиатурный буфер очищается. После завершения ввода и нажатия **<Enter>** вызывается функция криптопреобразования, а ее результат сравнивается с содержимым второго поля учетной записи, расположенной в файле **/etc/shadow**. При совпадении двух хэш-значений – вычисленного и эталонного – аутентификация считается успешно завершенной. Программа **login** вызывает командную оболочку и передает ей управление. Что характерно – при отсутствии в 7-м поле учетной записи имени командного интерпретатора он все равно будет загружен. Эксперименты показывают, что для успешной аутентификации учетная запись пользователя в **/etc/passwd** должна содержать как минимум первые четыре первых поля: символьный идентификатор, признак пароля, **UID** и **GID**, а в **/etc/shadow** – два первых поля.

Иногда в процессе отладки системы у администратора может возникнуть потребность временно запретить регистрацию новых пользователей. Это можно сделать, создавая файл **/etc/nologin**. Содержимое этого файла значения не имеет, он даже может быть пустым. Такой файл создается командой

```
touch /etc/nologin
```

Пока существует этот файл, ни один новый пользователь, кроме администратора, не сможет войти в систему. Как только необходимость запрета доступа пропадет, администратор должен удалить этот файл.

Результат будет зависеть от используемого загрузчика операционной системы и параметров, установленных в его конфигурационном файле. Так, описанный в [3, 13] доступ осуществляется в ходе стандартной загрузки компьютера с использованием штатного загрузчика **LILO** и конфигурационного файла **lilo.conf**. Приглашение к прерыванию загрузки пользователем предусмотрено в **lilo.conf** в параметре **prompt**. Приглашению к вводу команды выглядит так

```
LILO:
```

Во время тайм-аута необходимо успеть нажать любую клавишу. После этого система остановит загрузку и будет ожидать окончания ввода. Неверно введенный символ можно исправить. Далее следует ввести такую команду

```
linux init=/bin/bash
```

При этом вместо процесса **/sbin/init** запускается командная оболочка с полномочиями администратора. Значительная часть процессов еще не запущена, а файловая система доступна только для чтения. В таком режиме файлы с учетными записями модификации не подлежат. Поэтому далее в командной строке предлагается ввести следующую команду

```
mount –o remount,rw /
```

С ее помощью корневой каталог файловой системы перемонтируется в режиме для чтения и записи. Более полные сведения о возможностях монтирования файловых систем будут приведены ниже.

После этого можно редактировать файл паролей, например, с помощью редактора **vi**:

```
vi /etc/passwd
```

После завершения нельзя сразу перезагружать компьютер. Требуется восстановить исключительный режим чтения:

```
mount –o remount,ro /
```

После перезагрузки утилита **fsck** обнаружит неполадки в файловой системе и устроит ее внеплановую проверку, после которой, скорее всего, автоматически перезагрузит систему. После перезагрузки можно войти в систему без пароля.

Загрузчик **GRUB** (GRand Unified Bootloader) имеет собственный текстовый редактор, который может заменить интерпретатор команд. Перейти в режим редактирования конфигурационного файла системного загрузчика можно с помощью команды **e**. После этого в окно загрузчика будет выведено несколько строк подобного типа:

```
root (hd0,0)
kernel /boot/vmlinuz–2.6.18–5–686 root = /dev/hda1 ro
initrd /boot/initrd.img–2.6.18–5–686
savedefault
```

Опция **root** определяет текущее корневое устройство при загрузке системы. Параметры **(hd0,0)** указывают на фиксированный машинный носитель (магнитный диск) и раздел на нем. Опция **kernel** используется для загрузки ядра. Ее параметр указывает полный путь к файлу, который является ядром. Опция **initrd** сообщает, где находится образ виртуального диска, и имеет значения только на компьютерах со SCSI–дисками.

Выбираем строку **kernel** и опять вводим команду **e**. После этого в конец строки после пробела можно добавить ключевое слово **single** или цифру **1**, указав на необходимость загрузки системы в однопользовательском режиме. Далее нажимаем **<Enter>**, попадаем в предыдущее меню и вводим команду **b** для загрузки.

Если загрузка не удалась, следует попробовать после слова **single** и пробела ввести метку и опцию ядра **init=/bin/bash**, указывая на необходимость загрузки командного интерпретатора. После его загрузки следует повторить команды, ранее перечисленные для **LILO**.

Для того чтобы не позволить пользователю, присутствующему при загрузке системы, стать ее администратором, рекомендуется отредактировать конфигурационные файлы загрузчиков, в том числе предусматривая установку дополнительных паролей, которые будут запрашиваться при попытке пользователя на этапе загрузки перейти в интерактивный режим.

В наши планы не входит рассмотрение устройства и функционирования загрузчиков. Достаточно сказать, что критичные настройки безопасности расположены в конфигурационных файлах **lilo.conf**, **grub.conf** и др. Они традиционно представляют собой строки, содержащие разнообразные параметры и их значения. К параметрам, обеспечивающим безопасность загрузки, относятся следующие:

- **prompt** – включает ввод приглашения при загрузке без ожидания каких-либо нажатий клавиш. Если не предполагается выбирать одну из нескольких загружаемых систем, само приглашение будет совершенно лишней любезностью;
- **timeout**=50 – время ожидания 5 сек. Из тех же соображений тайм–аут можно приравнять к нулю;
- **passwd**=12345.

В файле **/etc/lilo.conf** дополнительный пароль на интерактивную загрузку хранится в открытом виде, поэтому администратору надлежит защитить файл от возможности чтения пользователями.

Если мультисистемная загрузка или тайм-аут в конфигурационных фалах **/etc/lilo.conf**, **/boot/grub/grub.conf** или **/boot/grub /menu.lst** не предусмотрены, существует возможность загрузить операционную систему Linux со сменного носителя. Наиболее компактные загрузочные варианты, пригодные для нейтрализации парольной защиты, могут поместиться даже на одну дискету. Вполне естественно, что загружаемая система не запрашивает никаких паролей и в благодарность превращает пользователя, запустившего ее, в администратора. После этого можно монтировать раздел Linux на фиксированном диске и манипулировать парольными файлами и доступной информацией.

Использование прав суперпользователя само по себе потенциально опасно для системы, поэтому администратор в обыденных ситуациях, когда вводить привилегированные команды не требуется, должен использовать права обычного пользователя. Для этого администратор может зарезервировать для себя какую-либо учетную запись либо, в крайнем случае,

использовать зарегистрированные имена других пользователей. Изменить права доступа можно с помощью утилиты **su** (substitute user – подстановка пользователя).

Если администратор вводит команду **su** с именем зарегистрированного пользователя, он становится им мгновенно, без запроса пароля. Для трансформации **root** может даже использовать заблокированную учетную запись пользователя – если приказывает суперпользователь, система не обращает внимания на такие пустяки. Для того чтобы повысить свой рейтинг и вновь стать администратором, ему вновь необходимо ввести команду **su** без аргументов, а после запроса ввести пароль **root**. В некоторых дистрибутивах возврат прав администратора не сопровождается запросом пароля – это явная угроза безопасности, допущенная программистами. Впрочем, как будет показано ниже, возврат статуса администратора без ввода пароля возможен и иным путем.

Использование таких утилит, как **su**, находится под пристальным вниманием системы и по умолчанию протоколируется в один из файлов аудита. Исполняемый файл с именем **su** обычно является объектом атак со стороны вирмейкеров. Если злоумышленнику удается внедрить в систему фальшивую программу **su** и обеспечить ее запуск при обращении к этой утилите, он может перехватить многие пароли, включая пароль суперпользователя.

Многие авторы указывают на необходимость ликвидации самого имени **root** или присвоения ему **UID** с произвольным номером, мотивируя это тем, что нарушитель, внедрившийся в систему в качестве пользователя, будучи не в силах обнаружить сеанс администратора, не станет атаковать систему. Автор не берется судить, насколько повлияет на поведение нарушителя присутствие пользователя с именем **root.** Но то, что нарушитель якобы не в силах найти в числе работающих пользователей администратора, – сущая фантазия. Любому пользователю доступны утилиты **w** и **id**, и с их помощью нетрудно вывести список пользователей и определить, кто из них имеет **UID**=0.

А вот соображение относительно опасности блокирования учетной записи **root**, высказанное Д. Бэндлом [3], полностью справедливо. Подлинный администратор системы, использующий другое имя, может оказаться беспомощным в случае блокировки консоли, если блокирующая программа будет запрашивать только пароль. Причина заключается в простоте алгоритма и лености программистов. Первая учетная запись с нулевым третьим полем, найденная в файле **/etc/passwd**, будет считаться единственной. Примитивная маскировка подлинного администратора окажется более продуманной, если учетная запись **root** будет перемещена в файле паролей ниже. Опять же, как и в случае с утилитой **id**, не стоит забывать, что файл **/etc/passwd** доступен для чтения и анализа любому пользователю.

## 1.3. Права доступа к файлам

Всего в операционной системе предусмотрено семь типов файлов, которые будут рассматриваться в отдельной главе. Пока достаточно считать, что существуют только обычные файлы и каталоги. Пользователи получают во владение созданные ими объекты файловой системы и имеют определенные системой или администратором права доступа к существующим объектам. Всего существует три первичных и относительно независимых вида доступа (mode) к файлам:

- чтение (**r** – read);
- запись (**w** – write);
- исполнение (**x** – execute).

Запись и чтение файла предполагают его поиск и открытие. На первом этапе система должна найти имя файла в явно указанном каталоге либо, если путь к файлу не указан, но содержится в переменной окружения оболочки, попытаться найти его. Если путь к файлу пролегает через несколько каталогов, каждый из них последовательно открывается и в нем производится поиск имени следующего подкаталога или целевого файла [6].

На втором этапе поиска по найденному имени в файловой системе обнаруживается нужный индексный дескриптор файла и считываются его метаданные, включая права доступа к нему. При открытии файла в числе параметров задается цель его открытия: для чтения, для записи, для добавления либо для чтения и записи. Права доступа к существующему файлу проверяются на этапе его открытия. Если цель открытия файла не соответствует правам пользователя, в доступе будет отказано. Следует отметить, что доступ к файлу означает доступ к блокам, где хранятся его данные. Метаданные недоступного файла не закрываются, и нарушитель может узнать адреса блоков данных, где хранится файл. Угрозу это представляет лишь в том случае, если нарушителю доступны операции с блоками дисковой памяти в обход файловой системы.

Если права доступа совпадают с затребованными, открываемому файлу присваивается учетный номер – файловый дескриптор (не путать с индексным дескриптором!), а найденные блоки данных файла копируются с дискового пространства в оперативную память. Информацию об открытых файлах и их связи с пользователями и процессами можно получить с помощью утилиты **lsof** (list opened file).

Если открываемый файл не существует, он может быть создан с правами доступа, задаваемыми явно или по умолчанию.

Право чтения позволяет считывать блоки данных из файла. Право записи в файл не требует его просмотра. В зависимости от того, какой режим записи предусматривается, данные могут записываться либо в конец файла (дописывание), либо в выбранный сегмент, начало которого обозначается

файловым указателем. При этом данные из буфера программы заменяют прежние данные.

После окончания доступа файл требуется закрыть, что означает разрыв связи между открытым файлом и файловым дескриптором, и сохранить изменения в файле на диск (если файл открывался для записи или добавления).

На самом деле файл при закрытии сохраняется только в дисковом кэше – специально выделенной области оперативной памяти. Обосновывается это тем, что дисковые операции являются самыми продолжительными, и системе не стоит обращаться к дисковой памяти ради такого пустяка, как сохранение изменений в одном файле. Сохранение данных на диск производится обычно в массовом порядке: при переполнении кэша, истечении определенного времени, завершении работы либо перезагрузке системы. Но при создании файла в нем можно предусмотреть специальный атрибут, требующий в исключительном порядке его сохранение сразу на диск. При проведении лабораторных работ по исследованию архитектуры файловых систем и восстановлению данных обучаемые ближе познакомятся с этой особенностью современных файловых систем.

Право исполнения имеет смысл только по отношению к программе. Операционная система не распознает типы обычных файлов, но для программ она делает исключение. Бинарные исполняемые файлы формата ELF распознаются по характерной сигнатуре **`0x7F454C46`** в начале файла.

Запуск файла на исполнение означает создание нового процесса. Первые фазы, связанные с поиском файла и проверкой прав доступа, в принципе совпадают с вышерассмотренными действиями. Полный путь к исполняемым файлам задается в переменной окружения **`PATH`**. Создание процесса также сопровождается копированием исполняемого файла в оперативную память, причем сегменты кода, данных и стека размещаются в раздельных областях памяти.

По отношению к текстовой программе – сценарию – одно только право исполнения ничего не дает. Для запуска сценария требуется еще право на его чтение, т. к. оболочка читает строки сценария, производит их лексический анализ и интерпретацию, т. е. преобразование текстовых команд в бинарный код, который оболочка передает центральному процессору.

Текстовую программу (сценарий) можно запустить, имея только право чтения. Для этого в командной строке требуется указать вначале имя командного интерпретатора, а затем имя сценария, например

```
bash abcd
```

Сценарий можно запустить как самостоятельную программу, если пользователю присвоено право на его чтение и запуск, а в заголоке файла будет указана «магическая» комбинация символов и полное имя командного интерпретатора

```
#! /bin/bash
```

Все зарегистрированные в системе пользователи по отношению к каждому из объектов доступа разделяются на три неравных по численности категории:

1) владелец файла. Им автоматически становится пользователь, создавший файл. Пользователь, скопировавший уже существующий чужой файл, автоматически становится владельцем копии;
2) члены группы, в которую входит владелец. Поскольку один пользователь может являться членом многих групп, здесь имеется в виду *основная*, первичная группа. Право группы владельца асимметрично. Групповые права на файлы и каталоги владельца имеют члены только его основной группы, сам же он пользуется групповыми правами по отношению к файлам пользователей основной и дополнительных групп, в которые его включил администратор;
3) все остальные зарегистрированные пользователи, за исключением владельца файла и членов его основной группы.

Для каждой из категорий определяется набор первичных прав доступа. Вывод информации о правах доступа к объектам файловой системы производится с помощью команды **`ls -l`** (list – список). Первый столбец в выводимой таблице как раз и указывает на тип файла и права доступа к нему.

Права доступа для каждой категории пользователей записываются в бинарном виде и представляют собой восьмеричную цифру. Отсутствующее право доступа обозначается дефисом, а в двоичном виде – нулем. Наличие права отображается латинским символом или единицей. Например:

**`r - x`** = 101 = 5 ;
**`- w x`** = 011 = 3 ;
**`r - -`** = 100 = 4 и т. д.

При создании нового файла права доступа к нему либо указываются явно, либо генерируются автоматически на основании ранее заданной маски доступа. Владельцем файла является его создатель.

Отдельной команды для создания обычного файла не предусмотрено, поскольку эта задача возложена на прикладные программы. Но, тем не менее, файлы можно создавать различными способами. Так, пустой файл может быть создан с помощью команды

```
touch <file_name>
```

Если файл существует, эта команда с определенными аргументами может использоваться для изменения временных отметок последнего доступа и модификации файла.

Файл может быть создан путем перенаправления вывода, о котором будет сказано ниже. Так, файл, состоящий из одной строки, можно создать командой

```
echo "Да будет файл!" > abcd
```

**echo ”” > pystoj_fail**

Файл может быть создан с помощью программы чтения файлов, если объект не будет явно задан. При этом в создаваемый файл направляются символы, введенные с клавиатуры.

**cat > abcd**

После ввода этой команды можно ввести символьные строки, переводя строку с помощью **<Enter>** и закрывая файл комбинацией клавиш **<Ctrl+D>**. Аналогичное можно проделать командой блочного копирования, указав в ней только имя создаваемого файла, например

**dd of=abcd**

Права доступа к вновь создаваемым или копируемым обычным файлам определяются маской, которая задается с помощью команды **umask.** Вызов этой команды без аргументов приводит к выводу текущего значения маски. Значение маски – восьмеричное число, которое вычитается из 0777 для исполняемого файла и каталога либо из 0666 – для неисполняемого файла. Например, для исполняемого файла или сценария **umask** = 0022 означает режим доступа 0777 – 0022 = 0755 (111 101 101 = **rwxr-xr-x**). Ноль в старшем разряде маски доступа указывает на то, что эффективные права автоматически не наследуются.

Действующее по умолчанию значение **umask** находится в файле **/etc/profile**. Каждый пользователь вправе изменить маску доступа для своих файлов по собственному разумению. Для этого ему достаточно ввести команду

**umask 0XXX**

или

**umask XXX**

с указанием нужного восьмеричного числа **XXX** (ноль в старшем разряде можно не указывать, так как эффективные права доступа при создании файлов не наследуются) либо записать приведённую выше команду в файл **.bash_profile** в своём домашнем каталоге. Администратор может изменить этот порядок, для чего ему потребуется сделать недоступной для пользователей утилиту **umask**, а также не допустить возможности редактирования пользователями конфигурационных файлов **.bash_profile**, присваивая им дополнительные атрибуты.

Создавая новый каталог, можно явно определить права доступа к нему. Это производится с помощью утилиты **mkdir** и опции **-m**, например

**mkdir –m 750 /home/petrov/mail**

В отношении каталогов права доступа интерпретируются несколько

иначе, чем для обычных файлов. Право на чтение в каталоге дает возможность искать в нем имена файлов. Однако если права на каталог ограничены только чтением, ничего кроме одних имен пользователь в нем не увидит (при условии, что файлы в нем есть). Чтобы посмотреть, как это будет выглядеть, достаточно выполнить команду **`ls / .`**

По отношению к каталогу право на исполнение – это возможность открыть каталог в целях поиска файлов или пройти *сквозь* него, если он является промежуточным объектом в полном пути к искомому файлу. Для вывода информации об атрибутах файлов с помощью команды **`ls -l`** также необходимо право исполнения каталога, поскольку для этого придется искать метаданные файлов. Большинство каталогов в файловой системе предоставляют всем зарегистрированным пользователям права на чтение и исполнение.

Право записи в сочетании с правом входа в каталог означает возможность создавать в нем новые файлы, удалять из него, а также копировать или перемещать в него существующие файлы. Каталог с правами записи и исполнения, но без права чтения, часто называют «темным», так как увидеть записанные в него объекты невозможно. Из «темного» каталога можно также копировать, перемещать или удалять файлы, но только в том случае, если известны их имена. Отдельно взятое право на запись в каталог никаких реальных возможностей не означает.

Примером «темного» каталога может являться каталог на ftp-сервере, который служит приемником в системе файлообмена. Любой посетитель сервера может скопировать свои файлы в этот каталог, но воспользоваться лежащими там файлами других пользователей он не сможет, т. к. каталог выглядит пустым. Эта мера вполне разумна, поскольку владелец сервера, если он не желает стать распространителем вредоносных программ и иной опасной информации, должен вначале проверить содержание поступивших файлов.

Право входа в каталог не эквивалентно праву выхода из него. Если пользователь каким-то чудесным способом сумел войти в недоступный для него каталог (например, администратор в своем каталоге **`/root`** с помощью команды

**`su <user_name>`**

«превращается» в обычного пользователя), то выйти обратно он сумеет беспрепятственно. В то же время каталог можно попытаться превратить в «тюрьму» для потенциально опасного пользователя. Простейшая «тюрьма» состоит из двух каталогов – внешнего и внутреннего. Внешний каталог недоступен пользователю на исполнение. По идее, если каким-то путем «поместить» пользователя внутрь, то наружу он выйти не сумеет.

Проверим это предположение простым экспериментом. При наличии в системе авторизованных пользователей, находящихся в своих домашних каталогах, от имени администратора командой

**chmod 700 /home**

временно «закроем» для пользователей внешнюю границу «домашнего» каталога. Переключившись в консоль любого пользователя, убедимся, что пользователи при запуске команд ничем, кроме обычных прав доступа, не ограничены. Пользователи не могут преодолеть воздвигнутый барьер с помощью команды **cd..**, но «перепрыгнуть» через него в любой доступный каталог (например, командами **cd /** или **cd /tmp**) вполне в состоянии. Но вернуться в свои каталоги пользователи уже не смогут. Как видно, создание «тюрьмы» для *внутренних* пользователей перспектив не имеет.

Однако свобода перемещения *внешних* пользователей по дереву каталогов должна быть ограничена. Например, клиенты Web-сервера или FTP-сервера ни в коем случае не должны получать доступ за пределы «гостевых» каталогов.

Этот подход предусматривает запуск специальной утилиты, которая называется **chroot** (change root – изменить корневой каталог). С ее помощью некоторый системный сервис, обслуживающий сетевых клиентов, «запирают» в специально созданной «ветке» дерева каталогов, объявляя точку монтирования «корнем» дерева. Делается это с помощью команды

**chroot <dir> <command>**

Пользователь, попавший в это пространство, может перемещаться в пределах этой «ветки», не угрожая основной части файловой системы. Подняться выше директории, которая объявлена корневой, пользователь не может, поскольку родительским каталогом для «корня» он сам и является. Для правдоподобия подкаталоги этой резервации называют по аналогии с основными каталогами файловой системы и копируют в них все необходимые для работы и камуфляжа файлы.

Файловые «тюрьмы» не являются местами лишения свободы. Они должны содержать внутри себя все необходимое для добросовестного пользователя – подкаталоги, программы, библиотечные, конфигурационные и справочные файлы.

Права на доступ к уже существующему файлу или каталогу можно изменять. Администратор может изменить права доступа к любому файлу, пользователь – только к своим файлам. Производится это с помощью утилиты **chmod** (change mode – изменить режим). У этой команды несколько форм записи, с которыми можно познакомиться с помощью краткого справочника в этой книге либо с помощью электронных руководств **man** или **info**. Ее наиболее компактная форма

**chmod XXXX <file_name>**,

где **XXXX** – 4 восьмеричных цифры, означающих права доступа к файлу: эффективные права, права владельца, его группы и всех остальных пользователей.

Обычно владельцу файла предоставляются полные права, а членам его группы и всем остальным пользователям – только самые необходимые. Но владелец файла вполне может ограничить себя в правах. Например, задав режим доступа к файлу в виде

**chmod 077 <file_name>**,

он предоставляет полные права на файл своей группе и всем зарегистрированным пользователям, обделяя себя. С такой же легкостью он может восстановить справедливость, поскольку является владельцем файла. Иногда к таким трюкам вынужден прибегнуть администратор, когда задача разграничения доступа обычным способом не решается.

С каждым процессом в системе связан идентификатор пользователя **UID**, от имени которого процесс запущен. Алгоритм, реализованный системной функцией, сравнивает **UID** процесса и **UID** файла и, обнаружив, что к файлу обращается его владелец, проверяет только его права доступа. Если у владельца нет нужных прав, – ему будет отказано в доступе. Его права как члена своей основной группы или иного пользователя даже не будут проверяться. Но пользователь, ограничивший себя в правах, может легко исправить положение, переназначив права доступа в свою пользу, – ведь он остается владельцем этого файла и имеет право применить к нему команду **chmod.**

Система не оценивает целесообразность присвоения обычным файлам и каталогам тех или иных прав доступа. Администратор или владелец файла может указать любые, даже самые нелепые права доступа к файлу, и они будут без возражений установлены.

Передача прав владения файлами в системе также предусмотрена. Она производится с помощью утилиты **chown** (change owner – сменить владельца) командой

**chown <user_name> <file_name>**

Обычным пользователям не разрешается передавать свои файлы другим пользователям, включая администратора, поскольку с позиций компьютерной безопасности такое действие рассматривается не как акт доброты, а как возможная информационная атака. Во-первых, передавая свои файлы администратору, пользователь может создать предпосылку для случайного запуска вредоносной программы с правами суперпользователя. Во-вторых, создав файл неограниченного объема и передав права на него другому пользователю (например, из числа своих недругов), пользователь может переполнить дисковую квоту атакуемого и вызвать отказ системы в его обслуживании. Поэтому в ОС Linux передавать права на объекты файловой системы может только администратор.

Права на удаление файла не предусмотрено, и любой пользователь может удалить любой файл, находящийся в доступном для него на запись и исполнение каталоге. Никаких прав доступа на удаляемый файл при этом

не требуется. Для защиты файлов, находящихся в общедоступных каталогах, от несанкционированного удаления создателям системы когда-то пришлось предусмотреть дополнительный атрибут, именуемый **sticky** («липкий») бит (это название давно утратило свое первоначальное назначение). Наличие этого атрибута в дополнение к первичным правам на запись и исполнение в каталоге разрешает удалять из него файлы только их владельцам. Само собой разумеется, что администратора подобный запрет не касается. Отображается этот дополнительный атрибут символом **t** вместо символа **x** для всех пользователей, например **drwxrwxrwt**. Если этот атрибут был определен, а права на исполнение каталога для группы пользователей «другие» не предусмотрено, символ **T** будет отображаться заглавным. Однако на самом деле дополнительный атрибут **t** является дополнением к праву на *запись*, и, если оно не установлено, то и **sticky**–бит никакого смысла не имеет.

Ранее уже обращалось внимание на существование особых команд, которые могут запускаться обычным пользователем, а выполняются с правами администратора. Такой командой, в частности, является **passwd**, поскольку ей приходится записывать новые данные в теневой файл **/etc/shadow**, чтение и запись которого разрешены только пользователю **root**. Такие исполняемые файлы имеют так называемый эффективный идентификатор **SUID** (Superuser UID).

Наряду с идентификатором **SUID**, но гораздо реже, используется эффективное право для группы владельца **SGID**. Команда **ls –l** отображает эффективные права символом **s** вместо символа **x** для владельца файла или его группы, например **rwsr-s--x**. Если при назначении прав этот атрибут был определен, а прав на исполнение файла для владельца не предусмотрено, символ **S** будет отображаться заглавным.

Всего в операционной системе насчитывается несколько десятков утилит, выполняемых с правами администратора или его группы. Их полный список может быть получен при запуске поисковой команды

**find / -perm +6000**,

где аргумент **-perm** является сокращением от permission – разрешение, а «плюс» перед числом указывает на объединение признаков **SUID** + **SGID**.

Назначение программ, имеющих установленные биты **SUID** и **SGID**, как правило, хорошо известно, а использование дополнительных прав обосновано. Если программа, содержащая бит эффективных прав **SUID** или **SGID**, не содержит ошибок и недокументированных возможностей, то ее запуск опасности не представляет. Тем не менее администратор должен по возможности сократить число таких программ (путем снятия опасных атрибутов) и периодически контролировать файловую систему на предмет

появления новых файлов с подобными свойствами. Наибольшую опасность представляют попытки внедрения таких программ на сменных машинных носителях. Для исключения такой опасности монтирование сменных носителей необходимо производить с указанием опции **nosuid**, что означает сброс опасных меток при монтировании.

На систему также возлагается функция «нераспространения» потенциально опасных свойств. Все операции, связанные с копированием, перемещением и переименованием файлов с установленными битами **SUID** и **SGID**, завершаются сбросом опасных атрибутов для преобразованных файлов.

В файловых системах **ext*fs** предусмотрены дополнительные свойства файловых объектов, обеспечиваемые файловой системой. В настоящее время поддерживаются следующие из них:

**i** – защита от любых изменений файла, включая изменение временных отметок и создание жестких ссылок;

**a** – запрет любых операций, кроме добавления данных;

**S** – синхронные обновления файла, при установке которых новые данные немедленно записываются на диск;

**A** – неизменяемость временной отметки последнего доступа к файлу;

**d** – игнорирование файла при операциях резервного копирования, выполняемого командой **dump**, что позволяет не расходовать дисковое или ленточное пространство на сохранение не очень нужных файлов.

Некоторые зарезервированные свойства, такие как создание и сохранение копии файла при его удалении, автоматическое сжатие и декомпрессия при записи/чтении файла, гарантированное стирание блоков данных при удалении файла, – безусловно полезны, однако поддерживаются не всеми версиями файловых систем Linux. Всего в метаданных файла предусмотрено 16 дополнительных свойств, 13 из них на момент написания книги документировано.

Установка дополнительных свойств файла производится с помощью команды

**chattr +(-)(=) option <file_name>**,

где знаки означают:

«**+**» – установление атрибутов,

«**-**» – удаление атрибутов,

«**=**» – установление *только* атрибута, указанного после знака равенства.

Следует обратить внимание на то, что дополнительные свойства файлов в действительности не являются дополнениями к базовым правам и вполне автономны.

Например, дополнительный атрибут **a** позволяет производить только дописывание информации в конец файла и вовсе не является дополнением к базовому праву записи в файл. Установка этого атрибута может производиться на файл, к которому ни одному из пользователей нет права записи. После установки администратором атрибута **a** на файл любого из пользо-

вателей он становится защищенным от любых изменений, включая запись, удаление файла или изменение права доступа. Этот запрет распространяется и на самого администратора. И никто, кроме него, не имеет права что–либо дописывать в такой файл.

Такой атрибут может быть временно установлен на некоторые файлы журналов, чтобы демон **syslogd** мог записывать в них заданные события, а злоумышленник не имел возможности удалять из них уличающие его записи. Конечно, при обновлении журнала (т. е. при удалении самого «старого» файла) этот атрибут потребуется снять. Для того чтобы злоумышленник, проникший в систему с правами суперпользователя, не смог быстро снять этот атрибут, настоящий администратор должен «спрятать» или переименовать утилиту **chattr**.

Установив дополнительный атрибут **i**, администратор не сможет *случайно* удалить файл, записать что-либо в него, изменить права доступа, ни даже создать или удалить жесткую ссылку. Соответственно этого не сможет сделать и никто иной. Этим весьма полезным свойством необходимо пользоваться в отношении наиболее ответственных исполняемых и конфигурационных файлов. Установка подобного атрибута ни в коей мере не является защитой файла от преднамеренных действий администратора, поскольку ее просто установить и не менее просто отменить. И не следует забывать, что защита на уровне файловой системы преодолевается путем доступа напрямую к дисковой памяти с помощью шестнадцатеричных редакторов.

Дополнительные атрибуты файла командой **ls** не отображаются, и для их чтения необходимо воспользоваться командой

```
lsattr <file_name>
```

Команда выводит 16 битовых флагов с указанием дефисов на месте отсутствующих свойств и характерных символов на месте установленных. Аналогичная команда без имени файла выведет дополнительные атрибуты файлов из текущего каталога.

## 1.4. Комбинированные права доступа

Логические объекты файловой системы (файлы) являются носителями своеобразных меток, которые привычно называют *правами доступа*. Некоторые метки действительно означают право выполнения определенного *действия* пользователя над этим объектом. Другие обозначают *состояние* защиты или *невосприимчивость* по отношению к определенным действиям. Третьи гарантируют в ходе предписанных действий наступление конкретного *результата*.

Существует 4 метки выполнения определенного *действия* пользователя над файловым объектом, которые соответствуют первичным правам на доступ к файлам:

- **r (read)** – право чтения файла, т. е. возможность его считывания из

памяти в целях отображения или воспроизведения;

- **w (write)** – право на запись и сохранение в файле своих данных, в том числе с возможностью замены и удаления уже имеющейся там информации;
- **x (execute)** – право на запуск (исполнение) файла;
- **(owner)** – право на владение и распоряжение файлом. На самом деле это не отдельная метка, а проверка условия (**UID** процесса = **UID** объекта). Владелец файла имеет на него полные права, включая право на удаление (при наличии права на запись в каталог) и изменение прав доступа к файлу.

Пятым первичным правом можно считать право суперпользователя. Наличие у пользователя идентификатора **UID** = 0 позволяет ему манипулировать объектами без проверки прав доступа.

Из меток невосприимчивости к действиям следует отметить:

- **i (immutable)** – свойство сохранности файла в неизменном виде. Как уже указывалось выше, оно обеспечивается дополнительным атрибутом файла, устанавливаемым командой **chattr +i**;
- **a (appendable)** – запрет любых операций, кроме добавления данных.

Третий тип меток может быть представлен свойством **s**, указывающим на необходимость физического стирания данных, принадлежащих удаляемому файлу.

На основе первичных прав с добавлением дополнительных свойств можно синтезировать несколько производных прав:

- **s (suid)** – комбинированное право на запуск программы от имени ее владельца. Обеспечивается при наличии основного права владельца на запуск файла и дополнительного бита **SUID**;
- **t (sticky)** – комбинированное право защиты файлов от удаления пользователем, не являющимся владельцем этих файлов. Устанавливается только для каталога, в который пользователи имеют право записи и исполнения;
- **c (create)** – право создания файла в определенном каталоге. Состоит из прав записи и входа в этот каталог;
- **d (delete)** – право на удаление файла из определенного каталога. Также состоит из прав записи и входа в этот каталог, а при установленном «флажке» **t** – право на удаление файла его владельцем;
- **cp (copy)** – право копирования файла в другой каталог. Кроме права чтения файла, требует прав поиска и чтения в исходном каталоге, а также прав поиска и записи в целевой каталог;
- **mv (move)** – право перемещения файла в другой каталог. Кроме права чтения файла, требует полных прав на исходный каталог, а также прав поиска и записи в целевой каталог;
- **ln (link)** – право создания непосредственных («жестких») ссылок

на существующие файлы. Требует прав поиска и записи в каталоге, где создается жесткая ссылка.

Обладание некоторыми правами автоматически означает возможность приобретения и иных прав. Так, владение файлом является правом его создателя. Обладание правами записи и исполнения в любом каталоге делает возможным создание файла. Создание файла превращает его создателя во владельца, а владелец имеет право распоряжаться «имуществом» по своему усмотрению. Соответственно владелец имеет право удалять свои файлы. Заблокировать эту цепочку зависимых прав можно, запрещая пользователю запись в соответствующий каталог.

Право на чтение какого-либо файла делает возможным его копирование, а владелец копии может устанавливать права на нее по своему усмотрению. Для того чтобы при копировании не происходило наследование «опасных» прав прежнего владельца, эффективные права на исполнение файла программы в его копии удаляются.

Сложные права доступа можно изображать в виде булевых соотношений в аналитической или табличной форме. Например, требуется скопировать файл **./a/b/file1** в каталог **./c/d**. Минимально необходимые права на файл и каждый из каталогов определяются с помощью булева выражения:

**cp = a(rwx) & b(rwx) & c(rwx) & d(rwx) & file1(rwx),**

где символом **&** обозначена операция конъюнкции, или логического умножения. С учетом фиксированного размещения битов, обозначающих права чтения, записи и исполнения, булево выражение для операции копирования пользователем *чужого* файла может быть представлено в виде

**cp = a(001) & b(101) & c(001) & d(011) & file1(100)**

В случае, если известно имя копируемого файла, права на чтение в каталоге **b** уже не требуется:

**cp = a(001) & b(001) & c(001) & d(011) & file1(100)**

Аналогично могут быть заданы логические условия для перемещения файла **./a/b/file1** в каталог **./c/d**:

**mv = a(001) & b(111) & c(001) & d(011) & file1(100)**

Для удаления *своего* файла **./a/b/file2**, имя которого предварительно нужно прочесть, необходимы права только на каталоги :

**rm = a(001000000) & b(111000000)**

Если пользователь знает имя своего файла, который он намеревается удалить, и может без ошибки записать его в командной строке, права на чтение из подкаталога **b** ему может не потребоваться:

**rm = a(001000000) & b(011000000)**

Для удаления *чужого* файла **./a/b/file2** потребуются следующие права:

**rm = a(001001000) & b(0001111010000)** ,

где старшие нулевые разряды указывают на отсутствие эффективных прав доступа, в том числе 4-й разряд – на отсутствие стикки-бита, который не позволяет удалять чужие файлы.

Для создания жесткой ссылки на чужой файл **./a/b/file3** из другого каталога **./c/d** также не требуется прав на сам файл:

**ln = a(001001000) & b(101101000) & c(001??????) & d(011??????)**

Аналогично можно записать булевы выражения для иных сравнительно сложных операций.

### 1.5. Решение практических задач на разграничение доступа

Базовые возможности ОС Linux в плане разграничения доступа субъектов к объектам весьма скупы и рациональны. Администратору при создании учетных записей пользователей и определении первичных прав доступа к файлам и каталогам порой приходится решать нелегкие задачи. Тем не менее большинство практических задач может быть успешно решено, причем несколькими способами.

Администратор при разграничении доступа может оперировать группами, учетными записями пользователей (и отдельными полями этих записей), правами на файловые объекты, а также их дополнительными свойствами. К сожалению, сложность практических задач не позволяет предусмотреть строгие алгоритмы для манипуляции правами доступа. В каждом конкретном случае задача разграничения доступа является эвристической и имеет несколько решений. Сформулируем ряд таких задач по возрастанию сложности и для некоторых из них предложим варианты решений.

1. В системе, где обрабатывается только открытая информация, зарегистрировано N пользователей с одинаковыми правами. У каждого пользователя имеется собственный (принадлежащий его владельцу) подкаталог с файлами, который в свою очередь размещен в каталоге **/home**. Создание и удаление файлов в каталоге, а также модификация существующих файлов разрешены только владельцу каталога. Файлы не являются программами. Чтение и копирование любого пользовательского файла разрешено каждому пользователю. Перемещение и копирование файлов (своих или чужих) в каталог другого пользователя не допускается. Пользовательских файлов вне домашних каталогов существовать не должно. Требуется определить владельцев и права доступа к катало-

гу **/home**, подкаталогам и файлам пользователей. Существует ли возможность нарушения запрета? В чем она заключается?

***Решение**. В этом довольно простом случае можно обойтись без планирования групповых прав, которые для определенности будем просто исключать. Владельцем каталога **/home** должен являться **root**, а права доступа к каталогу определяются маской доступа **drwx---r-x**. Будем считать, что все остальные каталоги файловой системы пользователям на запись недоступны. Этим допущением обеспечивается невозможность создания, перемещения или копирования файлов вне домашних каталогов (на самом деле обычным пользователям разрешен полный доступ в каталоги **/tmp**, **/var/tmp**, **/dev/shm**).*

*Определимся с минимально необходимыми правами пользователей на их домашние каталоги. По условию владельцы каталогов должны иметь возможность выполнять все действия с файлами, исключая их запуск. Следовательно, владельцы каталогов должны иметь на них полные права. Поскольку нам неизвестно, в какие группы входят те или иные пользователи, групповые права на файлы и каталоги исключим. Другие пользователи имеют право входить в любой домашний каталог и читать (следовательно, копировать) из них файлы. Но создавать свои файлы, копировать и перемещать в них, а также удалять из них любые файлы пользователи не вправе. Весь этот набор запретов обеспечивается отсутствием одного из базовых прав – на запись. Установки дополнительного бита **t** в данном случае не требуется. Таким образом, права доступа к домашним каталогам определяются строкой **rwx---r-x**.*

*Права доступа пользователей на их файлы описываются строкой **rw----r--**. Следовательно, пользователи должны иметь по умолчанию маску доступа на файлы **umask=0072**.*

*Возможность нарушения запрета существует, поскольку пользователи могут по своему усмотрению изменить право доступа к своим каталогам и маску доступа для вновь создаваемых или копируемых файлов.*

2. В системе, где обрабатываются открытые данные и информация, составляющая коммерческую тайну, зарегистрировано N пользователей. Из них M привилегированных пользователей (M < N) имеют равный доступ к коммерческой тайне. Функционально-зонального разграничения доступа не предусматривается (считаем, что все пользователи работают в одном подразделении, без какой–либо специализации). Подкаталоги и файлы всех пользователей размещены в каталоге **/home**, принадлежащем администратору. В каталогах привилегированных пользователей разрешено хранить только конфиденциальные данные. Пользовательские файлы не являются программами. Все пользователи имеют доступ ко всей открытой информации без права модификации чужих файлов, а также создания и удаления файлов в чужих каталогах. Читать и копировать конфиденциальные файлы могут только привилегирован-

ные пользователи, но они не должны создавать своих файлов в каталогах других пользователей, а также изменять или удалять чужие конфиденциальные данные. Привилегированные пользователи не имеют прав на копирование и перемещение конфиденциальных файлов в каталоги других пользователей, в том числе при их содействии.

Требуется определить владельцев, группы и права доступа к домашним каталогам и файлам пользователей. Учесть, что при создании новых пользователей без явного указания групп они (группы) создаются по умолчанию. Существует ли возможность нарушения запрета пользователями? В чем она заключается?

***Решение**. В данном случае без использования групп и групповых прав уже не обойтись. Администатору достаточно создать одну группу **private** и включить в нее всех привилегированных пользователей. Все остальные пользователи могут быть членами собственных групп, куда входят только они сами. Однако по умолчанию не исключено их членство в общей группе **users**. Этот нежелательный фактор следует учесть и нейтрализовать.*

*Рассмотрим требования на доступ к каталогу и файлам на примере пользователя **ivanov** из группы **private**. Пользователь должен иметь возможность выполнять все действия со своими файлами, исключая их запуск. Следовательно, он должен иметь на свой каталог полные права. Члены его основной и единственной группы должны иметь право входа в каталог **/home/ivanov** и чтения из него. Поскольку открытой информации в каталогах привилегированных пользователей нет, всем остальным пользователям доступ к этим каталогам должен быть запрещен. Таким образом, права доступа пользователей к каталогу **/home/ivanov** должны определяться маской защиты **rwxr-x---** .*

*Права владельца на конфиденциальные файлы должны разрешать их чтение и запись. Члены группы **private** могут файлы читать, а следовательно, и копировать. Лишение их права записи не позволит модифицировать чужие файлы. Доступ остальных пользователей к конфиденциальным данным запрещен на уровне каталога, однако лишний запрет не помешает. Таким образом, права доступа пользователей к файлам, хранимым в каталоге **/home/ivanov**, должны определяться строкой **rw-r-----** .*

*Определим права доступа к каталогу и файлам обычного пользователя, работающего с открытой информацией, на примере имени **petrov**. Пользователь должен обладать на свой каталог полными правами, групповые права по причине неопределенности исключаются, а для всех пользователей требуются права входа и чтения в каталоге. Владелец открытых файлов имеет на них права чтения и записи, групповые права исключаются, а для всех пользователей должно предусматриваться только право чтения. Соответственно права пользователей на каталог*

***/home/petrov*** *определяются строкой* ***rwx---r-x*** *, а на хранимые в нем файлы – строкой* ***rw----r--*** *.*

*Если привилегированный пользователь намеренно или случайно попытается скопировать чужие конфиденциальные данные в каталог другого пользователя, ему будет отказано в доступе, т.к. правом записи в каталоги других пользователей он не обладает.*

*Как и в предыдущем случае, существует возможность нарушения запрета, поскольку пользователи могут по своему усмотрению изменить права на доступ к своим каталогам и файлам.*

3. Дополнение к предыдущей задаче заключается в том, что в подкаталогах привилегированных пользователей могут храниться созданные ими *открытые* файлы, чтение и копирование которых разрешено всем, и *конфиденциальные* файлы, с которыми могут работать только их владельцы и члены группы ***private***. Какие дополнительные меры по разграничению доступа необходимо предусмотреть администратору ?

***Решение.*** *Права на каталоги и файлы обычных пользователей не изменятся, поэтому рассмотрим только права доступа к конфиденциальным данным пользователя* ***ivanov****. Поскольку в каталоге* ***/home/ivanov*** *появились открытые файлы, которые имеют право читать все пользователи, требуется открыть им этот каталог для поиска и чтения. Права на каталог* ***/home/ivanov*** *должны обеспечить свободный доступ на поиск и чтение как для членов конфиденциальной группы, так и для всех остальных пользователей, т. е.* ***rwxr-xr-x****. Права на конфиденциальные файлы будут выглядеть аналогично вышерассмотренному случаю* ***rw-r-----****, а права на открытые файлы –* ***rw-r--r--****. Как видно, использования прав на доступ к файлам вполне достаточно для решения задачи.*

4. Отличие от предыдущего задания состоит в том, что в организации есть группа руководителей численностью K < M. Руководители имеют право читать и копировать в свои каталоги любые файлы, но лишены прав на создание, удаление или модификацию файлов в каталогах пользователей. Каталоги и файлы руководителей должны быть доступны только им самим. Требуется определить владельцев, группы и права доступа к домашним каталогам и файлам пользователей. Решить задачу читателям предстоит самостоятельно.

5. В организации, работающей с информацией ограниченного доступа, все пользователи имеют допуск к коммерческой тайне. Сведения, содержащие коммерческую тайну, обрабатываются в трех подразделениях, сотрудники которых образуют пользовательские группы **alfa**, **beta** и **gamma**. Внутри каждого из подразделений пользователи имеют право совместно работать с конфиденциальной информацией. Для конфиден-

циальных документов подразделения создается отдельный подкаталог с одноименным названием, в котором хранятся файлы, доступные для чтения и записи любому пользователю данной группы. Владельцами файлов становятся сотрудники, которые эти файлы создают. Файлы с открытой информацией не создаются. По возможности следует предусмотреть защиту от случайного стирания результата длительной коллективной работы, которую можно уничтожить – точнее, зачеркнуть простой командой

**echo Привет! > <file_name>**

Доступ к конфиденциальным данным со стороны сотрудников иных подразделений должен быть исключен.

***Решение.*** *Первым шагом должно стать создание трех групп:*

***groupadd alfa; groupadd beta; groupadd gamma***

*Для каждой группы администратор создает одноименный групповой каталог. Следовательно, все три таких каталога по умолчанию будут принадлежать администратору. Но администратор не должен быть владельцем созданных каталогов, поскольку это может привести к наследованию пользователями опасных групповых прав. Гораздо лучше создать в каждой группе одну фиктивную учетную запись пользователя, которому предстоит получить во владение групповой каталог. Допустим, что такими пользователями будут* ***starkov***, ***sidorov*** *и* ***smirnov****. Действующих пользователей использовать для этой роли нельзя, так как владельцев каталогов предполагается лишить доступа к своему каталогу. Администратору необходимо предусмотреть блокирование учетных записей фиктивных пользователей, чтобы их правами никто не мог воспользоваться (например, для несанкционированного изменения прав доступа на каталоги).*

*Пока фиктивных учетных записей еще нет, администратор создает групповые каталоги от своего имени:*

***cd /home***

***mkdir –m 070 alfa***

***mkdir –m 070 beta***

***mkdir –m 070 gamma***

*Таким образом, полные права доступа на групповые каталоги имеют только члены соответствующих (одноименных) групп.*

*Затем создаются учетные записи пользователей с явным указанием их основных групп и групповых каталогов. Включение пользователей в дополнительные группы не планируется:*

***useradd –g alfa –m –d /home/alfa ivanov; passwd ivanov***

```
useradd -g alfa -m -d /home/alfa petrov; passwd petrov
useradd -g alfa -m -d /home/alfa starkov; passwd starkov
.........................................
useradd -g beta -m -d /home/beta sidorov; passwd sidorov
.........................................
useradd -g gamma -m -d /home/gamma smirnov; passwd smirnov
.........................................
```

*После этого надлежит передать владение каталогами фиктивным пользователям* **starkov**, **sidorov** *и* **smirnov**.

```
cd /home
chown starkov alfa
chown sidorov beta
chown smirnov gamma
```

*Любой из пользователей может создавать свои файлы с правами доступа 660 (чтение и запись для себя и для членов своей группы). Единую маску доступа* **umask=007** *администратор должен заранее, до создания учетных записей пользователей, предусмотреть в файле* **/etc/skel/.bash _profile**.

*Защиту групповых файлов от намеренной или случайной модификации и удаления со стороны членов группы с помощью базовых прав предусмотреть нельзя. Установка на файлы дополнительных атрибутов* **i** *и* **a** *лишена смысла, поскольку документы должны редактироваться и изменяться.*

6. Пользователь **semaforkin** входит в группу **beta** и обязан предоставить ее членам возможность читать и копировать его служебные документы. Ему также разрешено в пределах дисковой квоты создавать и хранить в своем каталоге личные файлы, и он вправе сделать их недоступными для других. В коллективе также работает пользователь **mironova**, которая членом вышеуказанной группы не является и которой **semaforkin** доверяет свои личные тайны, но не должен доверять служебную информацию.

***Решение.*** *Допустим, что служебные и личные файлы пользователя* **semaforkin** *будут храниться в его домашнем каталоге, в отдельных подкаталогах, не вложенных друг в друга. Например, подкаталог* **/home/semaforkin/doc** *хранит служебные файлы. Права на этот*

*каталог* **rwxr-x---** *и на служебные файлы* **rw-r-----** *обеспечивают требуемый доступ членам его группы и запрещают допуск посторонних. Если* **semaforkin** *разместит личные файлы в подкаталоге* **/home/semaforkin/private** *с установкой прав* **rwx---r-x** *на каталог и* **rw----r--** *на файлы, доступ пользователя* **mironova** *к файлам будет разрешен, а членам его группы – запрещен. Но, вместе с тем, открывается доступ к личным файлам со стороны всех иных зарегистрированных пользователей, не входящих в группу* **beta**. *Как видно, эта внешне простая задача одним только назначением прав доступа к файловым объектам не решается.*

*Один из вариантов решения задачи связан с дополнительными группами и сменой владельца каталога. Допустим, что пользователь* **mironova** *в единственном числе входит в собственную группу с таким же именем. Администратор при помощи команды*

**usermod –G mironova semaforkin**

*превращает пользователя* **semaforkin** *в члена этой группы. Второй командой*

**chown mironova /home/semaforkin/private**

*администратор передает* **mironova** *право владения каталогом. Наконец, пользователь* **mironova** *со своими полномочиями устанавливает права на доступ к этому каталогу в виде* **r-xrwx---**. *Задача заметно усложнится, если в основную группу* **mironova** *включены какие-либо иные пользователи. Вхождение пользователя* **semaforkin** *в группу* **mironova** *делает возможным его встречный доступ к ее файлам, а об этом в условиях задачи ничего не говорилось.*

*Второй вариант решения заключается в передаче личных файлов* **semaforkin** *во владение* **mironova**, *что можно сделать только с правами суперпользователя. При этом на эти файлы следует установить права* **r----rw-**, *а каталог* **/home/semaforkin/private** *по-прежнему будет доступен всем иным пользователям. Но передать во владение другому пользователю можно только уже существующие файлы, а что касается новых файлов, то они будут принадлежать создателю. Причем к этим файлам будет открыт доступ и иным пользователям, не входящим в группу* **beta**.

*Таким образом, для решения подобных задач требуется манипуляция группами и правами на файловые объекты. Предложенные варианты не отличаются строгостью решений.*

7. Сотрудник отдела кадров **mogolev** по вопросам службы подчиняется начальнику отдела и включен в основную группу **persona**, которая

связана с персональными данными сотрудников фирмы. В то же время он является ответственным членом профсоюзной организации, ведет ее документы и является членом дополнительной группы **trade**. Одновременно он является секретарем ячейки либерально-демократической партии и входит в дополнительную группу **liberty**. Разработка и редактирование служебных, профсоюзных и партийных документов производится на одном сетевом компьютере под управлением ОС Linux. Директор фирмы вынужден разрешить такое совмещение, но требует, чтобы утечки служебной информации по партийным и профсоюзным каналам не происходило. Этого же требуют и руководители общественных организаций. Члены групп пользователя должны иметь право входа только в определенные каталоги и чтения только предназначенных им файлов.

***Решение.*** *Допустим, что в домашнем каталоге пользователя* **mogolev** *создаются три отдельных подкаталога с именами* **persona**, **trade** *и* **liberty** *и файлами соответственно служебного, профсоюзного и партийного назначения.*

*По определению члены дополнительных групп пользователя по отношению к его файловым объектам имеют права всех остальных пользователей, иначе говоря, права членов дополнительных групп не разграничиваются. Групповые права на каталоги и файлы пользователя распространяются только на членов его основной группы. Следовательно, обычным способом задача разграничения доступа не решается.*

*Один из вариантов решения заключается в передаче подкаталогов* **persona**, **trade** *и* **liberty** *во владение каким-либо членам соответствующих групп (желательно – фиктивным). Тогда на каждый из подкаталогов можно установить права* ***r-xrwx---*** *, обеспечивая тем самым пользователю* **mogolev** *и членам функциональных групп возможность свободной манипуляции с файлами.*

8. За единственным в организации компьютером, категорированным для обработки и хранения информации ограниченного доступа и работающим под управлением операционной системы Linux, закреплено три пользователя: **ivanov**, **petrov** и **kondakov**. Пользователь **ivanov** допущен к обработке сведений, содержащих служебную тайну. Пользователь **petrov** работает с персональными данными, а **kondakov** – с коммерческой тайной. Пользователи работают за компьютером по временному графику, и одновременное присутствие за консолью более одного пользователя исключается. В случае временного отсутствия пользователя его виртуальная консоль блокируется. У каждого пользователя имеется свой одноименный домашний каталог, куда доступ других пользователей должен быть запрещен. Доступ к данным в обход файловой системы исключен с помощью организационных мер и путем бло-

кирования возможности загрузки компьютера со сменного машинного носителя.

***Решение.*** *Рассмотрим решение на примере пользователя* **ivanov**. *В его владении находится каталог* **/home/ivanov** *с установленными правами* **rwx------**. *Групповые права на каталог исключены, так как вышеназванные пользователи на этапе создания учетных записей по умолчанию могут быть включены в одну группу. Права на каталог обеспечивают решение задачи, и других механизмов защиты не требуется.*

9. В организации работают два пользователя: sergeev и **nikolaev**. **Sergeev** допущен к секретным сведениям, а **nikolaev** должен работать только с несекретными документами. Каждый из пользователей работает с фиксированным числом электронных документов (новые файлы не создаются, а существующие не удаляются). Файлы обоих пользователей хранятся в одном каталоге. **Sergeev** имеет право читать и редактировать свои файлы, а также читать файлы «несекретного» пользователя без возможности записи в них. **Nikolaev** может читать и редактировать свои файлы, а также имеет право дописывать (без возможности чтения и удаления существующих данных) свою информацию в файлы «секретного» пользователя. Группы пользователей включают только их самих. Ни один из пользователей не является администратором и не наделен особыми правами. Требуется составить матрицу доступа пользователей и обеспечить их разграничение средствами файловой системы Linux (табл. 1.1).

Таблица 1.1

| **Пользователь** | **Права доступа к объектам ФС** | |
|---|---|---|
| | **Секретные файлы** | **Несекретные файлы** |
| **Sergeev** | **rw** | **r** |
| **Nikolaev** | **a** | **rw** |

10. В дополнение к предыдущей задаче оба пользователя должны иметь возможность создавать новые файлы и удалять файлы, которые им принадлежат. Удаление чужих файлов, в том числе путем их полной перезаписи, запрещается. Требуется составить матрицу доступа пользователей и обеспечить их разграничение средствами файловой системы Linux (табл. 1.2). В целях разграничения доступа можно предусмотреть создание личных каталогов пользователей. Выполняются ли условия разграничения доступа, если файлы пользователей хранятся в одном каталоге?

Таблица 1.2

| Пользователь | Права доступа к объектам ФС | |
|---|---|---|
| | Секретные файлы | Несекретные файлы |
| **Sergeev** | **crwd** | **rt** |
| **Nikolaev** | **at** | **crwd** |

*Символ* ***c*** *обозначает право создания файлов,* ***d*** *– право удаления своих файлов, а* ***t*** *– запрет на удаление чужих файлов. Подобные права устанавливаются не на файлы, а на каталоги.*

*Попробуем решить задачу с одним общим каталогом для секретных и несекретных файлов, который никому из них не принадлежит. В одну группу пользователи не входят, следовательно, их право на каталог такое же, как и для всех остальных зарегистрированных пользователей. Для создания файлов каждому из них требуется право на вход и запись в каталог. Для удаления существующих файлов потребуется знать их имена, что предполагает право чтения в каталоге. В дополнение к полным базовым правам на каталог требуется установить дополнительный стикки–бит, чтобы помешать пользователям в удалении чужих файлов.*

*Таким образом, каталог создается администратором и принадлежит ему. Права на каталог задаются битовой строкой* ***------rwt*** *(администратору никаких прав не требуется). Права на файлы* **Sergeev** *записываются в виде* ***rw-----w-****, а права доступа к файлам* **Nikolaev** *представляются в виде* ***rw----r--****.*

11. В организации работают три пользователя: «секретный» – **smirnov**, «конфиденциальный» – **kostrov** и «несекретный» – **nikonov**. Ни один из пользователей не является администратором и не наделен особыми правами. Пользователи создают и редактируют документы соответствующих уровней конфиденциальности. Они имеют право удалять принадлежащие им файлы. Пользователи не имеют права создавать файлы, доступные для чтения «снизу», и записывать данные в существующие файлы более низкого уровня конфиденциальности. **Kostrov** и **nikonov** имеют право дописывать свои данные в файлы на один уровень выше, но не имеют прав на чтение информации более «высокой» категории. Требуется изобразить матрицу доступа (табл. 1.3) и предложить исчерпывающие меры по его разграничению.

Таблица 1.3

| Пользователь | Права доступа к объектам ФС | | |
|---|---|---|---|
| | Секретные файлы | Конфиденциальные файлы | Несекретные файлы |
| smirnov | crwd | rd | rd |
| kostrov | w | crw | rd |
| nikonov | - | w | rw |

12. В некоторой организации на компьютерах под управлением операционной системы Linux хранится и обрабатывается информация нескольких уровней конфиденциальности. Три пользователя (учетные имена можно задавать произвольно) имеют доступ к секретной информации. Четыре пользователя допущены к коммерческой тайне. Пять пользователей работают только с открытой информацией. Пользователи, имеющие допуск к информации одного уровня конфиденциальности, имеют право читать информацию из файлов своих коллег, а также из файлов более низкого уровня конфиденциальности, однако записывать информацию в файлы низшего уровня они не вправе.

13. Задание усложняется тем, что в каталогах, принадлежащих секретоносителям и обладателям коммерческой тайны, необходимо предусмотреть файлы, в которые разрешена запись (точнее – дописывание) «снизу». Возможность чтения «снизу» этих и иных файлов должна быть исключена. Файлы не являются программами, и право их исполнения исключается. Требуется изобразить матрицу доступа (табл. 1.4) и предложить меры по его разграничению.

Таблица 1.4

| Пользователь | Права доступа к объектам ФС | | |
|---|---|---|---|
| | Секретные файлы | Конфиденциальные файлы | Несекретные файлы |
| smirnov | crwd | rd | rd |
| kostrov | w | crw | rd |
| nikonov | - | w | rw |

## 1.6. Использование механизма SUDO

В ОС Linux пользователи бесправны, и любое ответственное действие требует полномочий администратора. Администратор по определению должен быть в системе один (операционная система не позволяет обычным путем создать еще одну учетную запись с **UID** = 0), а обязанностей у него чрезмерно много. Достаточно часто у администратора возникает потребность доверить избранным пользователям выполнение каких-либо ответственных операций. Но для этого администратор должен доверить другим святая святых – свой пароль. Чтобы избежать таких ситуаций, в ОС UNIX был разработан своеобразный механизм, позволяющий выборочно предоставлять определенным пользователям права на запуск ответственных команд. Этот механизм реализован в утилите **sudo** (**s**uper**u**ser **do**) и конфигурационном файле **/etc/sudoers**, который должен быть недоступным для всех, кроме администратора.

В качестве параметра команды **sudo** вводится команда, которую надо исполнить. Обычно, если это явно не определено в файле конфигурации, от пользователя потребуют ввести его собственный пароль, но только один раз в течение терминального сеанса, в последующем запрос пароля производиться не будет. При этом правильно написанная команда будет исполнена оболочкой с правами привилегированного пользователя (администратора).

Названия команд составляются администратором и записываются в файл **/etc/sudoers** в виде специальных строк. Строка должна связывать конкретного пользователя, работающего на определенном сетевом узле или локальном компьютере, с ответственными командами. Запись в строке должна содержать ответы на вопросы:

- **кто**? (зарегистрированное имя пользователя или группы);
- **где**? (доменное имя или IP-адрес компьютера);
- **от чьего имени**? (имя привилегированного пользователя);
- **команды** (полный перечень разрешенных команд, разделенных запятыми).

Приведем образцы записей в файле **/etc/sudoers** :

```
petrov ALL=/bin/shutdown, /bin/halt
```

– пользователю **petrov** на всех компьютерах, работающих под управлением данной операционной системы, можно принудительно завершать работу системы и всех пользователей;

**john localhost=/usr/sbin/lpc, /usr/sbin/lprm**

– пользователь **john** на своем компьютере имеет право распечатки документов;

**braun 192.168.1.1=/bin/useradd,/bin/passwd,/bin/usermod**

– пользователь **braun** на компьютере, который идентифицируется IP-адресом, имеет право создавать и модифицировать учетные записи пользователей, а также назначать им первичные пароли. Здесь допущен очевидный перебор в предоставлении прав, поскольку пользователь **braun** может сменить пароль самому администратору;

**%alfa ALL=/bin/mount /dev/cdrom, umount /dev/cdrom**

– пользователям группы **alfa** на всех компьютерах сети разрешено монтировать и размонтировать компакт-диски. Кстати, аналогичное право пользователям можно предоставить с помощью записи в конфигурационном файле **/etc/fstab**.

В файле **/etc/sudoers** могут быть определены псевдонимы (алиасы):

- для пользователей:

**User_Alias WEBMASTERS=sidorov,sazonov,bormotov**

- для сетевых узлов:

**Host_Alias WEBSERVERS=www.web_dis.com, 192.168.3.14**

- для команд:

**Cmnd_Alias APACHE=/usr/local/apache/bin/apachectl**

С помощью псевдонимов разрешающая запись приобретает более компактный вид:

**WEBMASTERS WEBSERVERS=APACHE**

Пользователь, наделенный временными полномочиями, должен знать, какую команду ему разрешено выполнять с правами администратора и как эта команда пишется. Команды, перечисленные в файле **/etc/sudoers**, должны приводиться с полными путевыми именами, чтобы пользователи были лишены возможности запускать свои экземпляры утилит или сценариев от имени администратора.

Механизм избирательного предоставления полномочий предусматри-

вает исполнение команд без ввода паролей. Для этого администратор в соответствующей строке между именем хоста и именем команды должен указать ключевое слово **NOPASSWD**:

```
ivanov 192.168.4.15=NOPASSWD: /bin/mount /dev/fd0
```

Вызов утилиты **sudo** по умолчанию протоколируется системой в одном из журналов аудита. Выяснить, в каком именно журнале, можно с помощью команды **lsof**, предоставляющей информацию об открытых файлах.

```
lsof –c sudo
```

Более подробно о возможностях команды **lsof** будет сказано ниже.

# 2. БЕЗОПАСНОЕ УПРАВЛЕНИЕ ПРОЦЕССАМИ

Операционные системы на базе ядра Linux являются многозадачными системами общего назначения, и управление процессами в них напрямую связано с обеспечением безопасности обрабатываемой компьютерной информации. Так, угрозами являются несанкционированный запуск опасного процесса, необоснованный и сверхнормативный захват процессорного времени, доступ пользовательского процесса в чужое адресное пространство и многое другое.

Процессом называется компьютерная программа на этапе исполнения. Но процесс – это не только действие, но и ресурсы, которыми операционная система наделяет программу. К числу этих ресурсов в первую очередь относится адресуемая виртуальная память, в которую загружаются программный код, данные (константы и переменные), библиотечные функции, стек и вспомогательная информация. Сложная программа может создать несколько одновременно выполняемых процессов. Повторяющийся запуск на исполнение одной и той же программы также создает множество независимых процессов. Поэтому правильнее назвать процессом выполняющийся экземпляр программы.

## 2.1. Общие сведения о процессах

По отношению к пользователю процесс может быть интерактивным и фоновым. Процесс может запускаться из ядра операционной системы либо из программного файла. В ОС Linux существует три вида процессов, отличающихся привилегиями, режимом исполнения и наличием порождающих объектов в файловой системе.

**Системные процессы**. Они не имеют собственных исполняемых файлов и запускаются из ядра операционной системы. Минимальный комплект системных процессов должен обеспечить функционирование виртуальной памяти, многозадачности и файловой системы. Системные процессы стартуют при загрузке операционной системы и выполняются в течение всего времени ее работы. Системные процессы являются частью ядра и всегда расположены в оперативной памяти. Системные процессы не имеют соответствующих им программ в виде исполняемых файлов и запускаются особым образом при инициализации ядра системы. Выполняемые инструкции и данные этих процессов находятся в ядре системы, таким образом, они могут вызывать функции и обращаться к данным, недоступным для остальных процессов. Эти процессы не общаются с пользователем, и он может узнать об их существовании, только просматривая список процессов. Рассматриваемая ниже команда **`ps -ef`**, начиная с версии 2.4 ядра

Linux, показывает процессы ядра в квадратных скобках (ранее имена «ядерных» процессов отображались в круглых скобках).

Есть один процесс, который по своей сути является системным, но запускается не из ядра, а из отдельного исполняемого файла с именем **/sbin/init**. Этот процесс является прародителем всех остальных процессов. Идентификатор процесса (**PID**) у него равен единице, несмотря на то, что до него загружаются процессы ядра. Принудительное завершение этого процесса командой **kill -9 1** , по сути, означает аварийный останов системы, но ОС Linux не позволяет сделать это даже администратору.

**Сервисные службы или демоны**. Даже если компьютер не выполняет функции сервера, система нуждается в существовании множества процессов для «самообслуживания»: направления заданий на печать и обеспечения их очереди, запуска заданий по расписанию, проверки целостности файловой системы и т. д. Набор утилит и системных программ, предназначенных для предоставления таких услуг, принято называть службами (сервисами) или демонами (от английского daemon). Последнее название не связано с какими-либо злыми мифическими духами и представляет собой акроним от Disk And Execution MONitor. Это процессы, которые выполняются в фоновом режиме, не связаны с конкретными терминалами, не общаются с обычными пользователями напрямую и не могут управляться ими.

Обычно демон активизируется в процессе загрузки системы после инициализации ядра, по инициативе администратора, по запросу пользовательской программы, по сетевому запросу или по наступлению какого-либо системного события. Каждому демону соответствует исполняемый файл. Идентифицировать большинство демонов можно по завершающему имя файла символу **d**: **syslogd, inetd, telnetd** и др.

Большинство демонов запускаются при старте операционной системы, после системных процессов. Их запуск происходит при участии процесса **init** из загрузочных сценариев (порядок загрузки системы изложен ниже). Администратор имеет право «вручную» запускать сервисы с помощью команды **start** и завершать их командой **stop**, но наличие данных команд зависит от используемого дистрибутива ОС Linux. Впрочем, по отношению к демонам можно использовать и обычную форму запуска в виде имени файла в командной строке. Завершить сервис, как и обычный процесс, можно направлением ему сигнала на завершение работы при помощи команды **kill <PID>** (см. ниже).

**Пользовательские процессы**. Запускаются из исполняемого (бинарного) файла пользователем. Они могут выполняться в интерактивном или фоновом режиме, и срок их выполнения обычно ограничен продолжитель-

ностью сеанса работы пользователя. Впрочем, как будет показано ниже, фоновые процессы продолжают существовать и после завершения пользовательского сеанса. Пользовательские процессы наследуют идентификатор пользователя и, как правило, имеют соответствующие права на доступ к объектам. Но некоторые пользовательские процессы могут при выполнении иметь больше прав, чем имеет создавший их пользователь.

Самый важный пользовательский процесс – командный интерпретатор (оболочка или шелл), обеспечивающий диалоговый режим работы с пользователем. К моменту, когда система предоставит пользователю право управлять собой, в ней уже будет существовать несколько десятков системных и сервисных процессов.

Система изолирует пользовательские процессы друг от друга, от системных процессов и демонов. Каждый процесс выполняется в своем виртуальном адресном пространстве и других процессов «не видит». В многозадачной операционной системе пользовательские процессы не имеют права читать данные чужого процесса, записывать свои данные в его адресное пространство, отбирать у других задач вычислительное время и право доступа к устройствам ввода-вывода.

Выполнение процессов может происходить в одном из двух возможных режимах: в режиме ядра либо в пользовательском режиме. В режиме ядра процесс может выполнять любые привилегированные инструкции по управлению центральным процессором. В пользовательском режиме выполняются обычные, непривилегированные инструкции. Любая программа, созданная для многозадачной операционной системы, использует вызовы системных функций. Большинство таких функций (обычные вычислительные процедуры, операции со строками и др.) не требуют каких-либо исключительных привилегий и могут выполняться в пользовательском режиме. Но создание, модификация или удаление объектов файловой системы, монтирование дисковых устройств, создание учетных записей и многое другое требуют системных привилегий.

Управление процессами осуществляется частью ядра, именуемой планировщиком задач или просто планировщиком (scheduler). В ОС Linux реализована принудительная или, по иному, вытесняющая многозадачность, когда планировщик задач в замкнутом цикле передает управление следующему процессу, не «спрашивая» согласия на это у предыдущего процесса. Для того чтобы переключить центральный процессор с одной задачи на другую, также требуется время. Планировщик задач обычно расходует 5-7 % процессорного времени на то, чтобы сохранить в памяти содержимое предыдущей задачи, загрузить в регистры центрального процессора адреса следующей задачи и передать ей управление. Каждый процесс выполняется в течение нескольких миллисекунд, и у пользователя создается

представление, что компьютер способен выполнять все задачи одновременно.

Процесс является носителем определенного приоритета, **nice number**, который принимается во внимание планировщиком задач при выборе очередности и продолжительности запуска. Приоритет процесса – это его право распоряжаться временем центрального процессора. Этот приоритет никак не связан с правами доступа к файлам. Некоторые системные процессы должны иметь большой приоритет, но если они будут владеть большей частью процессорного времени, у компьютера не останется возможностей для выполнения пользовательских задач, ради которых он, собственно, и работает. Поэтому системные процессы, как правило, не злоупотребляют своими полномочиями. Процессы, созданные в одинаковых условиях обычным пользователем и администратором системы, имеют примерно равные приоритеты. Избыточный приоритет – весьма опасная вещь, способная приводить к атакам на отказ в обслуживании.

Абсолютный приоритет – это число, находящееся в диапазоне от –20 (наивысший приоритет) до 19 (наименьший). Обычным образом запускаемая программа имеет нулевой приоритет. Именно такое значение будет выведено, если ввести команду **nice** без аргументов. Положительное значение этого фактора свидетельствует о понижении приоритета, и наоборот. Команда

```
nice -n <command>
```

позволяет задать приоритет, с которым будет выполняться процесс после запуска.

Утилита **renice** позволяет изменить приоритет (точнее – его относительное значение, которое называют фактором уступчивости) уже запущенного процесса. Привилегия суперпользователя позволяет запускать процессы с отрицательным фактором уступчивости или понижать это значение у выполняющегося процесса. Обычные пользователи и их процессы не имеют прав захватывать лишнее время у центрального процессора. Это означает, что пользователь может только замедлять свой процесс. Так, команда

```
renice +10 <PID>
```

позволит пользователю замедлить *свой* процесс на 10 пунктов (сколько миллисекунд потеряет при этом квант процессорного времени, выделенный этому процессу, зависит и от аппаратной платформы, и от загрузки системы). Но вернуть приоритет замедленного процесса в прежнее значение пользователь уже не сможет – ему запрещено указывать отрицательные значения фактора уступчивости.

С помощью утилиты **snice** можно изменить приоритет всем процессам, созданным определенным пользователем:

```
snice +5 ivanov
```

Существует категория так называемых «жадных» программ, которые способны захватывать ресурсы компьютерной системы в ущерб другим процессам. Например, программа, имеющая право повышать свой приоритет, может захватить все время центрального процессора и уйти в вечный цикл. Но пользовательским процессам такие проделки запрещены. В то же время «жадная» программа может затребовать непомерно большой объем виртуальной памяти и вызвать ее дефицит. Такой жадности способствует непонятная расточительность универсальных операционных систем, и Linux здесь не является исключением. Воспользовавшись внутренней командой оболочки **ulimit –a** , можно отобразить действующие в системе ограничения по процессорному времени (cpu time), виртуальной памяти (virtual memory) и размеру создаваемых файлов (file size). На рис. 2.1 можно видеть, что по умолчанию эти и некоторые иные ресурсы ничем не ограничены (unlimited).

```
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
file size               (blocks, -f) unlimited
max locked memory       (kbytes, -l) unlimited
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 447
virtual memory          (kbytes, -v) unlimited
```

Рис. 2.1. Информация о ресурсах, выводимая командой **ulimit –a**

Аналогичная информация доступна для просмотра в виртуальных файлах **limits**, которые располагаются в нумерованных директориях, выделенных для каждого процесса в каталоге **/proc** (параграф о содержимом этого каталога следует ниже). В файле, имеющем вид таблицы, для каждого параметра задаются «жесткие» и «мягкие» лимиты. «Жесткие» лимиты задаются для всех пользователей, включая администратора. В рамках «жестких» пределов каждый пользователь может задавать для своих процессов «мягкие» ограничения, которые не должны выходить за обозначенные общими правилами границы.

Неограниченные права любого процесса позволяют обычному пользователю произвести атаку на захват ресурсов компьютерной системы. Так, команда

```
cat /dev/zero > /tmp/abcd
```

позволяет создать в доступном общем каталоге файл неограниченных размеров.

Наличие в командном файле бесконечного цикла

```
while 1
mkdir 1
cd 1
touch 2
end
```

не только создает каталог бесконечной «глубины», но и путем создания множества пустых файлов истощает доступный ресурс индексных дескрипторов.

Десяток расточительных пользовательских процессов, порожденных командами типа

```
yes 12345 > /dev/null &
```

существенно замедлят выполнение полезных программ и сервисов.

Порождение процессов в цикле (возможное число процессов, которые может создать пользователь, хоть и не бесконечно, но довольно велико) может на некоторое время заблокировать администратору отображение самой информации о процессах, поскольку утилита **`ps –ef`** будет непрерывно обновлять свои данные.

К счастью, с помощью вышеназванной команды **`ulimit`** есть возможность урезать аппетиты пользовательских программ как в отношении дискового пространства, так и в отношении количества создаваемых процессов. Лимит на максимальный размер файлов устанавливается командой

```
ulimit –f 100
```

после чего создание файла размером более 100 блоков (1 блок = 1024 байта) станет невозможным. Ограничитель на число открываемых процессом файлов задается указанием значения при параметре **`-n`**. Для того чтобы ограничить максимальное число процессов, запущенных одним пользователем со всех терминалов, следует исполнить команду

```
ulimit -u 10
```

где величина **`-u`** указывает максимальное число процессов.

Нетрудно убедиться, что эти ограничения действуют только в теку-

щем сеансе и только на экземпляр оболочки, обслуживающей конкретного пользователя в конкретной консоли. Для установления «жестких» ограничений на все сеансы и процессы, запущенные любым пользователем, администратор должен записать строки

```
ulimit -u 10
ulimit -n 50
ulimit -f 1000
```

в файл **/etc/profile**. Ограничения начнут действовать сразу после нового входа пользователя в систему, т. е. когда процесс **login** создаст сеанс пользователя и запустит процесс **bash** в рамках этого сеанса, а процесс **bash** считает все установочные значения из всех своих имеющихся конфигурационных файлов. Пользователям (кроме администратора) не удастся превысить или переустановить эти пределы.

В операционной системе по умолчанию запускается довольно много системных процессов и сервисов. Многие из них напрасно расходуют процессорное время и оккупируют оперативную память. Некоторые сервисы откровенно небезопасны по причине своей уязвимости. Специалисты рекомендуют администраторам избавляться от подобного «багажа». Вот краткий список сервисов сомнительной полезности:

- **portmap, rpc.mountd, rpc.nfsd** – службы обеспечивают функционирование сетевой файловой системы NFS;
- **nmbd, smbd** – службы реализуют аналог с сетевыми ресурсами ОС Windows*;
- **named** – обеспечение службы доменных имен;
- **telnet, rlogin, rexec** – небезопасные службы для сетевого управления компьютером;
- **finger, comsat, chargen, identd, echo** – набор устаревших и небезопасных служб.

Все сущности в операционной системе имеют свои номера. Не являются исключением и процессы. Каждому из них система назначает уникальный 16-разрядный идентификатор (process identifier – **PID**). Таким образом, в системе может быть одновременно запущено большое число – 65536 процессов. Идентификационные номера присваиваются процессам по порядку, по мере их создания. Обычно первая сотня номеров присваивается системным процессам и демонам, поскольку они запускаются первыми. При завершении процесса система освобождает его идентификатор, но современные версии ОС в одном сеансе освобожденные номера повторно используют только после исчерпания доступного диапазона.

Процесс, запущенный другим процессом, называется дочерним (child) процессом или потомком. Соответственно созидающий процесс на-

зывается родительским (parent), родителем или просто – предком. Поэтому у каждого процесса наряду с идентификатором **PID** есть еще один числовой атрибут – **PPID** (Parent Process ID) – идентификатор родительского процесса.

Запускающий процесс создаёт своего «двойника», вызывая системную функцию **fork**: двойниками они являются по причине идентичности контекстов процессов, за исключением **PID** и **PPID**. Выполняя этот вызов, система создает дочерний процесс, являющийся почти полной копией родительского, но имеющий свой идентификатор **PID** и выполняющийся в собственном адресном пространстве. При этом родительский процесс ожидает завершения работы порождённого процесса. Для завершения работы процесса выполняется системный вызов **exit**. Полную самостоятельность порожденный процесс получает после выполнения системного вызова **exec**. Дочерний процесс сохраняет значение идентификатора родительского процесса **PPID**. «Родственники» могут взаимодействовать друг с другом посредством сигналов, функций межпроцессного взаимодействия, каналов и др.

Пользовательским процессам присваивается еще один идентификатор – **UID** – уникальный номер пользователя, который их запустил. Пользователь не может манипулировать логическими объектами непосредственно, и это делает за него программа. Обычные пользовательские программы имеют право делать только то, что разрешено их владельцу. Например, пользователь, запустивший процесс

```
ls –la /root
```

в целях просмотра содержимого подкаталога администратора, получит сообщение **permission denied**. На самом деле отказ в доступе получил процесс, запущенный от имени пользователя. Но такой же процесс, запущенный с правами обладателя нулевого **UID**, будет выполнен успешно.

Процессы могут иметь так называемые эффективные идентификаторы пользователя **EUID** и группы **EGID**. Такие процессы порождаются исполняемыми файлами с установленным битом **SUID** или **SGID**. Эффективный идентификатор позволяет процессу выполнять действия не от имени пользователя, запустившего процесс, а от имени владельца файла. Данный механизм необходим для выполнения некоторых задач, например пользователю в ходе работы разрешается сменить собственный пароль, если минимальный срок действия старого пароля еще не истек. Если лишить пользователей таких прав, нагрузка на администратора возрастет многократно.

Смена пароля сопровождается вычислением его хэш-образа, который должен быть записан в недосягаемый и невидимый для пользователя файл

**/etc/shadow**. Возникшее противоречие решается так: программа **passwd**, вычисляющая хэш-функцию введенного пароля и записывающая результат в теневой файл, выполняется не от имени пользователя, а от имени владельца утилиты – администратора. Аналогичными временными правами обладают процессы, порожденные при запуске таких утилит, как **su**, **mount** и некоторые другие.

## 2.2. Средства наблюдения за процессами

Для наблюдения за активными процессами пользователь должен ввести команду **ps** (process status). Команда **ps**, введенная без аргументов, показывает все пользовательские процессы, которые были запущены в течение текущей сессии. Утилита **ps** отличается большим количеством аргументов, в том числе дублирующих друг друга. Утилита **ps** в ОС Linux поддерживает опции в трёх разных стилях: UNIX (могут быть сгруппированы и должны иметь перед собой дефис), BSD (могут быть сгруппированы и не нуждаются в дефисе) и GNU с длинными опциями (опции записываются словом и должны иметь перед собой два дефиса). Хотя буквы в первых двух стилях могут использоваться одинаковые, выполнять они могут разные функции. Утилита читает и выводит информацию о процессах из виртуальной файловой системы **/proc**.

На рис. 2.2 изображен выборочный список процессов, полученный с помощью команды **ps –ef**. Опция **–e** отображает все процессы, **-f** – означает подробный вывод данных.

```
UID          PID  PPID  C STIME TTY          TIME CMD
root           1     0  0 20:48 ?        00:00:01 init [3]
root           2     0  0 20:48 ?        00:00:00 [kthreadd]
root           5     2  0 20:48 ?        00:00:00 [watchdog/0]
root           9     2  0 20:48 ?        00:00:00 [events/0]
root        1370     2  0 20:48 ?        00:00:00 [kjournald]
root        3060     1  0 20:49 ?        00:00:00 /usr/sbin/syslogd
root        3190     1  0 20:49 ?        00:00:00 /usr/sbin/sshd
root        4439     1  0 20:59 tty1     00:00:00 -bash
root        5672  5671  0 21:22 pts/1    00:00:00 -bash
root        5771  5770  0 21:23 pts/3    00:00:00 -bash
root        5804  5672  0 21:24 pts/1    00:00:03 top
ivanov      6188     1 92 21:32 ?        00:34:19 cat /dev/zero
root        7392  7391  0 21:58 pts/4    00:00:00 -bash
root        7550  7392 64 22:01 pts/4    00:05:00 /root/no 12345
root        7578  7392 53 22:02 pts/4    00:03:48 yes 12345
petrov      7768     1 49 22:06 ?        00:01:40 od /dev/zero
root        7924  7392  0 22:09 pts/4    00:00:00 ps -ef
```

Рис. 2.2. Фрагмент таблицы процессов, выводимой командой **ps -ef**

Подробная таблица процессов может содержать следующий набор параметров:

- **PID** – идентификатор процесса;
- **PPID** – идентификатор «родительского» процесса;
- **TTY** – терминал, из которого был запущен пользовательский процесс;
- **UID** – идентификатор пользователя, запустившего процесс;
- **CMD** – команда, которая была введена для запуска процесса;
- **PRI** – текущий динамический приоритет процесса;
- **NI** – относительный приоритет процесса;
- **TIME** – суммарное время выполнения процесса;
- **STAT** – состояние процесса.

Столбец **STAT** (иногда он обозначается одним символом **S**) может содержать следующие значения:

- **R** (Runnable) – процесс выполняется или готов к выполнению (состояние готовности);
- **D** (Delay) – процесс пребывает в состоянии задержки, например, в ожидании дисковой операции;
- **T** (Traced or Stopped) – процесс остановлен или трассируется отладчиком;
- **S** (Sleeping) – процесс ожидает наступления какого-либо события. Такое состояние характерно для демонов;
- **Z** (Zombied) – процесс-зомби;
- **<** – процесс с отрицательным значением фактора уступчивости **nice**;
- **N** – процесс с положительным значением **nice**.

Вопросительный знак в столбце имени терминала **TTY** указывает на то, что процесс был запущен еще при загрузке системы либо это фоновый процесс, который продолжает выполняться после завершения пользовательского сеанса.

Можно выборочно вывести параметры, которые интересуют наблюдателя. Делается это с помощью команды

```
ps –e –o uid,ppid,pid,tty,time,stat,cmd
```

После опции **–o** в нужной последовательности указываются требуемые столбцы таблицы.

Наблюдение за процессами в динамическом режиме можно организовать с использованием утилиты **top** (рис. 2.3).

```
top - 22:13:40 up  1:25,  5 users,  load average: 3.99, 3.59, 2.38
Tasks: 124 total,   5 running, 119 sleeping,   0 stopped,   0 zombie
Cpu(s): 22.0%us, 16.3%sy,  0.0%ni, 61.1%id,  0.5%wa,  0.1%hi,  0.1%si,
0.0%st
Mem:   2030288k total,   463752k used,  1566536k free,    24892k buffers
Swap:  2168732k total,        0k used,  2168732k free,   250904k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 7550 petrov    20   0  1980  672  504 R   50  0.0   7:08.37 no
 7578 ivanov    20   0  1980  676  504 R   50  0.0   5:56.62 yes
 7768 root      20   0  2004  756  576 R   50  0.0   3:48.28 od
 6188 root      20   0  1996  676  504 R   48  0.0  36:27.97 cat
    1 root      20   0   772  300  260 S    0  0.0   0:01.44 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.00 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   0:00.06 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root      RT  -5     0    0    0 S    0  0.0   0:00.00 migration/1
    7 root      15  -5     0    0    0 S    0  0.0   0:00.08 ksoftirqd/1
    8 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/1
    9 root      15  -5     0    0    0 S    0  0.0   0:00.04 events/0
   10 root      15  -5     0    0    0 S    0  0.0   0:00.04 events/1
   11 root      15  -5     0    0    0 S    0  0.0   0:00.02 khelper
 3060 root      20   0  1748  616  524 S    0  0.0   0:00.00 syslogd
 3064 root      20   0  1704  420  352 S    0  0.0   0:00.00 klogd
 3190 root      20   0  3928  956  668 S    0  0.0   0:00.00 sshd
 3284 root      20   0  1912  640  540 S    0  0.0   0:00.00 crond
 3286 daemon    20   0  1912  416  316 S    0  0.0   0:00.00 atd
 3330 root      20   0  1920  488  412 S    0  0.0   0:00.08 gpm
 4589 root      20   0  2780 1364 1024 S    0  0.1   0:00.00 startx
 4605 root      20   0  2764  792  656 S    0  0.0   0:00.00 xinit
 4606 root      19  -1  337m  10m 3820 S    0  0.5   0:38.96 X
 4639 root      20   0  2760 1304  984 S    0  0.1   0:00.00 sh
 5672 root      20   0  3132 1752 1228 S    0  0.1   0:00.00 bash
  123 root      20   0  2340 1004  752 R    0  0.0   0:00.00 top
```

Рис. 2.3. Фрагмент таблицы процессов, выводимой командой **top**

Утилиту назвали **top** потому, что процессы, наиболее активно использующие процессорное время, будут находиться вверху (top – верх).

Любопытное наблюдение – при отсутствии «прожорливых» процессов сам **top** может возглавлять список приоритетных задач. Это вполне объяснимо – этой программе необходимо опрашивать состояние системы с периодичностью в несколько секунд, что ощутимо нагружает центральный процессор (периодичность – причина того, что в приведенном примере «снимок» зафиксировал процесс **top** в числе наименее затратных процессов).

В заголовке утилита **top** выводит обобщенную информацию о пользователях, запущенных процессах и их состоянии, расходовании процессорного времени, а также расходе памяти – оперативной и виртуальной.

Поля, выводимые программой **top**

- **PID** — идентификатор процесса;
- **USER** — имя пользователя – владельца процесса;
- **PRI** — приоритет процесса;
- **SIZE** — размер памяти процесса в килобайтах (Кб), включая области кода, данных и стека;
- **RSS** — общий объем памяти, выделенной процессу (Кб);
- **STAT** — текущее состояние процесса;
- **%CPU** — процентная доля процессорного времени, выделенного на процесс с момента предыдущего обновления;
- **%MEM** — процентная доля памяти, выделенной процессу с момента предыдущего обновления;
- **TIME** — общее процессорное время, израсходованное процессом с момента его запуска;
- **COMMAND** — имя исполняемого файла.

Утилита выводит информацию о наиболее активных процессах, а поскольку их активность непостоянна, то выводимая таблица обновляется по умолчанию каждые две секунды. Программа работает в интерактивном режиме и управляется с клавиатуры. Например, при обнаружении процесса, активность которого представляет опасность для системы, его можно сразу удалить. Для этого следует нажать клавишу с первым символом соответствующей команды **k – kill**, а после запросов программы поочередно указать идентификатор процесса и номер посылаемого сигнала. Опция **u** позволит указать имя пользователя или его **UID** и наблюдать только за процессами, владельцем которых является этот **UID**. Нажатие символа **h** позволит вывести справку по интерактивным командам.

Вывод информации в файл производится командой

```
top -n 1 -b > /tmp/top.out ,
```

где **b** – выводить информацию в текстовом формате;

**n** – количество повторов.

Можно упомянуть еще несколько полезных утилит, используемых для наблюдения за процессами. Часто интерес предсталяют не только сами процессы, но и связанные с ними субъекты и объекты. Для отслеживания подобных связей неоценимую помощь может оказать утилита **lsof** (list open files). Она отображает список всех открытых файлов, директорий, библиотек, сокетов и устройств, связанных с заданным процессом.

Например, администратор обратил внимание на подозрительный процесс и желает узнать, с какими файлами тот работает. Эти данные выводятся с помощью команды

**lsof -p 3245** ,

где после аргумента **-p** указывается **PID** контролируемого процесса. Нужный процесс можно указать и по имени:

**lsof -c crond**

Эта утилита позволяет задавать связанные с процессами файлы специальных устройств. Так, с помощью команды

**lsof /dev/tty3**

можно установить, чем занят пользователь, работающий за третьим терминалом.

Список процессов, связанных с определенным пользователем, можно вывести, воспользовавшись командой

**lsof -u petrov**

И наконец, эта мощная утилита с помощью логических связей позволяет фильтровать условия. Так, команда

**lsof -u ivanov -ac bash**

покажет все файловые связи, открытые командным интерпретатором, обслуживающим пользователя **ivanov**. Символ **a** (and) указывается перед каждым аргументом, который следует связать логической функцией «И».

С помощью монитора процессов можно понаблюдать за изменением прав пользователя, который с этой целью применяет команду **su**. Изменение прав сопровождается выделением пользователю нового экземпляра командной оболочки. Но родительский процесс не завершается и ожидает окончания работы порожденного дочернего процесса. Если сеанс-потомок будет прекращен, управление будет возвращено родительскому процессу.

Так, с помощью команды **su petrov** администратор мгновенно становится одним из зарегистрированных пользователей. Поскольку он суперпользователь, программа **su** не запрашивает у него пользовательский пароль. Выполнив команду **exit** для завершения сеанса, запущенного ко-

мандой **su**, администратор возвращается в свой сеанс суперпользователя. Если же сеанс не завершать, а выполнить команду **su**, то будет предложено ввести пароль суперпользователя, т. к. в данный момент администратор является обыкновенным пользователем. После успешного ввода пароля будет создан очередной дочерний процесс, но уже с правами суперпользователя.

С помощью мониторов процесса любопытно понаблюдать за сменой пользовательского пароля. Для этого из одной консоли от имени обычного пользователя следует вызвать команду **passwd** и, не вводя запрошенный старый пароль, переключиться в другую консоль, откуда вызвать команду **ps -ef**. Найдя процесс **passwd**, можно обнаружить, что он выполняется от имени администратора.

## 2.3. Переменные окружения

На функциональность и защищенность процесса влияют переменные окружения. Они упрощают пользователю настройку программы без изменения способа ее вызова. Запустив **set** без опций, можно увидеть все определённые на данный момент переменные окружения вместе с их значениями. Получить список переменных окружения оболочки можно также командой **printenv**. Будет выведен большой список переменных и их значений (на рис. 2.4 он значительно сокращен):

```
HOSTNAME=samsung.localnet
SHELL=/bin/bash
USER=root
PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/
bin:/usr/games:/usr/lib/java/bin:/usr/lib/java/jre/bin:/usr/li
b/qt/bin
PWD=/root
LANG=ru_RU.KOI8-R
PS1=\u@\h:\w\$
PS2=>
HOME=/root
LOGNAME=root
VISUAL=mcedit
TERMINAL=mcedit
```

Рис. 2.4. Сокращенный список переменных окружения и их значений

Переменные окружения можно также прочитать в файлах **environ**, хранимых в нумерованных подкаталогах активных процессов в директории **/proc**. В качестве разделителя записей используются NULL-символы, и, чтобы представить их в удобной для чтения форме, следует применить конвейер из двух команд:

```
cat /proc/3456/environ|tr ”\0” ”\n”
```

Вторая команда заменяет нулевые разделители переводом строки.

Изменить переменную окружения можно несколькими способами, например:

```
HOME=/tmp
export HOME=/tmp
```

Во втором случае переменная становится глобальной и «видимой» другим процессам. Иначе говоря, переменная интерпретатора экспортируется в переменную среды. С помощью команды **export** переменная окружения не только объявляется, но ей присваивается определённое значение. Если переменная не существует, она будет создана. Если переменная уже имеет какое-то значение, оно будет потеряно.

Изменение значения переменной действительно в течение сеанса. Чтобы присвоить значения переменным окружения постоянно, их следует записать в конфигурационный файл **/etc/profile** для всех пользователей системы или **~/.bash_profile** для конкретного пользователя. Чтение значений переменных из этих файлов происходит при загрузке системы или при авторизации пользователей.

Узнать значение конкретной переменной также можно несколькими способами:

```
echo $HOME
printenv HOME
```

К числу переменных окружения, влияющих на безопасность, в первую очередь относится **PATH** – полный путь к программным файлам.

```
PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/
usr/bin:/bin:/usr/games:/usr/lib/java/bin:/usr/lib/qt
/bin
```

Переменная может принимать ряд значений, разделенных двоеточием, каждое из которых является полным путем к каталогу, в котором должны храниться исполняемые или командные файлы. Суть данной переменной окружения такова, что она позволяет не указывать полный путь к исполняемому файлу при его запуске на исполнение. Командный интер-

претатор считает первое введённое словосочетание в командной строке сначала своей встроенной командой, а затем, если введённое словосочетание не соответствует ни одной встроенной команде, исполняемым файлом (утилитой, программой пользователя и т.п.).

Если словосочетание не является встроенной командой интерпретатора, то он будет выбирать поочерёдно из переменной **PATH** каталог, «пристыковывать» к нему введённое пользователем словосочетание и осуществлять попытку запуска. Если после перебора всех каталогов из переменной пути данное словосочетание не будет найдено, то будет выдано сообщение о том, что «команда не найдена». Описанная переменная окружения имеет отличия у пользователя и суперпользователя.

В переменной окружения пользователя последним, в качестве каталога поиска, указывается текущий каталог «**.**». Это связано с тем, что пользователь также может написать программу или скопировать её откуда-нибудь в свой домашний каталог. Указание в переменной окружения пути поиска текущего каталога позволяет пользователю запускать исполняемый файл простым набором его имени при условии, что пользователь находится в том же каталоге, что и запускаемый файл.

А вот у суперпользователя такой возможности нет, и он должен явно указывать путь к исполняемому файлу, если тот не находится в вышеперечисленных каталогах. Так исключается возможность случайного запуска исполняемого файла из текущего каталога.

Команда **unset** удаляет любые, указанные в качестве параметра, назначенные переменные, уничтожая и саму переменную, и её значение. Оболочка **bash** после выполнения этой команды вообще «забудет» о том, что такая переменная существовала. Удаление переменной действует только в текущем сеансе.

### 2.4. Способы автоматического запуска и остановки программ

Автоматизация действий – одно из главных преимуществ ЭВМ. Следует различать два уровня автоматизации. На первом находятся действия, которые автоматически выполняются системой без участия пользователей. Второй уровень – это действия, которые хотел бы автоматизировать сам пользователь (включая администратора).

Выше уже говорилось о трех видах процессов. Демоны в своем большинстве должны стартовать при загрузке операционной системы. Исключением являются сетевые службы, запускаемые демоном **inetd** при попытке установления соединения с какой-нибудь сетевой службой (**ftp**, **telnet**, **pop3** и т. п.). Если данные службы не работают самостоятельно в

виде демонов, то сканирование возможно открытых портов заставляет **inetd** запускать соответствующие процессы.

После запуска самых необходимых системных процессов ядро системы запускает процесс **init**, который, подобно Адаму, является прародителем всех остальных процессов. Процесс **init** читает и исполняет собственный конфигурационный файл – **/etc/inittab**, листинг которого приведен на рис. 2.5. Несущественные строки из содержимого файла исключены.

```
# These are the default runlevels in Slackware:
#   0 = halt
#   1 = single user mode
#   2 = unused (but configured the same as runlevel 3)
#   3 = multiuser mode (default Slackware runlevel)
#   4 = X11 with KDM/GDM/XDM (session managers)
#   5 = unused (but configured the same as runlevel 3)
#   6 = reboot

# Default runlevel. (Do not set to 0 or 6)
id:3:initdefault:

# System initialization (runs when system boots).
si:S:sysinit:/etc/rc.d/rc.S

# Script to run when going single user (runlevel 1).
su:1S:wait:/etc/rc.d/rc.K

# Script to run when going multi user.
rc:2345:wait:/etc/rc.d/rc.M

# Runlevel 0 halts the system.
l0:0:wait:/etc/rc.d/rc.0

# Runlevel 6 reboots the system.
l6:6:wait:/etc/rc.d/rc.6

# If power is back, cancel the running shutdown.
pg::powerokwait:/sbin/genpowerfail stop

# These are the standard console login getties in multiuser
mode:
c1:1235:respawn:/sbin/agetty 38400 tty1 linux
c2:1235:respawn:/sbin/agetty 38400 tty2 linux
c3:1235:respawn:/sbin/agetty 38400 tty3 linux
c4:1235:respawn:/sbin/agetty 38400 tty4 linux
c5:1235:respawn:/sbin/agetty 38400 tty5 linux
c6:12345:respawn:/sbin/agetty 38400 tty6 linux
```

Рис. 2.5. Фрагменты содержимого файла **/etc/inittab**

Файл этот имеет довольно простую структуру. Как и в иных конфигурационных файлах, строки, начинающиеся символом **#**, являются комментариями и командой **init** не обрабатываются. Каждая строка, не являющаяся комментарием, состоит из 4 полей и имеет вид **id:runlevel:action:process**.

Здесь

1. **id** (идентификатор строки) – некоторая уникальная двух- или односимвольная метка (в современных системах максимальная длина идентификатора составляет 4 символа). Так, метка **si** означает system initialization, **su** – single user, а **rc** – run command;
2. **runlevel** (уровень выполнения) – слово, каждая буква или цифра которого соответствует определенному варианту начальной загрузки системы. Если поле уровня выполнения оставлено пустым, строка выполняется на всех уровнях;
3. **action** (действие) – способ запуска процесса, обозначаемый одним из ключевых слов: **respawn, boot, bootwait, ctrlaltdel, initdefault, kbrequest, off, once, ondemand, powerfail, powerokwait, powerwait, wait** и **sysinit**. Среди этих ключевых слов чаще встречаются следующие:

- **boot** – действие выполняется только один раз при загрузке системы,
- **bootwait** – загрузка останавливается до завершения указанного в строке сценария,
- **initdefault** – строка с этим полем запускается по умолчанию и определяет базовый уровень загрузки,
- **respawn** – запуск процесса происходит в фоновом режиме, а когда процесс завершится (например, с ошибкой), его запускают снова,
- **once** – запуск производится в фоновом режиме однократно,
- **wait** – запуск производится интерактивно, и пока процесс не завершится, никаких других действий не выполняется,
- **ctrlaltdel** – действие, которое выполняется при одновременном нажатии трех клавиш **<Ctrl>+<Alt>+<Del>**. Специалисты считают возможность перезагрузки небезопасной с позиций консольной атаки на систему и рекомендуют такую строку снабдить символом **#**,
- **sysinit** – действие, выполняемое до выдачи приглашения к регистрации. Система дожидается выполнения команды, а затем продолжает загрузку;

4. **process** – программа или скрипт (сценарий) для выполнения.

В ОС Linux предусмотрено несколько вариантов функционирования после начальной загрузки системы, которые называются уровнями выпол-

нения (**run levels**). Всего зарезервировано десять таких уровней, при этом реально используется только половина из них. Используемые уровни выполнения нумеруются с 0 до 6 (уровень 0 используется для останова системы, а 6 – для перезагрузки):

- уровень 1, или S (single), соответствует однопользовательскому режиму. При загрузке на первый уровень не запускается никаких служб;
- уровень 2 соответствует многопользовательскому режиму загрузки системы с отключенной службой NFS;
- уровень 3 обеспечивает сетевой многопользовательский режим. На этом уровне обычно работают компьютеры-серверы и рабочие места администратора;
- уровень 4 обычно не используется и зарезервирован для будущего применения (в Slackware – это режим X–Window);
- уровень 5 соответствует многопользовательскому графическому режиму. На этом уровне обычно функционируют рабочие станции, предоставляя пользователям возможность работать с графической подсистемой X-Window. Сеть на этом уровне настроена, но список запущенных сетевых служб может быть меньше, чем на третьем уровне, так как рабочая станция не предназначена для выполнения серверных функций.

Проанализируем основные рабочие строки в приведенном файле.

**id:3:initdefault:**

Загрузка будет происходить в сетевом многопользовательском текстовом режиме. Если эта строка отсутствует или будет закомментирована, в ходе загрузки последует запрос об установке базового уровня.

**si:S:sysinit:/etc/rc.d/rc.S**

Метод **sysinit** означает, что процесс запускается во время начальной загрузки системы, до регистрации пользователя и перехода на какой-нибудь уровень выполнения. Начальный сценарий, записанный в файле **/etc/rc.d/rc.S**, проверяет и монтирует дисковые файловые системы, инициализирует область подкачки – словом, делает все, без чего дальнейшая полноценная работа системы невозможна. Стартовые сценарии, а также каталоги, в которых они располагаются, имеют префикс **rc**, что означает run command – «команда запуска».

**rc:2345:wait:/etc/rc.d/rc.M**

На всех многопользовательских уровнях, включая третий, предусмотрен запуск сценария **/etc/rc.d/rc.M**. В этом командном файле много похожих фрагментов, связанных с запуском сервисов по условию. Типовой фрагмент для Slackware Linux выглядит так:

```
# Start the sendmail daemon:
if [ -x /etc/rc.d/rc.sendmail ]; then
  /etc/rc.d/rc.sendmail start
fi
```

Сценарий, запускаемый на каждом из уровней, должен завершить все текущие процессы и запустить другие, соответствующие новому уровню. Например, администратор в целях безопасной отладки системы с помощью команды **telinit 1** переходит из многопользовательского сетевого режима 3 на однопользовательский уровень 1. При этом все службы, «заведующие» многопользовательским сетевым режимом, должны завершиться (точнее – на уровне 1 никаких процессов кроме системных, **init** и **/bin/sh** не должно быть, т. е. при переходе на 1-й уровень никакие процессы и сервисы не стартуют).

В каталоге **/etc/rc.d** имеются директории с названиями **rc0.d**, **rc1.d**, **rc2.d**, **rc3.d**, **rc5.d**, **rc6.d**, где цифрами обозначены уровни выполнения. Все элементы каталогов **rc#.d** – символические ссылки. Сами сценарии находятся в каталоге **/etc/init.d**. Их имена начинаются буквами **K** (kill) и **S** (start). При входе на данный уровень запускаются сценарии на букву **S**, а при выходе с уровня – сценарии на букву **K**. Числа, следующие за буквой **K** или **S**, определяют порядок исполнения сценария.

При возникновении необходимости отключения на каком–либо уровне определенных служб необходимо удалить пару символических ссылок: одну на **S**, а другую на **K** (в RedHat это делает специальная утилита **chkconfig**). Если за запуск службы отвечает отдельный файл, его следует удалить или снять признак исполняемости. Если это делает фрагмент сценария, его можно *закомментировать*.

Все стартовые сценарии служб, которыми может воспользоваться система, принято хранить в каталоге **/etc/rc.d/init.d**. Эти сценарии используются для запуска или останова различных служб. Запустить или остановить службу можно, просто вызвав соответствующий сценарий с параметром **start** или **stop**. Часто ту же самую задачу выполняет и специальная утилита **service**, которая проверяет, есть ли указанный стартовый сценарий, и запускает его.

Из строчки с **initdefault** процесс **init** «узнает», что уровень выполнения по умолчанию – третий (многопользовательский консольный), и выполняет все строки из **inittab**, в поле «уровень исполнения» которых есть уровни 3. В частности, запускается сценарий **rc.M** из строки «**rc:**…». Метод запуска wait, поэтому процесс **init** ждет, пока не выполнится вышеуказанный сценарий, а потом продолжает разбор **inittab**.

Уровни 0 и 6 – специальные. Они соответствуют останову и перезагрузке системы. В сущности, это удобные упрощения для действий, обратных загрузке на базовый уровень: все службы останавливаются, диски размонтируются. Соответствующие каталоги **rc0.d** и **rc6.d** будут состоять почти сплошь из ссылок вида **K***, но как минимум один сценарий, **killall**, будет запущен с параметром start. Этот сценарий остановит все процессы, которые не были остановлены K-сценариями: программы пользователей, демоны, запущенные администратором вручную, и т. п. Строки остановки служб в сценарии выглядят так:

```
# Stop the Apache web server:
if [ -x /etc/rc.d/rc.httpd ]; then
  /etc/rc.d/rc.httpd stop
fi

# Shut down the SSH server:
if [ -x /etc/rc.d/rc.sshd ]; then
  /etc/rc.d/rc.sshd stop
fi
```

Остальные уровни никоим образом в ОС Linux не описаны, однако администратор может использовать их, определяя особый профиль работы системы.

Переход с уровня на уровень выполняется по команде **init N**, где **N** – номер уровня. Иногда для этого используют команду **telinit N**, которая является символической ссылкой на **init**. Узнать текущий уровень выполнения можно с помощью команды **runlevel**.

Отключение неиспользуемых служб в текущем сеансе производится администратором с помощью команды

**<service> stop** ,

где **<service>** – имя службы. Для полного исключения запуска ненужных служб в дальнейшем необходимо использовать утилиту **chkconfig** с соответствующими параметрами.

В дистрибутиве Slackware необходимо снять флаг исполнения с соответствующего файла в каталоге **/etc/rc.d/** или закомментировать строки запуска в файлах типа **rc.M** для демонов, не имеющих собственных сценариев запуска. Также не следует забывать про файл **/etc/rc.d/rc.local**.

Некоторые сетевые демоны запускаются посредством супердемона **inetd** (**xinetd**). Демоны, запускаемые из **inetd**, могут быть отключены путём комментирования соответствующих строк в файле **/etc/inetd.conf**.

Например, служба SSH запускается своим собственным скриптом **/etc/rc.d/rc.sshd**. Такую службу можно отключить, просто лишая этот файл права на исполнение:

```
chmod -x /etc/rc.d/rc.sshd
```

## 2.5. Периодически запускаемые процессы

За запуск в определённое время, а также за периодический запуск пользовательских процессов отвечают две службы: **cron** и **at** (они могут иметь свойственные демонам имена **crond** и **atd**).

Демон **crond** запускается во время начальной загрузки системы и остается в активном состоянии до ее выключения (если администратор его принудительно не завершит). Эта относительно простая программа-демон, как и другие демоны, проводит свободное время в состоянии «спячки», ежеминутно «просыпаясь» для чтения файлов заданий. Если задание на данную минуту имеется, демон «вырезает» из соответствующей строки запланированную команду/программу или последовательность команд/программ с параметрами либо без них и передает всё это для исполнения командному интерпретатору. Затем он снова «засыпает».

Служба **crond** обычно используется для периодического уничтожения старых журналов аудита, чистки файловой системы и прочих подобных потребностей.

Для управления демоном **crond** (точнее – его «озадачивания») предусмотрена отдельная утилита, именуемая **crontab**. Она запускается пользователем из командной строки, и ей можно передать один из трех ключей (если команду запускает суперпользователь, то в качестве параметра команды он может указать имя учетной записи пользователя, в этом случае действия будут выполняться над файлом заданий пользователя):

**-l** – вывести содержимое файла заданий;
**-e** – отредактировать файл с заданиями;
**-r** – удалить файл с заданиями.

Каждое задание в файле заданий представляет собой текстовую строку (как обычно, строка, начинающаяся символом **#** , является комментарием и демоном не исполняется), в которой записано пять временных отметок, определяющих время и/или периодичность исполнения команды: минуты, часы, дни месяца, порядковые номера месяца, дни недели и сама команда/последовательность команд либо программа/последовательность программ с параметрами или без них. Итого в строке шесть полей, разделенных пробелами:

**минуты часы день месяц день_недели команда** ,

причем в поле **команда** пробелы выполняют свою обычную роль разделителя аргументов. Команду следует вводить с указанием полного пути к программному файлу.

Временные параметры задаются числами, которые должны соответствовать своему диапазону:

**минуты** – от 0 до 59
**часы** – от 0 до 23
**день** – от 1 до 31
**месяц** – от 1 до 12
**день_недели** – от 0 до 6 (0 – воскресенье)

Символом-джокером ***** обозначается любое число. Для указания диапазонов могут использоваться дефисы и запятые. Вот некоторые примеры задания временных параметров в файле **crontab**

| | | | | | |
|---|---|---|---|---|---|
| * | * | * | * | * | каждую минуту |
| 10 | * | * | * | * | через 10 минут после начала каждого часа |
| */6 | * | * | * | * | через каждые 10 минут |
| 20 | 12 | * | * | * | каждый день в 12:20 |
| 15 | 9, 21 | 7 | * | * | в 9:15 и в 21:15 каждый седьмой день месяца |
| 30 | 21 | 8-12 | 4 | * | в 21:30 с восьмого по двенадцатое апреля |
| 10 | 0 | * | * | 0 | по воскресеньям в 00:10 |

Файлы заданий хранятся в каталогах **/var/spool/cron/tabs**. Каталог содержит по одному файлу на каждого из пользователей (если использование планировщика заданий им не запрещено) в случае, если пользователь создал задание. Эти файлы не появляются автоматически при регистрации нового пользователя. Они создаются при первом обращении пользователя к службе **crond**.

Имя **crontab**-файла совпадает с учетным именем пользователя. Поэтому задание, найденное, например, в файле **ivanov**, позволяет демону передать команду на исполнение соответствующему экземпляру оболочки. По имени файла производится поиск строки в файле **/etc/passwd**, из неё выбираются UID, GID и всё необходимое для создания процесса. При выполнении задания демон **crond** не делает отметки в **crontab**-файле, поэтому невыполненные задания с истекшим сроком повторно не выполняются. Демон не удаляет из файлов выполненные задания, он только читает файл заданий, и имеющиеся в нем строки при определенных обстоятельствах могут служить доказательством противоправного деяния.

В каталоге **/etc** должен существовать файл **cron.deny**, в котором построчно перечисляются учетные имена пользователей, которым *пользование* планировщиком **cron** запрещено. По умолчанию в нем перечислены имена псевдопользователей, случайным обладателем которых может стать злоумышленник. Для того чтобы разрешить *отдельным* пользователям планирование заданий, администратор должен создать в этом же каталоге другой файл **cron.allow**, в который построчно можно записать нужные учетные имена.

Если файл **cron.deny** существует, а файл **cron.allow** не создан, право запуска **crontab** имеют *все* пользователи, кроме тех, кому это явно не разрешено (в файле **cron.deny**). Вопреки правилу информационной безопасности «должно быть запрещено все, что явно не разрешено», записи в «разрешительный» файл имеют приоритет. Так, если один и тот же пользователь по невнимательности администратора оказался записан в оба файла, разрешение на команду **crontab** ему будет обеспечено. Парадоксальное решение – чтобы запретить *всем* пользователям (включая администратора!) запуск планировщика задач, необходимо в указанном каталоге создать *пустой* файл **cron.allow**. Если ни одного из двух указанных файлов не существует, пользоваться планировщиком заданий может только администратор.

Учитывая несомненную опасность доступа обычных пользователей к этим файлам, не вызывают сомнения запрещения их доступа к каталогам **/var/spool/cron** и **/var/spool/cron/tabs**. Право читать и редактировать эти файлы «вручную» имеет только администратор. Для того чтобы пользователи имели кратковременное право на запись в эти файлы, утилита **crontab** имеет установленный бит **SUID**.

В ОС Linux для неинтерактивного выполнения задач (команд, программ, последовательности команд, последовательности программ или их сочетания) используется специальный демон **atd** , так называемый диспетчер очередей задач.

Разновидностью неинтерактивного однократного запуска команд является формирование из них задания и постановка его на выполнение в определённое время. Реализуется это при помощи команды **at**. Формат команды выглядит так:

```
at hh.mm dd.mm.yy
```

Вызванная команда отвечает приглашением **at >** и ожидает ввода команд интерпретатора или программ с параметрами или без них. Необходимо иметь в виду, что выполняться задание будет под управлением командного интерпретатора **bash**. Правила набора команд в строках идентичны правилам для командного интерпретатора, т. е. их можно набирать

отдельно, группировать, объединять и т. д. Для завершения ввода команд используется комбинация **<Ctrl>-<D>**, после чего команда строкой

```
job 7 at 2009-09-17 12:31
```

извещает пользователя о постановке задания в очередь и присвоении ему номера.

При этом в каталоге **/var/spool/atjobs** создаётся файл с именем, например, **a00004013bbe08** (**a00004** означает задание номер 4 в очереди, в **013bbe08** закодированы время и дата 00:48 05.05.09), принадлежащий пользователю и группе **daemon** с маской доступа **700**. После выполнения задания этот файл удаляется.

Для того чтобы ознакомиться со списком заданий, следует ввести команду **atq** (at queue – очередь at). Задания выводятся построчно, и в каждой строке указываются номер задания, дата и время его выполнения, а также имя пользователя, в интересах которого выполняется задание.

Для удаления задания используется еще одна команда **atrm N**, где N – номер задания.

Для того чтобы разрешить или запретить пользователям однократное планирование процесса, система по умолчанию предусматривает создание в каталоге **/etc** двух файлов: **at.allow** и **at.deny**. Порядок их использования аналогичен правилам, относящимся к планировщику **crond.**

Периодические процессы, порождаемые одним или несколькими связанными в конвейер исполняемыми файлами, можно создать с помощью службы **watch**. Команда для запуска сервиса выглядит так:

```
watch -n 60 'ps -ef | grep syslogd'
```

например, эта команда каждые 60 секунд ищет в списке процессов службу **syslogd** и выводит ее на экран. Этот сервис больше подходит для наблюдения за быстро изменяющимися процессами.

### 2.6. Запуск и остановка программ в интерактивном и фоновом режимах

После прохождения процедуры аутентификации пользователя система запускает экземпляр командного интерпретатора, который обеспечивает диалоговый режим человека и машины. В качестве интерпретатора командной строки в Linux в основном используется **/bin/bash** (bash – Bourne Again Shell – «рождённый заново шелл», что является реализацией Unix shell, написанной в 1987 г. Brian Fox для GNU Project).

Небольшое число команд реализовано в самой оболочке, поэтому они называются внутренними. К ним относятся такие команды, как **fg**, **bg**, **alias**, **limits**, **history**, **echo**, **jobs** и другие. Подавляющее большинство команд являются внешними, и имя введенной команды считается именем какого-либо исполняемого файла.

Исполняемые файлы располагаются в нескольких каталогах: **/bin**, **/sbin**, **/usr/bin**, **/usr/sbin** и др., хотя запустить процесс можно из любого каталога, на который у пользователя есть права чтения и поиска. Вызывать команды можно, задавая абсолютный путь к ее исполняемому файлу либо используя «короткое» имя файла. Найти нужный файл по его короткому имени программе-оболочке помогает переменная окружения **PATH**. В ней обычно поименованы каталоги **/bin**, **/sbin**, **/usr/bin**, **/usr/local/bin**, разделенные двоеточием. Для администратора в этом перечне должен быть исключен текущий каталог, обозначаемый одной точкой «**.**», поскольку он может привести к случайному запуску опасных программ–двойников. В базовых настройках интерпретатора такой запуск исключен. Тем не менее администратору всегда нужно быть бдительным, и переменные окружения его командной оболочки не должны оставаться без внимания.

Команда обычно состоит из трех частей:

- имени самой команды;
- опций;
- операндов (аргументов).

Опции и операнды в простых командах могут отсутствовать. Опции определяют алгоритм выполнения программы. Они могут записываться в коротком или длинном виде. Короткие опции состоят из дефиса и одиночного символа в нижнем или верхнем регистре. Несколько коротких опций могут объединяться. Так, нижеприведенные команды являются эквивалентными:

```
ls --long --inode --all
ls -l -i -a
ls -lia
```

Существует соглашение относительно имен опций. Оно изложено в документе CNU Coding Standarts и является обязательным для программистов, пишущих программы для Linux. Так, обычно символом **l** (long) обозначают длинный (расширенный) вывод данных, символом **a** (all) – отображение всех объектов, а символом **h** (help) – вывод подсказки по синтаксису команды.

Длинные опции состоят из двух дефисов, после которых следует имя из символов нижнего и верхнего регистров. Такие опции легче запоминать и читать. Команды «понимают» (по крайней мере, должны понимать) и длинные, и короткие опции.

В то же время существуют программы, использующие иной синтаксис. Так, очень известная и необычайно полезная программа **dd** использует командную строку вида

```
dd if=/dev/fd0 bs=1024 count=100 skip=1
of=/mnt/floppy/fda conv=noerror,notrunc,fsync ,
```

где опции и входные данные могут записываться в произвольном порядке, причем между именем параметра и его величиной указывается знак равенства.

Если команда длинна, неудобна, имеет большое число обязательных аргументов и в то же время часто используется, пользователь может объявить для нее псевдоним (англ. alias). Например, пользователю приходится регулярно использовать сменный полупроводниковый носитель, и с целью укоротить команду его монтирования он может предусмотреть замену:

```
alias mf="mount -t vfat -o iocharset=koi8-r /dev/sdb1 /mnt/usb"
```

Замена вымышленного имени настоящей командой возлагается на командный интерпретатор. В списке процессов, который выводится утилитами **ps** и **top**, отображаются реальные команды. Псевдоним действует в течение одного сеанса и только от имени пользователя, который его объявил. Если следует сделать его постоянным, эту строку необходимо записать в файл **.bash_profile** в домашнем каталоге пользователя.

Команды могут исполняться как в интерактивном, так и в фоновом режимах. В интерактивном режиме командный интерпретатор выводит очередное приглашение для ввода только после завершения выполнения предыдущей команды. Указав в конце командной строки символ **&** (после пробела), пользователь может запустить фоновый процесс. При этом независимо от времени выполнения команды интерпретатор мгновенно выведет строку вида **[1] XXX** и приглашение для ввода следующей команды. В квадратных скобках отображается порядковый номер пользовательского фонового процесса, а следом за ним – его **PID.**

Если это специально не ограничено, пользователь может запустить произвольное число фоновых процессов. Чтобы узнать, какие фоновые процессы запущены, пользователю следует ввести команду **jobs**. Отобразится номер процесса в квадратных скобках и имя выполняемой команды. Возврат фонового процесса на «передний план» (интерактивный режим) производится при помощи команды **fg %1**, где цифра указывает уже упомянутый номер задания. Чтобы вновь вернуть процесс в фоновый режим, требуется нажать клавишу **<Ctrl>** и, удерживая её нажатой, нажать клавишу **<Z>** . После появления сообщения об остановке процесса ввести команду **bg %1.**

В одной строке можно ввести несколько команд подряд, разделяя их точкой с запятой, например:

```
clear; pwd; date
```

При использовании в качестве разделителя команд символов **&&** выполнение очередной команды будет производиться только после успешного завершения предыдущей команды. Если анализировать код ошибок, то успехом считается возврат нулевого кода, а неудачей – все остальные значе-

ния. В примере, приведенном ниже, с помощью команды **grep** идет поиск учетной записи пользователя сначала в файле паролей, а при ее обнаружении – поиск в файле групп:

```
grep "ivanow" /etc/passwd && grep "ivanow" /etc/group
```

Разделитель **||** используется тогда, когда надо запустить следующую команду при ошибочном завершении предыдущей команды, например:

```
ls –l /root || ls –l /home
```

Если эту строку запустит обычный пользователь, то будет исполнена только вторая команда, т. к. прав на чтение каталога администратора у него нет.

В консоли ОС Linux имеется возможность использования манипулятора «мышь». Для этого должен быть установлен, настроен и запущен демон **gpm**. Использование манипулятора «мышь» заключается в возможности выделения произвольного фрагмента экрана консоли (при этом выделяемый фрагмент сохраняется в буфере) и, в дальнейшем, вставки его в произвольное место на экране. Вставка фрагмента производится в позиции за курсором. Здесь необходимо уточнить, что вставка в произвольное место экрана возможна, если только пользователь использует в данный момент программу, работающую в полноэкранном режиме, например текстовый редактор **mcedit**, а в случае работы в командном интерпретаторе вставка возможна только в текущей строке. Таким образом, данную возможность можно использовать для копирования ранее введённых командных строк или их частей и последующей вставки в текущую командную строку даже в другой виртуальной консоли. Выделение фрагмента экрана производится при нажатии и удержании левой кнопки манипулятора «мышь». Дойдя до конца нужного фрагмента, следует отпустить кнопку. После необходимо нажать правую кнопку в двухкнопочном манипуляторе «мышь» или среднюю в трёхкнопочном, в том месте, где нужно сделать вставку.

Пользователю нет необходимости многократно вводить одни и те же команды. В командном интерпретаторе **bash** имеется буфер памяти команд. С помощью клавиши ↑ можно вернуться к предыдущей команде, а нажимая ее многократно, можно «пролистать» список команд в обратную сторону на нужное число позиций, аналогично использование клавиши ↓, только в обратную сторону. То же самое можно сделать с помощью команды **history** – при этом выводится перечень ранее введенных команд (по умолчанию запоминается список из 500 команд). Этот список хранится для каждого зарегистрированного пользователя в отдельном текстовом файле в его домашнем каталоге. Так, историю команд интерпретатор **bash** хранит в файле **.bash_history** (рис. 2.6).

```
mc
mount
dd if=/dev/hda6 of=/tmp/bootsect.lnx bs=1 count=512
lilo cnfig
mount /dev/hda1 -vfat /mnt/hda1
dd if=/dev/fd0 of=/tmp/bootsect.lnx bs=1 count=512
umount /mnt/floppy
```

Рис. 2.6. Фрагмент файла «истории» команд пользователя

В некоторых случаях существование файла истории команд может представлять угрозу, причем не только для взломщика, но и для администратора. Например, в список команд может попасть случайно введенный пароль. Стирание или редактирование текстового файла **.bash_history** проблему не решает, поскольку история команд находится в памяти процесса **/bin/bash** и перезаписывается в файл при завершении сеанса. Это, в частности, можно наблюдать, работая от имени одного пользователя из разных виртуальных консолей. В ходе сеанса в каждой из консолей будет отображаться «своя» история команд, которые при завершении работы переписываются в один файл. При этом наблюдаемая последовательность ввода команд может совершенно не совпадать с реальной очередностью. Злоумышленник, совершающий консольную атаку на систему, может для запутывания «командного следа» поочередно работать из разных виртуальных консолей.

Один из способов стирания истории команд злоумышленником заключается в удалении файла **.bash_history** и создании вместо него символической ссылки с аналогичным именем, которая указывает на специальный файл **/dev/null**. Команда записывается в виде

**ln –s /dev/null ~/.bash_history**

Для быстрой и кардинальной очистки файла истории команд рекомендуется запустить из командной строки команду

**export HISTSIZE=0** ,

устанавливая в нуль соответствующую переменную окружения оболочки. Файл истории команд будет обнулен и при завершении сеанса останется пустым.

Наконец, следует уделить внимание завершению процессов. Любой пользователь может завершить свой сеанс встроенной командой оболочки **exit**. Наряду с этим используется команда **logout** или просто комбинация клавиш **<Ctrl>-<D>**. При этом командный интерпретатор завершает свою работу, и пользователь вновь оказывается перед необходимостью прохождения процедуры регистрации.

Для того чтобы завершить работу всей системы, требуются полномочия администратора. Завершить работу можно одной из команд:

**shutdown**, **poweroff** или **halt**. Нажатие «магической» комбинации **<Ctrl>+<Alt>+<Del>** в зависимости от настроек в файле **/etc/inittab** может привести либо к останову, либо к перезагрузке системы.

У команды **shutdown** есть обязательный параметр – время начала останова (например, **–t 3** означает остановку системы через три минуты, а **–t now** — немедленный останов) и факультативный: **-r** (**r**eboot, перезагрузка) и **-h** (**h**alt, останов). Без необязательных параметров выполняется переход на первый уровень выполнения, для чего перезапускается стартовый командный интерпретатор суперпользователя. Нажатие **Ctrl+Alt+Del**, или кнопки выключения питания (в системах, где эта кнопка ничего не выключает, а лишь посылает соответствующий аппаратный сигнал), приводит к запуску команды **shutdown -r** или **shutdown -h**.

Останов системы может занимать больше времени, чем ее загрузка. Так, процессы, работающие с дисковой памятью, получив сигнал завершения, могут некоторое время заниматься дописыванием изменений в файл. Остановка сетевой службы также может длиться долго, так как требуется сообщить о закрытии сервиса каждому клиенту.

При сбое электроснабжения, когда сервис, обслуживающий устройство бесперебойного питания, сообщает, что ресурсы на исходе, предпочтительнее быстро остановить процессы, чем дожидаться отключения электроэнергии на работающей системе. Для этого можно послать всем процессам сначала сигнал **TERM**, а короткое время спустя – **KILL**. Для обработки таких ситуаций в **inittab** есть методы, начинающиеся со слова power, а в **/etc/rc.d** – специальный сценарий **rc.powerfail**. На самый крайний случай существуют команды **halt** и **reboot -f**, однако их почти мгновенное действие практически эквивалентно внезапному отключению питания, и использовать их не рекомендуется.

## 2.7. Средства взаимодействия между процессами

В ОС Linux имеются разнообразные средства межпроцессного взаимодействия. Из UNIX System V заимствован механизм **IPC** (interprocess communication), который включает в себя три вида коммуникаций: сообщения (messages), разделение памяти (shared memory) и семафоры (semaphores). Обмен сообщениями позволяет одним процессам передавать данные другим процессам. Разделение памяти даёт возможность процессам совместно использовать определённые области виртуального адресного пространства. Семафоры обеспечивают синхронизацию процессов. Помимо этого имеется механизм сигналов (signals) и каналов (pipes), рассмотрением которых мы ограничимся.

На уровне интерактивного взаимодействия с системой чаще всего приходится пользоваться сигналами и каналами. Сигнал сообщает процессу о наступлении асинхронного события. Как это не покажется странным, для посылки сигнала используется команда **kill** (дословно – убить). Действительно, на практике сигналы чаще всего используются для принудительного завершения какого-либо процесса, вышедшего из-под контроля. Получить перечень возможных сигналов позволяет команда **kill –l**, по которой выведится список из 62 сигналов. Некоторые сигналы вызывают конкретные действия, а иные зарезервированы. В качестве опции команды **kill** можно указывать как символьное, так и числовое значение сигнала.

Большинство сигналов адресуются непосредственно к программе, и для их обработки программисту необходимо написать соответствующую процедуру. Для исследования процессов при выполнении лабораторных работ М.Э. Пономаревым написана небольшая программа с именем **signignore**, которая позволяет игнорировать все сигналы, за исключением самого главного. Сигнал **kill -9 PID** адресуется не процессу, а планировщику задач, поэтому «непослушный» процесс не сможет такой сигнал перехватить и игнорировать.

Пользователь может послать сигнал только тем процессам, которые сам запустил. Администратор имеет право прекратить исполнение любого процесса. Для «убийства» всех процессов, созданных одной программой, также требуются полномочия суперпользователя, для чего он должен воспользоваться командой **kill** (см. руководство **man** или **info**).

Несколько большим удобством отличается утилита **skill**, позволяющая отправлять процессам сигналы, идентифицируя их не только по номеру, но и по имени пользователя и идентификатору терминала. С помощью этой утилиты администратор может заблокировать или разблокировать пользовательский ввод и вывод информации на произвольном локальном или сетевом терминале.

## 2.8. Перенаправление ввода/вывода

Все программы в UNIX/Linux запускаются с тремя открытыми файлами: стандартным вводом (**stdin**), стандартным выводом (**stdout**) и стандартным выводом сообщений об ошибках (**stderr**). В оболочке **bash** поддерживается возможность перенаправления информации в командной строке посредством использования символов «**<**» для перенаправления стандартного ввода и «**>**» для стандартного вывода. По умолчанию стандартный ввод и стандартный вывод ассоциированы с консолью (ввод с клавиатуры, вывод на монитор). Но часто возникает необходимость выводить информацию не на экран (пользователя в это время у компьютера может не оказаться либо объем выводимой информации может оказаться слишком

большим для ее чтения с экрана), а в файл, который можно прочитать или

распечатать потом.

Перенаправить данные можно различными способами. Например, команда

```
ls –la /home/user1 > /root/user1.ls
```

записывает список файлов пользователя в текстовый файл. Если этот файл не существует, то он создается. Если он существует, то он перезаписывается, т. е. его прежнее содержимое стирается.

```
echo ″Содержимое каталога /home/user1″ >>
/root/user1.ls
```

в этом случае выводимая информация дописывается к содержимому файла. Если файл не существует, при первом исполнении команды он будет создан.

Иногда вывод информации необходимо перенаправить в какое-либо устройство. В этом случае мы адресуемся к специальному файлу устройства, который является посредником между командным интерпретатором и драйверами устройства. Например, читаем ранее созданный образ дискеты и копируем его на сменный носитель:

```
cat image_fd > /dev/fd0
```

Выводим содержимое файла на принтер, подключенный к первому параллельному порту:

```
cat file1.txt > /dev/lp0
```

Воспроизводим звуковой файл через звуковой адаптер:

```
cat /usr/share/sndconfig/sample.au > /dev/audio
```

Аналогично можно изменить стандартный ввод информации, которым по умолчанию является клавиатура. С помощью перенаправления ввода можно записывать в файл сигналы с устройства, подключенного к последовательному интерфейсу. Комбинируя команды перенаправления ввода и вывода, можно передавать данные программе из одного файла и выводить результаты в другой файл. Так, например, утилита **iconv** позволяет изменить кодировку символов в файле, для чего целевой файл следует указать с операциями перенаправления вывода:

```
iconv -f utf-8 -t cp1251 < in_file_utf > out_file_win

iconv -f cp1251 -t utf-8 < in_file_win > out_file_utf

iconv -f koi8-r -t cp1251 < in_file_koi > out_file_win

iconv -f cp1251 -t koi8-r < in_file_win > out_file_koi
```

Еще одним средством перенаправления информационных потоков в Linux являются каналы, в этом случае они являются средством межпроцессного взаимодействия. Каналом называется однонаправленное логическое устройство, предназначенное для передачи данных от одного процесса другому. По своей сути канал представляет собой буфер памяти небольшого размера (обычно 4 Кб), в который один процесс может записывать данные, а другой – эти данные оттуда читать. И запись, и чтение данных осуществляется последовательно: данные всегда читаются в том порядке, в каком они были записаны. Канал может быть использован как средство связи между родственными процессами. Каналы могут иметь имя подобно файлам либо обходиться без него, т. е. могут быть именованными или неименованными.

Более простыми и распространенными являются неименованные каналы. В языке командных интерпретаторов неименованный канал обозначается символом «**|**». В некоторых источниках он именуется конвейером и служит средством группирования команд с передачей информации между стандартным выводом одной команды и стандартным вводом следующей за ней. Одним из простейших примеров использования неименованных каналов является постраничный просмотр на экране большого списка файлов, формируемый соответствующей командой. Так, чтобы последовательно просмотреть листинг файлов каталога большого объема, используют сочетание команд:

```
ls –la /bin|more
```

Вторая команда **more** как раз и обеспечивает поэкранный вывод данных с возможностью «листать» страницы вперед при нажатии любой клавиши. Комбинированная команда

```
ls –la /bin|less
```

позволяет постранично «листать» файл в обе стороны. Правда, при работе в графическом режиме и эмуляции текстового терминала в окне просматривать файл можно и без дополнительных команд, используя для этого боковые полосы прокрутки.

Приведем еще несколько примеров использования неименованных каналов. Утилита **cat** читает текстовый файл **file_name** и передает последовательный поток символов программе **wc**, которая подсчитывает число строк, слов и символов в файле:

```
cat <file_name>|wc
```

Впрочем, команда

```
wc <file_name>
```

делает то же самое, но пишется короче.

Утилита **ps** выводит таблицу процессов, а утилита **head** отображает

первые 20 строк этой таблицы:

```
ps –ef | head -20
```

Утилита **dd** читает содержимое оптического диска, установленного в привод **/dev/hdc**, а **grep** ищет в считываемых данных строку «Linux»:

```
dd if=/dev/hdc | grep "Linux"
```

В следующем примере уже упомянутая утилита **dd** считывает из логического раздела на жестком диске второй 4-килобайтный блок данных, находит в нем описатель 5-й группы блоков и выводит его шестнадцатеричный и символьный дамп на экран:

```
dd if=/dev/hda7 bs=4096 skip=1 count=1 | dd bs=32 skip=4 count=1 | xxd
```

Утилиты **ps** и **top** отображают каждый процесс, участвующий в конвейере, отдельной строкой.

Как уже указывалось, емкость канала ограничена размером буфера памяти, и если передающий процесс записывает в него данные быстрее, чем принимающий читает их, то он приостанавливается до тех пор, пока читающий процесс не освободит канал, и наоборот, если в канале ничего нет, то читающий процесс приостанавливается до появления информации в канале. Таким образом происходит синхронизация передачи по каналу.

Именованные каналы в UNIX системах представлены в виде специальных файлов, по своей сути подобных файлам устройств с последовательным доступом. За именованными каналами закрепилось еще одно название – файлы **FIFO** (First-In, First-Out – первым вошел, первым обслужен). К таким каналам может обратиться любой процесс. В консольном режиме именованные каналы создаются командой **mkfifo**, например:

```
mkfifo /tmp/fifo1
```

Каталог **/tmp** как директория с равным и полным доступом очень удобен для создания файлов, к которым будут обращаться многие. К созданному именованному каналу можно обращаться как к обычному файлу. Для того чтобы убедиться в этом, следует войти в систему из двух текстовых консолей. Затем в командной строке первой консоли ввести команду чтения из созданного именованного канала:

```
cat < /tmp/fifo1
```,

а во второй консоли – команду перенаправления вывода в канал:

```
cat > /tmp/fifo1
```

После этого во второй консоли можно ввести произвольную строку, завершить буферизированный ввод нажатием клавиши **<Enter>** и, перейдя в

первую консоль, прочитать введенную строку на экране. Удаляется именованный канал так же, как обычный файл.

Емкость именованного канала в системах Linux составляет 4 Кб. Внутри канала имеется буфер, способный воспринимать ввод, даже если из канала никто не читает.

## 2.9. Файловая система `/proc` как «зеркало» процессов

В ОС Linux присутствует виртуальная файловая система, монтируемая при загрузке к каталогу **`/proc`** **(**от слова process**)**. В отличие от других файловых систем, размещаемых на внешних машинных носителях, эта файловая система располагается в оперативной памяти, обеспечивает связь с ядром и предназначена для предоставления текущей информации о компьютерной системе (состояние ядра, процессы, параметры компьютера и т. д.), чем представляет большой интерес, в том числе с позиций компьютерной безопасности. Многие из утилит, выводящие информацию о системе (например, ранее рассмотренные команды **`ps`** и **`top`**), берут исходные данные именно из каталога **`/proc`**.

В виртуальной файловой системе **`/proc`** размер почти всех файлов равен нулю, и любой пользователь, включая суперпользователя, лишен права на запись в эти файлы. Право на чтение почти всех файлов имеет каждый зарегистрированный пользователь системы. В то же время некоторая часть виртуальных файлов доступна администратору на запись, например для установки параметров ядра.

Объект **`/proc`** включает в себя файлы и каталоги с числовыми и символьными именами. Каталоги с числовыми именами содержат информацию о каждом выполняющемся процессе, а само имя каталога образовано от идентификатора выполняемого процесса (**`PID`**). При создании процесса соответствующий каталог появляется, а при уничтожении процесса – исчезает. В каждом из таких «числовых» каталогов содержатся одни и те же файловые объекты (табл. 2.1).

Если в процессе выполнения сам исполняемый файл оказался удаленным, ссылка **`exe`** позволит восстановить удаленный файл, скопировав его образ из оперативной памяти [7].

В подкаталоге **`fd`** (**`fd`** – file descriptor) можно получить оперативную информацию обо всех файлах, которые были открыты данным процессом. Именно отсюда берет информацию команда **`lsof`**, часто используемая для наблюдения за открытыми файлами.

Виртуальный файл **`cmdline`** содержит полные параметры командной строки, использованные при запуске программы. Отсюда информация переписывается в файл истории команд.

Таблица 2.1

| Имя файла в подкаталоге /proc/PID | Содержимое файла |
|---|---|
| **cmdline** | Список аргументов командной строки процесса (параметры, передаваемые программе). Список представлен одной последовательностью, в которой аргументы отделяются друг от друга байтами вида «0x00» |
| **cwd** | Символическая ссылка на каталог, который установлен текущим (рабочим) для процесса (**cwd** – change working directory) |
| **environ** | Переменные окружения (**USER**, **HOME**, **PATH** и др.). Элементы списка разделяются друг от друга байтами вида «0x00» |
| **exe** | Ссылка на исполняемый файл процесса |
| **fd** | Каталог, содержащий символические ссылки на файлы, открытые данным процессом |
| **maps** | Карта распределения адресного пространства процесса в виде форматированного текстового файла |
| **mem** | Память процесса |
| **root** | Ссылка на корневой каталог процесса (обычно **/**) |
| **stat** | Состояние процесса на момент просмотра |
| **statm** | Состояние памяти процесса. Содержит список из 7 чисел, разделенных пробелами. Эти числа:<br>• общий размер процесса в мегабайтах<br>• размер резидентной части процесса<br>• размер совместно используемой памяти<br>• размер сегмента кода<br>• размер загруженных библиотек<br>• объем памяти стека<br>• число модифицированных страниц памяти |
| **status** | Состояние процесса в виде, предназначенном для пользователя. Файл содержит идентификатор процесса, родительского процесса, реальный и эффективный идентификаторы пользователя, статистику использования памяти и битовые маски, указывающие, какие сигналы процессом перехватываются, игнорируются или блокируются |

О назначении файлов и каталогов с символьными именами можно догадаться по этим именам. Например, сведения о последовательных портах содержатся в файле **/proc/tty/driver/serial**. Обращение к

этим файловым объектам позволяет получить текущую информацию об аппаратной платформе и драйверах устройств, а также много иной интересной информации.

Виртуальный текстовый файл **/proc/mounts** содержит информацию об уже смонтированных файловых системах. Почти такая же, но более подробная информация записывается в реальный файл **/etc/mtab**.

Если имеются сомнения в том, что утилита **ps** правильно отображает все процессы, можно обратиться к каталогу **/proc**. Достаточно ввести друг за другом две команды:

```
ls -d /proc/*|grep [0-9]|wc -l
```

и

```
ps ax|wc -l
```

и сравнить результаты. Первый программный конвейер считает имена нумерованных каталогов в каталоге **/proc**, а второй суммирует процессы, отображаемые утилитой **ps**. Результаты не должны сильно отличаться. Это поможет в случае подмены злоумышленником утилиты **ps**.

## 2.10. Терминальный режим и консольные атаки

UNIX-системы появились за десятилетие до создания корпорацией IBM первых массовых персональных компьютеров (модель IBM 5150 1981 года). ЭВМ 70-х годов, как правило, представляли собой громоздкое оборудование, размещенное в машинном зале, и много *терминалов*, расположенных на рабочих местах пользователей.

Изначально терминалы предназначались для передачи и отображения только текстовой информации, т. е. нажатие клавиши на клавиатуре приводило к формированию и передаче из терминала в ЭВМ кода символа, соответствующего нажатой клавише, а получение терминалом такого кода от ЭВМ приводило к отображению на экране монитора символа, соответствовавшего этому коду. В качестве такой информации выступали символы английского алфавита (строчные и прописные), знаки пунктуации и арабские цифры. По этой причине текстовые терминалы ещё называли алфавитно-цифровыми. Здесь необходимо добавить, что, помимо вышеперечисленных алфавитно-цифровых символов, необходимо было передавать и некоторые служебные символы, которые выполняли функции управления связью и форматирования текста. Всё это было использовано в терминалах в соответствии со стандартом ASCII (American Standard Code for Information Interchange — американский стандартный код для обмена информацией).

Терминал, который подключался непосредственно к процессорной стойке или плате, назывался *консолью*, и с него производилось управление ЭВМ. Консоль представляла собой автономное устройство, состоящее из

устройства отображения, устройства ввода и процессорного блока. Первые терминалы были телетайпами — дистанционно управляемыми пишущими машинками, поэтому текстовая консоль до настоящего времени обозначается **tty** (**teletype** – печать на расстоянии).

Терминалы обозначаются:

- **ttyS** – последовательные com-порты;
- **tty0-tty63** – 64 виртуальных терминала, с которыми работают программы–оболочки;
- **ttya0-ttyef** – 256 псевдотерминалов (с псевдотерминалами работают сетевые службы **sshd**, **telnetd** и т. п.);
- **vcN** – виртуальные консоли.

Классической реализацией текстового интерфейса, восходящей к первой половине XX века, является алфавитно-цифровое устройство ввода-вывода, например комплект из клавиатуры и АЦПУ (телетайпа). Впоследствии вместо АЦПУ стали применять мониторы, снабжённые знакогенератором, что позволило быстро и удобно организовывать диалог с пользователем. Такие комплекты из монитора и клавиатуры (иногда с добавлением мыши) называются консолью компьютера.

Настоящие текстовые терминалы сейчас очень редки, и производить подобные устройства экономически невыгодно. Подключить такие устройства к ЭВМ можно с помощью последовательных аппаратных интерфейсов. Но нередко консоли эмулируются персональными компьютерами с использованием аппаратуры, каналов и протоколов вычислительных сетей. Так, сетевой узел, с которого получен доступ к серверу, управляемому ОС Linux, по протоколу **telnet** или **ssh**, представляется серверной командной оболочке удалённым терминалом.

Видеопамять современных персональных компьютеров позволяет вместить в себя много текстовых экранов. Благодаря этому персональный компьютер с ОС Linux, имеющий один монитор и одну клавиатуру, может иметь множество консолей. Переключение между ними производится комбинацией клавиш **<Alt>-<Fn>**, где номер функциональной клавиши соответствует номеру активного терминала. Переключаясь между виртуальными терминалами, один человек может зарегистрироваться и работать в системе от имени нескольких пользователей и эксплуатировать в одиночку операционную систему в многопользовательском режиме.

В ПК, управляемом ОС UNIX, роль терминала, а точнее – консоли, выполняют клавиатура, контроллер клавиатуры, терминальный драйвер, видеоадаптер и монитор. Таким образом, за ввод информации отвечают первые три, а за вывод — последние три перечисленные компоненты. Таким образом, в персональном компьютере два различных физических устройства, подключенные своими интерфейсами одно к видеоконтроллеру, а второе – к контроллеру клавиатуры, объединяются в одно логическое устройство, и обращение к нему производится через один специальный файл и

общий драйвер.

Для переключения в текстовый режим из графического X–Window, не завершая при этом его работу, требуется нажать **<Ctrl>-<Alt>-<Fn>**. Впрочем, в X–Window пользователь может запустить приложение, которое эмулирует работу в режиме терминала (например, **xterm**). Ввиду наличия полос прокрутки они более удобны для отображения строк, не вмещающихся в стандартный текстовый терминал.

Удобство переключения между терминалами состоит в возможности практически одновременно работать в системе от имени разных субъектов. Например, администратор может использовать одну консоль для системных команд, требующих полномочий **root**, в другой консоли работать с документами от имени обычного пользователя, а в третьей – от имени третьего пользователя запустить длящийся вычислительный процесс. Не имея возможности справиться с «зависшим» или некорректно работающим процессом, пользователь переходит в другую консоль, регистрируется в ней, выводит с помощью команды **ps -ef** список процессов, определяет по номеру терминала и имени неуправляемого процесса его идентификатор **PID** и посылает планировщику задач команду **kill -9 PID** на завершение процесса.

Выполнение предусмотренных в учебном пособии лабораторных работ также основано на многоконсольном режиме. Управляя компьютером от имени администратора и нескольких пользователей, обучаемый может наблюдать процессы разграничения доступа к защищаемой информации. Исследуя файловую систему и работая с шестнадцатеричными числами, пользователь может переключаться в другую консоль, где у него для преобразования чисел в десятичный формат и производства вычислений запущен калькулятор **bc**.

Для того чтобы предоставить пользователям возможность регистрироваться в системе, первичный процесс **init** при загрузке системы на многопользовательском уровне 3 запускает несколько (по умолчанию – шесть) процессов **getty**, управляющих виртуальными консолями. Программа **getty** открывает последовательное устройство (текстовый терминал, виртуальный терминал или модем) и ожидает подключения. Программа выводит приглашение, а затем, после ввода имени пользователя, передает управление программе **login**. Существует много разновидностей **getty:** mgetty, mingetty, ugetty, agetty, gettyps, fbgetty и т. д. Процессы, обслуживающие виртуальные терминалы, можно увидеть в выводных данных команды **ps –ef** .

Если пользователь не смог успешно зарегистрироваться, программа регистрации через определенный промежуток времени завершается, закрывая открытую терминальную линию, а процесс **init** порождает для этой линии следующий getty-процесс, открывающий терминал вместо пре-

кратившего существование процесса. То есть всем определенным в **/etc/inittab** виртуальным консолям суждена «вечная» (до останова или перезагрузки системы) жизнь.

Для уменьшения числа виртуальных консолей достаточно удалить или закомментировать в файле **/etc/inittab** соответствующее число строк. А для увеличения – создать должное количество новых по образу и подобию существующих строк, изменяя только идентификатор и имя файла устройств:

```
c7:1235:respawn:/sbin/agetty 38400 tty7 linux
c8:1235:respawn:/sbin/agetty 38400 tty8 linux
```

и так далее, вплоть до

```
c63:1235:respawn:/sbin/agetty 38400 tty63 linux
```

В приведенных строках **respawn** означает запуск процесса **getty** в фоновом режиме, а когда процесс завершится (например, с ошибкой), его повторный запуск. Число 38400 (бит/сек) обозначает скорость передачи информации по псевдотерминальной линии. Число 63 – это максимально возможное количество виртуальных консолей. Обусловлено оно тем, что именно столько младших номеров резервировалось за устройствами этого класса. Переключение между консолями с такими номерами с помощью функциональных клавиш уже невозможно. К счастью, на такой чрезвычайный случай есть специальная, и очень простая, команда, не зависящая от количества функциональных клавиш:

**chvt N ,**

которая мгновенно перенесет нас в ту консоль, номер которой указан в качестве ее аргумента.

Не из всех виртуальных терминалов можно регистрироваться и, тем более, входить в систему с правами суперпользователя. В конфигурационном файле **/etc/securetty** определяются консоли, из которых разрешается аутентификация всех пользователей либо только администратора.

Управлять драйвером терминала можно с помощью особых комбинаций двух клавиш, первой из которых является **<Ctrl>**. Вот некоторые из этих комбинаций:

**<Ctrl>+<C>** – посылает сигнал завершения выполняющегося процесса, ассоциированного с текущим терминалом;
**<Ctrl>+<D>** – посылает признак конца файла (например, если файл создается из консоли с помощью команды **cat**) либо завершает сеанс;
**<Ctrl>+<S>** – останавливает вывод информации на экран;
**<Ctrl>+<Q>** –возобновляет вывод информации на экран.

Давайте посмотрим, что произойдет в случае чтения бинарного файла с помощью утилиты **cat** или **more**. Содержимое читаемого файла утилита выводит на терминал, а он ориентирован на вывод алфавитно-цифровой информации и символов псевдографики. Бинарный файл может содержать любые кодовые значения, и не все они соответствуют отображаемым символам. Терминальный драйвер, выводя содержимое бинарного файла на экран, «считает» его текстовым и соответствующим образом интерпретирует каждый байт. Поэтому возникнет ситуация, когда одни байты будут соответствовать отображаемым символам, другие — управляющим, третьи — вообще могут ничему не соответствовать.

В случае соответствия байтов двоичного файла отображаемым символам на экране монитора будет смесь из разных символов. Неотображаемые символы либо вообще не выводятся на экран, либо отображаются точками. Второй случай будет самым «неприятным» для драйвера терминала, и в большинстве случаев мы получим частично или полностью неработоспособную систему драйвер-видеоадаптер-монитор.

Для восстановления работы терминала рекомендуется «вслепую» набрать команду **reset** и завершить ее нажатием **<Enter>**, после чего необходимо выполнить сценарий **/etc/rc.d/rc.font**, чтобы восстановить настройки кодировки терминала. Если это не помогает, нужно комбинацией клавиш **<Ctrl>-<D>** завершить пользовательский сеанс и вновь зарегистрироваться либо открыть другую виртуальную консоль и повторить регистрацию в ней.

Один пользователь может поочередно работать из нескольких физических или виртуальных консолей. Но из одной консоли должен работать только один пользователь. Физических консолей с выполненным входом в систему и без пользователя существовать не должно. В ОС Linux пока не предусмотрено подтверждение полномочий пользователя в ходе сеанса (за исключением приложений, с которыми работает пользователь), поэтому нарушитель, занявший консоль вместо отсутствующего пользователя, воспринимается системой как ее законный обладатель. Покидая на время свое рабочее место, пользователь должен заблокировать консоль, чтобы нежданному гостю пришлось столкнуться с процедурой аутентификации. Особенно опасно оставлять консоль, в которой зарегистрировался администратор. Завершить сеанс в виртуальной консоли можно тремя способами: комбинацией клавиш **<Ctrl>-<D>**, командами **exit** и **logout**.

К сожалению, пользователи грешат тем, что, локально зарегистрировавшись в системе, они затем надолго оставляют без присмотра свой терминал, способствуя тем самым консольным атакам со стороны случайных посетителей. Либо, удаленно зарегистрировавшись в сети (проводной или беспроводной), они без завершения сеанса отключают свой ноутбук и уходят. Команды **w** и **who** (см. ниже) в таком случае могут показать наличие в системе «трудовых энтузиастов», которые работают уже много суток подряд.

Администратор вправе прекратить подобные безобразия, устанавливая переменную окружения оболочки ее внутренней командой

**export TMOUT=600**

В переменной задается число секунд, по истечении которых с момента последнего нажатия клавиши оболочка автоматически завершит пользовательский сеанс. Но данная команда будет действовать только в той консоли, из которой она была введена, и только на один сеанс. Пользователь, чей сеанс был принудительно завершен, после повторной регистрации вновь может оставлять свою консоль на неограниченный срок. На остальных пользователей эта команда совершенно не влияет. Вероятно, это не совсем то, что хотел бы добиться администратор.

Тайм-аут можно установить надолго, если записать эту команду отдельной строкой в файл **/etc/profile**. Установка тайм-аута на консольное бездействие будет действовать на всех пользователей, которые зарегистрировались позже. После перезапуска системы переменная окружения будет действовать на всех, включая администратора. Чтобы настройка во всех сеансах действовала только на выбранных пользователей, вышеуказанную строку следует записать в файлы **.bashrc** в их домашних каталогах. К сожалению, никто не мешает опытному и пренебрегающему безопасностью пользователю отредактировать *свой* файл **.bashrc** так, как он посчитает нужным.

В случае опасной пользовательской активности администратор может «заморозить» пользовательский ввод и вывод информации на произвольном локальном или сетевом терминале командой

**skill –STOP <tty>**,

где вместо **<tty>** указывается имя и номер терминала, например **tty2** или **pts/2**. Эта мера более эффективна, чем обычное принудительное завершение пользовательского сеанса, поскольку пользователь, «выброшенный» из системы, может повторно зарегистрироваться. Находясь перед заблокированным терминалом (если только это действительно терминал, а не самостоятельный компьютер), он ничего подобного сделать не сможет. Администратор же имеет право посылать сообщения на заблокированный терминал, и они будут исправно отображаться. Снять «блокаду» с терминала можно командой

**skill –CONT <tty>**

Чтение данных, введенных с клавиатуры другим пользователем, представляет собой угрозу конфиденциальности, а несанкционированный вывод данных на чужой экран является нарушением функциональности и может вызвать угрозы целостности или блокирования данных. Права доступа к консоли по умолчанию установливаются равными 620 = **rw--w----**. Владельцем консоли является пользователь, который зарегистрировался с нее. Он имеет право читать информацию с экрана и

вводить ее с клавиатуры. Члены терминальной группы пользователя (в файле **/etc/group** она поименована как **tty**) имеют право только вводить информацию. По умолчанию в эту группу включены все пользователи. Имя группы и права доступа к виртуальной консоли содержатся в конфигурационном файле **/etc/login.defs**.

Каждый пользователь имеет право запретить членам терминальной группы запись сообщений в свою консоль. Выполняется это намерение с помощью команды

**mesg n** ,

что равносильно установлению прав доступа к консоли с помощью другой команды

**chmod 600 /dev/ttyN**

Для того чтобы вновь разрешить запись в свой терминал, следует ввести строку

**mesg y**

На администратора, естественно, указанные запреты не действуют. С помощью утилиты **chmod** пользователь как бы имеет возможность выполнить не вполне рациональные действия, например запретить себе вводить данные с клавиатуры либо разрешить всем пользователям читать введенные им данные. Чтобы подобные действия не могли быть использованы для имитации неисправности, попытка блокировки собственной консоли системой игнорируется. Так, пользователь может ввести команду **chmod 000 /dev/ttyN**, после чего его консоль продолжает оставаться работоспособной.

**Tty** — это программа (команда), выводящая имя терминала, к которому подключен стандартный ввод **stdin**. Если запустить ее в текстовой консоли, будет выведена информация об используемом терминале:

```
$ tty
/dev/tty3
```

Запуск такой же команды в терминале X покажет иной результат:

```
$ tty
/dev/pts/2
```

С помощью команд **w** и **who** пользователь может определить, кто и с какого места работает в системе. В первую очередь он может узнать, не является ли его компьютер объектом удаленного доступа. Локальный терминал в системе называется **ttyN**, а доступ из сети с удаленного терминала обозначается **pts/N** или **ttypN**, где **N** – номер консоли.

На рис. 2.7 представлен результат выполнения команды **w**:

```
16:10:12 up 15 min,  4 users,  load average: 0,00, 0,00, 0,01

USER     TTY      FROM          LOGIN@   IDLE   JCPU   PCPU WHAT
root     tty1     -             15:55    1.00s  0.02s  0.00s  w
ivanov   tty2     -             16:07    3:08   0.01s  0.01s -sh
petrov   tty3     -             16:07    2:54   0.00s  0.00s -bash
john     pts/0    192.168.0.1   16.23    4:12   0.02s  0.00s –sh
```

Рис. 2.7. Информация о пользователях, выведенная командой **w**

Информация выведена в табличном виде и разделена по столбцам: **USER** – пользователь, **TTY** – терминал, **FROM** – имя домена или числовой IP-адрес удаленного хоста, **LOGIN@** – локальное время начала соединения, **IDLE** – период времени с момента выполнения последнего процесса, **JCPU** – время, использованное всеми процессами, обслуживающими данный терминал, **PCPU** – время процессора, использованное текущим процессом, указанным в последнем столбце, **WHAT** – последний из процессов, запущенных из данной консоли. Из листинга видно, что в системе работают четыре пользователя – три из виртуальных терминалов, а четвертый получил терминальный доступ с сетевого узла **192.168.0.1**.

Намного более краткая информация о пользователях и используемых терминалах выводится командой **who** (рис. 2.8). Пояснений здесь не требуется.

```
root     tty1         2008-11-05 15:55
ivanov   tty2         2008-11-05 16:07
petrov   tty3         2008-11-05 16:07
john     pts/0        2008-11-05 16.23
```

Рис. 2.8. Информация о пользователях, выведенная командой **who**

Команды **w** и **who**, отображающие сеансы пользователей, работающих в системе, берут информацию из журналов аудита (например, из файла **/var/log/utmp**). Поэтому эти команды неверно отображают информацию о пользователе, если он после регистрации командой **su** сменил свой имидж. Если пользователь в этой консоли «трансформировался» в администратора или администратор работает под именем обычного пользователя, команды **w** и **who** не покажут существующего статуса пользователей.

Утилиты, предназначенные для межконсольного обмена, называются **write** и **wall**. Они принадлежат владельцу **root** и группе **tty**. Права доступа к этим утилитам обозначены битовой строкой **r-xr-sr-x**. Установка эффективного права **SGID** предусмотрена для того, чтобы все поль-

зователи могли посылать и читать сообщения с правами группы **tty**.

С помощью утилиты **write** пользователи могут посылать сообщения другим пользователям с указанием конкретного имени и/или терминала, при условии, что запись в него разрешена. Ее синтаксис весьма прост:

```
write <user_name> [tty_name]
```

После ввода команды последует перевод строки, в которую необходимо вводить текст сообщения. Если сообщение длинное, завершать строку и переходить к новой строке следует с помощью клавиши **<Enter>**. Корреспондент будет получать сообщение построчно. Завершить сообщение следует комбинацией клавиш **<Ctrl>+<D>**. Но если подобным образом попытается остановить вывод ненужного сообщения получатель, его ждет завершение сеанса с необходимостью повторной регистрации.

Утилита **wall** служит для широковещательного оповещения и может использоваться для передачи во все доступные на запись консоли срочных сообщений. Она вводится без каких–либо аргументов. Запись и окончание сообщения производится по аналогии с командой **write**.

Для того чтобы предотвратить анонимную и безответственную передачу сообщений с помощью указанных команд, вывод информации сопровождается сведениями о том, откуда (номер терминала) и когда сообщение было отправлено.

Администратор, впрочем, может обойтись без таких команд, напрямую адресуясь к специальным файлам устройств:

```
echo "не отвлекайтесь от работы !" > /dev/tty5
```

## 2.11. Сокрытие процессов

Цель мониторинга за процессами заключается в выявлении исполняемых программ, которые потребляют несоразмерно много ресурсов, а также программ, которые были запущены внешним или внутренним нарушителем для атаки компьютерной системы. Подозрительными и заслуживающими внимания администратора следует считать процессы:

- запущенные от имени **root** с именами, которые обычно не отображаются в списке задач. Выявление таких процессов затруднено вследствие того, что они обычно не отображаются в списке задач;
- запущенные пользователями потенциально опасные системные утилиты с установленным битом SUID. Правда, в последнем случае, как будет показано ниже, владельцем процессов является суперпользователь **root**;
- имеющие странные имена, в том числе включающие в себя необычные символы. Это может быть следствием программной атаки на переполнение буфера.

Упомянув программные средства контроля над процессами, нельзя обойти вниманием способы, которые могут применить злоумышленники для их сокрытия. Нарушитель при запуске вредоносной программы закономерно попытается «спрятать» ее процесс от глаз администратора и других внимательных пользователей. Нелояльные пользователи, которые пытаются незаконно повысить свои полномочия, также могут скрытно от администратора запускать ответственные утилиты **su**, **sudo**, **passwd**, **mount**, **chmod** и др. Никто, впрочем, не запрещает использовать эти утилиты явно и по их прямому назначению. Разумеется, скрывать от визуального контроля имеет смысл только те процессы, которые выполняются достаточно долго. О нейтрализации программных средств аудита, которые, в частности, фиксируют мгновенные события, речь пойдет отдельно.

Возможности нарушителя определяются многими факторами, среди которых наиболее значимыми являются его права доступа, квалификация и программный инструментарий. Для того чтобы полностью скрыть или сделать неприметной отображаемую информацию, нарушитель может попытаться:

- изменить параметры процесса, по которым его можно отличить от других и идентифицировать;
- осуществить подмену самих утилит **ps** и **top**, отображающих информацию о процессах;
- изменить содержимое виртуальных файлов в директории **/proc/PID**, соответствующей номеру скрываемого процесса;
- произвести вмешательство в «ядерные» процессы в целях перехвата и замены отображаемой информации о процессах.

Обладая правами администратора, нарушитель может:

- внедрить программный модуль;
- обеспечить автоматический или периодический запуск системного процесса или службы;
- запустить процесс из командной строки и использовать методы его сокрытия на уровне ядра.

Так называемые «ядерные» методы сокрытия процессов основаны на соображениях, изложенных в [11]. Виртуальная файловая система **/proc** содержит по одному «числовому» каталогу на каждый выполняющийся процесс. Если скрыть такой каталог, то процесс с соответствующим идентификатором PID не будет отображаться утилитами **ps** и **top**.

Для сокрытия каталогов и процессов предлагается одинаковый подход. Он заключается в том, что каждый файл (и каталог) представлен в ядре структурой **file**, одно из полей которой является указателем на другую структуру **file_operations**, определяющей допустимые действия с файлами. Если заблокировать обращение к конкретной функции, то она перестанет вызываться. Например, для сокрытия каталога следует заблокиро-

вать вызов функции **readdir()**. Алгоритмические и программные особенности для программистов можно почерпнуть в [8, 10, 11].

Ядро ОС Linux не является монолитным, а использует модульную архитектуру с механизмом загружаемых модулей (LKM – Loadable Kernel Modules). Модульные ядра, как правило, не требуют полной перекомпиляции при изменении состава аппаратного обеспечения компьютера. Это позволяет один раз собрать компактное ядро и, по мере необходимости, добавлять или удалять из него требуемый функционал. Модулями ядра могут быть и драйверы аппаратных устройств компьютера. После загрузки модуля он становится частью ядра.

Модули могут быть загружены с помощью команды **modprobe** или **insmod**. Как правило, команда **modprobe** более предпочтительна, потому что она загружает все вторичные модули, от которых зависит основной загружаемый модуль. Просмотреть список загруженных модулей можно командой **lsmod** (рис. 2.9).

```
# lsmod
Module                  Size        Used by      Not tainted
usb-uhci                21676       0            (unused)
usbcore                 58464       1            [usb-uhci]
af_packet               12392       1            (autoclean)
pcnet32                 15140       1            (autoclean)
mii                      2544       0            (autoclean)
[pcnet32]
crc32                    2880       0            (autoclean)
[pcnet32]
floppy                  48568       0            (autoclean)
subfs                    4296       4            (autoclean)
ac                       1792       0
rtc                      6236       0            (autoclean)
ext3                    62288       2
jbd                     37852       2            [ext3]
```

Рис. 2.9. Получение списка загруженных модулей

В выводе команды присутствует наименование модуля, его размер, счетчик использования и имена модулей, с которыми он связан. Чтобы выгрузить модуль, используется команда **rmmod.**

Внедрение в ядро Linux динамически загружаемых программных модулей позволяет злоумышленнику управлять операционной системой на «ядерном» уровне. Эти возможности позволяют, в том числе, скрывать файлы и процессы.

Нарушитель, действующий с правами обычного пользователя, не располагает правами, позволяющими ему полностью скрыть от наблюдателя нежелательный процесс. Одних только прав **root** для этого также будет

маловато, нужны еще инструменты и квалификация. Но попытаться изменить уличающие его параметры процесса пользователь вполне может. Такими параметрами в первую очередь являются имя программы, обозначение (имя и номер) терминала, с которого процесс запущен, а также UID или регистрационное имя пользователя. Рассмотрим все перечисленное по порядку.

Наивно полагать, что нарушитель назовет исполняемый файл своей программы каким-нибудь зловещим именем вроде **virus_hiden**, **trojan666** или **agent007**. Для маскировки, вероятно, будут использованы имена программ, которые пользователь запускает наиболее часто. Большинство команд выполняется практически мгновенно, и их имена для камуфляжа непригодны. Так, наиболее часто используемая команда **ls** практически никогда не отображается в списке процессов, и если такое произойдет, то это скорее вызовет настороженность администратора. Маскировать опасные продолжительные процессы можно с помощью таких безобидных названий программ, как **mc**, **man**, **info**, **bc**. Даже очень подозрительному администратору трудно усмотреть угрозу в использовании файлового менеджера, справочной системы или калькулятора.

Системные утилиты, запускаемые оболочкой по «короткому» имени, найденному с помощью переменной окружения **PATH**, отображаются в списке процессов также только по имени файла. Но обычный пользователь, лишенный права записи в каталоги типа **/bin** и **/sbin**, сумеет запустить «двойника» такой программы только из своего домашнего каталога или каталога **/tmp** с указанием полного имени файла.

Довольно распространен способ сокрытия, основанный на символьном камуфляже. Он заключается в том, что создается процесс, по имени напоминающий привычную для глаза программу. Для этого используются похожие по начертанию символы в различных раскладках клавиатуры. Так, строчные символы **a**, **c**, **e**, **o**, **p** и заглавные символы **A**, **B**, **C**, **E**, **K**, **O**, **P**, **T**, **M**, **X** пишутся одинаково в русской и латинской кодировках.

Модифицируя исходный файл программы, используемый для проникновения в систему или перехвата данных, злоумышленник может заменить в нем аргументы командной строки, например, с помощью инструкции **strcpy(argv[0], «man»)**. После этого запущенный процесс будет отображаться как экземпляр справочной системы.

Особо следует сказать о маскировке процессов, запускаемых из файлов-сценариев. Например, нарушитель создал атаку на переполнение ресурса дисковой памяти и написал для этого небольшой сценарий:

```
cat >/tmp/...
#! /bin/bash
yes abc > /tmp/abc
Ctrl+d
```

Присваивая файлу имя, состоящее из трех точек, нарушитель желает

сделать его скрытым и неотображаемым при вводе команды **ls -l**. Затем, используя команду **chmod 700 ...** , он обеспечивает себе права на чтение и исполнение данного сценария, после чего запускает его, ожидая, что такой процесс будет «закамуфлирован». Но администратор, контролируя систему с помощью утилиты **ps -ef**, в это время может наблюдать два процесса, запущенные из пользовательской консоли и от его имени: **/bin/bash ...** и **yes**. Вероятно, это не совсем то, на что рассчитывал предприимчивый пользователь. Причем даже если наблюдение за процессами не было организовано и файл непомерной величины был создан, то установить его создателя и владельца будет нетрудно.

Скрытие имени процесса – наиболее простая задача. Обычному пользователю труднее скрыть от системных мониторов факт запуска программы с конкретного терминала и от имени определенного пользователя. Однако один из этих параметров закамуфлировать можно.

Самый простой путь скрытия терминала – преобразование интерактивного процесса в фоновый. Для этого командную строку следует закончить пробелом и символом **&**. Затем командой **exit** или **logout** следует завершить пользовательский сеанс, после чего вновь зарегистрироваться. Команда **ps -ef** вместо имени терминала отобразит вопросительный знак.

Команда **nohup <command>** позволяет процессу продолжить выполнение при потере управляющего терминала. Эту команду выгодно использовать при выполнении команды продолжительного действия. Команда запускается, после чего терминальный сеанс завершается, а программа при этом продолжает выполняться.

Что характерно: в фоновом режиме запускаются даже такие явно интерактивные команды, как **passwd** и **su**, требующие ввода пароля. Однако попытка завершить после этого пользовательский сеанс окончится неудачно: система предложит вначале завершить фоновый процесс.

Если пользователь вводит команду **passwd** или **su** и не торопится с вводом пароля, его процесс может быть зафиксирован из другой консоли с помощью команды **ps -ef**. Но владельцем процесса будет значиться не пользователь, а администратор. По существу дела так оно и есть – ведь эти утилиты запускаются с правами их владельца – **root**. Эта особенность может быть использована в целях сокрытия от администратора длительных попыток подбора пароля **root** с помощью утилиты **su.** Например, для этого пользователь создает жесткую ссылку на утилиту **su** из своего каталога, используя команду

```
ln /bin/su man
```

Создать в своем каталоге полноценную утилиту **su** пользователь не сможет – он не обладает правом ее чтения, а следовательно, не сможет ее скопировать. Даже если допустить, что он ее скопирует, то ценность копии будет невелика, поскольку при копировании у файла сбрасывается бит

эффективных прав доступа. Но создать жесткую ссылку на недоступный файл удается без проблем.

Хитрость здесь заключается еще и в том, что второй символ в имени **man** вводится в русскоязычной раскладке. В противном случае при вводе команды **man** было бы запущено настоящее справочное руководство, что в планы пользователя вовсе не входит.

После этого из другой консоли командой **ps -ef** можно наблюдать, что процесс **man** был запущен пользователем **root**. Продолжает демаскировать ложный процесс только имя консоли, из которой он запущен.

Еще один очевидный демаскирующий признак, отображаемый утилитами **ps** и **top**, – имя пользователя, запустившего процесс. Как уже отмечалось выше, запуск пользователем утилиты **passwd** или **su** «приписывается» владельцу этих файлов **root**. Файлы с установленным битом **SUID**, принадлежащие администратору, находятся под его контролем и, как правило, не могут быть использованы для деструктивных действий. Но пользователь может попробовать написать простой командный файл и присвоить ему эффективное право запуска от имени другого пользователя. Пользователь ожидает, что в списке процессов такая команда отобразится как запущенная другим лицом. Проверим, так ли это.

Зарегистрируемся в двух консолях с правами обычных пользователей (допустим, их имена **john** и **braun**). Пользователь **braun** создает в каталоге **/tmp** командный файл с именем **no** следующего содержания:

```
#! /bin/bash
yes 12345 > /dev/null &
```

Затем он командой **chmod** присваивает файлу **/tmp/no** права доступа 4777. Пробный запуск утилиты ее владельцем **braun** показывает, что она работоспособна. Аналогично пробуем запустить утилиту **/tmp/no** от имени пользователя **john**. После этого с правами **root** наблюдаем за процессами и видим, что замысел не удался. В колонке **USER** отображается пользователь **john**, и имя программы фиксируется не **no**, а **yes**. Следовательно, утилиты для отображения процессов не подвержены таким простым способам обмана.

## 2.12. Аудит событий и его безопасность

Следователю и эксперту вряд ли стоит рассчитывать на то, что пользователь, использующий компьютерную систему в своей противоправной деятельности, станет протоколировать свои действия. Наоборот, следует ожидать, что он постарается избавиться от уличающих его сведений. Однако система по умолчанию сама фиксирует многие важные события, которые сохраняются в так называемых системных журналах. Далеко не все

администраторы в достаточной степени осведомлены о возможностях протоколирования событий, пользователь системы часто о таких возможностях собственного компьютера может и не подозревать.

Интерес для администратора могут представлять файлы протоколов, а также файлы, которые создаются прикладными программами в процессе своего функционирования, но не являются объектом работы пользователя или протоколом работы прикладной программы. Файлы аудита могут иметь расширение **.log** , а системные файлы аудита обычно расширения не имеют.

Основные файлы системных протоколов (журналы) ОС Linux находятся в каталоге **/var/log**. Информация о функционировании системы и работе пользователей записывается системными программами в определённые файлы этого каталога. Большей частью журналы представляют собой текстовые файлы, в которые информация записывается построчно и последовательно. Некоторые файлы аудита являются двоичными и имеют специальную структуру, что требует для их просмотра использования специальных утилит, интерпретирующих содержимое этих файлов в удобный для просмотра вид.

Каталог **/var/log** обычно содержит следующие основные файлы аудита:

- **cron** – текстовый файл, содержащий информацию о фактах выполнения пользователями периодических заданий;
- **debug** – текстовый файл, содержащий отладочную информацию ядра системы и некоторых системных или прикладных программ;
- **faillog** – двоичный файл, содержащий информацию о неудачных попытках входа в систему (утилита для работы с этим файлом также называется **faillog**);
- **lastlog** – двоичный файл, содержащий информацию о последних входах в систему (утилита для работы с этим файлом – **last**);
- **maillog** – текстовый файл, содержащий информацию о работе почтовой системы **sendmail+procmail**;
- **messages** – текстовый файл, содержащий протоколируемую информацию о системе и процессах категории выше **info** и ниже **warn** (об уровнях значимости см. ниже);
- **syslog** – текстовый файл, содержащий протоколируемую информацию о системе и процессах категории **warn**;
- **secure** – текстовый файл, содержащий информацию о попытках получения и использования пользователями полномочий суперпользователя, о смене пароля, о регистрации и удалении пользователей и др.;
- **wtmp** – двоичный файл, содержащий информацию о всех регистрациях пользователей в системе (утилита для работы с этим файлом – **last**).

Наблюдение за событиями и их протоколирование производится с помощью системы регистрации **syslog**. Она состоит из трех компонентов:

- демона **syslogd**, который собственно отвечает за регистрацию событий,
- пользовательской программы **logger**,
- библиотечных функций **openlog()**, **syslog()** и **closelog()**, которые обслуживают службу регистрации сообщений.

Детали, связанные с использованием программы **logger** и вышеназванных библиотечных функций, с достаточной полнотой изложены в [1, 10].

Демон **syslogd** запускается автоматически процессом **init** из загрузочных сценариев и работает непрерывно до останова системы. Программ, обеспечивающих информацией службу регистрации, довольно много. К ним относятся такие известные программы, как **su**, **sudo**, **getty**, **passwd**, **inetd**, **crond**, **login**, **halt** и др. Источником сообщений ядра, кроме того, является файл специального устройства **/dev/klog** или **/dev/log**. Для отправки сообщения в файл протокола программа использует библиотечную функцию **syslog()** языка Си, а в сценариях, написанных на языке интерпретатора команд, используется утилита **logger**. Демон **syslogd** читает эти сообщения, фильтрует их и помещает в журнальные файлы. Фильтрующие параметры содержатся в текстовом конфигурационном файле **/etc/syslog.conf**.

Файл **/etc/syslog.conf** отличается простым форматом и состоит из строк, включающих два поля, разделенные одним или несколькими пробелами или знаками табуляции:

**селектор действие**

В поле **селектор** указывается **средство** (программа, служащая источником сообщений, или потребитель этих сообщений) и **уровень** значимости этих сообщений:

**селектор=средство.уровень**

Селекторы могут содержать символ * (все) и ключевое слово **none** (ничего). Существует более 20 допустимых имен **средств**, из которых наиболее часто встречаются:

| | |
|---|---|
| **kern** | ядро; |
| **user** | пользовательские процессы; |
| **mail** | почтовые программы; |
| **cron** | планировщик задач; |
| **mark** | периодические временные метки; |
| **auth** | процессы, обеспечивающие авторизацию и безопасность; |

**daemon** демоны;
**syslog** внутренние сообщения демона **syslogd**.

Предусмотрено 8 **уровней** значимости сообщений (в порядке убывания значимости):

**emerg** экстренные ситуации;
**alert** срочные ситуации;
**crit** критические состояния;
**err** ошибочные состояния;
**warning** предупреждающие сообщения;
**notice** необычные сообщения;
**info** обычная информация;
**debug** отладочная информация.

Поле **селектор** может содержать несколько записей, отделенных точкой с запятой, в которых **средство** может перечисляться через запятую.

В файле **syslogd.conf** уровень сообщения определяет его *минимальную* важность для того, чтобы быть зарегистрированным. Так, селектор **cron.warning** означает, что будут регистрироваться сообщения демона **crond** с уровнями **warning**, **err**, **crit**, **alert** и **emerg**.

Поле **действие** означает, как следует поступить с сообщением. В этом поле чаще всего указывается имя журнального файла, в который надлежит записать сообщение. Но также можно указать доменное имя или IP-адрес компьютера, в который будет переправлено сообщение, а также имя зарегистрированного пользователя, на консоль которого следует вывести информацию. Символ * предписывает вывод сообщения на экраны всех пользователей. При этом выполнение нескольких действий в одной строке не предусмотрено. Если требуется записать сообщение в файл и одновременно отобразить его в консоли пользователя, необходимо использовать две разные строки.

Необходимо отметить, что файлы, указанные в строках **/etc/syslog.conf**, заполняются текстовой информацией. В то же время присутствуют и двоичные регистрационные файлы, в частности указанный выше **/var/log/wtmp**, который непосредственно заполняется программами, ответственными за регистрацию пользователей в системе.

Для администратора важно не только собрать информацию о событиях безопасности, но и вовремя увидеть необходимое. Особым разнообразием записей отличается журнал **/var/log/messages**. Для облегчения визуального анализа журнала следует использовать возможность цветового форматирования строк [13].

Выборка из содержимого файла **/var/log/secure**, в который демон **syslogd** записывал сообщения, генерируемые программами **login**,

**su**, **passwd**, **useradd** и **userdel**, приведена на рис. 2.10. Формат и содержание записей в дополнительных пояснениях не нуждаются.

```
Jan 14 11:26:20 server login[2987]: ROOT LOGIN  on `tty1'
Jan 14 12:02:23 server useradd[7244]: new user: name=user, uid=1000, gid=100,
home=/home/user, shell=/bin/bash
Jan 14 12:02:24 server chfn[7245]: changed user `user' information
Jan 14 12:02:29 server passwd[7246]: password for `user' changed by `root'
Jan 14 17:15:16 server userdel[1193]: delete user `user'
Feb 17 18:53:15 samsung login[3877]: invalid password for `root'  on `tty6'
Feb 17 18:53:22 samsung login[3877]: ROOT LOGIN  on `tty6'
Feb 26 20:19:17 samsung useradd[6886]: new user: name=petrov, uid=1000,
gid=100, home=/home/petrov, shell=
Feb 26 20:19:32 samsung useradd[6896]: new user: name=ivanov, uid=1001,
gid=100, home=/home/ivanov, shell=
Feb 26 20:20:11 samsung passwd[6918]: password for `ivanov' changed by `root'
Feb 26 20:20:36 samsung passwd[6937]: password for `petrov' changed by `root'
Feb 26 20:20:50 samsung su[6959]: + pts/2 root-petrov
Feb 26 20:21:46 samsung su[7000]: + pts/2 petrov-ivanov
Mar 22 05:48:47 samsung login[4158]: ROOT LOGIN  on `tty1'
Mar 22 06:51:40 samsung su[7323]: + pts/1 root-ivanov
Mar 22 07:17:24 samsung su[8498]: + pts/1 root-ivanov
Mar 22 07:37:08 samsung passwd[8946]: password for `ivanov' changed by `ivanov'
Mar 22 07:54:14 samsung passwd[10205]: password for `ivanov' changed by `root'
Mar 22 08:13:06 samsung su[11070]: + pts/1 ivanov-petrov
```

Рис. 2.10. Выборка из содержимого файла **/var/log/secure**

Система обеспечения безопасности в первую очередь зависит от неуязвимости самой службы безопасности. В известной прибаутке говорится: «Не спит собака, дачу охраняет, и я не сплю – собаку стерегу». Эти соображения полностью относятся к системе аудита.

Система протоколирования событий в достаточной мере защищена от неправомерных действий нелояльных пользователей. Завершить или приостановить процесс регистрации событий они не могут, равно как и удалить/модифицировать информацию из журнальных файлов. Но они в состоянии создать фальшивые тревожные сигналы, которые будут зарегистрированы.

Кстати, любопытно, как система осуществляет аудит событий при запуске переименованных утилит типа **passwd** или **su**. Если пользователь создаст из своего каталога или каталога **/tmp** символическую ссылку на утилиту **su**, а затем запустит ее в целях подбора пароля администратора, то в списке процессов команда отобразится по имени созданной ссылки и как запущенная от имени **root**. Но система аудита нормально зафиксирует и запишет в журнальный файл запуск пользователем утилиты **su** под ее настоящим именем.

Необходимо остановиться на способах, применяемых нарушителями с правами **root** для нейтрализации системы аудита. Для этого могут использоваться нижеследующие операции.

1. Удаление определенных журнальных файлов, в которых

протоколируются факты проникновения в систему, повышение прав доступа, создание новых учетных записей и др. Удаление файлов для пользователя, имеющего права на поиск и запись в каталог **/var/log** , никаких проблем не представляет. Ликвидация бинарных файлов **wtmp**, **wtmp**, **lastlog**, **faillog** приводит к тому, что регистрация соответствующих событий будет прекращена, а удаленные файлы вновь не появятся. Подобное грубое вмешательство заведомо привлечет внимание администратора и иных пользователей хотя бы по той причине, что станет невозможно пользоваться некоторыми командами, такими как **who**, **w**, **last**.

2. Завершение работы демона **syslogd**. После этого дальнейшее пополнение регистрационных записей прекратится. Демаскирующий признак – отсутствие **syslogd** в списке процессов. Нарушителю имеет смысл завершать этот процесс до выполнения деструктивных действий, хотя само проникновение в систему и приобретение администраторских прав будет зафиксировано.
3. Нейтрализация работы демона **syslogd** с оставлением его в списке процессов может происходить путем его перезапуска с направлением вывода в нулевое устройство **/dev/null**.
4. Установка и запуск «исправленного» демона **syslogd**, который не будет протоколировать демаскирующие события.

Программы, используемые злоумышленниками для очистки журналов аудита, получили название **log cleaner** или **log wiper**. Они служат для избирательного удаления информации, свидетельствующей о незаконных действиях в системе. Стирание или замена строки, демаскирующей нарушителя, в текстовом log-файле программных затруднений не вызывает (если нарушитель имеет права администратора). Но такие файлы аудита, как **utmp**, **wtmp** и **lastlog**, имеют двоичный вид, и произвести в них удаление или замену записей непросто.

# 3. РАБОТА С ОБЪЕКТАМИ ФАЙЛОВОЙ СИСТЕМЫ

В файловых системах ОС Linux основным логическим объектом является файл. Все объекты, включая устройства ввода/вывода информации и каналы межпроцессного взаимодействия, называются файлами. Определено семь функциональных типов файлов:

- обычные файлы;
- каталоги;
- символические ссылки;
- именованные каналы;
- сокеты;
- файлы символических устройств;
- файлы блочных устройств.

Внутренняя структура обычного файла для операционной системы совершенно безразлична, и файл воспринимается ею как простая последовательность байтов. Для операционной системы Linux не имеют значения расширения имён файлов, по которым можно судить об их типе. Расширение имени файла в ОС UNIX, в том числе и Linux, не предусмотрено по причине того, что символ «**.**» входит в состав имени файла наравне с другими символами. Точнее, пользователи могут присваивать файлам имена, содержащие точки и символы, напоминающие характерные для ОС семейства Windows расширения, но только для своего удобства.

Таким образом, внутренняя структура файла в ОС Linux целиком зависит от пользовательской программы. Исключение составляют собственно исполняемые файлы формата ELF (Executable and Linking Format), так как они непосредственно исполняются центральным процессором и запускаются при наличии установленного признака исполняемости. Они имеют по нулевому смещению от начала стандартную четырехбайтную сигнатуру, которая называется «магическим числом» файла.

В остальных случаях файл с установленным признаком исполняемости считается текстовым, содержащим строки с командами оболочки. Если первой строкой такого текстового файла является строка вида **#!/bin/sh** (допускается наличие пробелов после «**!**»), то первые два символа являются «магической комбинацией», а **/bin/sh** есть программа, которая будет запущена с передачей ей всех последующих строк файла. Если в первой строке отсутствует вышеописанное, то строки файла последовательно обрабатывает оболочка, обслуживающая данный терминал. Установленный признак исполняемости является необходимым, но не достаточным условием. Любому файлу можно установить сигнатуру исполняемости, но результат его запуска на исполнение будет зависеть от его содержимого.

Одна из утилит ОС Linux, именуемая **file**, умеет различать довольно много разновидностей файлов по их «магическим числам» и некоторым иным признакам внутреннего формата.

### 3.1. Действия над обычными файлами

Ранее уже рассматривались способы создания обычных файлов. Для копирования файлов предназначается команда **cp** (copy). Это универсальная команда, с помощью которой можно выполнить несколько действий:

- создание копии файла с другим именем в том же каталоге

**cp -arg file1 file2**;

- копирование файла с прежним именем в другой каталог

**cp -arg file1 <dir>**;

- копирование файлов каталога **<dir1>** в каталог **<dir2>**

**cp -arg <dir1> <dir2>**.

В качестве наиболее часто используемых аргументов задаются:

**-i** – при наличии в месте назначения файла с таким именем будет выдан запрос на его переписывание;

**-f** – при наличии в месте назначения файла с таким именем он переписывается без запроса;

**-p** – сохраняется режим доступа к скопированному файлу, его владелец, группа владельца и временные отметки (без этого параметра файл переходит в собственность копирующего, права доступа устанавливаются согласно маске доступа, а временные отметки обновляются);

**-R** – выполняется рекурсивное копирование с учетом всех вложенных файлов и подкаталогов;

**-a** – аналог комбинации **-pR** с дополнительным копированием символических ссылок, что позволяет создать точную копию каталога.

Для копирования файла необходимо иметь право его чтения. Нужен еще доступ в два каталога – тот, где находится исходный файл, и тот, куда надлежит поместить его копию. Для копирования файла необходимо иметь права чтения и поиска в каталоге, откуда происходит копирование. Скопировать файл можно только в тот каталог либо на то устройство, на которое имеется право записи и поиска.

При необходимости копирования всех файлов из каталога задается маска с использованием символов-звёздочек. Команда

**cp /home/* /mnt/abcd**

производит копирование всех файлов из домашнего каталога в примонтированный каталог **abcd**.

Файлы создаются, изменяются и удаляются в файловой системе в соответствии с правилами этой системы, поэтому при копировании файла из одной файловой системы в другую неизменными остаются сами данные, а метаданные файла и всё, что с ними связано, будет зависеть от типа файловой системы, в которую копируется файл.

Логическое удаление файлов и каталогов обеспечивается утилитой

**rm -arg <file_name> <dir>**

В качестве аргументов можно указать:
**-f** – для безусловного (без дополнительных запросов и подтверждений) удаления файла. При обычном удалении файла система выводит запрос на удаление, который необходимо подтвердить символом **y** (yes) и **Enter**,
**-d** – для удаления непустого каталога,
**-r** – для рекурсивного удаления внутренних каталогов.

Удаление пустого каталога поддерживается командой

**rmdir <dir>**

Для гарантированного удаления файла с многократным (до 25 раз) стиранием **inode** и блоков данных псевдослучайными комбинациями в большинстве версий Linux имеется утилита

**shred –arg <file_name>**

Используемые командой стирания аргументы:
**-v** – показывать ход стирания,
**-n** раз – число повторов (25 раз по умолчанию),
**-s** – очистить N байт,
**-x** – не округлять размеры файлов до следующего целого блока,
**-u** – обрезать и удалять файл после перезаписи.

Перемещение файла – это комбинация двух команд: копирования файла в другой каталог и удаления исходного файла, но только в случае перемещения файла из одной файловой системы в другую. Если же файл перемещается в пределах одной файловой системы, то перемещения данных не происходит, а все изменения касаются только его метаданных.

Перемещение указанного файла в другой каталог производится командой

**mv –arg <file_name> <dir>**

В ОС Linux команда переименования файлов отсутствует как таковая, поскольку команда перемещения **mv** превосходно справляется с изменением имени файла

**mv file1 file2**

### 3.2. Работа со специальными файлами устройств

Многозадачные операционные системы не позволяют прикладным программам напрямую работать с аппаратными компонентами. Если нескольким программам в одно и то же время заблагорассудится вывести

данные на один и тот же принтер или монитор, последствия могут быть непредсказуемы. Поэтому в защищенном режиме центрального процессора инструкции, позволяющие непосредственно обращаться к портам ввода/вывода, являются привилегированными. Взаимодействие программ с устройствами происходит через ядро операционной системы посредством программных модулей, именуемых драйверами.

Пользовательский процесс не может непосредственно работать с драйверами. Однако в системе предусмотрены файлы специального типа, через которые можно читать данные из устройства или записывать их на него, как в обычные файлы, не обращая внимания на конкретную аппаратную реализацию устройств. Благодаря файлам устройств можно работать с дисковой и оперативной памятью, принтерами, консолью и другими устройствами в режиме чтения и записи, как с обычными файлами. Таким образом, в ОС Linux реализована единая идеология доступа к данным независимо от их физического размещения, источника формирования и вывода.

Различают символьные и блочные устройства, аналогично называются и соответствующие им специальные файлы. При выводе информации о файлах с помощью команды **`ls -l`** файлы символьных устройств можно распознать по первой букве «**`c`**», а блочные – по букве «**`b`**». К символьным относятся устройства, осуществляющие ввод/вывод данных в виде последовательного или посимвольного потока байтов (ленточные накопители, монитор, клавиатура, звуковые адаптеры, последовательные или параллельные порты). Блочные устройства осуществляют ввод/вывод фиксированными блоками данных определённой длины. Примером блочного устройства является магнитный или оптический диск – информация записывается на них и считывается блоками фиксированного размера, кратными размеру сектора. Подобная блочная структура организована и на долговременной полупроводниковой памяти.

Файлы специальных устройств (далее также – специальные файлы, файлы устройств) внешне напоминают обычные файлы. Их можно создавать (но только по определённым правилам специальной командой **`mknod`**), перемещать из каталога в каталог и удалять. Данные, помещаемые в такой файл, передаются драйверу устройства, а читаемые из файла – запрашиваются у драйвера. Копирование специального файла приведет к чтению из соответствующего устройства.

На специальные файлы не задаются права исполнения. Чтение из специального файла означает вывод потока данных из устройства. Так, с помощью команды

```
cat /dev/fd0 | xxd | more
```

производится чтение содержимого дискеты с поэкранным выводом. Утилита **`more`** одновременно отображает на текстовый экран несколько сотен символов (например, в режиме 25 строк х 80 символов = 2000), а емкость неименованного канала составляет 4 Кб. Поэтому чтение дискеты будет

производиться порциями. Рассматриваемая ниже утилита блочного копирования **dd** также умеет читать из файла устройства, причем она корректно работает с любыми байтами.

Запись в специальный файл означает вывод потока данных в устройство. Так, команда

```
cal 2009 > /dev/fd0
```

запишет календарь указанного года в начальные сектора дискеты, и при этом будет уничтожена хранимая там служебная информация. С помощью команды

```
echo privet! | dd of=/dev/fd0 bs=1 count=7
```

слово **privet!** будет записано в начало первого сектора дискеты. Таким путем можно превращать машинные носители в стегоконтейнеры, для чтения которых потребуется дисковый редактор или утилита **xxd.**

Распечатка файла на принтере также требует наличия права на запись в файл устройства **/dev/lp0** (в данном случае предполагается, что принтер подключен к первому параллельному порту).

У файлов специальных устройств есть несколько любопытных архитектурных особенностей. Во-первых, с этими файлами не связаны блоки данных на машинном носителе. Попробуем посмотреть данные об этих файлах с помощью команды **ls –li /dev**. Обращает на себя внимание необычный размер файлов. Вообще понятие «размер» для специального файла неприменимо, так как это не настоящий файл, а указатель на соответствующий драйвер. Вместо размера команда **ls** показывает для таких файлов два числа: «мажорный» и «минорный» номера устройств. Упрощенно можно считать, что «мажор» – это порядковый номер драйвера устройства, а «минор» – внутренний номер устройства в таблице обслуживающего его драйвера. Например, жесткому магнитному диску, подключенному ведущим (master) к первому IDE-интерфейсу и имеющему обозначение **hda** (условные обозначения файлов устройств см. ниже), соответствует «мажорный» номер 3. Первый раздел этого диска, обозначенный **hda1**, имеет «минорный» номер 1, второй раздел **hda2** – номер 2 и так далее.

Вторую особенность можно распознать, только воспользовавшись дисковым редактором. У каждого специального файла есть уникальный номер – **inode**. Но если открыть какой-нибудь индексный дескриптор, принадлежащий специальному файлу, то сразу в голову приходит мысль об ошибке – своей или программной. Редактор **lde** или **extview** отобразит описатель совершенно другого файла – как правило, обычного. И это не ошибка. Файлы специальных устройств не нуждаются не только в блоках данных – им не нужны и индексные дескрипторы. Поэтому команда **ls –li /dev** в каталоге специальных файлов отображает **inode**, принадлежащие другим файлам.

Специальный файл устройства имеет только имя в каталоге. Поэтому удаление этого имени равносильно удалению специального файла.

Обычно у администратора необходимости в создании специальных файлов не возникает, т. к. они создаются для всех известных настоящих и будущих устройств на этапе установки системы командой

**mknod /dev/filename { c | b } MAJOR MINOR**

В процессе функционирования ОС Linux специальная служба по установленным правилам создаёт и удаляет файлы подключаемых и отключаемых устройств. Если администратор случайно удалил файл корневой файловой системы, то он будет автоматически создан при последующей перезагрузке системы.

У пользователей по умолчанию нет прав для создания таких файлов, так как бесконтрольное создание и использование ссылок на драйверы весьма опасно для системы. Если пользователям будет позволено создавать файлы устройств и владеть ими, они смогут беспрепятственно и бесконтрольно осуществлять ввод и вывод информации.

Располагаются файлы устройств в каталоге **/dev** (device – устройство). Узнать, какому устройству соответствует специальный файл, можно по характерным именам. К числу символьных устройств относятся:

- **lp0, lp1** (**lp** – line port) – параллельные порты;
- **ttyS0, ttyS1** (**tty** – teletype) – последовательные порты **COM1** и **COM2**;
- **ttyN** – физический или виртуальный терминал;
- **audio** – звуковой адаптер;
- **ht0, st0** – IDE- и SCSI-накопители на магнитной ленте.

Блочными устройствами являются:

- **fd0, fd1** (**fd** – floppy disk) – соответственно первый и второй дисководы ГМД;
- **hdX[Y]** (**hd** – hard disk) – диск или логический раздел жесткого диска (магнитного или оптического) с IDE-контроллером. X – символы a,b,c,d, обозначающие: **a** – «master» на первом интерфейсном канале, **b** – «slave» на первом интерфейсном канале, **c** – «master» на втором интерфейсном канале, **d** – «slave» на втором интерфейсном канале, Y – номера разделов на жестком диске;
- **sdX[Y]** (**sd** – SCSI disk) – диск или логический раздел жесткого диска со SCSI-контроллером.

IDE-устройства в ОС Linux представлены следующими файлами:

- **/dev/hda** – «master» на первом интерфейсном канале;
- **/dev/hdb** – «slave» на первом интерфейсном канале;
- **/dev/hdc** – «master» на втором интерфейсном канале;

- **/dev/hdd** – «slave»на втором интерфейсном канале.

Например, при подключении исследуемого IDE-диска в качестве «master»-устройства ко второму интерфейсному каналу диск будет представлен как файл **/dev/hdc.**

Обращение к разделам на диске производится по их номерам, указываемым в конце имени файла диска. Всего на IDE-диске может быть адресовано 32 раздела. Первые 4 номера используются для обозначения первичных разделов, а остальные 28 номеров – для логических разделов. Например:

- **/dev/hda2** – второй первичный раздел диска,
- **/dev/hda6** – второй логический раздел диска.

SCSI-диски в ОС Linux представлены в виде следующих файлов:

- **/dev/sda** – первый диск,
- **/dev/sdb** – второй диск,
- **/dev/sdc** – третий диск,
- **. . .**
- **/dev/sdp** – шестнадцатый диск.

Количество SCSI-дисков в каталоге **/dev** зависит от контроллера и может быть равно 8 или 16, и в соответствии с этим количеством дискам назначаются буквы. На самом деле к SCSI-контроллеру можно подключить 7 или 14 дисков (по семь к каждому из двух каналов) по той причине, что одним из устройств в канале SCSI-интерфейса является сам контроллер.

Обращение к разделам на диске, так же как и к IDE-дискам, производится по их номерам, указываемым в конце имени файла диска. Всего на диске может быть адресовано 15 разделов. Первые 4 номера используются для обозначения первичных разделов, а остальные 11 номеров – для обозначения логических разделов. Например:

- **/dev/sda2** – второй *первичный* раздел первого диска,
- **/dev/sda6** – второй *логический* раздел первого диска.

SCSI-диски чаще встречаются на серверных платформах и являются редкостью в персональных компьютерах. По причине того, что ATAPI-интерфейс фактически является урезанной версией SCSI, разработчики решили сэкономить на именах устройств и обозначили символами **sd** жесткие магнитные диски с SATA-интерфейсом, а также полупроводниковые устройства памяти с USB-интерфейсом.

Информация о логических разделах нескольких фиксированных и съемных устройств дисковой памяти, выведенная командой **fdisk –lu**, приведена на рис. 3.1.

```
Disk /dev/hda: 160.0 GB, 160041885696 bytes
255 heads, 63 sectors/track, 19457 cylinders, total 312581808 sectors
Units = sectors of 1 * 512 = 512 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1              63     2104514     1052226   82  Linux swap
/dev/hda2   *     2104515    18892439     8393962+  83  Linux
/dev/hda3        18892440    29382884     5245222+   c  W95 FAT32 (LBA)
/dev/hda4        29382885   312576704   141596910    7  HPFS/NTFS

Disk /dev/sda: 750.1 GB, 750156374016 bytes
255 heads, 63 sectors/track, 91201 cylinders, total 1465149168 sectors
Units = sectors of 1 * 512 = 512 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *          63  1465127999   732563968+   7  HPFS/NTFS

Disk /dev/sdb: 320.0 GB, 320072933376 bytes
255 heads, 63 sectors/track, 38913 cylinders, total 625142448 sectors
Units = sectors of 1 * 512 = 512 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *          63   625137344   312568641    7  HPFS/NTFS

Disk /dev/sdc: 258 MB, 258998272 bytes
16 heads, 32 sectors/track, 988 cylinders, total 505856 sectors
Units = sectors of 1 * 512 = 512 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdc1   *          32      505854      252911+   b  W95 FAT32
Partition 1 has different physical/logical endings:
     phys=(986, 15, 32) logical=(987, 15, 31)
```

Рис. 3.1. Информация, выводимая командой **fdisk –lu**

Выведенная информация нуждается в некотором объяснении. К системному блоку персонального компьютера подключены три жестких магнитных диска и один съемный носитель USB-Flash. Один жесткий диск подключен к IDE-интерфейсу, два других используют SATA-интерфейсы. Если судить по объему памяти, устройства **sda** и **sdb**, отображенные в листинге, предположительно являются жесткими дисками с SATA-интерфейсом, а **sdc** – носителем USB-Flash. Второй и третий жесткие диски, а также полупроводниковая память ассоциируются со SCSI-дисками и обозначаются соответственно.

Первый диск с IDE-интерфейсом имеет емкость в 160.0 Гб = = 160041885696 байтов. Его трехмерная логическая геометрия имеет размерность 19457 цилиндров, 255 головок и 63 сектора на дорожку, что в совокупности дает 312581808 сектора по 512 байтов.

Трехмерная логическая геометрия диска CHS (цилиндры, головки, сектора) не соответствует реальности. Действительно, можно ли представить, что внутри герметичного блока жесткого диска размещено 255 магнитных головок или, соответственно, 128 дисков? Чаще всего в корпусе гермоблока имеется всего два жестких диска и, соответственно, 4 магнитных головки. Количество секторов на дорожку также указано неверно; в

различных пространственных зонах диска оно в зависимости от длины окружности дорожек находится в диапазоне от нескольких десятков до нескольких сотен секторов. Настоящая физическая геометрия диска известна только его контроллеру. Однако человек, не имея доступа внутрь гермоблока, видит дисковое пространство глазами операционной системы.

Обратим внимание на разделы IDE-диска. Таких разделов четыре (первичных), и они заняты различными файловыми системами. Одна логическая дорожка (63 сектора) зарезервирована, а первый сектор отведён под главную загрузочную запись (MBR). Первый раздел занят под виртуальную память (в Linux файл подкачки выделяется в отдельный раздел). Второй раздел занят файловой системой Linux (это может быть **ext2fs**, **ext3fs** либо **reiserfs**), а звездочка в столбце **Boot** указывает на то, что этот раздел является загрузочным. Третий раздел отведен под файловую систему FAT32, а четвертый – NTFS. Плюсы после количества блоков в разделах указывают на то, что раздел не кратен целому числу цилиндров.

Говоря о блочных устройствах, трудно обойти вниманием утилиты, предназначенные для контроля и установки параметров этих устройств. Одна из таких утилит называется **hdparm** и предназначается для контроля и настройки параметров накопителей с **IDE** и **SATA** интерфейсами.

С указанной утилитой рекомендуется [13] работать в однопользовательском режиме, в который можно перейти командой **telinit 1**. При отсутствии иных процессов, расходующих процессорное время, можно точнее оценить быстродействие устройств дисковой памяти. Так, информацию о быстродействии жесткого магнитного диска можно получить командой

```
hdparm –tT /dev/hda
```

или

```
hdparm –tT /dev/sda
```

Параметр **–T** тестирует всю подсистему кэширования дисковой памяти, включая процессор и разделы дисковой и оперативной памяти. Параметр **-t** оценивает скорость считывания данных без участия кэша. Для точной оценки команду следует запустить несколько раз, а показатели усреднить.

С помощью команды

```
hdparm /dev/hda
```

выводится информация о настройках жесткого диска, используемых по умолчанию. Обычно применяются самые безопасные, но далеко не оптимальные режимы. Оптимизировать параметры часто приходится на свой риск, особенно если необходимо копировать большие объемы данных. Оптимизация и контроль жестких магнитных дисков с интерфейсами **SCSI** производится с помощью аналогичной утилиты **sdparm**.

Достаточно эффективной защитой от несанкционированного исполь-

зования внешних устройств долговременной памяти может стать установка запрета на доступ к файлу устройства, например, с помощью команды

```
chmod 640 /dev/fd0
```

Но и этого может оказаться недостаточно, если не исключить скрытых групповых прав. Так, например, в файле **/etc/group** существует несколько псевдогрупп, в которые по умолчанию включены все пользователи. Утилита **useradd** и некоторые ей подобные, используемые для создания новых учетных записей, берут данные из уже упомянутого конфигурационного файла **/etc/login.defs** (defs является сокращением от defaults). В этом файле можно найти любопытную строку, которая выглядит так:

```
CONSOLE_GROUPS          floppy:audio:cdrom:video:plugdev
```

Это должно означать, что каждый из вновь зарегистрированных пользователей автоматически записывается в состав всех перечисленных дополнительных групп, о чем файл **/etc/group** нам ничего не сообщает. Одна из таких групп называется **floppy**. Проверив информацию о файле устройства **/dev/floppy/0** с помощью команды **ls –l** или **stat**, мы узнаем, что владельцем накопителя на гибких магнитных дисках является **root**, а *его* группа под названием **floppy** имеет права на чтение и запись гибких магнитных дисков. Получается, что система по умолчанию предоставляет любому пользователю полные права на работу с ГМД, но эта информация в учетных записях пользователей *не содержится*. Аналогичные права предоставлены пользователям к устройству **cdrom** и динамически подключаемым устройствам, которые обозначены группой **plugdev**. Чтобы проверить свои подозрения, запросим с помощью команды **ls –la** информацию о файле устройства **/dev/hdc**, интерфейс которого обычно используется для подключения приводов **CD/DVD**, а также устройства **/dev/sdc**, которое на конкретно взятом компьютере (см. рис. 3.1) обозначает подключаемую полупроводниковую память USB-Flash:

```
brw-rw---- 1 root cdrom 22, 0 2008-11-14 14:40
/dev/hdc

brw-rw---- 1 root plugdev 8, 32 2008-11-14 14:44
/dev/sdc
```

Для предупреждения угрозы использования устройств из строки

```
CONSOLE_GROUPS          floppy:audio:cdrom:video:plugdev
```

в файле **/etc/login.defs** необходимо удалить ненужные группы или полностью закомментировать эту строку.

Для решения некоторых задач в системе имеется несколько виртуальных устройств, которые не имеют аппаратных компонентов:

- **/dev/null** – «нулевое» устройство, своеобразная «черная дыра», поглощающая направленный в нее поток данных. В этот файл можно

только записывать,

- **/dev/zero** – «рог изобилия», файл, из которого можно бесконечно читать одни двоичные нули,
- **/dev/random** – устройство, генерирующее поток случайных чисел при активности пользовательского ввода. Движение мыши или нажатие нескольких клавиш на клавиатуре используется системой для генерации случайных двоичных чисел, представленных потоком байтов. Поэтому виртуальное устройство **/dev/random** может быть использовано как индикатор присутствия (активности) пользователя за компьютером,
- **/dev/loop** – устройство обратной связи, позволяющее имитировать виртуальное блочное устройство (диск).

Путем комбинации двух виртуальных устройств можно создать процесс, в буквальном смысле переливающий «из пустого в порожнее». Это достигается с помощью любой из двух команд

```
od /dev/zero > /dev/null
od < /dev/zero > /dev/null
```

Подобные «процессы» могут изрядно нагрузить центральный процессор, и мы воспользуемся такой имитацией при наблюдении за процессами. Перенаправление в **/dev/null** также будет использовано при проведении лабораторных работ для виртуального копирования большого объема данных.

«Генератор» **/dev/zero** может быть с успехом использован для программной очистки долговременной памяти от остатков конфиденциальной информации. Команда может выглядеть так:

```
cat /dev/zero > /dev/hda7
```

*Следует помнить, что после запуска такой команды восстановить удаленную информацию не удастся!* Очистка дискового пространства с помощью «генератора нулей» производится довольно быстро, но для гарантированного удаления конфиденциальных данных необходимо каждую ячейку памяти многократно переписать случайной последовательностью битов. Для гарантированного удаления данных необходимо использовать утилиту **shred**.

Использование **/dev/zero** или **/dev/random** для затирания содержимого файла приведёт не только к его затиранию, но и увеличению его длины до исчерпания свободного дискового пространства файловой системы, в которой находится затираемый файл.

Устройство обратной связи **/dev/loop** служит для имитации блочного устройства, имеющего вид обычного файла. Далее в нем можно создать файловую систему и произвести ее монтирование или, если в нём уже есть файловая система, например файл-образ компакт-диска, произвести ее монтирование. Поскольку о монтировании устройств еще ничего не говорилось, применение **/dev/loop** будет рассмотрено ниже.

### 3.3. Монтирование файловых систем

Большим преимуществом операционных систем UNIX/Linux является их способность «прозрачно» для пользователя работать с разнообразными файловыми системами. Эта возможность реализована с помощью многочисленных драйверов, «понимающих» архитектуру известных файловых систем и драйвера виртуальной файловой системы, которая их все объединяет. Перечень файловых систем, с которыми может работать ОС Linux, можно узнать из файла

**/usr/src/linux/Documentation/filesystems/00-INDEX**.

Монтированием (так дословно переведена на русский язык команда **mount**) называется операция подключения файловой системы к «дереву каталогов» работающей ОС Linux; в частности, это может быть файловая система внешнего устройства долговременной памяти. При монтировании содержимое файловой системы устройства дисковой памяти добавляется к существующему дереву каталогов в виде дополнительной ветви, «растущей» из точки монтирования. Монтирование – это привилегированная операция, доступная только суперпользователю. Любую известную ОС Linux файловую систему можно смонтировать при помощи утилиты **mount**. Это довольно сложная команда с большим числом параметров. В стандартном варианте она выглядит следующим образом:

**mount -t type -o option <device> <dir>** ,

где

- **type** – тип монтируемой системы (**ext2, ext3, msdos, vfat, ntfs, iso9660** и т. д.). Тип **auto** – это предоставление программе **mount** возможности автоматически определить монтируемую файловую систему
- **option** – параметры монтирования, например **ro** – только чтение, **rw** – чтение и запись, **remount,ro** – перемонтирование в режиме для чтения, **iocharset=koi8-r** – указание используемой кодировки для правильного отображения имени файлов и каталогов, если текущей локалью является кодировка **KOI8-R**. Более подробно параметры монтирования будут перечислены при рассмотрении конфигурационного файла **/etc/fstab**,
- **<device>** – имя специального файла устройства, которое идентифицирует диск с монтируемой файловой системой, например **/dev/hda2** или **/dev/fd0**,
- **<dir>** – имя каталога, к которому будет монтироваться файловая система (например, **/mnt/floppy** или **/mnt/ntfs**). Если в исходной файловой системе из точки монтирования выходили другие подкаталоги и файлы, то после монтирования они сделаются невидимыми – их «за-

кроют» ветви примонтированной файловой системы. «Закрытые» файлы не уничтожаются – они вновь станут видимы после размонтирования. По этой причине в качестве точки монтирования рекомендуется использовать пустой каталог. В то же время администратор с помощью камуфляжного монтирования может скрывать от пользователей и потенциальных взломщиков существование отдельных каталогов. Файлы, открытые до такого монтирования, продолжают оставаться доступными открывшим их программам.

Например, команда

```
mount –t vfat –o iocharset=koi8-r /dev/sda1 /mnt/usb
```

предназначена для монтирования файловой системы FAT32 на устройстве полупроводниковой памяти USB-Flash к точке монтирования **/mnt/usb** с использованием кодировки **KOI8-R** для правильного отображения имен каталогов и файлов, при этом точка монтирования **/mnt/usb** должна существовать.

Команда монтирования гибкого магнитного диска может быть записана в упрощенной форме (в этом случае используются параметры команды по умолчанию)

```
mount /dev/fd0 /mnt/floppy
```

При монтировании файловых систем, в которых не предусмотрено прав доступа (например, FAT), утилита **mount** назначает владельца файловых объектов и права доступа к ним по умолчанию. Владельцем объявляется пользователь, производивший монтирование, а права доступа к файлам и каталогам устанавливаются в **0755.**

Монтирование файловых систем пользователями производится на основе разрешений, записанных в текстовом конфигурационном файле утилиты **mount /etc/fstab**. Содержимое этого файла приведено на рис. 3.2 и представляет собой таблицу из шести столбцов.

```
dev/hda1   /mnt/ntfs    ntfs      defaults                  0   0
dev/hda2   /mnt/fat32   vfat      defaults                  0   0
dev/hda3   swap         swap      defaults                  0   0
dev/hda4   /            ext2      defaults                  1   1
devpts     /dev/pts     devpts    gid=5, mode=620           0   0
/proc      /proc        proc      defaults                  0   0
/dev/fd0   /mnt/floppy  msdos     defaults, users, noauto   0   0
/dev/hdc   /mtt/cdrom   iso9660   ro, user, noauto          0   0
```

Рис. 3.2. Содержимое файла **/etc/fstab**

В первом столбце указывается дисковое (блочное) устройство, точнее раздел диска или диск целиком, на котором содержится файловая система, подлежащая монтированию. Типовые названия блочных устройств были приведены в предыдущем параграфе. Кроме блочных устройств в таблице

присутствуют два псевдоустройства: файловая система **/proc**, рассмотренная в параграфе 2.5, и псевдотерминал **devpts** [3].

Второй столбец таблицы указывает точку (каталог) монтирования. Каталог к моменту монтирования должен существовать. Имена точкам монтирования обычно даются таким образом, чтобы они ассоциировались с конкретными устройствами (например, **/mnt/floppy** или **/mnt/cdrom**).

Третий столбец содержит тип файловой системы. Следует напомнить, что файл подкачки в системах Linux, идентифицируемый в таблице как **swap**, может размещаться на отдельном логическом разделе жесткого магнитного диска, отдельном диске или в файле; в случае его размещения в файле файловая система с файлом подкачки должна монтироваться до его подключения.

В четвертом столбце таблицы указываются параметры монтирования. Это те параметры, которые указываются после аргумента **–o** в командной строке утилиты **mount** :

- **ro** – (read only) – файловая система монтируется только для чтения (обычно этот параметр указывается для привода чтения оптических дисков, а также для других устройств памяти, монтирование которых не должно сопровождаться записью, например при проведении криминалистических исследований),
- **rw** – файловая система монтируется для чтения и записи (по умолчанию). При этом не следует забывать, что оптические диски CD-R/DVD-R по определению монтируются в режиме **read only**,
- **exec/noexec** – разрешить или запретить запуск исполняемых файлов, расположенных в данной файловой системе. Таким образом, например, можно запретить запуск неизвестных и потенциально опасных программ из подключаемой файловой системы. Запрет **noexec** эквивалентен снятию права на исполнение для всех обычных файлов и защищает только от случайного запуска исполняемого файла. Стоит скопировать файл программы на другой носитель и установить для него право на исполнение – и программу можно запускать. Этот запрет также не действует на файлы сценариев, которые, как известно, можно запускать посредством указания их имени после имени командного интерпретатора, имея только право на чтение,
- **suid/nosuid** – принимать во внимание или игнорировать дополнительные атрибуты **SUID/SGID**, позволяющие запуск исполняемых файлов из примонтированной файловой системы с правами владельца файла или его группы,
- **dev/nodev** – разрешить или запретить использование на примонтированном разделе файлов специальных устройств,
- **nouser/user(s)** – запретить или разрешить пользователям монтировать данную файловую систему. Параметр **user** указывает, что монти-

ровать файловую систему может любой пользователь. Отличие параметров **user** и **users** заключается в правах на размонтирование устройства. Параметр **user** означает, что размонтировать устройство может только тот, кто его монтировал, а **users** дает права на размонтирование любому пользователю,

- **defaults** – использовать параметры по умолчанию, что заменяет набор параметров **rw, suid, dev, exec, auto, nouser**. Если в четвертом столбце одновременно указаны параметры **defaults** и **user(s)**, то значения параметров по умолчанию изменяются на **noexec, nosuid** и **nodev**. Утилита **mount** обрабатывает параметры слева направо, замещая ранее встреченные значения параметров значениями, указанными далее. Поэтому указание **user** перед **defaults** равносильно отмене первого параметра.

Пятый столбец таблицы может содержать 0 или 1. Единица разрешает производить резервное копирование данной файловой системы утилитой **dump**, а нуль – не разрешает.

Шестой столбец используется утилитой проверки файловых систем **fsck** (file system check). Если указан «0», то файловая система не проверяется, цифры «1» или «2» указывают очередность проверки.

Аналогичный формат таблицы монтирования используется еще в двух файлах: виртуальном **/proc/mounts** и реальном **/etc/mtab**. В эти файлы процесс **mount** записывает информацию об уже смонтированных файловых системах.

Создание и монтирование файла, содержащего в себе файловую систему (в Linux возможно и такое!) заключается в следующем:

- с помощью утилиты **dd** создается файл требуемого объема (например, 10 Мб), состоящий из двоичных нулей:

```
dd if=/dev/zero of=/tmp/virt_disk1 bs=1M count=10;
```

- на виртуальном диске создается файловая система (например, **ext2fs**):

```
mke2fs -F /tmp/virt_disk1;
```

- создается точка монтирования (можно это сделать в том же каталоге **/tmp**):

```
mkdir /tmp/virt_fs;
```

- с помощью устройства обратной связи **/dev/loop0** монтируется виртуальный диск с созданной файловой системой ext2fs:

```
mount –t ext2 –o loop=/dev/loop0 /tmp/virt_disk1
/tmp/virt_fs.
```

Файл-образ диска, полученный путем копирования, монтируется также с указанием параметра **–o loop=/dev/loop0** .

Прежде чем подключать (монтировать) файловую систему, следует

проверить её целостность. Если файловая система не была отключена должным образом, при попытке подключить ее без проверки пользователю будет выдано сообщение о невозможности выполнить подключение и предложено воспользоваться утилитой проверки и восстановления файловой системы **fsck** (или **e2fsck**). Это справедливо для файловой системы **ext2fs**, в случае же наличия на исследуемом разделе файловой системы **ext3fs** ошибка будет исправлена с помощью журнала.

Порядок запуска утилиты при проверке:

**e2fsck -fy /dev/hda7** ,

где **f** – принудительное выполнение проверки в случае, если файловая система этого не требует; **y** – работа без запросов пользователю.

Демонтируются файловые системы с помощью команды **umount**. Она записывается в двух вариантах:

**umount <device>**

или **umount <dir>**

Демонтирование всех файловых систем, перечисленных в файле **/etc/fstab**, можно произвести с помощью одной команды

**umount -a**

Если файловая система занята (открыты ее каталоги, запущены исполняемые файлы, открыты обычные файлы и др.), ее демонтировать невозможно – система сообщит об ошибке. Выяснить, какой процесс использует файловую систему, можно с помощью команды **lsof +D <dir>** (следует указать нужную точку монтирования).

Нельзя извлекать носители из приводов или отключать/отстыковывать съёмные устройства, если не предусмотрено механической блокировки процесса извлечения, до окончания процесса размонтирования. В зависимости от состояния, в котором находился процесс ввода/вывода файловой системы извлекаемого/отключаемого устройства, будет зависеть целостность данных этой файловой системы.

В случае, если пользователи имеют права доступа на чтение к соответствующим устройствам, то запрет на монтирование, предусмотренный в конфигурационном файле **/etc/fstab**, еще не гарантирует, что пользователи не сумеют использовать устройства для блочного копирования. Ни в каком монтировании подобные действия не нуждаются.

### 3.4. Копирование и запись данных

Копирование является неотъемлемым свойством информации и связано с ее переносом на иные физические носители или в их пределах без изменения содержания. Следует различать копирование файлов, посекторное копирование носителя информации с одного на другой, посекторное

копирование носителя информации в файл и обратный процесс. Можно выделить побитовое копирование (клонирование) носителя информации на другой носитель или в файл. Например, при проведении некоторых компьютерно-технических экспертиз требуется создать точную копию исследуемого устройства памяти в целях сохранения доказательства противоправной деятельности; при этом носитель-копия должен в точности соответствовать оригиналу, включая физическую геометрию исправных и поврежденных секторов.

Наиболее распространенный вид копирования – это создание еще одного экземпляра файла. Файл можно скопировать в тот же каталог, где расположен оригинал, но необходимо изменить имя копии. Можно скопировать файл с прежним именем, но только в другой каталог. Варианты команды **`cp`** для указанных случаев были приведены выше. Для копирования файла или группы файлов надо иметь следующие права:

- на чтение каждого файла-оригинала,
- на чтение и исполнение каталога, в котором исходные файлы хранятся,
- на запись и исполнение в каталоге, куда предстоит записать копии.

Если имя копируемого файла известно и он доступен для чтения, его можно скопировать, имея лишь право на поиск (исполнение) в каталоге, где он находится. Возникает вопрос: достаточно ли права на поиск в каталоге, если из него копируется группа файлов, имена которых заданы по маске, в т. ч. с использованием символов «**`*`**» и «**`?`**». Ответ отрицательный: для копирования файла с неопределенным именем его имя следует установить, для чего необходимо прочитать файловые записи в каталоге и сопоставить их с маской. Такое копирование требует наличия прав на чтение и исполнение для каталога.

Уместно отметить различия между жесткой ссылкой и копией файла, а также между копией и оригиналом. Жесткая ссылка – это еще одно имя файла, указывающее на единственный индексный дескриптор. При копировании создается другой файл, имеющий не только отдельную запись в каталоге, но и свой уникальный индексный дескриптор, а также занимающий *иные* блоки на дисковом пространстве. Файл-копия может отличаться от оригинала правами доступа и обновленными временными отметками, а также владельцем файла, если копирование производилось другим пользователем. Одинаковым для файла-оригинала и его копии является только *содержимое* блоков данных.

Если на месте расположения копии уже существует файл с таким же именем, он обнуляется (точнее, его блоки данных объявляются свободными), а индексный дескриптор передается создаваемой копии. При копировании права доступа (код режима) файла не изменяются, за исключением сброса флагов эффективных идентификаторов **`SUID`** и **`SGID`**. Системная функция, записывающая копию файла, обнуляет биты эффективных прав доступа. Владелец копии и его группа меняются – ими становятся пользователь, копирующий файл и его группа. После копирования пользователь

может сделать со своим экземпляром файла всё, что ему заблагорассудится. Обычные пользователи имеют права на чтение в каталогах **/bin, /sbin, /usr/bin, /usr/sbin** и в принципе могут обзавестись своими копиями различных утилит. Сами по себе копии утилит опасности не вызывают, так как пользователь сможет запустить их только со своими правами.

Обычное файловое копирование производится с помощью уже упомянутой выше утилиты **cp**. С ее помощью можно скопировать один или группу файлов из одного каталога в другой. Имеется возможность копирования каталогов, в том числе и рекурсивного.

Резервное копирование является одной из распространённых мер защиты данных. От обычного копирования оно отличается тем, что пункт назначения находится на другом машинном носителе, возможно находящемся в составе другого компьютера. Резервному копированию и его политике уделяется довольно много внимания в различных источниках, посвященных администрированию систем. Для резервного копирования файлов и каталогов в ОС Linux используется несколько известных утилит, которые появились задолго до её появления.

При одинаковом объеме копируемых данных процесс файлового копирования происходит быстрее при большом размере файлов. В идеале следует копировать один большой файл. Для сборки файлов каталога в один непрерывный массив в Linux существует утилита **tar** (от **T**ape **Ar**chive – ленточный архив). Команда сборки выглядит так:

```
tar –cf backup.tar <dir>
```

С помощью утилиты **tar** можно группировать в один файл содержимое нескольких каталогов, отделяя их в командной строке пробелами, например:

```
tar –cf backup.tar /home /etc
```

Файлы копируются с указанием их полного имени, поэтому распаковка должна производиться из корневого каталога

```
tar –xf backup.tar <dir>
```

«Зеркальным» называют файловое копирование, при котором группа файлов копируется в другой каталог с сохранением всех файловых атрибутов в копиях. Подобное можно выполнить с помощью одной из команд

```
rsync –a <dir1> <dir2>
```

или

```
cp –a <dir1>/* <dir2>
```

При копировании каталогов с помощью утилиты **rsync** следует иметь в виду одно обстоятельство. Если исходный каталог завершается

символом **/**, например **/home/user1/**, то копируется содержимое каталога, но не сам каталог. Для копирования каталога со всем его содержимым его имя следует задавать без завершающего символа **/**, например **/home/user1**.

Если некоторые файлы из исходного каталога уже удалены, утилита **rsync** по умолчанию не удаляет их в пункте назначения. Для полной синхронизации файлов требуется задать в командной строке атрибут **–delete**.

Для выполнения резервного копирования дисковых разделов используется утилита **dump**, однако в дистрибутивах ОС Linux она не является штатной и может отсутствовать. С помощью этой утилиты можно производить полное или выборочное (дополняющее) копирование. Для копирования нулевого уровня (так называется полное резервное копирование) используется команда

```
dump –0 –u –f /dev/tape /home
```

**/dev/tape** – так обычно обозначается псевдоним (символическая ссылка) специального файла ленточного устройства. Далее предполагается, что домашние каталоги пользователей размещаются на отдельном логическом разделе, примонтированном к каталогу **/home**. При этом в конфигурационном файле **/etc/fstab** строка с **/home** в поле для резервного копирования должна содержать единицу; без нее утилита **dump** откажется работать.

Не рекомендуется производить резервное копирование разделов, которые в данный момент примонтированы к корневой файловой системе; их следует временно демонтировать.

Как уже отмечалось выше, файлам, которые не следует резервировать, с помощью утилиты **chattr** может присваиваться дополнительный атрибут **d**. Утилита **dump** игнорирует файлы с подобной меткой.

Файловому копированию свойственно несколько недостатков. Во-первых, в некоторых случаях оно ведет к изменению временных отметок файлов, а также их владельцев. Файлы-источники как минимум меняют время последнего доступа, а файлы-копии обновляют три временных отметки. Во–вторых, при копировании данных на носитель с иной файловой системой не дублируются некоторые специфические для исходной файловой системы метаданные файлов. Наконец, не копируются каталоги, которые на момент копирования могли быть закрыты примонтированными разделами. Эти обстоятельства совершенно неприемлемы в случаях, когда необходимо получить юридически достоверную копию и исключить какие-либо изменения в источнике копирования. Для решения таких задач может подойти утилита **dd**. Это универсальная утилита для блочного (с изменяемым размером блока) копирования файлов. По той причине, что в ОС Linux диски и разделы представлены в виде файлов, можно утверждать, что она будет работать и с ними. Копирование дисковых разделов не нуж-

дается в их монтировании, но при этом они не должны быть смонтированными и использоваться.

**dd if=<источник> of=<приемник> bs=<размер_блока>**
**seek= <число_блоков> skip=<число_блоков>**
**count=<число_блоков> conv=noerror,fsync**

**if=<источник>** – файл, откуда копируются данные. Если источник не указан, копируются данные из стандартного ввода **stdin**, в случае интерактивной работы – введенные с клавиатуры. Поток данных может передаваться команде **dd** из другой программы через конвейер; в этом случае параметр **if=** не указывается, так как используется ввод из **stdin**.

**of=<приемник>** – файл, в который записываются данные. В случае отсутствия адресуемого файла в файловой системе он будет создан. Если параметр **of=** не указан, данные выводятся в стандартный вывод **stdout** или на экран. В случае отсутствия параметра **of=** вывод утилиты через конвейер можно перенаправить другой программе.

**bs=<размер_блока>** – размер блока копируемых данных, который по умолчанию равен 512 байтов. Максимальная скорость копирования обеспечивается при размере блока 4096 байтов (4 Кб), что равно одной странице виртуальной памяти. Минимальный размер копируемого блока можно указать равным одному байту, но реальный размер считываемого блока все равно не может быть меньше величины одного сектора на диске. Размер блока может задаваться раздельно для источника (**ibs – input block size**) и для приемника (**obs – output block size**). Если копируемые блоки одинаковы, то задается величина **bs**. Размер может задаваться в байтах (единица измерения не указывается), килобайтах (K), мегабайтах (M), гигабайтах (G).

**skip=<число_блоков>** – количество (десятичное число) пропущенных при копирования из источника блоков указанного размера.

**seek=<число_блоков>** – количество пропущенных приемником блоков.

**count=<число_блоков>** – количество копируемых блоков указанного размера.

**conv=noerror,fsync** – режим обработки ошибок, при котором блок, скопированный с ошибкой контрольной суммы в приемнике, заполняется нулями, а процесс копирования не прерывается. При отсутствии этого параметра копирование завершается после первой ошибки чтения. Аргумент **fsync** служит для того, чтобы скопированные данные не «застревали» в дисковом кэше, а сразу записывались на диск.

Этим не исчерпываются возможности этой великолепной утилиты. Она может использоваться для просмотра элементов архитектуры файло-

вых систем вместо дискового редактора. С помощью **dd** можно также вставлять данные в нужные места адресуемой памяти. Варианты использования утилиты:

- для создания файл-образа гибкого магнитного диска

```
dd if=/dev/fd0 of=/tmp/floppy conv=noerror
```

- для создания файл-образа компакт–диска

```
dd if=/dev/hdc of=/tmp/cdrom1 conv=noerror
```

- для создания файл-образа 7-го раздела жесткого магнитного диска с IDE-интерфейсом

```
dd if=/dev/hda7 of=/tmp/hd7 bs=4k conv=noerror,fsync
```

- для записи произвольной строки на гибкий магнитный диск, например, в качестве дополнительной метки носителя

```
echo 1234567890|dd of=/dev/fd0 bs=1 seek=10 count=10
```

- для вывода на экран для просмотра содержимого первого сектора жесткого магнитного диска, который является главной загрузочной записью (MBR)

```
dd if=/dev/hda bs=512 count=1|xxd|more
```

- для вывода на экран для просмотра дампа описателя 6-й группы блоков файловой системы **ext2fs**

```
dd if=/dev/hda6 bs=4096 skip=1 count=1|dd bs=32 skip=5 count=1|xxd
```

- для вывода на экран для просмотра дампа 11-го индексного дескриптора файловой системы **ext2fs**

```
dd if=/dev/hda6 bs=4096 skip=4 count=1|dd bs=128 skip=10 count=1|xxd
```

Одним из недостатков утилиты **dd** является отсутствие информирования пользователя о процессе ее работы. Когда речь идет о копировании физических дисков или логических разделов большого размера, подобная неизвестность действует угнетающе. В руководстве по использованию утилиты предлагается из другой консоли периодически выдавать команду вида

```
killall -USR1 dd
```

В ответ на сигнал процесс **dd**, не прерывая копирования, информирует о его текущих результатах.

Копирование поврежденных секторов с блочного устройства находится за пределами возможностей даже такой утилиты, как **dd**. Копирование такого уровня можно производить только с помощью специальных утилит от производителя машинного носителя, которые умеют работать непосредственно с контроллером дисковой памяти и не принимают во внимание ошибки чтения.

Запись данных на оптический носитель CD/DVD также является разновидностью копирования. Для записи оптических носителей в режиме командной строки в составе большинства дистрибутивов Linux имеется утилита **cdrecord** для записи CD-дисков и утилита **growisofs** для записи DVD.

Для того чтобы после записи указанные диски могли быть прочитаны стандартными способами, копируемые файлы должны представлять собой образы файловой системы **iso9660**. Однако никто не запрещает записывать этими утилитами на диски любые файлы — главное, чтобы объем файла не превысил размер CD/DVD-диска. Но после такой записи стандартным способом читать диски бессмысленно, так как способ чтения будет зависеть от способа создания записанного файла.

Для записи компакт-диска в командной строке необходимо указать номер привода в эмуляции ide-scsi шины, скорость вращения диска при записи, а также «упаковать» копируемый каталог или логический раздел в файл-образ в формате файловой системы **iso9660**. Для упаковки каталога в файл-образ используется еще одна утилита под названием **mkisofs**. Команда для создания файл-образа будущего диска выглядит так:

```
mkisofs –R –l -o /tmp/disk.iso <dir>
```

Задавая полное имя каталога, следует помнить, что оно в файл-образ не переносится. Содержимое каталога будет помещено в корневой каталог записываемого компакт-диска. Для создания **iso**-файла полномочий администратора не требуется.

Если необходимо тиражировать уже записанный компакт-диск, снять с него файл-образ можно с помощью уже известной команды **dd.**

```
dd if=/dev/cdrom of=/tmp/image.iso
```

Для того чтобы узнать номер устройства, необходимо предварительно запустить команду **cdrecord** в режиме поиска доступных магистралей:

```
cdrecord --scanbus
```

Результат поиска представлен на рис. 3.3.

Команда для записи **iso**-файла на компакт–диск выглядит следующим образом:

```
cdrecord –v -eject -sao dev=1000,0,0 speed=6 image.iso ,
```

где **speed** – скорость записи для привода, которая указывается несколько меньше максимальной в зависимости от степени износа устройства. Если явно не указывать скорость записи, то будет определяться и указываться максимальная скорость.

```
Cdrecord-ProDVD-ProBD-Clone 2.01.01a57 (i686-pc-linux-gnu) Copyright
(C) 1995-2009 JЖrg Schilling
Linux sg driver version: 3.5.27
Using libscg version 'schily-0.9'.
scsibus2:
        2,0,0   200) 'ATA     ' 'WDC WD3200BEVT-2' '11.0' Disk
        2,1,0   201) *
        2,2,0   202) *
        2,3,0   203) *
        2,4,0   204) *
        2,5,0   205) *
        2,6,0   206) *
        2,7,0   207) *
scsibus1000:
        1000,0,0 100000) 'HL-DT-ST' 'DVDRAM GMA-4082N' 'PT06' Remov-
able CD-ROM
        1000,1,0 100001) *
        1000,2,0 100002) *
        1000,3,0 100003) *
        1000,4,0 100004) *
        1000,5,0 100005) *
        1000,6,0 100006) *
        1000,7,0 100007) *
```

Рис. 3.3. Результат поиска доступных интерфейсов

Если перед записью CD-RW требуется очистить от ранее записанных данных, необходимо выполнить команду

```
cdrecord –v dev=1000,0,0 blank=fast
```

Нетрудно убедиться в том, что записать на оптический диск можно не только файл–образ специального формата. Скопировать на оптический диск можно любой файл (но только один):

```
cdrecord -v -sao dev=1000,0,0 speed=8 <file_name>
```

Вот только прочитать записанный компакт–диск, не содержащий признаков известной файловой системы, обычным образом вряд ли удастся. Для этого потребуется использовать уже рассмотренную утилиту **dd** либо дисковый редактор.

Имея в виду компьютерные преступления, необходимо обратить внимание на особую категорию копирования данных, которую называют криминалистической или судебной [6, 7]. Эксперт-криминалист, как правило, не имеет права исследовать изъятый машинный носитель, чтобы не повредить источник доказательств. Он обязан вначале создать детальную копию этого источника и затем исследовать ее. Судебное копирование производится посекторно с применением надежных системных или специальных

утилит. При копировании исключается любая форма записи на исходный носитель.

Довольно часто практикуется сетевое копирование, особенно если компьютеры оснащены быстродействующими сетевыми адаптерами. Компьютер, с которого производится копирование, будем называть объектовым, а тот, на который данные копируются, – целевым. Если оба компьютера не включены в один сегмент локальной вычислительной сети, подключение можно произвести с помощью предварительно подготовленного кабеля **cross-over** (в случае, если сетевые адаптеры не «умеют» определять прямое подключение). Если компьютеры не включены в одну ЛВС, скорость копирования может быть ниже.

Для сетевого копирования необходимо знать IP-адрес или доменное имя объектового компьютера (первое предпочтительнее) и установить сетевой адрес на целевой машине, чтобы он соответствовал диапазону адресов в данной ЛВС. Если объектовый компьютер можно выключать и загружать операционной системой со сменного машинного носителя, а подключение компьютеров производится отдельным сетевым кабелем, то установка IP-адреса производится на объектовом узле.

Если предстоит копирование конфиденциальной информации через открытый канал, данные передаются в зашифрованном виде. Так, для копирования директории **/home** по сети на другой компьютер с использованием закрытого протокола Secure Shell (ssh) можно использовать уже упомянутую команду

```
rsync –a –e ssh /home 192.168.1.11
```

Создать копию на другом сетевом компьютере можно с помощью уже рассмотренной утилиты **tar**. При этом утилита **tar** необходима для создания «длинного» файла, чтобы сеанс сетевого копирования не прерывался по окончании отдельных файлов. Собственно для сетевого копирования рекомендуют использовать утилиту **nc** (netcat), которая запускается в клиент-серверном режиме. В ней задаются IP-адрес узла назначения и порт транспортного уровня. Первая команда вводится на целевом компьютере

```
nc –l –p 4444 | tar –cf backup.tar
```

На компьютере-источнике вводится вторая команда

```
tar –cf /home | nc –w 2 192.168.10.20 4444
```

При необходимости шифрования сетевого трафика утилиту **nc** можно заменить защищенной оболочкой **ssh** [14]:

```
tar –cf /home | ssh 192.168.10.20 “cat > backup.tar”
```

Следует оговориться, что такое возможно только при беспарольном использовании Secure Shell.

Если сетевому копированию подлежит раздел жесткого диска или диск целиком, на целевом компьютере вводится команда

**nc –l –p 2222 > /home/user/sda1**,

а командная строка, обеспечивающая сетевое копирование на компьютере–источнике, будет выглядеть так:

**dd if=/dev/sda1 bs=4k|nc –w 3 192.168.1.20 2222**

### 3.5. Использование «жестких» и символических ссылок

Обычный или регулярный файл состоит из трех частей. Его первая часть является файловой записью в каталоге, состоящей из пяти полей и включающей имя файла и номер его индексного дескриптора, т. е. фактически является указателем на индексный дескриптор. Вторая часть файла – это 128-байтный индексный дескриптор, в котором хранятся метаданные файла. Третья, и главная, часть – это собственно данные, которые содержатся в логических блоках, выделенных файлу в пределах конкретной файловой системы.

Соображениями экономии дискового пространства и удобства именования файлов мы обязаны существованию таких объектов, как файловые ссылки, или указатели. С помощью утилиты **ln** можно создать два типа ссылок: «жесткие» и символические. В то же время само существование и неверное использование этих объектов может представлять не только удобство, но и информационную опасность.

Вспомним традиционное определение файла: «файл – это *именованная* область памяти» (*именованная* – значит имеющая уникальное имя). Но в Linux уникальным идентификатором файла является не его имя, а номер его индексного дескриптора. В данном случае правильнее было бы сказать «файл – это *нумерованная* область памяти». В то же время для обозначения одного и того же файла может использоваться до 65536 имен. Множество имен, или символьных указателей на один и тот же индексный дескриптор, называются непосредственными или «жесткими» ссылками. В самом индексном дескрипторе ни одно из имен файла не хранится, но содержится их суммарное число. Удаление последнего имени файла сопровождается логическим удалением файла с освобождением его индексного дескриптора и блоков данных.

Имея в своем каталоге имя часто используемого файла, пользователь получает более удобную возможность обращения к нему. С точки зрения расхода дискового пространства, как будет показано ниже, «жесткая» ссылка оказывается гораздо экономнее, чем копия файла.

Команда, создающая жесткую ссылку, записывается в виде

**ln <file_name> <link_name>**

Несмотря на то, что жесткая ссылка является указателем на **inode** файла, в команде используется одно из существующих имен; они совершенно равнозначны. Чтобы создать жесткую ссылку на файл, пользователю не требуется никаких прав на него. Минимально необходимо иметь

только право на вход в каталог, содержащий целевой файл (этот файл нужно знать по имени, иначе для того, чтобы его увидеть, потребуется еще право на чтение каталога), а также права на вход и запись в каталог, в котором создается новое имя. Возможность создания жесткой ссылки на недоступный файл кажется противоестественной, ведь появление у файла еще одного имени увеличивает на единицу счетчик ссылок, который хранится в **inode** файла. Но для перезаписи **inode** прав на запись в сам файл вовсе не требуется. Так, например, без модификации файла изменяются хранимые в его индексном дескрипторе временные отметки последнего доступа и удаления файла.

Для того чтобы проверить, сколько непосредственных ссылок (или имен) имеет файл, можно использовать команду **stat** или **ls -l** с именем файла в качестве аргумента. Найти все имена обычного файла можно, зная его **inode**. Для этого используется команда

```
find / -inum N
```

При создании нового каталога в него по умолчанию записываются две жесткие ссылки: ссылка на самого себя (**.**) и ссылка на родительский каталог (**..**). Указатель на подкаталог содержится в родительском каталоге. Таким образом, на каталог может существовать много жестких ссылок: их количество равно числу подкаталогов первого уровня + 2.

Создание иных жестких ссылок на каталог обычно не разрешается никому, включая администратора. Например, если в полном имени файла **/home/a/b/c** файл **c** (точнее – файловая запись в каталоге **b**) является жесткой ссылкой на каталог **a**, то образуется петля, которая делает невозможным обращение к файлу. Как следствие, становится невозможным рекурсивный обход каталога, а также его удаление. Такой результат может быть получен путем редактирования блока данных каталога, в котором содержатся файловые записи. Реализовать эту угрозу обычному пользователю, не наделенному административными правами, не удастся.

Никто и ничто, кроме квоты на дисковое пространство, не может помешать пользователю в копировании доступных для чтения и известных по имени файлов из каталогов, в которые он может войти. Количество копий также не ограничивается. Угрозы в копировании и тиражировании доступных для чтения файлов нет, тем более что копии меняют владельца. Посмотрим, существуют ли реальные угрозы в создании нового имени у чужого файла.

Угроза конфиденциальности на файловом уровне пресекается запретом чтения файла для посторонних. Обладание именем недоступного для чтения файла не позволяет ни прочитать, ни скопировать его.

Угрозы целостности также не существует. Если целевой файл не доступен на запись, обладание одним из его имен не позволит дописать его либо изменить в нем что-нибудь. Удаление файла – это стопроцентное нарушение его целостности, но в данном случае это не угроза (если исклю-

чить случайное удаление). Но здесь мы имеем дело с воспрепятствованием удалению. Единственное доступное право обладателя его последнего имени заключается в удалении файла вместе с его последним именем. Кстати, если администратор намерен уберечь пользовательские файлы от случайного или преднамеренного удаления, он может наряду с резервным копированием создать на них жесткие ссылки из своего каталога.

Угроза блокирования существует, но только для нарушителя – ведь недоступный файл, которому он создал новое имя, остается для него заблокированным. Что касается пользователя как легитимного обладателя файла, то ничего противоестественного в том, что удаленный файл становится для него недоступным, не усматривается.

И все-таки определенная угроза в существовании жестких ссылок существует. Она заключается в том, что один пользователь может воспрепятствовать другому в удалении файлов. Как известно, файл может быть логически удален, если удаляется его последнее имя. Обладатель (не обязательно владелец!) исходного файла может не обратить внимания на увеличившееся количество ссылок на файл и удалить его. Упоминание о файле исчезает из каталога обладателя, и у него может создаться иллюзия удаления файла (на время позабудем, что само логическое удаление файлов – также иллюзия для пользователей). Обладатель перестает беспокоиться о секретности и сохранности (целостности) «удаленного» файла. Между тем этот файл (точнее, его имя) перешел другому пользователю, который его владельцем и обладателем прав доступа не стал (в индексном дескрипторе файла записан **UID** его прежнего владельца). Если такая угроза представляет опасность для организации, администратору следует обратить внимание на поиск таких файлов и даже написать для этого сценарий.

Зная, что все события в системе могут находиться под контролем администратора, нелояльный пользователь может попытаться скрыть некоторые свои действия от контроля и аудита. Например, пользователь хочет воспользоваться утилитой **su** и попробовать свои силы в подборе пароля администратора, в отношении которого он имеет некоторые догадки. Для скрытия своих неблаговидных действий он может применить жесткую ссылку на файл программы. Выглядит это так. Пользователь создает из временного каталога жесткую ссылку на файл **su** и делает это с помощью команды

```
ln /bin/su /tmp/ls
```

То, что он пытается замаскировать обращение к опасной утилите обращением к обычной команде, вполне понятно. Так же естественно его желание не привлекать внимания администратора к своему пользовательскому каталогу. Данная команда не нарушает ничьих прав доступа и безукоризненно выполняется.

Затем пользователь вводит команду **/tmp/ls** и получает приглашение для ввода пароля администратора. В это время в списке процессов ото-

бражается процесс **ls**, запущенный от имени **root**, но из пользовательской консоли. Для скрытия места запуска пользователю необходимо превратить интерактивный процесс в фоновый и закрыть сеанс в данной консоли. Это ему не удастся по двум причинам. Во-первых, в фоновом режиме он не имеет возможности вводить пароль. Во-вторых, вызванный процесс **login** может «уйти» в фоновый режим, однако не позволит завершить сеанс, не завершив аутентификацию.

Далее пользователь в течение некоторого времени пытается подобрать вожделенный пароль. Здесь его вновь подстерегают неприятности. Пароль нельзя вводить неограниченное число раз; этому препятствуют ограничения, установленные в файле **/etc/login.defs**. Наконец, демон **syslogd** контролирует вовсе не запуск команды **su**, а только соответствующий системный вызов, и в файле **/var/log/secure** (или в **/var/log/auth**) неизбежно появляются записи, говорящие о попытках аутентификации.

Наконец, последние огорчения поджидают пользователя, когда он попытается удалить из доступного каталога **/tmp** созданную жесткую ссылку. Если бы это был регулярный файл, созданный пользователем, удаление прошло бы без проблем. Но стикки–бит, установленный на каталог, не позволяет пользователю удалять чужой файл, чем созданная жесткая ссылка и является. Перед вами был представлен пример уязвимости, который на самом деле говорит о хорошей продуманности защитных механизмов системы. Жаль, что подобная защита не является «сплошной».

Если жесткая ссылка – это только одна из трех частей файла, то символические ссылки представляют собой самостоятельные файлы, но состоящие в большинстве случаев не из трех, а из двух частей. Они являются указателями не на индексный дескриптор, а на одно из имен настоящего файла. Обращение к символической ссылке трансформируется в аналогичное действие по отношению к адресуемому файлу.

На символическую ссылку нельзя задать права доступа: она доступна каждому зарегистрированному пользователю для чтения, записи и исполнения (маска прав доступа 777). Изменение прав доступа к символической ссылке с помощью команды **chmod** приводит к изменению прав доступа на объектовый файл, если у пользователя имеются на то надлежащие права. Ссылку на недоступный файл создать легко, но обратиться к нему все равно будет невозможно, так как права доступа проверяются при обращении к адресуемому файлу.

Символическая ссылка создается с помощью команды

```
ln -s <file_name> <link_name>
```

Допустимо, если в этом есть смысл, создать символическую ссылку на другую символическую ссылку. Информационной угрозы в этом не усматривается.

У большинства символических ссылок не существует блоков данных.

Если адресуемое имя не превышает 60 символов, оно записывается в индексном дескрипторе вместо номеров адресуемых блоков (15 блоков х 4 байта). При исследовании **inode** символической ссылки с помощью дискового редактора следует обратить внимание на то, что номера адресуемых блоков данных выглядят неестественно и явно выходят из допустимых диапазонов номеров. В этих полях справа налево записываются символы полного либо относительного имени адресуемого файла в ASCII-коде. Просмотр таблицы индексных дескрипторов в шестнадцатеричном и символьном форматах позволяет сразу отличить описатель символической ссылки: она начинается байтами **ff a1** и сопровождается именем файла в секции ASCII-кода. Если имя адресуемого файла длиннее 60 символов, для его символической ссылки система выделяет один блок данных.

Еще одно отличие между жесткими и символическими ссылками заключено в диапазоне их действия. Жесткую ссылку на файл можно создать только в рамках данной файловой системы, т. е. в диапазоне уже существующих индексных дескрипторов. Символическая ссылка может адресовать файл, расположенный в другой файловой системе, в том числе на другом физическом носителе и другом сетевом узле.

Символические ссылки могут использоваться в качестве инструмента для файловой атаки. Объектом атаки может являться любой недоступный для пользователя файл, который он хотел бы уничтожить. Источником атаки может стать любой процесс с системными правами, который в процессе работы создает временные файлы и помещает их в каталог **/tmp**. Примечательность этого каталога в том, что любой зарегистрированный пользователь имеет в нем права на чтение, запись и поиск (исполнение).

Допустим, что злоумышленник узнал или угадал имя временного файла в каталоге **/tmp** и предварительно успел записать в этот каталог символическую ссылку с таким же именем. Тогда программа будет записывать данные уже не в свой временный файл, а через подставленную символическую ссылку в объектовый файл, который будет в ней указан. Угроз конфиденциальности здесь нет – если бы пользователю надо было прочесть конфиденциальную информацию из временного файла, он скопировал бы его в свой каталог и прочитал. Здесь более явной является угроза целостности. Например, программа, запущенная с правами администратора, через символическую ссылку в каталоге **/tmp** перенаправит запись в любой указанный файл, например в файл паролей или иной файл конфигурации. В результате весьма ответственная информация будет уничтожена.

Для противодействия подобным атакам временные файлы в обоих семействах универсальных операционных систем Windows и Linux создаются с псевдослучайными именами, и функция создания временного файла возлагается не на приложение, а на операционную систему. В Linux имеются системные функции **mkstemp** и **tmpfile**, вызов которых решает вышеперечисленные проблемы. Так, функция **mkstemp** генерирует шестисимвольное имя файла, подбирая каждый символ случайным образом. На-

рушителю угадать имя файла весьма трудно, но появляется проблема для пользователей. Удаление временных файлов возлагается не на операционную систему, а на программу, которая их создает. И если автор программы не позаботился об удалении временных файлов, то они будут скапливаться и захламлять каталог **/tmp**. Поскольку имя временного файла генерируется случайно, спустя некоторое время уже трудно разобраться, какой программой и с какой целью он был создан, содержит ли нужную информацию и подлежит ли удалению.

Из командной строки генерацию временных файлов с псевдослучайными именами можно произвести с помощью команды **tempfile**. Задавая эту команду с произвольным символом, мы создаем в каталоге **/tmp** файл нулевой длины с именем **fileXXXXXX**, где после префикса **file** генерируется шестибайтная случайная последовательность символов.

Теперь нетрудно ответить на вопрос: какая из ссылок занимает меньше места в памяти? Жесткая ссылка представляет собой еще одну файловую запись в одном из каталогов. Длина такой записи равна количеству символов в имени файла + 8 байтов + дополнение до ближайшего числа, кратного четырем. При наличии в блоке данных каталога свободного места для новой файловой записи увеличения его объема не произойдет. В противном случае каталогу будет выделен еще один логический блок. Следовательно, создание жесткой ссылки в большинстве случаев к дополнительному расходу дисковой памяти не приведет.

Символическая ссылка как минимум состоит из файловой записи в каталоге и 128-байтного индексного дескриптора. Если имя целевого файла превысит 60 символов, то символическая ссылка увеличится на величину одного логического блока. Таким образом, если имена ссылок имеют одинаковую длину, символическая ссылка в большинстве случаев займет больше места в памяти.

# 4. БЕЗОПАСНОСТЬ ФАЙЛОВЫХ СИСТЕМ EXT*FS

Файловая система (ФС) представляет собой способ организации, хранения и именования данных на физических носителях информации. Она определяет логический формат хранения данных, которые принято группировать в виде файлов.

Файловая система может создаваться на всем физическом устройстве или на его отдельном разделе. В случае использования технологии RAID вводится ещё один уровень абстракции между файловой системой и физическим устройством, но, при аппаратной реализации RAID, он для операционной системы не виден, или, как выражаются программисты, «прозрачен».

В ОС Linux «родными» файловыми системами являются **ext2fs** и её последующие версии: **ext3fs** и **ext4fs**. Файловая система **ext3fs** отличается от предшествующей версии только наличием журнала транзакций. **Ext2fs** и **ext3fs** имеют идентичную структуру и могут взаимно преобразовываться друг в друга на этапе монтирования. Архитектура ФС **ext4fs** еще окончательно не сложилась, и по этой причине ее рассмотрение приводится в Приложении.

## 4.1. Архитектура файловых систем ext*fs

Дисковое пространство выделяется файлам целыми блоками фиксированной длины. Блок является адресуемой единицей дискового пространства и может иметь размер 1024, 2048 или 4096 байтов. В ОС Windows* аналогом блока является кластер. Нетрудно заметить, что размер блока кратен стандартному размеру одиночного сектора на диске (512 байтов на магнитном диске или 2048 байтов на CD/DVD). Непосредственно с секторами работает драйвер файловой системы, но информация, выводимая некоторыми утилитами, может подразумевать под названием «блок» и сектор, и логический блок. Большой размер блока сокращает число обращений к диску при чтении или записи файла, но увеличивает долю нерационально используемого пространства памяти, особенно при наличии большого числа маленьких файлов.

Параметры для утилиты создания файловой системы **mke2fs** по умолчанию определены в файле **/etc/mke2fs.conf**; эти параметры зависят от типа носителя данных и его размера. Стандартным по умолчанию является размер блока в 4 Кб, что ускоряет процессы подкачки в виртуальной памяти (размер страницы подкачки также равен 4 Кб).

Каталоги в файловых системах Linux распределяются по всему диску. Файлы, входящие в один каталог, группируются в непосредственной близости друг от друга. Так делается для того, чтобы минимизировать число перемещений блока магнитных головок накопителя на жестких дисках при обращении к файлам одного каталога.

Просматривая информацию об основных каталогах файловой системы (рис. 4.1), выведенной с помощью команды **ls -li /**, следует обратить внимание на значительное отличие в номерах индексных дескрипторов каталогов первого уровня. Это косвенно указывает на то, что каталоги распределены по всему доступному системе пространству логического диска.

```
 131329 drwxr-xr-x    2 root     root       4096 Мар 18 17:42 bin
 328321 drwxr-xr-x    4 root     root       4096 Мар 18 15:18 boot
 196993 drwxr-xr-x   20 root     root     118784 Мар 30 10:41 dev
 164161 drwxr-xr-x   55 root     root       4096 Мар 30 10:47 etc
 525313 drwxr-xr-x    2 root     root       4096 Мар 30 11:42 home
 541729 drwxr-xr-x    2 root     root       4096 Апр 29  2003 initrd
 558145 drwxr-xr-x    9 root     root       4096 Мар 18 17:50 lib
     11 drwx------    2 root     root      16384 Мар 18 17:22
lost+found
 230275 drwxr-xr-x    2 root     root       4096 Апр 29  2003 misc
 590977 drwxr-xr-x    5 root     root       4096 Мар 18 14:06 mnt
 607393 drwxr-xr-x    2 root     root       4096 Апр 29  2003 opt
      1 dr-xr-xr-x   69 root     root          0 Мар 30  2004 proc
 180577 drwxr-x---   17 root     root       4096 Мар 30 11:25 root
 640225 drwxr-xr-x    2 root     root       8192 Мар 18 17:54 sbin
 213409 drwxrwxrwt    9 root     root       4096 Мар 30 11:24 tmp
 229825 drwxr-xr-x   15 root     root       4096 Мар 18 17:30 usr
  32833 drwxr-xr-x   17 root     root       4096 Мар 18 17:36 var
```

Рис. 4.1. Информация об основных каталогах файловой системы Linux

Следует также обратить внимание на размер большинства каталогов. Его минимальное значение – 4 Кб, что соответствует стандартному размеру одного логического блока. Поскольку каталог по существу является таблицей соответствия имен файлов и их индексных дескрипторов, подавляющее большинство каталогов вполне вмещаются в этот объем. Лишь три каталога в рассматриваемом примере имеют большие размеры: для каталога **/lost+found** (потерянные и найденные) зарезервировано 16 Кб – на тот случай, если при проверке файловой системы будет обнаружено большое количество испорченных файлов, каталог **/sbin** содержит большое количество утилит, а каталог **/dev** вмещает очень большое число специальных файлов. Каталог **/proc**, имеющий нулевой размер, является псевдокаталогом, он расположен в оперативной памяти и места на дисковом пространстве не занимает.

Следует обратить внимание на большое число жестких ссылок на каталоги. Так, у каталога **/proc** в приведенном примере 69 имен! С учетом того, что пользователям запрещено создание жестких ссылок на каталоги, это выглядит странно. Ответ можно найти, воспользовавшись командой **ls -la /**. Она отображает в двух верхних строках списка скрытую ссылку каталога на свое же имя, обозначаемую как «**.**», и ссылку на родительский каталог, обозначаемую как «**..**». Даже если в каталоге нет подкаталогов,

эти две жесткие ссылки все равно будут существовать и отображаться. Шестьдесят девять имен у каталога **/proc** в рассмотренном примере – это ссылка каталога на себя, ссылка на родительский (корневой) каталог и 67 подкаталогов, каждый из которых ссылается на родительский каталог **/proc**. Более подробно структура файловых записей в каталоге будет рассмотрена ниже.

Блоки объединяются в группы блоков. Группы блоков в файловой системе и блоки внутри группы нумеруются последовательно, начиная с единицы (рис. 4.2). Первый блок на диске имеет номер 0 и принадлежит группе с номером 1. Полная группа содержит 32768 блоков. Последняя группа блоков может быть неполной. Начало каждой группы блоков имеет адрес, который может быть получен как (номер группы – 1)* (число блоков в группе).

| **Загрузчик ФС** | **Группа блоков 1** | **Группа блоков 2** | ……. | **Группа блоков N** |
|---|---|---|---|---|

Рис. 4.2. Группы блоков на логическом разделе Linux

Первые 1024 байта логического раздела Linux отведены на размещение загрузчика LILO или GRUB, и при размере блока в 1 Кб загрузчик занимает полный блок. Каждая группа блоков имеет одинаковое строение. Ее структура представлена на рис. 4.3.

| Суперблок (дублируется) |
|---|
| Описатели групп блоков (дублируются) |
| Битовая карта блоков |
| Битовая карта индексных дескрипторов |
| Таблица индексных дескрипторов |
| Область блоков данных |

Рис. 4.3. Структура группы блоков

Суперблок является начальной точкой файловой системы. Он имеет размер 1024 байта, но нужной информацией заполнен всего на четверть – остальная часть суперблока дополняется нулями. Что касается остатка логического блока при его размере в 4 кБ, то он может быть заполнен «мусором» или использоваться в качестве стеганографического контейнера. Наличие копий суперблока в некоторых группах объясняется чрезвычайной важностью этого элемента файловой системы. Дубликаты суперблока используются при восстановлении файловой системы после сбоев. Тем не

менее некоторые системные утилиты (например, **`mount`** – утилита монтирования файловой системы) не умеют использовать резервные копии и при повреждении первого экземпляра суперблока сообщают об ошибке.

Информация, хранимая в суперблоке, используется для организации доступа к остальным данным на диске. В суперблоке определяется размер файловой системы, максимальное число файлов в разделе, объем свободного пространства и содержится много иной важной информации. При запуске ОС суперблок копируется в оперативную память, и все изменения файловой системы записываются на диск только периодически. Это позволяет повысить производительность системы, так как многие пользователи и процессы постоянно обновляют файлы. В ходе лабораторных работ нетрудно будет убедиться в том, что измененные и даже удаленные файлы продолжают существовать на диске в своем прежнем виде еще длительное время. При выключении системы суперблок обязательно должен быть записан на диск. При внезапном выключении питающего напряжения в структуре файловой системы на диске возникнут несоответствия, что приведет к запуску программы **`fsck`** при очередной загрузке компьютера. Суперблок имеет структуру, изображенную в табл. 4.1.

Таблица 4.1

| **Размер поля, байт** | **Смещение от начала блока, байт** | | **Назначение** |
|---|---|---|---|
| 4 | 0 | 0 | Число индексных дескрипторов в файловой системе |
| 4 | 4 | 4h | Число блоков в файловой системе |
| 4 | 8 | 8h | Число зарезервированных блоков |
| 4 | 12 | Ch | Число свободных блоков |
| 4 | 16 | 10h | Число свободных индексных дескрипторов |
| 4 | 20 | 14h | Номер первого блока, содержащего данные |
| 4 | 24 | 18h | Индикатор размера логического блока: 0 = 1 Кб; 1 = 2 Кб; 2 = 4 Кб |
| 4 | 28 | 1Ch | Индикатор размера фрагментов (если фрагментация блоков предусмотрена) |
| 4 | 32 | 20h | Число блоков в каждой группе блоков |
| 4 | 36 | 24h | Число фрагментов в каждой группе блоков |
| 4 | 40 | 28h | Число индексных дескрипторов в каждой группе блоков |
| 4 | 44 | 2Ch | Время последнего монтирования файловой системы (в секундах с полуночи 1 января 1970 года) |
| 4 | 48 | 30h | Время последней записи в файловую систему |
| 2 | 52 | 34h | Число монтирований файловой системы. Если этот счетчик достигает значения, указанного в следующем поле, файловая система при перезапуске проверяется, а счетчик обнуляется |
| 2 | 54 | 36h | Предельное число монтирований файловой системы |
| 2 | 56 | 38h | «Магическое число» (0xEF53), указывающее, что файловая система принадлежит к ex2fs или ext3fs |

Окончание табл. 4.1

| Размер поля, байт | Смещение от начала блока, байт | | Назначение |
|---|---|---|---|
| 2 | 58 | 3Ah | Флаги, указывающие текущее состояние файловой системы (1 – проверенная файловая система, 2 – ФС содержит ошибки, 4 – обнаружены зависшие узлы) |
| 2 | 60 | 3Ch | Реагирование на сообщение об ошибках в суперблоке (1 – продолжение работы, 2 – перемонтирование в режиме для чтения, 3 – паника системы) |
| 2 | 62 | 3Eh | Дополнительная версия |
| 4 | 64 | 40h | Время последней проверки файловой системы |
| 4 | 68 | 44h | Максимальный период времени между принудительными проверками файловой системы |
| 4 | 72 | 48h | Указание на тип операционной системы, в которой создана файловая система |
| 4 | 76 | 4Ch | Основная версия файловой системы |
| 2 | 80 | 50h | UID пользователя, которому разрешено использовать зарезервированные блоки |
| 2 | 82 | 52h | GID группы, которой разрешено использовать зарезервированные блоки |
| 4 | 84 | 54h | Первый незарезервированный индексный дескриптор |
| 2 | 88 | 58h | Размер индексного дескриптора (00 80h или 0100h) |
| 2 | 90 | 5Ah | Группа блоков, в которую входит данный суперблок (для резервных копий) |
| 4 | 92 | 5Ch | Флаги совместимых функций |
| 4 | 96 | 60h | Флаги несовместимых функций |
| 4 | 100 | 64h | Флаги совместимых функций только для чтения |
| 16 | 104 | 68h | Идентификатор файловой системы |
| 16 | 120 | 78h | Имя тома |
| 64 | 136 | 88h | Путь последнего монтирования |
| 4 | 200 | C8h | Битовая карта использования алгоритма |
| 1 | 204 | CCh | Количество блоков, предварительно выделяемых файлам |
| 1 | 205 | CDh | Количество блоков, предварительно выделяемых каталогам |
| 2 | 206 | CEh | Не используется |
| 16 | 208 | D0h | Идентификатор журнала |
| 4 | 224 | E0h | Индексный дескриптор журнала |
| 4 | 228 | E4h | Журнальное устройство (в случае использования внешнего журнала) |
| 4 | 232 | E8h | Начало списка «зависших» индексных дескрипторов |
|  | 235 | ECh | Заполнение до 1024 байта |

Содержание некоторых полей нуждается в разъяснении. Обычно для журнала транзакций в файловой системе **`ext3fs`** отводится один большой файл в пределах основного раздела. Но в целях защищенного резервирования на сервере может быть предусмотрен и внешний журнал.

Файл логически стирается при удалении его последнего имени (жесткой ссылки). Но если к этому времени файл будет открыт каким-нибудь процессом, то удаления не происходит, а его индексный дескриптор поме-

щается в список «зависших» **inode**. При штатном завершении работы системы происходит освобождение открытых файлов, и те из них, которые объявлены удаленными, стираются. Автоматическая проверка файловой системы при загрузке выявляет файлы с неестественными характеристиками и сопровождается помещением их в каталог **/lost+found**. Поэтому после загрузки системы список «зависших» индексных дескрипторов в суперблоке должен быть пустым.

На рисунке 4.4 показано, как выглядит фрагмент суперблока при выводе информации с помощью команды блочного копирования **dd** с перенаправлением вывода в программу **xxd**, выводящую дамп памяти в шестнадцатеричном и символьном виде.

```
dd if=/dev/hda7 bs=1024 skip=1 count=1|dd bs=80
count=1|xxd
```

При отсутствии утилиты **xxd** можно воспользоваться командой **hexdump -C**.

```
           0  1  2  3  4  5  6  7    8  9  A  B  C  D  E  F
0x00000400 00 07 0E 00 00 00 1C 00 : 6F 66 01 00 F0 47 13 00  ........of...G..
0x00000410 D8 31 0C 00 00 00 00 00 : 02 00 00 00 02 00 00 00  .1..............
0x00000420 00 80 00 00 00 80 00 00 : 20 40 00 00 4A FA 68 40  ........ @..J.h@
0x00000430 4A FA 68 40 0B 00 27 00 : 53 EF 01 00 01 00 00 00  J.h@..'.S.......
0x00000440 73 94 59 40 00 4E ED 00 : 00 00 00 00 01 00 00 00  s.Y@.N..........
```

Рис. 4.4. Содержимое первых 80 (50h) байтов суперблока

Подчеркиванием выделены одинарные и двойные слова, поименованные в таблице. Шестнадцатеричные числа в приводимых здесь и далее дампах памяти представлены в обратном формате (т. е. читаются в обратном порядке, справа налево). Так, максимально возможное число файлов в системе представлено первыми четырьмя байтами суперблока **00 0E 07 00h**.

Для преобразования чисел из десятичной системы счисления в шестандцатеричную и обратно следует воспользоваться одной из программ-калькуляторов, встроенных в операционную среду и доступных в консоли или графической оболочке. В консольном режиме следует порекомендовать калькулятор **bc**, который удобнее запустить в отдельной консоли с правами обычного пользователя. После запуска команды и вывода приглашения следует ввести строку **ibase=16** и завершить ее нажатием **<Enter>**. После этого каждое введенное число будет интерпретироваться как шестнадцатеричное (символы A B C D E F в шестнадцатеричных числах должны быть заглавными) и выводиться как десятичное. Если необходимо преобразовывать числа из десятичных в шестнадцатеричные, следует указать **ibase=10** и **obase=16**.

Преобразовав шестнадцатеричное число **00 0E 07 00h**, получаем десятичный эквивалент в 919296 файлов (**inode**). Аналогичным путем

прочтем некоторые иные числа:

- число блоков в файловой системе **00 1C 00 00h** = 1835008, т. е. на каждый файл зарезервировано около 2 блоков, или 8 Кб (часть блоков при этом расходуется для размещения копий суперблоков, описателей групп блоков, битовых карт, таблиц **inode**);
- для администратора на случай переполнения отведенного пространства дисковой памяти зарезервировано **1666Fh** = 91759 блоков. Это около 10 % всего дискового пространства;
- на диске в данном логическом разделе свободно **1347F0h** = 1263600 блоков, или 4935,93 Мб;
- размер логического блока **1000h** = 4096 байтов;
- в каждой группе имеется **8000h** = 32768 блоков и **4020h** = 16416 индексных дескрипторов. Таким образом, для каждой группы блоков выделено по 128 Мб дискового пространства, на котором можно разместить 16416 файлов. Для размещения таблицы индексных дескрипторов система должна выделить в каждой группе 16416 : 32 = 513 блоков. Резервные суперблоки в случае повреждения первого можно искать в 32768, 65534 и последующих логических блоках (Б. Кэрриэ [6] утверждает, что копии супеблока размещаются не в каждой группе блоков, но указать алгоритм их размещения для произвольной системы Linux не представляется возможным).

Вслед за суперблоком в логическом блоке со следующим номером друг за другом расположены описатели групп блоков (**Group Descriptors**) размером 32 байта каждый. Каждый описатель представляет собой структуру со следующими полями (табл. 4.2).

Таблица 4.2

| Размер поля, байт | Смещение, байт | Назначение |
|---|---|---|
| 4 | 0 | Адрес блока, содержащего битовую карту блоков (block bitmap) данной группы |
| 4 | 4h | Адрес блока, содержащего битовую карту индексных дескрипторов (inode bitmap) данной группы |
| 4 | 8h | Адрес блока, содержащего таблицу индексных дескрипторов (inode table) данной группы |
| 2 | Ch | Число свободных блоков в данной группе |
| 2 | Eh | Число свободных индексных дескрипторов в данной группе |
| 2 | 10h | Число индексных дескрипторов в данной группе, которые являются каталогами |
| 14 | 12h | Заполнение |

Информация, хранимая в описании группы, позволяет найти битовые карты блоков и индексных дескрипторов, а также таблицу индексных деск-

рипторов. Если размер логического блока равен 4 Кб, дамп описателя первой группы блоков можно вывести на экран командой (рис. 4.5)

```
dd if=/dev/hda7 bs=4096 skip=1 count=1|dd bs=32 count=1
|xxd
```

```
          0  1  2  3  4  5  6  7    8  9  A  B  C  D  E  F
0x00000000  02 00 00 00 03 00 00 00 : 04 00 00 00 09 1A 14 40  ...............@
0x00000010  02 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
```

Рис. 4.5. Дамп описателя группы блоков

В описателе группы блоков нетрудно найти нужную информацию. Так, битовая карта блоков располагается в блоке 2 (**00 00 00 02h**), а битовая карта индексных дескрипторов – в блоке 3 (**00 00 00 03h**). Таблица индексных дескрипторов начинается с блока 4 (**00 00 00 04h**). В данной группе имеется еще 6665 (**1A09h**) свободных блоков и 16404 (**4014h**) свободных индексных дескриптора, которые могут позволить создать такое же количество файлов. В рассматриваемой группе, судя по представленной информации, всего 2 каталога.

С помощью отладчика файловой системы **debugfs** (справка о командах отладчика приведена в приложении) можно одновременно вывести информацию о суперблоке и группах блоков в более удобной форме. Для этого предлагается использовать команду (прил. 3)

**debugfs –R stats <dev>** ,

где вместо **<dev>** указывается конкретный файл блочного устройства, например **/dev/sda6**, который соответствует логическому разделу жесткого диска с файловой системой **ext*fs**. Результаты вывода приведены на рис. 4.6.

```
Filesystem volume name:   <none>
Last mounted on:          <not available>
Filesystem UUID:          0257eed4-e8f7-4366-89dc-8e0b56f3c258
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      ext_attr resize_inode dir_index filetype
sparse_super large_file
Default mount options:    (none)
Filesystem state:         not clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              438048
Block count:              1751077
Reserved block count:     87553
```

Рис. 4.6. Фрагмент листинга суперблока и групп блоков

```
Free blocks:              291996
Free inodes:              206685
First block:              0
Block size:               4096
Fragment size:            4096
Reserved GDT blocks:      427
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         8112
Inode blocks per group:   507
Filesystem created:       Tue Oct  7 17:28:31 2008
Last mount time:          Mon Nov  3 12:46:11 2008
Last write time:          Mon Nov  3 12:49:39 2008
Mount count:              3
Maximum mount count:      28
Last checked:             Mon Oct 27 19:39:22 2008
Check interval:           15552000 (6 months)
Next check after:         Sat Apr 25 20:39:22 2009
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:                 256
Default directory hash:   tea
Directory Hash Seed:      2588396c-13bb-4f22-8962-02fe457cd908
Directories:              15583

Group  0: block bitmap at 429, inode bitmap at 430, inode table at
431 0 free blocks, 2477 free inodes, 531 used directories
Group  1: block bitmap at 33197, inode bitmap at 33198, inode table
at 33199
3252 free blocks, 1301 free inodes, 796 used directories
Group  2: block bitmap at 65536, inode bitmap at 65537, inode table
at 65538
6909 free blocks, 467 free inodes, 616 used directories
Group  3: block bitmap at 98733, inode bitmap at 98734, inode table
at 98735
0 free blocks, 978 free inodes, 560 used directories
Group  4: block bitmap at 131072, inode bitmap at 131073, inode ta-
ble at 131074
4 free blocks, 4736 free inodes, 777 used directories
Group  5: block bitmap at 164269, inode bitmap at 164270, inode ta-
ble at 164271
287 free blocks, 1923 free inodes, 552 used directories
14677 free blocks, 2153 free inodes, 343 used directories
************************************************************
Group 51: block bitmap at 1671168, inode bitmap at 1671169, inode
table at 1671170
18140 free blocks, 2245 free inodes, 521 used directories
Group 52: block bitmap at 1703936, inode bitmap at 1703937, inode
table at 1703938
20528 free blocks, 2247 free inodes, 651 used directories
Group 53: block bitmap at 1736704, inode bitmap at 1736705, inode
table at 1736706
9131 free blocks, 2112 free inodes, 481 used directories
```

Рис. 4.6. Окончание

Выведенная информация, несомненно, более наглядна, чем шестнадцатеричный дамп. Из сведений о дисковом пространстве, выделенном каждой из групп блоков, можно узнать конкретный диапазон номеров индексных дескрипторов и логических блоков, что может быть использовано для дальнейших оценок и расчетов.

Битовая карта блоков (**block bitmap**) – это структура, в которой каждому логическому блоку соответствует один бит (рис. 4.7). Порядковый номер бита соответствует порядковому номеру блока. Если бит равен 1, то блок занят каким-либо файлом, если равен 0 – свободен. Эта карта служит для поиска свободных блоков в тех случаях, когда надо выделить место под файл. Вот как выглядит фрагмент 2-го логического блока. Байты **FF** в двоичном виде – это **1111 1111b** (все блоки заняты). Со смещения **24CCh** начинаются свободные блоки. Просматривая **block bitmap**, можно увидеть несколько чередующихся занятых и свободных полей блоков. Поскольку блоки идут в линейном порядке, это говорит о том, что система распределяет хранимую информацию по всему дисковому пространству.

Наличие в сплошном массиве байтов, отличных от **FF**, указывает на то, что в файловой системе присутствуют свободные блоки, ранее принадлежавшие каким-то файлам. При детальном изучении битовой карты можно сказать, сколько блоков занимал удаленный файл, и назвать порядковые номера этих блоков. К сожалению, эту работу простой не назовешь, но автоматическое восстановление удаленных файлов часто приводит к грубым просчетам и необратимой порче данных. О том, какие команды следует использовать для вывода информации о свободных блоках, будет сказано ниже.

```
0x00002460  FF FF FF FF FF FF FF FF : 3F FC 3F F0 FF FF FF FF  ........?.?.....
0x00002470  FF FF FF FF FF FF FF FF : FF FF FF FF FF FF FF FF  ................
0x00002480  FF FF FF FF FF FF FF FF : FF FF FF FF FF FF FF FF  ................
0x00002490  FF FF FF FF FF FF FF FF : FF FF FF FF FF FF FF FF  ................
0x000024A0  FF FF FF FF FF FF FF FF : FF FF FF FF FF FF FF FF  ................
0x000024B0  FF FF FF FF FF FF FF FF : FF FF FF FF FF FF FF FF  ................
0x000024C0  FF FF FF FF FF FF FF FF : FF FF FF 0F 00 00 00 00  ................
0x000024D0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000024E0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000024F0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x00002500  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x00002510  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x00002520  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x00002530  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
```

Рис. 4.7. Фрагмент битовой карты блоков

Битовая карта индексных дескрипторов имеет аналогичную структуру, но иную гранулярность: один бит соответствует 128-байтному фрагменту в таблице **inode** (в **ext4fs** размер **inode** увеличен до 256 байтов). Фрагмент битовой карты индексных дескрипторов приведен на рис. 4.8.

```
0x000037B0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000037C0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000037D0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000037E0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000037F0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x00003800  00 00 00 00 FF FF FF FF : FF FF FF FF FF FF FF FF  ................
0x00003810  FF FF FF FF FF FF FF FF : FF FF FF FF FF FF FF FF  ................
0x00003820  FF FF FF FF FF FF FF FF : FF FF FF FF FF FF FF FF  ................
0x00003830  FF FF FF FF FF FF FF FF : FF FF FF FF FF FF FF FF  ................
0x00003840  FF FF FF FF FF FF FF FF : FF FF FF FF FF FF FF FF  ................
0x00003850  FF FF FF FF FF FF FF FF : FF FF FF FF FF FF FF FF  ................
```

Рис. 4.8. Фрагмент битовой карты индексных дескрипторов

В приведенном фрагменте наблюдается иной порядок, чем в битовой карте блоков: вначале идут свободные индексные дескрипторы, затем занятые. По-видимому, в данной группе блоков свободные дескрипторы были зарезервированы в пространстве, выделенном какому-то каталогу.

Следующая область в структуре группы блоков служит для хранения таблицы индексных дескрипторов файлов. Каждому файлу на диске соответствует один индексный дескриптор файла, который идентифицируется своим порядковым номером – индексом файла. Индексные дескрипторы файлов данной группы блоков хранятся в логических блоках, расположенных следом за битовой картой индексных дескрипторов. При стандартном размере **inode** в 128 байтов один 4-килобайтный логический блок вмещает 32 дескриптора. Таким образом, 16416 индексных дескрипторов, приходящихся на одну группу, займут подряд 513 логических блоков. Такой расчет справедлив для файловой системы **ext2fs**, однако если на логическом разделе смонтирована файловая система **ext3fs**, информация, выводимая отладчиком **debugfs**, может оказаться неверна. В журнализируемой файловой системе **ext3fs** между битовой картой **inode** и таблицей индексных дескрипторов обычно размещается журнал транзакций, под который обычно отводится 8192 блока, или 32768 Мб. В таком случае в первой группе блоков таблица индексных дескрипторов будет начинаться не с 4-го, а с 8195-го логического блока. Правильные адреса в любом случае находятся в описателе группы блоков.

Индексный дескриптор файла в **ext2fs** и **ext3fs** имеет объем 128 байтов и структуру, изображенную в табл. 4.3.

Нетрудно заметить, что некоторые поля фрагментированы. Проектировщиков первых систем часто упрекают в непредусмотрительности, однако человеку действительно трудно увидеть будущее и предусмотреть грядущие потребности. Так, 4-байтовые поля, содержащие временные отметки файлов, переполнятся в январе 2038 года. Возможно, авторы первых систем UNIX рассчитывали на продолжительность только своего века и никак не предполагали, что их творение прослужит так долго.

Таблица 4.3

| Размер поля, байт | Смещение, байт | Описание |
|---|---|---|
| 2 | 0 | Тип файла и права доступа к нему |
| 2 | 2h | Идентификатор владельца файла UID (младшие разряды) |
| 4 | 4h | Размер файла в байтах (младшие разряды) |
| 4 | 8h | Время последнего обращения к файлу |
| 4 | Ch | Время создания файла. |
| 4 | 10h | Время последней модификации файла |
| 4 | 14h | Время удаления файла |
| 2 | 18h | Идентификатор группы GID (младшие разряды) |
| 2 | 1Ah | Счетчик числа связей («жестких» ссылок на файл) |
| 4 | 1Ch | Число секторов по 512 байт, занимаемых файлом |
| 4 | 20h | Флаги файла |
| 4 | 24h | Зарезервировано для ОС |
| 15x4 | 28h | Указатели на блоки, в которых содержатся данные файла |
| 4 | 64h | Версия файла (для NFS) |
| 4 | 68h | Блок расширенных атрибутов (ACL файла) |
| 4 | 6Ch | ACL каталога или старшие разряды размера файла |
| 4 | 70h | Адрес фрагмента |
| 1 | 74h | Номер фрагмента |
| 1 | 75h | Размер фрагмента |
| 2 | 76h | Не используется |
| 2 | 78h | Идентификатор владельца файла UID (старшие разряды) |
| 2 | 7Ah | Идентификатор группы GID (старшие разряды) |
| 2 | 7Ch | Не используется |

В файловой системе **ext4fs** размер индексного дескриптора увеличен до 256 байтов. Конкретный размер **inode** в шестнадцатеричном виде хранится в двухбайтовом поле суперблока по смещению **58h**. Но как разумно распорядиться дополнительным пространством, обеспечив при этом совместимость с прежними версиями ФС, разработчики еще не решили.

Таблицу индексных дескрипторов можно вывести поблочно и поэлементно с помощью конвейера команд:

```
dd if=/dev/hda7 bs=4096 skip=4 count=1|dd bs=128
skip=1 count=1|xxd
```

Задавая номер логического блока и смещение от его начала, мы можем вывести для просмотра или скопировать в файл любой нужный нам дескриптор. Вышеприведенной командой на экран выводится дамп 2-го индексного дескриптора, содержащего метаданные корневого каталога. Некоторое неудобство заключается в том, что в описателе группы адреса являются шестнадцатеричными числами, а программа **dd** «понимает» только десятичные числа.

Произвольно взятый фрагмент из двух индексных дескрипторов представлен для анализа на рис. 4.9. Размер каждого из **inode** в данном примере составляет 128 байтов, следовательно, интервал между ними – 80h.

```
0x02003A00  ED 41 00 00 00 10 00 00 : DC 09 69 40 0C 96 59 40  .A.......i..Y@
0x02003A10  0C 96 59 40 00 00 00 00 : 00 00 02 00 08 00 00 00  ..Y@............
0x02003A20  00 00 00 00 00 00 00 00 : 1B 82 06 00 00 00 00 00  ................
0x02003A30  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x02003A40  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x02003A50  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x02003A60  00 00 00 00 9C 79 17 E5 : 00 00 00 00 00 00 00 00  .....y..........
0x02003A70  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x02003A80  A4 81 00 00 37 00 00 00 : 1A 96 59 40 0C 96 59 40  ....7.....Y@..Y@
0x02003A90  12 6E A8 3E 00 00 00 00 : 00 00 01 00 08 00 00 00  .n.>............
0x02003AA0  00 00 00 00 00 00 00 00 : 1C 82 06 00 00 00 00 00  ................
0x02003AB0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x02003AC0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x02003AD0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
```

Рис. 4.9. Фрагмент таблицы индексных дескрипторов

Индексный дескриптор определяет порядок доступа к конкретному файлу, поэтому, модифицируя эту запись, можно произвольно манипулировать его атрибутами. Не обязательно редактировать **inode** прямо в таблице индексных дескрипторов – нужную запись нелегко идентифицировать с номером **inode**, и для этого удобнее использовать одну из команд отладчика **debugfs**.

Произведем разбор байтов в представленных фрагментах. Первые два байта несут очень важную информацию о типе файла и правах доступа к нему. Как уже было отмечено выше, в ОС Linux может быть 7 типов файлов. Информация о типе файла расположена в старшем нибле (октете) старшего байта (табл. 4.4).

Таблица 4.4

| **Тип файла** | **Обозначение типа файла в листинге, выводимом командой `ls -l`** | **Шестнадцатеричный символ** |
|---|---|---|
| Обычный файл | **-** | **8** |
| Каталог | **d** | **4** |
| Символическая ссылка | **l** | **C** |
| Сокет | **s** | **A** |
| Именованный канал | **p** | **1** |
| Файл блочного устройства | **b** | **6** |
| Файл символьного устройства | **c** | **2** |

Права доступа к файлу отображаются 12 двоичными разрядами, состоящими из 4 полей по 3 разряда в каждом. Для удобства интерпретации данных шестнадцатеричные числа следует преобразовать в двоичную форму, а затем биты сгруппировать удобным образом.

Двоичные разряды старшей тройки обозначают дополнительные (эффективные) права на файл. Они условно обозначаются: **SUID** – бит, позволяющий любому запускать исполняемый файл с правами его владельца (действует лишь в отношении бинарных исполняемых файлов), **SGID** – очень редко используемый бит, позволяющий запустить процесс с правами группы владельца, **Sticky bit** – «флажок», не позволяющий пользователю удалять из каталога файлы, ему не принадлежащие.

Все оставшиеся тройки интерпретируются одинаково: левый бит – право на чтение, средний – на запись, правый – на исполнение. Левая тройка – права владельца, средняя тройка – права пользователей из его группы, правая тройка – права для остальных зарегистрированных пользователей.

Так, в первых двух байтах первого **inode** на рис. 4.9 записано шестнадцатеричное число **41 EDh**. По цифре 4 в старшем разряде мы определяем, что перед нами – каталог. Оставшуюся часть числа преобразуем в двоичную форму, для наглядности группируя биты по три разряда:

**1EDh = 0001 1110 1101b = 000 111 101 101**

При определенном навыке читать права доступа можно и без подобного преобразования. Читаем:

- эффективных прав доступа к каталогу не установлено: **000**,
- владелец имеет на свой каталог полные права: **111 = rwx**,
- члены группы владельца имеют права на чтение и исполнение: **101 = r-x**,
- все остальные зарегистрированные пользователи также имеют права на чтение и исполнение (что вполне естественно для каталога общего назначения).

Следующее слово является младшими разрядами идентификатора владельца файла **Owner UID**. На это поле выделено два байта, чего с избытком хватает для регистрации любого разумного числа пользователей. Два старших байта **UID**, размещенные в другой ячейке, обычно пустуют. Только администратор системы имеет нулевой идентификатор. По числу **00 00h** убеждаемся, что владельцем каталога именно он и является.

Следующее двойное слово содержит младшие разряды размера файла в байтах. Старшие 4 байта, как видно из табл. 4.3, размещены в другой ячейке **inode** и обычно пустуют. **00 00 10 00h** = 4096 байтов – обычный при данном размере логического блока объем каталога.

Четыре поля по 4 байта в каждом отображают временные отметки файла:

- последнего обращения,
- последней записи в индексный дескриптор,
- последней модификации данных в файле,
- удаления.

Все поля, содержащие временные отметки, содержат число секунд, прошедших с полуночи 1 января 1970 года (как выражаются почитатели UNIX – с начала Эпохи). Чтобы узнать это время, нужно вначале преобразовать число в десятичное, затем пересчитать секунды в дату и время. Утилиты для преобразования в обычное представление даты и времени в штатной инсталляции операционной системы нет, но зато имеется системная функция, которую можно вызвать из исполняемого файла или сценария, написанного на языке Perl.

Для отображения различных сведений о файле, включая его временные отметки, предназначена утилита **stat**. Точнее, она выдает сведения об еще не удаленных файлах, поэтому времени логического удаления файла она не показывает:

**stat -c %x,%X abc** – время последнего доступа,
**stat -c %y,%Y abc** – время последней модификации,
**stat -c %z,%Z abc** – время создания.

Приведенные команды выводят временные отметки файла **abc** из текущего каталога вначале в секундах от начала «Эпохи», затем в обычном формате. Команда в коротком формате **stat abc** выведет всю информацию о файле **abc**, но временные отметки будут представлены в обычном виде.

Разработчики системы, предусмотрев в индексном дескрипторе файла время его удаления, предполагали, что удаление файла не сопровождается физическим стиранием данных и после удаления файла его **inode** какое-то время будет существовать. Это предположение справедливо только в отношении файловой системы **ext2fs**. Как будет показано ниже, удаление файла в этой ФС действительно не сопровождается ни удалением **inode**, ни стиранием информации в блоках, выделенных файлу. Удаление файла в **ext2fs** – это лишь стирание его последнего имени («жесткой» ссылки) в каталоге. Странная дата удаления всех «живых» файлов – 1 января 1970 года – указывает на то, что счетчик секунд в этом поле равен нулю.

По порядку расшифровываем следующие поля. Идентификатор группы имеет значение, равное **00 00h**, – это группа **root** (суперпользователя). На каталог имеется две жесткие ссылки (**00 02h**), что означает отсутствие внутри него подкаталогов. Каталог занимает 8 секторов по 512 байтов на диске, что соответствует размеру одного блока в 4096 байтов. Дополнительные права доступа (флаги) не устанавливались. Единственный логический блок, в котором хранится таблица соответствия между именами файлов данного каталога и их индексными дескрипторами, имеет порядковый номер **00 06 82 1Bh**. В этом блоке можно прочитать имена и **inode** файлов этого каталога.

Аналогично определим параметры другого файла, приведенные на рис. 4.9. Первая шестнадцатеричная цифра слова **81 A4h** указывает на то,

что это обычный файл, а двоичная комбинация

**1A 4h = 0001 1010 0100b = 000 110 100 100**

дает всю необходимую информацию о правах доступа:

- эффективные права доступа не установлены,
- владелец файла имеет право на чтение и запись,
- члены его группы и остальные пользователи имеют право на чтение,

Владельцем файла является **root** – администратор. Файл имеет размер **37h** = 55 байтов и так далее. По информации, содержащейся в **inode**, уже нетрудно найти и идентифицировать файл. Например, можно попытаться обнаружить файл по совокупности характеристик: типу, объему, правам доступа, временным отметкам:

```
find / -type f –a –size 12320c –a –perm 0644 –ctime ....
```

Это можно сделать еще проще, зная порядковый номер просматриваемого индексного дескриптора и группу блоков, в которой находится таблица **inode**. Вычислив порядковый номер индексного дескриптора, имя файла можно найти с помощью команды

**find / -inum 123456**,

где 123456 – порядковый номер **inode**.

В более удобной для анализа форме вывести информацию об индексном дескрипторе можно, воспользовавшись для этого утилитой **lde** (linux disk editor). К сожалению, дисковый редактор неправильно отображает 256-байтные inode современных ФС. На рис. 4.10 приведен пример вывода результатов этой команды для индексного дескриптора каталога **/bin**.

```
lde –i 131329 /dev/hda5

INODE: 131329 (0x00020101)
drwxr-xr-x      0        0             4096 Thu Mar 18
17:42:58 2004
TYPE:                   directory
LINKS:                  2
MODEFLAGS.MODE:         004.0755
SIZE:                   4096
BLOCK COUNT:            8
UID:                    00000
GID:                    00000
ACCESS TIME:            Tue Mar 30 11:46:37 2004
CREATION TIME:          Thu Mar 18 17:42:58 2004
MODIFICATION TIME:      Thu Mar 18 17:42:58 2004
DELETION TIME:          Thu Jan  1 00:00:00 1970
DIRECT BLOCKS:          0x00040203

INDIRECT BLOCK:
DOUBLE INDIRECT BLOCK:
TRIPLE INDIRECT BLOCK:
```

Рис. 4.10. Информация об **inode** каталога **/bin**, выведенная редактором **lde**

Ниже для сравнения приведена запись об этом же каталоге, выведенная командой **ls –ali /bin**. Утилита **ls** для вывода этой информации также обращается к таблице индексных дескрипторов.

```
131329 drwxr-xr-x    2 root     root         4096 Map 18 17:42
```

Еще раз обратим внимание на временные отметки файла на рис. 4.10. Время удаления файла, датируемое 1970 годом, – не временной парадокс, а единая точка отсчета всех 4 временных отметок. Если она не указывает правдоподобное время, то файл еще не удалялся.

Система адресации данных – это одна из самых существенных составных частей файловой системы. Всего в **inode** для целей адресации зарезервировано 15 полей по 4 байта. Номера первых 12 блоков хранятся непосредственно в **inode**; их еще иногда называют блоками с прямой адресацией (**direct blocks**). При размере логического блока 4 Кб таким образом можно создавать файлы размером до 4 х 12 = 48 Кб.

При большем объеме файла используется нелинейная система адресации данных. Очередное поле содержит адрес (номер) блока, в котором хранятся номера еще 256 блоков данных. Его называют блоком косвенной адресации (**indirect block**).

Если файл все же не помещается в пространство 256 х 4 + 48 = 1072 Кб, очередное поле **inode** указывает номер блока, в котором хранятся 256 номеров блоков косвенной адресации. Этот блок называют блоком двойной косвенной адресации (**double indirect block**). Наконец, если и этого пространства для размещения файла недостаточно, последнее поле адресует номер блока, в котором хранятся 256 номеров блоков двойной косвенной адресации. Его называют блоком тройной косвенной адресации (**triply indirect block**).

При логическом удалении файла и последующем использовании освободившихся блоков проще всего найти блоки прямой адресации (разумеется, если индексный дескриптор к этому времени не пострадал). Но если будет повторно использован один из блоков косвенной адресации, то все номера блоков, на которые он указывал, будут потеряны. Варианты восстановления данных в логически удаленных или поврежденных файлах будут рассмотрены ниже.

На рис. 4.11 приведен **inode**, а на рис. 4.12 – фрагмент блока данных файла, содержащего сценарий (командный файл). Для этого файла был установлен дополнительный бит **SUID**, а также дополнительные атрибуты, предписывающие блокирование любых изменений файла, его автоматическое сжатие и декомпрессию при записи/чтении, а также гарантированное стирание блоков данных при удалении файла. Установка дополнительных атрибутов производилась командой

```
chattr +ics file_name
```

```
INODE: 527744 (0x00080D80)
-rwsr-xr-x  0        0          69 Sat Apr  3 13:19:39 2004
TYPE:                   regular file
LINKS:                  1
MODEFLAGS.MODE:         010.4755
SIZE:                   69
BLOCK COUNT:            8
UID:                    00000
GID:                    00000
ACCESS TIME:            Sat Apr  3 13:19:39 2004
CREATION TIME:          Sat Apr  3 13:23:24 2004
MODIFICATION TIME:      Sat Apr  3 13:19:39 2004
DELETION TIME:          Thu Jan  1 05:00:00 1970
DIRECT BLOCKS:          0x001024F9

INDIRECT BLOCK:
DOUBLE INDIRECT BLOCK:
TRIPLE INDIRECT BLOCK:
```

Рис. 4.11. Информация о метаданных файла-сценария

```
0x024F9000  64 64 20 69 66 3D 2F 64 : 65 76 2F 66 64 30 20 6F  dd if=/dev/fd0 o
0x024F9010  66 3D 2F 68 6F 6D 65 2F : 66 6C 6F 70 70 79 5F 61  f=/home/floppy_a
0x024F9020  20 73 6B 69 70 3D 32 30 : 20 63 6F 75 6E 74 3D 31   skip=20 count=1
0x024F9030  30 30 20 63 6F 6E 76 3D : 6E 6F 65 72 72 6F 72 2C  00 conv=noerror,
0x024F9040  73 79 6E 63 0A 00 00 00 : 00 00 00 00 00 00 00 00  sync............
0x024F9050  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x024F9060  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x024F9070  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x024F9080  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
```

Рис. 4.12. Фрагмент блока данных файла-сценария

Однако в символьном виде **inode** (см. рис. 4.11) дополнительные атрибуты не отображаются. Их можно увидеть лишь в соответствующих полях таблицы индексных дескрипторов. В этом нет ничего странного: операционные системы непрерывно развиваются, и дополнительные атрибуты файлов появились несколько позже утилит, отображающих информацию о файлах.

Осталось рассмотреть, где и как размещается третий компонент файла – его символьное имя. У каждого **inode** может быть 65536 «жестких» ссылок, т. е. символьных имен. Ассоциативные связи между именами файлов и их индексными дескрипторами устанавливаются с помощью записей в каталогах, которые представляют собой разновидность текстовых файлов. Рассмотрим, как выглядит листинг одного из каталогов. На рис. 4.13 представлен фрагмент распечатки расширенных сведений о файлах, находящихся в каталоге **/bin**.

В первой строке приведена запись о самом каталоге **/bin** (он обозначен точкой). Двумя точками обозначается родительский, в данном случае корневой каталог. Расширенная информация содержит индексные дескрипторы, тип файлов и права доступа к ним, количество жестких ссылок

(имен) файлов, владельца и его группу, размер файла, дату и время его создания и, наконец, само имя файла.

```
131329 drwxr-xr-x    2 root     root         4096 Мар 18 17:42 .
     2 drwxr-xr-x   19 root     root         4096 Апр  6 12:53 ..
131559 -rwxr-xr-x    1 root     root         4594 Апр 25  2003 arch
131542 lrwxrwxrwx    1 root     root            4 Мар 18 17:29 awk -> gawk
131509 -rwxr-xr-x    1 root     root        15643 Май  4  2003 basename
131333 -rwxr-xr-x    1 root     root       626028 Апр 26  2003 bash
131510 -rwxr-xr-x    1 root     root        19812 Май  4  2003 cat
131334 lrwxrwxrwx    1 root     root            4 Мар 18 17:29 sh -> bash
131512 -rwxr-xr-x    1 root     root        23999 Май  4  2003 chmod
131513 -rwxr-xr-x    1 root     root        26124 Май  4  2003 chown
131514 -rwxr-xr-x    1 root     root        57792 Май  4  2003 cp
131957 -rwxr-xr-x    1 root     root        63871 Апр 29  2003 cpio
131515 -rwxr-xr-x    1 root     root        26305 Май  4  2003 cut
131516 -rwxr-xr-x    1 root     root        45838 Май  4  2003 date
131517 -rwxr-xr-x    1 root     root        35496 Май  4  2003 dd
131518 -rwxr-xr-x    1 root     root        38648 Май  4  2003 df
131560 -rwxr-xr-x    1 root     root         6537 Апр 25  2003 dmesg
131552 lrwxrwxrwx    1 root     root            8 Мар 18 17:30 domainname
```

Рис. 4.13. Фрагмент каталога **/bin**, отображенный командой **ls –ali**

Каталог, по сути, представляет собой таблицу, каждая строка (запись) которой имеет переменную длину и состоит из 5 полей:

- индексный дескриптор файла длиной 4 байта,
- длина записи 2 байта,
- длина имени файла 1 байт,
- тип файла длиной 1 байт. В отличие от соответствующего байта в индексном дескрипторе обычный файл здесь обозначается цифрой 1, каталог – 2, символическая ссылка – 7,
- имя файла в ASCII-кодировке. Имя файла имеет переменную длину, дополненную нулями до 4–байтной границы.

На рис. 4.14 представлен фрагмент блока данных каталога **/bin**.

```
0x40203000  01 01 02 00 0C 00 01 02 : 2E 00 00 00 02 00 00 00  ................
0x40203010  0C 00 02 02 2E 2E 00 00 : 06 01 02 00 0C 00 02 07  ................
0x40203020  73 68 00 00 05 01 02 00 : 0C 00 04 01 62 61 73 68  sh..........bash
0x40203030  B5 01 02 00 18 00 08 01 : 62 61 73 65 6E 61 6D 65  ........basename
0x40203040  39 39 35 66 34 34 34 00 : 04 01 02 00 10 00 06 01  995f444.........
0x40203050  6D 6B 74 65 6D 70 00 00 : 03 01 02 00 10 00 05 07  mktemp..........
0x40203060  62 61 73 68 32 00 00 00 : B2 01 02 00 10 00 05 01  bash2...........
0x40203070  65 67 72 65 70 00 00 00 : B3 01 02 00 10 00 05 01  egrep...........
0x40203080  66 67 72 65 70 00 00 00 : B4 01 02 00 0C 00 04 01  fgrep...........
0x40203090  67 72 65 70 B7 01 02 00 : 14 00 05 01 63 68 67 72  grep........chgr
0x402030A0  70 30 35 39 39 35 66 34 : B6 01 02 00 0C 00 03 01  p05995f4........
0x402030B0  63 61 74 00 E0 01 02 00 : 18 00 0D 07 64 6E 73 64  cat.........dnsd
0x402030C0  6F 6D 61 69 6E 6E 61 6D : 65 66 34 00 B8 01 02 00  omainnamef4.....
0x402030D0  10 00 05 01 63 68 6D 6F : 64 00 00 00 B9 01 02 00  ....chmod.......
0x402030E0  10 00 05 01 63 68 6F 77 : 6E 00 00 00 BA 01 02 00  ....chown.......
0x402030F0  0C 00 02 01 63 70 00 00 : BB 01 02 00 0C 00 03 01  ....cp..........
```

Рис. 4.14. Дамп блока данных, содержащего каталог **/bin**

Проанализируем информацию, содержащуюся в нескольких начальных записях.

Первые 4 байта записи 1 дают число **00 02 01 01h** = 131329, что совпадает с номером **inode** каталога **/bin**. Следующие два байта **00 0Ch** = 12 определяют длину записи в байтах. Третье поле содержит байт **01h** = 1, указывая на то, что имя каталога состоит из одного символа. Действительно, имя каталога обозначено одним символом – точкой. Четвертое поле – **02h** = 2 – указывает на то, что это – каталог. Точке, которой обозначено имя файла, соответствует байт **2Eh**. Оставшиеся три байта являются нулями и представляют собой дополнение до 4-байтной границы.

Запись 2 начинается 4-байтным числом **00 00 00 02h** = 2, что является номером индексного дескриптора родительского (корневого) каталога. Длина этой записи также равна **00 0Ch** = 12 байт. Третье поле: байт **02h** – дает нам длину имени каталога (имя родительского каталога – две точки). Четвертое поле **02h** – тип файла (каталог). Имя файла: две точки – соответствует двум байтам **2E 2Eh**. Еще два байта заполнения **00 00h** завершают эту запись.

Индексный дескриптор файла в третьей записи **00 02 01 06h** = 131334, что соответствует **inode** символической ссылки **sh** на командный интерпретатор **bash** (третья строка сверху на рис. 4.14). Имя этого файла тоже короткое, и длина записи по-прежнему составляет 12 байт. Третье поле: байт **02h** определяет длину имени файла. Тип файла (четвертое поле) равен **07h**, что соответствует символической ссылке. Наконец, байты **68 73h** последнего поля в ASCII-кодировке соответствуют символам имени файла **sh**. Приведенных примеров вполне достаточно, чтобы произвести разбор любой записи с любого ее конца.

Как уже указывалось, большинство каталогов по своему реальному объему вмещаются в один логический блок. Однако, если этого не хватает, система выделяет каталогу столько блоков, сколько необходимо. Отдельная запись в каталоге не может пересекать границу блока (то есть должна быть расположена целиком внутри одного блока). Поэтому, если очередная запись не помещается целиком в данном блоке, она переносится в следующий блок, а предыдущая запись продолжается нулями, чтобы они заполнили блок до конца.

## 4.2. Временные отметки файлов

Временные отметки (ВО) обычно фиксируются в компьютерной памяти в виде 4-байтных полей, содержащих число секунд с новогодней полуночи 1970 года. В индексном дескрипторе файла сохраняются 4 временные отметки [6]:

- время **C** (Create) – время последнего изменения индексного

дескриптора. Судя по справочным данным о системных вызовах (см. ниже), это значение обновляется при создании файла и записи в существующий файл, при изменении прав владения и доступа, а также изменении количества имен файла (прямых ссылок на `inode`). Для каталога и символической ссылки действуют эти же правила;

- время **A** (Access) – время последнего открытия файла. Как указано в [6], эта временная отметка обновляется при создании или чтении файла. Но по отношению к каталогам это не вполне верно. Время **A** также может обновляться или произвольно устанавливаться с помощью системного вызова `utime();`
- время **M** (Modify) – время последней модификации файла или каталога, хотя справедливее назвать его временем последнего сохранения файла. Время **M** устанавливается при создании файла и при записи в существующий файл или каталог, а также при повторной записи в файл прежних данных. Время **M** также может обновляться или произвольно устанавливаться с помощью системного вызова `utime();`
- время **D** (Delete) – время удаления файла (иначе – объявления его `inode` и занимаемого им дискового пространства свободными). В файловых системах `ext3fs` и `ext4fs` при удалении файла происходит стирание его индексного дескриптора, включая все временные отметки, поэтому использование этой временной отметки выглядит проблематично.

Работа системы с временными отметками файлов программно реализована на уровне системных вызовов. Системный вызов – это обращение прикладной программы к ядру операционной системы для выполнения какой-либо операции. Любая прикладная программа или системная утилита реализует низкоуровневый ввод-вывод путем вызова подпрограмм или функций из системной библиотеки.

Справочные данные о воздействии системных вызовов на временные отметки файлов сведены в табл. 4.5.

Таблица 4.5

| **Системный вызов** | **Временные отметки** | | |
|---|---|---|---|
| | **C** | **M** | **A** |
| **`creat()`** – создает новый или очищает существующий файл | + | + | + |
| **`read()`** – читает данные из файла в буфер памяти | | | + |
| **`write()`** – записывает данные из буфера памяти в файл | + | + | |
| **`link()`** – создает еще одно имя для существующего файла | + | | |
| **`unlink()`** – удаляет одно из существующих имен файла | + | | |
| **`utime()`** – устанавливает времена доступа и модификации указанного файла | + | + | + |

Таким образом, преобразование временных отметок файлов происходит по единым алгоритмам, что позволяет использовать ВО при расследовании компьютерных правонарушений в качестве юридических доказательств. И наоборот, временные отметки прошлого предполагается использовать как свидетельство ранее произведенных над файлами операций.

Многие сложные действия над файловыми объектами включают в себя несколько системных вызовов. Так, копирование файла из одного каталога в другой упрощенно представляет собой:

- открытие первого каталога в режиме чтения,
- открытие второго каталога в режиме записи,
- нахождение и чтение индексного дескриптора файла,
- чтение блоков копируемого файла в буфер памяти,
- создание во втором каталоге записи о новом файле,
- создание нового индексного дескриптора файла,
- выделение новому файлу необходимого количества блоков в новом месте,
- запись данных из буфера памяти в блоки нового файла,
- закрытие файлов и каталогов.

Установление и определение ВО файлов может быть связано с временными погрешностями. Системный таймер может накапливать ошибку отсчета текущего времени, и администратор должен систематически проверять и корректировать показание системных часов. Разряженный элемент питания CMOS-памяти приводит к сбросу показаний системных часов при отключении компьютера от источника питания. Файлы, скопированные с сетевых ресурсов, могут иметь иное системное время. В первом приближении будем считать, что системное время определяется с погрешностью, не превышающей одной секунды.

Быстродействие современных компьютерных систем позволяет выполнять основные файловые операции почти мгновенно. Если манипулировать файловыми объектами программно, то за секунду можно успеть изменить состояния и временные отметки у сотен файловых объектов. Но при участии пользователя процесс обновления временных отметок происходит гораздо медленнее. Сравнительно медленно может происходить копирование большого файла, копирование или перемещение группы файлов либо большого каталога.

Существует два подхода к изучению механизма файлового времяобразования. Первый заключается в полном доверии к информации о системных вызовах (см. табл. 4.5) и трассировке утилит и прикладных программ в целях определения реальности и последовательности этих вызовов. Второй состоит в наблюдении за временными отметками файлов до и после выполнения файловой команды. Опираясь на присущий исследователю здоровый элемент недоверия, выберем второй путь.

В целях автоматизации процесса наблюдения за динамикой

временных отсчетов файлов, а также для исключения ошибок, связанных с «ручными» операциями, автором было написано несколько сценариев, в которых использовались возможности системного вызова `stat`, а также системной утилиты с аналогичным именем. Вызов `stat` с именем файла возвращает метаданные этого файла, включая его временные отметки. При этом временные отметки промежуточных каталогов на пути к целевому файлу не искажаются. Утилита `stat` корректно работает с файловыми системами `ext*fs`, смонтированными в режиме `read only`.

Одна из составленных программ через фиксированные интервалы времени производит элементарные операции над файловыми объектами (ФО): создает несколько каталогов, помещает в них небольшие текстовые файлы, производит их чтение, дописывание и удаление, создает прямые и символические ссылки, изменяет права доступа к ФО, осуществляет копирование и перемещение файлов и др. После каждого действия программа выводит на экран или в файл информацию об изменившихся временных отметках ФО. Результаты работы сведены в табл. 4.6 и 4.7.

В табл. 4.6 отображены временные переходы при двухместных операциях, предполагающих различное местоположение исходных и целевых файловых объектов. Звездочки в ячейках временных отметок файла для случая его перемещения указывают на то, что исходный файл логически удаляется, а его временные отметки вместе с `inode` перестают существовать. В строке 10 таблицы отсутствие отметок указывает на то, что переход из каталога в каталог никаких следов не оставляет, если только при этом не нарушаются права доступа.

Таблица 4.6

| № п/п | Файловая операция | Временные отметки источника | | | | | | Временные отметки приемника | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Каталог | | | Файл | | | Каталог | | | Файл | | |
| | | C | M | A | C | M | A | C | M | A | C | M | A |
| 1 | Копирование файла обычное | | | | | | | | | | | | |
| 2 | Копирование с замещением файла | | | | | | | | | | | | |
| 3 | Копирование с переименованием файла | | | | | | | | | | | | |
| 4 | Перемещение файла | | | | * | * | * | | | | | | |
| 5 | Создание прямой ссылки на файл | | | | | | | | | | | | |
| 6 | Создание символической ссылки на файл | | | | | | | | | | | | |
| 7 | Чтение файла через символическую ссылку | | | | | | | | | | | | |
| 8 | Запись файла через символическую ссылку | | | | | | | | | | | | |
| 9 | Запуск файла через символическую ссылку | | | | | | | | | | | | |
| 10 | Переход в другой каталог | | | | | | | | | | | | |

В табл. 4.7 зафиксированы временные отметки, обновившиеся при одноместных операциях. Звездочки в ячейках временных отметок удаленного файла указывают на некорректность их задания.

Таблица 4.7

| № п/п | Файловая операция | Временные отметки | | | | | |
|---|---|---|---|---|---|---|---|
| | | Каталог | | | Файл | | |
| | | C | M | A | C | M | A |
| 1 | Создание каталога | | | | | | |
| 2 | Создание файла | | | | | | |
| 3 | Запись в файл | | | | | | |
| 4 | Чтение файла | | | | | | |
| 5 | Исполнение программного файла | | | | | | |
| 6 | Удаление файла | * | * | * | | | |
| 7 | Изменение владельца файла | | | | | | |
| 8 | Изменение прав доступа к файлу | | | | | | |
| 9 | Переименование файла | | | | | | |
| 10 | Поиск файла в каталоге | | | | | | |
| 11 | Монтирование файловой системы к пустому каталогу | | | | | | |
| 12 | Вход в каталог и выход из него | | | | | | |

Проведенные наблюдения существенно дополняют и корректируют сведения о ВО файлов, приведенных Б. Кэрриэ [6]. Можно говорить о почти однозначном соответствии между комбинацией синхронно измененных ВО и произведенным действием над файлом. Таким образом, по временным отметкам файлов, каталогов и символических ссылок можно с высокой степенью достоверности реставрировать события прошлого. Более подробные сведения о ретроспективном анализе ВО файлов выходят за рамки данного учебного пособия.

Ранее упомянутая утилита **`touch`** с именем существующего файла предназначается для изменения временных меток **`A`** и **`M`** до текущего значения. Используя аргументы, этой же командой можно устанавливать для файла произвольные значения даты и времени модификации и последнего доступа. Такими возможностями может воспользоваться и нарушитель. Для изменения временных отметок файла нужно обладать правом записи в файл и правом исполнения в каталоге, где этот файл находится.

### 4.3. Алгоритмы логического удаления и восстановления файлов

Многие исследователи и опытные пользователи утверждают, что операционные системы клона Linux весьма надежны и успешно противостоят сбоям в работе. Тем не менее опасность потери файлов с ценной информацией является угрозой для любой системы. Чаще эти угрозы исходят не от

операционной системы, а от пользователей. Для обычного пользователя, не имеющего доступа к элементам файловой системы, восстановление уничтоженной информации невозможно. Правда, графическая оболочка современных версий Linux предусматривает «корзину» для удаленных файлов.

Администратор имеет право использовать отладчики и редакторы файловой системы, но восстановление удаленных файлов требует хороших знаний о файловой системе и может потребовать немало времени.

Логическое удаление файла в Linux происходит тогда, когда из соответствующего каталога удаляется последнее имя (жесткая ссылка) файла. Кроме удаления записи в каталоге, система обнуляет биты, закрепленные за этим файлом в битовой карте блоков и индексных дескрипторов, увеличивая цифру свободных **inode** и блоков.

Удаление объектов файловой системы производится с помощью утилиты **rm** (remove). Ввод команды с опцией **–f** означает безусловное удаление файла, при указании **-r** происходит безусловное рекурсивное удаление каталога. При обычном удалении файла система выводит запрос на удаление, который необходимо подтвердить символом «**Y**» (Yes) и **<Enter>**.

Право на удаление файла в ОС Linux вообще не предусматривается. Для того чтобы удалить файл, пользователь должен иметь права на запись и исполнение в том каталоге, где удаляемый файл находится. На сам удаляемый файл пользователь может вообще не иметь никаких прав. В файловых системах Linux существует единственный каталог, в который имеет право записывать информацию любой пользователь, – это каталог **/tmp**. И любой пользователь прежде имел возможность в любой момент удалить из этого каталога любые файлы. Для предотвращения такой возможности уже давно используется специальный Sticky-бит, устанавливаемый на этот каталог. Так, права доступа к каталогу **/tmp** устанавливаются командой

**chmod 1777 /tmp** ,

где первая единица в правах доступа обозначает Sticky-бит.

Права доступа к этому каталогу, отображаемые командой **ls –l**, будут отображаться так: **drwxrwxrwt.** Последний символ **t** как раз и указывает на наличие дополнительных прав (или ограничений) доступа. Но если задать права доступа к этому же каталогу в виде команды **chmod 1776 /tmp**, то отображаемые права доступа будут выглядеть уже так: **drwxrwxrwT**. Прописной символ **T** указывает на противоречие в предоставленных правах: **Sticky**-бит установлен, но прав на исполнение (поиск в каталоге) для всех пользователей не предоставлено. Это очевидная нелепость, оставшаяся в наследство от давно забытых решений: **sticky**-бит имеет смысл только при установленном для всех пользователей праве на *запись* в каталог. Если права записи нет, то и в дополнительном праве (точнее – в запрете) потребности не возникает.

Универсальные операционные системы семейств UNIX и Windows* по умолчанию не предусматривают физического удаления файлов. Для га-

рантированного стирания файлов пользователям рекомендуется использовать специальные утилиты, которые именуются шредерами (от англ. shred – кромсать, резать). Утилита с таким названием имеется в штатной инсталляции современных операционных систем Linux. Она многократно (по умолчанию 25 раз) перезаписывает 128-байтный фрагмент **inode** и каждый из выделенных файлу блоков данных. При удалении данных записываемые комбинации бит меняются случайным образом, чтобы каждый элементарный домен на поверхности диска многократно перемагничивался противоположно направленными силовыми линиями магнитного поля. Синтаксис команды **shred** приведен в кратком справочнике по командам Linux в прил. 1.

Процесс удаления файлов по-разному происходит в файловых системах **ext2fs** и **ext3fs**.

В **ext2fs** удаляется запись об имени файла в каталоге, но индексный дескриптор подвергается лишь косметическим изменениям. Они заключаются в обнулении счетчика жестких ссылок и установке текущего значения времени удаления. Самая ценная информация, заключенная в индексном дескрипторе, заключена в номерах блоков данных, выделенных файлу.

Восстановление удаленных файлов в файловых системах Linux затрудняется рядом обстоятельств. Файл, как известно, состоит из трех частей: совокупности имен или указателей на номер файла (индексный дескриптор), самого индексного дескриптора и определенного количества блоков. При создании файла системной функции **create** передаются символьное имя файла в файловой системе и устанавливаемые права доступа к нему. Система в соответствии с заложенным алгоритмом выделяет для этого имени свободный индексный дескриптор и минимально необходимое количество свободных блоков.

Прослеживается логическая цепочка: каждое из имен файлов содержит ссылку на номер индексного дескриптора, а **inode** в свою очередь ссылается на номера блоков данных. Но обратной связи не существует: в блоках данных отсутствует информация о том, какому файлу они принадлежат (если только файл не имеет собственной сложной структуры с дублирующей информацией), а в **inode** нет упоминания об именах файла. Удаление файла начинается с удаления его последнего имени, поэтому искать удаленный файл по его прежнему имени часто бессмысленно. Пример этого будет приведен ниже.

Выделение блоков под данные производится таким образом, чтобы они, по возможности, располагались по порядку их номеров. Иначе говоря, большому файлу при его создании или копировании система не станет отводить ранее освобожденные одиночные блоки, разбросанные по диску. Но индексные дескрипторы выделяются по одному на файл, поэтому, если новый файл создается в группе каталогов, где находился удаленный файл, система неминуемо использует первый по порядку свободный **inode**. Та-

ким образом, вслед за последним именем файла исчезнет и его индексный дескриптор. В таком случае, если удаленный файл имеет простой формат и речь идет о восстановлении содержавшейся в нем смысловой информации, проще организовать контекстный поиск в еще сохранившихся блоках данных, еще не заполненных новой информацией.

Проверить, как происходит создание и удаление файлов в файловых системах **ext2fs** и **ext3fs**, читателям предлагается, выполнив лабораторное задание № 3.

Механизм журнализации, реализованный в современных файловых системах Linux, а также кэширование дисковой памяти и отложенный характер записи на диск чрезвычайно затрудняют подобные наблюдения. Создание новых файлов, удаление существующих, использование освобожденных **inode** и логических блоков происходит иногда с заметной задержкой, и у исследователя может создаться впечатление, что никаких изменений в файловой системе не происходит.

Теперь рассмотрим, что происходит с данными каталога при удалении файла в файловой системе **ext2fs**. На рис. 4.15 изображен фрагмент логического блока с номером **0x00100203**, содержащего каталог **/home**. В данном каталоге был удален файл с именем **Pr_Linux.doc**.

```
0x00203000  01 04 08 00 0C 00 01 02 : 2E 00 00 00 02 00 00 00  ................
0x00203010  20 00 02 02 2E 2E 00 00 : 65 0D 08 00 14 00 0C 01   .......e.......
0x00203020  50 72 5F 4C 69 6E 75 78 : 2E 64 6F 63 66 0D 08 00  Pr_Linux.docf...
0x00203030  14 00 09 01 72 69 73 5F : 66 73 74 61 62 00 00 00  ....ris_fstab...
0x00203040  67 0D 08 00 14 00 0B 01 : 72 69 73 5F 66 64 69 73  g.......ris_fdis
0x00203050  6B 5F 6C 00 68 0D 08 00 : 14 00 0A 01 72 69 73 5F  k_l.h.......ris_
0x00203060  6C 73 5F 6C 69 31 00 00 : 5A 07 08 00 18 00 0D 01  ls_li1..Z.......
0x00203070  72 69 73 5F 69 6E 6F 64 : 65 5F 62 69 6E 00 00 00  ris_inode_bin...
0x00203080  6A 0D 08 00 18 00 0D 01 : 72 69 73 5F 62 6C 6F 63  j.......ris_bloc
```

Рис. 4.15. Фрагмент дампа логического блока, содержащего каталог **/home** (до удаления файловой записи **Pr_Linux.doc**)

На рисунке 4.16 приведен тот же фрагмент логического блока каталога после удаления файловой записи. Необходимо отметить, что обновления информации об удалении файла на диске пришлось ожидать несколько десятков минут.

```
0x00203000  01 04 08 00 0C 00 01 02 : 2E 00 00 00 02 00 00 00  ................
0x00203010  20 00 02 02 2E 2E 00 00 : 7D 0D 08 00 14 00 02 01   .......}.......
0x00203020  30 30 30 30 30 30 75 78 : 2E 64 6F 63 66 0D 08 00  000000ux.docf...
0x00203030  14 00 09 01 72 69 73 5F : 66 73 74 61 62 00 00 00  ....ris_fstab...
0x00203040  67 0D 08 00 14 00 0B 01 : 72 69 73 5F 66 64 69 73  g.......ris_fdis
0x00203050  6B 5F 6C 00 68 0D 08 00 : 14 00 0A 01 72 69 73 5F  k_l.h.......ris_
0x00203060  6C 73 5F 6C 69 31 00 00 : 5A 07 08 00 18 00 0D 01  ls_li1..Z.......
0x00203070  72 69 73 5F 69 6E 6F 64 : 65 5F 62 69 6E 00 00 00  ris_inode_bin...
0x00203080  6A 0D 08 00 18 00 0D 01 : 72 69 73 5F 62 6C 6F 63  j.......ris_bloc
```

Рис. 4.16. Фрагмент дампа логического блока, содержащего каталог **/home** (после удаления файловой записи **Pr_Linux.doc**)

Рассмотрим содержание файловой записи до ее удаления (см. рис. 4.15). Разбор строки будем производить в обратном порядке, начиная от имени **Pr_Linux.doc**. Байт, стоящий левее (**01h**), указывает, что перед нами обычный файл; стоящий перед ним (**0Ch**) определяет длину имени – 12 байтов (в этом нетрудно убедиться). Два байта, стоящие левее (**00 14h**), также безошибочно определяют длину записи – 20 байтов. Наконец, 4-байтная последовательность **00 08 0D 65h** в десятичном представлении соответствует номеру **inode** = 527717, который был присвоен данному файлу при его создании или копировании.

Теперь посмотрим, что произошло с этой записью после логического удаления файла (см. рис. 4.16). Запись полностью не исчезла, от нее осталось окончание **ux.doc** (третья строка сверху). Первая часть имени файла заменена нулями, индексный дескриптор поменялся на **00 08 0D 7Dh** = = 527741, но проверка показывает, что такой **inode** ни одному файлу еще не выделялся. Длина записи и длина имени файла не соответствуют друг другу. Можно сделать вывод, что мы стали свидетелями не замены записей одного файла другим, а его затирания произвольным кодом. Вероятно, если имя файла имеет какую-то значимость, по имеющемуся остатку имя можно попытаться восстановить. Однако сказать, какому индексному дескриптору это имя соответствовало и, тем более, где располагаются блоки данных удаленного файла, невозможно.

Файл логически удаляется при совпадении двух условий: при удалении его последнего имени (жесткой ссылки) и в том случае, если он не открыт ни одним процессом. В индексном дескрипторе в числе других параметров содержится счетчик жестких ссылок. Если счетчик обнуляется, но файл открыт процессом, узел освобождается после его закрытия процессом.

Каталог **/lost+found** используется программами проверки целостности файловой системы. В этот каталог помещаются индексные дескрипторы, обозначенные в битовых картах **inode** как занятые, но не принадлежащие ни одному из файлов.

Операционная система Linux вне зависимости от используемых и монтируемых файловых систем заполняет неиспользуемые байты блоков нулями. Остаточные данные из удаленных файлов могут сохраняться в блоках, объявленных свободными, но еще не занятых новым файлом.

Авторы одной из методик рекомендуют при наличии логически удаленных файлов все же попытаться восстановить их по еще существующим (не перезаписанным) индексным дескрипторам. Они уверяют, что таким путем удается спасти до 80 % информации.

Вывод списка удаленных файлов производится с помощью внутренней команды **lsdel** отладчика **debugfs**. Можно просматривать этот список в интерактивном режиме либо перенаправить его в файл для фильтрации из него нужных сведений. Предлагаются средства «автоматизации»

процесса восстановления за счет использования нескольких командных строк.

Во-первых, в файл текущего каталога (назовем его **lsdel.out**) выводятся номера всех удаленных **inode** на данном логическом разделе:

```
lsdel | debugfs /dev/hdc3 > lsdel.out
```

Затем, предполагая, что список удаленных **inode**, сформированный командой **lsdel**, находится в файле **lsdel.out**, можно сделать так:

```
cut -c1-6 lsdel.out | grep "[0-9]" | tr -d " " > inodes
```

Новый файл с именем **inodes** содержит номера **inode**, подлежащих восстановлению, по одному в строке. Он используется в следующей команде:

```
sed 's/^.*$/stat <\0>/' inodes|debugfs /dev/hda5 > stats
```

результирующий файл **stats** содержит данные всех команд **stat.**

Специалисты рекомендуют воздерживаться от монтирования устройства (машинного носителя, логического раздела), на котором имеются удаленные файлы. В противном случае система может привести их в окончательно невосстанавливаемое состояние.

Для восстановления данных в файловой системе **ext2fs** нужно найти их метаданные и сделать так, чтобы они снова стали восприниматься операционной системой. Для этого существует два способа. Первый – изменить существующую файловую систему так, чтобы в удаленном **inode** был снят флаг удаления (число ссылок на индексный дескриптор и обнуление времени удаления файла), после чего довериться утилите автоматической проверки и восстановления файловой системы **fsck**. Другой способ, намного более безопасный, но медленный, – выяснить, где именно в разделе лежат данные, после чего записать их в новый файл в другой файловой системе.

Для реализации восстановления по первому варианту следует воспользоваться возможностями утилиты **fsck** (filesystem consistency check – проверка целостности файловой системы). Эта утилита способна в автоматическом режиме распознать следующие повреждения:

- индексные дескрипторы, на которые нет ссылок в каталогах либо содержащие большое число (более сотни) ссылок,
- блоки данных, поименованные в индексных дескрипторах, но обозначенные в битовой карте блоков как свободные,
- блоки данных, обозначенные в битовой карте блоков как занятые, но не числящиеся ни в одном индексном дескрипторе,
- блоки данных, на которые имеются ссылки в нескольких inode,
- каталоги, содержащие ссылки на индексные дескрипторы, значащиеся

свободными.

При обнаружении файла, у которого нет имени (отсутствуют ссылки в каталогах), утилита помещает такой файл в каталог **/lost+found**, присваивая ему новое имя, совпадающее с именем индексного дескриптора. Здесь, кстати, таится еще одна угроза безопасности. Восстановленный файл не обязательно сохраняет свои прежние ограничения в доступе. По умолчанию утилита **fsck** присваивает «потерянным и найденным» файлам права **rwxr-xr-x**. Просматривать список файлов в этом каталоге рекомендуется с помощью команды

```
ls –lad ‘locate /lost+found’
```

Поскольку утилита ничего самостоятельно не исправляет, ее можно использовать для автоматического или ручного восстановления файлов. Если файлы были удалены правильно, утилита ничего неестественного в этом не обнаружит. Поэтому необходимо поискать удаленные фрагменты файлов с помощью других утилит, внести в них изменения, а затем запустить **fsck** с тем, чтобы она обнаружила остальное. Таким способом администратор может избавить себя от значительной части рутинной работы.

Практические рекомендации по восстановлению удаленных и поврежденных файлов на логическом разделе Linux с файловой системой **ext2fs** сводятся к следующему:

1) логический раздел Linux с поврежденными или логически удаленными файлами не следует монтировать к дереву каталогов своей системы до ликвидации всех повреждений. В крайнем случае следует ограничиться монтированием «только для чтения»,

2) поиск удаленных файлов по их именам в **ext2fs** – занятие бесперспективное, поскольку имена файлов затираются в первую очередь. Искать следует удаленные индексные дескрипторы, т. е. **inode** с нулевым числом ссылок и установленной датой удаления. **Inode** сохраняют свою ценность до тех пор, пока содержат ссылки на номера блоков данных,

3) каждый найденный удаленный **inode**, если он содержит адреса блоков данных, следует модифицировать для дальнейшего использования. Модификацию можно провести с помощью отладчика **debugfs**. Для этого следует последовательно выполнить следующие команды (более подробное описание этих команд содержится в приложении):

- входим в командную среду отладчика

```
debugfs,
```

- открываем исследуемый раздел в режиме чтения/записи

```
open –w /dev/hdc3,
```

- выводим на экран список удаленных **inodes**

```
lsdel,
```

- выбираем один из найденных индексных дескрипторов и работаем с ним

**stat <inode>,**

- выводим таблицу **inode**, которая позволяет убедиться в том, что в ней сохранились адресуемые блоки данных и сохраняем содержимое индексного дескриптора в файл

**dump <inode> file.out,**

- пытаемся найти имя файла по его индексному дескриптору

**ncheck <inode>,**

- бит, соответствующий данному **inode** в битовой карте индексных дескрипторов, устанавливаем в «1». Тем самым указываем системе, что индексный дескриптор вновь занят

**seti <inode>,**

- в выводимой командой **mi** информации модифицируем две строки: устанавливаем в единицу число ссылок на индексный дескриптор и обнуляем время удаления файла. Затем процедура повторяется для каждого найденного **inode**

**mi <inode>,**

- закрываем логический раздел

**close,**

- выходим из отладчика

**quit.**

Если удаленные индексные дескрипторы не были найдены, это не означает, что на данном логическом разделе нечего искать. Индексные дескрипторы и логические блоки, как правило, адресуются независимо друг от друга, а **inode** удаленных файлов заполняются новыми метаданными в первую очередь. Поэтому следующим этапом будет являться поиск свободных блоков данных, содержащих утерянную информацию. Однако никаких утилит, позволяющих искать ранее освобожденные блоки данных, не существует. Это можно делать только после визуального просмотра битовой карты блоков, выявляя в ней байты, отличные от **FF**. Затем, последовательно копируя свободные блоки из нужного диапазона номеров (принадлежащих каталогу, из которого предположительно производилось удаление), можно создать файл, который затем можно посмотреть (он наверняка будет состоять из склеенных фрагментов различных файлов). Есть альтернатива – запустить одну из известных утилит контекстного поиска. Эти утилиты очень хорошо работают с англоязычным текстом, но с чтением кириллицы будут проблемы, особенно в кодировке **UNICODE.**

Просмотр отдельных блоков файлов со сложным внутренним форматом может быть связан с проблемами. Дело в том, что штатные приложения, предназначенные для работы с такими файлами, понимают только документы с неповрежденной структурой. Отдельные фрагменты файлов можно восстановить, если они содержат текст в одной из известных кодировок либо интерпретируемый программный код.

Для поиска известных сигнатур и текстовых фрагментов в большинстве случаев можно ограничиться использованием штатных утилит операционной системы. При поиске англоязычных фрагментов, сигнатур вредоносных программ, фрагментов рисунков вполне достаточно богатых возможностей утилит **grep, fgrep** и др.

В файловых системах **ext3fs** процесс удаления файлов происходит иначе. При удалении последнего имени файла это имя автоматически из записи в каталоге не стирается. Запись, принадлежащая удаленному файлу, присоединяется к предыдущей записи, увеличивая ее длину. Наряду с именем удаленного файла сохраняется и 4-байтный индексный дескриптор.

Но воспользоваться сохранившейся записью не удастся. Индексный узел удаленного файла заполняется нулями, в результате чего самое необходимое звено для восстановления файла оказывается разорванным.

Блоки данных удаленного файла продолжают сохранять прежнюю информацию. Однако узнать, где эти блоки располагались, можно только интуитивно. Освобожденные блоки данных будут заняты вновь созданным файлом, если они расположены компактно, а новый файл меньше прежнего.

В **ext3fs** возможен только один вариант восстановления удаленного файла. Он предполагает, что объем и содержимое удаленного файла известны. В этом случае по битовой карте блоков определяются освобожденные блоки в нужном количестве, устанавливаются их порядковые номера, а затем с помощью утилиты **dd** производится их копирование в файл. Если известны какие-либо слова или сигнатуры, входившие в состав удаленного файла, вывод **dd** можно перенаправить в утилиту **grep**.

# 5. СЕТЕВЫЕ ВОЗМОЖНОСТИ ОПЕРАЦИОННЫХ СИСТЕМ LINUX

Операционные системы UNIX развивались одновременно с вычислительными сетями. Включение ЭВМ в компьютерную сеть многократно увеличивает как функциональные возможности пользователя, так и степень уязвимости системы и обрабатываемой информации по отношению к сетевым атакам. Сетевые возможности операционных систем должны быть безопасными, однако это требование гораздо легче провозгласить, чем обеспечить.

Защите сетевой компьютерной информации и безопасной эксплуатации компьютерных сетей под управлением ОС Linux посвящены многие, в том числе довольно удачные книги [1, 3, 13]. Искусство системного администратора во многом определяется его умением правильно построить и грамотно эксплуатировать вычислительную сеть. Для этого операционные системы Linux располагают самыми подходящими возможностями. Под управлением ОС Linux надежно работают и серверные приложения, и межсетевые экраны, и системы обнаружения компьютерных атак.

К сожалению, материал по обеспечению сетевой безопасности настолько обширен, что для его рассмотрения пришлось бы написать отдельную книгу, а возможно, и не одну. Поэтому автор ограничился рассмотрением лишь некоторых механизмов, на которых строится здание сетевой защиты.

При изучении материала предполагается, что читатели знакомы с основами построения компьютерных сетей и с сетевыми протоколами стека TCP/IP.

## 5.1. Контроль и настройка сетевых интерфейсов

Сетевой адаптер – это программно управляемое устройство, благодаря которому персональный компьютер или сервер превращается в интеллектуальный приемопередатчик и приобретает возможность обмена информацией с другими компьютерами в локальной вычислительной сети. Сетевой адаптер можно использовать как устройство перехвата всех или фильтрации определенных пакетов. Компьютер с двумя сетевыми адаптерами может служить транслятором (мост, шлюз, фильтр) между двумя различными ЛВС.

Штатная утилита **`ifconfig`** используется для настройки любых сетевых устройств, подключенных к компьютеру, а также для получения справочной информации о состоянии и работоспособности каждого из них. Для настройки и диагностики беспроводных адаптеров Wi-Fi служит другая утилита, именуемая **`iwconfig`**. Но и в случае работы в беспроводной сети возможности утилиты **`ifconfig`** остаются востребованными.

Для получения текущей информации о состоянии сетевых интерфейсов, в том числе и неактивных, используется команда **`ifconfig –a`**

(рис. 5.1). Для читателя, знакомого с принципами функционирования компьютерных сетей и их терминологией, выведенная информация должна быть понятна. Утилита отображает информацию о состоянии двух физических сетевых интерфейсов: проводного **eth0**, беспроводного **ath0**, а также одного виртуального **lo**.

```
ath0      Link encap:Ethernet  HWaddr 00:1e:58:a1:fd:bd
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0      Link encap:Ethernet  HWaddr 00:02:A5:AB:D9:CC
          inet addr:192.168.0.4  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:5 Base address:0x4000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

Рис. 5.1. Информация, выводимая с помощью команды **ifconfig –a**

Параметр **Link encap** указывает тип интерфейса инкапсуляции линии связи. Такими интерфейсами, в частности, являются физические адаптеры типа Ethernet и логическое построение в стеке протоколов, носящее наименование «обратная петля» (Local Loopback).

**Hwaddr** – это шестибайтный MAC-адрес сетевого адаптера. Большинство адаптеров позволяют его изменять программным путем. Но, поскольку изменение аппаратного адреса сетевого адаптера производится с помощью утилиты **ifconfig** только на сеанс (до выключения питания), представляется, что перепрограммирование ниже уровня драйвера устройства не опускается. Необходимость в изменении аппаратного адреса может возникнуть при скрытном подключении компьютера к исследуемой локальной сети. Некоторые операционные системы регистрируют случаи, когда в сети появляются два узла с одинаковыми аппаратными адресами. Но, с другой стороны, нет ничего неправильного в том, что одному MAC-адресу соответствует несколько IP-адресов.

Для того чтобы перепрограммировать MAC-адрес, необходимо вначале отключить сетевой интерфейс от стека протоколов. Делается это с помощью команды

```
ifconfig eth0 down
```

Затем вводится командная строка, изменяющая аппаратный адрес

**ifconfig eth0 hw ether 01:02:03:04:05:06**

Наконец, адаптер вновь встраивается в стек сетевых протоколов

**ifconfig eth0 up**

Вводя команду **ifconfig eth0**, нетрудно убедиться, что адрес изменен, а интерфейс активен (**up**).

Задать или изменить IP-адрес еще проще. Для этого достаточно ввести команду

**ifconfig eth0 192.168.0.1 netmask 255.255.255.0**

При установке или смене IP-адресов отключать и затем включать интерфейс не требуется. Маску сети можно опустить, и она будет введена по умолчанию (предполагается, что это сеть класса С).

При необходимости одному физическому адаптеру можно поставить в соответствие несколько IP-адресов (сколько именно – выяснить не удалось). Делается это с помощью любой из двух команд:

**ifconfig eth0:1 192.168.0.2**

**ifconfig eth0 add 192.168.0.2**

В некоторых версиях Linux с помощью второй команды удается добавить только один IP-адрес. Система не отказывается выполнить вторую команду неограниченное число раз, но при этом второй из адресов просто перезаписывается. Указывать виртуальные интерфейсы **eth0:1**, **eth0:2**, **eth0:3**, **eth0:4**, **eth0:5** и т. д. с различными IP-адресами оказывается надежнее.

Таким образом, на одном компьютере можно создать небольшую виртуальную сеть. Например, используя два адреса для сетевого обмена, можно имитировать трафик, а с третьего адреса запустить анализатор пакетов для перехвата трафика.

Следует обратить внимание на индикаторы **UP** и **RUNNING** , которые отображают состояние сетевого интерфейса. Индикатор **UP** означает, что адаптер работает в стеке сетевых протоколов в составе компьютера. Индикатор **RUNNING** указывает на подключение к сети и режим сетевого обмена. Если извлечь из адаптера сетевой кабель, то надпись **RUNNING** не будет отображаться.

Для того чтобы узел был доступен для сетевого обмена, для него должен быть задан как аппаратный, так и IP-адрес. Информация о соответствии между этими двумя адресами формируется с помощью двух протоколов, ARP и RARP, и хранится в области оперативной памяти, которая именуется ARP-кэшем. ARP-кэш представляет собой таблицу размером до 254 записей [3]. Информация в ARP-кэше устаревает и стирается через 30-120 сек после очередного обновления.

Для того чтобы сделать узел недоступным и относительно невидимым в ЛВС, можно отключить ARP-отклик. Делается это с помощью команды

**ifconfig eth0 -arp**

После этого всякое участие компьютера в сетевом обмене становится невозможным. Предполагается, что компьютер станет невидимым из локальной сети, поскольку не будет отвечать на протокольные запросы о соответствии адресов. Но компьютер может быть физически подключен к сети и при этом не иметь установленного IP-адреса.

Информация, выведенная утилитой **ifconfig** после ввода параметров **promisc** и **–arp**, изображена на рис. 5.2.

```
eth0  Link encap:Ethernet  HWaddr 00:02:A5:AB:D9:CC
inet addr:192.168.0.4  Bcast:192.168.0.255  Mask:255.255.255.0
UP BROADCAST RUNNING NOARP PROMISC MULTICAST  MTU:1500  Met-
ric:1
RX packets:12 errors:0 dropped:0 overruns:0 frame:0
TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:910 (910.0 b)  TX bytes:774 (774.0 b)
Interrupt:5 Base address:0x4000
```

Рис. 5.2. Информация, выведенная командой
**ifconfig eth0 promisc -arp**

Эксплуатация беспроводных сетей Wi-Fi вызвала необходимость в еще одной утилите, получившей название **iwconfig**. На рис. 5.3 отображена информация о состоянии беспроводного адаптера, выведенная с помощью команды **iwconfig ath0**. С помощью этой же утилиты производится установка необходимых параметров.

```
ath0      IEEE 802.11g  ESSID:"abcd"  Nickname:""
          Mode:Ad-Hoc  Frequency:2.412 GHz  Cell: 02:1E:58:A1:FD:B9
          Bit Rate:0 kb/s   Tx-Power:15 dBm   Sensitivity=1/1
          Retry:off   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=0/70  Signal level=-93 dBm  Noise level=-93 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

Рис. 5.3. Информация, выведенная командой **iwconfig ath0**

В первой строке отображается название поддерживаемого сетевого протокола **IEEE 802.11g**, идентификатор **ESSID** и условное имя **Nickname** беспроводной сети. Основными параметрами беспроводного приемопередатчика являются:

- **mode** – режим работы. Беспроводный адаптер может работать (или имитировать работу) в режимах **ad-hoc** (работа в одноранговой сети *точка-точка* без выделенной точки доступа), **managed** (обычный режим при работе с точкой доступа), **master** (адаптер, работающий в режиме точки доступа) либо **monitor** (режим перехвата всех пакетов на данном канале – поддерживается не всеми адаптерами),
- **channel** – номер канала (от 1 до 11) или **freq** – точное значение частоты приемопередатчика,
- **ap** – аппаратный адрес точки доступа (в таблице отображается ключевым словом **Cell**),
- **rate** – скорость передачи данных [бит/сек],
- **Tx-Power** – относительная мощность передатчика в [дБ] относительно 1 мкВт,
- **key** – ключ шифрования.

Отображается также информация о чувствительности приемника, относительном уровне сигнала и шума.

Еще одна утилита **iwlist** показывает некоторые параметры беспроводной карты, которые недоступны **iwconfig**. Так, команда

```
iwlist <dev> scan
```

позволяет получить список доступных точек доступа и компьютеров, настроенных в режим подключения компьютер – компьютер, а также такие параметры, как имя сети, качество сигнала, частота, режим работы и другое.

Некоторые параметры с помощью утилиты **iwconfig** можно не только контролировать, но и устанавливать. Утилита позволяет в одной команде устанавливать несколько параметров:

```
iwconfig ath0 mode adhoc channel 1 essid "abcd"
```

Указывая значение параметра **auto**, мы доверяем интеллекту компьютерной программы (а точнее – ее автора).

Некоторые беспроводные адаптеры позволяют производить конфигурацию нескольких виртуальных интерфейсов на базе одного физического устройства. Так, с помощью одной команды можно эмулировать несколько беспроводных устройств различного типа на базе одного сетевого адаптера **wifi0**.

**wlanconfig ath create wlandev wifi0 wlanmode <virt_dev>**

Виртуальные адаптеры могут работать в режиме точки доступа (**AP - access point**), сетевого адаптера в одноранговой сети (**adhoc**), монитора (**monitor**) и др. Однако одновременно заставлять работать один физический радиопередатчик в несовместимых режимах нельзя. Так, невозможно одновременно эмулировать работу точки доступа и обычной точки ad-hoc.

## 5.2. Разведка сети

Прежде чем отправить сообщение, установить сеанс связи, требуется узнать сетевой адрес абонента, убедиться в наличии и активности сетевого узла и нужной сетевой службы. Для получения этой информации написано и используется множество утилит. В некоторых случаях получение информации о доступности сетей, узлов и протоколов транспортного уровня является прелюдией к сетевой атаке.

Наиболее простая и известная команда **ping** использует специальный протокол **icmp** и служит для зондирования эхо-запросами сетевых узлов для установления их наличия и доступности.

**ping <параметры> <адрес_хоста> <номер_порта>**

**<параметры>** (в зависимости от типа ОС могут использоваться иные символы)**:**

**-l count** или **-c count** – отправка указанного числа пакетов. По умолчанию (в зависимости от версии ОС) посылается либо один пакет, либо бесконечная серия пакетов с интервалом в одну секунду. Непрерывная отправка прерывается нажатием **Ctrl – C**,

**-s count_byte** – общее количество байтов в **icmp**-пакете с эхо-запросом (длина заголовка **icmp**-пакета – 8 байтов),

**-i timeout** – временной интервал в следовании пакетов в секундах,

**-f** – направление пакетов с максимально возможной скоростью (только с правами **root**),

**<адрес_хоста>** – доменное имя или IP – адрес целевого компьютера,

**<номер_порта>** – номер, закрепленный за сетевой службой, запущенной на удаленном компьютере (смотри файл **/etc/services**).

Например, команда

**ping –c 3 –i 5 192.168.1.2 21**

направляет 3 стандартных **icmp**-пакета с пятисекундным интервалом в адрес FTP-сервера (порт 21) на узле с IP-адресом 192.168.1.2.

Утилита выводит данные построчно в следующем порядке: число

байтов в принятом пакете, IP-адрес исследуемого узла, порядковый номер пакета, счетчик «жизни» пакета и время возврата.

Более сложным инструментом для сетевой разведки является утилита **nmap** (netmap – карта сети). Она использует девять различных видов сканирования сетевых узлов. Принципы сканирования основаны на передаче в адрес интересующего узла сетевых пакетов с определенным «наполнением» и анализом отклика. При этом используются особенности в реализации стека протоколов TCP/IP, присущие известным операционным системам и сетевым службам. Направляемые пакеты могут имитировать процесс установления или завершения сеанса, направление дейтаграммы, различные ошибочные ситуации и др.

Команда для сетевого сканирования выглядит так:

**nmap <тип_сканирования> <параметры> <список узлов или сетей>**

**<тип_сканирования>**

**-sT** – обычное TCP-сканирование с установлением соединения. Используется по умолчанию и может запускаться обычным пользователем,

**-sP** – обычное ping-сканирование,

**-sS** – TCP-сканирование с помощью сообщений SYN. Утилита инициирует установление TCP–сеанса, отправляя в адрес узел:порт первый пакет с установленным битом SYN. Адресат отвечает пакетом с установленными битами SYN и ACK, чем обозначает себя. Но вместо согласия на установление соединения утилита посылает пакет с установленным битом RST, чем разрывает соединение. Считается наилучшим из методов TCP-сканирования,

**-sU** – UDP-сканирование, при котором в адрес каждого порта направляется пустой UDP–пакет. Если порт закрыт, адресат отправляет клиенту пакет с установленным битом RST. Если порт открыт, он принимает пакет без ответа,

**-sF** – FIN-сканирование. Направляется пакет, сигнализирующий о разрыве соединения TCP (которое еще не было установлено). Если указанный порт закрыт, система отвечает пакетом с установленным битом RST, если открыт – пакет не направляется (кроме ОС Windows*),

**-sN** – нуль-сканирование. Направляется пакет, в котором не установлено ни одного битового флага. Результат аналогичен FIN-сканированию.

**<параметры>**

**-O** – режим изучения откликов для определения типа удаленной операционной системы. Большинство ОС обладают своей спецификой при управлении сетевыми протоколами. Для установления типа ОС программа посылает определенные пакеты в адреса конкретных портов и фиксирует реакцию на них,

**-p <диапазон>** – диапазон портов, которые будут сканироваться (указываются через запятую или дефис),

**-v** (**-vv**) – режим вывода подробной информации,
**-T <число>** – темп сканирования от «0» – очень медленно (один пакет в пять секунд) до «5» – максимально быстро (один пакет за 0.3 секунды).

**<список узлов или сетей>**

Доменные имена в списке указываются через запятую. Диапазон IP–адресов указывается в виде номера сети и сетевой маски, например 192.168.1.00/24. Любое число можно заменить символом звездочки *. Диапазон адресов в любом из октетов можно указывать в виде начального и конечного значения, через дефис, например 1–24. Наконец, нужные числовые значения можно указать через запятую, без пробела, например 192.168.2.3,7,17,24.

Исчерпывающую информацию о программе **nmap** можно получить в [3].

## 5.3. Перехват и анализ сетевого трафика

Утилита **tcpdump** является мощнейшим средством перехвата и анализа сетевого трафика. Эта универсальная утилита для прослушивания моноканала присутствует почти во всех дистрибутивах Linux, а для ее запуска необходимы права суперпользователя. Команда автоматически переводит сетевой адаптер в режим захвата всех пакетов в моноканале, но отображает только отфильтрованные пакеты.

**tcpdump <параметры> <параметры_фильтрации>**

Утилиту можно запустить без аргументов с помощью одноименной команды. В этом случае один (или единственный) сетевой интерфейс переводится в режим беспорядочного захвата пакетов, а текстовая информация о заголовках перехваченных пакетов выводится на экран. Это не очень удобно, так как приход каждого нового пакета сопровождается, как минимум, одной новой строкой, а при их обилии продуктивное чтение и анализ трафика становятся невозможными.

Утилита очень богата возможностями, и в ее командной строке предусмотрено несколько десятков параметров. Рассмотрим наиболее важные из них.

1. С помощью параметра **-w <имя_файла>** производится запись перехваченной информации в файл специального формата. Прочитать такой файл с выводом информации на экран можно только с помощью **tcpdump**, задав для этого аргумент **-r <имя_файла>**. В то же время отфильтрованные утилитой данные можно сохранить в обычном текстовом файле, используя перенаправление вывода «**>**».
2. По умолчанию в каждом пакете захватывается для анализа 68 байтов. Почему выбрано именно это число? Заголовок канального уровня (Ethernet–кадр) состоит из 14 байтов. Минимальные размеры заголовков IP

и TCP пакетов составляют по 20 байтов. Еще 14 байтов отводится для распознавания инкапсулированного пакета прикладного уровня. Для явного задания длины «отрезаемой» для анализа части пакета в байтах служит аргумент **-s <длина_пакета>**. В случае необходимости перехвата всего пакета (это может посягать на конфиденциальность передаваемых данных!) его длина задается равной 1514 байтов (14 байтов заголовка кадра Ethernet + 1500 байтов как максимальный размер вложимого кадра).

3. Число перехваченных пакетов можно ограничить путем задания аргумента **-c <число_пакетов>** с завершением работы после выполнения задания.

4. При наличии в составе компьютера нескольких сетевых интерфейсов аргумент **-i <интерфейс>** позволяет определить тип сетевого адаптера (например, **-i eth1**) или модема (**-i ppp1**), с помощью которого производится перехват пакетов. Если физические интерфейсы будут заняты в сетевом обмене, для перехвата данных можно задействовать логический интерфейс обратной петли **lo**.

5. По умолчанию заголовок канального уровня не перехватывается, и внешним является IP–пакет. Для перехвата заголовка кадра Ethernet с MAC–адресами передатчика и приемника необходимо указать параметр **-e**.

6. Для вывода более подробной текстовой информации можно воспользоваться аргументами **-v, -vv, -vvv**. Стандартный формат текстовой строки анализатора может включать следующие поля:

- отметку времени перехвата, в которой три пары цифр, разделенных двоеточиями, указывают часы, минуты и секунды, а последние шесть цифр – дробную часть секунды,
- доменное имя или IP–адрес хоста–отправителя,
- номер порта получателя,
- обозначение установленных битовых флагов TCP–заголовка, которые несут информацию об этапе в установлении сеанса,
- начальный и конечный (через двоеточие) порядковые номера TCP–сегмента, а также (в скобках) – число переданных байт,
- размер TCP–окна в байтах.

7. Для отображения заголовков и содержимого пакетов в шестнадцатеричном коде служат аргументы **-x (-xx)**. Если возникает потребность в отображении содержимого пакетов в шестнадцатеричных и ASCII– кодах, можно воспользоваться аргументом **-X**.

**<параметры фильтрации>** используют несколько ключевых слов:

- протокол **(proto)** – указывает, какие именно пакеты подлежат перехвату. Среди часто используемых можно указывать ключевые слова: **ether, ip, arp, rarp, tcp, udp, icmp, ip6**,
- направление – указывает источники и получателей сообщений: **src**

(source – источник), **dst** (destination – получатель) или их комбинации: **src or dst** или **src and dst**,

- объекты прослушивания, к которым могут относиться:

**host** (номер или имя) – сетевой узел, являющийся источником или получателем сообщений,

**net** (сетевая часть адреса) – локальная сеть или ее часть,

**port** – номер или символическое обозначение службы, указанной в таблице **/etc/services**.

В параметры фильтрации могут входить математические выражения. Например, **'ip[6:2] & 0x1FFF == 0'** условия фильтрации выполняются, если результат побитового логического умножения 6-го и 7-го байтов заголовка пакета с маской **0x1FFF** равен нулю. Ключевые слова и математические выражения могут объединяться с использованием логических условий: **not, and** и **or**.