Министерство образования и науки Российской Федерации

Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина

**В.В. Бакланов**

# ЗАЩИТНЫЕ МЕХАНИЗМЫ ОПЕРАЦИОННОЙ СИСТЕМЫ LINUX

*Допущено УМО по образованию в области информационной безопасности в качестве учебного пособия для студентов, обучающихся по специальностям «Компьютерная безопасность» и «Комплексное обеспечение информационной безопасности автоматизированных систем»*

Научный редактор чл.-кор. АК РФ, проф., д-р техн. наук Н.А. Гайдамакин

Екатеринбург
УрФУ
2011

УДК 691.3.07
ББК 32.973.26
Б 19

Рецензенты:

кафедра информационной безопасности Тюменского государственного университета (зав. кафедрой проф., д-р физ.-мат. наук А.А. Захаров);

доц., канд. физ.-мат. наук О.Н. Соболев (Институт математики и компьютерных наук УрФУ)

**Бакланов В.В.**

Защитные механизмы операционной системы Linux: учебное пособие / В.В. Бакланов. под ред. Н.А. Гайдамакина. Екатеринбург: УрФУ, 2011. 354 с.

ISBN 978-5-321-01966-5

В учебном пособии рассматриваются вопросы связанные с угрозами безопасности и защитой информации в операционных системах на базе ядра Linux. Пособие содержит теоретическую часть из пяти глав, лабораторный практикум и несколько приложений.

Учебное пособие предназначено для студентов, обучающихся по специальностям «Компьютерная безопасность», «Информационная безопасность автоматизированных систем», «Информационная безопасность телекоммуникационных систем». Пособие также может быть полезно широкому кругу читателей – от опытного пользователя и системного администратора, до преподавателя вуза и компьютерного эксперта-криминалиста.

Библиогр.:

УДК 691.3.07
ББК 32.973.26

ISBN 978-5-321-01966-5

## 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**.

## ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Предлагаемый 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.

## ОБЩИЕ ТРЕБОВАНИЯ

1. Практикум рассчитан, как минимум, на 28 часов лабораторных занятий.
2. Перед проведением лабораторных занятий обучаемые должны прослушать теоретический курс в объеме настоящего пособия либо освоить его самостоятельно. Непосредственно перед проведением каждой работы преподаватель по своему усмотрению организует в устной или письменной форме допуск обучаемых к самостоятельному выполнению работ.
3. Для проведения занятий необходим компьютерный класс из расчета одно автоматизированное рабочее место на базе персонального компьютера на каждого пользователя. Сложные задания по усмотрению преподавателя разрешается проводить подгруппами в составе двух человек.

На каждом компьютере должна быть инсталлирована одна из распространенных версий операционных систем GNU Linux. Желательно использовать дистрибутивы последних версий с последними версиями ядра операционной системы, что обеспечит поддержку наиболее передового аппаратного обеспечения компьютера. Если инсталляция программного обеспечения на жесткий диск невозможна, рекомендуется использовать загружаемые рабочие системы, установленные на компакт-диске (так называемый Live-CD, содержащий полноценную версию ОС).

Перед выполнением конкретного задания на каждом рабочем месте должны быть созданы учетные записи пользователей, предусмотренные заданием на лабораторную работу. При выполнении некоторых заданий обучаемые наделяются правами суперпользователя и создают учетные записи самостоятельно.

В ходе занятий используются встроенные команды интерпретатора **`/bin/bash`** (Bourne Again Shell) и штатные утилиты операционной системы. Для исследования структуры файловых систем автором рекомендуются свободно распространяемая утилита **`extview`** (автор – Желтышева Е.Д.), и сценарий **`extv`** на языке Perl (автор – Пирожкова М.В.). Кроме того, в работах 2 и 4 используются две утилиты, созданные М.Э. Пономаревым.

Формой контроля является письменный отчет и защита работы.

## ПАМЯТКА ОБУЧАЕМЫМ

- Все занятия проводятся в самостоятельном режиме в соответствии с пунктами задания. Задания изложены в логической последовательности, и изменять порядок их выполнения не рекомендуется. В случае затруднений обращайтесь к преподавателю.
- Все операции с файлами и каталогами производите только в режиме текстовой консоли или эмуляции текстового терминала. Там, где это предусмотрено заданием, используйте возможности файлового менеджера **Midnight Commander**.
- Обучаемые при проведении лабораторных работ должны работать в нескольких текстовых консолях. Переключая консоль, вы можете поочередно работать с объектами операционной системы от имени нескольких пользователей и администратора системы. Переключение текстовой консоли производится комбинацией клавиш **Alt-Fn**, где **n** – номер консоли.
- Наибольшие затруднения и наиболее частые ошибки у начинающих, а также пользователей, не искушенных в интерфейсе командной строки, вызывает ввод команд. Если образец команды предусмотрен заданием, постарайтесь разобраться с ее синтаксисом и назначением элементов командной строки. Многие команды даны с указанием объектов в угловых скобках, например **<dev>**. Не копируйте команды механически и правильно подставляйте в них нужные параметры. Перед тем как завершить введеную команду клавишей **<Enter>**, проверьте правильность ее написания. В вашем распоряжении краткий справочник по командам, приведенный в приложении, примеры команд, сопровождающие текст пособия, а также электронные руководства **man** и **info**.
- При выполнении ряда заданий вам придется работать с системой, имея полномочия суперпользователя. Будьте особо внимательны при вводе команд. Помните, что операционная система, как правило, не контролирует целесообразность команд суперпользователя и ваше ошибочное или необдуманное действие может привести к краху системы. Ориентируйтесь на характерную форму приглашения к вводу команд (**$** – пользователь, **#** – администратор). Не бравируйте своим умением быстрого ввода команд. Наиболее ответственные команды, которые могут привести к разрушению операционной системы или тотальному стиранию данных с машинных носителей, перед запуском не стесняйтесь показать преподавателю.
- Автор намеренно усложнил отдельные задания. В некоторых случаях обучаемым предлагается выполнить действия, заведомо приводящие к ошибкам. Например, предлагается создать, прочитать или удалить файл из каталога, который для него недоступен. Обучаемый должен обнаружить причину отказа в доступе, отразить ее в отчете, а затем выполнить

задание, изменив права на доступ. Подобные «ошибки» способствуют более прочному закреплению учебного материала.

- Ответы на поставленные вопросы (с указанием пунктов задания) заносите в отчет, который можно вести в произвольной форме в рукописном или электронном виде.
- Защита выполненных работ производится индивидуально при предъявлении отчета. Основная форма защиты – ответы на вопросы тестового задания (в виде программированного контроля). Преподаватель может по своему усмотрению изменять форму защиты.

## ЛАБОРАТОРНАЯ РАБОТА № 1
## «Исследование файловых объектов с правами пользователя»

### Выполнение работы

1. Изучите теоретический материал, изложенный в параграфах 1.3, 3.1 и 3.5 учебного пособия.
2. Зарегистрируйтесь в первой консоли как **root** с паролем, который вам будет сообщен преподавателем.
3. Для исследования файловых объектов вам потребуются еще два пользователя, учетные записи которых предстоит создать. Для этого воспользуйтесь краткой формой команды

```
useradd -m user1
```

Аргумент **-m** предписывает создание домашнего каталога пользователя с его именем. После этого пользователю необходимо присвоить пароль. Это делается командой

```
passwd user1
```

На запрос введите простой пароль 1234567890.

4. Повторите команды п. 3 для создания учетной записи пользователя **user2**. Помните, что оба пользователя по умолчанию являются членами одной группы **users**.
5. Аналогично откройте третий текстовый терминал и зарегистрируйтесь как **user1**.

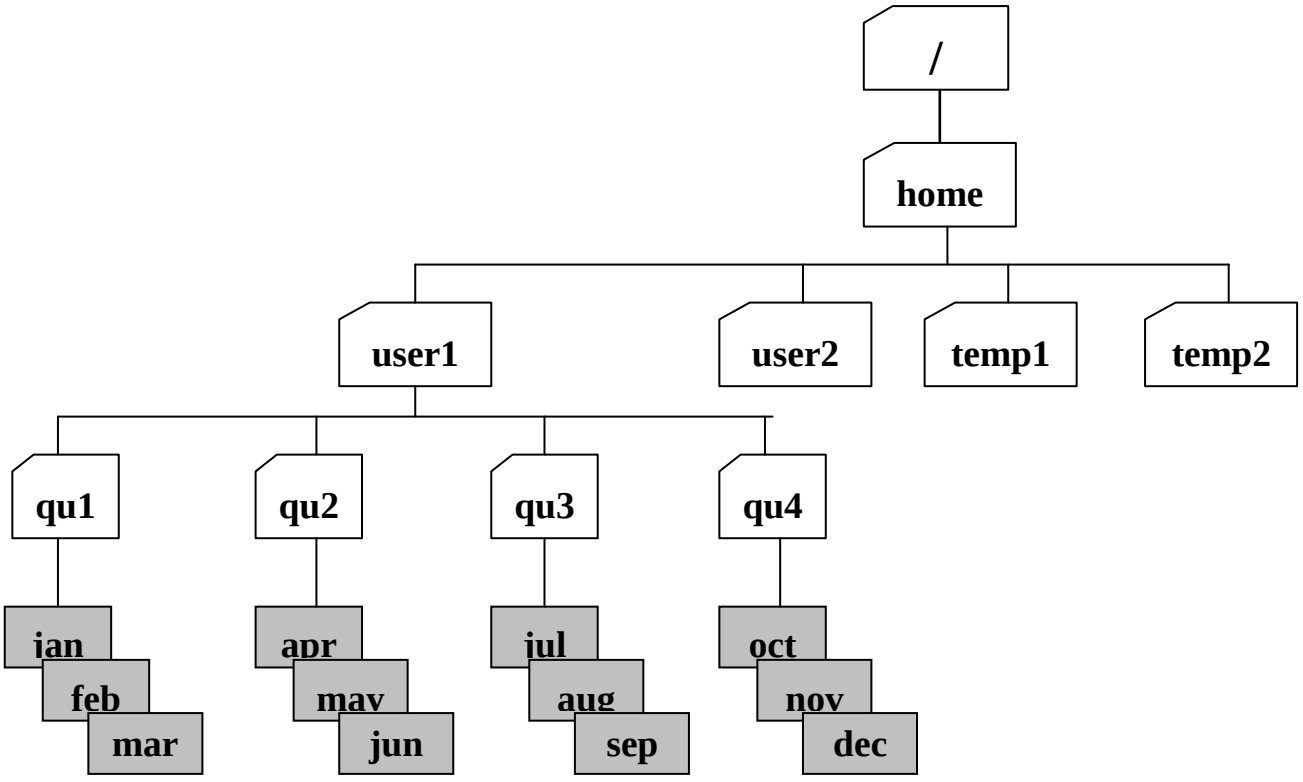


Рис. 1. Пояснение к пп. 10–13

6. С помощью **Alt-F2** откройте второй текстовый терминал и зарегистрируйтесь как **user2** с паролем 1234567890.
7. Нажатием **Alt-F1** вернитесь в первую консоль. Теперь, переключая консоль, вы можете работать с объектами операционной системы от имени двух разных пользователей и администратора системы. Основная часть задания выполняется с правами обычных пользователей. Переходите в первую консоль и используйте права **root** только при выполнении соответствующих пунктов задания. Будьте внимательны при вводе команд! Помните, что операционная система не контролирует действий администратора и ваше ошибочное действие может привести к краху системы. Ориентируйтесь на характерную форму приглашения к вводу команд, в котором отражены ранг и имя пользователя.
8. С правами **user1** с помощью команды **cd /root** попробуйте войти в каталог суперпользователя. Объясните результат. Затем с помощью команды **ls -la /** просмотрите список основных каталогов и укажите, в какие из них вы имеете право войти.
9. Проверьте права доступа к домашним каталогам пользователей **/home/user1** и **/home/user2**: они должны быть установлены в 755.
10. Переключитесь в консоль администратора и создайте два новых временных каталога:

```
mkdir -m 777 /home/temp1
```

и

```
mkdir -m 1777 /home/temp2
```

11. Вернитесь в консоль **user1** и, пользуясь командой **mkdir,** создайте в домашнем каталоге пользователя **/home/user1** четыре каталога с именами **qu1, qu2, qu3, qu4**. При создании каталогов объявите следующие права доступа к ним: (**qu1** - 777, **qu2** - 404, **qu3** - 1333, **qu4** – 505). Пример:

```
cd; mkdir -m 777 qu1
```

С помощью команды **ls /home/user1** убедитесь в том, что каталоги созданы. Какие из предоставленных прав кажутся вам лишенными смысла? Почему?

12. Задайте права доступа к файлам «по умолчанию». Для этого установите маску доступа **umask 022.** Поясните, какие права к вновь создаваемым файлам и каталогам будут предоставляться пользователю, членам его группы и остальным.
13. В каждом из каталогов создайте по три текстовых файла с именами (**jan, feb, mar**), (**apr, may, jun**), (**jul, aug, sep**), (**oct, nov, dec**). В каждый файл запишите календарь на определенный месяц текущего года. Например, команда **cal 1 2011 >jan** создает в текущем каталоге файл **jan** и записывает в него календарь на январь

2011 года. Не забывайте, что использование относительного (короткого) имени файла требует, чтобы вы находились в нужном каталоге. В противном случае следует указывать полный путь к создаваемому файлу. Для навигации по каталогам используйте команды **cd** и **pwd.** В каком случае создание файлов не удалось? Почему?

14. С помощью команды **chmod** установите нужные права доступа в «недоступные» каталоги **qu2**, **qu4** и создайте там указанные файлы. После этого верните каталогам прежние права доступа.
15. С помощью команд **cd** и **ls** войдите в каждый из созданных каталогов и просмотрите список созданных файлов. При просмотре содержимого каталогов используйте два режима: **ls** без аргументов и **ls -l**. В каких случаях не удалось войти в каталог? В каких случаях не удалось посмотреть список файлов? Почему?
16. Прочитайте содержимое одного из файлов в «темном» каталоге (например, **cd /home/user1/qu3; cat aug**). Сделайте выводы.
17. Перейдите во 2-ю консоль и с правами пользователя **user2** войдите в каталог **/home/user1/qu1.** Создайте в каталоге **/home/user2** новый файл **quart1** путем конкатенации нескольких имеющихся:

```
cat jan feb mar >/home/user2/quart1
```

С помощью команды **file** определите тип созданного файла. Попробуйте вывести его на экран командой **cat**. Что представляет собой данный файл?

18. С помощью команды **chmod** установите права доступа 077 на созданный файл **quart1**. Вновь попробуйте прочесть его. Ответьте, почему владельцу файла запрещается доступ, если файл доступен для всех? Что необходимо сделать, чтобы вернуть владельцу права на доступ?
19. Установите для файла **quatr1** права на доступ 4700. Кому и какие права вы при этом предоставили? Как воспользоваться этими правами? Какие из предоставленных прав не имеют смысла?
20. Перейдите в консоль администратора и передайте право владения на файлы **may** и **aug** пользователю **user2** (команда **chown**). Поочередно из консолей **user1** и **user2** проверьте, как изменились права владения файлами после его передачи. Может ли пользователь **user2** воспользоваться предоставленными правами? (Пользователи **user1** и **user2** при создании учетных записей по умолчанию отнесены к одной группе **users.**)
21. Правами пользователя **user1** из каталогов **/home/temp1** и **/home/temp2** с помощью команды **ln** создайте две «жесткие» ссылки на файл **dec** с именами **dec_h1** и **dec_h2**

```
ln /home/user1/qu4/dec /home/temp1/dec_h1
```

Чем созданные ссылки отличаются от исходного файла?

22. С помощью команды **ln -s** создайте из каталогов **/home/temp1** и **/home/temp2** две символические ссылки на файл **dec** с именами **dec_s1** и **dec_s2**. Чем отличаются созданные ссылки от исходного файла? Попытайтесь прочитать содержимое файлов символических ссылок. Что они собой представляют?
23. Правами пользователя **user2** с помощью команды **cp** создайте в каталогах **/home/temp1** и **/home/temp2** по одной копии файла **dec** с другим именем (**dec_copy1**). Чем отличаются исходный файл и его копия (обратите внимание на то, кто является владельцем исходного файла и его копии)? Чем отличаются права доступа на эти файлы? Вернитесь в консоль **user1.**
24. С правами пользователя **user1** создайте жесткие ссылки из его домашнего каталога на файл **/bin/su** (доступен обычным пользователям только на исполнение) и на файл **/etc/shadow** (для обычных пользователей недоступен). Ответьте, какими правами на объектовый файл нужно обладать, чтобы создать на него жесткую ссылку? Какую выгоду получает обладатель жестких ссылок на недоступные файлы?
25. С правами пользователя **user1** скопируйте в его домашний каталог утилиту **/bin/mount**. Сравните между собой оригинал и копию и укажите все отличия. Сможет ли пользователь применить копию опасной утилиты во вред политике безопасности? Почему?
26. С помощью команды **rm** удалите файл **dec**. Что произошло с «жесткими» и символическими ссылками на данный файл? Что произошло с его копиями? Что нужно сделать для того, чтобы файл перестал существовать (на логическом уровне)?
27. Правами **user1** удалите файлы из каталогов **/home/temp1** и **/home/temp2.** Какие файлы не удалось удалить? Почему? Попробуйте удалить оставшиеся файлы правами пользователя **user2**. Объясните результат.
28. Попробуйте удалить любой из каталогов **qu1, qu2, qu3, qu4** с помощью команды **rmdir** (не удаляя предварительно из них файлов). Объясните результат.
29. Войдите в консоль администратора и с правами **root**, пользуясь командой **chattr,** заблокируйте файл **feb** от любых изменений (предварительно ознакомьтесь с синтаксисом команды). Установите параметр запрета любых операций, кроме добавления данных для файла **mar**. Вернитесь в консоль **user1**. С помощью команды **lsattr -l** проверьте наличие дополнительных атрибутов у файлов.
30. Правами пользователя **user1** добавьте одну строку **finish** в конец файлов **feb** и **mar** . Воспользуйтесь для этого командой

```
echo finish >> file_name
```

Убедитесь в успешном завершении операции, объясните результат.

31. Правами пользователя **user1** с помощью команды **rm -rf** последовательно удалите ранее созданные каталоги **qu2, qu3, qu4** вместе с файлами. Объясните результат.
32. С правами пользователя **user1** создайте в **/home/user1** два каталога **mkdir –m 400 src** и **mkdir –m 200 dst**. Внутри каталога **src** командой **echo 1234567890 > abc** создайте текстовый файл (если для этого прав доступа к каталогу недостаточно, временно измените их). Для созданного файла установите права доступа **chmod 100 abc**.
33. Манипулируя правами доступа на созданные файловые объекты, установите минимально необходимые права, позволяющие копировать и перемещать указанный файл из каталога **src** в каталог **dst**. Зафиксируйте результаты в отчете.
34. Определите, какие минимально необходимые права на файловые объекты нужно иметь, чтобы выполнять копирование и перемещение файла от имени администратора.
35. Понаблюдайте и зафиксируйте в отчете изменения временных отметок файлов и каталогов, происходящие при файловых операциях.
36. Ответьте на указанные преподавателем тестовые вопросы.
37. После успешной защиты лабораторной работы, получив разрешение преподавателя, из консоли суперпользователя удалите все созданные вами файлы. *Соблюдайте осторожность: с правами* **root** *и с помощью утилиты* **rm** *вы можете вызвать крах системы!*
38. После выполнения работы командами **exit** завершите пользовательские сеансы во второй и третьей консолях, а из первой консоли правами **root** выполните команду останова системы **halt**.

**Контрольные вопросы**

1. Какие дополнительные атрибуты можно присвоить файлу в файловой системе **ext*fs**? Как эти атрибуты влияют на обеспечение конфиденциальности, целостности и доступности информации?
2. Как можно создать текстовый файл без помощи текстового редактора?
3. В чем различие между копиями файла, его «жесткими» и символическими ссылками? Для чего используются символические ссылки?
4. Для каких целей могут использоваться «темные» каталоги?
5. Какие права по отношению к файлам и каталогам вам необходимо иметь для копирования файла? Как изменяются при этом атрибуты копии?
6. Администратор по невнимательности ввел следующую команду:

**chmod –R 555 /** . Что последует за выполнением этой команды?

## ЛАБОРАТОРНАЯ РАБОТА № 2
## «Исследование архитектуры файловых систем **ext*fs**»

1. Изучите теоретический материал, изложенный в главе 4, а также в параграфах 3.2 – 3.4 учебного пособия.
2. В Приложении к учебному пособию найдите и прочитайте справочный материал по использованию утилиты **extview**.
3. Зарегистрируйтесь в первой текстовой консоли с правами **root.** Пароль администратора узнайте у преподавателя.
4. С помощью **Alt+F2** откройте второй текстовый терминал и зарегистрируйтесь как пользователь **user1**.
5. Вспомните синтаксис команд **mount** и **umount**.
6. С помощью команды **which <file_name>** убедитесь в наличии на дисковом разделе Linux программных файлов **extview** (или **extv**) и **fillfile**. Если указанные файлы отсутствуют, смонтируйте предоставленный преподавателем сменный носитель к одному из подкаталогов **/mnt** и скопируйте с него поименованные файлы в каталог **/sbin**. Размонтируйте носитель (до размонтирования не извлекать!) и верните его преподавателю.
7. С помощью команды **ls -li /** просмотрите список файлов корневого каталога. В первом столбце указаны индексные дескрипторы файлов (**inode**). Почему номера индексных дескрипторов основных подкаталогов так сильно различаются? Можете ли вы из выведенной на экран информации определить размер логического блока на диске (1, 2 или 4 Кб)?
8. Аналогичным образом посмотрите начало списка файлов одного из каталогов, например:

```
ls -li /bin | more
```

   Сравните номера **inode** файлов, входящих в данный подкаталог. Какие вы можете сделать выводы относительно размещения файловых объектов на дисковом пространстве?
9. С помощью команды **cat** прочитайте файл **/etc/mtab**. Зафиксируйте в отчете смонтированные устройства и файловые системы, а также права на их монтирование. Запомните наименование логического раздела жесткого диска, на котором смонтирована файловая система **ext*fs** (например, **/dev/sda3**). К этому имени файла-устройства (далее оно будет обозначаться как **<dev>**) вы будете обращаться, запуская дисковый редактор **extview**.
10. Перейдите в каталог **/home**. С помощью команды

```
fillfile file_size symbol
```

   создайте файл, размер которого больше размера одного сектора, но

меньше размера одного блока. Например, с помощью команды **fillfile 600 b** вы создадите в текущем каталоге файл с именем **b** объемом 600 байтов, который содержит повторяющиеся символы **b**. С помощью таких файлов с определенным наполнением вам будет удобно наблюдать за дисковым пространством.

11. Командой **ls -li** просмотрите список файлов в текущем каталоге и найдите созданный файл. В первом столбце списка найдите и запишите номер индексного дескриптора файла. С помощью команды

**extview -i <inode> <dev>**

(вместо **<dev>** укажите нужный Linux-раздел на диске) выведите индексный дескриптор данного файла. По размеру файла и иным признакам убедитесь в том, что вы действительно наблюдаете **inode** созданного вами файла. Письменно ответьте на вопросы:

- К какому типу относится данный файл?
- Каковы права доступа на данный файл у владельца, группы владельца и прочих пользователей?
- Что означает число **block count**?
- Сколько блоков с непосредственной адресацией выделила система под данный файл?
- Запишите шестнадцатеричный номер блока данных этого файла.

12. С помощью команды

**extview -b <block_number> <dev> | more**

выведите поэкранный дамп блока данных файла. Убедитесь в том, что он действительно заполнен определенными повторяющимися символами.

13. Командой **rm <file_name>** удалите созданный файл. С помощью дискового редактора вновь посмотрите таблицу **inode** и блок данных данного файла. Что с ними произошло?
14. С помощью программы **fillfile** вновь создайте в текущем каталоге файл с другим именем и заполнением. Размер очередного файла должен быть немного меньше предыдущего (например, 500 байтов).
15. С помощью редактора посмотрите таблицу **inode** и блок данных повторно созданного файла. Убедитесь в том, что это действительно он. Остался ли в блоке данных информационный «мусор» от прежнего файла?
16. Если вы обнаружили, что утилита **extview** выводит сведения об уже удаленном файле, это означает, что содержимое дискового кэша, откуда берет информацию утилита **ls**, еще не записано на диск, с которым работает **extview**. Принудительно сохранить на диск вновь созданный файл можно путем перезагрузки системы (**reboot**) либо с помощью дисковой утилиты командой

**hdparm –f <dev>**

17. Вновь удалите созданный файл. Создайте третий файл с иным именем и размером более одного блока (например, 5000 байтов). Убедитесь в наличии и заполнении файла, зафиксируйте номер последнего блока данных. Вновь удалите файл.
18. Создайте четвертый файл с отличающимся именем и размером менее одного блока (например, 3000 байтов). После того как вы убедитесь в создании очередного файла, посмотрите содержимое последнего блока, оставшегося от третьего файла. Какие вы можете сделать выводы?
19. Создайте пятый файл с размером 10000 байтов. С помощью команды

**chattr +s <file_name>**

установите для данного файла дополнительный атрибут, обозначающий гарантированное стирание блоков данных при удалении файла. Убедитесь, что индексные данные, выводимые командой **extview**, отображают дополнительные атрибуты файла.

20. Удалите пятый файл, после чего проверьте ранее занятые им **inode** и блоки данных. Сделайте выводы и занесите их в отчет. (Поддержка некоторых дополнительных атрибутов файлов реализована не во всех версиях ОС.)
21. С помощью команды **extview -b 0 <dev> | more** запустите дисковый редактор на поэкранный вывод содержимого суперблока (блок номер 0). Первые 1024 байтов (**400h**) блока отведены для размещения загрузчика (**LILO** или **GRUB**), но могут пустовать, если загрузочная программа размещается в Master Boot Record (MBR). Суперблок начинается со смещения **400h** и имеет размер 1024 байтов. С помощью справочных данных о файловой системе **ext*fs**, приведенных в главе 4 учебного пособия и в Приложении 2, исследуйте содержимое суперблока. Обратите внимание на то, что дамп памяти выводит шестнадцатеричные слова в обратном порядке. Для перевода чисел из шестадцатеричной системы счисления в десятичную из пользовательской консоли запустите калькулятор **bc** и задайте его внутренней командой **ibase=16** шестнадцатеричный ввод чисел. Теперь, вводя шестнадцатеричное число (символы A–F должны быть заглавными!) и нажимая **<Enter>**, получаем его десятичный эквивалент. В результате исследования полей суперблока письменно ответьте на вопросы:

- Сколько файлов может быть создано в данной файловой системе?
- Чему равен размер одного логического блока?
- Каков размер одного индексного дескриптора?
- Какой объем диска зарезервирован для нужд суперпользователя?
- Сколько блоков на момент исследования занято?
- Когда в последний раз проверялась файловая система? Каковы результаты проверки?
- Какой номер у индексного дескриптора файлового журнала?

- Что содержит неиспользуемая часть суперблока?

22. Рассмотрите описатели групп блоков **Group Descriptors**, которые обычно располагаются в первом блоке. Выберите первый из описателей размером 32 байта (две строки по 16 байтов). Найдите в нем и запишите в отчет шестнадцатеричные адреса:

- битовой карты блоков (**block bitmap**),
- битовой карты индексных дескрипторов (**inode bitmap**),
- таблицы индексных дескрипторов (**inode table**).

23. По аналогии прочитайте описатель 10-й группы блоков. Найдите в нем адреса битовых карт блоков и индексных дескрипторов. С помощью **extview** и утилиты **dd** рассмотрите битовые карты и сделайте выводы об алгоритмах использования дискового пространства.
24. Посмотрите шестнадцатеричный дамп одного из индексных дескрипторов в **inode table** 10-й группы блоков (при использовании обычного 128–байтного **inode** записи в блоке идут с интервалом в **80h** байтов). С помощью таблицы, приведенной в главе 4 учебного пособия, определите и отразите в отчете:

- тип этого файла и права доступа к нему,
- идентификатор владельца (выяснить, кому он принадлежит),
- размер файла в байтах,
- число имен этого файла,
- наличие дополнительных атрибутов этого файла,
- число блоков, занимаемых файлом,
- номера блоков данных с непосредственной и косвенной адресацией.

25. С помощью редактора **extview** и утилиты **dd** просмотрите несколько блоков данных исследуемого файла.
26. По имеющимся у вас данным определите, **inode** какого файла вы рассматривали. Для этого следует определить порядковый номер **inode** в своей группе, а также суммарное число индексных дескрипторов в 9 пропущенных группах. Для нахождения имени (или имен) файла по его номеру используйте возможности утилиты **find** (см. справку о командах). Укажите в отчете полный путь к этому файлу.
27. С помощью дискового редактора посмотрите, что представляет собой файл каталога. Для этого, воспользовавшись командой **ls -li /**, найдите номер **inode** одного из подкаталогов первого уровня, откройте его с помощью **extview**, найдите номер первого непосредственно адресуемого блока и изучите его. Сравните с форматом каталога, приведенным в теоретической части пособия. Сделайте выводы.
28. Если вы исследуете журнализируемую файловую систему, найдите в суперблоке индексный дескриптор файлового журнала и откройте его для просмотра. Обратите внимание на объем и расположение журнала. Изучите содержимое блока косвенной адресации.
29. Аналогичным способом посмотрите, что представляет собой файл сим-

волической ссылки. Обратите внимание на то, где и в каком виде в файле символической ссылки хранится путь к целевому файлу. Если не сумеете найти ответ, найдите и рассмотрите шестнадцатеричный и символьный дампы индексного дескриптора символической ссылки. Сделайте выводы и отразите их в отчете.

30. Проведите наблюдения за временными отметками обычных файлов и каталогов. Для этого можно использовать права одного из пользователей, создав от его имени два каталога **time1** и **time2**. В одном из каталогов с помощью команды **echo** требуется создать несколько текстовых файлов. С помощью утилиты **stat** вывести и записать в табл. 1 все временные отметки созданных объектов. Какую информацию можно извлечь из временных отметок файлов?

Таблица 1

| № п/п | Команда | Файл | | | Каталог | | |
|---|---|---|---|---|---|---|---|
| | | Время A | Время C | Время M | Время A | Время C | Время M |
| 1 | **ls** | | | | | | |
| 2 | **ls -l** | | | | | | |
| 3 | **touch** | | | | | | |
| 4 | **stat** | | | | | | |
| 5 | **chmod** | | | | | | |
| 6 | **chown** | | | | | | |
| 7 | **mv** | | | | | | |
| 8 | **cp** | | | | | | |

## Контрольные вопросы

1. Как файловая система **ext2fs** операционной системы Linux использует дисковое пространство? Можно ли по индексным дескрипторам файлов и каталогов представить себе их взаимное расположение на жестком диске?
2. Какую роль в монтировании сменных носителей и логических разделов жесткого диска играет файл **fstab**? Чем отличаются автоматическое и «ручное» монтирование?
3. Почему дискету не следует извлекать из дисковода до ее размонтирования? Что произойдет в случае пренебрежения этим правилом?
4. В параметрах файла **fstab** могут быть указаны разрешения монтирования сменных машинных носителей для **user** и **users**. Чем отличаются эти разрешения?
5. Механизмы образования и удаления информационного «мусора» в файловых системах ОС Linux.
6. Экономно ли используется дисковое пространство в ОС Linux? Приведите доказательства своего ответа.
7. Пользователь может читать и записывать информацию на сменные машинные носители, даже если ему запрещено их монтировать. Как можно запретить использование сменных машинных носителей?
8. Какую опасность для системы представляют файлы, не имеющие владельца?
9. Как можно найти файл по номеру его **inode**?

## ЛАБОРАТОРНАЯ РАБОТА № 3
## «Восстановление данных программными средствами ОС Linux»

1. Изучите теоретический материал, изложенный в параграфах 3.3 – 3.4, 4.3 учебного пособия.
2. Зарегистрируйтесь в первом текстовом терминале с учетной записью **root**.
3. С помощью **Alt+F2** откройте второй текстовый терминал и зарегистрируйтесь как пользователь **user1**.
4. Используя права обычного пользователя, введите команду **fdisk –lu**. Повторите ввод команды с правами суперпользователя. Посмотрите, какие устройства долговременной памяти соединены с аппаратными интерфейсами персонального компьютера, из каких логических разделов состоит дисковая память и какие операционные системы на них установлены.
5. Создайте в файловой системе новый пустой каталог **mkdir /mnt/abcd**, который будет служить точкой монтирования. С помощью команды **locale** установите, какая из кодировок по умолчанию поддерживается загруженной операционной системой (**koi8-r, cp1251** и т. д.).
6. Используя команду

**mount –t <type> –o iocharset=koi8-r <dev> <dir>**,

где **<type>** – монтируемая файловая система (**ntfs, vfat** и др.),
**<dev>** – файл блочного устройства, соответствующий монтируемому логическому разделу (например, **/dev/sda3**),
**<dir>** – созданная точка монтирования (**/mnt/abcd**),
примонтируйте к дереву каталогов Linux файловую систему ОС Windows*. Опция **iocharset=koi8-r** указывает на используемую кодировку, что позволит просматривать имена файлов в неискаженном виде. После монтирования запустите редактор **Midnight Commander** и посмотрите каталоги и файлы примонтированной системы. Если русскоязычные имена файлов отображены неправильно, демонтируйте раздел в соответствии со следующим пунктом и попробуйте задать иную кодировку.

7. Демонтируйте файловую систему Windows* с помощью команды **umount <dev>** или **umount <dir>** (вместо **<dev>** или **<dir>** укажите конкретные значения). К моменту монтирования все каталоги и файлы смонтированной файловой системы должны быть закрыты.
8. Получите у преподавателя поврежденный сменный машинный носитель (дискету, оптический диск, USB-Flash) с информацией, которую необходимо спасти. С помощью команды **dd** поблочно скопируйте часть дисковой памяти в файл **/home/abcd** (этот файл будет создан).

Количество копируемых блоков зависит от состояния поврежденного носителя. Обязательно укажите последнюю опцию команды **conv=noerror**, позволяющую пропускать отсутствующие и поврежденные блоки. Рекомендуемый синтаксис команды

```
dd if=/dev/fd0 of=/home/floppy bs=512 count=10
conv=noerror,fsync.
```

Размер полученного файла должен соответствовать суммарному объему правильно прочитанных секторов дискеты. С помощью редактора (F3) файлового менеджера **Midnight Commander** или команды **mcedit** прочитайте созданный файл-образ машинного носителя. Убедитесь в том, что информация скопирована достоверно (в противном случае попытку создания файл-образа придется повторить).

9. Пользуясь командой **dd** , запишите произвольный текст в первый сектор анализируемой дискеты. Например:

```
echo 1234567890 | dd of=/dev/fd0 bs=1 seek=10 count=10
```

Затем просмотрите первый сектор дискеты командой **cat /dev/fd0 | more** и найдите следы своей записи.

10. С помощью команды

```
dd if=/dev/hda bs=512 count=1|xxd|more
```

выведите на монитор содержимое первого сектора главной загрузочной записи и рассмотрите ее. После того как вы убедились в том, что видите именно Master Boot Record (MBR), скопируйте загрузочную запись в файл. Обычно подобные копии создаются на сменном носителе и предназначены для использования в случае порчи жесткого магнитного диска. Запишите в тетради команду обратного копирования (ее вводить не нужно!) и покажите преподавателю. Будьте внимательны при вводе команд, иначе вы можете сделать жесткий диск недоступным!

11. При наличии у вас собственного носителя на полупроводниковой памяти USB-Flash (кафедра не может обеспечить учебный процесс подобными устройствами) примонтируйте его к файловой системе. После подключения носителя к USB-интерфейсу запускается команда **fdisk –lu**. Допустим, отображается единственный раздел **/dev/sda1**. Он монтируется командой

```
mount –t vfat –o iocharset=koi8-r /dev/sda1 /mnt/usb
```

(кодировка символов может быть иной, а вместо каталога **/mnt/usb** можно указать другой пустой каталог). С помощью команды **cat** или файлового менеджера просмотрите файлы на примонтированном носителе.

12. Командой **umount /mnt/usb** отключите устройство и извлеките его. Путем просмотра файлов в каталоге **/var/log** найдите запись, которая зафиксировала факт подключения мобильного носителя (предполо-

жительно это файл **/var/log/message**).

13. С помощью команды **cat /root/.bash_history** просмотрите текстовый файл истории команд. Найдите в нем команды, введенные вами. Каким образом их лучше удалить? Если вы вводили команды от имени одного из зарегистрированных пользователей, то аналогичный файл истории команд вы найдете в его домашнем каталоге.
14. Вспомните, в каком порядке происходит логическое удаление файлов в файловых системах **ext2fs** и **ext3fs**.
15. Используя утилиту **fillfile**, по методике, изложенной в лабораторной работе № 2, создайте в рабочем каталоге пользователя пять текстовых файлов с именами от “A” до “E” размером от 500 до 10000 байтов. Убедитесь в их создании. С помощью редактора **extview** посмотрите индексный дескриптор и блок данных одного из них.
16. С помощью утилиты **fillfile** в рабочем каталоге пользователя создайте дополнительно три файла с длинными именами (например, для того, чтобы удобнее было отслеживать файловые записи в каталоге).
17. Найдите и просмотрите блок данных, выделенный рабочему каталогу пользователя. Найдите в нем и исследуйте записи обо всех созданных вами файлах.
18. Зафиксируйте в отчете номера **inode** и логических блоков, выделенных каждому созданному файлу. Эти данные вы будете использовать при анализе фрагментов удаленных файлов.
19. С помощью команды **rm –f** удалите все созданные вами файлы. Используя команду **ls –l,** просмотрите рабочий каталог пользователя и убедитесь, что логическое удаление файлов состоялось.
20. Создайте еще семь новых файлов с произвольными именами. Просматривая содержимое рабочего каталога пользователя с помощью команды **ls –li**, найдите вновь созданные файлы и установите, какие из индексных дескрипторов ранее удаленных файлов система передала новым объектам.
21. С помощью редактора **extview** просмотрите файловые записи в блоке данных рабочего каталога пользователя и установите, какие фрагменты удаленных файлов подлежат восстановлению. Обратите особое внимание на записи удаленных файлов с «длинными» именами. Результаты осмотра отразите в отчете.
22. С помощью команды **cat** прочитайте файл **/etc/fstab** и определите, как обозначается файл специального устройства, за которым закреплен логический раздел жесткого диска с файловой системой **ext2fs**.
23. Командой **debugfs** войдите в среду отладчика файловой системы и откройте логический раздел **ext2fs** для чтения (команда **open device**). Вместо **device** укажите файл специального устройства с логическим разделом Linux.
24. Вводя команду отладчика **lsdel**, посмотрите список **inode** удаленных

файлов. Найдите среди них номера файлов, которые были созданы и удалены вами (в файловой системе **ext3fs** отладчик **debugfs** удаленных файлов не обнаружит).

25. Выберите один из обнаруженных индексных дескрипторов удаленных вами файлов и с помощью команды **stat <inode>** установите, что записано в таблице и сохранились ли в ней ссылки на блоки данных. Обратите внимание на атрибуты, указывающие на удаление файла.
26. Если файл подлежит восстановлению, установите в «1» бит соответствующего **inode** в битовой карте индексных дескрипторов. Делается это с помощью внутренней команды отладчика **seti <inode>.**
27. С помощью команды отладчика **mi <inode>** сбросьте время удаления файла и установите в «1» число ссылок на файл.
28. Попытайтесь восстановить символьное имя удаленного файла. Это делается с помощью команды отладчика **ncheck <inode>.**
29. Командой **close** закройте логический раздел **ext2fs** на жестком диске и затем, вводя **quit,** выйдите из среды отладчика **debugfs**.
30. Выведите листинг рабочего каталога пользователя и проверьте, удалось ли вам восстановить удаленный файл. Сделайте выводы и отразите их в отчете.
31. Повторите операцию восстановления для другого удаленного файла.
32. По материалам Приложения изучите возможности утилиты **extview** по восстановлению удаленных файлов.
33. Предложите собственную методику восстановления данных удаленного файла, если его индексный дескриптор уже оказался занятым.
34. Воспользовавшись утилитой **shred,** удалите один из файлов, созданных в последнюю очередь. Используя вышеизложенную методику, попытайтесь найти составные части удаленного файла. Сделайте выводы и отразите их в отчете.

### Контрольные вопросы

1. С какого времени файл можно считать логически удаленным? Каковы признаки логического удаления файла?
2. В какой последовательности «исчезают» компоненты файла при его логическом удалении в файловых системах **ext2fs** и **ext3fs**?
3. Какие методики восстановления логически удаленных файлов вам известны? В чем они заключаются?
4. Каким образом можно найти блоки данных, ранее принадлежавшие удаленному файлу (предполагаем, что файл был текстовым и его содержание приблизительно известно). Какие команды для этого следует использовать?
5. Каким образом можно «спасти» остатки логически удаленного файла, если его индексный дескриптор уже занят?
6. Как скопировать данные с поврежденного машинного носителя?

7. Какие требования предъявляются к гарантированному удалению конфиденциальной информации?
8. Какие программные средства гарантированного удаления данных имеются в современных операционных системах Linux?
9. Восстановите поврежденный или логически удаленный файл, предложенный преподавателем.

## ЛАБОРАТОРНАЯ РАБОТА № 4
## «Реализация политики разграничения доступа средствами ОС Linux»

1. Зарегистрируйтесь в системе с правами суперпользователя.
2. С помощью команды **`cat /etc/group`** откройте для просмотра файл групп. *Файл представляет собой таблицу, каждая строка которой является отдельной учетной записью, состоящей из 4 полей, разделенных двоеточиями. Первое и третье поле соответственно – имя и номер (GID) группы, второе – обычно отсутствующий групповой пароль, четвертое – имена пользователей, включенных в данную группу дополнительно. Они обладают групповыми правами на файлы владельцев, входящих в данную основную группу.*
3. С помощью команды **`cat /etc/passwd`** откройте для просмотра файл учетных записей. *Файл представляет собой таблицу, каждая строка которой является отдельной учетной записью, состоящей из 7 полей. Обратите внимание на характерные разделители полей (регистрационное имя, признак пароля, идентификатор пользователя, идентификатор группы, дополнительная информация, домашний каталог, имя командного процессора) в виде двоеточия. Символ **x** в поле признака пароля указывает на то, что хэшированный пароль находится в другом файле –* **`/etc/shadow`**.
4. Аналогично изучите содержимое «теневого» файла паролей **`/etc/shadow`**. *Он также представляет собой таблицу, каждая строка которой состоит из 9 полей, разделенных двоеточиями (регистрационное имя, хэшированный пароль, контрольные сроки в днях, среди которых:*

- *число дней с 01.01.70 до дня последнего изменения пароля,*
- *минимальное число дней действия пароля со дня его последнего изменения,*
- *максимальное число дней действия пароля,*
- *число дней до устаревания пароля, за которые система начнет выдавать предупреждения,*
- *число дней со времени обязательной смены пароля до блокировки учетной записи,*
- *день блокировки учетной записи).*

*Последнее, девятое, поле зарезервировано и не используется.*

5. Запустите утилиту **`pwck`** и проверьте с ее помощью файлы учетных записей **`/etc/passwd`** и **`/etc/shadow`** на отсутствие ошибок. Разберитесь в обнаруженных ошибках и, если они представляют угрозу, покажите их преподавателю. Самостоятельное редактирование парольных файлов, не предусмотренное настоящим заданием, запрещено.
6. С помощью команды **`groupadd omega`** создайте новую группу без указания ее числового идентификатора. Откройте для просмотра файл

групп и найдите в нем созданную запись. Обратите внимание на номер созданной группы и запишите его.

7. С помощью команды **useradd –m john** зарегистрируйте в системе нового пользователя. Аргумент **–m** указывает на необходимость создания домашнего каталога пользователя с его именем. После успешной регистрации, не назначая пользователю пароль, посмотрите содержимое текстовых файлов **/etc/passwd** и **/etc/shadow**. Обратите внимание на заполнение второго поля в созданной учетной записи, которая будет располагаться в конце каждого из файлов. Проверьте наличие и содержимое домашнего каталога пользователя, включая «скрытые» файлы. В какую группу по умолчанию включен зарегистрированный пользователь?
8. Командой **passwd john** присвойте новому пользователю какой-либо простой пароль (например, 12345). Программа напомнит вам о том, что пароль слишком простой, но администратор может с этим мнением не считаться. После завершения процедуры регистрации вновь откройте теневой файл паролей **/etc/shadow**. Что изменилось в учетной записи этого пользователя?
9. С помощью **Alt+F2** перейдите во вторую консоль и зарегистрируйтесь с именем и паролем вновь созданного пользователя. Чем отличаются символы в приглашениях ввода команд для администратора и обычного пользователя?
10. С правами пользователя **john** создайте в его домашнем каталоге текстовый файл с содержимым, позволяющим его идентифицировать (например, **echo 'my name' john > /home/john/j**). Затем установите права доступа 0750 на каталог **/home/john** и 0640 на созданный текстовый файл.
11. Аналогично предыдущим пунктам создайте учетную запись второго пользователя **useradd –m –g omega braun**. Присвойте ему такой же пароль. Затем посмотрите содержимое теневого файла **/etc/shadow**. Сравните учетные записи созданных пользователей и попытайтесь объяснить, почему при одинаковых паролях их хэшированные функции различаются?
12. С помощью **Alt+F1** перейдите в первую консоль и правами **root** создайте командой **touch /etc/nologin** пустой файл, запрещающий регистрацию в системе новых пользователей. Перейдите в третью консоль (**Alt+F3**) и попытайтесь зарегистрироваться там. Убедитесь в бесполезности таких попыток.
13. Из первой консоли удалите правами **root** созданный файл **/etc/nologin**, переключитесь в третью консоль и повторите попытку входа в систему пользователем **braun**. На этот раз попытка входа должна закончиться удачно.
14. С правами пользователя **braun** создайте в его домашнем каталоге тек-

стовый файл с содержимым, позволяющим его идентифицировать (например, **echo my name braun > /home/braun/b**). Затем установите права доступа 0750 на каталог **/home/braun** и 0640 на созданный текстовый файл.

15. С правами пользователя **braun** попытайтесь войти в домашний каталог пользователя **john** и прочитать его файл. Сделайте выводы.
16. С помощью **Alt+F2** перейдите во вторую консоль и с правами пользователя **john** попытайтесь войти в домашний каталог пользователя **braun** и прочитать его файл. Сделайте выводы.
17. Перейдите в первую консоль и с правами администратора командой **usermod –G omega john** включите названного пользователя в дополнительную группу. Откройте для просмотра файл **/etc/group** и зафиксируйте изменения.
18. Правами пользователей **john** и **braun** из соответствующих консолей повторите попытки доступа к чужим каталогам и файлам. Объясните произошедшие изменения.
19. С помощью команд **who** и **w** посмотрите список пользователей, которые сейчас работают в системе. Какую информацию вам удалось из этого извлечь? Чем отличаются результаты, выведенные командами **who** и **w**?
20. Не выходя из консоли администратора, запустите редактор **mcedit** с именем файла учетных записей **/etc/passwd** в качестве аргумента. В учетной записи пользователя **john** вместо символа **x** во втором поле поставьте символ восклицательного знака. Сохраните (F2) изменения в файле и выйдите из редактора.
21. Перейдите (**Alt+F2**) во вторую консоль и командой **exit** завершите сеанс работы пользователя **john**. Попытайтесь вновь зарегистрироваться в системе с этим именем. Почему вам это не удалось?
22. Вернитесь в консоль администратора (**Alt+F1**) и с помощью редактора **mcedit** разблокируйте учетную запись пользователя **john**. Вновь из второй консоли войдите в систему с его именем.
23. Вернитесь в консоль администратора (**Alt+F1**) и попытайтесь удалить учетную запись пользователя **john** командой **userdel –r john** (аргумент **–r** позволяет удалить с учетной записью и домашний каталог пользователя **/home/john**). Почему система не позволяет удалить эту учетную запись?
24. С помощью команды **ps –elf** выведите на экран список процессов, исполняющихся в системе. В последних строках файла найдите процесс командной оболочки, закрепленной за пользователем **john**. Прочитайте во втором столбце числовой идентификатор этого процесса (PID) и «убейте» его командой **kill -9 PID**. Перейдите во вторую консоль и убедитесь в том, что сеанс пользователя **john** завершен. Теперь учетную запись можно удалить (но сейчас это лучше не делать).

25. Попытайтесь присвоить пользователю **braun** нулевой идентификатор (это можно сделать командой **usermod –u 0 braun**). Почему администратору в такой попытке было отказано?
26. Вновь с правами администратора запустите **mcedit** и откройте в нем файл **/etc/passwd**. Найдите учетную запись пользователя **braun** и замените его числовой идентификатор **UID** на **0**. Сохраните изменения в файле (F2) и завершите редактирование.
27. Перейдите в третью консоль и попытайтесь прочитать теневой файл паролей (**cat /etc/shadow**). Почему пользователю в этом было отказано? Командой **exit** завершите сеанс пользователя **braun**, а затем вновь зарегистрируйтесь с этим именем. Почему изменился символ в строке приглашения? Вновь попытайтесь прочитать файл паролей. Почему на этот раз все получилось? Какую информацию о пользователях на этот раз выдает команда **who**?
28. Из третьей консоли запустите редактор и верните в исходное состояние учетную запись пользователя **braun**. Сделайте выводы о роли учетных записей пользователей.
29. Из консоли администратора просмотрите текстовые файлы в каталоге **/var/log** и найдите записи аудита, в которых зафиксированы входы в систему администратора и пользователей. Имеется ли доступ к файлам аудита у обычных пользователей?
30. Вам необходимо создать учетные записи и определить права доступа для десяти (10) сотрудников: **w_gromov, n_kalinina, e_ivanova, r_klinova, b_rebrov, k_beglov, i_frolov, d_lavrov, m_kruglov, t_uporov**, работающих в одном подразделении и занятых созданием и редактированием текстовых документов различного уровня конфиденциальности. Разграничение доступа к информации должно быть произведено на основании следующих требований:

- допуск к секретным сведениям имеют четыре пользователя: **w_gromov, n_kalinina, b_rebrov, k_beglov;**
- три пользователя: **n_kalinina, b_rebrov, k_beglov** работают над созданием секретных документов, каждый по своему профилю. Их домашние каталоги и файлы должны быть полностью недоступными как друг для друга, так и для всех остальных, исключая **w_gromov;**
- три пользователя: **i_frolov, d_lavrov, e_ivanova** имеют допуск к конфиденциальной информации и работают над документами с соответствующим грифом. Они имеют право читать файлы с конфиденциальной информацией, созданные своими коллегами, без права их модификации;
- все секретоносители имеют право знакомиться с конфиденциальными файлами;
- три пользователя: **r_klinova, m_kruglov, t_uporov** могут ра-

ботать только с открытой информацией. Их файлы должны быть доступны для чтения каждому сотруднику подразделения без права модификации;

- **w_gromov** является редактором подразделения и имеет право читать и копировать файлы всех сотрудников и всех уровней конфиденциальности. Завершенные документы копируются пользователем **w_gromov** в его домашний каталог, который должен быть недоступен для всех остальных сотрудников подразделения;
- всем сотрудникам, включая редактора, запрещается вносить изменения и удалять документы других пользователей, независимо от уровня конфиденциальности.

31. Укажите в отчете, какие коллизии вы усматриваете в сформулированных требованиях? Как реализовать указанные требования таким образом, чтобы пользователи не могли по своему усмотрению изменять установленный порядок?
32. С помощью команды **groupadd** создайте четыре пользовательских группы: **alfa, beta, nabla, sigma.** Формат команды **groupadd -g GID group_name.** Идентификатор группы **GID** можно назначать произвольно, начиная с номера 100 **(**например**, groupadd –g 101 alfa).**
33. Создайте учетные записи для вышеуказанных десяти новых пользователей. Регистрационные данные (кроме паролей и групп) сведены в табл. 2. Пароли назначайте произвольно, длиной не менее 8 символов, не забывая фиксировать их в черновике отчета. Для пользователей **e_ivanova, r_klinova** задайте одинаковые пароли. Распределите сотрудников по группам таким образом, чтобы удовлетворить вышеперечисленным требованиям. Изобразите в отчете наглядную схему, поясняющую разграничение доступа сотрудников подразделения к компьютерной информации. Заполните требуемыми правами доступа ячейки табл. 3.

Таблица 2

| **Пользователь** | **UID** | **Пароль** | **Группа** | **Домашний каталог** | **Дата удаления учетной записи** |
|---|---|---|---|---|---|
| i_frolov | 2001 | | | /home/i_frolov | T+ 10 дней |
| m_kruglov | 2002 | | | /home/m_kruglov | T+ 30 дней |
| b_rebrov | 2003 | | | /home/b_rebrov | T+ 12 дней |
| d_lavrov | 2004 | | | /home/d_lavrov | T+ 60 дней |
| e_ivanova | 2005 | | | /home/e_ivanova | T+ 30 дней |
| t_uporov | 2006 | | | /home/t_uporov | T+ 15 дней |
| k_beglov | 2007 | | | /home/k_beglov | T+ 45 дней |
| n_kalinina | 2008 | | | /home/n_kalinina | T+ 30 дней |
| r_klinova | 2009 | | | /home/r_klinova | T+ 90 дней |
| w_gromov | 2010 | | | /home/w_gromov | не удалять |

Таблица 3

| Пользователь | Права доступа к объектам ФС | | | | | |
|---|---|---|---|---|---|---|
| | Файл редактора | Файлы «С» | | | Файлы «К» | Файлы «Н» |
| | | С1 | С2 | С3 | | |
| i_frolov | | | | | | |
| m_kruglov | | | | | | |
| b_rebrov | | | | | | |
| d_lavrov | | | | | | |
| e_ivanova | | | | | | |
| t_uporov | | | | | | |
| k_beglov | | | | | | |
| n_kalinina | | | | | | |
| r_klinova | | | | | | |
| w_gromov | | | | | | |

34. Пять первых пользователей (**w_gromov, n_kalinina, e_ivanova, r_klinova, b_rebrov)** зарегистрируйте с помощью команды **useradd.** Например, **useradd –u 501 –g sigma –d /home/n_kalinina –p v5g7K2S4 –e 2010-01-07 n_kalinina**. Параметр **-m** обеспечивает создание домашнего каталога пользователя, если он еще не существует. Прочие параметры команды можно не указывать. Идентификаторы пользователей **UID** назначать, начиная с 2001. Дату удаления учетной записи пользователя вводить в формате ГГГГ-ММ-ДД.
35. Пять последних пользователей зарегистрируйте с помощью командного файла **adduser**, которая запрашивает значения в интерактивном режиме. При вводе данных ориентируйтесь на подсказки системы [в квадратных скобках]. Все параметры, кроме имени пользователя, его идентификатора, имени группы, пароля и домашнего каталога, можно игнорировать. Для ввода параметра по умолчанию вводить **Enter**. В некоторых дистрибутивах ОС Linux командный файл **adduser** отсутствует, и в этом случае регистрацию остальных пользователей следует произвести аналогично предыдущему пункту. При использовании **adduser** символ подчеркивания в имени пользователя может не восприниматься. Если это так, замените этот символ в именах пользователей на символ дефиса.
36. Зарегистрировавшись администратором во вспомогательной консоли, отслеживайте из нее изменения, происходящие в файлах **/etc/passwd** и **/etc/shadow** по мере создания новых учетных записей.
37. Обратите внимание на то, что утилита **useradd** записывает пароли в файл **/etc/shadow** в открытом виде. Поскольку система воспринимает эту запись в качестве хэш-функции пароля, у пользователя с подобной учетной записью нет никаких шансов войти в систему. Правами администратора установите пользователям требуемые пароли с помощью команды **passwd.**

38. Регистрируясь в системе от имени пользователей **w_gromov**, **b_rebrov**, **d_lavrov** и **m_kruglov**, создайте их правами и в их домашних каталогах по одному текстовому файлу. Путем пробного доступа к этим файлам на чтение и запись от имени иных пользователей проверьте, удалось ли вам реализовать требуемую политику безопасности.
39. Правами администратора с помощью команды **usermod** по своему усмотрению измените основную группу пользователю **d_lavrov**. Проверьте, как изменились его права доступа к файлам иных пользователей.
40. Из первой консоли с помощью команды **su** измените права администратора на права пользователя **w_gromov**. Почему система не запрашивает пароль? С помощью команды **exit** верните себе права администратора. Был ли запрошен пароль? Почему?
41. Пользователь **d_lavrov** уволен за дисциплинарный проступок. С помощью команды **userdel d_lavrov** удалите его учетную запись. Кто теперь стал владельцем его домашнего каталога?
42. Зарегистрируйте вместо уволенного пользователя нового сотрудника **f_mironov** с предоставлением ему аналогичных прав (пароль должен быть новым!). Приобрел ли новый пользователь права на каталог ранее удаленного сотрудника? Для того чтобы подобное не происходило, при удалении учетных записей администратору необходимо вначале скопировать в недоступную для других директорию файлы пользователя, представляющие ценность для организации, а затем удалить учетную запись пользователя вместе с каталогом.
43. Пользователь **r_klinova** убыла в командировку сроком на две недели. Заблокируйте ее учетную запись любым из известных вам способов. Попытайтесь зарегистрироваться во второй консоли с правами **r_klinova** и убедитесь в том, что для этого пользователя система недоступна.
44. Зарегистрируйтесь во второй консоли с правами пользователя **k_beglov,** вызовите команду **passwd** и измените свой пароль. В качестве нового пароля введите **qwerty**.
45. Перейдите в консоль администратора и назначьте пользователю **k_beglov** новый пароль **zxcvbnm.** Затем с помощью команды **chage** (change aging – изменить информацию об устаревании) установите для этого пользователя минимальное время действия паролей, равное 5 дням. С какой целью устанавливается минимальный срок действия пароля?
46. Вспомните теоретический материал, связанный с файлом **/etc/sudoers.** Отредактируйте его таким образом, чтобы предоставить следующим пользователям дополнительные права за счет использования команды **sudo**:

- пользователю **e_ivanova** – право монтировать файловые системы типа **iso9660** на оптических дисках на своем компьютере,

- пользователю **b_rebrov** – право изменения владельца файлов на всех компьютерах локальной сети,
- пользователю **f_mironov** – право изменения учетных записей пользователей на своем компьютере без обязательного ввода пароля.

47. Из второй консоли с правами пользователя **f_mironov** создайте файл **cal 2011 > /home/f_mironov/cal2011.** С помощью команды **su** переключите консоль на пользователя **b_rebrov** и с помощью временно предоставленных ему привилегий передайте права на созданный **f_mironov** файл другому владельцу **n_kalinina.** Каким еще путем можно предоставить подобные права пользователям, не передавая им «опасных» полномочий администратора?
48. Просмотрите с правами администратора системные журналы в каталоге /**var/log** и убедитесь, что система зафиксировала факты присвоения полномочий администратора.

**Временная нейтрализация парольной защиты**

49. Отработайте вариант временной нейтрализации пароля администратора (например, для случая его утраты). Получите у преподавателя загрузочный оптический диск с ядром и минимальным набором утилит ОС Linux. Загрузите компьютер со сменного носителя. При необходимости измените порядок загрузки с использованием настроек в Setup BIOS. По завершении загрузки вы должны увидеть символ **#** и приглашение для ввода команды.
50. Введите команду **fdisk –lu** для отображения информации о разделах фиксированных и пристыкованных машинных носителях и установленных на них файловых системах. Найдите название файла блочного устройства, на котором установлен корневой раздел Linux (например, **/dev/sda6**).
51. Примонтируйте файловую систему Linux (предположительно это ФС **ext2fs** или **ext3fs**) на разделе HDD с помощью команды **mount –t ext2 /dev/hda6 /mnt**.
52. В случае успешного монтирования откройте файл учетных записей в смонтированной системе с помощью команды **vi /mnt/etc/passwd**. Текстовый редактор **vi** отобразит требуемый файл с мигающим курсором под первым символом учетной записи **root**. С помощью клавиш управления курсором переместите его под первый символ справа от двоеточия после слова **root**. Затем с помощью клавиши **x** требуется удалить все символы между первым и вторым двоеточием. В результате начало первой строки редактируемого файла должно выглядеть так: **root::0:0:**
53. Переключитесь в режим ввода команд редактора с помощью клавиши с двоеточием. После того как двоеточие и мигающий курсор после него появятся в последней строке, введите команду на сохранение изменений

и выход из редактора **wq <Enter>**. Если вы ошиблись и удалили лишние символы, проще перейти в режим команд и выйти из редактора без сохранения изменений с помощью **q! <Enter>** (затем следует повторить попытку редактирования).

54. При наличии в операционной системе встроенного редактора **Midnight Commander** процесс модификации учетной записи можно сделать более удобным. Файл открывается командой **mcedit /etc/passwd**, а результаты сохраняются функциональной клавишей F2.
55. Введите команду **reboot** для перезагрузки компьютера и извлеките компакт-диск. После загрузки Linux с жесткого диска на запрос об авторизации введите имя **root** и система зарегистрирует вас своим администратором без пароля. После завершения доступа требуется либо сменить пароль администратора с помощью команды **passwd,** либо восстановить удаленный признак пароля (по умолчанию это символ **x** между первым и вторым двоеточием в учетной записи **root** файла **/etc/passwd**). Редактирование файла паролей можно произвести встроенным редактором (F4) файлового менеджера **Midnight Commander**. Можно не использовать целиком файловый менеджер, ограничившись запуском его редактора **mcedit /etc/passwd**. Сохраните изменения (F2) и закройте программу.
56. Найдите в каталоге **/var/log** файл аудита, в котором зафиксирован вход в систему администратора без пароля.

### Контрольные вопросы

1. Достаточно ли трех базовых прав доступа к файлам для реализации в ОС Linux требуемой политики безопасности?
2. Какие изъяны вы усматриваете в использованных утилитах регистрации учетных записей пользователей?
3. Может ли пользователь закрыть для себя доступ к собственному файлу? Каким образом? Почему система не соотносит владельца файла ни с его группой, ни со всеми остальными?
4. Существуют ли способы ограничения доступа суперпользователя к некоторым конфиденциальным файлам?
5. Какие дополнительные права администратор может предоставить пользователям с помощью утилиты **sudo** и регистрационного файла **/etc/sudoers**?

## ЛАБОРАТОРНАЯ РАБОТА № 5
## «Исследование процессов в ОС Linux»

### Подготовка к работе

1. Зарегистрируйтесь в первом текстовом терминале с учетной записью **root.**
2. Комбинацией клавиш **Alt+F2** откройте второй текстовый терминал и зарегистрируйтесь как пользователь **user1** с паролем **1234567890**. В случае, если такой учетной записи нет, создайте ее по методике лабораторной работы 4.
3. Смонтируйте предоставленный преподавателем носитель к одному из подкаталогов каталога **/mnt**. Скопируйте с носителя один исполняемый файл с именем **signorer** в каталог **/bin.** Размонтируйте носитель и верните его преподавателю.

### Наблюдение за файловой системой /proc

4. Все наблюдения за содержимым этой директории следует производить с правами администратора. Поскольку суперпользователь обладает правами на запись практически любого файла в таблице процессов, ему следует быть весьма осторожным в работе с ними.
5. Командой **cd /proc** перейдите в директорию **/proc.**
6. С помощью команды **ls -l|more** выведите на экран содержимое файловой системы **/proc**. Обратите внимание на то, что в строках листинга отображаются странные каталоги и файлы, имеющие нулевой размер. Большая часть каталогов вместо имен в последнем столбце содержит номера. Для каждого активного процесса в каталоге **/proc** существует подкаталог, имя которого совпадает с идентификатором этого процесса **PID**. При создании процесса каталог с его номером автоматически генерируется и удаляется при завершении процесса.
7. Командой **ls -li|more** выведите также информацию об индексных дескрипторах отображаемых файлов и каталогов. С помощью редактора **extview** убедитесь, что эти номера либо не существуют, либо принадлежат другим файлам. В дальнейшем для удобства просмотра информации можете использовать **Midnight Commander.**
8. Обратите внимание на время модификации псевдофайлов. Оно равно времени запуска соответствующей команды.
9. Откройте несколько нумерованных подкаталогов подряд. Обратите внимание на то, что все подкаталоги содержат одни и те же «файлы». С помощью команды **ps –ef** найдите **PID** командного интерпретатора, с которым администратор работает из первой консоли, и в дальнейшем наблюдайте за каталогом с этим номером.
10. Прочитайте информацию, относящуюся к одному из процессов (например, **cat /proc/103/status**). Чем может быть полезна такая

информация для администратора и пользователя? Представляет ли эта информация интерес для информационного нарушителя?

11. Открывая файлы для просмотра, вы можете получить необходимую справочную информацию о системе. Именно отсюда ее берут многочисленные утилиты:

- открывая командой **cat** файл **version**, прочитайте версию ядра ОС, которая сейчас запущена, а также сведения об его компиляции. Примерно такую же информацию предоставляет утилита **uname -r**,
- командой **ls -l kcore** можно вывести информацию об объеме ОЗУ в байтах. Не следует читать файл **kcore** – это эквивалентно попытке чтения всей оперативной памяти. Если в этом чтении есть потребность, следует узнать конкретный адрес размещения интересующих данных и читать их из специального файла **/dev/mem** с помощью утилиты **dd**,
- запустив команду **cat cpuinfo**, получите информацию о центральном процессоре (или процессорах),
- открывая для чтения файл **mounts**, можно получить динамическую информацию о смонтированных устройствах. Сравните ее с содержимым каталога **/etc/mtab**,
- файл **devices** содержит список старших номеров зафиксированных системой символьных и блочных устройств,
- файл **meminfo** хранит сведения об использовании системной памяти. Получите аналогичную информацию с помощью команды **free**.

12. Зайдите в каталог, номер которого соответствует **PID** командного интерпретатора, который в данный момент обслуживает вашу консоль. Прочтите значения установленных переменных окружения оболочки из псевдофайла **environ**. Командой

```
HOME=$HOME:/tmp
```

временно измените свой домашний каталог. Пронаблюдайте соответствующие изменения в файле **environ**. Проверьте изменение переменной окружения командой **echo $HOME**. Введите эту команду еще раз. Какие изменения произошли?

Если сброс переменной при ее чтении не происходит, посмотрите, как она изменится после завершения процесса. Для оболочки – это команда **exit** и повторная регистрация в этой же консоли (проверьте, как меняется **PID** оболочки).

### Просмотр и анализ информации о процессах

13. Из консоли пользователя командой **ps -efl | more** выведите расширенный поэкранный список исполняемых процессов (перечень параметров для расширенного вывода информации можно уточнить с помощью электронного справочника **man ps)**. Разберитесь с выводимой информацией. Определите процессы:

- по типу: системные, демоны, пользовательские (тип процесса определяется по косвенным признакам, в частности по имени процесса в квадратных скобках, связи процессов с определенными владельцами и терминалами и др.);
- по состоянию **S:** (исполняющиеся – **R** или **O**, ожидающие записи на диск – **D**, ожидающие событий – **S**, приостановленные – **T**, зомби – **Z**),
- по текущему динамическому приоритету **PRI** (наименьшее значение у высокоприоритетных процессов),
- по относительному приоритету **NI.**

14. Комбинацией клавиш **Alt+F3** откройте третий текстовый терминал и зарегистрируйтесь в нем как суперпользователь. В этой консоли запустите утилиту **top** для текущего контроля процессов. Утилита позволяет отобразить наиболее активные процессы (сколько их помещается на экран) с достаточно полной информацией о них (для пользователя утилита представляет ограниченный набор выводимых параметров).
15. Из первой консоли создайте процесс **od /dev/zero > /dev/null.** В соответствии с введенной командой утилита **od** читает и выводит непрерывный поток байтов из «рога изобилия» в нулевое устройство. Переключившись в третью консоль, с помощью команды **top** просмотрите список наиболее активных процессов. Найдите и идентифицируйте запущенный процесс, найдите по идентификатору **PPID** его «родителя», определите его приоритет (возможно, это – величина переменная), долю загрузки центрального процессора **%CPU** и оперативной памяти **%MEM.**
16. Поочередно из первой и второй консолей с правами администратора и пользователя с помощью команды **od /dev/zero > /dev/null &** создайте по 2-3 одинаковых фоновых процесса.
17. По мере создания новых процессов отслеживайте в третьей консоли их текущий приоритет, загрузку процессора и памяти. Имеются ли различия в приоритете процессов, выполняемых от имени администратора и пользователя?
18. С консоли пользователя **user1** измените приоритет одного из принадлежащих ему процессов. Для этого воспользуйтесь командой **renice -10 PID**. Изменился ли относительный приоритет процесса?
19. Повторите предыдущий пункт, используя права **root.**
20. Переключитесь в консоль пользователя и измените приоритет одного из принадлежащих ему процессов командой **renice +5 PID.** Произошло ли изменение приоритета?
21. Проконтролируйте из третьей консоли изменение приоритетов запущенных процессов.
22. Из консоли пользователя восстановите приоритет ранее замедленного процесса командой **renice –5 PID.** Произошло ли восстановление прежнего приоритета? Почему?
23. С разрешения преподвателя завершите созданные вами процессы.

**Управление процессами**

24. С правами пользователя создайте в своей директории сценарий с именем **abcd.** Сценарий можно создать с помощью команды **cat:**

```
cat >abcd
#! /bin/bash
while : rem обратите внимание на пробел перед двоеточием!
# примечание вводить не нужно
do
echo HELLO!
done
Ctrl+d
```

25. Используя команду **chmod**, присвойте пользователю права на чтение и исполнение данного сценария. Запустите сценарий на исполнение (на экран должны непрерывно выводиться приветствия **HELLO!**).
26. Перейдите в третью консоль, с помощью команды **top** просмотрите список процессов и найдите в нем «зависший» процесс, запущенный пользователем (на самом деле это только имитация «зависания», которое пользователь легко может прекратить сам). Прочитайте идентификатор процесса **PID**.
27. Нажатием **Ctrl+C** из второй консоли остановите процесс. Как изменилось при этом состояние процесса?
28. Повторно запустите из второй консоли процесс, перейдите в первую консоль и отправьте «зависшему» процессу сигнал на останов (командой **kill** или внутренней командой **k** из работающей утилиты **top**).
29. Запустите ранее скопированную в каталог **/bin** утилиту **signorer**. Отправьте ей из этой же консоли несколько прерывающих сигналов в виде комбинаций клавиш (**Ctrl-C, Ctrl-\, Ctrl-Z**). Понаблюдайте за реакцией процесса. Как вы полагаете, по какой причине этот процесс не удается остановить?
30. Перейдите в другую консоль и отправьте «непослушному» процессу сигнал **kill -20 PID.** Как реагирует процесс на данный сигнал? Посмотрите в электронном справочнике, что означает данный сигнал.
31. С помощью команды **kill -9 PID** отправьте этому процессу сигнал принудительного завершения. С другой консоли проконтролируйте выполнение команды. Остановился ли процесс? Остался ли он в списке процессов? Какая программа на самом деле перехватывает и исполняет команду **kill -9 PID**?
32. С помощью команды **echo $PATH** поочередно из консоли администратора и пользователя **user1** выведите список директорий, в которых

производится поиск исполняемых файлов, заданных только по имени. В чем заключается различие выведенных списков? Почему в списке **PATH** администратора отсутствует текущий каталог (**.**)? Почему в списке **PATH** пользователя отсутствует каталог **/sbin**? Имеет ли пользователь возможность изменить порядок проверки каталогов для администратора?

33. Перейдите в первую консоль и повторите запуск утилит с правами суперпользователя.
34. Убедитесь в том, что пользователю разрешен запуск указанных утилит. Объясните, почему пользователь не может запустить утилиты с некоторыми «критичными» параметрами? Где, по вашему мнению, расположен механизм контроля за ходом исполнения таких команд (в ядре операционной системы, в командной оболочке, в самой утилите?). Ответ обоснуйте.
35. С правами пользователя скопируйте в свой рабочий каталог один из исполняемых файлов с параметром **SUID** каталога **/bin** (исполняемые файлы выделены цветом и символом *, а параметр **SUID** отмечен символом **s** в правах владельца на исполнение). Как изменились права доступа к файлу после его копирования?
36. Из второй консоли с правами пользователя скопируйте в свой домашний каталог утилиту, которую разрешено запускать только администратору (например, **chattr**). Копирование производите с параметрами, гарантирующими переход копии во владение пользователю. Убедитесь, что пользователь имеет на скопированный файл все необходимые права. Попробуйте использовать свою копию утилиты по ее назначению (в случае копирования утилиты **chattr** установите дополнительный атрибут **+i** одному из своих файлов). Сделайте выводы.

### Работа с консолями

В тексте задания, хотя это не вполне верно, под консолью и терминалом будет пониматься один и тот же объект.

37. С помощью команды **useradd –m <user_name>** создайте в системе учетные записи трех новых пользователей: **alisa**, **berta** и **wanda**. Присвойте им одинаковые пароли **12345** и войдите под их именами в систему из виртуальных консолей **/dev/tty2 (Alt+F2)**, **/dev/tty3 (Alt+F3)** и **/dev/tty4 (Alt+F4)**.
38. Поочередно запустите из консолей каждого из пользователей несколько команд: **tty; mesg; ls –l $tty; id –G** . С их помощью определите:

- как правильно именуются файлы специальных устройств, связанные с консолями,
- доступны ли консоли для вывода информации для членов специальной «консольной» группы,

- какие права доступа определены на консоли для владельцев, членов их группы и иных пользователей,
- включены ли пользователи в какие–либо общие группы. В какие именно?

39. Используя в каждой из консолей команды **whoami**, **who** и **w**, определите, зависит ли вывод информации от имени ее инициатора (исключая первую команду). Насколько подробна и достаточна информация, выводимая каждой из утилит?
40. Перейдите в консоль пользователя **berta** и заблокируйте ее на запись командой **mesg n**. Командой **ls –l $tty** выведите права доступа пользователей к этой консоли. В соответствующем столбце должно отобразиться что–то похожее на **crw-------** .
41. Перейдите в консоль пользователя **alisa**. Определите рабочую консоль пользователя **berta** и командой **write** отправьте ей произвольное сообщение. Проверьте, дошло ли отправленное сообщение до адресата. Почему?
42. С правами одного из обычных пользователей отправьте иным пользователям «широковещательное» сообщение с помощью команды **wall**. Получено ли сообщение? Почему? Может ли пользователь создать помеху для администратора, забрасывая его потоком сообщений? Можно ли сделать это скрытно? Повторите отправку сообщений с помощью команды **wall**, используя права **root**.
43. Используя права одного из обычных пользователей, попытайтесь полностью открыть для других (на чтение и запись) его терминал. Затем попробуйте из консоли другого пользователя записать что–либо в доступную консоль, а также прочитать из нее вводимую информацию. Объясните полученные результаты.
44. Используя права одного из обычных пользователей, попытайтесь полностью заблокировать его консоль (например, в целях имитации отказа системы).
45. С правами суперпользователя заблокируйте программный интерпретатор пользователя командой

**skill STOP tty3**,

после чего переключитесь в консоль «заблокированного» пользователя, убедитесь, что он беспомощен в отношении вводимой и выводимой информации.

46. От имени суперпользователя отправьте «заблокированному» пользователю сообщение с помощью команды

**echo Ваша консоль заблокирована > /dev/tty3**

Попробуйте от имени пользователя откликнуться на полученное сообщение.

47. Из консоли суперпользователя разблокируйте пользовательскую консоль командой

**skill CONT tty3**

Проверьте результат выполнения этой команды.

### Работа с каналами

48. Создайте неименованный канал (конвейер). Для этого можно использовать команду

**od /dev/zero | tr '\0' '1'| more**

Перейдя в другую консоль, с помощью команды **top** убедитесь, что были созданы 3 процесса: программы восьмеричного дампа **od**, программы перекодировки и транслитерации **tr**, заменяющей нули на единицы, и программы поэкранного вывода **more**. Все процессы запускаются и действуют одновременно. Различаются ли приорететы запущенных процессов?

49. Остановите запущенный процесс комбинацией клавиш **<Ctrl-C>.**

50. Изучите порядок создания и функционирования именованного канала. Для этого:

- правами **user1** создайте в каталоге хранения временных файлов именованный канал **mkfifo /tmp/fifo,**
- убедитесь в его создании и наличии прав на чтение из канала и запись в него **ls -l /tmp/fifo,**
- переключитесь во вторую консоль и введите команду чтения из канала со стандартного вывода (экрана) **cat < /tmp/fifo,**
- переключитесь в первую консоль и введите команду записи в канал со стандартного ввода (клавиатуры) **cat > /tmp/fifo,**
- наберите в первой консоли произвольный текст и нажмите **<Enter>** (буферизованный ввод),
- переключитесь во вторую консоль и прочитайте введенный текст. Повторите процедуру несколько раз,
- из первой консоли (где производится ввод в канал) комбинацией клавиш **<Ctrl+D>** введите команду на закрытие канала **FIFO**,
- удалите именованный канал командой **rm /tmp/fifo** .

### Исследование опасных команд

51. Правами пользователя создайте в каталоге **/tmp** файл неограниченного размера, замаскированный под имя временного файла:

**yes 12345 > /tmp/file2AF5DS &** ,

где после префикса **file** в имени файла должна стоять шестибайтная случайная последовательность символов. Дождитесь сообщения систе-

мы о нехватке памяти и с помощью команды **ls -l** выведите информацию о размере созданного файла. С помощью утилиты **cat** посмотрите содержание созданного файла и убедитесь, что он заполнен строками из цифр 12345.

52. С помощью утилиты **df** оцените расход дискового пространства.
53. Перейдите в консоль другого пользователя и командой

```
echo privet! > /tmp/a
```

попробуйте создать небольшой текстовый файл. Чем завершилась попытка?

54. Попытайтесь создать такой же файл **/tmp/a** с правами администратора. Сделайте выводы об опасности подобных атак. Командой **rm** удалите созданный файл **/tmp/file2AF5DS.**
55. С помощью команды **ulimit –f 100** из консоли пользователя установите лимит на размер создаваемых файлов (в блоках).
56. Повторно правами пользователя попытайтесь создать в каталоге **/tmp** файл неограниченного размера:

```
yes 12345 > /tmp/file2 &
```

57. Дождитесь сообщения о создании файла и командой **ls –l /tmp** проверьте его объем. Ограничения на иные системные ресурсы устанавливаются аналогично.
58. Произведите атаку на истощение доступного ресурса индексных дескрипторов. Для этого от имени пользователя создайте в каталоге **/tmp** командный файл **abcd** со следующим содержанием:

```
cat >abcd
```

```
#! /bin/bash
while :
do
mkdir 1
cd 1
touch 2
done
```

```
Ctrl+d
```

59. Командой **chmod** присвойте владельцу указанного файла права на чтение и исполнение. Запустите файл из командной строки. Перейдите в каталог администратора (или другого пользователя) и ожидайте результат. Объясните его. Удалите созданный командный файл.
60. Произведите атаку путем запуска большого числа процессов. Для этого от имени пользователя создайте в каталоге **/tmp** командный файл **abcd** следующего содержания:

```
cat >abcd
#! /bin/bash
export RUN=$((RUN+1))
echo $RUN…
$0
Ctrl+d
```

Командой **chmod** присвойте владельцу указанного файла права на чтение и исполнение.

61. Перейдите в консоль администратора и введите команду **ps -ef** (но пока не запускайте ее!).
62. Вернитесь в консоль пользователя и запустите созданный командный файл. Вновь перейдите в консоль администратора. Запустите команду **ps -ef** и наблюдайте пользовательские процессы. Попробуйте остановить этот процесс правами администратора.
63. Система должна остановить этот процесс и вывести в консоль пользователя сообщение о большом числе созданных файлов.
64. Сделайте выводы о степени опасности перечисленных атак.

### Контрольные вопросы

1. Чем отличаются системные процессы, «демоны» и пользовательские процессы?
2. Какую информацию можно извлечь из каталога **/proc**?
3. Как система распределяет процессорное время между конкурирующими процессами? Как на это влияет приоритет процессов?
4. Почему пользователю не разрешается повышать приоритет процессов? Усматриваете ли вы какую-либо опасность в возможности понижения приоритета пользовательских процессов?
5. В консоли, с которой вы работаете, произошло «зависание», вызванное неправильным исполнением одного из ваших процессов. Каким образом можно повлиять на возникшую ситуацию?
6. Какие угрозы безопасности связаны с использованием эффективного идентификатора **SUID**? Почему администратор должен учитывать и контролиролировать такие файлы?
7. В чем заключается различие между именованным и неименованным каналами? Для чего они используются?
8. Как можно заблокировать и разблокировать пользовательскую консоль?
9. Какие пользовательские атаки на исчерпание ресурсов системы вам известны?

## ЛАБОРАТОРНАЯ РАБОТА № 6
## «Исследование сетевых возможностей ОС Linux»

Лабораторная работа должна выполняться в проводной локальной вычислительной системе звездообразной топологии с концентратором (хабом). Убедитесь в наличии в исследуемой операционной системе Linux штатных утилит **ifconfig, netstat, nmap (netmap), nc (netcat)** и **tcpdump**. При отсутствии нужной утилиты самостоятельно либо с помощью преподавателя установите (скопируйте) ее.

1. Внимательно прочитайте задание и справку по синтаксису основных сетевых команд Linux.
2. Зарегистрируйтесь в системе в первой консоли с правами администратора.
3. Зарегистрируйтесь во второй консоли с правами пользователя.
4. Из консоли администратора с помощью команды **ifconfig -a** выведите на экран данные о текущем состоянии всех сетевых интерфейсов компьютера. Какую информацию из прочитанного вывода вы извлекли? Запомните, как обозначается основной Ethernet-адаптер (он может обозначаться **eth0, eth1, eth2**), и в дальнейшем используйте в сетевых командах это имя. Далее в тексте задания он упоминается как **eth0**.
5. Выведите информацию о сетевых интерфейсах с помощью команды **netstat –ai**, сравните возможности двух использованных утилит.
6. Активизируйте отключенную по умолчанию сетевую службу **telnet server**. Для этого запустите **Midnight Commander**, найдите конфигурационный файл **/etc/inetd.conf** и в режиме редактирования (F4) удалите символ комментария **#** перед строкой **telnet stream tcp nowait**, после чего сохраните изменения в файле. Если в системе используется демон **xinetd**, то активизация протокола производится в конфигурационном файле **/etc/xinet.d/telnet**, строка **disable = no**. Затем следует перезапустить систему.
7. Командой

```
ps –ef | more
```

выведите список процессов и убедитесь, что сетевой процесс **inetd** работает. Иначе его нужно запустить вручную командой **inetd**. Если исследуемая версия ОС не содержит сервера **telnet** (по причине его явной уязвимости некоторые версии Linux не предусматривают использования этого протокола), соответствующие пункты задания выполните с защищенной программной оболочкой Secure Shell (**SSH**).

8. Отключите сетевой адаптер командой **ifconfig eth0 down** (см. справку по сетевым командам). Присвойте сетевому интерфейсу временный MAC-адрес **A0:B1:C2:D3:E4:N**, где **N** – двузначный номер

компьютера в классе (при использовании в ЛВС одинаковых аппаратных или сетевых адресов возможны коллизии). Подключите адаптер к сети. Убедитесь в том, что его аппаратный адрес изменен.

9. Назначьте основному сетевому интерфейсу компьютера временный IP-адрес и маску подсети. Для этого введите команду

**`ifconfig eth0 192.168.0.N netmask 255.255.255.0`** ,

где **`N`** – номер компьютера. Повторным вводом команды **`ifconfig eth0`** убедитесь в том, что запись введенной информации произведена. Присвоенные сетевые адреса будут действовать до перезагрузки компьютера.

10. Проверьте работоспособность петли обратной связи, послав на свой же компьютер эхо-запрос **`ping 127.0.0.1`**. Убедившись, что отклики поступают, остановите зондирование комбинацией клавиш **`Ctrl-C`**.
11. Присвойте сетевому адаптеру дополнительный IP-адрес **`192.168.0.20+N,`** где **`N`** – номер компьютера. Проверьте прохождение ICMP-пакетов между сетевыми адресами на локальном компьютере.
12. Организуйте сеанс **`telnet`** на собственном компьютере, используя для этого интерфейс обратной петли или дополнительный IP-адрес. Для этого перейдите в консоль пользователя, наберите команду **`telnet 127.0.0.1`** (или **`telnet localhost`**) и после сообщения об успешном соединении введите **`login`** и пароль администратора. Почему вам было отказано в доступе? Почему соединение было закрыто? Можно ли считать эти меры надежной защитой, пресекающей передачу опасной информации по каналу связи в открытом виде?
13. Еще раз установите сеанс **`telnet`** через петлю обратной связи, используя на этот раз учетную запись обычного пользователя. После установления сеанса просмотрите список каталогов и файлов в нескольких директориях, список процессов и убедитесь, что в «удаленном» режиме доступа вы можете выполнять все команды, которые доступны пользователю, зарегистрированному на удаленном узле.
14. Перейдите в консоль администратора и с помощью команды **`w`** или **`who`** посмотрите, сколько сейчас пользователей в системе, кто они и с каких терминалов работают. Обратите внимание на то, как обозначаются локальный и удаленный терминалы.
15. С помощью команды **`netstat -a`** проконтролируйте список запущенных сервисов и их состояние. Найдите сеанс **`telnet`**.
16. Вернитесь в консоль пользователя и завершите локальный сеанс **`telnet`** командой **`exit`**. Получите сообщение о закрытии сетевого соединения. Проверьте эту информацию с помощью команды **`netstat`**.
17. Попробуйте войти на один из компьютеров сети в сеансе **`telnet`** (учетные записи пользователей на всех компьютерах должны быть одинаковы). Что потенциально опасного вы можете сделать на удаленном

компьютере? Приобретите на удаленном хосте права **root**. Проверив свои возможности по манипуляции удаленным компьютером, завершите сеанс командой **exit.**

18. Поскольку на вашем компьютере **telnet**-сервер активизирован, он тоже может стать объектом доступа. Периодически с помощью команд **netstat –a, ps –ef** или **w** проверяйте, не зафиксировали ли они подозрительные соединения, процессы или удаленных пользователей.
19. Войдите на произвольный узел по протоколу Secure Shell (команда **ssh** с указанием IP-адреса хоста, после запроса необходимо ввести пароль администратора). Проверьте свои возможности по манипуляции удаленным компьютером.
20. С помощью команды **arp –a** посмотрите таблицу соответствия сетевых и аппаратных адресов. Где расположен ARP-кэш и почему он сейчас пуст?
21. С помощью утилиты **ping** исследуйте локальную сеть, к которой подключен ваш компьютер в диапазоне адресов, идентифицирующих конкретный компьютер в сети, от 1 до 40 (**192.168.0.1/40**). Объясните, в чем уязвимость и неудобство такого метода сканирования.
22. Проверьте, обновилась ли после сканирования динамическая ARP-таблица. Если она содержит нужную вам информацию о сети, ее можно сохранить в файле командой **arp –a > /home/arp1** (через несколько минут информация о сетевых узлах будет изменена, и если тот или иной сетевой узел не проявляет активности, данные о нем в кэше будут утрачены). С помощью команды

**arp –s <IP-адрес> <MAC-адрес>**

создайте статическую **arp**-таблицу. Выясните местоположение этой таблицы.

23. Ознакомьтесь с синтаксисом команды **nmap** (netmap – карта сети). С помощью утилиты **nmap** исследуйте локальную сеть, к которой подключен ваш компьютер. Адреса в диапазоне можно вводить с использованием символов-джокеров и через дефис. Что вы можете сказать о полученной информации?
24. Исследуйте различные виды сканирования с помощью утилиты **nmap.** На этот раз в качестве объекта выберите один из компьютеров. Типы сканирования перечислены в справке по команде **nmap.** Какую дополнительную информацию о локальной сети вы получили? Каким образом эта утилита определяет тип операционной системы, управляющей сетевым узлом?
25. Отключите ответы своего сетевого адаптера на ARP-запросы других хостов командой

**ifconfig eth0 –arp**

С помощью команды **ifconfig eth0** убедитесь, что настройка выполнена.

26. Выждите несколько минут для сброса ARP-таблиц на компьютерах локальной сети и с одного из компьютеров сети с помощью утилиты **ping** постарайтесь обнаружить отклик своего компьютера. Достаточно ли надежно защищает компьютер от сканирования данная мера? Насколько нарушается при этом возможность работы в сети? Включите **arp**-отклик командой **ifconfig eth0 arp**.
27. Отключите сетевой интерфейс на своем компьютере с помощью команды **ifconfig eth0 down**. Повторите попытку обнаружения своего компьютера с одного из соседних узлов. Можете ли вы сами при этом проявлять какую-либо сетевую активность? Сделайте выводы. Вновь включите сетевой адаптер с помощью команды **ifconfig eth0 up**.
28. Переведите сетевой адаптер своего компьютера в режим перехвата всех пакетов с помощью команды **ifconfig eth0 promisc**. С помощью команды **ifconfig eth0** убедитесь, что настройка выполнена. При этом сетевой адаптер превращается в устройство подслушивания, но одновременно он становится очень уязвимым к сетевым атакам на отказ в обслуживании.
29. Ознакомьтесь с синтаксисом команды **tcpdump**.
30. С помощью утилиты **tcpdump** перехватите и прочитайте сетевые пакеты:

- отправленные из одного определенного адреса,
- являющиеся результатом сетевого обмена между двумя хостами,
- только Ethernet и IP-заголовки пакетов, направленных в адрес любого из компьютеров сети,
- сеансы **telnet** и **ssh** в локальной сети,
- ICMP-запросы в адрес вашего хоста,
- иные пакеты по указанию преподавателя.

31. Запишите несколько перехваченных пакетов в файл и просмотрите их в шестнадцатеричном коде. Найдите характерные поля и идентификаторы в заголовках канального, сетевого и транспортного уровней. Определите аппаратные и сетевые адреса, номера портов, иные характерные признаки, идентифицирующие сетевые протоколы.
32. Выясните, можно ли использовать **tcpdump** на сетевом интерфейсе, за которым не закреплен ни один IP-адрес.
33. Используя виртуальные сетевые адреса на локальном компьютере, организуйте сеанс сетевого копирования и канал наблюдения за ним. Для этого потребуется работа с трех консолей с правами администратора.
34. В первой консоли подготовьте (но пока не вводите!) команду для копирования (передачи) небольшого текстового файла, например файла паролей:

```
cat /etc/passwd | nc –w 2 192.168.0.22 3333
```

35. Во второй консоли подготовьте команду для приема копируемого файла:

```
nc –l –p 3333 > /home/password1
```

36. В третьей консоли подготовьте команду для перехвата копируемой информации:

```
tcpdump –i lo –xx –vv –s 100 > /home/password2
```

37. После проверки синтаксиса команд произведите их поочередный запуск: вначале **tcpdump** из третьей консоли, затем команду приема данных из второй консоли и, наконец, команду передачи данных. Дождитесь завершения команд в первой и второй консолях и затем комбинацией клавиш **Ctrl+C** остановите сеанс прослушивания **tcpdump**.
38. С помощью команды **cat** или **mcedit** просмотрите результаты копирования и перехвата. Обратите внимание на то, что **tcpdump** перехватил, по меньшей мере, семь пакетов, из которых три первых и три последних предназначались для установления и завершения TCP-сеанса. По этой причине большой объем копируемых данных должен представлять собой один непрерывный поток. Для этого рекомендуется использовать утилиту блочного копирования **dd** или утилиту **tar** (см. Справочник по командам Linux). Сравните между собой содержимое скопированного и перехваченного файлов.
39. На двух произвольно выбранных компьютерах в ЛВС с моноканалом произведите копирование большого массива данных. Предварительно рекомендуется произвести контроль установленных по умолчанию параметров фиксированного жесткого диска. В отношении HDD с IDE-интерфейсом для этого рекомендуется использовать команду **hdparm**. Для хронометража процедуры копирования рекомендуется выполнить совместно с утилитой **time**. Если результаты копирования некуда записывать, перенаправьте вывод в нулевое устройство.
40. На первом компьютере следует запустить команду

```
time dd if=/dev/hda count=10000|nc –w 2 192.168.0.22
3333
```

41. На втором компьютере следует запустить команду

```
time nc –l –p 3333 | dd of=/dev/null
```

42. На любом из компьютеров для контроля запустите программу **tcpdump**.

```
tcpdump –i lo –xx –vv –c 10 –s 100 > /home/hda
```

43. Оцените скорость копирования.
44. Выполните файловое копирование с помощью команд **tar** и **nc**. Команда **tar** используется для создания в качестве объекта копирования одного большого (здесь уместнее сказать – длинного) файла.

```
tar –czvf /home | nc –w 2 192.168.0.22 3333
```

## Контрольные вопросы

1. Как закрепить за одним сетевым адаптером несколько IP–адресов?
2. Как программным путем изменить аппаратный адрес сетевой карты?
3. Можно ли перехватывать трафик без установленного IP–адреса?
4. Для чего нужно отключать ARP–отклик?
5. Где находится ARP–кэш? Как долго хранятся в нем данные?
6. Перечислите известные вам виды сетевого сканирования.
7. Запишите команду перехвата шести **`icmp`**–пакетов, исходящих из узла с IP–адресом 192.168.0.3 .
8. Как производится сетевое копирование данных?

## ЛАБОРАТОРНАЯ РАБОТА № 7
## «Исследование беспроводной сети WiFi под управлением ОС Linux»

1. Загрузите операционную систему Linux с компакт–диска (Live-CD). Загрузку произведите в текстовом режиме. Зарегистрируйтесь в двух первых консолях с правами **root**, пароль администратора отображается в подсказке.
2. С помощью команды **ifconfig –a|more** проверьте доступные сетевые интерфейсы. Обратите внимание на беспроводный адаптер, обозначенный как **ath0** (или **ath1**).

```
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:e3:32:db:1b
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:1782 (1.7 KiB)
          Interrupt:19

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)
```

3. Аналогично посмотрите более точную информацию о состоянии беспроводного адаптера с помощью команды **iwconfig ath0**. В каком состоянии находится этот адаптер?

```
ath0      IEEE 802.11g  ESSID:""  Nickname:""
          Mode:Managed  Channel:0  Access Point: Not-Associated
          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=-94 dBm  Noise level=-94 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

4. Произведите конфигурацию виртуальных беспроводных интерфейсов. Виртуальные устройства следует создавать в заданной последовательности:

- с помощью команды **wlanconfig ath0 destroy** удалите виртуальную точку доступа,

- создайте новые виртуальные интерфейсы:

**wlanconfig ath create wlandev wifi0 wlanmode adhoc**

Слово **adhoc** означает, что адаптер будет работать в режиме одного из приемопередатчиков в одноранговой сети,

**wlanconfig ath create wlandev wifi0 wlanmode monitor**

Этот адаптер будет работать в режиме перехвата всех пакетов на заданном канале (**monitor**).

Созданные устройства автоматически получат порядковые номера **ath0** и **ath1**. Информацию об их состоянии можно получить с помощью команды **iwconfig**:

```
ath0      IEEE 802.11g  ESSID:""  Nickname:""
          Mode:Ad-Hoc  Channel:0  Cell: Not-Associated
          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=-94 dBm  Noise level=-94 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0

ath1      IEEE 802.11g  ESSID:""  Nickname:""
          Mode:Monitor  Channel:0  Access Point: Not-Associated
          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=-94 dBm  Noise level=-94 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

5. У созданных виртуальных устройств пока не установлены рабочие частоты и имя сети, а также иные параметры. Для установки некоторых параметров воспользуйтесь командой

**iwconfig ath0 channel 1 essid "abcd"**

После ввода команды состояние виртуального адаптера должно отображаться следующим образом:

```
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
```

6. Следующим шагом будет задание IP-адресов. Для этого используйте уже испытанную утилиту **ifconfig**:

**ifconfig ath0 192.168.0.1**

**ifconfig ath1 up**

Монитору нет необходимости иметь сетевой адрес, но активизировать его нужно. Результат, выведенный командой **ifconfig -a** после ввода параметров, отображается следующим образом:

```
ath0      Link encap:Ethernet  HWaddr 00:1e:58:a1:fd:bd
          inet addr:192.168.0.1  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:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

ath1      Link encap:UNSPEC  HWaddr 06-1E-58-A1-FD-BD-00-00-00-00-00-00-00-
00-00-00
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2011 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:295763 (288.8 KiB)  TX bytes:0 (0.0 B)

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:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:632 (632.0 B)  TX bytes:632 (632.0 B)
```

7. Теперь попытайтесь проверить сетевую активность иных беспроводных устройств в локальной сети. Для этого рекомендуется воспользоваться командой

**iwlist ath1 scan**

Методом проб и ошибок нетрудно убедиться, что в режиме сканирования радиодиапазона может работать либо монитор, либо одноранговая точка. Заставить работать в данном режиме виртуальную точку доступа не удастся. Результаты выполнения команды **iwlist** могут быть такими:

```
ath1      Scan completed :
          Cell 01 - Address: 02:1E:58:A1:FD:B9
                    ESSID:"abcd"
                    Mode:Ad-Hoc
                    Frequency:2.412 GHz (Channel 1)
                    Quality=50/70  Signal level=-45 dBm  Noise level=-95
dBm
                    Encryption key:off
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 6 Mb/s; 9 Mb/s
                              11 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s; 36 Mb/s
                              48 Mb/s; 54 Mb/s
                    Extra:bcn_int=100
```

```
                    Ex-
tra:wme_ie=dd180050f2020101830002a3400027a4000042435e0062322f00
          Cell 02 - Address: 06:1E:58:A1:FD:B9
                    ESSID:"abcd"
                    Mode:Master
                    Frequency:2.412 GHz (Channel 1)
                    Quality=53/70  Signal level=-42 dBm  Noise level=-95
dBm
                    Encryption key:off
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 6 Mb/s
                              9 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s; 36 Mb/s
                              48 Mb/s; 54 Mb/s
                    Extra:bcn_int=100
                    Ex-
tra:wme_ie=dd180050f2020101830002a3400027a4000042435e0062322f00
                    Extra:ath_ie=dd0900037f01010024ff7f
```

8. Следующей тактической задачей является перехват заголовков пакетов, для чего можно использовать утилиту **tcpdump**. При отсутствии внешней сетевой активности ее можно создать с помощью команды **ping**, адресованной в один из внешних или внутренних IP-адресов.

```
tcpdump –i ath1 –xx –vv –c 5
```

9. Произведите разбор данных, выведенных в заголовке и шестнадцатеричном дампе команды **tcpdump**.
10. Отключите WEP-ключ шифрования для двух виртуальных точек **ad-hoc**. Для этого используйте команду

```
iwconfig ath0 key off
```

11. Подготовьте передачу незашифрованного трафика между двумя точками в одноранговой сети. Для этого наберите соответствующие команды:

- на клиентском (приемном) узле

```
nc –l –p 2222 > /dev/null ,
```

чтобы не записывать ненужную информацию на диск.

- на серверном (передающем) узле

```
tar –czvt /etc/* |nc –w 2 192.168.0.1 2222
```

Разберитесь с синтаксисом и смыслом введенных команд.

12. На виртуальном мониторе запустите команду **tcpdump** в режиме перехвата содержимого передаваемых пакетов. После этого запустить процесс передачи незашифрованных данных вначале на клиентском, а затем на серверном узле. По информации, выводимой анализатором пакетов, убедитесь в перехвате открытой информации.
13. Произведите настройку криптоалгоритма WEP на клиентском и серверном узлах. Для этого воспользуйтесь командой

```
iwconfig ath0 key 0123-4567-89AB-CDEF
```

14. Повторите передачу, прием и перехват зашифрованных данных.
15. С помощью утилиты **airdump-ng**, задавая номер канала, имя файла

для выводной информации и адрес узла, произведите взлом WEP–пароля. В зависимости от выбранного пароля требуется перехватить не менее 100 Мб данных.

16. Сделайте выводы относительно надежности используемого механизма шифрования.

**Контрольные вопросы**

1. В каких режимах может работать беспроводный сетевой адаптер?
2. В чем состоит функциональное назначение точки доступа?
3. В чем заключаются преимущества и недостатки беспроводной одноранговой сети?
4. Как можно получить параметры открытой беспроводной сети?
5. Какими командами устанавливаются параметры беспроводного сетевого интерфейса?
6. В чем заключается процесс «взлома» криптозащиты беспроводной локальной сети?
7. Каким образом можно перехватить и отобразить данные, передаваемые в беспроводной локальной сети?

## ЛАБОРАТОРНАЯ РАБОТА № 8
## «Наблюдение и аудит в ОС Linux»

1. Зарегистрируйтесь в системе в консольном режиме с правами **root**. Намеренно сделайте несколько ошибок при вводе пароля, чтобы «отметиться» в журналах аудита.
2. С помощью команды **md5sum** вычислите и запишите контрольную сумму для одного из файлов в каталоге **/home/user1/qu1**. С помощью команды **echo** добавьте один символ в этот файл:

```
echo a >>/home/user1/qu1/jan
```

Вновь вычислите контрольную сумму файла и сравните два результата.

3. С помощью команды **md5sum** вычислите и запишите в файл контрольную сумму всех файлов в каталоге **/bin**.
4. С помощью команды **find** найдите в корневом каталоге файлы:

- имеющие атрибуты **SUID (find / -perm +4000);**
- имеющие атрибуты **SGID (find / -perm +2000);**
- файлы, которые разрешено модифицировать всем:

```
find / -type f -perm +2 ;
```

- файлы, не имеющие владельца **(find / -nouser**);
- файлы и каталоги, имеющие UID незарегистрированных в системе пользователей.

Объясните, какой интерес могут представлять для администратора указанные категории файлов?

5. Путем просмотра процессов убедитесь в том, что системный сервис **syslogd** запущен. В целях контроля его действий найдите в журналах аудита записи о попытках своего входа в систему (это может быть файл **secure** или **auth** в каталоге **/var/log**).
6. С помощью команды **grep** найдите уязвимые учетные записи в файле паролей:

- имеющие числовые идентификаторы суперпользователя и его группы: **UID**=0 и **GID**=0;
- имеющие право на вход в систему без ввода пароля (отсутствующее второе поле в виде **::**).

Если у вас есть проблемы с вводом команды и установкой соответствующих фильтров, вы можете просто внимательно просмотреть содержимое файла паролей **/etc/passwd**.

7. С помощью команды **id user_name** посмотрите список основной и дополнительных групп пользователей. Найдите дополнительные группы **floppy**, **cdrom** и **plugdev**, дающие право использовать сменные машинные носители **/etc/cdrom**, **/etc/fd0** и т.д. для бесконтрольного блочного копирования данных.

**Задание 1**

Просматривая файл истории командной строки одного из пользователей, администратор обнаружил следующий фрагмент (см. ниже). Требуется исследовать приведенную последовательность команд, выявить намерения пользователя и установить, удалось ли ему нарушить установленную по умолчанию политику безопасности. Наиболее важные команды сопроводите своими комментариями.

```
su
su
su
su root
whereis su
pwd
cp /din/su
cp /bin/su
cp --help
cp --help | more
cp /bin/su /home/petrov
ls -l /bin
ls -l /bin | more
chmod 777 /bin/su
ls -l /bin
cp /bin/chmod /home/petrov
chmod 777 /bin/su
/home/petrov/chmod 777 /bin/su
ls -l /bin | more
ls -l
/home/petrov/chmod 4777 chmod
ls -l
/home/petrov/chmod 777 /bin/su
ls -l
chown root chmod
fdformat --help
fdformat /dev/fd0
mke2fs /dev/fd0
cat /etc/fstab
man mount
ls /mnt
mount -t ext2 /dev/fd0 /mnt/floppy
ls -l /mnt/floppy
cd /mnt/floppy
echo #! /bin/bash > ls
echo chmod 777 /bin/su >> ls
cat ls
```

```
chmod 777 ls
umount /dev/fd0
cd
umount /dev/fdo
mount -t ext2 /dev/fd0 /bin
ls -l /
mount -t ext2 /dev/fd0 /mnt/floppy
/mnt/floppy/ls
```

**Задание 2**

Проанализируйте содержимое файла **/etc/passwd** и сделайте выводы в отношении потенциальных угроз безопасности. Проставьте в необходимых местах свои комментарии.

```
abcd: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:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/bin/bash
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody::0:99:Nobody:/:/bin/bash
vcsa:x:69:69:virtual console memory own-
er:/dev:/sbin/nologin
rpm:x:37:37::/var/lib/rpm:/bin/bash
xfs:x:43:43:X Font Server:/etc/X11/fs:/sbin/nologin
rpc:x:32:32:Portmapper RPC user:/:/sbin/nologin
mailnull:x:47:47::/var/spool/mqueue:/sbin/nologin
smmsp:x:51:51::/var/spool/mqueue:/sbin/nologin
gdm:x:42:42::/var/gdm:/sbin/nologin
nscd:x:28:28:NSCD Daemon:/:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
pcap:x:77:77::/var/arpwatch:/sbin/nologin
root:x:501:501::/home/ivanov:/bin/vi
gromov:x:502:1::/home/gromov:/bin/bash
user1:x:503:504::/home/user1:/bin/bash
user2:!!:504:505::/home/user2:/bin/bash
```

**Задание 3**

Почти в каждой книге по безопасности операционных систем UNIX/Linux отмечается опасность запуска пользователями специально подготовленных файлов (исполняемых или командных) с установленным атрибутом **SUID**. Указывается на необходимость установки в файле **/etc/fstab** запрета на запуск файлов со сменных машинных носителей (noexec), а тем более – с установленным атрибутом SUID (nosuid). Предлагается проверить реальность такой угрозы. Дискету или носитель USB-Flash с «опасным» сценарием подготовьте с правами администратора, отредактируйте файл **/etc/fstab**, а затем проверьте наличие угрозы с консоли пользователя. На машинном носителе с помощью утилиты **mke2fs** должна быть установлена файловая система **ext2fs**, в противном случае не удастся скопировать на нее файл с нужными атрибутами. По результатам выполнения задания сделайте выводы.

**Задание 4**

Предлагается выяснить, существует ли надежный способ исключить возможность копирования пользователем конфиденциальной информации с жесткого магнитного диска на сменный машинный носитель. При этом учтите, что запрет на монтирование дисков не исключает возможности блочного копирования компьютерной информации.

**Задание 5**

У вас возникли подозрения, что в ваше отсутствие злоумышленники получили физический доступ к компьютеру на вашем рабочем месте. На единственном жестком диске компьютера установлена операционная система Linux с офисными приложениями и конфиденциальной информацией. В составе компьютера имеются устройства чтения и записи на ГМД и CD-R(W), а также интерфейсы USB. Системный блок компьютера был опечатан, и печати остались неповрежденными. Можно ли установить факт и характер несанкционированного доступа в операционной среде компьютера? Если да, то каким образом?

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК


1. UNIX: руководство системного администратора. Для профессионалов. /Э.Немет, Г.Снайдер, С.Сибасс, Т.Хейн. 3-е изд. СПб.: Питер; Киев: Издательская группа BHV, 2003. 925 с.
2. Баррет Д.Дж. Linux: основные команды. Карманный справочник: пер. с англ. /Д.Дж. Баррет. М.: КУДИС–ПРЕСС, 2008. 288 с.
3. Бэндл Д. Защита и безопасность в сетях Linux. Для профессионалов / Д. Бэндл. СПб.: Питер, 2002. 480 с.
4. Киркланд Д. Linux. Устранение неполадок / Д. Киркланд, К. Тинкер, Г. Тинкер; пер. с англ. А.А.Слинкина. М.: НТ Пресс, 2009. 490 с.
5. Костромин В.А. ОС Linux на вашем персональном компьютере / В.А. Костромин (электронный вариант книги).
6. Кэрриэ Б. Криминалистический анализ файловых систем / Б. Кэрриэ. СПб.: Питер, 2007. 480 с.
7. Мандиа К. Защита от вторжений. Расследование компьютерных преступлений / К. Мандиа, К. Просис. М.: Лори, 2005. 476 с.
8. Митчелл М. Программирование для Linux. Профессиональный подход: пер. с англ. / М. Митчелл, Дж. Оулдем, А. Самьюэл. М.: Издательский дом «Вильямс», 2003. 288 с.
9. Робачевский А.М. Операционная система UNIX / А.М. Робачевский. СПб.: БХВ-Санкт-Петербург, 2000. 528 с.
10. Роббинс А. Linux: программирование в примерах / А. Роббинс. М.: КУДИЦ-ОБРАЗ, 2005. 656 с.
11. Скляров И.С. Программирование боевого софта под Linux / И.С. Скляров. СПб.: БХВ-Санкт-Петербург, 2007. 416 с.
12. Таненбаум Э. Современные операционные системы. / Э. Таненбаум. 2-е изд. СПб.: Питер, 2002. 1040 с.
13. Фленов М.Е. Linux глазами хакера / М.Е. Фленов. СПб.: БХВ–Петербург, 2005. 544 с.
14. Фликенгер Р. Взломы и настройка LINUX. 100 профессиональных советов и инструментов: практ. пособ; пер. с англ. / Р. Фликенгер М.: ЭКОМ, 2006. 288 с.
15. The new ext4 filesystem: current status and future plans / A. Mathor, M. Cao, S.Bhattacharya, A. Dilger, A. Tomas, and L. Vivier; Proceedings of the 2007 Ottawa Linux Symposium, 2007. 16 с.
16. Ext4 block and inode allocator improvements / A. Kumar, M. Cao, J. Santos, and A. Dilger; Proceedings of the 2008 Ottawa Linux Symposium, 2008. 14 с.
17. Tim Johns. Анатомия ext4. Режим доступа: www.ibm.com

## ПРИЛОЖЕНИЕ 1

# КРАТКИЙ СПРАВОЧНИК ПО КОМАНДАМ LINUX

### Просмотр содержимого файлов

**cat [arg] <file_name>** – просмотр содержимого текстового файла. Команда корректно работает лишь при выводе алфавитно-цифровых символов.

**od <file_name>** – вывод восьмеричного представления файла. Обычно используется для двоичных файлов.

**xxd <file_name>** – вывод комбинированного шестнадцатеричного и символьного представления файла. Обычно используется для двоичных файлов с текстовыми фрагментами.

**mcedit <file_name>** – открытие файла с произвольным форматом для чтения и редактирования в редакторе оболочки **Midnight Commander**.

### «Навигация» по файловой системе

**cd <dir>** − изменение текущего каталога. Варианты переходов:

- **cd** (без аргументов) – переход в «домашний» каталог,
- **cd /** – в корневой каталог,
- **cd ..** – в родительский каталог,
- **cd /home/user1** – переход в домашний каталог пользователя **user1**.

**pwd** (print working directory) – вывод имени текущего каталога.

**find <dir> [arg] <file_name>** – поиск файла по имени или иным параметрам. Аргументы поиска:
**-name** «*шаблон*» – по именам файлов,
**-inum <inode>** – по номеру индексного дескриптора,
**-mtime** «*число*» – по числу дней до последнего изменения файла,
**-type** «*тип_файла*» – по типу файлов (обычные – f, каталоги – d, ссылки – l, сокеты – s и др.),
**-perm** «*режим*» – по указанному режиму доступа и т. д.

### Общие операции над обычными файлами, каталогами и ссылками

**mkdir -arg dir** – создание нового каталога. Атрибут **–m** mode задает права доступа к каталогу. Пример: **mkdir –m 1555 /mnt/usb**.

**rm -arg [file_name, dir]** – удаление файлов и каталогов. Аргументы:
**–f** – безусловное удаление файла,

**-d** – удаление непустого каталога,
**-r** – рекурсивное удаление каталогов.
При обычном удалении файла система выводит запрос на удаление, который необходимо подтвердить символом «**y**» (yes) и **<Enter>**.

**rmdir <dir>** – удаление пустого каталога.

**shred [arg] <file_name>** – гарантированное удаление файла с многократным (25 раз) «затиранием» **inode** и блоков данных псевдослучайными комбинациями. Аргументы:

**-v** – показывать процесс удаления,
**-u** – без этого аргумента удаление не происходит,
**-n** раз – число повторов.

**mv [arg] file1 file2** – изменение имени (переименование) файла.

**mv [arg] <file_name> <dir>** – перемещение указанного файла в другой каталог.

**md5sum /sbin/* >> /home/md_sbin** – вычисление контрольных сумм файлов каталога и запись их в файл в целях контроля целостности.

**md5sum -c /home/md_sbin** – повторное вычисление контрольных сумм и сравнение с прежними результатами.

**ln <file_name> <link>** – создание жесткой ссылки на файл.

**ln -s <file_name> <link>** – создание символической ссылки на файл.

**chattr +(-)[arg] <file_name>** – установка дополнительных атрибутов файла. Аргументы:

**+i** – блокирование любых изменений файла,
**+a** – запрет любых операций, кроме добавления данных,
**+A** – отключение режима автоматического обновления временной отметки последнего доступа.

Знак "**+**" означает присвоение атрибута, знак "**-**" – его удаление.

**ls [arg] <dir>** – вывод списка файлов в директории. Аргументы:

**-l** – подробная информация,
**-a** – все файлы и подкаталоги,
**-i** – вывод номеров **inode**.

**ls -li <file_name>** – получение подробной информации о конкретном файле и его индексном дескрипторе.

**ls /dev/hd*** – получение информации о логических и физических IDE-дисках.

**stat <file_name>** – получение информации о метаданных конкретного файла.

**lsattr <file_name> [*dir*]** – вывод информации о дополнительных атрибутах файла (-ов).

**file <file_name>** – получение информации о типе файла. Информация о разпознаваемых системой типах файлов хранится в **/usr/share/magic**.

**fdisk –lu <device>** – вывод информации о логических разделах на физическом диске.

**fdisk –lu** – вывод информации о логических разделах всех подключенных устройств дисковой памяти.

### Непосредственная работа с машинными носителями

**cat /dev/fd0 > /home/floppy** – копирование всей дискеты в файл.

**cat /home/file1 > /dev/fd0** – копирование файла **file1** на дискету, начиная с ее первого сектора.

**cat /home/file2 >> /dev/fd0** – копирование файла **file2** на дискету, начиная с первого свободного сектора.

### Изменение прав доступа

**chmod mode <file_name>** – изменение прав доступа к файлу.

**Вариант 1**: **chmod wXp <file_name>** ,

где

- вместо **w** подставляется: **u** (user) – пользователь, **g** (group) – группа пользователя, **o** (other) – остальные пользователи, **a** (all) – все,
- вместо **X** подставляется: **+** – предоставление права, **–** – лишение права, **=** – установление указанного права вместо имеющегося,
- вместо **p** подставляется символ, обозначающий соответствующее право: **r** (чтение), **w** (запись), **x** (выполнение), **s** (бит SUID), **t** (sticky bit).

*Пример:* **chmod o -wx /home/user1/file1** – лишение остальных пользователей прав на запись и исполнение указанного файла.

**Вариант 2**: **chmod XXXX <file_name>,** где **X** – восьмеричное число (порядок записи слева направо): дополнительные права, права пользователя, права группы, права остальных.

**umask XXX** (user mask) – изменение режима доступа по умолчанию для вновь создаваемых файлов. **Umask** без аргументов – вызов текущего значения маски.

**chown [arg] <user> <file_name>** – передача прав на файл другому владельцу.

### Работа с процессами

**ps [arg]** – (process status) вывод информации о существующих процессах. Аргументы:

**-a** или **-e** – отобразить все процессы,
**-f** – полный листинг,
**-l** – расширенный листинг.

**top** – вывод с обновлением через определенное время.

**top –b –n1 > outfile** – сохранение однократной информации о наиболее активных процессах в файле с текстовым форматом.

**kill sign PID** – направление одного из сигналов процессу с указанным **PID** (**sign** = 15 – обычный сигнал о завершении, 9 – сигнал планировщику задач для принудительного завершения процесса).

**kill -9 0** – остановка всех активных процессов пользователя.

**skill –STOP ttyN** – блокировка физического или виртуального терминала.

**skill –CONT ttyN** – разблокировка физического или виртуального терминала.

### Работа с учетными записями пользователей

**groupadd –g GID <group_name>** – создание новой группы пользователей.

**groupdel <group_name>** – удаление зарегистрированной группы.

**su** (substitute user) – подстановка пользователя. Если объект не указан – попытка приобрести полномочия суперпользователя (администратора). Ввод команды сопровождается запросом пароля (за исключением случая, когда администратор хочет стать обычным пользователем).

**su -<user_name>** – смена пользователя со сменой пользовательского окружения.

**su <user_name>** – смена пользователя без смены окружения.

**passwd** – смена пароля текущего пользователя (будет предложено ввести новый пароль).

**passwd <user_name>** – назначение или смена пароля пользователю от имени администратора.

**chage –l <user_name>** – (change aging) получение или изменение

информации об устаревании пароля названного пользователя. Пользователи могут запрашивать только сведения об устаревании собственного пароля.

**adduser** – добавить учетную запись пользователя (ввод данных производится в интерактивном режиме).

**useradd -u UID -g GID –G <group_name> -d <dir_home> -m -e <date_del_user> <user_name>** – создание новой учетной записи пользователя с именем **<user_name>.** Аргументы:
**-u** – числовой идентификатор пользователя **UID**,
**-g** – числовой идентификатор группы **GID**,
**-G** – имена дополнительных групп (через запятую и без пробелов), в которые включается пользователь,
**-d** – путь к домашнему каталогу пользователя,
**-m** – указание создать домашний каталог пользователя по умолчанию,
**-e** – дата удаления учетной записи пользователя.

**usermod –G alfa,beta petrov** – добавление указанного пользователя к ранее созданным группам.

**userdel -r <user_name>** – удаление учетной записи пользователя (до удаления пользователь должен завершить сеанс работы). При наличии аргумента **-r** учетная запись удаляется вместе с каталогами и файлами пользователя.

**w <user_name>** – вывод данных о текущих процессах и терминалах каждого пользователя. При указании имени пользователя – сведения только о нем.

**who <user_name>** – вывод данных о пользователях, зарегистрированных в системе.

**id –[arg] <user_name>** – (identifier) вывод имен и числовых идентификаторов указанного пользователя и его групп. Если пользователь не указан – вывод информации о пользователе, зарегистрированном в данной консоли.

### Клавиши быстрого вызова и общие команды

**Alt+Fn** – переключение на другой текстовый терминал.

**Ctrl+Alt+Fn** – переключение на текстовый терминал из графического режима.

**Ctrl+D**, или **logout**, или **exit** – завершение работы с терминалом.

**Tab** (после набора одного или нескольких первых символов команды) – вывод списка команд, начинающихся данными символами.

**history** – перечень ранее введенных команд (по умолчанию список от 500 до 1000 команд).

↑ – возврат к предыдущей команде.

**man <name_command>** – вызов электронной справки по нужной команде. Выход из электронного справочника по **q**.

**info <name_command>** – еще один вариант вызова электронной справки по нужной команде. Выход из электронного справочника по **q**.

**<command_name> --help** – вызов краткой справки о команде.

**shutdown -h t** – полная остановка системы через **t** минут.

**shutdown -h 0**, или **poweroff**, или **init 0**, или **halt** – немедленная полная остановка (завершение работы системы все равно потребует несколько минут).

**shutdown -r** , или **reboot**, или **init 6**, или **Ctrl+Alt+Del** – перезагрузка операционной системы.

### Отображение информации о дисковом пространстве и файловых системах

**fdisk -lu <dev>**

**<dev>** – имя специального файла устройства, которое идентифицирует физический диск, например **/dev/hda**.

**fdisk -lu** – вывод информации обо всех устройствах памяти, подключенных к IDE, SCSI, SATA и USB интерфейсам данного компьютера.

**fdisk –lu /dev/hda** – вывод информации о разделах жесткого магнитного диска с IDE-интерфейсом, подключенного в качестве ведущего к первому интерфейсу.

**extview –i <inode_number> <dev>** – вывод информации, содержащейся в указанном **inode**.

**extview –b <block_number> <dev> | more** – вывод поэкранного дампа логического блока с указанным номером. Обычно логические блоки нумеруются шестнадцатеричными числами, в этом случае номер блока указывать в виде **0x12345678**.

### Монтирование и размонтирование файловых систем

**mount -t type -o option <dev> <dir>**

**type** – тип монтируемой системы (**ext2, ext3, msdos, vfat, ntfs, iso9660** и т. д.). Тип **auto** – предоставление системе возможно-

сти автоматически определить монтируемую файловую систему.

**option** – дополнительные опции монтирования:

**ro** или **rw** – файловая система монтируется только для чтения либо для чтения и записи,
**iocharset=koi8-r** – кодировка для отображения символов кириллицы (просмотр кодировки по команде **locale**),
**exec/noexec** – разрешить или запретить запуск исполняемых файлов, расположенных в примонтированной файловой системе.
**suid/nosuid** – принимать во внимание или игнорировать дополнительные атрибуты **SUID/SGID**, позволяющие запуск исполняемых файлов из данной файловой системы с правами владельца файла или его группы.

**<dev>** – имя специального файла устройства, которое идентифицирует логический диск с монтируемой файловой системой, например **/dev/hda2**.

**<dir>** – точка монтирования (каталог), который к моменту монтирования уже должен существовать.

**umount <dev>** или **umount <dir>** – размонтирование файловой системы. Вместо **<dev>** или **<dir>** следует указать конкретные значения (см. команду **mount**). К моменту монтирования все каталоги и файлы смонтированной файловой системы должны быть закрыты.

**umount /mnt/cdrom** – размонтирование оптического диска.

### Копирование данных

**cp [arg] file1 file2** – создание копии файла с другим именем.

**cp [arg] file1 <dir>** – копирование файла с прежним именем в другой каталог.
**cp [arg] <dir1> <dir2>** – копирование файлов из **<dir1>** в **<dir2>**.

Аргументы:

**-a** – сохранение атрибутов файла,
**-p** – сохранение режима доступа к файлу, его принадлежности и временных отметок (без этого параметра файл переходит в собственность копирующего).

**tar –cf backup.tar /home /etc** – копирование (группировка) в «ленточный» резервный файл содержимого одного или нескольких каталогов (каталоги отделяются в командной строке пробелами).

**tar –xf backup.tar <dir>** – распаковка данных из «ленточного» файла.

**dd if=<источник> of=<приемник> bs=<размер_блока> seek= <чис-**
**ло_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.

**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. 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 копировании 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.

***Варианты использования утилиты dd***

**dd if=/dev/fd0 of=/home/floppy conv=noerror,sync** – создание файл-образа гибкого магнитного диска.

**echo 1234567890 | dd of=/dev/fd0 bs=1 seek=10 count=10** – запись десяти цифр на гибкий магнитный диск со смещением 10 байтов относительно его начала.

**dd if=/dev/hda bs=512 count=1|xxd** – вывод на экран для просмотра содержимого первого сектора главной загрузочной записи жесткого магнитного диска. Утилита **xxd** позволяет просмотреть данные в шестнадцатеричном виде и ASCII кодировке.

**dd if=/dev/hda6 bs=4096 skip=1 count=1|dd bs=32 skip=5 count=1|xxd** – вывод на экран для просмотра дампа описателя 6-й группы блоков файловой системы **ext2fs.**

**dd if=/dev/hda6 bs=4096 skip=4 count=1|dd bs=128 skip=10 count=1|xxd** – вывод на экран для просмотра дампа 11-го индексного дескриптора файловой системы **ext2fs.**

**dd if=/dev/hda7 of=/home/hd7 bs=4k conv=noerror,fsync** – создание в домашнем каталоге целевого компьютера файл-образа 7-го раздела жесткого магнитного диска с IDE-интерфейсом.

**dd if=/dev/sda1 bs=4k|nc –w 3 192.168.1.20 2222** – сетевое копирование на целевой компьютер 1-го раздела первого SATA-диска. Указан IP-адрес и номер порта транспортного уровня целевого компьютера.

**nc –l –p 2222 > /home/sd1** – командная строка, обеспечивающая сетевое копирование на целевом компьютере.

**cdrecord –v -eject -sao dev=1000,0,0 speed=6 image.iso**

Команда для записи файла на компакт–диск (CD). Аргументы:
**speed** – скорость записи для привода,
**dev=1000,0,0** – номер устройства для чтения и записи CD-RW,
**eject** – открывание лотка привода CD после окончания записи,
**image.iso** – имя файл–образа в формате **iso9660** либо другого файла.
Номер устройства определяется с помощью команды

**cdrecord --scanbus**

Очистка CD-RW перед записью

**cdrecord –v dev=1000,0,0 blank=fast**

Подготовка файл-образа будущего диска

**mkisofs –R –l -o /tmp/disk.iso <dir>**

### Синтаксис некоторых сетевых команд

1. Команда **ping** служит для зондирования эхо-запросами сетевых узлов для установления их наличия и доступности.

**ping <параметры> <адрес_хоста> <номер_порта>**

Параметры:

**-l count** или **-c count** – отправка указанного числа пакетов. По умолчанию (в зависимости от ОС) посылается либо один пакет, либо бесконечная серия пакетов с интервалом в одну секунду. Прерывается нажатием **Ctrl – C**,
**-s count_byte** – общее количество байтов в **icmp**-пакете с эхо-запросом (длина заголовка **icmp**-пакета – 8 байт),
**-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: **ping –c 3 –i 5 192.168.1.2 21** – направление трех стандартных пакетов с пятисекундным интервалом в адрес FTP-сервера 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.

2. Команда **arp** служит для просмотра или изменения содержания ARP-таблицы. Аргументы:

**-a <hostname>** – отображение ARP-записей для IP-адреса указанного узла или всех известных узлов, если **hostname** не задается,
**-d** – удаление всех ARP-записей,
**-s <hostname> hwaddr** – принудительное задание ARP-записи в таблицу.

3. Протокол **telnet** (сетевой протокол телекоммуникации) служит для удаленного доступа на уровне команд между узлами сети.

**telnet <адрес_хоста> <адрес_порта>**

**<адрес_хоста>** – доменное имя или IP – адрес целевого компьютера.
По умолчанию происходит TCP-соединение с портом 23 на удаленной машине. Чтобы соединиться с другим портом, надо указать его номер в конце командной строки:

**telnet 192.168.0.10 31337**

После ввода команды и успешного соединения выводится приглашение для ввода идентификатора и пароля. Система обычно не допускает удаленной регистрации с правами суперпользователя. Поэтому на удаленном компьютере нужно иметь учетную запись обычного пользователя и регистрироваться с его правами. После регистрации и получения соответствующего уведомления можно управлять удаленным компьютером в режиме командной строки, как своим собственным. В ходе сеанса с помощью команды **su**

можно повысить свои права (при знании пароля). Сеанс завершается по команде **exit**.

4. Команда **ifconfig** служит для отображения или установки параметров сетевого интерфейса (адаптера или модема).

**ifconfig <интерфейс> <адрес> <параметры>**

**<интерфейс>**
**eth0** … **ethN** – установленные сетевые адаптеры Ethernet,
**lo** (Local Loopback) – «кольцевой интерфейс» или «обратная петля», для нее обычно устанавливается IP-адрес 127.0.0.1,
**ppp0 ... pppN** - модемы,
**vmnet0 ... vmnetN** – виртуальные сетевые интерфейсы,
**tr0 ... trN** – сетевые интерфейсы Token Ring.

**<адрес>** – IP или MAC адрес, присваиваемый узлу (IP-адрес присваивается на сеанс работы).

**<параметры>**
**-a** – вывод информации обо всех сетевых интерфейсах компьютера, включая виртуальные,
**(-) promisc** – выключение или включение режима «беспорядочного» перехвата всех пакетов в физическом канале,
**(-) arp** – выключение или включение ответов на **arp**-запросы с других узлов сети,
**mtu N** – установка параметра Maximum Transfer Unit (MTU),
**down** – отключение IP-трафика через интерфейс,
**up** – включение интерфейса (аргумент обязателен, когда производится перезапуск интерфейса, который был временно выключен опцией **down**).

***Варианты использования утилиты* ifconfig**

**ifconfig -a** – вывод информации обо всех сетевых интерфейсах данного компьютера,

**ifconfig eth0** – вывод информации о первом (или единственном) Ethernet-адаптере (его номер не обязательно равен нулю),

**ifconfig eth0 192.168.0.3 netmask 255.255.255.0** – динамическое (до перезагрузки компьютера) присвоение интерфейсу сетевого адреса и маски сети,

**ifconfig eth0:1 192.168.0.3** – динамическое присвоение интерфейсу дополнительного IP-адреса,

**ifconfig eth0 –arp** – отключение ответов на ARP-запросы других узлов сети,

**ifconfig eth0 promisc** – перевод Ethernet-адаптера в режим перехвата всех пакетов,

**ifconfig eth0 down** – отключение Ethernet-адаптера,

**ifconfig eth0 hw ether 01:02:03:04:05:06** – динамическая установка нового MAC-адреса (производится после отключения интерфейса с аргументом **down**),

**ifconfig eth0 up** – активизация Ethernet-адаптера после смены MAC-адреса.

5. Команда **netstat** предназначена для получения разнообразной информации о состоянии сети. Аргументы:

**-a** – вывод полной информации,
**-i** – вывод информации о сетевых интерфейсах. В последней колонке таблицы кратко отображается информация о состоянии интерфейса (B – установлен широковещательный адрес, L – интерфейс задает устройство loopback, M – интерфейс работает в режиме захвата всех пакетов, O – ARP выключен, R – интерфейс работает),
**-t** – вывод информации о **tcp**-сеансах,
**-u** – вывод информации о **udp**-сеансах,
**-n** – отображение только IP-адресов вместо имен хостов.

6. Команда **nmap** служит для разведки сети.

**nmap <тип_сканирования> <параметры> <список узлов/сетей>**

**<тип_сканирования>**
**-sT** – TCP-сканирование с установлением соединения. Используется по умолчанию и может запускаться обычным пользователем,
**-sS** – TCP-сканирование с помощью сообщений SYN (считается наилучшим из методов TCP-сканирования),
**-sU** – UDP-сканирование,
**-sP** – ping-сканирование,
**-sF** – FIN-сканирование,
**-sN** – нуль-сканирование.

**<параметры>**
**-O** – режим изучения откликов для определения типа удаленной операционной системы,
**-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).

7. Команда **tcpdump** – универсальная утилита для прослушивания моноканала. Автоматически переводит сетевой адаптер в режим захвата всех пакетов, но отображает или сохраняет в файл только отфильтрованные пакеты.

**tcpdump <параметры> <параметры_фильтрации>**

**<параметры>** (приводятся наиболее важные параметры):
**-v, -vv, -vvv** – вывод информации с различной степенью подробности,
**-c <число_пакетов>** – завершение работы после захвата указанного числа пакетов,
**-s <длина_пакета>** – размер перехватываемого пакета в байтах. Обычные размеры заголовков пакетов: Ethernet – 14 байтов, IP – 20 байтов, TCP – 20 байтов. По умолчанию перехватываются первые 68 байтов из пакета,
**-w <имя_файла>** – запись перехваченной информации в файл специального формата,
**-r <имя_файла>** – чтение и вывод на экран содержимого ранее записанного файла,
**-i <интерфейс>** – тип сетевого адаптера или модема, с помощью которого производится перехват пакетов,
**-n** – вывод номеров хостов вместо их символьных имен,
**-e** – вывод MAC-адресов передатчика и приемника,
**-x (-xx)** – отображение содержимого IP-пакета в шестнадцатеричном виде,
**-X** – отображение содержимого пакета в шестнадцатеричных и ASCII– кодах.

**<параметры фильтрации>** используют несколько ключевых слов:

- **Протокол (proto)** – указывает, какие именно пакеты подлежат перехвату. Используются ключевые слова: **ether, ip, arp, rarp, tcp, udp**;
- **Направление** – указывает источники и получателей сообщений: **src** (source – источник), **dst** (destination – получатель) или их комбинацию: **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.

Ключевые слова и математические выражения могут объединяться с использованием логических условий: **not**, **and** или **or**.

***Варианты использования утилиты*** **tcpdump**

**tcpdump –i eth0 –c 25 –vv -w /home/net_dump1 ip host 192.168.0.34** – перехват Ethernet-адаптером 25 сетевых пакетов (**ip**) стандартного размера, адресованных узлу **192.168.0.34** или исходящих от него, с детальным описанием и записью информации в указанный файл,

**tcpdump –i eth0 –c 100 –s 1514 ether src 00:01:02:03:04:05** – перехват 100 пакетов канального уровня длиной 1514 байтов, передаваемых сетевым адаптером Ethernet (**ether**) с указанным MAC-адресом (ключевое слово **host** после **src** или **dst** можно не указывать),

**tcpdump –i eth0 –c 50 –v –w /home/tcp_dump2 src 192.168.0.1 and dst 192.168.0.12** – перехват 50 пакетов любого типа, направленных от узла **192.168.0.1** в адрес узла **192.168.0.12**, с записью информации в файл,

**tcpdump –r /home/tcp_dump2|more** – чтение и вывод ранее записанного файла с сетевым дампом,

**tcpdump –i eth0 (tcp or udp) port 53** – перехват сетевым адаптером Ethernet всех **tcp** и **udp** пакетов, исходящих или направляемых в 53-й порт,

**tcpdump –i eth0 (ip or arp or rarp) src net 128.3** – перехват всех пакетов формата **ip, arp** и **rarp**, исходящих из сети с указанной маской,

**tcpdump –i eth0 not src net 192.168 and (ip[19] = 0xff or ip[19] = 0x00)** – перехват всего широковещательного трафика от любого внешнего источника, кроме собственной подсети,

**tcpdump –i eth0 udp[2:2] >= 33000 and udp[2:2] < 34000** – фильтр для отслеживания попыток установить соединение с **udp**-портами в диапазоне 33000 – 33999.

## Контроль и конфигурация беспроводной ЛВС

**Команда iwconfig**

**iwconfig <dev> mode master** – перевести адаптер в режим работы точки доступа,

**iwconfig <dev> mode managed** – обычный режим при работе с точкой доступа,

**iwconfig <dev> mode ad-hoc** – установка одноранговой сети точка-точка без выделенной точки доступа,
**iwconfig <dev> mode monitor** – перевод адаптера в режим перехвата всех пакетов на данном канале (поддерживается не всеми картами),
**iwconfig <dev> essid any** – отключить проверку ESSID и подключаться к любой доступной сети,
**iwconfig <dev> essid "abcd"** – задать ESSID сети,
**iwconfig <dev> key 1234567890123456789012345678** – установить 128 bit WEP или 26-символьный hex-ключ,
**iwconfig <dev> key 1234567890** – установить 64 bit WEP или 10-символьный hex-ключ,
**iwconfig <dev> key off** – отключить WEP ключ,
**iwconfig <dev> key open** – установить открытый режим, не требующий авторизации при подключении и шифрования трафика,
**iwconfig <dev> channel N** – установить рабочий канал в диапазоне от 1 до 11,
**iwconfig <dev> channel auto** – автоматический выбор канала,
**iwconfig <dev> freq 2.422G** – канал также может быть задан в виде указания определенной частоты,
**iwconfig <dev> ap 11:11:11:11:11:11** – установка MAC-адреса точки доступа,
**iwconfig <dev> rate 11M** – указание скорости обмена в [Мбит/сек],
**iwconfig <dev> rate auto** – автоматический выбор скорости обмена.

**Команда iwlist**

**iwlist** показывает некоторые параметры беспроводной карты, которые недоступны **iwconfig**:
**iwlist <dev> scan** – получить список доступных точек доступа и компьютеров, настроенных в режим точка–точка, а также такие параметры как имя сети, качество сигнала, частота, режим работы и другое,
**iwlist <dev> channel** – получить список доступных каналов и частот доступных устройств,
**iwlist <dev> rate** – список скоростей доступных устройств,
**iwlist <dev> key** – список и размеры ключей для подключения,
**iwlist <dev> power** – показать режимы управления питанием адаптера,
**iwlist <dev> txpower** – показать диапазоны доступной мощности сигнала адаптера,
**iwlist <dev> ap** – получить список точек доступа и уровень их сигнала.

**Команды `Madwifi-ng`**

**`MadWiFi`** поддерживает виртуальные точки доступа (**`VAPS`**), что помогает эмулировать несколько беспроводных устройств на основе одной физической карты (основная карта = **`wifi0`**). Поддерживается картами, созданными на основе чипсетов **`Atheros`**. Для краткости устройство называется **`athx`** – хотя в конкретной системе она может называться **`ath0`** или **`ath1`**:

**`wlanconfig athx destroy`** – убрать виртуальную точку доступа **`athx`**,

**`wlanconfig athx create wlandev wifi0 wlanmode sta`** – создать работающую в режиме **`managed`** виртуальную точку доступа **`athx`**,

**`wlanconfig athx create wlandev wifi0 wlanmode ap`** – создать виртуальную точку доступа **`athx`**,

**`wlanconfig athx create wlandev wifi0 wlanmode adhoc`** – создать работающую в режиме **`Ad-Hoc`** виртуальную точку доступа **`athx`**,

**`wlanconfig athx create wlandev wifi0 wlanmode monitor`** – создать работающую в режиме **`Monitor`** виртуальную точку доступа **`athx`**.

# ПРИЛОЖЕНИЕ 2

## АРХИТЕКТУРНЫЕ ОСОБЕННОСТИ ФАЙЛОВОЙ СИСТЕМЫ EXT4FS

С 2006 года в операционных системах Linux начато использование новой файловой системы **ext4fs**, сначала в виде дополнения к существующей ФС **ext3fs**, а затем, с 2008 года, в виде стабильной, законченной файловой системы. Однако многие из запланированных преимуществ ФС еще не реализованы ее разработчиками, и эта система продолжает развиваться и дорабатываться. Новая ФС содержит много особенностей и сильно отличается от предшествующей версии.

Несмотря на некоторую незавершенность файловой системы, она уже устанавливается по умолчанию во многих дистрибутивах Linux и стремительно набирает популярность. 15 января 2010 года компания Google объявила о переводе своих серверов с файловой системы **ext2** на файловую систему **ext4**.

Поскольку файловая система разработана недавно и все еще находится в процессе развития, документации на русском языке по ней практически нет. Отсутствуют и утилиты для исследования файловой системы и ее структур. Формат данных, хранимых на диске, в **ext4** несколько изменился, поэтому программы, предназначенные для исследования и восстановления данных в системах **ext2** и **ext3**, с новой ФС работают некорректно.

### Использование экстентов

В файловой системе адресация данных выполнялась традиционным образом, поблочно. Такой способ адресации становится менее эффективным с ростом размера файлов. Экстенты позволяют адресовать большое количество (до 128 Мб) последовательно идущих блоков одним дескриптором. До 4 указателей на экстенты может размещаться непосредственно в индексном дескрипторе, что достаточно для файлов маленького и среднего размера [15]. Экстенты используются многими файловыми системами, в том числе: **HFS Plus** (для Mac OS), **NTFS** (для Windows), **UDF**, **XFS**, **JFS**, **Reiser4**, **Btrfs** и **HPFS**.

### 48-битные номера блоков

**Ext3** является 32-битной файловой системой. Это значит, что ее максимальный размер при размере блока в 4 Кб составляет $2^{(32+12)} = 17$ Тб. **Ext4** при размере блока 4 Кб позволяет адресовать до одного экзабайта: $2^{48} \cdot 4KB = 2^{50} \cdot 1KB = 2^{60} B = 1EB$. Многие популярные файловые системы, например XFS, являются 64-битными. 48-битная адресация блоков в **ext4** была выбрана вместо полной 64-битной адресации по причине того, что предел для файловой системы в 1 Эб еще много лет будет достаточным.

При существующих скоростях одна полная проверка файловой системы в 1 Эб утилитой **e2fsck** займет 119 лет, что в 65 536 раз меньше, чем потребуется для системы с 64-битной адресацией [16]. Но возможности для поддержки 64-битной адресации создаются уже сейчас: некоторые поля системы зарезервированы под 64-битную адресацию.

**Новые алгоритмы выделения блоков и индексных дескрипторов**

*Выделение блоков группами* (multiblock allocation). Файловая система хранит не только информацию о местоположении свободных блоков, но и количество свободных блоков, идущих друг за другом. При выделении места файловая система находит такой фрагмент, в который данные могут быть записаны без фрагментации. Это снижает уровень фрагментации файловой системы в целом.

**Ext3** одинаково выделяла блоки как для маленьких, так и для больших файлов, что увеличивало фрагментированность ФС. **Ext4** в зависимости от размера файла применяет разные алгоритмы выделения дискового пространства. Маленькие файлы из одной директории ФС старается держать рядом, чтобы минимизировать время и количество обращений к диску. Большие файлы размещаются отдельно, в непрерывной области памяти [18].

*Отложенное выделение блоков* (delayed allocation). Выделение блоков для хранения данных файла происходит непосредственно перед физической записью на диск (например, при вызове **sync**), а не при вызове **write**. В результате операции выделения блоков можно делать не по одной, а группами, что в свою очередь минимизирует фрагментацию, снижает нагрузку на процессор и ускоряет процесс выделения блоков. Отложенное выделение блоков также предотвращает ненужное резервирование блоков для короткоживущих файлов. С другой стороны, это увеличивает риск потери данных в случае внезапного пропадания питания.

*Предварительное выделение* (persistent preallocation). Сейчас приложения для того, чтобы гарантированно занять какое-то место, заполняют нулями. В **ext4** появилась возможность зарезервировать множество блоков для записи и не тратить на инициализацию лишнее время. Если приложение попробует прочитать данные, оно получит сообщение о том, что они не проинициализированы. Таким образом, несанкционированно прочитать удалённые данные не получится.

*Резервирование индексных дескрипторов при создании каталога* (directory inodes reservation). При создании каталога резервируется несколько индексных дескрипторов. Впоследствии при создании файлов в этом каталоге сначала используются зарезервированные индексные дескрипторы, и если таких не осталось, выполняется обычная процедура выделения **inode** [17].

**Формат индексного дескриптора**

*Размер индексного дескриптора*. Размер индексного дескриптора (по умолчанию) увеличен с 128 до 256 байтов. Это дает возможность реализовать некоторые преимущества, например хранить расширенные атрибуты в самом дескрипторе, использовать более точные временные метки и т. д.

*Версия индексного дескриптора.* В индексном дескрипторе появился номер, который увеличивается при каждом изменении индексного дескриптора файла. Это может использоваться, например, для того чтобы узнавать, изменился ли файл.

*Хранение расширенных атрибутов в индексном дескрипторе* (EA in inode). Хранение расширенных атрибутов, таких как ACL, атрибутов SELinux и прочих, позволяет повысить производительность. Атрибуты, для которых недостаточно места в индексном дескрипторе, хранятся в отдельном блоке размером 4 Кб. Предполагается снять это ограничение в будущем [17].

**Временные отметки с наносекундной точностью**

Временные отметки в **ext3** имеют секундную точность. Однако некоторым приложениям, особенно клиент-серверным, требуется более высокая точность. **Ext4** предоставляет такую возможность, увеличивая точность времен, хранимых в индексном дескрипторе, суперблоке и журнале. Достигается это путем увеличения разрядности временной отметки с 4 до 8 байтов. Диапазон хранящихся времён тоже расширен: если раньше верхней границей хранимого времени было 18 января 2038 года, то теперь – это 25 апреля 2514 года [17].

**Общая организация системы**

Основные принципы организации хранения данных в **ext4** соответствуют организации файловых систем **ext2** и **ext3**. Весь жесткий диск разбивается на блоки данных, которые являются минимально адресуемыми единицами информации.

Здесь и проявляется основное отличие **ext4** от предшественницы – теперь номера блоков могут быть 48-битными, если есть поддержка со стороны ядра.

Блоки группируются на группы блоков, в каждой группе есть битовая карта блоков, битовая карта индексных дескрипторов, таблица индексных дескрипторов и собственно блоки данных. Битовая карта блоков определяет свободные и занятые блоки в группе установкой или сбросом соответствующего бита. Битовая карта индексных дескрипторов аналогичным образом определяет свободные или занятые дескрипторы. Статическая таблица индексных дескрипторов хранит информацию о файлах, которым эти дескрипторы соответствуют.

Начальной точкой файловой системы является суперблок. Он находится по смещению 1024 байта от начала диска (в самом начале диска размещается загрузчик) и занимает 1 Кб. Суперблок определяет многие важные параметры файловой системы: раз-

мер блока, число блоков, параметры групп блоков, параметры системы и т.д.

В следующем блоке располагаются дескрипторы групп блоков. Они указывают на блоки, в которых находятся битовые карты блоков и индексных дескрипторов, начало таблицы индексных дескрипторов и другую информацию о группе [17].

Традиционная организация файловой системы предполагает, что группы блоков идут на диске одна за другой и каждая начинается с битовых карт и таблиц индексных дескрипторов. Такая разбивка файловой системы представлена на рис. П.2.1.

В **ext4** вводится опция так называемых гибких групп блоков. Гибкая группа блоков состоит из нескольких групп блоков. В пределах гибкой группы битовые карты и таблицы индексных дескрипторов хранятся вместе, и лишь затем идут блоки данных [18]. Это позволяет уменьшить время на чтение информации о свободных блоках и дескрипторах, а также ускорить проверку диска с помощью **e2fsck**.

Разбивка диска при использовании опции «гибкая группа блоков» представлена на рис. П.2.2.

Гибкие группы блоков можно наблюдать с помощью редактора **debugfs**. Команда **debugfs -R stats <device>** выводит в том числе и группы блоков. На рис. П.2.3 видно, что в каждой группе блоков битовые карты имеют не смежные номера блоков, зато в соседних группах номера соответствующих битовых карт возрастают последовательно.

**Ext4** не резервирует блоки для журнала после описателей групп блоков, как это делалось в **ext3**. Вместо этого журнал имеет определенный индексный дескриптор и хранится в выделенных для него блоках, которые в принципе могут находиться в любой области диска. Перед началом битовых карт групп блоков пространство зарезервировано под описатели групп. В **ext4** существует понятие «системная зона» – это блоки, выделенные под хранение метаданных, которые нельзя использовать индексным дескрипторам [18]. К метаданным относятся суперблок, в том числе и его копии, описатели групп блоков и журнал.

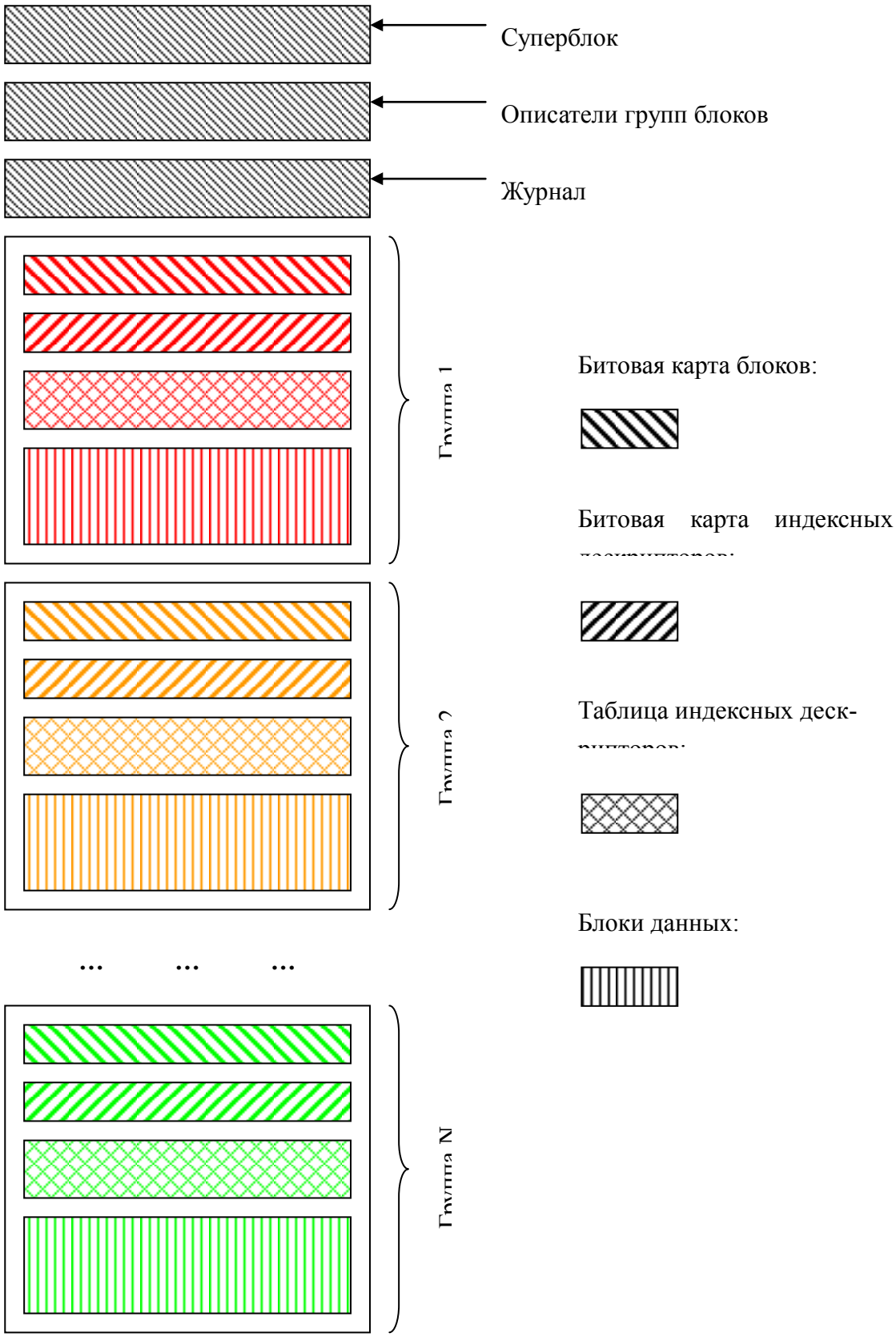


Рис. П.2.1. Группы блоков в файловой системе **`ext3fs`**

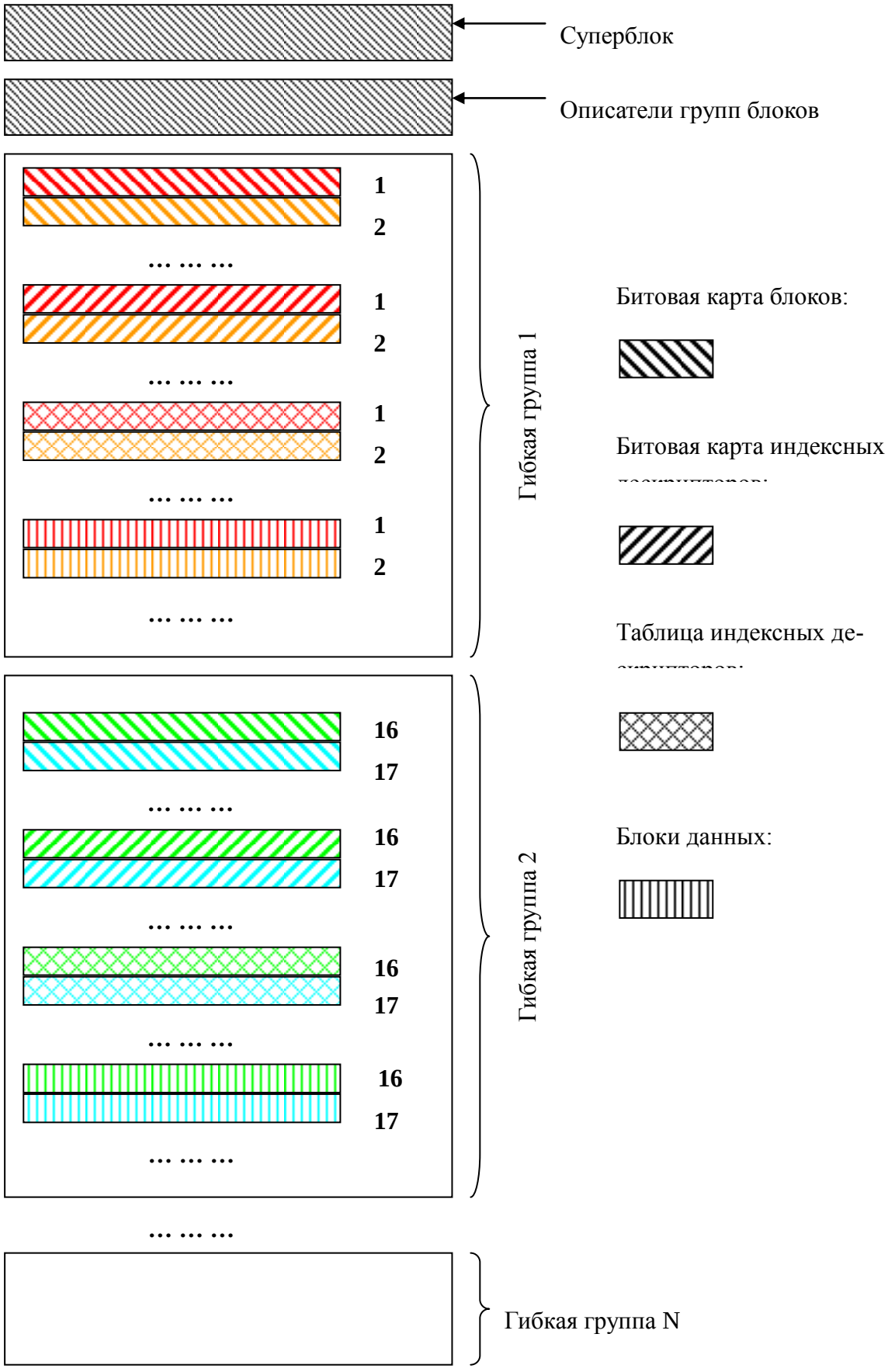
Суперблок
Описатели групп блоков
1
2
… … …
1
2
… … …
1
2
… … …
1
2
… … …
Гибкая группа 1
Битовая карта блоков:
Битовая карта индексных
Таблица индексных де-
16
17
… … …
16
17
… … …
16
17
… … …
16
17
… … …
Гибкая группа 2
Блоки данных:
… … …
Гибкая группа N

Рис. П.2.2. Гибкие группы блоков в файловой системе **ext4fs**

```
Group  0: block bitmap at 591, inode bitmap at 607, inode table at 623
           3436 free blocks, 29 free inodes, 4 used directories, 0 unused
inodes
           [Checksum 0x6dab]
 Group  1: block bitmap at 592, inode bitmap at 608, inode table at 1134
           4182 free blocks, 32 free inodes, 753 used directories, 0 unused
inodes
           [Checksum 0x2f06]
 Group  2: block bitmap at 593, inode bitmap at 609, inode table at 1645
           5587 free blocks, 24 free inodes, 702 used directories, 0 unused
inodes
           [Checksum 0x29d8]
 Group  3: block bitmap at 594, inode bitmap at 610, inode table at 2156
           7755 free blocks, 50 free inodes, 625 used directories, 0 unused
inodes
           [Checksum 0x5b66]
 Group  4: block bitmap at 595, inode bitmap at 611, inode table at 2667
           6894 free blocks, 186 free inodes, 1488 used directories, 0 un-
used inodes
           [Checksum 0x9d36]
 Group  5: block bitmap at 596, inode bitmap at 612, inode table at 3178
           8311 free blocks, 70 free inodes, 362 used directories, 0 unused
inodes
           [Checksum 0xf453]
 Group  6: block bitmap at 597, inode bitmap at 613, inode table at 3689
           7086 free blocks, 0 free inodes, 2167 used directories, 0 unused
inodes
           [Checksum 0x07c0]
 Group  7: block bitmap at 598, inode bitmap at 614, inode table at 4200
           2314 free blocks, 0 free inodes, 1658 used directories, 0 unused
inodes
           [Checksum 0xdc31]
 Group  8: block bitmap at 599, inode bitmap at 615, inode table at 4711
           4177 free blocks, 141 free inodes, 450 used directories, 0 un-
used inodes
           [Checksum 0x50f9]
```

Рис. П.2.3. Фрагмент гибкой группы блоков, выводимой **debugfs**

Кроме собственно числа блоков в ФС, ее размер ограничен также максимально возможным числом групп блоков. В **ext3** копии всех описателей групп блоков должны храниться в первой группе блоков. В **ext4** размер описателя группы блоков составляет 64 байта (если используется 48-битная адресация блоков). При размере группы блоков в 128 Мб всего в **ext4** может быть $2^{27}/64 = 2^{21}$ групп блоков. Это ограничивает размер файловой системы до $2^{21} \cdot 2^{27} = 2^{48}$ байтов, или 256 Тб. Для снятия этого ограничения используют метагруппы блоков. Файловые системы **ext4** разделены на метагруппы блоков [17].

Метагруппа – это кластер из групп блоков, описатель которого может храниться в одном блоке. При размере блока в 4 Кб одна метагруппа может содержать 64 группы блоков, или 8 Гб. Описатели групп блоков теперь хранятся не в первой группе блоков, а «рассыпаны» по всей файловой системе, что записано в первый блок метагруппы. Резервные копии хранятся во втором и последнем блоках метагруппы [15].

Таким образом, максимальное число групп блоков в системе может достигать $2^{32}$, что соответствует 1 Эб дискового пространства.

**Суперблок и описатели групп блоков**

Формат суперблока определен в заголовочном файле **ext4.h** исходного кода ядра ОС Linux как структура **ext4_super_block**.

Таблица П.2.1

Формат суперблока

| **Размер поля, байт** | **Смещение** | | **Назначение** |
|---|---|---|---|
| | **десятичное** | **шестнадцатеричное** | |
| 4 | 0 | 0 | Число индексных дескрипторов в ФС |
| 4 | 4 | 4h | Число блоков в ФС |
| 4 | 8 | 8h | Число блоков, зарезервированных для нужд суперпользователя |
| 4 | 12 | Ch | Число свободных блоков |
| 4 | 16 | 10h | Число свободных индексных дескрипторов |
| 4 | 20 | 14h | Номер первого блока, содержащего данные (0 или 1) |
| 4 | 24 | 18h | Индикатор размера логического блока |
| 4 | 28 | 1Ch | Индикатор размера фрагментов |
| 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 ФС |
| 4 | 48 | 30h | Время последней записи в ФС |
| 2 | 52 | 34h | Число монтирований ФС. При достижении предельного числа монтирований обнуляется |
| 2 | 54 | 36h | Предельное число монтирований ФС |
| 2 | 56 | 38h | Магическое число **`0xEF53`**, указывающее, что ФС принадлежит к **`ext*fs`** |
| 2 | 58 | 3Ah | Флаги текущего состояния ФС |
| 2 | 60 | 3Ch | Флаги процедур обработки сообщений об ошибках |
| 2 | 62 | 3Eh | Младший номер устройства |
| 4 | 64 | 40h | Время последней проверки ФС |
| 4 | 68 | 44h | Максимальный период времени между проверками |
| 4 | 72 | 48h | Указание на тип ОС, в которой создана ФС. Коды операционных систем: 0 – Linux, 1 – Hurd, 2 – Masix, 3 – FreeBSD, 4 – Lites |
| 4 | 76 | 4Ch | Версия ФС |
| 2 | 80 | 50h | UID по умолчанию для зарезервированных блоков |
| 2 | 82 | 52h | GID по умолчанию для зарезервированных блоков |
| 4 | 84 | 54h | Номер первого индексного дескриптора, не зарезервированного системой |
| 2 | 88 | 58h | Размер индексного дескриптора |
| 2 | 90 | 5Ah | Номер группы блоков, в которой находится этот суперблок |

Окончание табл. П.2.1

| | | | |
|---|---|---|---|
| 4 | 92 | 5Ch | Флаги COMPAT |
| 4 | 96 | 60h | Флаги INCOMPAT |
| 4 | 100 | 64h | Флаги ROCOMPAT |
| 16 | 104 | 68h | UUID тома |
| 16 | 120 | 78h | Имя тома |
| 64 | 136 | 88h | Каталог, в котором последний раз была смонтирована система |
| 4 | 200 | C8h | Алгоритм использования битовых карт (для сжатия) |
| 1 | 204 | CCh | Число блоков, которые надо попытаться зарезервировать. Используется, если включена COMPAT-опция DIR_PREALLOC |
| 1 | 205 | CDh | То же для директорий. Используется, если включена COMPAT-опция DIR_PREALLOC |
| 2 | 206 | CEh | Зарезервированные блоки на групповой дескриптор. |

| | | | |
|---|---|---|---|
| | | | Используется, если включена COMPAT-опция DIR_PREALLOC |
| 16 | 208 | D0h | UUID журнального суперблока |
| 4 | 224 | E0h | Номер индексного дескриптора файла журнала |
| 4 | 228 | E4h | Номер устройства файла журнала |
| 4 | 232 | E8h | Указатель на список индексных дескрипторов, которые были удалены, но использовались каким-либо приложением в момент удаления |
| 16 | 236 | ECh | Зерно для хэш-функции H-дерева |
| 1 | 252 | FCh | Версия хэш-функции по умолчанию |
| 1 | 253 | FDh | Зарезервировано |
| 2 | 254 | FEh | Размер описателя группы блоков |
| 4 | 256 | 100h | Параметры монтирования по умолчанию |
| 4 | 260 | 104h | Первая метагруппа блоков |
| 4 | 264 | 108h | Время создания ФС |
| 68 | 268 | 10Ch | Бэкап для файла журнала |
| 4 | 336 | 150h | Старшие 32 значащих бита числа блоков |
| 4 | 340 | 154h | Старшие 32 бита зарезервированных блоков |
| 4 | 344 | 158h | Старшие 32 бита числа свободных блоков |
| 2 | 348 | 15Ch | Минимальный размер фиксированных полей индексного дескриптора |
| 2 | 350 | 15Eh | Новые индексные дескрипторы должны резервировать столько байт для фиксированных полей |
| 4 | 352 | 160h | Смешанные флаги |
| 2 | 356 | 164h | RAID stride |
| 2 | 358 | 166h | Число секунд в ожидании проверки MMP |
| 8 | 360 | 168h | Блок для защиты от множественного монтирования |
| 4 | 368 | 170h | Блоки на всех дисках с данными (N*stride) |
| 1 | 372 | 174h | Размер гибкой группы блоков |
| 1 | 373 | 175h | Зарезервировано |
| 2 | 374 | 176h | Зарезервировано |
| 8 | 376 | 178h | Число килобайт, записанных на диск за время жизни системы |
| 640 | 384 | 180h | Заполнение до конца суперблока |

Вывести информацию, содержащуюся в суперблоке, можно с помощью команды **`debugfs –R stats <device>`**. Результат выполнения этой команды приведен на рис. П.2.4.

```
Filesystem volume name:   <none>
```

```
Last mounted on:          /
Filesystem UUID:          aaca0a90-3a12-41b8-9f3d-c82b1b48680c
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode dir_index
filetype needs_recovery extent flex_bg sparse_super large_file huge_file
uninit_bg dir_nlink extra_isize
Filesystem flags:         signed_directory_hash
Default mount options:    (none)
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              605024
Block count:              2415766
Reserved block count:     120788
Free blocks:              1211855
Free inodes:              378277
First block:              0
Block size:               4096
Fragment size:            4096
Reserved GDT blocks:      589
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         8176
Inode blocks per group:   511
Flex block group size:    16
Filesystem created:       Mon Nov  2 16:24:05 2009
Last mount time:          Sun Feb 21 17:46:13 2010
Last write time:          Mon Feb 15 18:17:16 2010
Mount count:              14
Maximum mount count:      29
Last checked:             Sat Feb 13 18:03:59 2010
Check interval:           15552000 (6 months)
Next check after:         Thu Aug 12 19:03:59 2010
Lifetime writes:          29 GB
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:           256
```

```
Required extra isize:     28
Desired extra isize:      28
Journal inode:            8
First orphan inode:       142509
Default directory hash:   half_md4
Directory Hash Seed:      c24ea3e3-8847-4a42-8330-498487728212
Journal backup:           inode blocks
Directories:              30357
```

Рис. П.2.4. Вывод информации о файловой системе отладчиком **debugfs**

Для обеспечения совместимости с предыдущими версиями файловой системы и нормальной работы утилиты проверки файловой системы первые поля суперблока не претерпели никаких изменений. Основы для будущих возможностей **ext4** были заранее заложены еще в исходном коде файловой системы **ext3**. Так, в суперблоке **ext3** есть зарезервированные поля для 64-битных номеров блоков, размера гибкой группы блоков, размера фиксированных полей индексного дескриптора и т. д. Совершенно новыми для **ext4** оказались только 2 поля: размер описателя группы блоков и общее число килобайтов, записанных на диск за «время жизни» файловой системы.

Ниже приведены различные флаги файловой системы, которые содержатся в суперблоке. В табл. П.2.2 приведены значения флагов состояния файловой системы. Этими флагами файловая система помечается после проверки утилитой **e2fsck** в зависимости от результатов работы утилиты. В табл. П.2.3 приведены флаги, определяющие реакцию ФС при обнаружении ошибок файловой системы.

Таблица П.2.2

Флаги текущего состояния файловой системы

| **Флаг** | **Описание** |
|---|---|
| 0x0001 | При размонтировании не произошло ошибок |
| 0x0002 | Замечены ошибки |
| 0x0004 | Режим обработки логически удаленных, но все еще открытых каким-либо процессом индексных дескрипторов |

Таблица П.2.3

Флаги процедур обработки сообщений об ошибке

| **Флаг** | **Описание** |
|---|---|
| 0x0001 | При наличии ошибок продолжить работу ФС |
| 0x0002 | Перемонтировать ФС в режим «только для чтения» |
| 0x0004 | «Паника» при ошибках: вывод сообщения об ошибке, прекращение рабо- |

| | |
|---|---|
| | ты файловой системы и уведомление пользователя о необходимости перезагрузки |

Основные опции ФС, определяющие ее свойства, содержатся в трех полях суперблока: это флаги COMPAT, INCOMPAT и ROCOMPAT. В **ext4** появились новые опции, а именно: поддержка очень больших файлов, контрольное суммирование описателей групп блоков, поддержка неограниченного числа подкаталогов, поддержка протокола MMP, введение фиксированных полей в индексном дескрипторе, поддержка экстентов, 64-битной адресации блоков и гибких групп блоков.

Описатели групп блоков занимают следующий за суперблоком блок данных. Формат описателя группы блоков определен в заголовочном файле **ext4.h** как структура **ext4_group_desc**. Эта структура представлена в табл. П.2.4. В **ext4** описатель групп блоков имеет размер 64 байта, вместо 32 байтов в **ext3**. Однако если не включена INCOMPAT-опция EXT_64BIT (64-битная поддержка), система монтируется с 32-байтными описателями [18].

В **ext4** в описатель групп блоков вводятся новые поля: флаги, контрольная сумма описателя, число неиспользованных индексных дескрипторов, а также старшие значащие биты в случае использования 64-байтного описателя. Число неиспользованных индексных дескрипторов в общем случае отличается от числа свободных дескрипторов: если в группе был удален индексный дескриптор, число свободных дескрипторов станет на 1 больше, а число неиспользованных не изменится. Флаги описателя группы блоков EXT4_BG приведены в табл. П.2.5, они позволяют ускорить проверку файловой системы: если битовая карта блоков или индексных дескрипторов помечена как неиспользованная, e2fsck пропускает ее, делая проверку только в том случае, если не совпадает контрольная сумма.

Таблица П.2.4

Формат описателя группы блоков

| **Размер поля, байт** | **Смещение, dec (hex)** | **Назначение** |
|---|---|---|
| 4 | 0 | Адрес битовой карты блоков (младшие значащие биты) |
| 4 | 4 (4h) | Адрес битовой карты индексных дескрипторов |
| 4 | 8 (8h) | Адрес блока с началом таблицы индексных дескрипторов |
| 2 | 12 (Ch) | Число свободных блоков |
| 2 | 14 (Eh) | Число свободных индексных дескрипторов |
| 2 | 16 (10h) | Число директорий |
| 2 | 18 (12h) | Флаги EXT4_BG |
| 4x2 | 20 (14h) | Зарезервировано под контрольные суммы битовых карт блоков и индексных дескрипторов |

| | | |
|---|---|---|
| 2 | 28 (1Ah) | Число неиспользованных индексных дескрипторов |
| 2 | 30 (1Ch) | Контрольная сумма нескольких полей |
| 4 | 32 (1Eh) | Старшие значащие биты адреса битовой карты блоков |
| 4 | 36 (22h) | Старшие значащие биты адреса битовой карты индексных дескрипторов |
| 4 | 40 (26h) | Старшие значащие биты адреса первого блока таблицы индексных дескрипторов |
| 2 | 44 (2Ah) | Старшие значащие биты числа свободных блоков |
| 2 | 46 (2Ch) | Старшие значащие биты числа свободных индексных дескрипторов |
| 2 | 48 (2Eh) | Старшие значащие биты числа директорий |
| 2 | 50 (30h) | Старшие значащие биты числа неиспользованных индексных дескрипторов |
| 14 | 52 (32h) | Заполнение |

Таблица П.2.5

Флаги описателя группы блоков

| **Флаг** | **Название** | **Описание** |
|---|---|---|
| 0x0001 | INODE_UNINIT | Таблица и битовая карта индексных дескрипторов не используется |
| 0x0002 | BLOCK_UNINIT | Битовая карта блоков не используется |
| 0x0004 | INODE_ZEROED | Таблица индексных дескрипторов заполнена нулями |

**Формат индексного дескриптора**

Формат индексного дескриптора описан в заголовочном файле **`ext4.h`** как структура **`ext4_inode`**, формат приведен в табл. П.2.6.

Ext4, как и предыдущая файловая система, резервирует некоторые индексные дескрипторы, например 2-й отводится для корневой директории, 8-й – для журнала транзакций. В ext4 по умолчанию используются 256-байтные индексные дескрипторы. Чтобы не переписывать заново многочисленные функции ядра и утилиту проверки файловой системы **`e2fsck`**, первые 128 байтов индексного дескриптора оставили фактически без изменений [15]. Обобщенная структура индексного дескриптора представлена на рис. П.2.5.

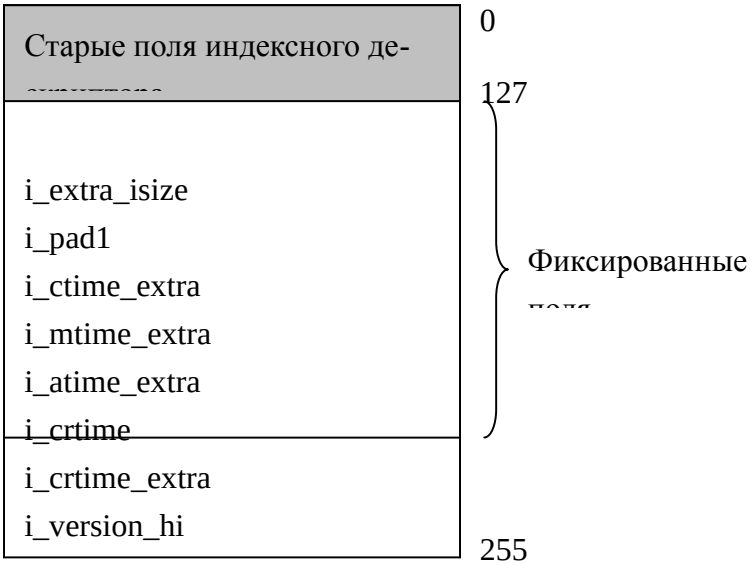


Рис. П.2.5. Индексный дескриптор в **ext4**

Фиксированные поля выделяются динамически, в зависимости от того, какая информация есть у текущего ядра об этих полях. Размер области фиксированных полей хранится в каждом индексном дескрипторе в поле *i_extra_isize*. В самом суперблоке также содержатся два поля: *s_min_extra_isize*, которое гарантирует минимальный объем области фиксированных полей в каждом индексном дескрипторе, и *s_want_extra_isize*, которое указывает на желаемый размер области фиксированных полей во вновь создаваемом индексном дескрипторе, но вовсе не гарантирует, что именно такой объем будет в нем выделен [18].

Таблица П.2.6

Формат индексного дескриптора

| **Размер, байт** | **Смещение (dec)** | **Смещение (hex)** | **Назначение** |
|---|---|---|---|
| 2 | 0 | 0 | Тип файла и права доступа к нему |
| 2 | 2 | 2h | Младшие 16 бит идентификатора пользователя |
| 4 | 4 | 4h | Размер файла в байтах |
| 4 | 8 | 8h | Время последнего доступа |
| 4 | 12 | Ch | Время изменения индексного дескриптора |
| 4 | 16 | 10h | Время последнего изменения данных файла |
| 4 | 20 | 14h | Время удаления файла |
| 2 | 24 | 18h | Младшие 16 бит идентификатора группы |
| 2 | 26 | 1Ah | Число жестких ссылок на файл |
| 4 | 28 | 1Ch | Число блоков или секторов, занимаемых файлом |
| 4 | 32 | 20h | Флаги файла |

| | | | |
|---|---|---|---|
| 4 | 36 | 24h | Зарезервировано для ОС |
| 15x4 или 5x12 | 40 | 28h | В зависимости от способа адресации 15 указателей на блоки данных или 1 заголовок экстента (12 байтов) и непосредственно 4 экстента |
| 4 | 100 | 64h | Версия файла (для NFS) |
| 4 | 104 | 68h | ACL файла |
| 4 | 108 | 6Ch | Старшие 32 бита размера файла в байтах |
| 4 | 112 | 70h | Устаревшее поле: адрес фрагмента |
| 12 | 116 | 74h | Структура, зависящая от типа ОС, определяющая номера фрагментов |
| 2 | 128 | 80h | Размер фиксированных полей inode |
| 2 | 130 | 82h | Зарезервировано |
| 4 | 132 | 84h | Дополнительные 32 бита ctime |
| 4 | 136 | 88h | Дополнительные 32 бита mtime |
| 4 | 140 | 8Ch | Дополнительные 32 бита atime |
| 4 | 144 | 90h | Время создания файла |
| 4 | 148 | 94h | Дополнительные 32 бита к ВО создания файла |
| 4 | 152 | 98h | Старшие 32 бита версии файла |

Оставшееся пространство в индексном дескрипторе может быть использовано для хранения быстрых расширенных атрибутов файла. Это позволяет не искать каждый раз внешний блок данных с расширенными атрибутами, что для некоторых приложений может весьма увеличить производительность [15].

В **`ext4`** поле, где располагалось время создания файла, заменили на поле с временной отметкой последнего изменения индексного дескриптора. Время создания файла вынесено за пределы стандартных полей.

Индексный дескриптор содержит флаги, которые называются файловыми атрибутами и выводятся утилитой **`lsattr`**. Эти флаги приведены в табл. П.2.7.

Таблица П.2.7

Флаги индексного дескриптора

| **Флаг** | **Название** | **Описание** |
|---|---|---|
| 0x00000001 | SECRM_FL | Безопасное удаление файла |
| 0x00000002 | UNRM_FL | Неудаляемый файл |
| 0x00000004 | COMPR_FL | Сжатый файл |
| 0x00000008 | SYNC_FL | Синхронное обновление |
| 0x00000010 | IMMUTABLE_FL | Неизменяемый файл |
| 0x00000020 | APPEND_FL | Любая запись может только добавляться в конец файла |

| | | |
|---|---|---|
| 0x00000040 | NODUMP_FL | Не дампировать данные файла |
| 0x00000080 | NOATIME_FL | Не обновлять время последнего доступа к файлу |
| 0x00004000 | JOURNAL_DATA_FL | Данные файла должны журналироваться |
| 0x00040000 | HUGE_FILE_FL | Гигантский файл. Размер файла указывается в логических блоках, а не в секторах |
| 0x00080000 | EXTENTS_FL | Файл использует экстенты |

**Расширенные атрибуты**

В **ext3** расширенные атрибуты хранились в отдельном блоке на диске. В индексном дескрипторе в стандартных полях содержится поле ACL, в котором хранится номер блока с расширенными атрибутами. Атрибуты могут быть прочитаны независимо от основных данных файла. Несколько индексных дескрипторов с одинаковыми расширенными атрибутами могут ссылаться на один и тот же блок. Сходство блоков определяется по истории недавних вызовов файлов. **Ext4** может хранить расширенные атрибуты как в отдельном блоке, так и прямо в индексном дескрипторе, если они туда поместятся.

ACL или Access Control List состоит из набора записей. Каждый из трех типов пользователей представлен записью в ACL. Минимальный ACL содержит 3 записи: владелец, группа владельца и остальные. Этот набор указан в первых 2 байтах индексного дескриптора и выводится командой **`ls -l`**. Расширенный ACL содержит больше записей: в них можно указывать конкретного пользователя и конкретную группу, а также маску [18].

Записи в ACL могут быть нескольких типов, которые представлены в табл. П.2.8. Каждый ACL состоит из заголовка, описателей элементов (или просто элементов) и их значений.

Таблица П.2.8

Типы записей в ACL

| **Тип** | **Текстовая форма** |
|---|---|
| Owner | user::rwx |
| Named user | user:name:rwx |
| Owning group | group::rwx |
| Named group | group:name:rwx |
| Mask | mask::rwx |
| Others | other::rwx |

Заголовки атрибутов, хранимых в индексном дескрипторе и в блоке, разные, но

хранятся элементы списков в одинаковом формате. Формат ACL, распределенного по свободному пространству блока или внутри индексного дескриптора, может быть представлен в следующем виде:

Заголовок

Элемент 1

Элемент 2

Элемент 3

4 нулевых байта

…

Величина 3

Величина 2

Величина 1

В отдельном блоке диска дескрипторы элементов сортируются, в индексном дескрипторе элементы несортированы. Значения атрибутов (величины) выровнены по концу блока без особого порядка [17].

Заголовки в блоке и в индексном дескрипторе отличаются друг от друга. В табл. П.2.9 представлен формат заголовка в блоке.

Таблица П.2.9

Формат заголовка ACL в блоке

| **Размер, байт** | **Поле** |
|---|---|
| 4 | Магическое число 0xEA020000 |
| 4 | Число ссылок |
| 4 | Число использованных блоков на диске |
| 4 | Хэш-функция от всех атрибутов |
| 4x4 | Зарезервировано |

В качестве заголовка ACL в индексном дескрипторе используется магическое число **`0xEA020000`**. Описатели элементов расширенного атрибута имеют формат, представленный в табл. 2.10.

Таблица П.2.10

Формат элемента расширенного атрибута

| **Размер, байт** | **Поле** |
|---|---|
| 1 | Длина имени |

| | |
|---|---|
| 1 | Индекс имени атрибута |
| 2 | Смещение величины атрибута от начала блока |
| 4 | Номер блока, в котором хранится значение |
| 4 | Размер величины атрибута |
| 4 | Хэш-функция от имени и значения атрибута |
| - | Имя атрибута |

Установить расширенные атрибуты для файла можно утилитой **setfacl**. Например, установить для пользователя право на запись в файл можно командой **setfacl –m u:username:rw- <path_to_file>**. Просмотреть расширенные атрибуты можно командой **getfacl <path_to_file>**.

В **ext4** расширенные атрибуты хранятся прямо в индексном дескрипторе, что можно наблюдать на рис. П.2.6, где представлен индексный дескриптор с расширенными атрибутами, хранимыми внутри самого дескриптора за пределами фиксированных полей.

В листинге выделено подчеркиванием магическое число **0xEA020000**, идущее сразу после фиксированных полей индексного дескриптора. Следующие 16 байтов занимает заголовок элемента атрибута (его имя имеет нулевую длину), а последние 28 байтов (именно такой размер указан в заголовке элемента) в блоке занимает значение атрибута. При этом поле «ACL» индексного дескриптора (выделено подчеркиванием) содержит нулевое значение, то есть не содержит ссылок на внешний блок данных.

```
0x00000000  B4 81 00 00 44 42 00 00 : 1E 23 82 4B 55 21 82 4B  ....DB...#.KU!.K
0x00000010  79 68 81 4B 00 00 00 00 : 00 00 01 00 28 00 00 00  yh.K........(...
0x00000020  00 00 08 00 01 00 00 00 : 0A F3 01 00 04 00 00 00  ................
0x00000030  00 00 00 00 00 00 00 00 : 05 00 00 00 EA 88 0C 00  ................
0x00000040  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x00000050  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x00000060  00 00 00 00 A9 91 3B 8A : 00 00 00 00 00 00 00 00  ......;.........
0x00000070  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x00000080  1C 00 00 00 C4 6B A7 33 : F4 8A DA C3 20 4E 14 B9  .....k.3.... N..
0x00000090  79 68 81 4B CC 7A F7 AC : 00 00 00 00 00 00 02 EA  yh.K.z..........
0x000000A0  00 02 44 00 00 00 00 00 : 1C 00 00 00 00 00 00 00  ..D.............
0x000000B0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000000C0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000000D0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000000E0  00 00 00 00 01 00 00 00 : 01 00 06 00 02 00 06 00  ................
```

```
0x000000F0  E8 03 00 00 04 00 04 00 : 10 00 06 00 20 00 04 00  ............ ...
```

Рис. П.2.6. Индексный дескриптор с расширенными атрибутами

**Формат директорий**

В **ext3** директория представляет собой фиксированный список сопоставлений индексного дескриптора файла и его имени. Структура записи в директории показана в табл. П.2.11.

Таблица П.2.11

Структура записи в директории в **ext3**

| **Размер, байт** | **Поле** |
|---|---|
| 4 | Индексный дескриптор |
| 2 | Длина записи |
| 2 | Длина имени файла |
| 255 | Имя файла |

В **ext4** структура записи в директории претерпела небольшие изменения, что показано в табл. П.2.12.

Таблица П.2.12

Структура записи в директории в **ext4**

| **Размер, байт** | **Поле** |
|---|---|
| 4 | Индексный дескриптор |
| 2 | Длина записи |
| 1 | Длина имени файла |
| 1 | Тип файла |
| 255 | Имя файла |

Типы файлов: 0 – неизвестный, 1 – обычный, 2 – каталог, 3 – файл символьного устройства, 4 – файл блочного устройства, 5 – именованный канал, 6 – сокет, 7 – символическая ссылка.

В **ext4** для превышения лимита в 32 000 подкаталогов введено индексирование директорий. Если установлен соответствующий флаг файловой системы, записи в директориях хэшируются. Неиндексированная директория занимает минимум 1 блок. Минимальный размер индексированной директории должен составлять 3 блока. 0-й блок

при этом становится корнем дерева, остальные становятся «листьями».

Блоки-«листья» представляют собой традиционный список имен файлов и индексных дескрипторов. Поэтому, если корень дерева окажется поврежденным, можно будет отыскать файлы, используя традиционную схему. Корень дерева содержит хэш-функции блоков-«листьев», каждый такой хэш представляет последовательность записей директории. Хэш в **ext4** может быть как 32-битным, так и 64-битным. Для обеспечения совместимости с файловыми системами, не использующими индексирование, первые 8 байтов корня дерева содержат указание на следующий блок директории, т. е. ядро, не поддерживающее индексирование, будет игнорировать набор хэш-функций как удаленные из директории файлы.

Хэш-версии (алгоритмы хэширования) определяются в суперблоке файловой системы и могут принимать следующие значения: 0 – LEGACY, 1 – HALF MD4, 2 – TEA, 3 – LEGACY UNSIGNED, 4 – HALF MD4 UNSIGNED, 5 – TEA UNSIGNED.

Для хэширования используется так называемое «зерно» – некоторое инициализирующее значение, которое также определено в соответствующем поле суперблока. Если зерно содержит нули по какой-то причине, используется зерно по умолчанию [15].

Реализация H-дерева сейчас ограничена 510 – 511 4-килобайтными блоками-«листьями»: именно такое количество может быть проиндексировано 2-уровневым деревом (дерево должно иметь постоянную глубину). Возможно, в будущем количество уровней увеличат до 3. Также в директориях существует ограничение на размер файла в 2 Гб, потому что размер поля i_size в директории ограничен 32 битами [17].

Предполагается целиком помещать индексный дескриптор директории в саму директорию для ускорения чтения-записи при обращении к функции **readdir**. Также предполагается при динамическом выделении индексных дескрипторов использовать директории как контейнер для таблицы индексных дескрипторов. Для файлов, имеющих жесткие ссылки в других директориях, проблему решает счетчик числа связей в индексном дескрипторе.

Также разработчики предлагают хранить одно или несколько имен файлов в самом индексном дескрипторе в поле расширенных атрибутов файла. Имя должно хранить номер индексного дескриптора директории, к которой файл принадлежит [16].

Удаление файла из директории в англоязычной литературе носит название **unlink**, то есть удаление ссылки. На самом деле при удалении файла в предыдущей записи директории модифицируется поле «длина записи», которое может указывать на конец директории или на следующую запись. Сам индексный дескриптор и имя файла остаются в директории и могут быть прочитаны, что используется некоторыми программами по восстановлению файлов [17].

При чтении директорий с диска игнорируются ошибки ввода/вывода, чтобы у пользователей была возможность восстановить часть данных, если в директории есть сбойные сектора [18].

**Формат дерева экстентов**

Экстент представляет собой непрерывную последовательность блоков, идентифицируемую номером первого блока последовательности и длиной последовательности.

Один экстент может адресовать до 128 Мб. Для отображения файлов большего размера используется структура дерева. Экстенты хранятся в узлах дерева. Конечные узлы, содержащие указатели на сами экстенты, называются «листьями», остальные узлы, содержащие указатели на другие узлы дерева, называются «индексами» [16].

Каждый узел, лист или индекс, даже хранимый непосредственно в индексном дескрипторе, начинается с заголовка узла, представленного в табл. П.2.13. Формат экстента представлен в табл. П.2.14, а формат индекса экстента – в табл. П.2.15.

Таблица П.2.13

Формат заголовка узла

| **Размер поля, байт** | **Поле** |
|---|---|
| 2 | Магическое число 0xF30A |
| 2 | Число входов |
| 2 | Максимальное число входов |
| 2 | Глубина дерева |
| 4 | Поколение дерева (пока не используется) |

Таблица П.2.14

Формат экстента

| **Размер поля, байт** | **Поле** |
|---|---|
| 4 | Первый логический блок экстента |
| 2 | Длина экстента |
| 2 | Старшие 16 битов номера физического блока |
| 4 | Младшие 32 бита номера физического блока |

Таблица П.2.15

Формат индекса экстента

| **Размер** | **Поле** |
|---|---|

| **поля, байт** | |
|---|---|
| 4 | Номер логического блока, с которого начинается индекс |
| 4 | Указатель на физический блок следующего уровня |
| 2 | Старшие 16 битов физического блока |
| 2 | Зарезервировано |

Максимально возможное число блоков, которые может адресовать один экстент, составляет $2^{15} = 32768$, поскольку старший значащий бит в поле «длина экстента» используется для указания на неинициализированный (зарезервированный) экстент. Если это поле принимает максимальное значение, т. е. **0x8000**, то, несмотря на единичное значение старшего значащего бита, это инициализированный экстент, имеющий длину 32768. Таким образом, максимальная длина инициализированного экстента – 32768, а неинициализированного – 32767.

На рис. П.2.7 представлено дерево экстентов, состоящее из основных описанных выше структур [17].

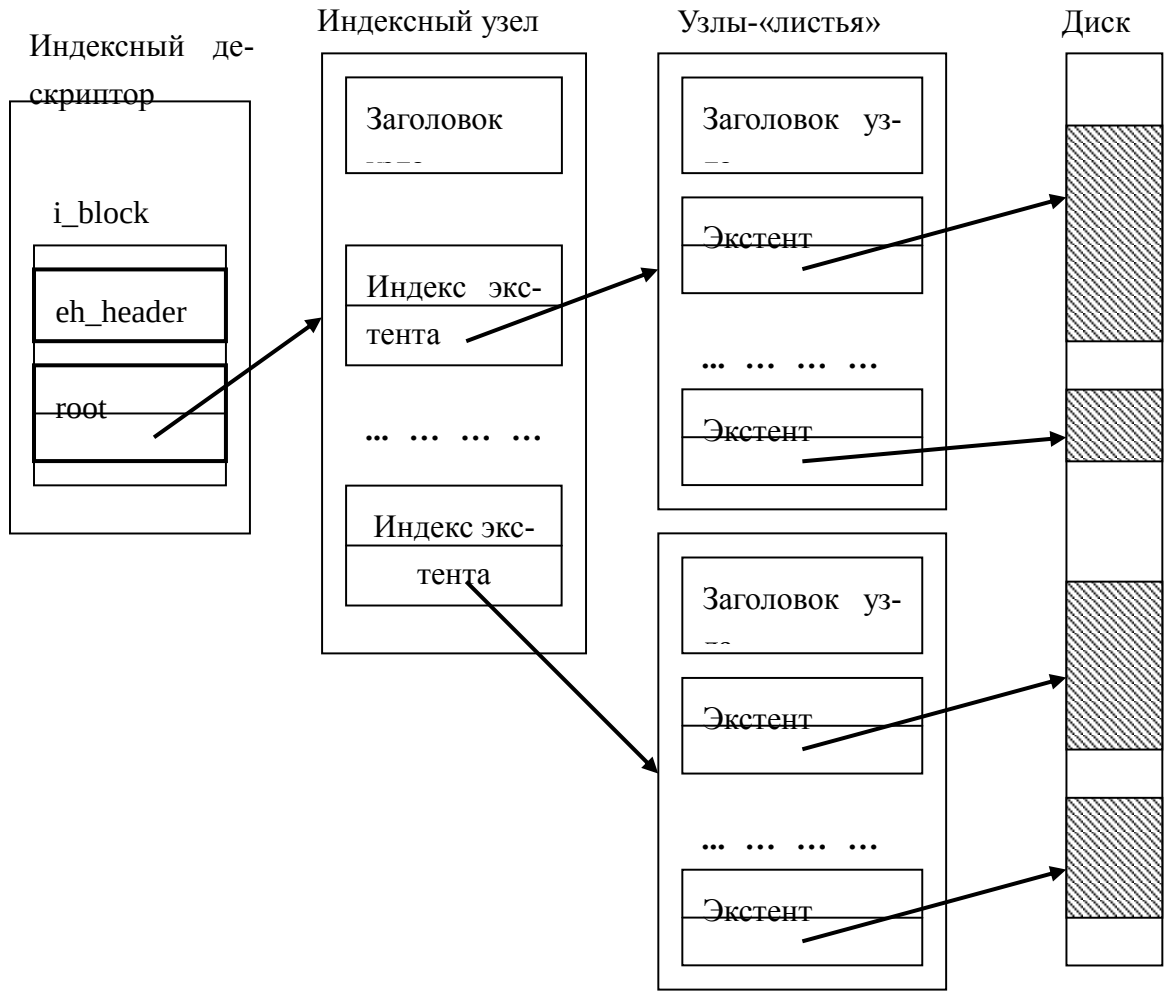


Рис. П.2.7. Структура дерева экстентов

После того как ФС смонтирована, все новые файлы создаются с картой экстен-

тов.

Карта экстентов – не слишком эффективный способ хранения разрозненных или сильно фрагментированных файлов. Разработчики собираются ввести новый тип экстента, с картой блоков. Другое магическое число в заголовке экстента будет определять новый тип «листа», содержащего список номеров выделенных блоков, так, как это делается сейчас в **ext3** [17]. Карта блоков по-прежнему доступна из **ext4**, если в индексном дескрипторе не выставлен флаг EXTENTS_FL.

**Принципы выделения блоков и индексных дескрипторов**

В **ext4** алгоритмы выделения блоков подчиняются принципам уменьшения фрагментации диска и уменьшению времени обращения к файлам.

Исходные коды ядра содержат разнообразные алгоритмы выделения (аллокации) блоков. За выделение блоков группами (множественное выделение блоков) отвечает так называемый аллокатор Орлова.

При выделении блоков для файла ФС учитывает номер группы, в которой находится данный индексный дескриптор. Таким образом, предпочтение отдается блокам той же группы, где находится дескриптор, а дочерние дескрипторы по возможности размещаются недалеко от родительского.

Если контрольная сумма группы блоков оказывается неверной, группа становится доступной только для чтения, и в ней запрещается выделение блоков и индексных дескрипторов.

Если группа блоков еще не инициализирована при выделении в ней индексного дескриптора, она помечается как инициализированная [15].

**Выделение блоков**

В индексном дескрипторе есть список предварительно зарезервированных блоков, в котором указывается начало зарезервированного пространства (логический и физический блоки), длина и доступное свободное место в этом зарезервированном пространстве.

При выделении блоков файлу аллокатор первым делом смотрит именно на этот список. Если файловая система не смогла выделить блоки из зарезервированного для индексного дескриптора пространства или если используется групповое резервирование, происходит поиск блоков в зарезервированном пространстве локальной группы.

Одним из новшеств **ext4** является множественное выделение блоков, что подразумевает поиск длинного непрерывного пространства на диске. Перед запросом на выделение блоков количество требуемых блоков нормализуется, т. е. выделяется больше блоков, чем файлу необходимо. Лишние блоки добавляются в список зарезервированных блоков для файла. Нормирование запроса происходит либо на основе данных о файле (если используется индивидуальное выделение блоков), либо, для группового

выделения, на основе постоянной величины в 512 блоков. Эта величина содержится в **/sys/fs/ext4/<partition/mb_group _prealloc** и может быть изменена.

При выделении блоков ищется группа с наиболее длинной последовательностью блоков и именно она выбирается. Для большей безопасности записи все остальные операции поиска экстента в других группах и в этой блокируются, пока система не завершит выделение [18].

**Резервирование блоков**

Некоторые приложения, например базы данных или потоковые медиапрограммы, нуждаются в выделении блоков для файла заранее, при этом не инициализируя блоков данными или нулями. Предварительное выделение гарантирует, что под файл будет выделено столько непрерывного дискового пространства, сколько возможно. Эта функция полезна, если приложению заранее известно, сколько пространства необходимо будет выделить под файл. Файловая система интерпретирует выделенные, но неинициализированные блоки как заполненные нулями. Предварительное выделение должно быть устойчиво к перезагрузкам [16].

Для приложений, использующих только последовательную запись в файл, довольно легко отделить инициализированную и неинициализированную части файла. Это можно сделать, добавив к файлу «водяной знак», определяющий размер инициализированной части файла. Однако это неэффективно для баз данных или других приложений, в которых запись в предварительно выделенные блоки происходит случайным образом в разные участки. ФС должна определять последовательность неинициализированных блоков данных в середине файла.

В **ext4** это решается использованием старшего значащего бита в поле длины экстента, который определяет, содержит ли экстент неинициализированные данные. При чтении неинициализированного экстента переключатель виртуальной файловой системы VFS возвращает блоки, заполненные нулями. При записи экстент может быть разбит на инициализированную и неинициализированную части, вторая из которых может быть слита со смежным экстентом, если пространство на диске непрерывно [16].

В **ext4** можно выделить два вида резервирования блоков: групповое и индивидуальное для конкретного индексного дескриптора. В зависимости от размера файла система решает, делать ли групповое или индивидуальное резервирование. Порог этот по умолчанию равен 16 блокам. Если файл займет на диске меньше 16 блоков, происходит групповое резервирование для того, чтобы маленькие файлы лежали на диске как можно ближе друг к другу. Пороговое значение может быть изменено в разделе **/sys/fs/ext4/<partition>/mb_stream_req**.

Дескриптор резервирования содержит информацию о типе резервирования (групповое или индивидуальное), список индексных дескрипторов или групп, начале и длине последовательности блоков.

Дескриптор резервирования активен, пока блок не будет помечен на битовой

карте как занятый. Дескриптор меняется только после того, как изменена битовая карта [18].

**Выделение индексных дескрипторов**

Если создаваемый индексный дескриптор относится к каталогу, система ищет группы блоков с большим свободным пространством и маленьким отношением количества каталогов к количеству обычных файлов. Если создаваемый индексный дескриптор относится к обычному файлу, ФС старается поместить его в ту же группу блоков, где находится родительский каталог [18].

Аллокатор Орлова для каталогов учитывает количество уже существующих директорий в группе блоков (их не должно быть слишком много), число свободных индексных дескрипторов (их не должно быть слишком мало), число свободных блоков (их также не должно быть слишком мало).

Дескрипторы каталогов стараются помещать в первую группу гибкой группы блоков, а дескрипторы обычных файлов – во все остальные (т. е. начиная со второй).

При удалении файла (последнего имени) индексный дескриптор очищается после того, как удалена запись в директории. Очистка индексного дескриптора происходит перед тем, как он помечается в битовой карте как свободный.

Функция множественного выделения блоков хранит последние N выделений в памяти. Их можно просмотреть в файле **`/proc/fs/ext4/<dev>/mb_history`**. В истории сохраняются данные о выделении и удалении блоков, о предварительном резервировании и снятии резервирования [18].

**Журналирование**

Структуры журнала описаны в заголовочном файле **`jbd2.h`**. Файл находится по адресу **`/usr/include/linux/jbd2.h`** или **`/usr/src/linux-headers-2.6.31-17/include/linux/jbd2.h`** (зависит от версии ядра системы). Журнал для файловой системы может быть и внешним, хранимым на отдельном блочном устройстве.

В **`ext3`** и **`ext4`** используется «физическое журналирование», т. е. в качестве основной единицы ведения журнала используется физический блок. Подход, когда хранятся только изменяемые байты, а не целые блоки, называется «логическим журналированием» (используется, например, XFS).

Драйвер файловой системы отслеживает обработку целых блоков данных и группирует их в отдельный объект, называемый транзакцией. Транзакция будет завершена только после записи на диск всех данных.

Транзакции являются групповыми операциями, выполняемыми или невыполняемыми как одна единая операция, т. е. атомарно. Журналируемые файловые системы (ext3, NTFS) в случае сбоя делают автоматический «откат» при следующей загрузке, и потери кластеров/блоков не происходит. Создание/удаление/переименование файла –

это атомарные операции, которые не могут допустить промежуточных состояний. А вот перемещение файла или запись на диск – более сложные операции. Поддержка транзакций не может застраховать от потери записываемых данных.

Журнал состоит из транзакций, имеющих непрерывно возрастающие номера. Как только достигнут конец журнала, запись начинается сначала. Таким образом, журнал – это файл с круговой записью. Если система была размонтирована без ошибок, то запись всегда начинается с самого начала [15].

Когда в файловой системе происходят какие-либо изменения, в журнал записываются целые блоки с метаданными, которые подлежат изменению (это может быть таблица индексных дескрипторов, суперблок и т.д.), затем эти блоки переписываются на жесткий диск и после этого в журнал записывается подтверждение.

Каждая транзакция состоит из одного и более дескрипторов, а также блоков данных. Последний дескриптор в транзакции – это блок подтверждения, который сигнализирует о том, что транзакция прошла успешно и данные из предыдущих дескрипторов были записаны.

Существует еще два вида дескрипторов: блоки отмены и блоки, содержащие тэги (иногда их называют просто дескрипторами). Блоки отмены заполняются номерами блоков, которые должны быть удалены из журнала во время этой транзакции. Дескриптор состоит из последовательности тэгов и идет перед блоками с метаданными.

Тэг – это структура, которая определяет последовательность журнальных блоков (не блоков файловой системы), данные с которых должны быть записаны в указанные блоки файловой системы [16]. Номер журнального блока, который должен быть записан в указанный блок файловой системы, определяется порядковым номером тэга в дескрипторе. Структура тэга определена в заголовочном файле **`jbd2.h`** как *journal_block_tag_s* и приведена в табл. П.2.16.

Таблица П.2.16

Структура тэга

| **Размер, байт** | **Смещение** | **Назначение** |
|---|---|---|
| 4 | 0 (0h) | Номер блока на диске |
| 4 | 4 (4h) | Флаги |
| 4 | 8 (8h) | Если включена поддержка 48-битной адресации, в тэг записываются старшие значащие биты номера блока |

Первый блок журнала содержит «журнальный суперблок». Его структура определена в заголовочном файле **`jbd2.h`** как *journal_superblock_t* и приведена в таблице П.2.17.

Таблица П.2.17

Формат журнального суперблока

| **Размер, байт** | **Смещение** | **Назначение** |
|---|---|---|
| 12 | 0 (0h) | Заголовок (одинаков для всех дескрипторов) |
| 4 | 12 (Ch) | Размер блоков журнального устройства. Блоки журнала отличаются от блоков файловой системы |
| 4 | 16 (10h) | Количество блоков в журнальном устройстве |
| 4 | 20 (14h) | Первый блок журнала, содержащий информацию |
| 4 | 24 (18h) | Первый ID подтверждения, ожидаемый журналом |
| 4 | 28 (1Ch) | Номер блока начала журнала |
| 4 | 32 (20h) | Код ошибки (устанавливается функцией jbd2_journal_abort() ) |
| Следующие поля действительны только для журнального суперблока версии 2 | | |
| 4 | 36 (24h) | Флаги compat |
| 4 | 40 (28h) | Флаги incompat |
| 4 | 44 (2Ch) | Флаги rocompat |
| 16 | 48 (30h) | Журнальный UUID |
| 4 | 64 (40h) | Число файловых систем, совместно использующих журнал |
| 4 | 68 (44h) | Номер блока динамической копии суперблока журнала |
| 4 | 72 (48h) | Максимальное число журнальных блоков на транзакцию |
| 4 | 76 (4Ch) | Максимальное число блоков с данными на транзакцию |
| 176 | 80 (50h) | Заполнение |
| 16x48 | 256 (100h) | ID всех файловых систем, совместно использующих журнал |

На рис. П.2.8 приведен фрагмент дескриптора, содержащего тэги. Номера физических блоков выделены подчеркиванием.

```
0x00094000  C0 3B 39 98 00 00 00 01 : 00 01 62 0D 00 00 04 BC  .;9.......b.....
0x00094010  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x00094020  00 00 00 00 00 00 05 E6 : 00 00 00 02 00 00 04 A5  ................
0x00094030  00 00 00 02 00 00 18 6B : 00 00 00 02 00 00 04 BB  .......k........
0x00094040  00 00 00 02 00 00 0E 69 : 00 00 00 02 00 00 05 84  .......i........
0x00094050  00 00 00 02 00 08 02 02 : 00 00 00 02 00 08 00 23  ...............#
0x00094060  00 00 00 02 00 00 02 74 : 00 00 00 02 00 08 00 21  .......t.......!
0x00094070  00 00 00 02 00 00 03 F5 : 00 00 00 02 00 00 02 80  ................
```

Рис. П.2.8. Фрагмент журнального блока-дескриптора

Заголовок журнала имеет тот же вид, что и заголовки остальных дескрипторов. Заголовок определен в файле **`jbd2.h`** как структура *journal_header_s*. Формат заголовка дескриптора приведен в табл. П.2.18.

Таблица П.2.18

Формат заголовка журнального дескриптора

| **Размер, байт** | **Смещение** | **Назначение** |
|---|---|---|
| 4 | 0 (0h) | Магическое число 0xc03b3998 (первые 4 байта /dev/random) |
| 4 | 4 (4h) | Тип дескриптора: 1 — дескриптор-тэг или просто дескриптор; 2 — подтверждение; 3 — суперблок версия 1; 4 — суперблок версия 2; 5 — блок отмены |
| 4 | 8 (8h) | Номер последовательности |

Существует несколько режимов журналирования для файловых систем **`ext3/ext4`**. Режим работы журнала определяется на этапе монтирования.

1. Journal. В журнал записываются как метаданные, так и содержимое файла до того, как их запись будет подтверждена в основной файловой системе. В этом случае риск потери данных даже при пропадании питания минимален. Однако в большинстве случаев такой режим сильно снижает производительность системы, так как данные необходимо записывать дважды: в основную файловую систему и в журнал.

2. Ordered. В этом режиме журналируются только метаданные, но не содержимое файлов. Однако ФС гарантирует, что содержимое файла будет записано на диск до того, как метаданные, относящиеся к этой транзакции, будут помечены как подтвержденные. Этот режим используется по умолчанию в большинстве дистрибутивов Linux. Если происходит пропадание питания или критическая ошибка ядра во время записи файла, журнал при следующем включении находит новый файл, создание которого не было подтверждено, и запускает процесс очистки, т.е. может записать в журнал подтверждение. Однако бывают ситуации, когда файл перезаписан поверх другой информацией. В этом случае может быть восстановлено промежуточное состояние файла. Худший из случаев – если в восстановленном файле будут перемешаны старые и новые данные.

3. Writeback. Журналируются только метаданные. Содержимое файла может быть записано на диск как до, так и после того, как обновляется журнал. В результате файлы, записанные непосредственно перед крахом системы, могут оказаться поврежденными. Например, файл, присоединенный к другому, может иметь в журнале большую длину, чем есть на самом деле, таким образом, конец файла забивается мусором. Также после восстановления журнала могут неожиданно появиться старые версии файлов. Однако во многих случаях разрыв в синхронизации между журналом и данными

невелик. Например, JFS по умолчанию использует именно этот режим.

В **Ext3** не предусмотрено контрольного суммирования блоков журнала. Недостаток такого метода в том, что если при пропадании питания будут повреждены какие-то блоки журнала, но записан блок подтверждения, система при следующей загрузке посчитает, что журнал не содержит ошибок, и проведет неверные транзакции. В **ext4** при несовпадении контрольных сумм журнала транзакция не состоится, что предотвращает запись поврежденных данных на диск [17].

Контрольные суммы транзакций хранятся в блоке подтверждения. Тип контрольной суммы определяется заголовком этого блока и может принимать значения: CRC32, MD5, SHA1.

Журнал необходим файловой системе для сохранения целостности метаданных, в том числе даже после сбоя системы, но не для восстановления целостности данных пользователя, например данных, по неосторожности удаленных с диска. Существуют программные продукты, которые могут восстанавливать файлы, анализируя журнал, иногда с помощью довольно сложных эвристических алгоритмов, однако полной гарантии восстановления они не дают [18].

## ПРИЛОЖЕНИЕ 3

# СПРАВКА ОБ ОТЛАДЧИКЕ DEBUGFS

DebugFS является самой известной утилитой, предназначенной для работы с файловыми системами **ext2fs** и **ext3fs**. DebugFS является программой интерактивного режима, и основные ее команды реализуются только «внутри» отладчика. Больших удобств в плане наглядности такой режим не предоставляет, тем более что пользователь при многократном повторении одних и тех же команд лишен возможности использовать память командной строки, причем вывод информации в файл реализуется только для отдельных команд.

Отладчик может ограниченно работать с одиночными командами или заранее подготовленным пакетным файлом. Но с большинством «внутренних» команд приходится использовать номер **inode** в угловых скобках **< >**, которые командный интерпретатор воспринимает как команды перенаправления ввода/вывода.

В режиме ввода одиночных команд отладчик можно использовать только для одного практического случая:

```
debugfs –R stats <dev>
```

выводит достаточно полную информацию о суперблоке и обо всех группах блоков. Фрагмент, выведенный такой командой, был показан на рис. 4.6.

Вход в интерактивный режим отладчика производится командой **debugfs** без аргументов. Однако ничего полезного при этом сделать еще нельзя. На попытку ввода любой команды отладчик ответит, что файловая система еще не открыта. Для ее открытия следует задать следующую команду:

```
open –w <dev>
```

Вместо **<dev>** указывается логический раздел блочного устройства. Можно сразу воспользоваться командой

```
debugfs –w <dev>
```

Атрибут **–w** указывается, если нужен доступ к файловой системе в режиме чтения и записи. Все команды, модифицирующие, устанавливающие или удаляющие что-либо, нуждаются в таком режиме. Нелишне предупредить, что ошибка в режиме записи может привести к нежелательным последствиям.

После открытия логического раздела или физического устройства система может известить пользователя об успехе или просто предложить ввести следующую команду. Всего в арсенале отладчика несколько десятков команд, но администратору могут понадобиться только некоторые из них. Если есть необходимость в использовании других команд, следует познакомиться с электронным справочником **man**. Рассмотрим основные ко-

манды (они сгруппированы по выполняемым функциям или последовательности применения):

**clri <file_name>** – очистить индексный дескриптор указанного файла,

**freeb block_number** – команда устанавливает в «0» бит, соответствующий данному логическому блоку в битовой карте блоков, тем самым объявляя его свободным,

**setb block_number** – противоположная по смыслу команда устанавливает соответствующий бит в «1», объявляя блок занятым. Если эти установки произведены необдуманно, утилита **fsck**, будучи автоматически запущена при очередной загрузке системы, найдет эти блоки и поместит их в каталог **/lost+found**,

**freei <inode_number>** – команда устанавливает в «0» бит указанного индексного дескриптора в битовой карте **inode**, объявляя его свободным,

**setb <inode_number>** – команда устанавливает соответствующий бит в «1», объявляя блок **inode** занятым,

**icheck block_number** – команда, позволяющая по номеру логического блока данных найти его индексный дескриптор. Требует от нескольких десятков секунд до нескольких минут поиска на диске. Не всегда может найти **inode** логически удаленного файла,

**ncheck inode_number** – команда, позволяющая узнать имя файла по заданному номеру индексного дескриптора. Также нуждается в непродолжительном времени поиска,

**stats** – эта команда уже приводилась выше. В интерактивном режиме она выводит информацию о суперблоке и всех группах блоков на анализируемом дисковом пространстве,

**stat <inode_number>** – команда позволяет вывести краткую справку о содержимом заданного индексного дескриптора,

**lsdel** – команда не нуждается в дополнительных атрибутах и выводит перечень удаленных **inode**, которые исследователю придется запомнить или записать. Для вывода информации в файл рекомендуется использовать конструкцию с неименованным каналом:

**lsdel | debugfs /dev/hdc3 > /home/file_lsdel**,

**mi <inode_number>** – команда, позволяющая модифицировать каждую запись в указанном индексном дескрипторе. Строки выводятся поочередно: **Mode**, **UID**, **GID**, **Size**, 4 временные отметки и так далее. Слева от курсора выводится действующее значение, в позицию курсора пользователь может ввести, что ему требуется. Пример использования команды:

```
debugfs:  mi <148003>
Mode  [0100644]
User ID  [503]
Group ID  [100]
Size  [6065]
Creation time  [833201524]
Modification time  [832708049]
Access time  [826012887]
Deletion time  [833201524]
Link count  [0] 1
Block count  [12]
File flags  [0x0]
Reserved1  [0]
File acl  [0]
Directory acl  [0]
Fragment address  [0]
Fragment number  [0]
Fragment size  [0]
Direct Block #0  [594810]
Direct Block #1  [594811]
Direct Block #2  [594814]
Direct Block #3  [594815]
Direct Block #4  [594816]
Direct Block #5  [594817
Direct Block #6  [0]
Direct Block #7  [0]
Direct Block #8  [0]
Direct Block #9  [0]
Direct Block #10  [0]
Direct Block #11  [0]
Indirect Block  [0]
Double Indirect Block  [0]
Triple Indirect Block  [0]
```

**help** – вывод справки о командах отладчика,

**close** – закрыть файловую систему, открытую командой **open**,

**quit** – выйти из отладчика в режим командной строки стандартной оболочки.

ПРИЛОЖЕНИЕ 4

# СТРУКТУРА И ИСХОДНЫЙ КОД ПРОГРАММЫ EXTVIEW

Дисковый редактор **lde** и отладчик ФС **debugfs** долго и успешно использовались для исследования и ремонта файловой системы **ext2fs**. Но они давно не обновлялись, и для работы с третьей и четвертой системами фактически непригодны.

**Lde** некорректно выводит информацию об индексных дескрипторах, начиная с файловой системы **ext3**. **Debugfs** не выводит списка удаленных индексных дескрипторов, хотя при восстановлении файлов это может сыграть существенную роль. Кроме того, **debugfs** требует почти во всех случаях открытия файловой системы. Ни одна из этих программ не работает с экстентами и не может отображать информацию из журнала файловой системы.

На основании полученных в результате исследования данных о файловой системе **ext4**, Желтышевой Екатериной Дмитриевной разработана альтернативная программа для просмотра и изучения объектов этой ФС под названием **extview** (то есть обозреватель файловой системы ext). Она работает на всех файловых системах ext2/3/4, просматривает как экстенты, так и блоки данных и имеет достаточную функциональность для отображения объектов системы. При этом программа не изменяет данные на диске и не требует монтирования рассматриваемой файловой системы.

Для разработки программы был выбран язык программирования C/C++, поскольку этот язык считается «родным» для Linux, а в операционной системе имеется встроенный компилятор **gcc**.

Для удобства чтения кода и написания программы исходный код **extview** разделен на несколько модулей, каждый из которых выполняет свои функции независимо от других. Благодаря этому программа легко расширяема: достаточно добавить в систему новый модуль.

Сбор отдельных частей программы в единое целое осуществляется средствами автосборки ОС Linux, а именно утилитой *make*.

Разработанная программа открывает непосредственно файл устройства в режиме «только для чтения». Это позволяет ей работать как при загрузке с внешнего носителя, так и на работающей системе. Система не обязательно должна быть смонтирована, а также не обязательно может являться блочным устройством, это может быть и файл-образ файловой системы.

Программа может выводить информацию на двух языках: русском, если используется языковая локаль с кодировкой UTF-8, и английском во всех остальных случаях.

**Характеристика модулей программы**

Модули программы и связи между ними представлены на рис. П.4.1.

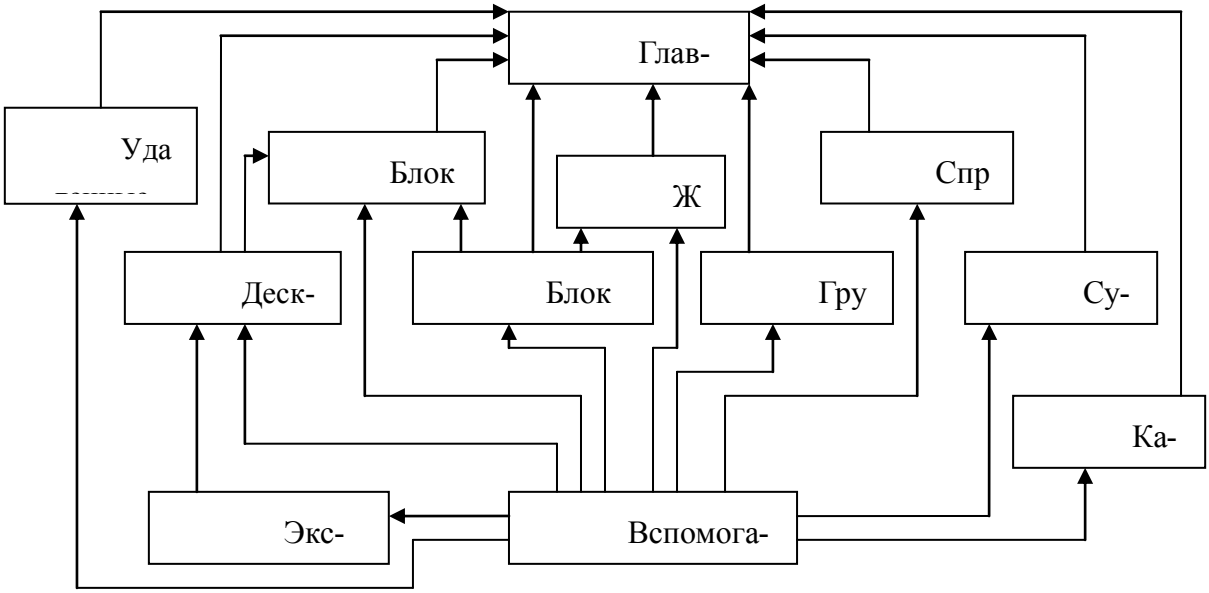


Рис. П.4.1. Состав и связи модулей программы

*Main* (главный) включает в себя все остальные модули. Этот модуль ведет обработку информации, поступающей из командной строки, частично обрабатывает ошибки и решает, какому из модулей передать полученную информацию.

Модуль *option* относится к вспомогательным модулям. В нем описаны функции, необходимые для работы всех остальных модулей. Это чтение с диска определенного количества байтов, определение размера блока файловой системы, преобразование в шестнадцатеричные или бинарные числа и чтение языковой переменной окружения. В перспективе программа сможет поддерживать несколько языков и кодировок.

К вспомогательным модулям, которые отсутствуют в исходном коде main.c, относится также модуль *exttree* (дерево экстентов). Он входит в состав модуля *inodeinfo* (индексный дескриптор) и отвечает за вывод на экран дерева экстентов, если адресация экстентами используется в файле. Модуль состоит всего из одной рекурсивной функции. Функция ищет в блоке данных магическое число экстентов, определяет глубину дерева, считывает и выводит блоки данных, в том числе индексы экстентов.

Модуль *get_block* (блок данных) выводит на экран определенный

блок файловой системы в шестнадцатеричном виде. За основу берется интерфейс дискового редактора *lde*. В данном случае по функционалу *get_block* ничем не отличается от *lde*, этот модуль нужен для удобства пользования программой.

Модуль *group* (группы) выводит информацию о группах блоков в файловой системе. За основу взят интерфейс редактора *debugfs*, в котором отдельной такой команды не было, информация о группах блоков выводилась наряду с информацией о файловой системе командой **`debugfs –R stats <device>.`**

Модуль *inode_block* (блок индексного дескриптора) выводит на экран индексный дескриптор в шестнадцатеричном виде с указанием, в каком блоке и по какому смещению находится запрашиваемый дескриптор. Это нужно для проверки действий программы или для вывода полей, которые в программе не отображаются.

Модуль *inodeinfo* (индексный дескриптор) используется для чтения и вывода на экран информации об указанном индексном дескрипторе. За основу взят интерфейс программы *lde*. Кроме того, в модуле *inodeinfo* содержится несколько новых функций, отсутствующих в программе *lde*: например, вывод экстентов или расширенных атрибутов файла.

Модуль *super* (суперблок) выводит информацию о файловой системе, анализируя суперблок. За основу взят интерфейс встроенного отладчика файловой системы debugfs, вывод команды **`debugfs –R stats <device>`**.

Модуль *direct* (каталоги) выводит элементы директории, указанной по индексному дескриптору, в том числе удаленные и скрытые файлы. В сочетании с опцией «B» интерпретирует входной аргумент как номер блока и пытается вывести элементы директорий из этого блока. Эта опция заложена на будущее, если придется искать имя файла в блоках данных (такая возможность в процессе написания программы рассматривалась), тогда найденный блок можно попытаться интерпретировать как директорию, даже если ее индексный дескриптор неизвестен.

Модуль *jour* (журнал) позволяет просматривать журнальные блоки, если в файловой системе есть внутренний журнал. Позволяет вывести информацию о блоке журнала, а при указании опции «B» еще и сам журнальный блок.

Модуль *lost* (удаленные файлы) используется для поиска удаленных файлов в журнале и на диске. Если указано только имя устройства, выводит список удаленных индексных дескрипторов, перед этим запрашивая время, по которому необходимо отсортировать удаленные дескрипторы. Дескрипторы, удаление которых произошло до указанного времени, на эк-

ран не выводятся.

Модуль *help* выводит справку о программе на русском или английском языке, в зависимости от настроек языковой локализации.

## Make-файл

```
extview: main.o inodeinfo.o option.o exttree.o super.o group.o get_block.o inode_block.o helping.o jour.o lost.o direct.o context.o
	gcc -o extview main.o inodeinfo.o option.o exttree.o super.o group.o get_block.o inode_block.o helping.o jour.o lost.o direct.o context.o

helping.o: helping.c helping.h
	gcc -c helping.c

exttree.o: exttree.c exttree.h
	gcc -c exttree.c

option.o: option.c option.h
	gcc -c option.c

inodeinfo.o: inodeinfo.c inodeinfo.h
	gcc -c inodeinfo.c

super.o: super.c super.h
	gcc -c super.c

group.o: group.c group.h
	gcc -c group.c

get_block.o: get_block.c get_block.h
	gcc -c get_block.c

inode_block.o: inode_block.c inode_block.h
	gcc -c inode_block.c
```

```
jour.o: jour.c jour.h
 gcc -c jour.c

lost.o: lost.c lost.h
 gcc -c lost.c

direct.o: direct.c direct.h
 gcc -c direct.c

context.o: context.c context.h
 gcc -c context.c

main.o: main.c
 gcc -c main.c

clean:
 rm -f *.o
 rm -f extview
```

## Модуль main

Файл main.c

```
#include <stdio.h>
#include <getopt.h>
#include "inodeinfo.h"
#include "super.h"
#include "group.h"
#include "get_block.h"
#include "inode_block.h"
#include "helping.h"
#include "jour.h"
#include "lost.h"
#include "direct.h"
#include "context.h"
void main(int argc,char *argv[])
{
int opt,type1;
unsigned int block,inode;
FILE *disk;
char diskname[80];
char *file=NULL;
int n;
char entr,entr2;
opt=getopt(argc,argv,"i:sgld:cb:j:h-");
switch (opt)
{
      case 'i':
      inode=atoi(optarg);
      if (argc<4) printf("Not enough parameters\n");
      else
      {
            sscanf(argv[3],"%s",&diskname);
            disk=fopen(diskname,"r");
            if (disk!=0)
            {
                  inodeinfo(inode,disk);
                  fclose (disk);
            }
            else printf("Can not open file %s\n",diskname);
```

```
}
break;

case 's':
if (argc<3) printf("Not enough parameters\n");
else
{
      sscanf(argv[2],"%s",&diskname);
      disk=fopen(diskname,"r");
      if (disk==0) printf("Can not open file %s\n",diskname);
      else {super(disk); fclose(disk);}
}
break;

case 'g':
if (argc<3) printf("Not enough parameters\n");
else
{
      sscanf(argv[2],"%s",&diskname);
      disk=fopen(diskname,"r");
      if (disk==0) printf("Can not open file %s\n",diskname);
      else {group(disk); fclose(disk);}
}
break;

case 'l':
if (argc<3) printf("Not enough parameters\n");
if (argc==3)
{
      type1=3;
      sscanf(argv[2],"%s",&diskname);
      inode=0;
      disk=fopen(diskname,"r");
      if (disk==0) printf("Can not open file %s\n",diskname);
      else {lost(disk,inode,type1); fclose(disk);}
}
if (argc==4)
{
      type1=1;
      sscanf(argv[2],"%d",&inode);
```

```
      sscanf(argv[3],"%s",&diskname);
      disk=fopen(diskname,"r");
      if (disk==0) printf("Can not open file %s\n",diskname);
      else {lost(disk,inode,type1); fclose(disk);}
}
if (argc>=5)
{
      sscanf(argv[2],"%c",&entr);
      if (entr=='i') type1=1;
      if (entr=='b') type1=2;
      if (entr!='i'&&entr!='b') type1=1;
      sscanf(argv[3],"%d",&inode);
      sscanf(argv[4],"%s",&diskname);
      disk=fopen(diskname,"r");
      if (disk==0) printf("Can not open file %s\n",diskname);
      else {lost(disk,inode,type1); fclose(disk);}
}
break;

case 'd':
inode=atoi(optarg);
if (argc<4) printf("Not enough parameters\n");
else
{
      sscanf(argv[3],"%s",&diskname);
      if(argc>4) sscanf(argv[4],"%c",&entr);
      if (entr=='B') type1=1;
      else if (entr=='F') type1=2;
      else type1=0;
      if (argc>5) sscanf(argv[5],"%c",&entr2);
      if (entr2=='F'&&entr=='B') type1=3;
      disk=fopen(diskname,"r");
      if (disk==0) printf("Can not open file %s\n",diskname);
      else {direct(inode,disk,type1); fclose(disk);}
}
break;

case 'c':
if (argc<3) printf("Not enough parameters\n");
else
```

```
{
      sscanf(argv[2],"%s",&diskname);
      disk=fopen(diskname,"r");
      if (disk==0) printf("Can not open file %s\n",diskname);
      else {context(disk); fclose(disk);}
}
break;

case 'b':
block=atoi(optarg);
if (argc<4) printf("Not enough parameters\n");
else
{
      sscanf(argv[3],"%s",&diskname);
      disk=fopen(diskname,"r");
      if (disk==0) printf("Can not open file %s\n",diskname);
      else {get_block(block,disk); fclose(disk);}
}
break;

case 'j':
block=atoi(optarg);
if (argc<4) printf("Not enough parameters\n");
else
{
      sscanf(argv[3],"%s",&diskname);
      disk=fopen(diskname,"r");
      int j_type;
      char arg4=0;
      if (argc==5) sscanf(argv[4],"%c",&arg4);
      if (arg4=='B') j_type=1;
      else j_type=0;
      if (disk==0) printf("Can not open file %s\n",diskname);
      else {jour(block,disk,j_type); fclose(disk);}
}
break;

case 'h':
helping();
break;
```

```
      case '-':
      break;
      case '?':
      printf("Unknown argument\n");
      break;
}
const struct option long_opts[] = {
      { "inodeblock", required_argument, NULL, 'a'},
      { "help", no_argument, NULL, 'h'},
      { NULL, 0, NULL, 0}
};
opt=getopt_long(argc,argv,"a:h",long_opts,NULL);
if (opt=='a')
{
      inode=atoi(optarg);
      if (argc<4) printf("Not enough parameters\n");
      else
      {
            sscanf(argv[3],"%s",&diskname);
            disk=fopen(diskname,"r");
            if (disk==0) printf("Can not open file %s\n",diskname);
            else {inode_block(disk,inode); fclose(disk);}
      }
}
if (opt=='h')
      helping();
}
```

## Модуль option

Файл option.h

```
int power(int p1,int p2);
int read_2(FILE *f, unsigned int block_number,int
block_size,\ int block_offset);
unsigned int read_4(FILE *f,unsigned int block_number, \
int block_size,int block_offset);
int get_block_size(FILE *disk);
void hexing(unsigned int decimal,char hex[8],int del[8]);
void binaring(int number, char binar[4]);
int get_local();
```

Файл option.c

```
#include <stdio.h>
#include "option.h"
int power(int p1,int p2){
  int i,result;
  result=p1;
  for (i=0;i<(p2-1);i++)
    result=result*p1;
  if (p2==0) result=1;
  return (result);
}
//Функция считывает с диска 2 байта из указанного номера
блока
//по указанному смещению
int read_2(FILE *f,unsigned int block_number,int
block_size,int block_offset) {
  int g1,g2,i;
  int result;
  int k=0;
  fseek(f,0,SEEK_SET);
  //для быстроты чтения с диска читаем большими блоками
  for (i=0;i<8192;i++)
  {
    if (block_number>524287) block_number=block_number-
524287;
    else {k=i; break;}
  }
```

```
  for (i=0;i<k;i++)
    fseek(f,524287*block_size,SEEK_CUR);
  fseek(f,block_number*block_size,SEEK_CUR);
  fseek(f,block_offset,SEEK_CUR);
  g1=fgetc(f);
  g2=fgetc(f);
  result=g1+g2*256;
  return(result);
}
//Функция считывает 4 байта с диска из блока с указанным
номером
//по указанному смещению
unsigned int read_4(FILE *f,unsigned int block_number,int
block_size,int block_offset)
{
  int g1,g2,g3,g4,i,k;
  int result;
  fseek(f,0,SEEK_SET);
  k=0;
  for (i=0;i<8192;i++)
  {
    if   (block_number>524287)   block_number=block_number-
524287;
    else {k=i; break;}
  }
  for (i=0;i<k;i++)
    fseek(f,524287*block_size,SEEK_CUR);
  fseek(f,block_number*block_size,SEEK_CUR);
  fseek(f,block_offset,SEEK_CUR);
  g1=fgetc(f);
  g2=fgetc(f);
  g3=fgetc(f);
  g4=fgetc(f);
  result=g1+g2*256+g3*256*256+g4*256*256*256;
  return(result);
}
int get_block_size(FILE *disk)
{
  int block_indic,block_size;
  block_indic=read_4(disk,1,1024,24); //1024+24
```

```
    switch(block_indic)
    {
      case 0: block_size=1024; break;
      case 1: block_size=2048; break;
      case 2: block_size=4096; break;
      default: block_size=0;
    }
    return(block_size);
  }
  //Функция записывает шестнадцатеричное представление числа
в //строку и в числовой массив
  void hexing(unsigned int decimal,char hex[8],int del[8]) {
    int i,step;
    for (i=0;i<8;i++){
      step=power(16,8-i-1);
      if (decimal>=step){
        del[i]=decimal/step;
        switch(del[i]){
          case 1: hex[i]='1'; break;
          case 2: hex[i]='2'; break;
          case 3: hex[i]='3'; break;
          case 4: hex[i]='4'; break;
          case 5: hex[i]='5'; break;
          case 6: hex[i]='6'; break;
          case 7: hex[i]='7'; break;
          case 8: hex[i]='8'; break;
          case 9: hex[i]='9'; break;
          case 10: hex[i]='A'; break;
          case 11: hex[i]='B'; break;
          case 12: hex[i]='C'; break;
          case 13: hex[i]='D'; break;
          case 14: hex[i]='E'; break;
          case 15: hex[i]='F'; break;
          default: hex[i]='?';
        }
        decimal=decimal-(del[i]*step);
      }
      else {hex[i]='0'; del[i]=0;}
    }
  }
```

```
    //Функция записывает в символьный массив двоичное пред-
ставление переданного ей числа
    void binaring(int number, char binar[4]) {
      int i,step;
      for (i=0;i<4;i++) {
        step=power(2,3-i);
        if (number>=step) {
          binar[i]='1';
          number=number-step;
        }
        else binar[i]='0';
      }
    }
    //Функция возвращает 1, если используется русская кодиров-
ка UTF-8 и 0, если другие кодировки
    int get_local()
    {
      char *value;
      value=(char*)getenv("LANG");
      if (strcmp("ru_RU.UTF-8",value)==0) return(1);
      else return(0);
    }
```

## Модуль exttree

Файл exttree.h

```
int exttree(FILE *disk,unsigned int offsets[2],\
int block_size,int depth);
```

Файл exttree.c

```
#include <stdio.h>
#include "option.h"
#include "exttree.h"
int exttree(FILE *disk,unsigned int offsets[2],int
block_size,int dep)
{
  int magic,magic_set,depth,entries;
  int i,j,ext_len;
  unsigned int ext_block,next_block,next_offsets[2];
  char hex[8];
  int del[8];
  int loc=get_local();
  magic=read_2(disk,offsets[0],block_size,offsets[1]+40);
  if (magic==62218) magic_set=40+offsets[1];
  else
  {
    magic=read_2(disk,offsets[0],block_size,offsets[1]);
    if (magic==62218) magic_set=0+offsets[1];
    else {
      { if (loc==1)
        printf("Этот узел не содержит магического чис-
ла\n");
        else printf("This node does not contain magic num-
ber");}
        return(1);}
  }
  depth=read_2(disk,offsets[0],block_size,magic_set+6);
  entries=read_2(disk,offsets[0],block_size,magic_set+2);
  if (depth==0)
  {
    for (i=0;i<entries;i++)
    {
    ext_block = read_4 (disk, offsets[0], block_size, \\
```

```
magic_set+20+i*12);
        ext_len = read_2 (disk, offsets[0], block_size, \\
magic_set+16+i*12);
          for (j=0;j<dep;j++) printf("\t");
          if (loc==1) printf("%d-й экстент: первый блок: %d \\
(0x",i+1,ext_block);
          else printf("Extnent %d: first block: %d \\
(0x",i+1,ext_block);
          hexing(ext_block,hex,del);
          printf("%s",hex);
          if (loc==1) printf("), длина: %d\n",ext_len);
          else printf("), length: %d\n",ext_len);
        }
      }
      else
      {
        for (i=0;i<entries;i++)
        {
          next_block = read_4 (disk, offsets[0], block_size,
\\ magic_set+16+i*12);
          hexing(next_block,hex,del);
          for (j=0;j<dep;j++) printf("\t");
          if (loc==1) printf("Следующий узел дерева в блоке:
%d \\ (0x%s)\n",next_block,hex);
          else printf("Next node in block: %d (0x%s)\n", \\
next_block,hex);
          next_offsets[0]=next_block;
          next_offsets[1]=0;
          exttree(disk,next_offsets,block_size,dep+1);
        }
      }
    }
```

## Модуль get_block

Файл get_block.h

```
int get_block(unsigned int block_number, FILE *disk);
```

Файл get_block.c

```
#include <stdio.h>
#include "option.h"
#include "get_block.h"
/*Вместо целого значения функция выводит восьмеричное
 *представление числа
 */
void hexing_pos (int sym,char pos[2])
{
  int del[8];
  char hex[8];
  hexing(sym,hex,del);
  pos[0]=hex[6];
  pos[1]=hex[7];
}
/*Функция считывает подряд 8 байт от текущей позиции кур-
сора в
 *файле и выводит их на экран
 */
void get8(FILE *disk,char points[9])
{
  int sym,j;
  char pos[2];
  for (j=0;j<8;j++)
  {
    sym=fgetc(disk);
    hexing_pos(sym,pos);
    printf(" ");
    printf("%c",pos[0]);
    printf("%c",pos[1]);
    points[j]=sym;
    if (sym<32||sym>126) points[j]='.';
  }
  points[8]='\0';
}
int get_block(unsigned int block_number, FILE *disk)
```

```
    {
      int magic,block_size,loc;
      unsigned int block_pos,block_count;
      int i,j,k=0,str_count,del[8],sym;
      char hex_pos[8],pos[2],points1[9],points2[9];
      magic=read_2(disk,1,1024,56);
      loc=get_local();
      if (magic!=61267)
      {
        if (loc==1)
          printf("Файл устройства не содержит магического чис-
ла\n");
        else printf("File does not contain magic number\n");
        return(1);
      }
      block_size=get_block_size(disk);
      str_count=block_size/16;          /*Вычисляем       число
строк в
                                        *блоке  (строки  по  16
                                  байт)
                                        */
      block_pos=block_number*block_size;    /*Позиция   курсо-
ра, нужна
                                        *для вывода левой
                                        *колонки */
     block_count=block_number;          /*Вспомогательная   пе-
ременная
                                   *для установки курсора в
                                   *начало блока */
      //Устанавливаем курсор в начало блока
      fseek(disk,0,SEEK_SET);
      for (i=0;i<8192;i++)
      {
        if    (block_count>524287)    block_count=block_count-
524287;
        else {k=i; break;}
      }
      for (i=0;i<k;i++)
        fseek(disk,524287*block_size,SEEK_CUR);
      fseek(disk,block_count*block_size,SEEK_CUR);
```

```
  //Считываем байты и выводим на экран
  printf("\n");
  for (i=0;i<str_count;i++)
  {
    hexing(block_pos,hex_pos,del);
    printf("0x%s ",hex_pos);
    get8(disk,points1);
    printf(" :");
    get8(disk,points2);
    printf("  %s%s",points1,points2);
    printf("\n");
    block_pos=block_pos+16;
  }
  printf("\n");
  return (0);
}
```

## Модуль group

Файл group.h

```
int group(FILE *disk);
```

Файл group.c

```
#include <stdio.h>
#include "option.h"
#include "group.h"
int group(FILE *disk)
{
  int magic,block_size,block_count,blopergro;
  unsigned int gr_number,gr_block,gr_count;
  unsigned int block_bitmap, inode_bitmap, inode_table,
  unsigned int free_blocks, free_inodes, directories,
  unsigned int unused_inodes, check_sum;
  char hex_check[8];
  int del[8];
  int loc=get_local();
  int i,j;
  int desc_size=32;
  magic=read_2(disk,1,1024,56);
  if (magic!=61267)
  {
    if (loc==1)
      printf("Файл устройства не содержит магического чис-
ла\n");
    else printf("File does not contain magic number\n");
    return(1);
  }
  block_size=get_block_size(disk);
  if (block_size==1024) gr_block=2;
  else gr_block=1;
  block_count=read_4(disk,1,1024,4);
  blopergro=read_4(disk,1,1024,32);
  gr_count=block_count/blopergro+1;
  if (loc==1) printf("Число групп блоков: %d\n",gr_count);
  else printf("Block group count: %d\n",gr_count);
  for (i=0;i<gr_count;i++)
  {
    if (loc==1) printf("Группа %d:\t",i);
```

```
        else printf("Group %d:\t",i);

block_bitmap=read_4(disk,gr_block,block_size,i*desc_size);
        if (loc==1) printf("карта блоков: %d, ",block_bitmap);
        else printf("block bitmap at: %d, ",block_bitmap);

inode_bitmap=read_4(disk,gr_block,block_size,4+i*desc_size);
        if (loc==1)
          printf("карта дескрипторов: %d,\n",inode_bitmap);
        else printf("inode bitmap at: %d,\n",inode_bitmap);

inode_table=read_4(disk,gr_block,block_size,8+i*desc_size);
        if (loc==1)
          printf("\t\tтаблица   индексных   дескрипторов:   \\
%d,\n",inode_table);
        else printf("\t\tinode table at: %d,\n",inode_table);

free_blocks=read_2(disk,gr_block,block_size,12+i*desc_size);
        if (loc==1)
          printf("\t\tсвободных блоков: %d, ",free_blocks);
        else printf("\t\tfree block count: %d, ",free_blocks);

free_inodes=read_2(disk,gr_block,block_size,14+i*desc_size);
        if (loc==1)
          printf("свободных дескрипторов: %d,\n",free_inodes);
        else printf("free inodes: %d,\n",free_inodes);
        directo-
ries=read_2(disk,gr_block,block_size,16+i*desc_size);
        if      (loc==1)      printf("\t\tдиректорий:      %d,
",directories);
        else printf("\t\tdirectories: %d, ",directories);
        unused_inodes = read_2 (disk, gr_block, block_size, \\
28+i*desc_size);
        if (loc==1)
          printf("неиспользованных      дескрипторов:      \\
%d.\n",unused_inodes);
        else printf("unused inodes: %d.\n",unused_inodes);

check_sum=read_2(disk,gr_block,block_size,30+i*desc_size);
        hexing(check_sum,hex_check,del);
```

```
    if (loc==1) printf("\t\tКонтрольная сумма: 0x");
    else printf("\t\tCheck sum: 0x");
    for (j=0;j<4;j++) printf("%c",hex_check[j+4]);
    printf("\n\n");
  }
  return (0);
}
```

## Модуль inodeinfo

Файл inodeinfo.h

```
int inodeinfo(unsigned int inode, FILE *disk);
```

Файл inodeinfo.c

```
#include <stdio.h>
#include <time.h>
#include <string.h>
#include "option.h"
#include "exttree.h"
#include "inodeinfo.h"
//Функция вычисляет номер блока введенного индексного дескрип-
тора и смещение его в этом блоке, и записывает их в двухэлементный
массив
void inode_offset(FILE *disk, unsigned int inode, int \\
block_size, unsigned int offsets[2])
{
  int inopergro; /*Число индексных дескрипторов в каждой группе
                   *блоков */
  int groupn; //Номер группы блоков
  int gr_desc_size; //Размер описателя группы блоков
  unsigned int inotab; /*Блок, в котором начинается таблица
                     *индексных дескрипторов в данной группе
                     */
  int inode_size; //Размер индексного  дескриптора
  int inoperblo; /*Число индексных дескрипторов в одном блоке
                   *таблицы дескрипторов */
  int offinoblo; /*Относительный номер блока с индексным
                   *дескриптором в группе блоков */
  unsigned int inode_block; /*Номер блока, в котором лежит
                           *индексный дескриптор */
  int inode_num; /*Относительный номер индексного дескриптора в
                   *блоке */
  int inooff; //Смещение индексного дескриптора в блоке
  int p1,p2;
  //Ищем, в какой группе блоков находится индексный дескриптор
  inopergro=read_4(disk,1,1024,40);
  groupn=inode/inopergro;
  if ((inopergro*groupn)==inode) groupn--;
  /*Ищем, в каком блоке начинается таблица индексных
```

```
   *дескрипторов
   */
  gr_desc_size=read_2(disk,1,1024,254);
  if (gr_desc_size==0) gr_desc_size=32;
  if (block_size==1024)
    inotab=read_4(disk,2,block_size,groupn*gr_desc_size+8);
  else inotab=read_4(disk,1,block_size,groupn*gr_desc_size+8);
  //Ищем, в каком блоке лежит индексный дескриптор
  p1=inopergro*groupn;
  p2=inode-p1; /*Относительный номер индексного дескриптора в
            *группе */
  inode_size=read_2(disk,1,1024,88);
  inoperblo=block_size/inode_size;
  offinoblo=p2/inoperblo;
  if ((inoperblo*offinoblo)==p2) offinoblo--;
  inode_block=inotab+offinoblo;
  offsets[0]=inode_block;
  //Ищем относительное смещение индексного дескриптора в блоке
  p1=offinoblo*inoperblo;
  inode_num=p2-p1;
  inooff=(inode_num-1)*inode_size;
  offsets[1]=inooff;
}
//Функция выводит временные метки файла на экран
void time_marks(FILE *disk,unsigned int offsets[2], \\
int block_size)
{
  unsigned int call_time,create_time,modif_time,delete_time;
  time_t call,create,modif,delete;
  struct tm *times;
  int loc=get_local();
  call_time=read_4(disk,offsets[0],block_size,offsets[1]+8);
  create_time=read_4(disk,offsets[0],block_size,offsets[1]+12);
  modif_time=read_4(disk,offsets[0],block_size,offsets[1]+16);
  delete_time=read_4(disk,offsets[0],block_size,offsets[1]+20);
  call=call_time;
  create=create_time;
  modif=modif_time;
  delete=delete_time;
  times=localtime(&call);
```

```
  if (loc==1) printf("Последнее обращение:\t");
  else printf("Last access:\t\t");
  fputs(asctime(times),stdout);
  times=localtime(&create);
  if (loc==1) printf("Создан:\t\t\t");
  else printf("Created:\t\t");
  fputs(asctime(times),stdout);
  times=localtime(&modif);
  if (loc==1) printf("Изменен:\t\t");
  else printf("Modified:\t\t");
  fputs(asctime(times),stdout);
  times=localtime(&delete);
  if (loc==1) printf("Удален:\t\t\t");
  else printf("Deleted:\t\t");
  if (delete_time==0)
  { if (loc==1) printf("Не удалялся\n");
    else printf("Is not deleted yet\n");}
  else fputs(asctime(times),stdout);
}
//Функция выводит на экран тип файла и права доступа к нему
void file_type(FILE *disk,unsigned int offsets[2], \\
int block_size) {
  int type,type2,rights;
  char right[12],resul_right[10];
  char symb_type;
  int i,step;
  int loc=get_local();
  type2=read_2(disk,offsets[0],block_size,offsets[1]);
  if (loc==1) printf("Тип файла:\t\t");
  else printf("File type:\t\t");
  type=type2/4096;
  switch (type) {
    case 8: if (loc==1) printf("Обычный файл\n");
      else printf("Regular file\n"); symb_type='-'; break;
    case 4: if (loc==1) printf("Каталог\n");
      else printf("Directory\n"); symb_type='d'; break;
    case 10: if (loc==1) printf("Символическая ссылка\n");
      else printf("Symlink\n"); symb_type='l'; break;
    case 12: if (loc==1) printf("Сокет\n");
      else printf("Socket\n"); symb_type='s'; break;
```

```
        case 1: if (loc==1) printf("Именованный канал\n");
          else printf("Named pipe\n"); symb_type='f'; break;
        case 6: if (loc==1) printf("Файл блочного устройства\n");
          else printf("Block device\n"); symb_type='b'; break;
        case 2: if (loc==1) printf("Файл символьного устройст-
ва\n");
          else printf("Symbolic device\n");symb_type='c'; break;
        default: if (loc==1) printf("Неизвестный тип файла\n");
          else printf("Unknown file type\n"); symb_type='?';
      }
      rights=type2-(type*4096);
      for (i=0;i<12;i++) {
        step=power(2,(12-i-1));
        if (rights<step) right[i]='-';
        else {
          rights=rights-step;
          if (i==3||i==6||i==9) right[i]='r';
          if (i==4||i==7||i==10) right[i]='w';
          if (i==5||i==8||i==11) right[i]='x';
          if (i<3) right[i]=1;
        }
      }
      resul_right[0]=symb_type;
      for (i=1;i<10;i++) {
        resul_right[i]=right[i+2];
      }
      if (right[0]==1&&right[5]=='x') resul_right[3]='s';
      if (right[0]==1&&right[5]=='-') resul_right[3]='S';
      if (right[2]==1&&right[11]=='x'&&symb_type=='d')
        resul_right[9]='t';
      if (right[2]==1&&right[11]=='-') resul_right[9]='T';
      if (loc==1) printf("Права доступа:\t\t%s\n",resul_right);
      else printf("Rights:\t\t\t%s\n",resul_right);
    }
    /*Функция находит идентификатор пользователя или группы в
     *служебном файле и записывает его имя в символьный массив
     */
    void ugid(FILE *f,int id, char username[60]) {
      int k=0,i,id2;
      char pass[100];
```

```
  do {
    fgets(pass,100,f);
    int l=0,len1,len2;
    char uidstr[6];
    for(i=0;i<100;i++) {
      if (pass[i]==':') l++;
      if (l==2) {len1=i; l++;}
      if (l==4) {len2=i; break;}
    }
    for (i=0;i<(len2-len1-1);i++) uidstr[i]=pass[i+len1+1];
    uidstr[len2-len1]='\0';
    id2=atoi(uidstr);
    k++;
  } while (id2!=id&&k<65536);
  if (id2!=id) username[0]='0';
  else {
    for (i=0;i<60;i++) {
      username[i]=pass[i];
      if (pass[i]==':') {username[i]='\0'; break;}
    }
  }
}
//Определение и вывод на экран расширенных атрибутов файла
void lsattr(int flag_mas[8]) {
  char attr1[4],attr2[4],attr3[4],attr4[4],attr5[4];
  char bin_attr[21],symb_attr[21],str_attr[300];
  int i,str_count=0;
  int loc=get_local();
  str_attr[0]='\0';
  binaring(flag_mas[7],attr1);
  binaring(flag_mas[6],attr2);
  binaring(flag_mas[5],attr3);
  binaring(flag_mas[4],attr4);
  binaring(flag_mas[3],attr5);
  for (i=0;i<4;i++) bin_attr[4-i-1]=attr1[i];
  for (i=0;i<4;i++) bin_attr[8-i-1]=attr2[i];
  for (i=0;i<4;i++) bin_attr[12-i-1]=attr3[i];
  for (i=0;i<4;i++) bin_attr[16-i-1]=attr4[i];
  for (i=0;i<4;i++) bin_attr[20-i-1]=attr5[i];
  bin_attr[20]='\0';
```

```
for (i=0;i<20;i++) {
  if (bin_attr[i]!='0')
  switch(i){
    case 0: symb_attr[i]='s';
      if(str_count!=0) strcat(str_attr,", ");
      if (loc==1)
        strcat(str_attr,"с гарантированным удалением");
      else strcat(str_attr,"secure deletion");
      str_count++;break;
    case 1: symb_attr[i]='u';
      if(str_count!=0) strcat(str_attr,", ");
      if (loc==1) strcat(str_attr,"неудаляемый");
      else strcat(str_attr,"undelete");
      str_count++;break;
    case 2: symb_attr[i]='c';
      if(str_count!=0) strcat(str_attr,", ");
      if (loc==1) strcat(str_attr,"сжатый");
      else strcat(str_attr,"compress file");
      str_count++;break;
    case 3: symb_attr[i]='S';
      if(str_count!=0) strcat(str_attr,", ");
      if (loc==1) strcat(str_attr,"синхронно обновляемый");
      else strcat(str_attr,"synchronous updates");
      str_count++;break;
    case 4: symb_attr[i]='i';
      if(str_count!=0) strcat(str_attr,", ");
      if (loc==1) strcat(str_attr,"недосягаемый");
      else strcat(str_attr,"immutable file");
      str_count++;break;
    case 5: symb_attr[i]='a';
      if(str_count!=0) strcat(str_attr,", ");
      if (loc==1)
        strcat(str_attr,"только добавление в конец");
      else strcat(str_attr,"writes to file may only append");
      str_count++;break;
    case 6: symb_attr[i]='d';
      if(str_count!=0) strcat(str_attr,", ");
      if (loc==1) strcat(str_attr,"недампируемый");
      else strcat(str_attr,"do not dump file");
      str_count++;break;
```

```
            case 7: symb_attr[i]='A';
              if(str_count!=0) strcat(str_attr,", ");
              if (loc==1)
                strcat(str_attr,"не обновлять время доступа");
              else strcat(str_attr,"do not update atime");
              str_count++;break;
            case 14: symb_attr[i]='j';
              if(str_count!=0) strcat(str_attr,", ");
              if  (loc==1)  strcat(str_attr,"с  журналированием  дан-
ных");
              else strcat(str_attr,"file data should be journaled");
              str_count++;break;
            case 15: symb_attr[i]='t';
              if(str_count!=0) strcat(str_attr,", ");
              if (loc==1) strcat(str_attr,"без склеивания хвостов");
              else strcat(str_attr,"file tail should not be merged");
              str_count++;break;
            case 18: symb_attr[i]='-';
              if(str_count!=0) strcat(str_attr,", ");
              if (loc==1) strcat(str_attr,"гигантский файл");
              else strcat(str_attr,"huge file");
              str_count++;break;
            case 19: symb_attr[i]='e';
              if(str_count!=0) strcat(str_attr,", ");
              if (loc==1) strcat(str_attr,"экстенты");
              else strcat(str_attr,"extents");
              str_count++;break;
            default: symb_attr[i]='-';
           }
           else symb_attr[i]='-';
         }
      symb_attr[20]='\0';
      if (loc==1) printf("Расширенные атрибуты:\t%s\n",symb_attr);
      else printf("Extended attributes:\t%s\n",symb_attr);
      printf("              (");
      if   (str_attr[0]=='\0')   {if   (loc==1)   printf("нет");   else
printf("no");}
      else printf("%s",str_attr);
      printf(")\n");
      }
```

```
     /*Если в файле используется адресация блоками, функция выводит
      *номера блоков данных */
     void   block_info(FILE   *disk,unsigned   int   offsets[2],int
block_size)
     {
       int i;
       unsigned int block_number;
       int loc=get_local();
       if (loc==1) printf("Блоки данных: ");
       else printf("Blocks: ");
       for (i=0;i<15;i++)
       {
         block_number=read_4(disk,offsets[0],block_size,   \\   off-
sets[1]+40+4*i);
         if (block_number==0)
           { if(i==0) {if (loc==1) printf("\t\tНет");
             else printf("\t\tNo");} break;}
         else printf("\n   N %d: %d",i+1,block_number);
       }
       printf("\n");
     }
     //Функция выводит на экран информацию о файле
     void print_inode(FILE *disk,unsigned int offsets[2], \\
     int block_size)
     {
       int uid,gid,links;
       unsigned int file_size,sectors;
       int flags,flag_mas[8];
       FILE *passwd,*group;
       char username[60],groupname[60],hex_flags[8],flag4[4];
       int loc=get_local();
       //Временные метки файла
       time_marks(disk,offsets,block_size);
       //Тип файла и права доступа к нему
       file_type(disk,offsets,block_size);
       //Идентификатор владельца
       uid=read_2(disk,offsets[0],block_size,offsets[1]+2);
       passwd=fopen("/etc/passwd","r");
       if (passwd==0) printf("Error opening file passwd\n");
       ugid(passwd,uid,username);
```

```
fclose(passwd);
if (username[0]!='0')
{ if (loc==1) printf("Владелец:\t\t%s\n",username);
  else printf("User:\t\t\t%s\n",username);}
else {if (loc==1) printf("Владелец:\t\t%d\n",uid);
  else printf("User ID:\t\t\t%d\n",uid);}
//Идентификатор группы
gid=read_2(disk,offsets[0],block_size,offsets[1]+24);
group=fopen("/etc/group","r");
if (group==0) printf("Error opening file group\n");
ugid(group,gid,groupname);
fclose(group);
if (groupname[0]!='0')
{ if (loc==1) printf("Группа владельца:\t%s\n",groupname);
  else printf("Group:\t\t\t%s\n",groupname);}
else {if (loc==1) printf("Группа владельца:\t%d\n",gid);
  else printf("Group ID:\t\t\t%s\n",groupname);}
//Размер файла
file_size=read_4(disk,offsets[0],block_size,offsets[1]+4);
if (loc==1) printf("Размер файла:\t\t%d байт\n",file_size);
else printf("File size:\t\t%d bytes\n",file_size);
//Число жестких ссылок на файл
links=read_2(disk,offsets[0],block_size,offsets[1]+26);
if (loc==1) printf("Число связей:\t\t%d\n",links);
else printf("Links:\t\t\t%d\n",links);
//Число секторов по 512 байт, занимаемых файлом
sectors=read_4(disk,offsets[0],block_size,offsets[1]+28);
if (loc==1) printf("Число секторов:\t\t%d\n",sectors);
else printf("Sectors:\t\t%d\n",sectors);
//Флаги файла
flags=read_4(disk,offsets[0],block_size,offsets[1]+32);
hexing(flags,hex_flags,flag_mas);
if (loc==1) printf("Флаги файла:\t\t0x%s\n",hex_flags);
else printf("File flags:\t\t0x%s\n",hex_flags);
//Расширенные атрибуты файла
lsattr (flag_mas);
//Способ адресации
binaring(flag_mas[3],flag4);
if (loc==1) printf("Способ адресации:\t");
else printf("Way of mapping:\t\t");
```

```
        if (flag4[0]=='1')
        {
          {
            if (loc==1) printf("Экстенты\n");
            else printf("Extents\n");
          }
          exttree(disk,offsets,block_size,0);
        }
        else
        {
          {
            if (loc==1) printf("Поблочно\n");
            else printf("Blocks\n");
          }
          block_info(disk,offsets,block_size);
        }
      }
      int inodeinfo(unsigned int inode, FILE *disk)
      {
        int magic,block_size;
        unsigned int max_inode;
        unsigned int offsets[2];
        int loc=get_local();
        //Определяем, является ли файловая система ext-системой
        magic=read_2(disk,1,1024,56); //1024+56
        if (magic!=61267)
        {
          if (loc==1)
            printf("Файл устройства не содержит магического чис-
ла\n");
          else printf("File does not contain magic number\n");
          return(1);
        }
        //Вычисляем размер блока
        block_size=get_block_size(disk);
        if (block_size==0)
        {
          {if (loc==1) printf("Ошибка чтения размера блока дан-
ных\n");
           else printf("Error in block size reading\n");} return (2);
```

```
    }
  /*Вычисляем количество индексных дескрипторов в файловой
   *системе */
  max_inode=read_4(disk,1,1024,0);
  if (inode<1||inode>max_inode)
  {
    if (loc==1)
      printf("Вы ввели несуществующий индексный дескриптор\n");
    else printf("Such inode does not exist\n");
    return(3);
  }
  /*Вычисляем номер блока и смещение в блоке, по которому
   *находится индексный дескриптор */
  inode_offset(disk,inode,block_size,offsets);
  if (loc==1)
    printf("Индексный дескриптор %d находится в блоке: %d ",\\
inode,offsets[0]);
  else printf("Inode %d is in block: %d ",inode,offsets[0]);
  if (loc==1) printf("по смещению: %d байт\n",offsets[1]);
  else printf("offset is: %d bytes\n",offsets[1]);
  //Выводим информацию о файле
  print_inode(disk,offsets,block_size);
  return (0);
}
```

## Модуль super

Файл super.h

```
int super(FILE *disk);
```

Файл super.c

```
#include <stdio.h>
#include <time.h>
#include "super.h"
#include "option.h"
int super (FILE *disk)
{
  int magic;
  int loc=get_local();
  int first_block,block_size,fragment_size;
  unsigned int inode_count, block_count, free_blocks;
```

```
       unsigned int free_inodes, su_blocks;
       int   frag_indic,   blopergro,   frapergro,   inopergro,
mount_count;
       unsigned int max_mount,os_type;
       unsigned int cur_flags,err_flags;
       int fs_version,def_uid,def_gid,first_inode,inode_size;
       unsigned                                              int
compat,incompat,rocompat,jor_inode,first_meta;
       unsigned int inode_min,inode_wanted,flex_size;
       unsigned int last_mount, last_write, last_check;
       unsigned int check_int,create;
       char
hex[8],hex_compat[8],hex_incompat[8],hex_rocompat[8];
       int del[8],i;
       time_t mount_time,write_time,check_time,create_time;
       struct tm *times;
       magic=read_2(disk,1,1024,56);
       if (magic!=61267)
       {
         if (loc==1)
           printf("Файл устройства не содержит магического чис-
ла\n");
         else printf("File does not contain magic number\n");
         return(1);
       }
       if  (loc==1)  printf("\tИнформация  о  файловой  систе-
ме:\n");
       else printf("\tFilesystem information:\n");
       inode_count=read_4(disk,1,1024,0);
     if        (loc==1)        printf("Число        индексных
дескрипторов:\t\t%d\n",inode_count);
       else printf("Inode count:\t\t\t%d\n",inode_count);
       block_count=read_4(disk,1,1024,4);
       if (loc==1)
        printf("Число         блоков         в         файловой
системе:\t%d\n",block_count);
       else printf("Block count:\t\t\t%d\n",block_count);
       su_blocks=read_4(disk,1,1024,8);
       if (loc==1)
       printf("Число                 блоков                 для
```

```
суперпользователя:\t%d\n",su_blocks);
      else          printf("Block          count          for
superuser:\t%d\n",su_blocks);
      free_blocks=read_4(disk,1,1024,12);
      if (loc==1)
        printf("Число                                свободных
блоков:\t\t\t%d\n",free_blocks);
      else printf("Free block count:\t\t%d\n",free_blocks);
      free_inodes=read_4(disk,1,1024,16);
      if (loc==1)
      printf("Число свободных индексных дескрипторов:\t%d\n",
\\ free_inodes);
      else printf("Free inodes count:\t\t%d\n",free_inodes);
      first_block=read_4(disk,1,1024,20);
      if (loc==1)
       printf("Номер         первого         блока         с
данными:\t\t%d\n",first_block);
      else     printf("First     block     containing     da-
ta:\t%d\n",first_block);
      block_size=get_block_size(disk);
      if (loc==1)
        printf("Размер блока данных:\t\t\t%d\n",block_size);
      else printf("Logical block size:\t\t%d\n",block_size);
      frag_indic=read_4(disk,1,1024,28);
      switch(frag_indic)
      {
        case 0: fragment_size=1024; break;
        case 1: fragment_size=2048; break;
        case 2: fragment_size=4096; break;
        default: fragment_size=0;
      }
      if (loc==1)
        printf("Размер фрагмента:\t\t\t%d\n",fragment_size);
      else printf("Fragment_size:\t\t\t%d\n",fragment_size);
      blopergro=read_4(disk,1,1024,32);
      if          (loc==1)          printf("Блоков          в
группе:\t\t\t%d\n",blopergro);
      else printf("Blocks per group:\t\t%d\n",blopergro);
      frapergro=read_4(disk,1,1024,36);
      if (loc==1)
```

```
        printf("Фрагментов в группе:\t\t\t%d\n",frapergro);
      else printf("Fragments per group:\t\t%d\n",frapergro);
      inopergro=read_4(disk,1,1024,40);
      if (loc==1)
        printf("Индексных дескрипторов в
группе:\t%d\n",inopergro);
      else printf("Inodes per group:\t\t%d\n",inopergro);
      create=read_4(disk,1,1024,264);
      create_time=create;
      times=localtime(&create_time);
      if (loc==1) printf("Время создания:\t\t\t\t");
      else printf("Creation time:\t\t\t");
      fputs(asctime(times),stdout);
      last_mount=read_4(disk,1,1024,44);
      mount_time=last_mount;
      times=localtime(&mount_time);
      if (loc==1) printf("Последнее монтирование:\t\t\t");
      else printf("Last mounted:\t\t\t");
      fputs(asctime(times),stdout);
      last_write=read_4(disk,1,1024,48);
      write_time=last_write;
      times=localtime(&write_time);
      if (loc==1) printf("Время последней записи:\t\t\t");
      else printf("Last write time:\t\t");
      fputs(asctime(times),stdout);
      last_check=read_4(disk,1,1024,64);
      check_time=last_check;
      times=localtime(&check_time);
      if (loc==1) printf("Время последней проверки ФС:\t\t");
      else printf("Last check time:\t\t");
      fputs(asctime(times),stdout);
      check_int=read_4(disk,1,1024,68);
      if (loc==1)
        printf("Максимальный интервал между проверка-
ми:\t%d\n", \ check_int);
      else printf("Maximum check inter-
val:\t\t%d\n",check_int);
      mount_count=read_2(disk,1,1024,52);
      if (loc==1)
        printf("Число монтирований:\t\t\t%d\n",mount_count);
```

```
       else printf("Mount count:\t\t\t%d\n",mount_count);
       max_mount=read_2(disk,1,1024,54);
       if (loc==1)
         printf("Предельное                                    число
монтирований:\t\t%d\n",max_mount);
       else printf("Maximum mount count:\t\t%d\n",max_mount);
       hexing(magic,hex,del);
       if (loc==1) printf("Магическое число:\t\t\t0x");
       else printf("Magic number:\t\t\t0x");
       for (i=4;i<8;i++) printf("%c",hex[i]);
       printf("\n");
       cur_flags=read_2(disk,1,1024,58);
       hexing(cur_flags,hex,del);
       if (loc==1) printf("Флаги текущего состояния:\t\t0x");
       else printf("Current flags:\t\t\t0x");
       for (i=4;i<8;i++) printf("%c",hex[i]);
       printf("\n");
       err_flags=read_2(disk,1,1024,60);
       hexing(err_flags,hex,del);
       if (loc==1) printf("Флаги обработки ошибок:\t\t\t0x");
       else printf("Error flags:\t\t\t0x");
       for (i=4;i<8;i++) printf("%c",hex[i]);
       printf("\n");
       os_type=read_4(disk,1,1024,72);
       if (loc==1) printf("Тип ОС:\t\t\t\t\t");
       else printf("OS type:\t\t\t");
       if (os_type==0) printf("Linux\n");
       else printf("%d\n",os_type);
       fs_version=read_4(disk,1,1024,76);
       if (loc==1)
         printf("Версия файловой системы:\t\t%d\n",fs_version);
       else printf("Filesystem version:\t\t%d\n",fs_version);
       def_uid=read_2(disk,1,1024,80);
       if           (loc==1)           printf("UID           по
умолчанию:\t\t\t%d\n",def_uid);
       else printf("Default UID:\t\t\t%d\n",def_uid);
       def_gid=read_2(disk,1,1024,82);
       if           (loc==1)           printf("GID           по
умолчанию:\t\t\t%d\n",def_gid);
       else printf("Default GID:\t\t\t%d\n",def_gid);
```

```
       first_inode=read_4(disk,1,1024,84);
       if (loc==1)
         printf("Первый                                    индексный
дескриптор:\t\t%d\n",first_inode);
       else              printf("First              non-reserved
inode:\t%d\n",first_inode);
       inode_size=read_2(disk,1,1024,88);
       if (loc==1)
         printf("Размер                                    индексного
дескриптора:\t\t%d\n",inode_size);
       else printf("Inode size:\t\t\t%d\n",inode_size);
       compat=read_4(disk,1,1024,92);
       hexing(compat,hex_compat,del);
       if (loc==1) printf("Флаги COMPAT:\t\t\t\t0x");
       else printf("COMPAT flags:\t\t\t0x");
       for(i=0;i<8;i++) printf("%c",hex_compat[i]);
       printf("\n");
       incompat=read_4(disk,1,1024,96);
       hexing(incompat,hex_incompat,del);
       if (loc==1) printf("Флаги INCOMPAT:\t\t\t\t0x");
       else printf("INCOMPAT flags:\t\t\t0x");
       for (i=0;i<8;i++) printf("%c",hex_incompat[i]);
       printf("\n");
       rocompat=read_4(disk,1,1024,100);
       hexing(rocompat,hex_rocompat,del);
       if (loc==1) printf("Флаги ROCOMPAT:\t\t\t\t0x");
       else printf("ROCOMPAT flags:\t\t\t0x");
       for(i=0;i<8;i++) printf("%c",hex_rocompat[i]);
       printf("\n");
       jor_inode=read_4(disk,1,1024,224);
       if (loc==1)
         printf("Индексный                                 дескриптор
журнала:\t\t%d\n",jor_inode);
       else printf("Journal inode:\t\t\t%d\n",jor_inode);
       first_meta=read_4(disk,1,1024,260);
       if (loc==1)
         printf("Первая                                    метагруппа
блоков:\t\t%d\n",first_meta);
       else printf("First metagroup:\t\t%d\n",first_meta);
       inode_min=read_2(disk,1,1024,348);
```

```
       if (loc==1)
         printf("Минимальный          размер          дескрипто-
ра:\t\t%d\n",inode_min);
       else printf("Inode extra isize:\t\t%d\n",inode_min);
       inode_wanted=read_2(disk,1,1024,350);
       if (loc==1)
         printf("Желаемый                                  размер
дескриптора:\t\t%d\n",inode_wanted);
       else printf("Inode wanted isize:\t\t%d\n",inode_wanted);
       fseek(disk,1396,SEEK_SET);
       flex_size=fgetc(disk);
       if (loc==1)
         printf("Размер              гибкой              группы
блоков:\t\t%d\n",flex_size);
       else printf("Flex block group size:\t\t%d\n",flex_size);
       return (0);
     }
```

**ПРИЛОЖЕНИЕ 5**

## ПРИМЕРЫ РАБОТЫ С ПРОГРАММОЙ EXTVIEW

Ниже представлены листинги с примерами работы программы. **Extview** имеет в своем составе различные опции, состав которых может быть расширен в дальнейшем.

Справка по программе выводится командой **extview --help**. На рис. П.5.1 показан вывод этой команды.

```
Extview - программа для просмотра файловых систем ext2/3/4.
Для выполнения программы нужно право на чтение диска или образа диска.
Может выполняться как на монтированной системе, так и на размонтированной.
Открывает файл устройства только для чтения, ничего не изменяя на диске.
Использование (опции):
-i <inode_number> <device>
      Выводит информацию об индексном дескрипторе с номером <inode_number>
      <device> - раздел диска или образ файла блочного устройства
      Например: extview -i 2 /dev/sda1
-s <device>
      Выводит информацию о файловой системе на устройстве <device>
      Например: extview -s /dev/sda4
-g <device>
      Выводит информацию о группах блоков в файловой системе
      на устройстве <device>
      Например: extview -g /dev/hda5
-b <block_number> <device>
      Выводит шестнадцатеричное представление блока с номером <block_number>
      с устройства <device>
      Например: extview -b 997 /dev/sda5
-d <dir_inode/dir_block> <device> [B]
      Выводит элементы директории, указанной по индексному дескриптору, в том
      числе удаленные и скрытые файлы. Если указана необязательная опция B,
      интерпретирует входной параметр как номер блока и пытается прочесть блок
      данных как каталог.
      Например: extview -d 2 /dev/hda4 - выводит корневой каталог
-j <journal_block_number> <device> [B]
      Выводит информацию об указанном блоке из журнала устройства <device>
      Если указать опцию B, выводит еще и сам блок данных журнала
      Например: extview -j 300 /dev/sda5 B
```

```
-l [b/i] [<inode_number/block_number>] <device>
      Выводит список блоков журнала, в которых есть упоминание об указанном
      индексном дескрипторе. Если указана необязательная опция b, то выводит
      список журнальных блоков с этим блоком. Если перед номером индексного
      дескриптора опций не указано, то считает введенный номер номером
      индексного дескриптора
      При указании только имени устройства выводит список удаленных индексных
      дескрипторов на этом устройстве
      Например: extview -l i 161121 /dev/sda3
-c <device>
      Поиск ключевой фразы на устройстве <device>. При запуске требуется
      ввести фразу для поиска и диапазон блоков, в котором следует вести поиск
      Выдает номера блоков данных, где была найдена ключевая фраза.
      Например: extview -c /dev/sda2--inodeblock <inode_number> <device>
      Выводит шестнадцатеричное представление индексного
      дескриптора <inode_number> с устройства <device>
      Например: extview --inodeblock 2 /dev/hdb3
-h, --help
      Выводит справку о программе
```

Рис. П.5.1. Справочная информация

Индексный дескриптор файла можно вывести с помощью опции **–i**. На рис. П.5.2 показан вывод информации об индексном дескрипторе для небольшого файла, представленного двумя экстентами. Команда **extview –i 131634 /dev/sda5**.

```
Индексный дескриптор 131634 находится в блоке: 524371 по смещению: 256 байт
Последнее обращение:    Sun Dec 27 12:41:08 2009
Создан:                 Sun Dec 27 12:41:08 2009
Изменен:                Tue Dec 19 05:33:13 2006
Удален:                 Не удалялся
Тип файла:              Обычный файл
Права доступа:          -rwxrwxrwx
Владелец:               madcat
Группа владельца:       madcat
Размер файла:           4540213 байт
Число связей:           1
Число секторов:         8872
Флаги файла:            0x00080000
Расширенные атрибуты:   ------------------e
```

```
                        (экстенты)
Способ адресации: Экстенты
1-й экстент: первый блок: 1231872 (0x0012CC00), длина: 1024
2-й экстент: первый блок: 1275904 (0x00137800), длина: 85
```

Рис. П.5.2. Вывод информации об индексном дескрипторе среднего файла

Для файла с количеством экстентов, большим, чем 4, программа выводит дерево экстентов полностью. Пример представлен на рис. П.5.3, команда **`extview -i 161121 /dev/sda5`**.

```
Индексный дескриптор 161121 находится в блоке: 526214 по смещению: 0 байт
Последнее обращение:    Sun Feb 14 11:41:56 2010
Создан:                 Fri Nov 27 21:11:59 2009
Изменен:                Fri Nov 27 21:11:59 2009
Удален:                 Не удалялся
Тип файла:              Обычный файл
Права доступа:          -rw-r--r--
Владелец:               madcat
Группа владельца:       madcat
Размер файла:           777038562 байт
Число связей:           1
Число секторов:         1517664
Флаги файла:            0x00080000
Расширенные атрибуты:   ------------------e
                        (экстенты)
Способ адресации: Экстенты
Следующий узел дерева в блоке: 786607 (0x000C00AF)
      1-й экстент: первый блок: 804864 (0x000C4800), длина: 2048
      2-й экстент: первый блок: 813056 (0x000C6800), длина: 4096
      3-й экстент: первый блок: 894976 (0x000DA800), длина: 30720
      4-й экстент: первый блок: 925696 (0x000E2000), длина: 21504
      5-й экстент: первый блок: 947200 (0x000E7400), длина: 29696
      6-й экстент: первый блок: 976896 (0x000EE800), длина: 13076
      7-й экстент: первый блок: 989972 (0x000F1B14), длина: 32736
      8-й экстент: первый блок: 1022708 (0x000F9AF4), длина: 25868
      9-й экстент: первый блок: 1114112 (0x00110000), длина: 29963
```

Рис. П.5.3. Вывод информации об индексном дескрипторе большого файла

Вывод индексного дескриптора в шестнадцатеричном виде представлен на рис. П.5.4. Команда

**extview --inodeblock 131634 /dev/sda5**

```
Индексный дескриптор 131634 лежит в блоке 524371 по смещению 256

0x00000000  FF 81 E8 03 35 47 45 00 : 94 0F 37 4B 94 0F 37 4B  ....5GE...7K..7K
0x00000010  49 33 87 45 00 00 00 00 : E8 03 01 00 A8 22 00 00  I3.E.........".."
0x00000020  00 00 08 00 01 00 00 00 : 0A F3 02 00 04 00 00 00  ................
0x00000030  00 00 00 00 00 00 00 00 : 00 04 00 00 00 CC 12 00  ................
0x00000040  00 04 00 00 55 00 00 00 : 00 78 13 00 00 00 00 00  ....U....x......
0x00000050  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x00000060  00 00 00 00 80 87 68 6A : 00 00 00 00 00 00 00 00  ......hj........
0x00000070  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x00000080  1C 00 00 00 60 CA CD 61 : 00 00 00 00 84 BD D6 9E  ....`..a........
0x00000090  94 0F 37 4B 38 0C FB 34 : 00 00 00 00 00 00 00 00  ..7K8..4........
0x000000A0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000000B0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000000C0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000000D0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000000E0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
0x000000F0  00 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
```

Рис. П.5.4. Вывод шестнадцатеричного представления индексного дескриптора

Вывод информации о файловой системе в целом представлен на рис. П.5.5. Команда **extview –s /dev/sda5**.

```
      Информация о файловой системе:
Число индексных дескрипторов:       605024
Число блоков в файловой системе:    2415766
Число блоков для суперпользователя: 120788
Число свободных блоков:             1247814
Число свободных индексных дескрипторов:   378130
Номер первого блока с данными:            0
Размер блока данных:                4096
Размер фрагмента:                   4096
Блоков в группе:                    32768
Фрагментов в группе:                32768
```

```
Индексных дескрипторов в группе:    8176
Время создания:                     Mon Nov  2 16:24:05 2009
Последнее монтирование:             Mon Feb 22 22:29:25 2010
Время последней записи:             Mon Feb 15 18:17:16 2010
Время последней проверки ФС:        Sat Feb 13 18:03:59 2010
Число монтирований:                 18
Предельное число монтирований:      29
Магическое число:                   0xEF53
Флаги текущего состояния:           0x0001
Флаги обработки ошибок:             0x0001
UID по умолчанию:                   0
GID по умолчанию:                   0
Первый индексный дескриптор:        11
Размер индексного дескриптора:      256
Флаги COMPAT:                       0x0000003C
Флаги INCOMPAT:                     0x00000246
Флаги ROCOMPAT:                     0x0000007B
Индексный дескриптор журнала:       8
Размер гибкой группы блоков:        4
```

Рис. П.5.5. Вывод информации о файловой системе

На рис. П.5.6 представлен вывод программой информации о журнальном суперблоке, команда **`extview –j 0 /dev/sda5`**

```
Первый блок журнала:          1081344
Номер последовательности:     0
Тип блока:                    Журнальный суперблок v.2
Размер блоков журнала:        4096
Число блоков в журнале:       32768
Первый блок с информацией:    1
Первый ожидаемый ID:          91310
Блок начала журнала:          1
Код ошибки:                   0
Флаги COMPAT:                 0x00000000
Флаги INCOMPAT:               0x00000001
Флаги ROCOMPAT:               0x00000000
Журнальный UUID:              AACA0A90-3A1241B8-9F3DC82B-1B48680C
Число ФС для этого журнала:   1
Копия суперблока в блоке:     0
```

Рис. П.5.6. Вывод информации о журнальном суперблоке

Вывод журнального блока, который не является дескриптором, представлен на рис. П.5.7. В команду добавлена опция «B», благодаря чему выведен весь блок данных, а не только информация о предыдущем и следующем блоках-дескрипторах. Команда **extview –j 1805 /dev/sda5 B**

```
Первый блок журнала:       1081344
Тип блока:                 Обычный блок
  Предыдущий дескриптор:          Следующий дескриптор:
  Блок 1800                       Блок 1810

0x0070D000  A1 1F 02 00 0C 00 01 02 : 2E 00 00 00 A0 1F 02 00  ................
0x0070D010  0C 00 02 02 2E 2E 00 00 : BA 16 02 00 18 00 0C 01  ................
0x0070D020  73 6D 62 2D 6E 65 74 77 : 6F 72 6B 3A 51 32 55 55  smb-network:Q2UU
0x0070D030  FD 01 02 00 14 00 09 01 : 63 6F 6D 70 75 74 65 72  ........computer
```

Рис. П.5.7. Фрагмент вывода журнального блока

Вывод информации о группах блоков представлен на рис. П.5.8. Команда **extview –g /dev/sda5**

```
Число групп блоков: 74
Группа 0:   карта блоков: 591, карта дескрипторов: 607,
            таблица индексных дескрипторов: 623,
            свободных блоков: 3436, свободных дескрипторов: 0,
            директорий: 4, неиспользованных дескрипторов: 0.
            Контрольная сумма: 0x3462

Группа 1:   карта блоков: 592, карта дескрипторов: 608,
            таблица индексных дескрипторов: 1134,
            свободных блоков: 4107, свободных дескрипторов: 32,
            директорий: 753, неиспользованных дескрипторов: 0.
            Контрольная сумма: 0x9185

и т. д.
```

Рис. П.5.8. Фрагмент вывода информации о группах блоков

На рис. П.5.9 представлен вывод программой указанного блока данных. Команда **extview –b 590848 /dev/sda5**. В этом блоке содержится фрагмент HTML-файла.

```
0x90400000  3C 21 44 4F 43 54 59 50 : 45 20 48 54 4D 4C 20 50  <!DOCTYPE HTML P
0x90400010  55 42 4C 49 43 20 22 2D : 2F 2F 57 33 43 2F 2F 44  UBLIC "-//W3C//D
0x90400020  54 44 20 48 54 4D 4C 20 : 34 2E 30 31 20 54 72 61  TD HTML 4.01 Tra
0x90400030  6E 73 69 74 69 6F 6E 61 : 6C 2F 2F 45 4E 22 3E 0A  nsitional//EN">.
0x90400040  0A 3C 68 74 6D 6C 3E 0A : 20 20 3C 68 65 61 64 3E  .<html>.  <head>
0x90400050  0A 20 20 20 20 3C 6D 65 : 74 61 0A 20 20 20 20 20  .    <meta.
0x90400060  6E 61 6D 65 3D 22 67 65 : 6E 65 72 61 74 6F 72 22  name="generator"
0x90400070  0A 20 20 20 20 20 63 6F : 6E 74 65 6E 74 3D 0A 20  .     content=.
0x90400080  20 20 20 22 48 54 4D 4C : 20 54 69 64 79 20 66 6F     "HTML Tidy fo
0x90400090  72 20 4C 69 6E 75 78 2F : 78 38 36 20 28 76 65 72  r Linux/x86 (ver
0x904000A0  73 20 31 73 74 20 4A 75 : 6C 79 20 32 30 30 32 29  s 1st July 2002)
```

Рис. П.5.9. Фрагмент шестнадцатеричного представления блока данных

Вывод директорий по индексному дескриптору приведен на рис. П.5.10, в том числе указываются удаленные и скрытые файлы. Команда **extview –d 131360 /dev/sda5**

```
Первый блок директории: 532637
      131360, . (dir)
      287, .. (dir)
      136095, apelsin.jpg
      139264, Снимки (dir)
      138745, Шихан.jpg
      172731, 461920.jpg
      138306, 459436.jpg
      138690, Невьянск.jpg
      138862, Верхотурье.jpg
      172743, dir.png
      172771, joursup.png
      137613, lost.png
      171414, lde-super.png
      160758, before.png - удален
      160892, after.png
      161424, dirview.png - удален
```

```
      160807, journal.png
      160809, find.png - удален
      161388, helpp.png
      163184, tree.png
      172812, jourblock.png
```

Рис. П.5.10. Вывод записей в директории

На рис. П.5.11 представлен вывод программой списка удаленных индексных дескрипторов. Перед выполнением поиска программа запрашивает время, после которого следует искать дескрипторы. Команда **`extview –l /dev/sda5`**

```
Введите время удаления, после которого искать файлы
1266850000
Mon Feb 22 19:46:40 2010
Список удаленных индексных дескрипторов:
1     178952, удален Mon Feb 22 23:14:42 2010
2     179092, удален Mon Feb 22 23:14:42 2010
3     237127, удален Mon Feb 22 23:14:42 2010
4     237133, удален Mon Feb 22 23:14:42 2010
Файлы, удаленные после: (1266850000) Mon Feb 22 19:46:40 2010
```

Рис. П.5.11. Список недавно удаленных файлов

**ПРИЛОЖЕНИЕ 6**

# ТЕСТОВЫЕ ВОПРОСЫ ДЛЯ ПРОГРАММИРОВАННОГО КОНТРОЛЯ ЗНАНИЙ

## Файловые операции и права доступа

1. Что означает право на исполнение каталога?
- Возможность запускать из него файлы программ.
- Право войти в каталог.
- Право скопировать в него файл.
- Возможность изменить имеющиеся в нем файлы.
- Право удаления имеющихся в нем файлов.

2. Какие права на родительский каталог необходимы для создания в нем подкаталога?
- Для этого никаких прав не требуется.
- Право владения каталогом.
- Право чтения и поиска в каталоге.
- Право записи и поиска в каталоге.
- Все основные права.
- Право поиска в каталоге.

3. Какие права на файл необходимы для перезаписи его индексного дескриптора?
- Для этого никаких прав не требуется.
- Право владения.
- Право чтения.
- Право записи.
- Право исполнения.
- Все основные права.

4. Администратор обнаружил нарушение, вызванное недозволенной пользовательской активностью, и решил на время «заблокировать» пользователей в их домашних каталогах. Для этого он установил на каталог **/home** права доступа **rwx------**. Каким будет результат?
- Система не позволит администратору выполнить такую команду в отношении каталога **/home.**
- Пользователи не смогут работать, поскольку все утилиты для них окажутся недосягаемы.
- Пользователи окажутся заблокированными в своих каталогах, но смогут продолжать работу.

- Установленный «барьер» пользователи могут преодолеть. Так, команда **`cd ..`** будет запрещена, но команду **`cd /`** и все иные команды выполнить можно. Однако пользователи, покинувшие свои каталоги, не смогут вернуться обратно и получить доступ к своим файлам.
- В результате все зарегистрированные пользователи автоматически будут перемещены в корневой каталог, будут иметь доступ к программам, однако не сумеют получить доступ к своим файлам.
- Команда будет выполнена, однако первая же пользовательская команда приведет к «зависанию» системы.

5. Какие минимальные права нужно иметь для создания жесткой ссылки на файл?
- Право на чтение файла и вход в содержащий его каталог.
- Право на запись в файл и запись в содержащий его каталог.
- Никаких прав не требуется.
- Только право на вход и запись в каталог, где содержится объектовый файл.
- Право на запись в файл и вход в каталог, где содержится объектовый файл.
- Только право на запись в каталог, в котором создается жесткая ссылка.
- Право на вход и чтение каталога, в котором хранится прежнее имя, и право на вход и запись в каталог, в котором создается новое имя. Прав на сам файл не требуется.
- Право на вход и чтение каталога, в котором хранится прежнее имя, и право на вход и запись в каталог, в котором создается новое имя. На сам файл требуется право записи, чтобы изменить число ссылок в его индексном дескрипторе.
- Здесь нет правильного ответа.

6. Пользователь **`john`** установил жесткую ссылку из своего каталога на секретный файл пользователя **`braun`**. Файл имеет права доступа **`rw-- -----`** . Что произойдет, когда **`braun`** удалит свой файл?
- При удалении файла будут уничтожена установленная жесткая ссылка на него.
- **`John`** получит права на чтение файла.
- **`John`** станет единственным владельцем файла и сможет изменить права доступа к нему.
- Файл исчезнет из каталога **`braun`**, но останется в каталоге **`john`**. Владелец файла и права доступа к нему не изменятся.
- Файл не будет удален и сохранится в каталогах обоих пользователей.

7. Можно ли создать жесткую ссылку на каталог?

- Можно, но только администратору.
- Можно, но только администратору или владельцу каталога.
- Можно всем пользователям, имеющим право записи в этот каталог.
- Можно всем пользователям, имеющим право входа в этот каталог.
- Это не разрешено никому.

8. Администратор вводит следующую команду:
**`echo 'umask 022' >> /etc/skel/bash_profile`**. Для чего он это делает?
- Чтобы каждый вновь создаваемый пользователь унаследовал указанную маску доступа на создаваемые им файлы.
- Чтобы изменить маску доступа на создаваемые файлы для всех уже зарегистрированных пользователей.
- Чтобы изменить маску доступа на создаваемые файлы для всех уже зарегистрированных пользователей, включая и самого администратора.
- Чтобы изменить маску доступа на создаваемые файлы для себя лично.
- Маска доступа устанавливается не для обычных файлов, а для каталогов.

9. Какие из приведенных команд не предназначены для создания нового файла (предполагается, что файл **`abcd`** к этому времени еще не существует)?
- **`echo 1 > abcd`**
- **`touch abcd`**
- **`stat abcd`**
- **`cat abcd`**
- **`dd of=abcd`**
- **`renice abcd`**

10. Вводя команду **`umask 022`**, администратор тем самым:
- Изменяет права доступа для всех файлов, находящихся в домашних каталогах пользователей, на **`rwxr--r--`** .
- Устанавливает права доступа **`rwxr--r--`** для всех файлов, которые будут создаваться пользователями в их домашних каталогах.
- Устанавливает права доступа **`rwxr--r--`** для всех файлов, которые будут создаваться лично администратором.
- Устанавливает права доступа **`rwxr--r--`** для всех файлов, которые будут создаваться в автоматическом виде операционной системой.
- Изменяет на текущий сеанс права доступа к программам, располагаемым в каталогах **`/bin`** и **`/sbin`**.

11. Какие права доступа получает пользователь на скопированный им файл?
- Это определяется параметрами, заданными в команде копирования.
- Они задаются маской доступа, установленной для данного пользователя.
- Копия наследует все права доступа файла-оригинала.
- Они определяются переменной окружения процесса, производящего копирование.
- Здесь нет правильного ответа.

12. Какие минимальные права нужно иметь для создания символической ссылки на файл?
- Право на чтение объектового файла и вход в каталог.
- Право на запись в каталог, в котором хранится объектовый файл.
- Только право на запись в каталог, в котором создается символическая ссылка.
- Никаких прав иметь не требуется.
- Право владения объектовым файлом.

13. Пользователь создал с помощью команды **`ln –s link_1 abcd`** символическую ссылку на файл **`abcd`**. Но оказалось, что такого файла в текущем каталоге нет (пользователь забыл, что такой файл хранится в другом каталоге). Что произойдет при записи данных в созданную символическую ссылку например, путем перенаправления вывода **`cal 2009 > link_1`** ?
- Система сообщит пользователю о том, что искомый файл не найден.
- Система сообщит пользователю об ошибке.
- Объектовый файл будет создан и заполнен записываемыми данными.
- Такой ситуации не может быть, т. к. система не позволит пользователю создать символическую ссылку на несуществующий файл.
- Объектовый файл будет создан, но окажется пустым.

14. Какие права необходимо иметь для переименования файла?
- Необходимо быть владельцем файла.
- Необходимо иметь право на чтение файла.
- Необходимо иметь право на запись в файл.
- Необходимо иметь право на вход и запись в каталог (так как изменяется только имя файла в соответствующей записи каталога).
- Здесь нет правильного ответа.

15. При переименовании файла:

- Происходит смена его владельца.
- Меняется его индексный дескриптор.
- Меняется время его последней модификации.
- Меняются его индексный дескриптор и блоки данных.
- Увеличивается на единицу число «жестких» ссылок на файл.

16. Какие права необходимо иметь для удаления файла?
- Право владения каталогом, в котором содержится удаляемый файл.
- Право поиска и записи в содержащем его каталоге.
- Право на запись в файл (т. к. удаление файла эквивалентно его стопроцентной модификации).
- Право владения файлом.
- Никаких прав не требуется.

17. Установка для каталога **`sticky`**-бита не оказывает действия:
- На администратора.
- На владельца каталога.
- На владельцев удаляемых файлов.
- На всех пользователей.
- На процедуру удаления файлов.

18. Можно ли командой **`chmod`** изменить права доступа на символическую ссылку?
- Можно, но только администратору.
- Можно, но только владельцу символической ссылки.
- Можно, но только владельцу целевого файла.
- Можно, если использовать соответствующий параметр в команде **`chmod`.**
- Разрешено любому пользователю, если это не превышает прав доступа на целевой файл.
- В этом нет смысла, поэтому система не позволяет это сделать никому.

19. Что произойдет, если администратор командой **`chmod`** изменит права доступа на любую из символических ссылок?
- Ничего не произойдет.
- Изменятся права доступа к целевому файлу, на который указывает символическая ссылка.
- Изменятся права доступа к символической ссылке.
- Система сообщит пользователю о недопустимости операции.
- Произойдет сбой в работе системы.

20. Что нужно сделать для того, чтобы можно было непосредственно запускать файл сценария?
- Откомпилировать его.
- Командой **chmod** присвоить файлу право на его исполнение.
- Командой **umask** изменить маску доступа.
- Записать в его заголовке строку **#! /bin/bash** и присвоить файлу право на исполнение.
- Сценарии положено запускать только в качестве аргумента после имени командного интерпретатора.
- Присвоить ему эффективное право доступа.

21. Какие минимальные права необходимы для копирования файла в другой каталог?
- Право чтения файла, чтения и входа в исходный каталог, записи и входа в целевой каталог.
- Право исполнения файла.
- Полные права на оба каталога, право на файл несущественно.
- Право исполнения файла, входа в исходный каталог, записи в целевой каталог.
- Право записи в файл, чтения из исходного каталога, записи и входа в целевой каталог.
- Право чтения файла, чтения из исходного каталога, записи в целевой каталог.
- Здесь нет правильного ответа.

22. Если известно имя копируемого файла, то на каталог, где файл хранится, достаточно иметь право поиска (или исполнения). Какие права на каталог надо иметь, чтобы скопировать из него все файлы, не зная их имени, с использованием символа-заменителя * ?
- Первое утверждение ложно. Для копирования любого количества файлов нужно иметь права чтения и исполнения на исходный каталог.
- При задании списка файлов с указанием символа * мы неявно даем указание системе прочитать имя каждого файла. Следовательно, на исходный каталог нужны права чтения и исполнения.
- Поскольку имен файлов знать не надо, то достаточно иметь право на поиск (исполнение) в каталоге.
- Право на поиск в каталоге автоматически означает возможность узнавать имена и атрибуты каждого файла. Для копирования всех файлов достаточно иметь право на исполнение каталога, в котором они хранятся.
- Здесь нет правильного ответа.

23. Что нужно сделать с файлом, чтобы система воспринимала и отображала его как скрытый?
- Установить с помощью программы-оболочки соответствующий атрибут файла.
- Поставить в качестве первого символа в имени файла точку.
- Поставить в качестве последнего символа в имени файла тильду (**~**).
- Вставить в произвольном месте имени файла символ подчеркивания.
- Изменить с помощью команды **stat** тип файла.

24. Пользователь имеет на чужой для него каталог права доступа **-w-** . Что он может делать?
- Входить в каталог и выходить из него с помощью команды **cd**.
- Создавать в каталоге файлы, не входя в него (например, командой **echo 12345 > /catalog1/file1**).
- Удалять из каталога любые файлы.
- Удалять из каталога свои файлы.
- Ему запрещен доступ в этот каталог и любые действия с находящимися там файлами.
- Создавать и удалять файлы в этом каталоге.
- Просматривать содержимое каталога командой **ls -l .**
- Здесь нет правильного ответа.

25. Пользователь имеет на чужой для него каталог права доступа **--x** . Что он может делать?
- Входить в каталог и выходить из него с помощью команды **cd**
- Создавать в каталоге файлы, не входя в него (например, командой **echo 12345 > /catalog1/file1**).
- Удалять из каталога любые файлы.
- Удалять из каталога свои файлы.
- Создавать и удалять любые файлы в этом каталоге.
- Просматривать содержимое каталога командой **ls -l**
- Ему запрещены любые действия с находящимися там файлами.
- Здесь нет правильного ответа.

26. Пользователь имеет на чужой для него каталог права доступа **r--** . Что он может делать?
- Входить в каталог и выходить из него с помощью команды **cd**
- Создавать в каталоге файлы, не входя в него (например, командой **echo 12345 > /catalog1/file1**).
- Удалять из каталога свои файлы.
- Создавать и удалять файлы.

- Просматривать содержимое каталога командой **ls –l**
- Просматривать содержимое каталога командой **ls**
- Ему запрещены любые действия с файлами.
- Здесь нет правильного ответа.

27. Администратор, находясь в своем каталоге **/root**, с помощью команды **su user1** «превращается» в одного из пользователей. Для обычных пользователей каталог администратора полностью закрыт. Сможет ли «бесправный» администратор выйти из недоступного каталога ?
- Сможет, ведь права доступа предусматривают запрет только на вход в каталог.
- Не сможет. При использовании команды **cd** последует сообщение о недостатке прав.
- Не сможет. Команду **cd** система в этом случае игнорирует.
- Сможет, так как система «помнит» предыдущий статус этого пользователя.
- Результат зависит от наличия или отсутствия дефиса перед именем пользователя в команде **su**

28. Как можно защитить файл от любых изменений со стороны всех пользователей, включая администратора?
- Поместить файл в один из системных каталогов.
- Отменить право на запись в файл для всех пользователей.
- Снять право на запись в каталог, где находится данный файл.
- Использовать дополнительный атрибут **+i**, вводя его с помощью команды **chattr**
- Защитить файл контрольной суммой с помощью команды **md5sum**

29. Что означает опция **nosuid**, установленная для сменного машинного носителя в файле **/etc/fstab** ?
- Предупреждение пользователям, чтобы они не запускали файлов с установленным битом **setuid** с этого носителя.
- Игнорирование ядром системы битов **setuid** в исполняемых файлах, размещенных на сменном носителе.
- Запрет на копирование специальных файлов типа **su, passwd, mount, umount** с жесткого диска на сменный носитель.
- Предупреждение пользователям об ограничении использования утилиты **chmod**.
- Здесь нет правильного ответа.

30. Пользователь с правами **root** ввел из консоли команду **chmod 055 chmod**. Что ему теперь следует сделать?

- Перезагрузить систему.
- Создать учетную запись нового пользователя, путем редактирования файла **/etc/passwd** присвоить ему **UID=0**, зарегистрироваться и с правами нового администратора вернуть себе утраченные права на файл **chmod**.
- Загрузить систему Linux со сменного носителя, примонтировать с правами записи раздел Linux на фиксированном диске, найти на нем файл **chmod** и восстановить прежние права его владельца.
- Создать резервную копию всех данных и переустановить систему.
- Ничего делать не надо. Администратору по умолчанию разрешен доступ к любому файлу, в том числе и к **chmod**.

31. Пользователь с правами **root** ввел из консоли команду **chmod –R 555 /** Что в результате произойдет?

- Ничего опасного не произойдет.
- Система остановится, так как всем процессам будет запрещено создавать, удалять и модифицировать файлы.
- Пользователи получат возможность читать и запускать файлы, к которым у них обычно нет доступа.
- Администратор лишится возможности создавать новые и удалять существующие файлы и каталоги.
- Будут уничтожены все учетные записи пользователей, включая администратора.

32. Пользователь с правами **root** ввел из консоли команду **chmod –R 555 /** Что ему теперь следует сделать?

- Выполнить команду **dd if=/home/* of=/dev/sdb1** и переустановить систему.
- Ввести команду **inetd 1**
- Ввести команду **telinit 0**
- Ввести команду **chmod –R 750 /**
- Написать заявление об увольнении.
- Загрузить компьютер операционной системой со сменного носителя (Live CD), примонтировать раздел жесткого диска с ОС Linux и с помощью команды **chmod** установить прежние права на файлы и каталоги (например, по образцу другой ОС).

33. Имеется текстовый файл **abcd** с правами доступа **000**, состоящий только из одной команды **cat /dev/zero > /dev/null**. Что произойдет, если администратор введет команду **abcd** ?

- Такая команда не выполнится, ведь файл не является программой.
- Такая команда не выполнится, так как на запуск файла не установлены права, в том числе и для **root**.
- Процесс перекачивания нулей «из пустого в порожнее» запустится, так как команда администратора – закон для системы.
- Автоматически откроется справка по командной оболочке.
- Произойдет сбой в консоли, из которой была запущена команда.

34. Пользователь изменил права доступа к принадлежащему ему файлу командой **chmod 5630 file1**. Какую информацию о правах доступа выведет команда **ls –l file1** ?

- **-rwS-wx--T**
- **d-wsr-s-wt**
- **-wt--sxrw-**
- **lrwsr-x-w-**
- **-rw—wx—t**

35. Пользователь изменил права доступа к принадлежащему ему файлу командой **chmod 4237 file2**. Какую информацию о правах доступа выведет команда **ls –l file2** ?

- **-r-t-wx--S**
- **d-wsr-s-wt**
- **--wS-wxrwx**
- **cr-sr-x-w-**
- **-rw—wx—t**

36. Требуется скопировать файл **./a/b/file1** в каталог **./c/d**. Указать минимально необходимые права на файл и каждый из каталогов, чтобы копирование прошло успешно. Записать булево выражение в виде

**cp = a(rwx) x b(rwx) x c(rwx) x d(rwx) x file1(rwx).**

В скобках указывается элементарное произведение или дизъюнктивная нормальная форма.

37. Требуется переместить файл **./a/b/file1** в каталог **./c/d**. Указать минимально необходимые права на файл и каждый из каталогов, чтобы перемещение прошло успешно. Записать булево выражение в виде

**mv = a(rwx) x b(rwx) x c(rwx) x d(rwx) x file1(rwx).**

38. Требуется удалить файл **./a/b/file1**. Указать минимально необходимые права на файл и каждый из каталогов, чтобы удаление прошло успешно. Записать булево выражение в виде

**rm = a(rwx) x b(rwx) x file1(rwx).**

39. Требуется дописать данные в файл **./a/b/file1**. Указать минимально необходимые права на файл и каждый из каталогов, чтобы модификация прошла успешно. Записать булево выражение в виде

**add = a(rwx) x b(rwx) x file1(rwx).**

40. Требуется переименовать файл **./a/b/file1**. Указать минимально необходимые права на файл и каждый из каталогов, чтобы переименование прошло успешно. Записать булево выражение в виде

**ren = a(rwx) x b(rwx) x c(rwx) x d(rwx) x file1(rwx).**

41. Требуется создать жесткую ссылку на файл **./a/b/file1** из каталога **./c/d**. Указать минимально необходимые права на файл и каждый из каталогов, чтобы создание ссылки прошло успешно. Записать булево выражение в виде

**ln = a(rwx) x b(rwx) x c(rwx) x d(rwx) x file1(rwx).**

42. Пользователи Петров и Борисов входят в одну группу. На файлы Борисова установлены права доступа **rwx---r-x**. Петров пытается скопировать их в свой каталог. Каким будет результат?

- Файлы скопируются, так как прав на чтение, установленных для пользователей, для копирования вполне достаточно.
- Вопрос поставлен некорректно, т. к. неизвестны права доступа пользователей к исходному и целевому каталогам.
- Последует отказ в доступе, так как у Петрова как члена группы нет никаких прав.
- Последует отказ в доступе, так как при копировании должна измениться временная отметка последнего доступа к исходному файлу, а для модификации его **inode** требуются права на запись в файл.
- Здесь нет правильного ответа.

43. Пользователи Джон и Браун являются членами групп с аналогичными именами. Пользователь Браун включен в дополнительную группу **John**. На файлы Джона установлены права доступа **rw-rw-r--**, а на каталог, где эти файлы хранятся, – **rwx--xr--**. Браун пытается скопировать их в свой каталог. Каким будет результат?

- Операция будет выполнена, так как предоставленных прав для перемещения файлов вполне достаточно.
- Операция будет выполнена, если Браун знает и явно укажет в команде имена копируемых файлов.

- Последует отказ в доступе, так как у Брауна как члена группы нет соответствующих прав.
- Последует отказ в доступе, так как при копировании должна измениться временная отметка последнего доступа к исходному файлу, а для модификации его **inode** требуются права на запись в каталог.
- Здесь нет правильного ответа.

44. Пользователи Джон и Браун являются членами групп с аналогичными именами. Пользователь Браун включен в дополнительную группу **John**. На файлы Джона установлены права доступа **rw-rw-r--**, на каталог, где эти файлы хранятся, – **rwx--xr--**. Браун пытается переместить их в свой каталог. Каким будет результат?
- Операция будет выполнена, так как предоставленных прав для перемещения файлов вполне достаточно.
- Операция будет выполнена, если Браун знает имена перемещаемых файлов.
- Последует отказ в доступе, так как у Брауна, как члена группы, нет соответствующих прав на каталог.
- Последует отказ в доступе, так как при перемещении должна измениться временная отметка последнего доступа к исходному файлу, а для модификации его **inode** требуются права на запись в каталог.
- Здесь нет правильного ответа.

45. Пользователи по умолчанию могут сменить свой пароль. Однако ввод нового пароля сопровождается его математическим преобразованием и записью полученной хэш-функции в теневой файл **/etc/shadow**, куда всем пользователям доступ запрещен. Каким образом решается это противоречие?
- Сигналы на смену пароля поступают администратору, и он своими правами разрешает перезапись в теневой файл.
- Некоторые файлы, включая утилиту **passwd**, имеют дополнительный бит **SUID**, позволяющий запускать программу с правами ее владельца, т. е. в данном случае – с правами **root**.
- При запуске таких программ, как **passwd, su, mount** и др., система не проверяет текущие права пользователя.
- При запуске команды **passwd** теневой файл **/etc/shadow** на очень короткое время становится доступным для записи пользователю, меняющему свой пароль. После записи хэш-функции первоначальные права доступа восстанавливаются.
- Здесь нет правильного ответа.

46. Администратор с помощью команды **chmod** изменил права доступа к файлам пользователя. Что еще кроме прав доступа изменится в этих файлах?

- Больше ничего.
- Время создания и время последнего доступа к файлам.
- Тип файлов.
- Владелец файлов.
- Группа владельца.

47. Кому можно пользоваться командой **chmod** ?

- Только администратору системы.
- Только владельцу объектового файла.
- Пользователю, у которого есть право исполнения объектового файла.
- Администратору системы или владельцу объектового файла.
- Любому пользователю по отношению к файлам, хранимым в каталоге **/home**

48. Права доступа пользователей на подкаталог описываются выражением **r-T**. Что оно означает?

- Противоречивые права. Существует запрет на удаление из подкаталога чужих файлов, но права на вход в каталог не объявлено.
- Противоречивые права. Существует запрет на удаление из подкаталога чужих файлов, но права на запись в каталог не объявлено.
- Можно просматривать подкаталог на предмет обнаружения в нем скрытых файлов.
- Этот каталог разрешено удалять вместе с хранимыми в нем файлами.
- Файлы, хранимые в этом каталоге, не подлежат модификации.

49. Почему обычным пользователям не разрешено передавать другим право владения файлами?

- В этом усматривается опасность вирусного инфицирования.
- Пользователь может передать другому файлы очень большого размера и тем самым исчерпать дисковую квоту другого пользователя.
- Такое действие явной опасности не представляет, но на всякий случай его запретили.
- Эта мера была предусмотрена в целях противодействия компьютерной преступности, чтобы хакер не мог передать другому программы, с помощью которых он совершал преступления.
- Здесь нет правильного ответа.

50. Как пользователь может запустить сценарий, на который у него есть только право чтения?

- Он должен запустить командную оболочку с именем этого файла.
- Это может сделать только администратор.
- Скопировать файл в свой каталог, установить право исполнения копии, после чего запустить ее.
- Сценарии не требуют права на запуск, оно действует только на бинарные файлы формата ELF.
- Без права на исполнение запустить сценарий невозможно.

51. Что означает команда **`find / -perm +4000 –type f`** ?
- Сортировку по объему файлов, имеющих размер более 4000 байтов.
- Поиск в корневом каталоге всех именованных каналов емкостью более 4000 байтов.
- Поиск во всей файловой системе файлов с установленным битом **`SUID`**.
- Отбор пользовательских файлов, созданных более 4000 секунд назад.
- Здесь нет правильного ответа.

52. Что означает команда **`find / -perm +1000 –type d`** ?
- Поиск и сортировку файлов, имеющих размер более 1 Кб.
- Поиск каталогов с установленным **`sticky`**-битом.
- Поиск в корневом каталоге всех именованных каналов емкостью более 1000 байтов.
- Отбор пользовательских файлов, созданных менее 1000 секунд назад.
- Здесь нет правильного ответа.

53. Что означает команда **`find / -inum 123456 –type f`** ?
- Отбор файлов с индексными дескрипторами более установленной величины.
- Поиск всех имен файла с указанным индексным дескриптором.
- Поиск группы блоков, которая включает указанный индексный дескриптор.
- Поиск файла, который включает в себя логический блок с указанным номером.
- Поиск каталогов, в которых есть файловые записи с указанными номерами.

54. Как система распознает файлы бинарных программ?
- По типовым расширениям в имени файла.
- По установленному праву на исполнение файла.
- По определенной комбинации байтов («магическому числу») в начале файла.
- Она их не различает. При запуске файла, не являющегося программой, формируется ошибка центрального процессора.

- По определенному атрибуту в индексном дескрипторе файла.

**Устройство и монтирование файловых систем, удаление и восстановление данных**

55. Какое устройство обозначается как **`/dev/sdb`** ?
- Любой машинный носитель с аппаратными интерфейсами SCSI, SATA или USB.
- Интерфейс RS-232C.
- Принтер.
- Любой жесткий магнитный диск.
- Жесткий магнитный диск с IDE-интерфейсом.
- Устройство записи/чтения CD/DVD.

56. Какое устройство может обозначаться как **`/dev/hdc`** ?
- Любой машинный носитель с аппаратными интерфейсами SCSI, SATA или USB.
- Интерфейс RS-232C.
- Жесткий магнитный диск, подключенный в качестве ведущего ко второму IDE-интерфейсу.
- Любой жесткий магнитный диск.
- Виртуальная консоль.
- Устройство записи/чтения CD/DVD.

57. Какая информация будет выведена командой **`fdisk`** ?
- Справка о внутренних командах утилиты.
- Сведения о разделах и файловых системах всех блочных устройств дисковой памяти.
- Справка обо всех файловых системах, доступных в Linux.
- Предложение указать имя блочного устройства.
- Идентификатор загрузочного диска.
- Эта команда написана с ошибкой.

58. Какая информация будет выведена командой **`fdisk -lu`**?
- Справка о внутренних командах утилиты.
- Сведения о разделах, емкости и файловых системах всех устройств дисковой памяти.
- Справка обо всех файловых системах, доступных в Linux.
- Сообщение об ошибке.
- Идентификатор загрузочного диска.

59. Что такое «суперблок» ?

- Файл с учетными записями пользователей.
- Файл, принадлежащий системному загрузчику.
- Область в начале логического диска, которая иначе именуется LILO.
- Так называется единица дискового пространства, равная 1024 блокам.
- Область диска, хранящая индексные дескрипторы файлов.
- Здесь нет правильного ответа.

60. Размер одного блока файловой системы **ext*fs** равен 4 Кб. Чему равен размер одной группы блоков?
- 32 Мб.
- 64 Мб.
- 128 Мб.
- 256 Мб.
- 512 Мб.

61. Где можно найти адреса таблиц индексных дескрипторов?
- В суперблоке.
- В описателе групп блоков.
- В каталоге **/boot**
- В главной загрузочной записи.
- Здесь нет правильного ответа.

62. Каким числом ограничены номера, присваиваемые индексным дескрипторам?
- Длина записи в каталоге и в суперблоке предусматривает максимальный номер **inode**=FF FF FF FFh=4294967295.
- Это число определено для конкретных версий и явно указано в суперблоке.
- Декларация GNU предписывает, чтобы это число не превышало 1999999.
- Это число не должно превосходить количества логических блоков, а оно в каждом конкретном случае определяется размерами блока и доступным дисковым пространством.
- Здесь нет правильного ответа.

63. Укажите максимальную длину файловой записи в каталоге.
- Она равна максимальной длине имени файла (255 байтов) плюс 8 байтов на прочие поля (4 байта – **inode**, 2 байта – длина записи, 1 байт – длина имени и 1 байт – тип файла).
- Она равна 4096 – 12 = 4084 байтов.
- В случае удаления файла его запись присоединяется к предыдущей, а длина результирующей записи соответственно увеличивается. Следовательно, максимальная длина записи 255 + 255 + 8 = 518 байтов.
- При пустом каталоге длина одной записи равна размеру каталога.

- Поскольку на длину записи отведено 2 байта, то ее максимальная длина равна $2^{16} = 65536$ байтов.
- Здесь нет правильного ответа.

64. Укажите минимальную длину файловой записи в каталоге.
- 4 байта.
- 8 байтов.
- 9 байтов.
- 12 байтов.
- 255 байтов.

65. При наличии необходимых прав доступа происходит копирование файла в иной каталог. Как при этом меняются временные отметки файловых объектов?
- Для исходного и целевого каталога изменяются времена **A** и **M**.
- У копии файла одновременно изменяются три временных отметки **A**, **C** и **M**.
- У исходного файла и у обоих каталогов изменяется время **A**.
- У целевого каталога изменяются временные отметки **A** и **M**, а у исходного каталога – время **A**.
- Время **A** изменяется у исходного каталога и файла, а временные отметки **A** и **M** – у целевых каталога и файла.
- Здесь нет правильного ответа.

66. При наличии необходимых прав доступа происходит перемещение файла в иной каталог. Как при этом меняются временные отметки файловых объектов?
- Для исходного и целевого каталога изменяются времена **A** и **M**, а у перемещенного файла – время **A**.
- У перемещенного файла одновременно изменяются три временных отметки **A**, **C** и **M**.
- У файла и у обоих каталогов изменяется время **A**.
- У целевого каталога изменяются временные отметки **A** и **M**, а у исходного каталога – время **A**.
- Время **A** изменяется у исходного каталога и файла, а временные отметки **A** и **M** – у целевого каталога.
- Здесь нет правильного ответа.

67. Символическая ссылка является указателем на файл **/etc/fstab**. В какой части символической ссылки хранится этот адрес?
- В последних 32 байтах индексного дескриптора.
- В единственном блоке данных файла символической ссылки.

- В полях индексного дескриптора, где должны храниться адреса блоков данных.
- В соответствующей записи каталога, сразу после имени самой символической ссылки.
- В таблице символических ссылок, хранимой в файле **/var/log/symbol**.

68. Администратор намерен лишить всех пользователей права копировать данные с жесткого диска на сменные устройства полупроводниковой памяти USB Flash. Как ему это сделать?
- Такой запрет можно установить только с помощью настроек Setup BIOS.
- Необходимо исключить модуль ядра, поддерживающий функционирование USB-интерфейса.
- Установить права доступа **rwx------** на все файлы **/dev/sd***.
- Исключить право монтировать устройства **/dev/sd*** в файле **/etc/fstab**.
- Здесь нет правильного ответа.

69. Что требуется для восстановления логически удаленного, но еще повторно не использованного индексного дескриптора файла в файловой системе **ext2fs** ?
- Установить в «1» число ссылок на файл и сбросить в «0» время удаления файла.
- Определить адреса блоков данных, выделенных файлу.
- Создать символическую ссылку на удаленный файл.
- Создать жесткую ссылку на удаленный файл из любого каталога этой же файловой системы.
- Установить в «1» соответствующий бит в битовой карте индексных дескрипторов.

70. Как происходит логическое удаление файла в файловой системе **ext2fs** ?
- Все части файла удаляются одновременно и полностью.
- В каталоге удаляется запись, содержащая последнее имя файла, **inode** и блоки данных файла не стираются, но объявляются свободными.
- Удаляются файловая запись в каталоге и индексный дескриптор файла.
- Удаляются только блоки данных с косвенной адресацией.
- В каталоге сохраняется имя файла, но удаляется ссылка на индексный дескриптор.

71. Сколько 4-килобайтных блоков занимает в системе **ext*fs** файл, имеющий размер 50 кБ?
- Двенадцать.
- Тринадцать.
- Четырнадцать.
- Восемь.
- Пятнадцать.

72. Что требуется для восстановления логически удаленного, но еще повторно не использованного индексного дескриптора файла в файловой системе **ext3fs** ?
- Установка в «1» числа ссылок на файл и сброс в «0» времени удаления файла.
- Индексный дескриптор в этой файловой системе восстанавливать бесполезно.
- Создание символической ссылки на удаленный файл.
- Создание жесткой ссылки на удаленный файл из любого каталога этой же файловой системы.
- Установка в «1» соответствующего бита в битовой карте индексных дескрипторов.

73. Как происходит логическое удаление файла в файловой системе **ext3fs** ?
- Удаляется запись в каталоге, содержащая последнее имя файла, **inode** и блоки данных файла не стираются, но объявляются свободными.
- Все части файла удаляются одновременно и полностью.
- Очищается индексный дескриптор файла.
- Удаляются блоки данных с прямой адресацией.
- В каталоге удаляется 4-байтный указатель на индексный дескриптор файла.

74. При удалении из каталога имени одного из файлов:
- Объем каталога уменьшается на величину удаленной записи.
- Его объем остается прежним, а длина предыдущей записи увеличивается на величину удаленной записи.
- Никакого удаления записи не происходит, только сбрасывается бит, указывающий на ее существование.
- Удаляется не имя файла, а ссылка на его индексный дескриптор.
- Ничего не происходит.

75. Какие данные хранятся в индексном дескрипторе файла **ext*fs**?
- Временные отметки.

- Адреса блоков данных, отведенных файлу.
- Идентификатор владельца файла.
- Идентификатор процесса, создавшего файл.
- Размер файла.
- Имена файла.
- Имя логического раздела, на котором хранится файл.

76. Какие временные отметки не хранит **inode** обычного файла?
- Время компиляции.
- Время создания.
- Время смены прав доступа.
- Время последнего изменения.
- Время последнего открытия.
- Время последней распечатки.
- Время удаления.
- Время смены владельца.

77. Время последнего доступа к файлу (время **A**) *не* изменяется:
- При создании файла.
- Создании новой жесткой ссылки на файл.
- Создании символической ссылки на файл.
- Копировании файла (в отношении оригинала).
- Перемещении файла.
- Открытии файла для его просмотра.

78. Время последнего доступа к каталогу **/bin** (время **A**) изменяется:
- При создании каталога.
- Входе в каталог по команде **cd /bin**.
- Выходе из каталога по команде **cd ..**
- Создании символической ссылки на каталог.
- Изменении прав доступа к каталогу.
- Копировании в каталог новых файлов.

79. В каких случаях обновляется время создания файла (время **C**) ?
- При передаче прав на файл другому пользователю.
- При изменении прав доступа к файлу с помощью команды **chmod .**
- При копировании файла (в отношении копии).
- При перемещении файла в другой каталог.
- При удалении последней жесткой ссылки на файл.
- При создании у файла нового имени (новой жесткой ссылки).

80. В каких случаях обновляется время модификации файла (время **M**)?
- При изменении прав доступа к файлу.
- При копировании файла (в отношении копии).
- При копировании файла (в отношении оригинала).
- При передаче прав на файл другому пользователю.
- При перемещении файла в другой каталог.
- При удалении последней жесткой ссылки на файл.
- При создании у файла нового имени (новой жесткой ссылки).

81. Командой **ln** создана еще одна жесткая ссылка на существующий файл, а затем символическая ссылка на одно из имен файла. Имена обеих ссылок содержат по 8 символов в ASCII-кодах. Каталоги, в которых созданы ссылки, до конца не заполнены файловыми записями. Какая из ссылок занимает больше места на диске ?
- Жесткая ссылка по объему больше.
- Символическая ссылка по объему больше.
- Обе ссылки одинаковы по объему.
- Обе операции не сопровождаются выделением дополнительного дискового пространства.
- Здесь нет правильного ответа.

82. Где хранится имя файла?
- В таблице индексных дескрипторов.
- В описателе групп блоков.
- В суперблоке.
- В каталоге.
- В оперативной памяти процесса, создавшего файл.

83. Как можно увидеть и установить право на удаление файла?
- Такого права не предусмотрено.
- Права на модификацию **w** и удаление файла совпадают.
- Для этого следует открыть для просмотра и редактирования индексный дескриптор файла.
- Для этого следует запустить отладчик файловой системы.
- При просмотре и установке права записи на каталог, содержащий этот файл.

84. Что произойдет при выполнении команды **mount -t ntfs /dev/sda2 /bin** ?
- В результате монтирования будет заблокирован каталог с системными утилитами.

- Произойдет повреждение первых секторов логического раздела **/dev/sda2** .
- Будет выдано сообщение о недоступности точки монтирования.
- Произойдет повреждение суперблока файловой системы.
- Операция монтирования логического раздела произойдет успешно.

85. Права доступа пользователей к утилите **mount** описываются выражением **rwsr-xr-x**. Но, несмотря на права чтения и исполнения с правами владельца, пользователю может быть отказано в запуске этой утилиты. Почему?
- Запрет может быть обусловлен отсутствием прав на объект монтирования.
- Запрет может вытекать из параметров монтирования, указанных в файле **/etc/fstab** .
- Соответствующий запрет следует искать в файле **/etc/mtab**.
- Запрет установлен в файле **/proc/mount**.
- Все дело в отсутствии нужных драйверов.

86. Что делает данная команда: **mount –t vfat –o ro /dev/sdb2 /mnt/usb** ?
- Монтирует для чтения и записи логический раздел с файловой системой FAT12.
- Монтирует только для чтения логический раздел с файловой системой FAT16 или FAT32.
- Делает доступным для чтения сетевой диск **/mnt/usb**.
- Размонтирует раздел с операционной системой Windows Vista.
- Монтирует оптический диск CD-R/DVD-R.

87. Какие права нужно иметь на каталог, выбранный в качестве точки монтирования?
- Эта операция производится администратором, а для него проверка прав доступа не предусмотрена.
- Право на исполнение.
- Право на чтение и исполнение.
- Право на запись и исполнение.
- Полные права.
- Для монтирования никаких прав доступа не требуется.

88. Производится монтирование сменного носителя USB-Flash с файловой системой **vfat**. Кто будет являться владельцем файлов на примонтированном разделе, и какие права доступа к ним будут установлены?

- Владельцем будет пользователь, производивший монтирование, а права доступа определяются его маской.
- Владельцем будет **root**, а права доступа определяются по умолчанию как 0755.
- Владельцем будет пользователь, производивший монтирование, а права доступа определяются как 0744.
- Владельцем будет **root**, а права доступа определяются его маской.
- Поскольку монтируется система FAT, то владелец файлов и права доступа к ним будут отсутствовать.

89. Чем отличается информация, содержащаяся в конфигурационных файлах **/etc/fstab** и **/etc/mtab**?

- Эти файлы дублируют друг друга и отличаются по названию ради совместимости различных версий системы.
- Файл **/etc/fstab** перечисляет файловые системы, которые *могут* быть смонтированы пользователями с указанием их прав, а файл **/etc/mtab** содержит информацию об *уже* смонтированных ФС.
- Файл **/etc/mtab** перечисляет файловые системы, которые *могут* быть смонтированы, а файл **/etc/fstab** содержит информацию об *уже* смонтированных ФС.
- Файл **/etc/fstab** перечисляет файловые системы, которые монтируются автоматически, а файл **/etc/mtab** информирует об уже смонтированных системах.
- Здесь нет правильного ответа.

90. Пользователь на своем домашнем компьютере, где он имеет права администратора, скопировал на сменный носитель файл **chmod**, после чего установил для него бит эффективных прав доступа **setuid**. С помощью подготовленной утилиты он намерен захватить права администратора на рабочем месте. Как этому должен противодействовать настоящий администратор?

- Необходимо для сменных носителей в конфигурационном файле **/etc/fstab** установить опцию **noexec**.
- Необходимо для сменных носителей в конфигурационном файле **/etc/fstab** установить опцию **nosuid**.
- Необходимо в конфигурационном файле **/etc/mtab** для сменных носителей установить опцию **default**.
- Реагирования не потребуется, т. к. в конфигурационном файле **/etc/fstab** для сменных носителей по умолчанию предусмотрен параметр **default**, который означает запреты **noexec**, **nosuid** и **nodev**. Но администратору в этом следует убедиться.
- Здесь нет правильного ответа.

91. Что произойдет, если смонтированный сменный носитель (компакт-диск, дискету, USB-Flash) извлечь из привода или отстыковать от интерфейса без размонтирования ?
- Автоматически запустится утилита **fsck** и произведет проверку файловой системы.
- Поступит запрос на внеочередную проверку файловой системы.
- Ничего не произойдет.
- При очередной выгрузке дискового кэша на физический носитель система сообщит о фатальной ошибке либо «зависнет».
- Утилита **fsck** запустится и произведет проверку файловой системы при очередной перезагрузке операционной системы.
- Произойдет очередная перезапись данных в суперблоке.

92. Что произойдет при вводе команды **mount** без каких-либо аргументов?
- Автоматически запустится утилита **fsck** и произведет проверку файловой системы.
- Поступит сообщение о неверном синтаксисе введенной команды.
- Будет выведен запрос о вводе недостающих параметров.
- Отобразится таблица уже смонтированных файловых систем из **/proc/mounts**.
- Будет отображена таблица файловых систем, подлежащих монтированию, из файла **/etc/fstab**.
- Ничего не произойдет.

93. Какие из приведенных команд копирования содержат ошибки ?
- **cat /bin/* > /tmp**
- **dd of=/home/hd1 if=/dev/hda1 count=800**
- **tar –zcvf /var/log > /root/audit.log**
- **cp /sbin/d* /usr/sbin**
- **od /dev/fd0 > home/floppy1**
- Все команды написаны правильно.

94. Что делает данная команда?

**dd of=/dev/fd0 if=/dev/sda count=63**

- Копирует на сменный носитель суперблок файловой системы.
- Копирует на дискету главную загрузочную запись с жесткого диска.
- Записывает начальные 63 сектора с дискеты на оптический диск.
- Создает теневой раздел памяти BIOS в «нижней» оперативной памяти.
- Стирает начальные сектора на дискете и на жестком магнитном диске.

95. Что делает данная команда (раздел **sda2** жесткого диска содержит файловую систему **ext*fs**)?

**dd if=/dev/sda2 bs=4096 skip=1 count=1|dd bs=32 skip=4 count=1|xxd**

- Отображает фрагмент битовой карты блоков.
- Выводит на экран содержимое описателя пятой группы блоков.
- Копирует с жесткого диска значимые системные данные суперблока.
- Отображает четвертый индексный дескриптор.
- Читает заголовок блока данных корневого каталога.

96. Что делает данная команда (раздел **sda3** жесткого диска содержит файловую систему **ext2fs**)?

**dd if=/dev/sda3 bs=4096 skip=11 count=1|dd bs=128 skip=5 count=1|xxd**

- Отображает фрагмент битовой карты **inode.**
- Выводит на экран содержимое описателя шестой группы блоков.
- Копирует с жесткого диска фрагмент журнала трансакций.
- Отображает двести тридцатый индексный дескриптор.
- Читает заголовок блока данных каталога **/lost+found.**

97. Что делает данная команда ?

**echo!|dd of=/etc/shadow bs=1 seek=5 count=1**

- Вызывает сбой в системе по причине неправильного синтаксиса команды.
- Блокирует учетную запись последнего зарегистрированного пользователя.
- Устанавливает символ **!** в начале хэш–функции пароля администратора и стирает оставшуюся часть его учетной записи.
- Принудительно завершает выполнение процесса **init.**
- Запись данных в это место файла к опасным последствиям не приводит.

98. Пользователь командой **echo 1234567890 > abcd** перенаправляет стандартный вывод команды **echo** из консоли в существующий файл объемом 20 Кб. Что при этом произойдет?

- Система предупредит пользователя о существовании такого файла и запросит разрешение на его замену.
- Индексный дескриптор и блоки данных существующего файла будут стерты и полностью обновлены.
- Создается новый файл с прежним именем, для которого будет использован индексный дескриптор и первый блок удаленного файла. Остальные

блоки прежнего файла объявляются свободными и временно сохраняют прежнюю информацию.

- Такой файл будет создан заново, а существующий будет переименован в **$abcd**.
- Старый файл окажется стертым, а новый – пустым.
- Будет уничтожена файловая запись **abcd** в каталоге, прежде содержавшем этот файл.

99. Администратор с помощью команды **echo текст >> /var/log/audit1** намеревался дописать одну строку в файл аудита, размер которого составлял 42 Кб. По небрежности он вместо парного символа **>>** ввел одинарный. Что при этом произойдет?

- Процесс **syslog** предупредит администратора о недопустимости стирания файлов в этом каталоге.
- Файл **audit1** на диске будет полностью стерт.
- Будет очищен первый блок файла и в него будет записана строка, введенная командой **echo**. Остальные блоки данных будут объявлены свободными, и в них сохранится прежняя информация.
- Поскольку запись обновленных данных производится вначале в журнал и в дисковый кэш, администратор, воспользовавшись дисковыми утилитами, успеет спасти важную информацию.
- Здесь отсутствует правильный ответ.

100. Пользователь удалил из своего каталога текстовый файл объемом 120 Кб и после этого создал там же три новых файла с размерами 5 Кб, 14 Кб и 470 Кб. Спустя некоторое время пользователь пожалел о своей оплошности и обратился к администратору за помощью в спасении остаточной информации. Файловая система использует логические блоки размером 4 Кб. Сколько данных администратор успеет спасти, если своевременно воспользуется дисковыми утилитами ? Пользователь помнит отдельные слова из удаленного файла.

- 43 Кб.
- 96 Кб.
- 92 Кб.
- 87 Кб.
- 25 Кб.

101. Как можно использовать команду **time <file_name>**?

- Для получения текущего времени.
- Для получения времени создания файла.
- Для получения времени последней модификации файла.
- Для определения времени, затраченного на выполнение программы.

- Для получения времени с момента авторизации пользователя в системе.
- Такой команды в ОС Linux не существует.

102. Как можно использовать команду **file <file_name>**?
- Для получения текущего времени суток.
- Для получения астрономического времени
- Для получения временных отметок файла.
- Для определения типа файла.
- Такой команды в ОС Linux не существует.

**Работа с учетными записями, приобретение прав доступа**

103. Какой результат будет достигнут при вводе администратором следующей команды ?

**echo 'ivan:x:0:0::/root' >> /etc/passwd**
- Файл паролей будет необратимо испорчен.
- Будет создана учетная запись еще одного администратора системы с именем **ivan**, но ею невозможно будет воспользоваться.
- Система сообщит, что у администратора уже имеется учетная запись.
- Будет создана работоспособная учетная запись пользователя с именем **ivan** с обычными правами.
- Произойдет сбой в системе.
- Произойдет сбой в текущей консоли.

104. В каком конфигурационном файле и в каком виде содержатся записи о членстве пользователей в дополнительных группах?
- Такие сведения содержатся в табличном виде в конфигурационном файле **/usr/etc/.group**.
- Такие сведения содержатся в домашнем каталоге пользователя в скрытом файле **.bash_logout**.
- В файле **/etc/passwd**. Имена групп записываются в 4-е поле учетной записи пользователя, через запятую после числового идентификатора основной группы.
- В файле **/etc/group**. Имена пользователей, дополнительно включенных в группу, дописываются через запятую в 4-е поле записи для данной группы.
- В файле **/etc/shadow**. Имена групп записываются в 9-е поле учетной записи пользователя.
- Здесь отсутствует правильный ответ.

105. В правах на доступ к файлу указываются права группы. О каких группах идет речь?

- Об основной группе, которая была явно указана или записана по умолчанию при создании учетной записи пользователя.
- О дополнительных группах, в которые пользователь был явно включен администратором.
- Об основной и всех дополнительных группах, в которые был включен пользователь.
- О группе пользователя, создавшего данный файл.
- О группе администратора.

106. Можно ли «создать» пользователя, не входящего ни в одну из групп? Как это сделать?
- Это сделать невозможно, т. к. группа предусмотрена даже у администратора.
- Можно при условии редактирования файла **/etc/group** с указанием в четвертом поле учетной записи пользователя несуществующего номера группы.
- Можно при указании в команде **useradd** или **usermod** с аргументом **-g** несуществующего номера группы.
- Можно, если при создании новой учетной записи пользователя имя или номер группы явно не указывать.
- Можно с помощью команды **usermod -g '' user_name** .
- Можно, если предусмотреть специальную запись в конфигурационном файле **/etc/login.defs**.

107. Как можно создать в системе еще одного администратора, кроме **root**?
- С помощью утилиты **useradd** путем явного указания имени пользователя с **UID**=0.
- С помощью сценария **adduser**, в котором для этого предусмотрена специальная опция.
- Система не позволит это сделать.
- С помощью команды

**echo 'petrov::0:0::/root:/bin/bash' >> /etc/passwd** .
- Путем создания обычной учетной записи с последующим изменением **UID** пользователя на нулевой в файле **/etc/passwd**.

108. Может ли пользователь по своему усмотрению дополнительно включить себя в другую группу?
- Только в том случае, если он является администратором или ему дано право запускать с правами **root** команды **useradd** или **usermod**.
- Может с помощью команды **useradd** или **usermod**.
- Такие права обычному пользователю недоступны.
- Может с помощью команды **newgrp**.

- Может с помощью команды **sg**.

109. Чем различаются команды **newgrp** и **sg**?

- С помощью команды **newgrp** можно поменять основную группу пользователя, а команда **sg** служит для присвоения групповых эффективных прав.
- С помощью команды **newgrp** можно создать дополнительную группу и включить в нее пользователя, а команда **sg** служит для смены основной группы пользователя.
- Команда **sg** является символической ссылкой на утилиту **newgrp**, следовательно, они ничем не отличаются.
- С помощью команды **newgrp** можно создать дополнительную группу, а команда **sg** служит для включения в эту группу пользователя.
- Это одинаковые команды, но **newgrp** изменяет основную группу пользователя надолго, а команда **sg** изменяет группу на сеанс.

110. Как изменить основную группу пользователя?

- С помощью команды **newgrp** можно поменять основную группу пользователя. При этом он не исключается из других групп, в которых до этого состоял.
- С помощью команды **newgrp** можно поменять основную группу пользователя. При этом он исключается из всех других групп, в которых до этого состоял.
- С помощью команды **newgrp** можно поменять основную группу пользователя. При этом он исключается из прежней основной группы.
- С помощью команды **usermod –g new_group user_name**.
- Для этого необходимо исключить его из прежней группы, а затем назначить в новую группу.
- С помощью команды **sg** можно поменять основную группу пользователя. При этом он исключается из прежней основной группы и всех дополнительных групп.

111. Администратор с помощью команды **su petrov** пытается трансформироваться в указанного пользователя, но забыл, что сам заблокировал учетную запись этого пользователя (восклицательный знак в поле признака пароля). Каким будет результат?

- Администратор получит сообщение **login incorrect**.
- Трансформация произойдет успешно.
- Команда будет игнорирована.
- Из-за коллизии произойдет принудительное завершение текущего сеанса в консоли администратора.

- Произойдет «зависание» системы.

112. Чем отличаются права на файлы владельца со стороны пользователей его основной и всех дополнительных групп?
- Члены основной группы пользуются групповыми правами, а члены дополнительных групп – правами всех остальных пользователей.
- Ничем не отличаются.
- Для членов дополнительных групп недоступен запуск файлов с правом его владельца.
- Для членов дополнительных групп недоступно право на модификацию файлов.
- Эти отличия зависят от версии операционной системы.

113. Как можно получить информацию об основной и дополнительных группах пользователя?
- Это можно сделать путем просмотра и сопоставления записей в конфигурационных файлах **/etc/group** и **/etc/passwd**.
- Такая информация выводится только для пользователя, вводящего команду **id**.
- На запрос администратора **id user_name** будет выведена информация о группах любого пользователя.
- С помощью команды **id user_name** любой пользователь может получить информацию об **UID**, **GID** и символьных именах групп любого пользователя, зарегистрированного в данный момент в системе.
- С помощью команды **id user_name** любой пользователь может получить информацию об **UID**, **GID** и символьных именах групп любого пользователя, имеющего учетную запись.

114. Пользователь **semaforkin** является членом основной группы **alfa**. Но создаваемые этим пользователем текстовые файлы должны быть доступны для чтения только членам его дополнительной группы **beta**. Как это сделать?
- Без помощи администратора здесь не обойтись.
- Эта задача средствами Linux не решается.
- Сменить основную группу на **beta** с помощью команды **usermod**
- Установить права доступа на эти файлы **rw----r--**
- Командой **chgrp** передать право группового владения этими файлами группе **beta**

115. На домашний каталог пользователя **bob** установлены права **rwxr-x--t**. В системе зарегистрированы член основной группы **merilin**, член дополнительной группы **braun** и пользователь **alisa**, не имею-

щий никакого отношения к группам **bob**. Какие из высказываний верны?

- Пользователь **merilin** имеет право входить в каталог и манипулировать файлами в зависимости от прав доступа на них.
- Пользователь **merilin** имеет право удалять файлы **bob**.
- Пользователь **bob** имеет право удалить свой домашний каталог.
- Пользователь **alisa** имеет право на копирование и переименование файлов **bob**.
- Пользователь **alisa** не имеет права удалять файлы **bob**.
- Пользователь **braun** имеет право создавать файлы в каталоге **bob**.
- Пользователь **braun** не имеет прав на копирование и перемещение файлов **bob**

116. Учетная запись пользователя в файле **/etc/passwd** имеет следующий вид: **pavlov::1:2:** .

Какие проблемы возникнут у этого пользователя?

- Никаких. При вводе идентификатора пользователь будет сразу авторизован в системе.
- При вводе идентификатора пользователь будет сразу авторизован в системе, но у него не будет домашнего каталога.
- При вводе идентификатора пользователь будет сразу авторизован в системе, но из-за отсутствия командной оболочки не сможет управлять системой.
- Ему будет отказано в авторизации.
- Проблемы возникнут у администратора. Наличие в файле паролей подобной записи сделает невозможной авторизацию других пользователей.

117. Учетная запись пользователя в файле **/etc/passwd** имеет следующий вид: **petrov:::**

Какие проблемы возникнут у этого пользователя?

- Никаких. При вводе идентификатора пользователь будет сразу авторизован в системе.
- При вводе идентификатора пользователь будет сразу авторизован в системе, но у него не будет домашнего каталога.
- При вводе идентификатора пользователь будет сразу авторизован в системе, но из-за отсутствия командной оболочки не сможет управлять системой.
- Ему будет отказано в авторизации.
- Программа запросит пароль, однако любая введенная комбинация будет отвергнута.

- Проблемы возникнут у администратора. Наличие в файле паролей подобной записи сделает невозможной авторизацию других пользователей.

118. Учетная запись пользователя в файле **/etc/shadow** имеет следующий вид:

**braun:$1$Yj7IcY8O$oCX9V9QYipDtYflbfOOBE1:12496:99999:**

Какие проблемы возникнут у этого пользователя, если в первой части учетной записи (в файле **/etc/passwd**) все в порядке ?

- Никаких.
- Система «замучает» этого пользователя напоминаниями о необходимости смены пароля, но сменить ему пароль не позволит.
- Он не сможет войти в систему из-за неверной хэш-функции пароля.
- Его пароль заблокирован.
- Ему будет отказано в авторизации по причине отсутствия трех последних временных меток.

119. Может ли пользователь изменить свою учетную запись?

- Может, но только общие сведения о себе, которые фиксируются в пятом поле записи в файле **/etc/passwd**.
- Это позволено только администратору.
- Может, но только свой пароль.
- Может, но только временные атрибуты пароля.
- Может, но только свою основную группу.

120. Может ли пользователь по своему усмотрению исключить себя из какой-либо дополнительной группы?

- Только в том случае, если он является администратором или ему дано право запускать с правами **root** команду **useradd** или **usermod**.
- Может с помощью команды **useradd** или **usermod**.
- Такие права обычному пользователю недоступны.
- Может с помощью команды **newgrp**.
- Может с помощью команды **sg** с вводом группового пароля.

121. Может ли пользователь по своему усмотрению исключить себя из основной группы?

- Таких прав нет даже у администратора.
- Только в том случае, если он является администратором или ему дано право запускать с правами **root** команду **useradd** или **usermod**.
- Может с помощью команды **useradd** или **usermod**.

- Может, включая себя в другую основную группу с помощью команды **newgrp**.
- Может с помощью команды **sg** с вводом группового пароля.

122. Может ли пользователь досрочно сменить свой пароль?
- Таких прав нет даже у администратора.
- Может.
- Может, если предварительно отредактирует скрытый файл **.bash_logout** в своем домашнем каталоге.
- Может, если предварительно командой **chage** изменит минимальный срок действия пароля.
- Не может.

123. Может ли пользователь вопреки предупреждению системы установить для себя простой для запоминания пароль?
- Таких прав нет даже у администратора.
- Может.
- Может, если предварительно установит минимальную длину пароля в скрытом файле **.bash_logout** своего домашнего каталога.
- Не может.
- Может, если повторит ввод желаемого пароля несколько раз подряд.
- Может, если введет пароль с аргументом **-p** в команде **usermod**.

124. Чем принципиально отличаются дополнительные полномочия на запуск команд, предоставляемые механизмом **sudo** и эффективным битом **SUID** ?
- По существу, это одинаковые права запуска программ от имени иного, привилегированного лица.
- Это одинаковые права, если в файле **sudoers** разрешение на запуск записано в виде **ALL ALL=(root) NOPASSWD: mount**
- Это неодинаковые права, поскольку в файле **sudoers** можно предусмотреть права на привилегированный запуск команды от конкретного пользователя с конкретного места, а установленный бит **SUID** дает право запуска данной команды любому пользователю с любого узла сети.
- Это неодинаковые права, поскольку установленный бит SUID дает право запуска данной команды любому пользователю с любого узла сети, а в файле **sudoers** можно предусмотреть разрешение вида **ALL=ALL .**
- Здесь отсутствует правильный ответ.

125. Пользователи работают со сведениями различного уровня конфиденциальности, и администратору необходимо жестко зафиксировать их

права доступа к существующим и вновь создаваемым файлам. Как это лучше сделать?

- Задать для пользователей требуемую маску доступа для вновь создаваемых и копируемых файлов и лишить пользователей возможности исполнять команды **chmod** и **umask**.
- Достаточно установить на файл **chmod** права доступа **rwxr-x-r--**
- Добавить запись **umask 0077** в конец файла **/etc/skel/bash_profile**, а затем произвести перерегистрацию пользователей, работающих с конфиденциальными данными.
- Средствами ОС Linux такую задачу решить невозможно.
- Для нескольких пользователей такое ограничение можно задать, объединив их в группу и включив в нее администратора. После этого доступ группы к каждому из защищаемых файлов запретить.

126. Администратор создал новую учетную запись с помощью команд **useradd semaforkin; passwd semaforkin** .

В каком каталоге окажется вновь созданный пользователь после регистрации?

- В каталоге **/root** .
- В каталоге **/home** .
- В каталоге **/home/semaforkin**.
- В каталоге **/tmp** .
- В корневом каталоге.
- Поскольку администратор не предоставил пользователю домашний каталог, последнему не удастся войти в систему.
- Здесь не указано правильного ответа.

127. Администратор удалил учетную запись уволенного сотрудника, но не стал удалять его домашний каталог с ценными файлами. Как изменятся права владения этими файлами?

- Они не изменятся. Со временем их владельцем станет новый сотрудник, которому при регистрации будет присвоен UID прежнего пользователя.
- Они не изменятся. Файлы окажутся бесхозными и будут принадлежать всем пользователям.
- Поскольку UID уволенного сотрудника больше использоваться не будет, то файлы окажутся заблокированными для всех, кроме администратора.
- Файлы перейдут во владение администратора.
- Файлы перейдут во владение одного из псевдопользователей.

**Процессы**

128. Какая из команд не приведет к останову системы?

- **halt**
- **shutdown –t 0**
- **poweroff**
- **inin 0**
- **exit**
- **<Ctrl>+<Alt>+<Del>**

129. Пользователь ввел и запустил в командной строке следующую последовательность команд:

**dd if=/dev/hda count=1 | xxd | more**

Какие из этих процессов сможет увидеть администратор с помощью команды **ps –ef** ?

- Утилита **ps** не предназначена для отображения конвейерных операций.
- Каждый из процессов, участвующих в конвейере, будет отображен отдельной строкой в порядке его исполнения.
- Можно будет увидеть только первый процесс.
- Можно будет увидеть только последний процесс.
- В командной строке допущена ошибка, поэтому утилита **ps** ни один из этих процессов не покажет.

130. Какая из команд не приведет к перезагрузке системы?

- **init 6**
- **reboot**
- **telinit 6**
- **rehalt**
- **<Ctrl>+<Alt>+<Del>**

131. Вам необходимо передать немедленное сообщение системному администратору. Каким способом это сделать невозможно?

- С помощью команды **wall**.
- С помощью команды **write**.
- С помощью команды **mesg**.
- С помощью команды **talk**.
- С помощью команды **echo сообщение > /dev/tty1**, где **tty1** – терминал администратора.

132. Как защитить свой терминал от возможности перехвата клавиатурного ввода другими пользователями?

- С помощью команды **mesg y**, запущенной с этого терминала.

- Терминальный ввод на каждом рабочем месте защищается системой по умолчанию, пользователям дополнительных мер защиты предпринимать не нужно.
- С помощью команды **chmod 600 /dev/ttyN**, где **N** – номер терминала.
- Действенной защиты от перехвата клавиатурного ввода в операционных системах Linux не существует.
- Это можно сделать только с помощью криптозащиты.

133. Для чего предназначена команда **mesg** ?
- Для блокирования или разрешения записи в терминал, из которого она запущена.
- Для блокирования или разрешения чтения из терминала, из которого она запущена.
- Для блокирования или разрешения записи в терминал произвольного пользователя.
- Команда аналогична переключателю **chmod 620 /dev/tty – chmod 600 /dev/tty** .
- Команда аналогична переключателю **chmod 660 /dev/tty – chmod 620 /dev/tty** .

134. Каким образом заблокировать терминал **tty1** от поступления сообщений?
- С помощью команды **mesg n**, запущенной с этого терминала.
- С помощью команды **exit**.
- С помощью команды **chmod 600 /dev/tty1** .
- Путем передачи прав на файл **/dev/tty1** другому пользователю.
- Путем редактирования файла **.bash_logout** в домашнем каталоге пользователя.

135. Какие разрешения установлены на файлы команд **write** и **wall**?
- **rwxr-sr-x**
- **rwxr-xr-x**
- **rwxr-x---**
- **rw-r--r--**
- **rwsr-xr-x**

136. Какая команда необходима для выяснения, кто из пользователей, на каком рабочем месте и в течение какого времени работает в системе?
- **w**
- **who**

- **users**
- **why**
- **when**
- Здесь нет правильного ответа.

137. Пользователь решил скрытно запустить процесс, напрасно расходующий все процессорное время. Для этого он написал сценарий следующего содержания:

```
#! /bin/bash
cat /dev/zero > /dev/null
```

сохранил его под именем **...** в каталоге **/tmp**, присвоил ему необходимые права для запуска и запустил. Под каким именем данный процесс будет отображаться утилитами **ps** и **top**?

- Такой процесс вообще не запустится по причине допущенной ошибки.
- Процесс будет иметь имя **...**
- Процесс будет иметь имя **bash**.
- Процесс будет иметь имя **bash ...**
- Процесс будет иметь имя **cat**.
- Процесс будет расходовать процессорное время, но останется невидимым.

138. Администратор зарегистрировался в первой консоли с правами **root**. Затем в целях безопасности с помощью команды **su** он временно стал пользователем **petrov**. Через некоторое время он таким же путем трансформировался в пользователя **ivanov**. В это время другой пользователь с помощью команды **w** решил узнать, кто, кроме него, работает в системе. Чье имя он увидит в первой консоли?

- **root**
- **petrov**
- **ivanov**
- Вместо имени будет отображаться знак вопроса.
- Команда **w** не выводит информацию о пользовательских сеансах.

139. Администратор, редактируя обычный документ, с помощью команды **su** временно трансформировался в пользователя **petrov**. Какая из команд позволит ему восстановить свой прежний статус, не вводя пароля?

- **su root**
- **su -root**
- **su**
- **exit**
- **fg %1**

140. Как стереть файл истории команд текущего пользователя?

- С помощью команды **export HISTSIZE=0** .
- С помощью команды **echo ! > ~/.bash_history** .
- С помощью редактора **vi** или **mcedit**.
- С помощью команды **cat /dev/zero > history** .
- С помощью команды **cat /dev/null > history** .

141. Какому из процессов планировщик задач выделит больше процессорного времени?

- Процессу, запущенному от имени администратора.
- Процессу, запущенному от имени пользователя.
- Сервису (демону), запущенному системой на этапе загрузки.
- Процессу, имеющему более высокий приоритет.
- Процессу, занимающему больше виртуальной памяти.

142. Может ли администратор завершить свой сеанс командой **exit** и хотя бы на время «покинуть» систему?

- Администратор может завершить свою работу только командами **shutdown**, **poweroff**, **halt** или **reboot**.
- Может. Система ведь работает и до регистрации суперпользователя.
- Может и должен. Администратору рекомендуется, если это не вызывается особой необходимостью, работать в системе с правами обычного пользователя.
- Администратор не должен завершать свой сеанс. Но он может на время с помощью команды **su** стать другим пользователем.
- Система не позволяет суперпользователю запускать команду **exit**.

143. Из консоли с правами администратора запущена команда **kill –9 1** для завершения процесса **init**. Что за этим последует?

- Перезагрузка системы.
- Завершение сеанса администратора.
- Завершение работы системы.
- Игнорирование команды.
- Сообщение о недостатке прав доступа.
- Сообщение о невозможности выполнить команду.

144. Из первой консоли с правами администратора запущена команда **skill –STOP tty2**. Что за этим последует?

- Перезагрузка системы.
- Блокировка ввода-вывода на второй консоли.

- Завершение пользовательского сеанса.
- Командный интерпретатор сообщит об отсутствии такой команды.
- Ничего не произойдет.

145. Пользователь из третьей виртуальной консоли запустил команду **chmod 606 /dev/vc/3**. Что за этим последует?
- Консоль окажется заблокированной.
- Ничего не произойдет.
- Консоль станет недоступной для сообщений других пользователей, исключая администратора.
- Весь клавиатурный ввод третьей консоли будет автоматически транслироваться на монитор администратора.
- Любой из пользователей, введя команду **cat /dev/vc/3** , может перехватывать (копировать) клавиатурный ввод с третьей консоли, а с помощью команды **echo сообщение > /dev/vc/3** передавать на нее сообщения.
- Пользователь получит сообщение о нарушении прав доступа.

146. Из первой консоли с правами администратора запущены две команды **skill –STOP tty2 && echo как дела? > /dev/tty2**. Что за этим последует?
- Произойдет блокировка ввода-вывода на второй консоли.
- Сообщение об ошибке.
- Заблокированный терминал не будет реагировать на пользовательский ввод, но сообщение администратора получит.
- Произойдет сбой во второй консоли с нарушением кодировки.
- Ничего не произойдет.
- Зависание системы.

147. Какой командой следует завершить системный сервис **syslogd**?
- **syslogd stop**
- **stop syslogd**
- **kill –9 syslogd**
- **skill stop syslogd**
- **syslogd halt**
- **syslogd kill**

148. Какой командой можно перезагрузить операционную систему?
- **restart**
- **reboot**
- **telinit 6**

- **init 0**
- **poweroff**
- **shutdown**

149. Какой командой можно запустить демон **inetd** ?
- **inetd**
- **inetd start**
- **start inetd**
- **inetd run**
- **run inetd**

150. Какие из приведенных команд выполняют одно и то же действие?
- **which ls**
- **find / -name ls**
- **touch ls**
- **ls –l ls**
- **ps –e ls**

151. Какие из приведенных команд выполняют одно и то же действие?
- **which**
- **who**
- **w**
- **wireless**
- **users**

152. Какие из приведенных команд выполняют одно и то же действие?
- **cat > abcd**
- **dd of=adcd**
- **touch abcd**
- **od > abcd**
- **hexdump abcd**

**Сетевые возможности системы**

153. Что произойдет при вводе команды **ifconfig eth0 -arp** ?
- Этот сетевой адаптер становится невидимым в локальной сети.
- Невозможно обнаружить компьютер по его IP-адресу.
- Невозможно обнаружить компьютер по его MAC-адресу.
- Компьютер полностью изолирован от локальной сети.
- Здесь отсутствует правильный ответ.

154. В строке состояния сетевого адаптера, выводимой по команде **ifconfig eth0**, отображается следующее: **UP BROADCAST MULTICAST**. Что это означает?

- Сетевой адаптер в рабочем состоянии.
- Сетевой адаптер можно использовать для перехвата всех пакетов в моноканале.
- Сетевой адаптер отключен от локальной сети.
- Компьютер прослушивает сеть, но сам остается невидимым.
- Здесь отсутствует правильный ответ.

155. В строке состояния сетевого адаптера, выводимой по команде **ifconfig eth0**, отображается следующее:
**BROADUP CAST RUNNING ARP NOPROMISC MULTICAST**
Что это означает?

- Подобная строка состояния не может отображаться.
- Сетевой адаптер в рабочем состоянии.
- Сетевой адаптер можно использовать для перехвата всех пакетов в моноканале.
- Сетевой адаптер отключен от локальной сети.
- Компьютер прослушивает сеть, но сам остается невидимым.
- Здесь отсутствует правильный ответ.

156. Что произойдет при вводе команды
**arp –s 192.168.0.1 A0:B1:C2:D3:E4:F5** ?

- Будет создан файл с записью соответствия IP и MAC-адресов данного сетевого адаптера.
- Будет создана статическая запись о соответствии IP и MAC-адресов в ARP-кэше.
- В этой команде нет никакого смысла.
- В ЛВС будет направлен ARP-запрос на соответствие сетевого и аппаратного адресов.
- Будет создан файл с записью соответствия указанных IP и MAC-адресов в данной ЛВС.

157. Что произойдет при вводе команды **arp –d 192.168.0.1** ?

- Будет удалена динамическая запись об указанном сетевом адресе.
- Будет удалена статическая запись об указанном сетевом адресе.
- В этой команде нет никакого смысла.
- Придется заново указывать IP-адрес сетевому адаптеру.
- В ЛВС будет направлен ARP-запрос на поиск узла с указанным адресом.

- Будет создан файл с записью соответствия указанных IP и MAC-адресов в данной ЛВС.

*Учебное издание*

***Бакланов** Валентин Викторович*

**ЗАЩИТНЫЕ МЕХАНИЗМЫ ОПЕРАЦИОННОЙ СИСТЕМЫ LINUX**

Редактор *Л.Ю. Козяйчева*

Подписано в печать 13.10.11 Формат 60x84 1/16.
Бумага типографская. Плоская печать. Усл. печ. л. 20,6.
Уч. - изд. л. 23,6. Тираж 100 экз. Заказ

Редакционно-издательский отдел УрФУ
620002, Екатеринбург, ул. Мира, 19
rio@mail.ustu.ru
Ризография НИЧ УрФУ
620002, Екатеринбург, ул. Мира, 19

370