Федеральное агентство по образованию

ГОУ ВПО «Уральский государственный технический университет – УПИ»

В. В. Бакланов

АДМИНИСТРИРОВАНИЕ И БЕЗОПАСНОСТЬ ОПЕРАЦИОННЫХ СИСТЕМ LINUX

Учебное пособие

Научный редактор проф., д-р. техн. наук, Н.А. Гайдамакин

Екатеринбург
2005

УДК 661.3.066
ББК 32.973.26

Рецензенты:

кафедра систем и технологий защиты информации Уральского государственного университета путей сообщения (зав. кафедрой проф., д-р физ.-мат. наук Ю.И.Ялышев)

доц., канд. физ.–мат. наук О.Н. Соболев (Институт переподготовки и повышения квалификации сотрудников ФСБ России, г. Екатеринбург)

Автор: В.В.Бакланов

Администрирование и безопасность операционных систем Linux:
учебное пособие / В.В.Бакланов. Екатеринбург: ГОУ ВПО УГТУ-УПИ, 2005. 93 с.

ISBN

В учебном пособии рассматриваются вопросы, связанные с защитой компьютерной информации в автономных компьютерных системах, работающих под управлением операционных систем семейства Linux. Основное внимание уделено защите информации на уровне файловых систем. Детально излагается структура распространенных файловых систем EXT2FS и EXT3FS, что позволяет приобрести знания и умения по восстановлениию утерянной информации. Рассматриваются разнообразные вопросы, имеющие отношение к компьютерной безопасности и администрированию операционных систем. В учебное пособие включено пять лабораторных работ по исследованию защитных механизмов операционной системы Linux.

Учебное пособие предназначено для студентов, обучающихся по специальностям 075200 - Компьютерная безопасность, 075500 - Комплексное обеспечение информационной безопасности автоматизированных систем, 075600 - 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.

УДК 661.3.066
ББК 32.973.26

ISBN

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Операционные системы клона UNIX приобретают все большую популярность среди пользователей в различных сферах их деятельности. Надежность, простота и практически неограниченные возможности этих систем всегда привлекали компетентных специалистов, а разработка средств графического интерфейса и программных приложений для обработки текста и графики, совместимых с приложениями офисного пакета Microsoft, постепенно стирают пока еще имеющиеся различия в функциональных возможностях операционных систем UNIX и Windows.

У операционных систем UNIX/Linux есть ряд особенностей, которые выгодно отличают их от других универсальных операционных систем:

1. Системы клона UNIX эксплуатируются во всем мире уже четвертое десятилетие и успешно пережили не одно поколение компьютеров, доказав возможность работы на различных аппаратных платформах. Хорошая 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 надежной защиты компьютерной информации.
3. Все объекты, с которыми приходится работать пользователю, являются файлами. При этом файлами называются и аппаратные компоненты, типа блочных или символьных устройств, и объекты межпроцессного взаимодействия, как именованные каналы. Этим достигается общий подход к работе с разнообразными объектами, включая применение политики безопасности.
4. Операционные системы UNIX разработаны программистами для программистов, и опытный пользователь с правами администратора системы имеет возможность реализовывать на своем компьютере с ОС UNIX 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 прочитать и записать информацию на машинный носитель, совершенно не обращая внимания на файловую систему, отредактировать содержимое оперативной памяти, изменить настройки выполняющегося процесса и т.д.

Операционные системы UNIX представляют интерес для защиты компьютерной информации еще по одной причине. Благодаря своей изощренности и способности реализовать любые возможности, эти системы стали излюбленным

программным оружием хакеров. Компьютеры с ОС UNIX очень часто используются для подготовки и проведения сетевых атак: сканирования сети, перенаправления трафика, сниффинга, 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 UNIX не использовались в созидательной деятельности, они заслуживали бы изучения в среде специалистов по компьютерной безопасности в качестве программного средства совершения преступлений.

Никто не отрицает, что при своей простоте операционные системы UNIX довольно сложны для администрирования. Отсутствие избыточных функций порой заставляют администратора идти на различные ухищрения, 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 возможных ошибок при управлении операционной системой весьма большой. Поэтому труд администратора ОС UNIX нельзя назвать простым.

В последнее время большую популярность приобрел еще один клон системы UNIX, который получил название Linux по имени своего уже легендарного создателя – финского студента Линуса Торвальдса. Причиной особой популярности этой системы является свободно распространяемое программное обеспечение, которое можно скопировать с различных сайтов Интернет. 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 некоторых государств, которые разрабатывают на базе Linux защищенные версии операционных систем, предназначенные для обработки конфиденциальной информации.

В рамках данного учебного пособия предполагается ограничиться рассмотрением возможностей операционной системы, инсталлированной на автономном персональном компьютере. Операционные системы Linux являются сетевыми, но вопросам защиты информации в сетях Linux в настоящее время посвящают свое внимание много авторов. Если это потребуется в дальнейшем, автор рассмотрит вопросы администрирования и защиты сетей Linux отдельное учебное пособие.

Рассмотрение сущностей операционной системы будет проводиться в логическом порядке: субъекты (пользователи), процессы и объекты (файловая система).

1. ПОЛЬЗОВАТЕЛИ И ИХ ПРАВА

Каждому пользователю при регистрации присваивается уникальный идентификатор (UID – User ID). Пользователей в системе может быть довольно много. Для их нумерации зарезервировано два байта, что дает $2^{16} = 65536$ пользователей. 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.

После успешной аутентификации система помнит пользователя за каждым терминалом только по номеру и передает этот номер каждому созданному процессу и файлу.

Пользователь с UID = 0 носит имя root и является для системы администратором (суперпользователем). Можно создать произвольное число учетных записей с нулевым идентификатором, и каждый из зарегистрированных пользователей будет обладать правами администратора. Как говорят, root – это не право, а возможность не считаться ни с какими правами. Существует очень небольшое число команд, которые система откажется выполнять от имени суперпользователя, и в этом усматривается определенная угроза компьютерной безопасности, связанная с человеческим фактором. С одной стороны, достаточно много ответственных действий в системе можно выполнять только с правами администратора. С другой, привычка постоянно работать в системе с полными правами является пагубной. Администратор тоже человек, и ему свойственно ошибаться. Иногда незначительная ошибка при вводе команды может привести к фатальным последствиям, поскольку Linux не является нудной системой и без необходимости не спрашивает «Вы уверены в том, что хотите удалить этот каталог?». Можно привести в качестве примера две команды:

```
rm -rf /home/user1/file1
```

Этой командой администратор хотел удалить один файл.

```
rm -rf / home/user1/file1
```

Но в команде ошибочно введен лишний пробел, в результате чего будет уничтожен весь корневой каталог.

Пользователей можно объединять в группы, которым тоже присваиваются символьные имена и уникальные числовые идентификаторы GID (Group ID). Точнее, при правильном планировании вначале создаются группы, а затем в них включаются пользователи.

Для регистрации пользователей и создания групп тоже требуются полномочия администратора. Новая группа создается командой **groupadd** (краткий справочник по командам Linux приведен в Приложении 1. Полный электронный справочник по командам доступен при вводе команды **man command_name**). Пользователь может являться членом большого числа групп – $2^{16} = 65536$. Но у

каждого конкретного файла может быть только один владелец и только одна основная группа.

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 идентификаторы, пароль пользователя, срок его годности, домашний каталог пользователя, имя командного интерпретатора. Если какие-либо из этих параметров администратор явно не указывает, они подставляются системой по умолчанию.

Утилита **useradd** достаточно трудна в запоминании, поскольку в командной строке в виде опций необходимо ввести все требуемые параметры. Эта утилита во многих дистрибутивах Linux отличается своей «кривизной»: она «забывает» вычислять хэш-функцию и записывает пароль в теневой файл /etc/shadow в открытом виде! В результате созданной учетной записью просто невозможно воспользоваться, не говоря уже о возросшей опасности несанкционированного доступа к открытым паролям. Поэтому при создании учетных записей с помощью этой утилиты пароль пользователю следует задавать отдельной командой **passwd,** которая рассмотрена ниже.

Сценарий **adduser** позволяет после ввода команды просто отвечать на запросы программы, вводя данные в интерактивном режиме. При этом программа подсказывает значения по умолчанию [в квадратных скобках]. Ввод параметра сопровождается нажатием <Enter>. Необязательные параметры можно игнорировать, просто нажимая <Enter>. Но в некоторых дистрибутивах ОС Linux командный файл **adduser** отсутствует.

В графической оболочке системы тоже имеются свои варианты программ с оконным интерфейсом, позволяющие создать новую учетную запись и сделать этот процесс более наглядным. Но ортодоксально мыслящие администраторы вообще не склоны признавать графические оболочки.

При необходимости администратор может напрямую редактировать файлы паролей и групп. Текстовые файлы group, passwd и shadow, располагащихся 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.

Наиболее простым является файл /etc/group. В каждой записи указывается только имя группы, ее идентификатор и символ присутствия пароля (пароли на группы, как правило, не назначаются). Если группа является дополнительной, в ее строке через запятые указываются идентификаторы всех дополнительных пользователей.

Файл /etc/passwd является таблицей, каждая строка которой представляет отдельную учетную запись, состоящую из 7 полей: регистрационное имя, признак пароля, идентификатор пользователя, идентификатор группы, дополнительная информация, "домашний" каталог, имя командного процессора. Обратите внимание на характерные разделители полей в виде двоеточия.

```
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:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
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:x:99:99:Nobody:/:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/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
```

Листинг 1. Содержимое файла /etc/passwd

Сами хэшированные пароли в файле /etc/passwd не хранятся. Если в учетной записи пользователя вместо пароля стоит символ x, это указывает на то, что хэшированный пароль находится в другом файле - /etc/shadow.

Файл /etc/shadow представляет собой таблицу, каждая строка которой состоит из 9 полей, разделенных двоеточиями:

- регистрационное имя пользователя или псевдопользователя,
- хэшированный пароль,
- число дней с полуночи 1.01.70 г. до дня последнего изменения пароля,
- минимальное число дней действия пароля со дня его последнего изменения,
- максимальное число дней действия пароля,
- число дней до того дня, когда система начнет выдавать предупреждения о необходимости смены пароля,
- число дней со времени обязательной смены пароля до блокировки учетной записи,
- день блокировки учетной записи.

Последнее, девятое поле, зарезервировано и не используется.

```
root:$1$Yj7IcY8O$oCX9V9QYipDtYflbfOOBE1:12496:0:99999:7:::
bin:*:12495:0:99999:7:::
daemon:*:12495:0:99999:7:::
adm:*:12495:0:99999:7:::
lp:*:12495:0:99999:7:::
sync:*:12495:0:99999:7:::
shutdown:*:12495:0:99999:7:::
halt:*:12495:0:99999:7:::
mail:*:12495:0:99999:7:::
news:*:12495:0:99999:7:::
uucp:*:12495:0:99999:7:::
operator:*:12495:0:99999:7:::
games:*:12495:0:99999:7:::
gopher:*:12495:0:99999:7:::
ftp:*:12495:0:99999:7:::
nobody:*:12495:0:99999:7:::
vcsa:!!:12495:0:99999:7:::
rpm:!!:12495:0:99999:7:::
xfs:!!:12495:0:99999:7:::
rpc:!!:12495:0:99999:7:::
mailnull:!!:12495:0:99999:7:::
smmsp:!!:12495:0:99999:7:::
gdm:!!:12495:0:99999:7:::
nscd:!!:12495:0:99999:7:::
ntp:!!:12495:0:99999:7:::
pcap:!!:12495:0:99999:7:::
```

Листинг 2. Содержимое файла /etc/shadow

Обычно 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 еще не истек). **Passwd** – это одна из утилит, которая запускается обычным пользователем, а выполняется с правами администратора, поскольку ей приходится записывать новые данные в теневой файл /etc/shadow, доступ к которому разрешен только пользователю root. Такие исполняемые файлы имеют так называемый эффективный идентификатор SUID, речь о котором пойдет ниже. Программа **passwd** запрашивает у пользователя старый пароль, а затем требует ввести новый. На систему возлагается обязанность проверять вновь введенный пароль по словарю и по длине. Если введенный пароль окажется слишком простым, программа предупредит пользователя об опасности и запросит у него другой пароль.

Для модификации существующей учетной записи можно использовать команду **usermod**, которая имеет такой же синтаксис, что и команда **useradd**. В команде указываются только те опции, которые подлежат изменению.

Команда **userdel** позволяет администартору удалить учетную запись. До удаления учетной записи пользователь должен завершить сеанс работы. Обычное задание команды без опций:

`userdel user_name` - удаляет учетную запись пользователя, сохраняя его каталоги, которые могут содержать файлы, необходимые учреждению. При задании опции **-r** учетная запись удаляется вместе с каталогами и файлами пользователя.

Пользователи получают во владение созданные ими объекты файловой системы и имеют определенные системой или администратором права доступа к существующим объектам. Всего существует три вида доступа:

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

Остальные права образуются путем комбинации первичных прав.

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

- владелец файла,
- члены группы, в которую входит владелец. Поскольку один пользователь может являться членом многих групп, здесь имеется в виду основная, первичная группа,
- все остальные зарегистрированные пользователи, за исключением владельца файла и 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** (list - список). Результат выполнения этой команды представлен в листингах 5 и 15. 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. Например:

`r - x` = 101 = 5

`- w x` = 011 = 3

`r - -` = 100 = 4 и т.д.

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

```
mkdir –m 1555 /home/user1
```

В отношении каталогов права доступа интерпретируются несколько иначе. Право на чтение в каталоге дает возможность только посмотреть имена файлов, записанные в нем. Для вывода расширенной информации о файлах (например, с помощью команды **ls –l**) необходимо уже право исполнения. По отношению к каталогу право на исполнение – это право войти в каталог с целью поиска файлов или пройти через него, если он является промежуточным объектом в

полном пути к искомому файлу. Большинство каталогов в системе предоставляют всем зарегистрированным пользователям права на чтение и исполнение. Право записи в каталог одновременно означает возможность создавать в нем новые файлы, а также копировать или перемещать в него существующие файлы. Можно создать каталог с правами записи и исполнения – в этом случае мы получим так называемый «темный» каталог, записывать в который можно, но что в нем находится – не видно.

Отдельной команды для создания обычного файла не предусмотрено, но файлы можно создавать различными способами, например, путем перенаправления вывода в отсутствующий файл (о перенаправлении вывода тоже будет сказано ниже). При этом права доступа к вновь создаваемым файлам определяются маской, которая задается с помощью команды **umask.** Вызов этой команды без аргументов приводит к выводу текущего значения маски. Значение маски – тоже восьмеричное число, которое вычитается из 0777 для исполняемого файла или из 0666 – для неисполняемого. Например, для исполняемого файла **umask** = 022 означает режим доступа 755 (111 101 101 = rwxr-xr-x). Текущее значение **umask** для администратора и обычных пользователей записывается в один из конфигурационных файлов.

Права на доступ к уже существующему файлу можно изменять. Администратор может изменить права доступа к любому файлу, пользователь – только к своим. Производится это с помощью утилиты **chmod** (change mode – изменить режим). У этой команды несколько форм записи, с которыми можно ознакомиться в кратком справочнике. Более компактная форма –

`chmod XXXX file_name`

Обычно владельцу файла предоставляются полные права, а членам его группы и всем остальным – по необходимости. Но владелец файла вправе ограничить себя в правах. Например, задав режим доступа **chmod 077 file_name** он предоставляет полные права всем зарегистрированным пользователям, кроме себя. При этом программа, проверяющая права доступа, не относит владельца ни к его группе, в которой он зарегистрирован, ни к другим пользователям. 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 него нет прав, – ему будет отказано в доступе, хоть он и является членом своей группы. Но пользователь, допустивший такую оплошность, может легко исправить положение, переназначив права доступа в свою пользу – ведь он остается владельцем этого файла и может применить к нему команду **chmod**.

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

`chown user file_name` – передача прав на файл другому владельцу.

Пользователям не разрешается передавать свои файлы другим пользователям, включая администратора, поскольку это может повлечь за собой несколько информационных атак. Во-первых, передавая свои файлы администратору, пользователь может создать предпосылки для случайного запуска вредоносной программы с правами суперпользователя. Создав несколько тысяч объемных файлов и передав права на них другому пользователю (например, из числа недругов), пользователь может переполнить дисковую квоту атакуемого и вызвать отказ в его обслуживании. Поэтому практически во всех дистрибутивах Linux передавать права на объекты файловой системы может только администратор.

В современных версиях ядра можно задавать дополнительные права на файлы. Это делается с помощью утилиты:

`chattr +(-) option file_name` – установка или отмена дополнительных атрибутов файла. В качестве опций можно задать:

i – блокирование любых изменений файла,

a – запрет любых операций, кроме добавления данных,

c – автоматическое сжатие и декомпрессию при записи/чтении файла,

s – гарантированное стирание блоков данных при удалении файла.

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

Пользоваться этой утилитой также разрешается только администратору. При этом администратор может установить запрет, который сам преодолеть будет не в силах (вопреки базовым правилам UNIX). Например, установив дополнительный атрибут +i, администратор не сможет ни удалить файл, ни записать что-либо в него, ни изменить права доступа. Этим весьма полезным свойством необходимо пользоваться в отношении наиболее ответственных исполняемых файлов.

Дополнительные атрибуты файла командой **ls** не выводятся, и для их чтения необходимо воспользоваться командой **`lsattr file_name.`**

Пользование правами администратора само по себе потенциально опасно для системы, поэтому он в обыденных ситуациях, когда вводить привелегированные команды не требуется, должен использовать права обычного пользователя. Для этого администратор может зарезервировать для себя несколько учетных записей, либо использовать зарегистрированные имена других пользователей. Изменить права доступа можно с помощью команды **su** (substitute user **-** подстановка пользователя).

Если администратор вводит команду **su** с именем зарегистрированного пользователя, он становится им мгновенно, без запроса пароля. 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 приходилось наблюдать, что возврат прав администратора не сопровождался запросом пароля – это явная угроза безопасности, допущенная программистами.

Использование таких команд, как **su**, находится под пристальным вниманием системы и по умолчанию протоколируется в один из файлов аудита. Ис-

полняемый 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 вызове этой команды, программа может перехватить многие важные пароли, включая пароль суперпользователя.

Довольно часто у администратора возникает потребность доверить одному из пользователей выполнение какой-либо операции, на которую требуются права root. Но для этого администратор должен доверить другому святая святых – свой пароль! Для того чтобы избежать таких ситуаций, была разработана еще одна утилита, позволяющая выборочно предоставлять определенным пользователям конкретные права. Эта утилита именуется **sudo** и вводится с именем команды, которую надо исполнить. Названия команд придумываются администратором и записываются в специальный конфигурационный файл **/etc/sudoers**, который должен быть недоступным для всех остальных. В этом текстовом файле присутствуют строки, связывающие определенных пользователей с определенными ответственными командами. Пользователь, наделенный временными полномочиями, должен знать, какую команду ему разрешено выполнять с правами администратора и как эта команда пишется. Система, получив команду **sudo**, запрашивает у пользователя пароль, но это должен быть его собственный пароль, а не пароль root. После проверки пароля программа сравнивает имя пользователя с введенной командой, и при наличии соответствующей записи выполняет эту команду. Вызов команды sudo также по умолчанию протоколируется системой в одном из журналов аудита.

2. ПРОЦЕССЫ

Операционные системы клона UNIX являются многозадачными системами общего назначения, и управление процессами в них напрямую связано с обеспечением безопасности обрабатываемой компьютерной информации. Так, запуск вредоносной программы немыслим без создания нового процесса, блокирование компьютерной системы реализуется посредством повышения приоритета «жадного» процесса, нарушение целостности данных осуществляется путем записи информации в чужое адресное пространство и др.

Процесс – это 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 тоже создает множество независимых процессов. Поэтому правильнее назвать процессом ***выполняющийся экземпляр программы***.

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

1. Системные процессы. Они не имеют собственных исполняемых файлов и запускаются из ядра операционной системы. Системные процессы стартуют при загрузке операционной системы и выполняются в течение всего сеанса работы. Они запускаются от имени операционной системы и обладают соответствующими правами. Эти процессы не общаются с пользователем, и он может их обнаружить только просматривая список процессов. Соответственно, пользователь, если он не является администратором системы, не имеет права распоряжаться этими процессами, посылая им сигналы.

Есть один процесс, который по своей сути является системным, но запускается не из ядра, а из отдельного исполняемого файла с именем init. Этот процесс является прародителем всех остальных процессов.

2. Демоны. Это процессы, которые также не общаются с пользователем напрямую и не могут управляться им. Они выполняются в фоновом режиме и отвечают за предоставление сервисов: печать, доступ к сетевым ресурсам и пр. Но каждому демону соответствует исполняемый файл, который обычно запускается автоматически, при загрузке системы.
3. Пользовательские процессы. Запускаются из исполняемого (бинарного) файла пользователем. Они могут выполняться в интерактивном или фоновом режимах и срок их выполнения ограничен продолжительностью сеанса работы пользователя. Пользовательские процессы наследуют идентификатор пользователя и, как правило, имеют соответствующие права на доступ к объ-

ектам. Но некоторые пользовательские процессы могут при выполнении иметь больше прав, чем имеет создавший их пользователь – об этом подробнее будет сказано несколько позже.

Самый важный пользовательский процесс – командный интерпретатор, обеспечивающий диалоговый режим 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 доступ 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 (вычислительные процедуры, операции со строками и др.) не требуют каких-либо исключительных привилегий и могут выполняться в пользовательском режиме. Но создание или удаление объектов файловой системы, обращение к устройствам требует системных привилегий. Если пользовательскому процессу не хватает прав, он обращается к ядру с системным вызовом и продолжает выполнение в другом контексте уже в привилегированном режиме ядра.

Управление процессами осуществляется частью ядра, именуемой планировщиком задач или просто планировщиком (scheduler). В системах UNIX реализована принудительная (или по-другом, вытесняющая) многозадачность, когда планировщик задач передает управление следующему процессу, не «спрашивая» согласия на это у предыдущего процесса. Для того чтобы переключить центральный процессор с одной задачи на другую, тоже 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. Пользовательские процессы заведомо обладают меньшими правами по отношению к объектам доступа. Однако

на их приоритетах это не сказывается. 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 одинаковые приоритеты. Избыточный приоритет – весьма опасная вещь, способная приводить к атакам на отказ в обслуживании.

Каждому процессу, кроме приоритета, может быть присвоен так называемый фактор уступчивости – коэффициент, который по умолчанию равен нулю. Положительное значение этого фактора свидетельствует о понижении приоритета и наоборот.

Процесс с привилегиями суперпользователя может запускать процессы с отрицательным фактором уступчивости или понижать это значение у выполняющегося процесса. Обычные пользователи и их процессы не могут захватывать лишнее время у центрального процессора.

Как это не покажется странным, в некоторых случаях можно атаковать компьютерную систему, преднамеренно уменьшая свой приоритет [2,5].

Существует категория так называемых «жадных» программ, которые способны захватить 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. «Жадная» программа может затребовать непомерно большой объем виртуальной памяти и вызвать ее дефицит.

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 громадное 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.

Процессы в UNUX «размножаются» путем бинарного деления. Для этого запускающий процесс должен создать своего «двойника», вызывая системный запрос, именуемый **fork** (дословно – вилка). Выполняя этот запрос, система создает дочерний процесс, являющийся почти полной копией родительского, но имеющий свой идентификатор PID и выполняющийся в собственном адресном пространстве. Дочерний процесс сохраняет значение идентификатора родительского процесса Parent Process ID (PPID). «Родственники» могут взаимодействовать друг с другом посредством сигналов, общего адресного пространства и др.

Пользовательским процессам присваивается еще один идентификатор – UID, уникальный номер пользователя, который их запустил. Пользователь не может манипулировать файловыми объектами непосредственно, и это делает за

него программа. Обычные пользовательские программы имеют право делать только то, что разрешено пользователю, запустившему их. Например, пользователь, запустивший процесс **`ls -la /root`** с целью просмотра содержимого подкаталога администратора, получит сообщение “permission denied”. На самом деле отказ в доступе получил процесс, запущенный от имени пользователя. Но тот же процесс, запущенный с правами root, будет выполнен безукоризненно.

Некоторые исполняемые файлы имеют так называемые эффективные идентификаторы пользователя EUID. Файл с таким идентификатором должен иметь установленный бит SUID (см. главу 3). Эффективный идентификатор позволяет процессу выполнять действия не от имени пользователя, запустившего процесс, а от имени владельца файла. Например, пользователям в ходе работы разрешается сменить собственные пароли (если минимальный срок действия старого пароля еще не истек). В принципе, их можно было лишить такого права.

Для наблюдения за активными процессами пользователь должен ввести команду **ps** (process status). В листинге 3 изображен список процессов, вызванный с атрибутами **–el**. Опция –e отображает все процессы, l – означает расширенный (длинный) вывод данных.

```
S   UID   PID PPID C PRI NI   TIME CMD
S     0     1     0  0  24   0 00:00:04 init
S     0     2     1  0  24   0 00:00:00 [keventd]
S     0     3     1  0   5  19 00:00:00 [ksoftirqd_CPU0]
S     0     8     1  0  14   0 00:00:00 [bdflush]
S     0     4     1  0  24   0 00:00:00 [kswapd]
S     0     5     1  0  24   0 00:00:00 [kscand/DMA]
S     0    10     1  0  14   0 00:00:00 [mdrecoveryd]
S     0    19     1  0  24   0 00:00:00 [kjournald]
S     0    71     1  0  14   0 00:00:00 [khubd]
S     0   647     1  0  24   0 00:00:00 syslogd -m 0
S     0   651     1  0  24   0 00:00:00 klogd -x
S    32   663     1  0  24   0 00:00:00 [portmap]
S     0   768     1  0  24   0 00:00:00 [sendmail]
S    51   777     1  0  14   0 00:00:00 [sendmail]
S     0   787     1  0  24   0 00:00:00 gpm -t imps2 -m /dev/psaux
S     0   796     1  0  24   0 00:00:00 crond
S     0   807     1  0  24   0 00:00:00 cupsd
S    43   889     1  0  24   0 00:00:00 [xfs]
S     0   898     1  0   0  19 00:00:00 anacron -s
S     0   912     1  0  24   0 00:00:00 login -- root
S     0   917     1  0  17   0 00:00:00 /sbin/mingetty tty6
S     0   918   912  0  17   0 00:00:00 /bin/sh /usr/X11R6/bin/startx
S     0   968   918  0  19   0 00:00:00 xinit /etc/X11/xinit/xinitrc --
S     0   981   968  0  24   0 00:00:00 /usr/bin/gnome-session
S     0   997   981  0  24   0 00:00:00 ssh-agent /etc/X11/xinit/Xclients
S     0  1008     1  0  24   0 00:00:00 /usr/libexec/gconfd-2 11
S     0  1010     1  0  24   0 00:00:00 /usr/libexec/bonobo-activation-server --
S     0  1012     1  0  24   0 00:00:00 gnome-settings-daemon --oaf-activate-iid
S     0  1017   749  0  24   0 00:00:00 [fam]
R     0  1090     1  0  23   0 00:00:01 gnome-terminal
S     0  1091  1090  0  16   0 00:00:00 [gnome-pty-helpe]
S     0  1092  1090  0  24   0 00:00:00 bash
R     0  1129  1092  0  24   0 00:00:00 ps -eo s,uid,pid,ppid,c,pri,ni,time,cmd
```

Листинг 3. Фрагмент таблицы процессов, выводимой командой **ps**.

Символом S обозначен статус процесса. Могут встречаться следующие значения:
O или SW – процесс выполняется,
S – процесс не активен и ожидает наступления какого-либо события,
T – процесс остановлен,
R – разрешен к выполнению

Параметры PID, PPID и UID уже ранее упоминались. В столбце PRI отображается текущий динамический приоритет процесса. Параметр NI указывает на относительный приоритет, TIME – суммарное время выполнения процесса центральным процессором. Столбец CMD указывает имя команды (программы), соответствующей данному процессу.

2.1. Средства взаимодействия между процессами

Процессы могут обмениваться друг с другом информацией через специальные файлы или фрагменты памяти, предоставляемые операционной системой. Эти средства взаимодействия называются сигналами (signals), сообщениями (messages), каналами (pipes), семафорами (semaphores) или просто разделяемой памятью. Здесь мы ограничимся только рассмотрением сигналов и каналов.

На уровне пользователя и администратора для межпроцессного взаимодействия чаще приходится пользоваться сигналами. Сигнал – это своего рода команда, передаваемая процессу. Как это не покажется странным, для посылки сигнала используется команда **kill** (дословно – убить). Действительно, на практике сигналы чаще всего используются для принудительной остановки какого-либо процесса, вышедшего из-под контроля. Сигналы распознаются по номерам (всего их в Linux-системах существует несколько десятков). Для каждого сигнала в программах предусматривается специальный обработчик, определяющий, что должно произойти с процессом при получении конкретного сигнала.

Для исследования 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 его 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 **killall**.

2.2. Перенаправление ввода/вывода

Согласно идеологии UNIX стандартные потоки ввода и вывода можно перенаправлять. По умолчанию стандартный ввод и стандартный вывод ассоции-

рованы с консолью (ввод с клавиатуры и вывод на монитор). Но часто возникает необходимость выводить информацию не на экран (пользователя в это время у компьютера может не оказаться, либо выводимая информация окажется чрезмерно велика для ее чтения с экрана), а в файл, который можно прочитать или распечатать потом. Именно так организуется аудит опасных событий.

Выводить информацию можно двумя командами. Например:

```
ls -la /home/user1 > /etc/syslog.ls
```

Данная команда записывает список файлов пользователя в текстовый файл. Если этот файл не существует, то он создается. Если он существует, его прежнее содержимое вначале стирается.

```
logger >> /etc/syslog.ls
```

В этом случае выводимая информация дописывается к содержимому файла аудита. 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 буфером между командным интерпретатором и драйверами устройства).

```
cat file_name > /dev/fd0
```

Читаем ранее созданный образ дискеты и копируем его на сменный носитель.

```
cat file_name > /dev/lp0
```

Выводим содержимое файла на принтере, подключенном к первому параллельному порту.

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

Воспроизводим звуковой файл через звуковой адаптер.

Аналогично можно изменить стандартный ввод информации, которым по умолчанию является клавиатура. Так, 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 файла и выводить результаты в другой файл.

2.3. Каналы (неименованные каналы)

Каналом в UNIX называется однонаправленное логическое устройство, предназначенное для передачи данных от одного процесса другому. По своей сути канал представляет собой буфер памяти небольшого размера, в который один процесс может записывать данные, а другой – эти данные читать. И запись, и чтение данных осуществляются последовательно: данные всегда чита-

ются в том порядке, в каком они были записаны. Канал может быть использован как средство связи между родственными процессами.

На языке командных интерпретаторов канал обозначается символом |. В некоторых источниках он именуется конвейером и служит средством группирования команд. Наиболее часто канал используется при выводе на экран файлов большой длины. Так, чтобы последовательно просмотреть листинг файлов в каталоге большого объема, используют команду **`ls -la /bin|more`**. 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.

Еще несколько примеров использования каналов:

```
cat file_name|wc
```

Утилита cat читает текстовый файл file_name и передает последовательный поток символов программе wc, которая подсчитывает число строк, слов и символов в файле,

```
ps -ef | head -20
```

Утилита ps выводит таблицу процессов, а утилита head отображает первые 20 строк этой таблицы,

```
dd if=/dev/fd0 | grep "Linux"
```

Утилита dd читает по секторам дискету, а grep ищет в считываемых данных строку “Linux”.

Как уже указывалось, емкость канала ограничена размером буфера памяти, и, если передающий процесс записывает в него данные быстрее, чем принимающий читает их, канал переполнится. В этом случае передающий процесс будет заблокирован. Читать из пустого канал тоже нельзя – будет заблокирован принимающий процесс.

2.4. Именованные каналы

У этих каналов есть имя, и они почти подобны обычным файлам последовательного доступа. 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
```

Именованный канал будет создан в каталоге tmp. Любой пользователь (псевдопользователь) системы обладет правом на чтение, записи и исполнения

этого каталога. К созданному именованному каналу можно обращаться как к обычному файлу. Для того чтобы убедиться в этом, следует войти в систему с двух текстовых консолей. Затем в командной строке первой консоли ввести команду чтения из именованного канала, внашем примере **`cat < /tmp/fifo1`**. А во второй 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.

2.5. Файловая система /proc

В системе Linux присутствует виртуальный файловый объект, именуемый каталогом **/proc (process)**. Точнее сказать, этот объект существует только во время работы системы в оперативной памяти компьютера. Это виртуальная файловая система, которая обеспечивает связь с ядром и монтируется в каталогу **/proc**. Виртуальный каталог **/proc** предназначен для предоставления текущих параметров о компьютерной системе и представляет большой интерес, в том числе с позиций компьютерной безопасности. Многие из утилит, выводящие информацию о системе (например, команда **ps**), берут исходные данные именно из этого каталога.

В виртуальной файловой системе /proc размер почти всех файлов равен нулю, и любой пользователь, включая администратора, 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 и 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 идентификатором 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. 1).

Таблица 1

Содержимое «числовых» подкаталогов каталога /proc

Имя «файла» в подкаталоге /proc/PID	**Содержимое «файла»**
cmdline	Список аргументов командной строки процесса (параметры, передаваемые программе). Список представлен одной строкой, в которой аргументы отделяются друг от друга нулевыми символами
cwd	Ссылка на каталог, из которого была выполнена команда, запустившая данный процесс

Продолжение табл. 1

environ	Переменные окружения (USER, HOME, PATH и др.). Элементы списка разделяются нулевыми символами.
exe	Ссылка на исполняемый файл процесса
fd	Каталог, содержащий записи о файлах, открытых данным процессом. Каждая запись – это символическая ссылка на файл или устройство.
maps	Исполняемые файлы и библиотеки, отображенные в память процесса
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: • общий размер процесса в Мб • размер резидентной части процесса • размер совместно используемой памяти • размер сегмента кода • размер загруженных библиотек • объем памяти стека • 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.

2.6. Запуск программ в интерактивном режиме

После прохождения процедуры аутентификации пользователя система запускает экземпляр командного интерпретатора, который обеспечивает диалоговый режим человека и машины. Как уже указывалось, существует три основных типа интерпретаторов, работающих в режимах командной строки, текстовом меню или графического интерфейса. В любом случае система ожидает от зарегистрированного пользователя ввода команды.

Программа, которая общается с пользователем и выполняет введенные им команды, называется командным интерпретатором или оболочкой (**Shell**). Небольшое число команд исполняется самой оболочкой, и эти команды называются внутренними. Подавляющее большинство команд являются внешними, и имя введенной команды тождественно имени какого-нибудь исполняемого файла или сценария. Исполняемые файлы обычно располагаются в двух каталогах: /bin и /sbin, хотя запустить процесс можно из любого каталога, на который у пользователя есть право доступа. Вызывать команды можно, задавая абсолютный путь к ее исполняемому файлу, либо пользуясь коротким именем файла. Найти нужный файл по его короткому имени программе-оболочке помогает переменная окружения PATH (дословно – дорожка, путь). В ней обычно поименованы каталоги /din, /sbin, /usr/bin, /usr/local/bin, разделенные двоеточием. Для администратора в этом перечне должен быть исключен текущий каталог, обозначаемый одной точкой «.», поскольку это потенциально опасно.

Пользователь может управлять системой в режиме командной строки, и в этом случае он работает с одной из оболочек, имеющих название **Born shell** (/bin/sh), **C shell** (/bin/csh), **Korn** shell (/bin/ksh), **Bash** (/bin/bash). Пользователи, привыкшие к удобству, предпочитают файловый менеджер **Midnight Commander.** Наконец, современные дистрибутивы систем поддерживают графические оболочки типа **Gnom** или **KDE**.

Введенная команда должна состоять из трех частей:

- имени самой команды,
- опций или флагов,
- входных данных.

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 эквивалентыми:

```
ps –e –l –f = ps –elf
```

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. Интерпретаторы понимают и длинные, и короткие опции. Но запускаемая программа тоже должна понимать опции обоих видов.

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

```
dd if = /dev/fd0 of = /mnt/floppy/fda count = 10 skip = 1
conv = noerror,sync
```

где опции и входные данные чередуются.

Команды могут вводиться и 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 приглашение для ввода следующей команды. Пользователь может запустить произвольное число таких процессов.

В одной строке можно ввести несколько команд, разделяя их символом «**;**», например:

```
clear; pwd; date
```

Если разделить две команды в одной строке символами **&&**, то вторая команда будет выполнена только при благополучном (безошибочном) завершении первой. Если анализировать код ошибок, то успехом считается возврат нулевого кода, а неудачей – все остальные значения. В примере 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, если надо запустить вторую команду при ошибочном завершении первой. 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.

Пользователю нет необходимости многократно вводить одни и те же команды. С помощью клавиши ↑ можно вернуться к предыдущей команде, а нажимая ее многократно, можно «пролистать» список команд в обратную сторону на нужное число позиций. То же самое можно сделать с помощью команды **history** – при этом выводится перечень ранее введенных команд (по умолчанию запоминается список из 1000 команд). Этот список хранится для каждого зарегистрированного пользователя в отдельном текстовом файле в его домашнем каталоге.

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

```
passwd
dd if=/dev/fd0 of=floppy1 count=2800 conv=noerror,sync
fdisk -l /dev/hda6
ls -li /
lde -i 131329 /dev/hda6
mc
lde -b 0x00040203 /dev/hda6|more
mc
debugfs /dev/hda6
cd /home
ls
cd /etc
ls -li
debugfs -R stats /dev/hda6
```

Листинг 4. Фрагмент перечня команд пользователя выведенных командой history

3. ФАЙЛОВЫЕ СИСТЕМЫ EXT2FS И EXT3FS

Файловая система представляет собой способ логической организации внешней и оперативной памяти компьютера. Современные многозадачные операционные системы, работающие в защищенном режиме, а также интеллектуальные контроллеры магнитных дисков не позволяют работать с памятью на физическом уровне. Поэтому для администратора очень важно разобраться в структуре файловых систем.

Файловые системы EXT2FS и EXT3FS отличаются только наличием журнала транзакций. В остальном они имеют идентичную структуру и могут взаимно преобразовываться на этапе монтирования. В файловой системе Linux основным логическим объектом является файл. Все объекты, включая устройства ввода/вывода информации и каналы межпроцессного взаимодействия, называются файлами. Определено 7 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 структурированная последовательность байт. Для операционных систем семейства UNIX не имеют значения, принятые в ОС Windows* и MS DOS, расширения файлов, по которым можно судить об их типе. Однако одна из утилит, именуемая **file**, умеет различать довольно много разновидностей файлов по их «магическим» числам и характерному формату.

Дисковое пространство выделяется файлам целыми блоками. Блок является адресуемой единицей дискового пространства и может иметь размер 1024, 2048 или 4096 байт. В ОС Windows* похожей величиной является кластер. Нетрудно заметить, что размер блока кратен размеру одиночного сектора на диске (512 байт). Непосредственно с секторами работать в файловой системе не приходится, но некоторая выводимая утилитами информация 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, особенно при наличии большого числа маленьких файлов. В современных дистрибутивах ОС Linux по умолчанию используется размер блока в 4 Кб, что ускоряет процессы подкачки в виртуальной памяти (размер страницы подкачки тоже 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 перемеще-

ний блока магнитных головок накопителя на жестких дисках при обращении к файлам одного каталога.

Просматривая информацию об основных каталогах файловой системы (листинг 5), выведенной с помощью команды **ls –li /**, следует обратить внимание на то, как сильно различаются номера индексных дескрипторов каталогов. Это косвенно указывает на то, что каталоги распределены по всему диску.

```
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+foun
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
```

Листинг 5. Информация об основных каталогах файловой системы Linux

Следует обратить внимание на размер большинства каталогов. Его минимальное значение - 4 Кб, что соответствует размеру одного логического блока. Поскольку каталог по сути является таблицей соответствия имен файлов и их индексных дескрипторов, подавляющее большинство каталогов вполне вмещаются в этот объем. Лишь два каталога 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, а каталог /sbin содержит очень большое количество утилит. Каталог /proc, имеющий нулевой размер, является псевдокаталогом, он расположен в оперативной памяти и на дисковом пространстве места не занимает. Более подробно структура файловых записей в каталоге будет рассмотрена ниже.

Загрузчик (Linux Loader)	**Группа блоков 1**	**Группа блоков 2**	…….	**Группа блоков N**

Рис. 1. Группы блоков на логическом разделе Linux

Блоки объединяются в группы блоков. Группы блоков в файловой системе и блоки внутри группы нумеруются последовательно, начиная с единицы. Пер-

вый блок на диске имеет номер 1 и принадлежит группе с номером 1. Последняя группа блоков может быть неполной. Начало каждой группы блоков имеет адрес, который может быть получен как ((номер группы - 1)* (число блоков в группе)).

Первые 1024 байт логического раздела Linux отведены на размещение загрузчика LILO (Linux Loader), и при размере блока в 1 кБ загрузчик занимает полный блок. Каждая группа блоков имеет одинаковое строение. Ее структура представлена на рис. 2.

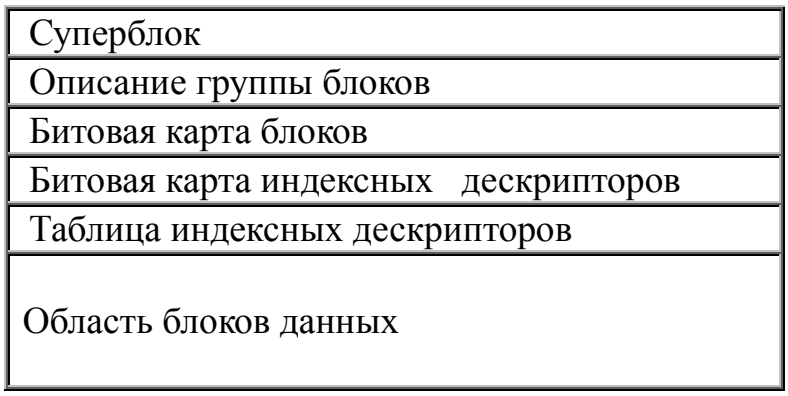

Рис. 2. Структура группы блоков

Суперблок является начальной точкой файловой системы. Он имеет размер 1024 байта, но реальной информацией заполнены только первые 80 байт – остальная часть суперблока и оставшееся пространство логического блока (при размере блока в 2 или 4 кБ) дополняются нулями. Наличие копий суперблока в каждой группе объясняется чрезвычайной важностью этого элемента файловой системы. Дубликаты суперблока используются при восстановлении файловой системы после сбоев. Тем не менее некоторые утилиты (например mount – утилита монтирования файловой системы) используют только первый экземпляр суперблока и при его повреждении 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 информация о том, где искать незанятые участки. При запуске ОС суперблок считывается в память и все изменения файловой системы вначале находят отображение в копии суперблока, находящейся в оперативной памяти, и записываются на диск только периодически. Это позволяет повысить производительность системы, так как многие пользователи и процессы постоянно обновляют файлы. При выключении системы суперблок обязательно должен быть записан на диск, что не позволяет выключать компьютер простым выключением питания. Суперблок имеет следующую структуру (табл. 2.):

Таблица 2
Структура суперблока

Размер поля в байтах	Смещение байт	Назначение
4	0	Число индексных дескрипторов в файловой системе (возможное число файлов)
4	4h	Число блоков в файловой системе
4	8h	Число блоков, зарезервированных для нужд суперпользователя
4	Ch	Число свободных блоков
4	10h	Число свободных индексных дескрипторов
4	14h	Номер первого блока, содержащего данные (0 или 1)
4	18h	Индикатор размера логического блока: 0 = 1 Кб; 1 = 2 Кб; 2 = 4 Кб.
4	1Ch	Индикатор размера фрагментов (если фрагментация блоков предусмотрена)
4	20h	Число блоков в каждой группе блоков
4	24h	Число фрагментов в каждой группе блоков
4	28h	Число индексных дескрипторов в каждой группе блоков
4	2Ch	Время последнего монтирования файловой системы (в секундах с 1 января 1970 года)
4	30h	Время последней записи в файловую систему
2	34h	Число монтирований файловой системы. Если этот счетчик достигает значения, указанного в следующем поле, файловая система при перезапуске проверяется, а счетчик обнуляется.
2	36h	Предельное число монтирований файловой системы
2	38h	"Магическое число" (0xEF53), указывающее, что файловая система принадлежит к ex2fs или ext3fs
2	3Ah	Флаги, указывающие текущее состояние файловой системы.
2	3Ch	Флаги, задающие процедуры обработки сообщений об ошибках
2	3Eh	Заполнение
4	40h	Время последней проверки файловой системы
4	44h	Максимальный период времени между проверками файловой системы
4	48h	Указание на тип операционной системы, в которой создана файловая система
4	4Ch	Версия файловой системы
944	50h	Заполнение до 1024 байт

В листинге 6 показано, как выглядит суперблок при выводе информации с помощью редактора Linux Disk Editor (синтаксис команд редактора приведен далее по тексту и в прил. 2).

```
            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..........
```

Листинг 6. Содержимое первых 80 (50h) байт суперблока

Подчеркиванием выделены слова и двойные слова, поименованные в таблице. Шестнадцатеричные числа в приводимых здесь и далее дампах памяти представлены в перевернутом формате (т.е. читаются в обратном порядке, справа налево). Так, первые четыре байта представляют двойное слово 00 0E 07 00h. Воспользовавшись любым из калькуляторов, доступных в консоли или графической оболочки (следует порекомендовать калькулятор bc), 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 расходуется для размещения копий суперблоков, описателей групп блоков, битовых карт, таблиц inode).
- Для администратора зарезервировано 1666Fh = 91759 блоков. Это около 10% всего дискового пространства.
- На диске в данном логическом разделе свободно 1347F0h = 1263600 блоков или 4935,93 Мбайт.
- Размер логического блока 1000h = 4096 байт.
- В каждой группе – 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. Резервные суперблоки в случае повреждения первого можно найти в 32768, 65534 и последующих логических блоках.

Вслед за суперблоком, в логическом блоке со следующим номером расположено описание группы блоков (Group Descriptors) размером 32 байта. Оно представляет собой массив со следующей структурой:

Таблица 3

Структура описателя группы блоков

Размер поля в байтах	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 дескрипторов.

```
           0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
0x00001000 02 00 00 00 03 00 00 00 : 04 00 00 00 09 1A 14 40  ...............@
0x00001010 02 00 00 00 00 00 00 00 : 00 00 00 00 00 00 00 00  ................
```

Листинг 7. Дамп описателя группы блоков

В описании группы блоков можно найти довольно интересную информацию. Например (см. листинг 7), битовая карта блоков располагается в блоке 2 (00 00 00 02h), а 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 дескрипторов, 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 прил. 2) можно одновременно вывести информацию о суперблоке и группах блоков в более удобной форме. Для этого предлагается использовать команду:

`debugfs -R stats device`

Где вместо **device** указывается конкретный файл блочного устройства, например /dev/hda7, который соответствует логическому разделу жесткого диска с файловой системой ext2fs

```
Filesystem volume name:   Ural
Last mounted on:          <not available>
Filesystem UUID:          e21e03f0-78d6-11d8-8d23-e14cf2ed36f3
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal filetype needs_recovery
sparse_super
Default mount options:    (none)
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              919296
Block count:              1835008
```

```
Reserved block count:     91759
Free blocks:              1263777
Free inodes:              799179
First block:              0
Block size:               4096
Fragment size:            4096
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         16416
Inode blocks per group:   513
Filesystem created:       Thu Mar 18 17:22:11 2004
Last mount time:          Wed Mar 31 10:51:19 2004
Last write time:          Wed Mar 31 10:51:19 2004
Mount count:              12
Maximum mount count:      39  `максимальное число монтирований
Last checked:             Thu Mar 18 17:22:11 2004
Check interval:           15552000 (6 months)
Next check after:         Tue Sep 14 18:22:11 2004
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               128
Journal UUID:             <none>
Journal inode:            8
Journal device:           0x0000
First orphan inode:       215730  `первый «осиротевший» дескриптор
Directories:              6542
 Group  0: block bitmap at 2, inode bitmap at 3, inode table at 4
           6907 free blocks, 16404 free inodes, 2 used directories
 Group  1: block bitmap at 32770, inode bitmap at 32771, inode table at
32772      21539 free blocks, 13804 free inodes, 87 used directories
 Group  2: block bitmap at 65536, inode bitmap at 65537, inode table at
65538      23284 free blocks, 14394 free inodes, 158 used directories
 Group  3: block bitmap at 98306, inode bitmap at 98307, inode table at
98308      23711 free blocks, 14240 free inodes, 190 used directories
 Group  4: block bitmap at 131072, inode bitmap at 131073, inode table at
131074     23287 free blocks, 14918 free inodes, 116 used directories
 Group  5: block bitmap at 163842, inode bitmap at 163843, inode table at
163844     22903 free blocks, 14532 free inodes, 84 used directories
```

Листинг 8. Фрагмент листинга суперблока и групп блоков, полученный с помощью отладчика Debugfs

Выведенная информация несомненно более наглядна. Вероятно, это единственный случай, когда отладчик файловой системы оказывается более удобным, чем редактор Lde. Из инфориации 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. 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 бло-

ков в тех случаях, когда надо выделить место под файл. Вот как выглядит фрагмент 2-го логического блока (block bitmap). Байты FF в двоичном виде – это 1111 1111 (все блоки заняты). Со смещения 24CCh начинаются свободные блоки. Наличие байтов, отличных от FF, указывает на то, что в файловой системе присутствуют свободные блоки, ранее принадлежавшие каким-то файлам.

```
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  ................
```

Листинг 9. Фрагмент битовой карты блоков

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. О том, какие команды следует использовать для вывода информации о свободных блоках, будет сказано ниже.

Битовая карта индексных дескрипторов имеет аналогичную структуру, но иную гранулярность: один бит соответствует 128-байтному фрагменту в таблице inode. Фрагмент битовой карты индексных дескрипторов приведен в листинге 10. Здесь наблюдается иной порядок, чем в битовой карте блоков: вначале идут свободные индексные дескрипторы, затем занятые.

```
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  ................
```

Листинг 10. Фрагмент битовой карты индексных дескрипторов.

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 порядко-

вым номером - индексом файла. Индексные дескрипторы файлов данной группы блоков хранятся в логических блоках, расположенных следом за битовой картой индексных дескрипторов. В одном 4-килобайтном логическом блоке помещается 32 таких записи. Таким образом, 16416 индексных дескрипторов, приходящихся на одну группу (см. листинг 8), 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, будет неверна. В журнализируемой файловой системе EXT3FS между битовой картой inode и таблицей индексных дескрипторов размещается журнал транзакций, под который обычно отводится 8192 блока. Таким образом, в первой группе блоков таблица индексных дескрипторов будет начинаться не с 4-го, а с 8195-го логического блока.

Индексный дескриптор файла имеет объем 128 байт и следующее строение:

Таблица 4

Структура индексного дескриптора файла

Размер поля в байтах	Смещение байт	**Описание**
2	0	Тип файла и права доступа к нему
2	2h	Идентификатор владельца файла UID
4	4h	Размер файла в байтах
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
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	ACL файла
4	6Ch	ACL каталога
4	70h	Адрес фрагмента
1	74h	Номер фрагмента
1	75h	Размер фрагмента
2	76h	Заполнение
4x2	78h	Зарезервировано

Таблицу индексных дескрипторов можно вывести поблочно с помощью команды **`lde -b block_numder device`**. Произвольно взятый фрагмент из двух индексных дескрипторов представлен для анализа в листинге 11. Размер каждого из inode составляет 128 байт, следовательно, интервал между ними – 80h и их начала выровнены по смещению 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  ................
```

Листинг 11. Фрагмент таблицы индексных дескрипторов

Индексный дескриптор определяет порядок доступа к конкретному файлу, поэтому, модифицируя эту запись, можно манипулировать файлом как угодно. 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. Используя вышеприведенную таблицу inode, попробуем расшифровать шестнадцатеричные записи.

Первые два байта несут очень важную информацию о типе файла и правах доступа к нему. Как уже было указано, в ОС UNIX может быть 7 типов файлов. Информация о типе файла расположена в старшем нибле (полубайте) старшего байта. Информацию о типах файлов можно свести в табл. 5.

Таблица 5
Обозначение функциональных типов файлов

Тип файла	Обозначение типа файла в листинге	Шестнадцатеричная цифра
Обычный файл	**-**	**8**
Каталог	**d**	**4**
Символическая ссылка	***l***	**C**
Сокет	**s**	**A**
Именованный канал	**f**	**1**
Файл блочного устройства	**b**	**6**
Файл символьного устройства	**c**	**2**

Права доступа к файлу отображаются 12 двоичными разрядами, состоящими из 4-ех полей по 3 разряда в каждом. Для удобства чтения оставшиеся 3 шестнадцатеричных цифры следует преобразовать в двоичную форму.

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 – бит, не позволяющий пользователю удалять из каталога файлы, ему не принадлежащие (устанавливается только для каталога в отношении пользователей, не являющихся его владельцами).

Все оставшиеся тройки интерпретируются одинаково: левый бит – право на чтение, средний – на запись, правый – на исполнение. Левая тройка – права владельца, средняя тройка – права его пользователей из его группы, правая тройка – права для остальных зарегистрированных пользователей.

Так, в первых двух байтах первого inode в листинге 11 записано шестнадцатеричное число 41 EDh. По старшей цифре 4 мы определяем, что перед нами – каталог. Оставшуюся часть числа преобразуем в двоичную форму, для наглядности отделяя 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.
- владелец файла имеет на него полные права,
- члены группы владельца имеют права чтения и исполнения,
- все остальные зарегистрированные пользователи также имеют права на чтение и исполнение (вполне естественно для каталога).

Следующее слово является идентификатором владельца файла Owner UID. На это поле выделено два байта, поэтому всего пользователей (включая псевдопользователей) может быть 65536. Только администратор системы имеет нулевой идентификатор. По числу 00 00h убеждаемся, что владельцем каталога именно он и является.

Следующее двойное слово содержит информацию о размере файла в байтах. 00 00 10 00h = 4096 байт - обычный при данном размере логического блока объем каталога.

Четыре поля по 4 байта в каждом отображают временные отметки файла:

- последнего обращения,
- создания,
- последней модификации,
- удаления.

Разработчики системы, предусмотрев в индексном дескрипторе файла время его удаления, предполагали, что удаление файла – процесс логический, после совершения которого inode какое-то время будет существовать. Как будет показано ниже, удаление файла фактически не сопровождается ни удалением inode, ни стиранием информации в блоках, выделенных файлу. Удаление файла – это лишь стирание его последнего имени («жесткой» ссылки) в каталоге.

Все поля, содержащие временные отметки, содержат число секунд, прошедших с полуночи 1 января 1970 года. Чтобы узнать это время, нужно вначале преобразовать число в десятичное, затем преобразовать секунды в дату и время. Утилиты для преобразования в обычное представление даты и времени в штат-

ной инсталляции операционной системы нет, но зато есть системная функция, которую можно вызвать из исполняемого файла или сценария, написанного на Perl.

По порядку расшифровываем следующие поля. Идентификатор группы имеет значение равное 00 00h – это группа root (администратора). На каталог имеются две жесткие ссылки (00 02h). Каталог занимает 8 секторов по 512 байт на диске, что соответствует размеру одного блока в 4096 байт. 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.

Аналогично определим параметры другого файла, приведенные в листинге 11. Первая шеснадцатеричная цифра слова 81A4h указывает на то, что это обычный файл, а двоичная комбинация **1A4h = 0001 1010 0100b = 000 110 100 100** дает всю необходимую информацию о правах доступа:

- эффективные права доступа не установлены,
- владелец файла имеет право чтения и записи,
- члены его группы и остальные пользователи имеют право на чтение,
- владельцем файла является root – администратор.

Файл имеет размер 37h = 55 байт и так далее. По информации, содержащейся в inode, уже нетрудно найти и идентифицировать файл.

В более удобной для анализа форме вывести информацию об индексном дескрипторе можно, воспользовавшись для этого командой lde. Ниже приведен пример вывода 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 05:00:00 1970
DIRECT BLOCKS:          0x00040203

INDIRECT BLOCK:
DOUBLE INDIRECT BLOCK:
TRIPLE INDIRECT BLOCK:
```

Листинг 12. 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 в листинге 12. Время удаления файла, датируемое 1970 годом, – не временной парадокс, а единая точка отсчета всех 4 временных отметок. Если она не указывает реальное время, значит факт удаления файла еще не состоялся.

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 4x12 = 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*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.

В листинге 13 приведены inode и фрагмент блока данных файла – сценария. 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
```

Однако в распечатке inode дополнительные атрибуты не отображаются. Их можно увидеть лишь в соответствующих полях в таблице индексных дескрипторов. В этом нет ничего странного: операционные системы непрерывно

развиваются и дополнительные атрибуты файлов появились несколько позже утилит, отображающих информацию о файлах.

```
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:
```

Листинг 13. Информация о файле-сценарии

```
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  ................
```

Листинг 14. Блок данных файла-сценария

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. В листинге 15 представлен фрагмент распечатки расширенных сведений о файлах, находящихся в каталоге /bin.

```
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 dnsdomainname ->
```

Листинг 15. Фрагмент каталога /bin, отображенный командой ls –ali /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. Каталог по сути представляет собой таблицу, каждая строка 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 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.

```
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..........
```

Листинг 16. Дамп блока данных, содержащего каталог /bin

В листинге 16 представлен фрагмент блока данных каталога /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 2E. 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 (11 строка сверху в листинге 15). 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 соответствуют символам имени файла - sh. Приведенных примеров вполне достаточно, чтобы произвести разбор любой записи – хоть с ее начала, хоть с конца.

Как уже указывалось, большинство каталогов помещаются 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.

3.1. Восстановление удаленных файлов

Многие исследователи и опытные пользователи операционных систем клона UNIX утверждают, что они 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.

В операционных системах Linux обычно не предусматривается «корзины» для удаленных файлов, и для обычного пользователя, не имеющего доступа к элементам файловой системы, восстановление уничтоженной информации будет недоступно. 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».

Администратору системы можно порекомендовать вообще закрыть доступ к утилите **rm** для всех пользователей и оставить ее только для собственного использования. Взамен рекомендуется написать сценарий с таким же именем, который помещает удаляемые файлы в специально созданный каталог – он и будет прообразом «корзины». Приводить пример такого сценария нет смысла – это одно из заданий для обучаемых. Однако, если о существовании такого каталога ненадолго забыть, он превратится в своеобразную «Авгиеву конюшню» для операционной системы.

В правах доступа на файл право на его удаление 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 UNIX 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.

Универсальные операционные системы семейств UNIX и Windows* по умолчанию не предусматривают физического удаления файлов. Для гарантированного стирания файлов пользователям рекомендуется использовать специальные утилиты, которые именуются шредерами (от англ. shred – кромсать, резать). Утилита с таким названием имеется в штатной инсталляции современных опе-

рационных 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.

Восстановление удаленных файлов в файловых системах Linux затрудняется рядом обстоятельств. Файл, как известно, состоит из трех частей: совокупности имен – «жестких» ссылок на индексный дескриптор одного индексного дескриптора, и определенного количества блоков. При создании файла системной функции **creat** передается символьное имя файла в файловой системе и устанавливаемые права доступа к нему. Система в соответствии с заложенным алгоритмом выделяет для этого имени свободный inode и 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 той же группе каталогов, где находился удаленный файл, система неминуемо использует первый по порядку свободный inode. Таким образом вслед за последним именем файла исчезнет и его индексный дескриптор. Поэтому, 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.

Проверим, как очищается блок перед его заполнением новыми данными. Для этого воспользуемся специально написанной М.Э.Пономаревым утилитой **fillfile**, которая создает файл нужного размера, заполняет его выбранными символами и обозначает имя файла таким же символом. Пример: команда **`fillfile 600 b`** создает в текущем каталоге файл с именем **b** и объемом 600 байт, который содержит одни символы **b**. С помощью таких файлов с определенным наполнением весьма удобно наблюдать за выделением дискового пространства.

1. Перейдем в каталог **/home** и командой **`fillfile 3000 a`** создадим в нем файл размером 3000 байт (около ¾ размера логического блока) с именем «a», заполненный этими же буквами. Далее с помощью команды ls –li выведем список файлов текущего каталога, найдем в списке файл «a», удостоверимся в том, что его размер составляет 3000 байт, и прочитаем его inode. Допустим, его inode = 234567. С помощью редактора файловой системы **`lde -i 234567 /dev/hdc3`** (в данном случае /dev/hdc3 – логический раздел Linux) выведем информацию, содержащуюся в этом индексном дескрипторе.
2. Найдем в inode = 234567 прямую ссылку на единственный логический блок, отведенный файлу. Допустим, номер этого блока 0x000167A8. Заглянем с помощью команды **`lde -b 0x167A8 /dev/hdc3`** в этот блок и обнаружим, что он от начала до смещения 3000 байт (BB8h) заполнен символами «a».
3. Удалим с помощью команды **`rm -f a`** созданный файл. Просматривая его inode и единственный адресуемый блок, убеждаемся, что никакого удаления не произошло. Два изменения, произошедшие в записях inode, указывают на неиспользуемый файл (NOT USERS в заголовке дескриптора и реальное время удаления файла).
4. С помощью утилиты fillfile создадим файл “b” размером 300 байт и заполненный символами “b”. Просматривая каталог /home с помощью команды ls –ali /home, мы можем увидеть этот файл и прочитать его индексный дескриптор. Не вызывает особого удивления, что система выделила новому файлу номер, ранее закрепленный за старым файлом. Как видно, inode после удаления файла сохраняют прежнюю информацию недолго.
5. Выведя для осмотра inode файла “b”, можно убедиться также в том, что ему отведен для размещения данных блок только что удаленного файла “a”. Просмотр содержимого блока данных позволил установить различные результаты этой записи в различных версиях одной и той же операционной системы. Так, для систем с ядром Linux не выше 2.3 блок при перезаписи предварительно не очищается. 300 байт от начала блока занимают символы ‘b”, затем следуют 212 нулевых байт (дополнение до длины сектора), а оставшаяся часть блока до размера 3000 байт заполнена символами “a”. Как видно, системы старых версий не перезаписывают полностью логический блок, благодаря чему можно обнаружить «технологический мусор» от прежних файлов. Но в системах с ядром 2.4 и выше подобного «мусора» уже не наблюдается: системы очищает все логические блоки, выделяемые файлам.

Механизм журнализации, реализованный в файловых системах Linux, чрезвычайно затрудняет 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. Подобные наблюдения можно производить, монтируя логические разделы EXT3FS как EXT2FS (т.е. без журнализации).

Теперь рассмотрим, что происходит с данными каталога при удалении файла. На листинге 17 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
```

Листинг 17. Фрагмент дампа логического блока, содержащего каталог /home (до удаления файловой записи Pr_Linux.doc)

В листинге 18 приведен тот же фрагмент логического блока каталога после удаления файловой записи. Необходимо отметить, что в журнализируемой файловой системе EXT3FS для обновления информации об удалении файла 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
```

Листинг 18. Фрагмент дампа логического блока, содержащего каталог /home (после удаления файловой записи Pr_Linux.doc)

Рассмотрим содержание файловой записи до ее удаления (листинг 17). 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 байт (в этом нетрудно убедиться). Два байта, стоящие левее (00 14h), также безошибочно определяют длину записи – 20 байт. 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 (листинг 18). Запись полностью не исчезла, от нее осталось окончание «ux.doc». 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 его можно попытаться восстановить. Однако сказать, какому индексному дескриптору это имя соответствовало и, тем более, где располагаются блоки данных удаленного файла, невозможно.

Авторы одной из методик [11,12] 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 состояние, когда их оканчательно нельзя будет восстановить.

Для восстановления данных нужно найти их на устройстве и сделать так, чтобы они снова стали восприниматься операционной системе. Для этого существует два способа. Первый - изменить существующую файловую систему так, чтобы в удаленных 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.

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 исследуемом логическом разделе Linux 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 – 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 следует модифицировать. Модификацию можно провести с помощью отладчика debugfs. Для этого следует последовательно выполнить следующие команды (более подробное описание этих команд содержится в приложении):

```
debugfs
```

Входим в командную среду отладчика.

```
open -w /dev/hdc3
```

Открываем исследуемый раздел в режиме чтения/записи.

lsdel

Выводим на экран список удаленных inode.
Далее выбираем один из найденных индексных дескрипторов и осуществляем дальнейшую работу с ним.

stat <inode>

Выводим таблицу inode, которая позволяет убедиться в том, что в ней сохранились адресуемые блоки данных.

dump <inode> out_file

Сохраняем содержимое индексного дескриптора в файл.

ncheck inode_number

Пытаемся найти имя файла по его индексному дескриптору.

seti <inode>

Бит, соответствующий данному inode в битовой карте индексных дескрипторов, устанавливаем в «1». Тем самым указываем системе, что индексный дескриптор вновь занят.

mi <inode>

В выводимой информации модифицируем две строки: устанавливаем в единицу число ссылок на inode и обнуляем время удаления файла.
Затем процедура повторяется для каждого найденного inode,

close

Закрываем логический раздел.

quit

Выходим из отладчика.

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 затираются новыми данными 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.

4. РАБОТА С ОБЪЕКТАМИ ФАЙЛОВОЙ СИСТЕМЫ

4.1. Работа системы с аппаратными устройствами

Многозадачные операционные системы не позволяют программным приложениям напрямую работать с аппаратными компонентами компьютерной системы. 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 инструкции, позволяющие непосредственно обращаться к портам ввода/вывода в защищенном режиме центрального процессора, 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 или записывать данные на устройства, как в обычные файлы, не обращая внимание на конкретную аппаратную реализацию устройств. Пользователь, имеющий необходимые права, а также созданные им процессы, имеют неограниченные возможности по работе с компьютерной аппаратурой.

Файлы специальных устройств внешне напоминают обычные файлы. Их можно перемещать из каталога в каталог с помощью команды mv (move) и удалять командой rm (remove). Но с этими файлами не связаны блоки данных на машинном носителе. Данные, помещаемые в такой файл, передаются драйверу устройства, а читаемые из файла – запрашиваются у драйвера. Копирование специального файла приведет к чтению из соответствующего устройства.

Различают символьные и блочные устройства, так же называются и соответствующие им специальные файлы. При выводе информации о файлах с помощью команды ls –la файлы символьных устройств можно распознать по первой букве «c», а блочные – по символу “b”. К символьным относятся устройства, читающие и записывающие данные в виде потока байтов (ленточные накопители, клавиатура, звуковые адаптеры, устройства, подключенные к последовательным или параллельным портам). Примером блочного устройства является магнитный или оптический диск – информация записывается на них и соответственно считывается блоками фиксированного размера (секторами).

Зная имя устройства, можно создать для него специальный файл. Это производится с помощью команды **mknod**, исполнение которой разрешается только суперпользователю. Обычно создавать файлы специальных устройств не требуется, поскольку они присутствуют в стандартной инсталляции операционной системы. Располагаются файлы устройств в каталоге /dev (device – устройство). Узнать, какому устройству соответствует специальный файл, можно по характерным именам:

- **fd0, fd1** (fd – floppy disk) – соответственно первый и второй дисководы ГМД,
- **hdXY** (hd – hard disk) – логический раздел жесткого диска (магнитного или оптического) с IDE-контроллером. X – символы a,b,c,d, обозначающие по-

рядковый номер шлейфа (a – первый (master) накопитель основного шлейфа, b – второй (slave) накопитель основного шлейфа и т.д.),

- **sdXY** (sd – SCSI disk) – логический раздел жесткого диска со SCSI-интерфейсом,
- **lp0, lp1** (lp – line port) – параллельные порты,
- **ttyS0, ttyS1**(tty – teletype) – последовательные порты,
- **tty** – терминал,
- **audio** – звуковой адаптер,
- **ht0, st0** – IDE и SCSI накопители на магнитной ленте.

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.

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. Например:

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

Например, при подключении исследуемого IDE-диска в качестве “master”-устройства к “secondary” интерфейсному каналу, диск будет представлен как файл /dev/hdc.

Для того чтобы выяснить трехмерную логическую геометрию диска CHS, количество и типы находящихся на нём разделов, необходимо, работая в режиме суперпользователя, выполнить команду **fdisk –l device**, где device – файл конкретного устройства (исследуемого диска):

```
Disk /dev/hdc: 10.2 GB, 10248118272 bytes
16 heads, 63 sectors/track, 19857 cylinders
Units = cylinders of 1008 * 512 = 516096 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hdc1   *           1        4161     2097112+   c  W95 FAT32
(LBA)
/dev/hdc2            4162        4364      102312   83  Linux
/dev/hdc3            4365       19467     7611912   83  Linux
/dev/hdc4           19468       19857      196560    f  W95 Ext'd
(LBA)
/dev/hdc5           19468       19857      196528+  82  Linux swap
```

Листинг 19. Информация, выводимая командой **`fdisk –l /dev/hdc`**

Информация, приведенная в листинге 19, выведенная командой **`fdisk –l /dev/hdc`**, говорит о следующем:

- Исследуемый диск имеет общую ёмкость 10,2 Гбайт (10248118272 байта);

- Логическая геометрия диска: 63 сектора/16 головок/19857 цилиндров. Трехмерная логическая адресация диска CHS (cylinders, heads, sectors) не соответствует реальному физическому размещению данных на магнитных дисках. Действительно, можно ли представить, что внутри герметичного блока жесткого диска размещены 16 магнитных головок или 8 дисков? Чаще всего в корпусе гермоблока имеется всего два жестких диска и, соответственно, 4 магнитных головки. А количество цилиндров в соответственное число раз больше (19857*4=79428). Количество секторов тоже указано неверно; в различных пространственных зонах диска оно в зависимости от длины окружности дорожек находится в диапазоне от нескольких десятков до сотен секторов. Настоящая физическая геометрия диска известна только его контроллеру. Однако человек, не имея доступа внутрь гермоблока, видит дисковое пространство глазами операционной системы.
- Единица отображения размещения разделов – цилиндр; размер цилиндра равен 516096 байт (63 *16 * 512 – число секторов * число головок * размер сектора). Поскольку контроллер жесткого диска скрывает истинную геометрию дискового пространства, данная информация не имеет большого значения;
- **/dev/hdc1** – активный первичный раздел № 1 с типом системы “W95 FAT32 (LBA)” (Id раздела равен “c” в шестнадцатеричном виде), размещается с 1-го по 4161-й цилиндр и имеет размер 2097112 блоков по 1024 байта;
- **/dev/hdc2** – первичный раздел № 2 с типом системы “Linux” (Id раздела равен “83” в шестнадцатеричном виде), размещается с 4162 по 4364 цилиндр и имеет размер 102312 блоков по 1024 байта;
- **/dev/hdc3** – первичный раздел № 3 с типом системы “Linux” (Id раздела равен “83” в шестнадцатеричном виде), размещается с 4365 по 19467 цилиндр и имеет размер 7611912 блоков по 1024 байта;
- **/dev/hdc4** – расширенный раздел № 4 с типом системы “W95 Ext'd (LBA)” (Id раздела равен “f” в шестнадцатеричном виде), размещается с 19468 по 19857 цилиндр и имеет размер 196560 блоков по 1024 байта;
- **/dev/hdc5** –логический раздел № 5 с типом системы “Linux swap” (Id раздела равен “82” в шестнадцатеричном виде), размещается с 19468 по 19857 цилиндр и имеет размер 196528 блоков по 1024 байта.

Плюсы после количества блоков в разделах указывают на то, что раздел не кратен целому числу цилиндров.

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.

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 обозна-

чения первичных разделов, а остальные 11 номеров – для обозначения логических. Например:

- **/dev/sda2** – обозначает второй первичный раздел диска;
- **/dev/sda6** – обозначает второй логический раздел диска.

Для решения некоторых практических задач система содержит несколько виртуальных устройств, которые не нуждаются в аппаратных компонентах:

- **/dev/null** – своеобразная «черная дыра», удаляющая любой поток данных. В этот файл можно только записывать. Если перенаправить вывод в null, данные не будут отображаться на экране,
- **/dev/zero** – «рог изобилия», файл, из которого можно бесконечно читать поток из одних нулей,
- **/dev/random** – устройство, генерирующее поток случайных чисел при активности пользовательского ввода (движение мыши или нажатие нескольких клавиш на клавиатуре вызывает какое-то число случайных байт – от нескольких до нескольких сотен). Это виртуальное устройство может быть использовано и 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.

4.2. Монтирование файловых систем

Точкой монтирования обязательно должен быть каталог. При монтировании файловой системы его содержимое добавляется к существующему дереву каталогов в виде дополнительной ветви, «растущей» из точки монтирования. Это несколько напоминает «мичуринскую прививку».

Если в исходной файловой системе из точки монтирования выходили другие подкаталоги и файлы, то после монтирования они станут невидимы – их «закроют» ветви примонтированной файловой системы. Старые данные не уничтожаются – они вновь станут видимы после размонтирования. Поэтому точка монтирования должна представлять собой пустой каталог.

Монтирование – это привилегированная операция, разрешаемая только суперпользователю. Монтирование файловых систем на этапе загрузки производится с помощью небольшого текстового файла /etc/fstab. Файл /etc/fstab, со-

держимое которого приведено в листинге 20, представляет собой таблицу из шести столбцов.

dev/hda1	**/**	**ext2**	**defaults**	**1**	**1**
dev/hda2	**/mnt/ntfs5**	**ntfs**	**defaults**	**0**	**0**
dev/hda3	**/mnt/fat32**	**vfat**	**defaults**	**0**	**0**
dev/hda4	**swap**	**swap**	**defaults**	**0**	**0**
devpts	**/dev/prs**	**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**

Листинг 20. Содержимое файла /etc/fstab

1. В первом столбце указывается дисковое (блочное) устройство, точнее логический раздел диска, на котором содержится файловая система, подлежащая монтированию. Типовые названия этих устройств перечислялись в параграфе 4.1. Исключением являются два псевдоустройства: файловая система /proc, рассмотренная в параграфе 2.5, и псевдотерминал devpts.
2. Второй столбец таблицы указывает точку (каталог) монтирования. Каталог к моменту монтирования уже должен существовать. Имена точкам монтирования даются таким образом, чтобы они ассоциировались с конкретными устройствами (например, /mnt/floppy или /mnt/cdrom). Это в какой-то степени заменяет пользователю привычное обращение к логическим дискам.
3. Третий столбец указывает на тип файловой системы. Система Linux позволяет работать с многими файловыми системами, в том числе msdos, vfat (FAT-32), ntfs, ufs, iso9660, proc и т.д. Следует обратить внимание на то, что файл подкачки в системах Linux размещается в отдельном логическом разделе жесткого магнитного диска.
4. В четвертом столбце таблицы указываются параметры монтирования. Эти параметры в таблице fstab и в командной строке инструкции mount указываются различным способом.

- **ro** – (read only) – файловая система монтируется «только для чтения» (обычно это указывается для привода чтения оптических компакт-дисков),
- **rw** – файловая система монтируется для чтения и записи,
- **async/sync** – обмен данными между оперативной памятью и данной файловой системой производится асинхронно/синхронно,
- **exec/noexec** – разрешить или запретить запуск исполняемых файлов, расположенных в данной файловой системе. Таким образом, например, можно запретить запуск потенциально опасных программ с дискеты или оптического диска,
- **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,

- **nouser/user(s)** – запретить или разрешить пользователям монтировать данную файловую систему. Параметр user указывает, что монтировать файловую систему в данную точку монтирования может любой пользователь. Если требуется указать конкретного пользователя с данными правами, это записывается так: user=ivanov. 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, async. Если в четвертом столбце одновременно указаны параметры defaults и user(s), то значения параметров по умолчанию изменяются на noexec, nosuid и nodev.

5. Пятый столбец таблицы может содержать 0 или 1. Единица предписывает производить резервное копирование данной файловой системы, а 0 – не производить.
6. Шестой столбец используется утилитой проверки файловых систем fsck (file system check). Если указан «0», то файловая система не проверяется (штатная утилита fsck может корректно проверить только файловые системы ext2fs, ext2fs), цифры «1» или «2» указывают очередность проверки.

Файловую систему можно смонтировать и «вручную» с помощью команды mount. Это довольно сложная команда с разнообразным синтаксисом. В наиболее простом варианте данная команда выглядит следующим образом:

`mount -t type_fs device dir,`

например**: `mount -t msdos /dev//hda5 /mnt/floppy`**

Прежде чем подключать (монтировать) большую 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 **e2fsck**. Это справедливо для файловой системы **EXT2FS**, в случае же наличия на исследуемом разделе файловой системы **EXT3FS** ошибка фиксироваться не будет, так как журналируемая файловая система по умолчанию воспринимается исправной.

В случае повреждения первого и последующих суперблоков файловой системы запуск программы **e2fsck** обязателен.

Порядок запуска утилиты при проверке:

`e2fsck -f /dev/hdc3`

или

```
e2fsck -fy /dev/hdc3
```

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**

Если файловая система занята (открыты ее каталоги, запущены исполняемые файлы, открыты обычные файлы и др.), ее демонтировать нельзя – система сообщит об ошибке. Нельзя извлекать дискету из дисковода и оптический диск из лотка до размонтирования.

4.3. Копирование файлов

Копирование – это создание еще одного экземпляра файла. Можно скопировать файл в тот же каталог, где расположен оригинал, но изменить имя копии. Можно скопировать файл с прежним именем в другой каталог. Наконец, можно скомбинировать эти два варианта. Для копирования файла надо иметь следующие права:

- право на чтение файла-оригинала,
- право на чтение и исполнение каталога, в котором исходный файл хранится,
- право на запись и исполнение в каталоге, куда предстоит записать копию.

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 GUID. 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, /dev и в принципе могут обзавестись своими копиями различных утилит и специальных файлов.

Сами по себе копии утилит опасности не вызывают, ведь пользователь сможет запустить их только со своими правами. Но дело в том, что некоторые защитные механизмы встроены в сами утилиты. Многие из них могут выполнять потенциально опасные действия в зависимости от используемых опций. Так, например, утилита **ping** может быть запущена с целью проверки существования сетевого узла от имени любого пользователя. Но отправка по определенному сетевому адресу большого числа пакетов ICMP (Echo_Request), следую-

щих с максимальной скоростью, это уже разновидность атаки на отказ в обслуживании, и запуск программы ping с соответствующими опциями – это прерогатива администратора. Программа, запущенная с подобными опасными опциями (или вызываемый ею библиотечный код) должна сама проверить UID и, если он не равен нулю, информировать пользователя о невозможности выполнения команды (либо просто ее игнорировать).

Модифицируя программный код, злоумышленник может создать экземпляры утилит, которые способны выполнять некоторые опасные функции независимо от прав запустившего их лица (в том случае, если проверка допустимости исполнения не возложена на ядро операционной системы). Так появляются «инструменты администратора» – Root Kit, позволяющие осуществлять взлом системы с приобретением недозволенных прав.

Вторая опасность использования копий часто используемых утилит заключается в следующем. Пользователь-злоумышленник создает в своем каталоге вредоносную программу, которой он присваивает такое же имя, как утилите. Вредоносной может быть и модифицированная копия утилиты. Для выполнения вредоносного действия необходимо, чтобы эта программа была запущена от имени администратора. Атака может выглядеть следующим образом. Пользователь-злоумышленник звонит администратору и обращается к нему с жалобой на то, что система почему-то не выполняет одну из команд, например команду вывода списка файлов ls. Администратор входит в каталог пользователя и запускает оттуда указанную команду. Оказывается, она действительно не запускается – ведь запущенная на исполнение вредоносная программа выполняет какие-то иные действия, не связанные с выводом списков файлов, после чего уничтожает или переименовывает собственный файл. Озадаченный администратор делает еще одну попытку – и тут нужная команда благополучно запускается (теперь, при отсутствии фальшивой утилиты запускается уже настоящая). Администратору, у которого вечно не хватает времени, проще списать все на очередной сбой в работе сложной операционной системы.

Обычно программы запускаются в командной строке с использованием короткого имени файла. Задача найти этот файл возлагается на командный интерпретатор, который использует для поиска свою переменную окружения PATH. Если в этой переменной на первом месте будет значиться текущий каталог, а только потом – каталоги /bin и /sbin, несомненно оболочка первой найдет и запустит фальшивую утилиту, а поиском других файлов с таким же именем утруждать себя не станет. С учетом этого неплохой привычкой для администратора может стать использование абсолютных имен файлов.

Поэтому выявление копий системных утилит в рабочих каталогах пользователей является одной из функций администратора безопасности, а их обнаружение должно стать причиной разбирательства, в ходе которого обнаруженная копия должна быть исследована на наличие вредоносных функций.

Оказание помощи пользователям входит в задачу администратора. Но в таких ситуациях администратор должен вначале ввести команду su с именем пользователя, ведь реальные затруднения могут быть вызваны и различием в

правах доступа. После того как администратор окажется на месте и с правами пользователя, он сможет проверить состоятельность его жалоб, не рискуя навредить системе.

4.4. Использование «жестких» и символических ссылок

Традиционное определение файла: «Файл – это именованная область памяти» для ОС UNIX не совсем верно. Как уже было рассмотрено ранее, имен у одного и того же файла может быть довольно много – 65536. А вот номер – inode, или индексный дескриптор, у файла один. Символьные имена являются просто указателями на этот уникальный номер, позволяющими пользователю или его процессу найти файл с известным именем в каталоге и открыть его для чтения, записи или исполнения. Множество символьных указателей на один и тот же индексный дескриптор называются «жесткими» ссылками. Жесткие ссылки следует отличать от копий файлов. При создании копии файла создается другой файл, занимающий иные блоки на дисковом пространстве, и имеющий свой уникальный индексный дескриптор. Файл-копия имеет свои временные отметки, в частности, время создания. Одинаковым для файла-оригинала и его копии является только содержимое блоков данных.

Символические ссылки – это не вполне полноценные файлы. Они являются указателями не на индексный дескриптор, а на одно из имен настоящего файла. Но действие с символической ссылкой трансформируется в аналогичное действие по отношению к адресуемому файлу. На символическую ссылку нельзя задать права доступа: каждому зарегистрированному пользователю можно по отношению к символической ссылке выполнять операции чтения, записи и исполнения. Но права доступа проверяются при обращении к адресуемому файлу. Ссылку на недоступный файл создать легко, но обратиться к нему все равно будет невозможно.

У большинства символических ссылок нет блоков данных. Если адресуемое имя меньше 48 символов, оно записывается в номера непосредственно адресуемых блоков: 12 блоков х 4 байта. При исследовании inode символической ссылки следует обратить внимание на то, что номера адресуемых блоков данных выглядят неестественно и явно выходят из допустимых диапазонов номеров. Но если имя файла длиннее 48 символов, для него приходится выделять один блок данных.

Символические ссылки могут использоваться в качестве инструмента для файловой атаки. Объект атаки – программа, которая в процессе работы создает временные файлы и помещает их в каталог /tmp. Например, такой программой может быть текстовый редактор, который создает резервные копии редактируемого документа на случай внезапного сбоя в работе. Примечательность этого каталога в том, что любой зарегистрированный пользователь имеет права на чтение, запись и поиск (исполнение) в этом каталоге.

Вторая особенность связана с механизмом создания новых файлов. Если система создает в каком-либо месте новый файл, но файл с таким именем уже

существует, то его блоки данных освобождаются, а индексный дескриптор с изменением временных меток передается новому файлу. Еще раз следует напомнить, что символическая ссылка, указывающая на полное имя файла длиной до 48 символов, в блоках данных не нуждается, и содержит ссылку на файл в самом inode. Теперь 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 тот 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 перезаписана.

Для противодействия подобным атакам временные файлы в обоих семействах универсальных операционных систем Windows и UNIX 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. Угадать имя файла весьма трудно, но появляется иная проблема. Временные файлы автоматически не удаляются и, спустя некоторое время уже практически невозможно разобраться, какой программой они были созданы, содержат ли конфиденциальную информацию и подлежат ли удалению.

4.5. Файлы аудита

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 обычно имеют расширение .log (для системы расширения имен не требуется, оно предназначается для пользователя или администратора) и располагаются в каталогах /etc/init.d/, /etc/rc*, /etc/rc.d/, var/log/.

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 представляют собой двоичные файлы определённой структуры, содер-

жимое которых необходимо просматривать системными программами, интерпретирующими содержимое этих файлов в удобный для просмотра вид.

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

- "***cron***" - текстовый файл, содержащий информацию о фактах выполнения периодических заданий;
- "***debug***" - текстовый файл, содержащий отладочную информацию ядра системы и каких-либо системных или прикладных программ; в этом файле может находиться информация о подключении к сервису ***"pop3***" пользователей для получения своей почты;
- "***faillog***" - двоичный файл, содержащий информацию о неудачных попытках входа в систему, он также служит для установки максимального числа неудачных попыток входа в систему до блокирования учётной записи пользователя (утилита для работы с этим файлом - ***"faillog");***
- "***lastlog***" - двоичный файл, содержащий информацию о последних входах в систему (утилита для работы с этим файлом - ***"lastlog");***
- "***maillog***" - текстовый файл, содержащий информацию о работе почтовой системы ***"sendmail+procmail***" (файл сложен для визуального анализа, можно предложить свободно распространяемую программу ***"mreport");***
- "***messages***" - текстовый основной файл, содержащий протоколируемую информацию о системе и процессах категории выше ***"info"*** и ниже ***"warn"***;
- "***secure***" - текстовый файл, содержащий информацию о попытках получения пользователями полномочий root;
- "***wtmp***" - двоичный файл, содержащий информацию о всех регистрациях пользователей в системе (утилита для работы с этим файлом - ***"last"***).

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. Лабораторный практикум рассчитан на 20 часов лабораторных занятий.
2. Перед проведением лабораторных занятий студенты должны прослушать теоретический курс в объеме настоящего пособия, либо освоить его самостоятельно. Непосредственно перед проведением каждой работы преподаватель организует в устной или письменной форме допуск студентов к выполнению работ.
3. Для проведения занятий необходим компьютерный класс из расчета один персональный компьютер на одного студента.

На каждом компьютере должна быть инсталлирована одна из распространенных версий операционных систем GNU Linux. Для проведения лабораторных работ рекомендуется использование следующих дистрибутивов ОС Linux:

- RedHat;
- Mandrake;
- ASPLinux;
- Slackware.

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. Можно порекомендовать так называемый Live-CD (диск № 2) из состава дистрибутива Slackware Linux 9.1, который содержит полноценную консольную версию (кроме графической подсистемы и графических приложений).

Перед выполнением работ на каждом рабочем месте должны быть созданы учетные записи пользователей, предусмотренные заданием на лабораторную работу. При выполнении лабораторной работы 4 обучаемые создают и удаляют учетные записи самостоятельно.

В ходе занятий используются встроенные команды интерпретатора Bash (Bourne-Again shell) и штатные утилиты операционной системы. Для внутреннего исследования файловой системы применяется свободно распространяемая утилита Lde (Linux Disk Editor). 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 обратитесь к преподавателю.
- Все операции с файлами и каталогами производите только в режиме текстовой консоли или эмуляции текстового терминала. Использование оболочки **Midnight Commander** допускается лишь эпизодично, если пользователь не может решить задачу обычным способом. **Помните!** *В режиме эмуляции нескольких терминалов на одном компьютере использование* **Midnight Commander** *приводит к некорректному приобретению прав других пользователей.*
- При выполнении ряда заданий вам придется работать с полномочиями суперпользователя. Будьте внимательны при вводе команд. Помните, что операционная система большей частью не контролирует действий администратора, и ваше ошибочное действие может привести к краху системы. Ориентируйтесь на характерную форму приглашения к вводу команд (**$** - пользователь, **#** - администратор).
- Обучаемым при проведении лабораторных работ рекомендуется поочередно работать в нескольких текстовых консолях. Переключая консоль, вы можете поочередно работать с объектами операционной системы от имени нескольких пользователей и администратора системы.
- Некоторые задания преподавателем намеренно усложнены. Например, обучаемому 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 свои права на доступ.
- Руководствуйтесь кратким справочником по командам **Linux** (прил. 1). При необходимости обращайтесь к электронному справочнику **man.**
- Ответы на поставленные вопросы (с указанием пунктов задания) заносите в отчет, который можно вести в произвольной форме в рукописном или электронном виде.

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

Выполнение работы

1. Зарегистрируйтесь в первой консоли как **user1** с паролем, который Вам будет сообщен преподавателем.
2. С помощью **Ctrl+Alt+F2 (Alt+F2)** откройте второй текстовый терминал и зарегистрируйтесь как **user2** с тем же паролем.
3. Аналогично откройте третий текстовый терминал и попросите преподавателя зарегистрироваться в нем с правами суперпользователя.
4. Нажатием **Ctrl+Alt-F1 (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 обычного пользователя. Переходите в третью 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 (**$** -пользователь, **#** - администратор).
5. С правами **user1** попробуйте войти в каталог **/root**. Объясните результат. С помощью команды **ls -la /** просмотрите список основных каталогов и укажите, каких прав доступа вам недостает для входа в каждый из каталогов.

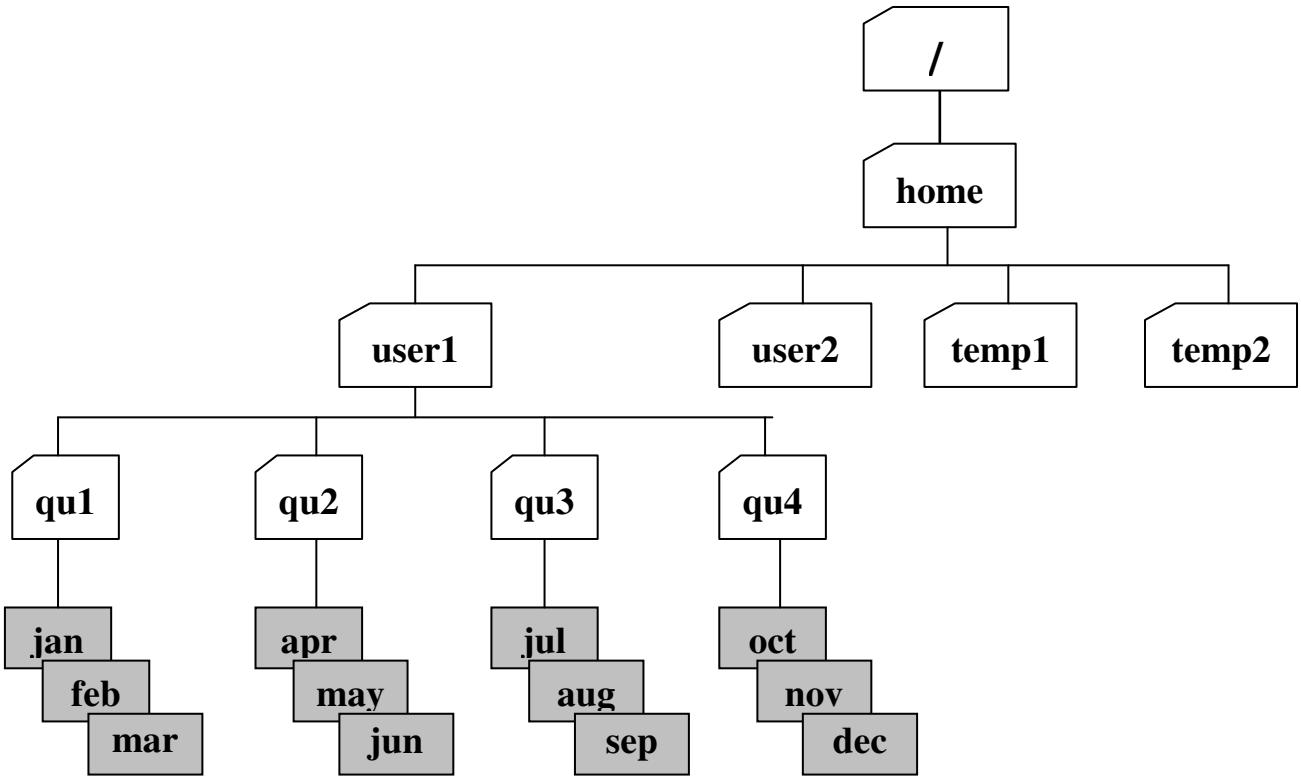

Рис. 1. Пояснение к п. 6-7 задания

6. Переключитесь в консоль администратора и создайте два новых временных каталога **`mkdir -m 777 /home/temp1`** и **`mkdir -m 1777 /home/temp2`**. Проверьте права доступа к каталогам **/home/user1** и **/home/user2**: они должны быть установлены в 755. Вернитесь в консоль **user1**.
7. 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?
8. Задайте права доступа к файлам "по умолчанию". Для этого установите **`umask 022.`** Поясните, какие права к вновь создаваемым файлам и каталогам будут предоставляться пользователю, членам его группы и остальным.
9. 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 2004 >jan`** создает в текущем каталоге файл **jan** и записывает в него календарь на январь 2004 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?
10. С помощью команды **chmod** измените нужные права доступа в "недоступные" каталоги **qu2, qu4** и создайте там указанные файлы. После этого верните каталогам прежние права доступа.
11. С помощью команд **cd** и **ls** войдите в каждый из созданных каталогов и просмотрите список созданных файлов. Для просмотра каталога необходимо последовательно ввести две команды: **cd** и **ls**. При просмотре 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?
12. Прочитайте содержимое одного из файлов в "темном" каталоге (например, **`cd /home/user1/qu3; cat aug`**). Сделайте выводы.
13. 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?
14. 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?

15. 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?
16. 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 отнесены к одной группе **users**).
17. 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** (пример: **`ln /home/user1/qu4/dec /home/temp1/dec_h1`**). Чем созданные ссылки отличаются от исходного файла? На сколько байт уменьшилось дисковое пространство после создания этих ссылок?
18. 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?
19. 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.**
20. 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)?
21. 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.
22. Попробуйте удалить любой из каталогов **qu1, qu2, qu3, qu4** с помощью команды **rmdir** (не удаляя предварительно из них файлов). Объясните результат.
23. Войдите в консоль администратора и с правами **root**, пользуясь командой **chattr,** заблокируйте файл **feb** от любых изменений. 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.
24. 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.

25. Правами пользователя **user1** с помощью команды **rm -rf** последовательно удалите ранее созданные каталоги **qu2, qu3, qu4** вместе с файлами. Объясните результат.
26. С помощью команды **md5sum** вычислите и запишите контрольную сумму для одного из файлов в каталоге **/home/user1/qu1**. Добавьте один символ в этот файл с помощью команды **echo** (например, **`echo a >>/home/user1/qu1/jan`**). Вновь вычислите контрольную сумму файла и сравните два результата.
27. С помощью команды **cat /dev/fd0** попробуйте прочесть специальный файл устройства. Объясните результат? Для чего служат специальные файлы?
28. Воспользовавшись правами суперпользователя, удалите все созданные Вами файлы. *Соблюдайте осторожность: с правами **root** и с помощью утилиты **rm** Вы можете вызвать крах системы!*
29. После выполнения работы штатными командами закройте 1-ю и 2-ю консоли, а из третьей консоли выполните команду останова системы **halt**.

Контрольные вопросы:

- Какие дополнительные атрибуты можно присвоить файлу в файловой системе **EXT2FS**? Как эти атрибуты влияют на обеспечение конфиденциальности, целостности и доступности информации?
- Как создать файл без помощи текстового редактора?
- В чем различие между копиями файла, его «жесткими» и символическими ссылками? Для чего используются символические ссылки?
- Для каких целей могут использоваться «темные» каталоги?
- Какие права по отношению к файлам и каталогам вам необходимо иметь для копирования файла? Как изменяются при этом атрибуты копии?
- Администратор по невнимательности ввел следующую команду:

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

ЛАБОРАТОРНАЯ РАБОТА № 2
«Детальное исследование файловой системы EXT2FS»

Регистрация в системе

1. Зарегистрируйтесь в первом тестовом терминале с правами **root.** Пароль администратора будет сообщен вам преподавателем или введен им.
2. С помощью **Ctrl+Alt+F2 (Alt+F2)** откройте второй текстовый терминал и зарегистрируйтесь как **user1**.

Исследование файловой системы с помощью редактора lde

3. С правами **root** создайте в каталоге **/mnt** подкаталог **floppy**. Смонтируйте предоставленную преподавателем дискету к указанному подкаталогу с помощью команды **`mount -t auto /dev/fd0 /mnt/floppy`**. Скопируйте с дискеты два исполняемых файла: **fillfile** и **lde** в каталог **/sbin**. Программа **lde** представляет собой один из распространенных редакторов файловой системы **ext2fs (linux disk editor**). С помощью программы **fillfile** вы будете создавать файлы с произвольным наполнением. Размонтируйте дискету командой **`umount /mnt/floppy`** (до размонтирования не извлекать!) и верните ее преподавателю.
4. 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)?
5. 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 по этому и предыдущему пунктам задания?
6. С помощью команды **cat** прочитайте файл **/etc/fstab**. Зафиксируйте в отчете смонтированные устройства и файловые системы, а также права на монтирование. Запомните наименование логического раздела жесткого диска, на котором смонтирована файловая система **EXT2FS** (например, **/dev/hda3**). К этому имени файла-устройства вы будете обращаться, запуская **linux disk editor**. Если на логических разделах смонтирована журнализируемая файловая система, например EXT3FS или ReiserFS, обратитесь к преподавателю. В журнализируемом варианте файловой системы вам, возможно, не удастся полноценно провести наблюдение за логическим удалением и созданием файлов.
7. 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 байт, который содержит одни символы **b**. С помощью таких файлов с определенным наполнением вам будет удобно наблюдать за дисковым пространством.

8. 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 **`lde –i inode_number /dev/hdXX`** выведите таблицу **inode** данного файла. По размеру файла и иным признакам убедитесь в том, что вы действительно наблюдаете индексный дескриптор данного 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.

9. С помощью команды **`lde –b block_number /dev/hdXX | more`** выведите поэкранный дамп блока данных файла. Убедитесь в том, что он действительно заполнен определенными повторяющимися символами.
10. Командой **`rm file_name`** удалите созданный файл. С помощью дискового редактора вновь посмотрите таблицу **inode** и блок данных данного файла. Что с ними произошло?
11. 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 байт).
12. С помощью редактора посмотрите таблицу **inode** и блок данных созданного файла. Убедитесь в том, что это действительно он. Остался ли в блоке данных информационный «мусор» от прежнего файла? Почему?
13. Вновь удалите созданный файл. Создайте третий файл с иным именем и размером более одного блока (например, 5000 байт). Убедитесь в наличии и заполнении файла, зафиксируйте номер последнего блока данных. Вновь удалите файл.
14. Создайте четвертый файл с отличающимся именем и размером менее одного блока. После того как вы убедитесь в создании очередного файла, посмотрите содержимое последнего блока, оставшегося от третьего файла. Какие вы можете сделать выводы?
15. Создайте пятый файл с размером 10000 байт. 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 **lde –i ….**, дополнительных атрибутов файла не отображают.
16. 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).
17. С помощью команды **`lde –b 0 /dev/hdXX | more`** запустите дисковый редактор на поэкранный вывод содержимого суперблока (блок номер 0).

Первые 1024 байт (до 400h) блока отведены для размещения загрузчика (**LILO - linux loader**). Суперблок начинается со смещения 400h и имеет размер 1024 байт. С помощью справочных данных о файловой системе **EXT2FS** исследуйте первые 80 байт суперблока (остальные байты не используются). Обратите внимание на то, что дамп памяти выводит шестнадцатеричные слова в обратном порядке. В результате исследования полей суперблока письменно ответьте на вопросы:

- Сколько файлов может быть создано в данной файловой системе?
- Чему равен размер одного блока?
- Какой объем диска зарезервирован для нужд суперпользователя?
- Сколько блоков на момент исследования занято?
- Сколько еще раз может быть смонтирована файловая система? Что произойдет после последнего монтирования?
- Когда в последний раз проверялась файловая система? Каковы результаты проверки?

18. Рассмотрите описатель группы блоков **Group Descriptors**, который в зависимости от используемого дистрибутива и версии ОС может располагаться либо по смещению **800h** в нулевом блоке, либо в начале первого блока. Найдите первые три элемента размером по 4 байта (они также записаны в обратном порядке). Прочитайте их и запишите в отчет:
- шестнадцатеричный адрес битовой карты блоков (**block bitmap**),
- адрес битовой карты индексных дескрипторов (**inode bitmap**),
- адрес таблицы индексных дескрипторов (**inode table**).

19. Посмотрите шестнадцатеричный дамп одного из индексных дескрипторов в **inode table** (записи в блоке идут с интервалом в **80h** байт. Несколько десятков первых inode желательно пропустить)**.** С помощью таблицы, приведенной в справочных данных, определите и отразите в отчете:
- тип этого файла и права доступа к нему,
- идентификатор владельца (выяснить, кому он принадлежит),
- размер файла в байтах,
- число жестких ссылок на **inode**,
- число блоков, занимаемых файлом,
- номера блоков данных с непосредственной и косвенной адресацией.

20. С помощью редактора **lde** просмотрите несколько блоков данных исследуемого файла.
21. По имеющимся у вас данным определите, **inode** какого файла вы рассматривали. Укажите в отчете полный путь к этому файлу.
22. С помощью редактора посмотрите, что представляет собой файл каталога. Для этого, воспользовавшись командой **`ls -li /`,** найдите номер **inode** одного из подкаталогов в корневом каталоге, откройте его с помощью **lde -i** …, найдите номер непосредственно адресуемого блока и изучите его. Сравните с форматом каталога, приведенным в теоретической части пособия. Сделайте выводы.

23. Аналогичным способом посмотрите, что представляет собой файл символической ссылки. Обратите внимание на то, где и в каком виде в файле символической ссылки хранится путь к целевому файлу. Сделайте выводы и отразите их в отчете.
24. С помощью команды **`fdisk -l /dev/hda`** (если система установлена на другом диске, укажите иной блочный файл) посмотрите, какие логические разделы имеются на жестком магнитном диске и какие операционные системы на них установлены. Образец данных, выводимых по команде **fdisk**, приведен на листинге 19.
25. Создайте в файловой системе новый пустой каталог **`mkdir /mnt/abcd`**, который будет служить точкой монтирования. Используя команду **`mount -t type device dir`,** где **type** – монтируемая файловая система (ntfs, vfat и др.), **device** – файл блочного устройства, соответствующий монтируемому логическому разделу, а **dir** – созданная точка монтирования, примонтируйте к дереву каталогов Linux файловую систему ОС Windows*. После монтирования запустите оболочку **Midnight Commander** и посмотрите каталоги и файлы примонтированной системы.
26. Демонтируйте файловую систему Windows* с помощью команды **`umount device`** или **`umount dir`**. К моменту монтирования все каталоги и файлы системы должны быть закрыты.
27. Создайте в корневом каталоге новый каталог для удаляемых файлов с именем **/erase**, принадлежащий администратору, и правами доступа 1703. Сделайте команду **rm** недоступной для пользователей. Напишите сценарий, позволяющий заменить опасную утилиту удаления файлов и каталогов **rm**.
28. С помощью команды **find** с правами администратора найдите в корневом каталоге файлы:

- имеющие атрибуты **SUID** (**`find / -type f -perm -4000`**);
- файлы, которые разрешено модифицировать всем (**`find / -type f -perm -2`**);
- файлы, не имеющие владельца (**`find / -nouser`**);
- объясните, какой интерес могут представлять для администратора указанные категории файлов?

Контрольные вопросы

- 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?
- Какую роль в монтировании сменных носителей и логических разделов жесткого диска играет файл **fstab**? Чем отличаются автоматическое и «ручное» монтирование?
- Почему дискету не следует извлекать из дисковода до ее размонтирования? Что произойдет в случае пренебрежения этим правилом?

- В параметрах файла **fstab** могут быть указаны разрешения монтирования сменных машинных носителей для **user** и **users**. Чем отличаются эти разрешения?
- Механизмы образования и удаления информационного «мусора» в файловых системах ОС **Linux**.
- Экономно ли используется дисковое пространство в ОС **Linux**? Приведите доказательства своего ответа.
- Пользователь может читать и записывать информацию на сменные машинные носители, даже если ему запрещено их монтировать. Как можно запретить использование сменных МНИ?
- Какую опасность для системы представляют файлы, не имеющие владельца?
- Как можно найти файл по его **inode**?

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

Регистрация в системе

1. Зарегистрируйтесь в первом текстовом терминале с правами root. Пароль администратора будет сообщен вам преподавателем или введен им.
2. С помощью Ctrl+Alt+F2 (Alt+F2) откройте второй текстовый терминал и зарегистрируйтесь как user1.

Работа с поврежденными гибкими магнитными дисками

3. Получите у преподавателя две дискеты: одну сбойную с информацией, которую необходимо спасти, вторую - заведомо исправную, но не отформатированную. Отформатируйте исправную дискету с помощью команды **`superformat /dev/fd0`** или **`fdformat /dev/fd0.`**
4. С помощью команды **dd** поблочно скопируйте поврежденную дискету в файл **/home/user1/floppy**. Количество копируемых блоков зависит от состояния поврежденной дискеты. При копировании неповрежденной дискеты укажите количество блоков, равное 100. Если часть дискеты механически удалена, во избежание повреждения магнитных головок и износа обрезанного края дискеты ограничьтесь копированием 10 блоков размером 512 байт. Обязательно укажите последнюю опцию команды **`conv=noerror`**, позволяющую пропускать отсутствующие и поврежденные блоки.
5. С помощью команд **cat** попытайтесь прочесть созданный файл-образ дискеты. Убедитесь в том, что дискета была скопирована правильно (в противном случае попытку создания файл-образа придется повторить). Размер полученного файла должен соответствовать суммарному объему правильно прочитанных секторов дискеты.
6. С помощью команды **cat** скопируйте файл-образ на предварительно размеченную дискету (**`cat file_name > /dev/fd0`**).
7. Примонтируйте исправную дискету к точке монтирования **/mnt/floppy**. Монтирование завершится удачно, если служебные области дискеты, по которым определяется файловая система, будут исправны. В противном случае можно ограничиться только «спасенными» данными.
8. Просмотрите файловую систему на дискете, попробуйте просмотреть отдельные файлы.
9. Размонтируйте дискету. Сделайте выводы в отношении возможностей восстановления компьютерной информации на поврежденных сменных носителях.
10. 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 байт. Убедитесь в их создании. С помощью редактора lde посмотрите индексный дескриптор и блок данных одного из них.

11. С помощью утилиты **fillfile** в рабочем каталоге пользователя создайте дополнительно три файла с длинными именами (например, **abcdefghijkl**).
12. Найдите и просмотрите блок данных, выделенный рабочему каталогу пользователя. Найдите в нем и исследуйте записи о всех созданных вами файлах.
13. Зафиксируйте в отчете номера **inode** и логических блоков, выделенных каждому созданному файлу. Эти данные вы будете использовать при анализе фрагментов удаленных файлов.
14. 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.
15. 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.
16. С помощью редактора lde 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.
17. 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.
18. 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.
19. Вводя команду отладчика **lsdel** посмотрите список **inode** удаленных файлов. Найдите среди них номера файлов, которые были созданы и удалены вами.
20. 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.
21. Если файл подлежит восстановлению, установите в «1» бит соответствующего inode в битовой карте индексных дескрипторов. Делается это с помощью внутренней команды отладчика **`seti <inode>.`**
22. С помощью команды отладчика **`mi <inode>`** сбросьте время удаления файла и установите в «1» число ссылок на файл.
23. Попытайтесь восстановить символьное имя удаленного файла. Это делается с помощью команды отладчика **`ncheck <inode>.`**
24. Командой **close** закройте логический раздел EXT2FS на жестком диске и затем, вводя **quit,** выйдите из среды отладчика debugfs.
25. Выведите листинг рабочего каталога пользователя и проверьте, удалось ли вам восстановить удаленный файл. Сделайте выводы и отразите их в отчете.
26. Повторите операцию восстановления для другого удаленного файла.

27. Предложите собственную методику восстановления данных удаленного файла, если его индексный дескриптор уже оказался занятым.
28. 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

- С какого времени файл можно считать логически удаленным? Каковы признаки логического удаления файла?
- В какой последовательности «исчезают» компоненты логически удаленного файла?
- Какие методики восстановления логически удаленных файлов вы знаете? В чем они заключаются?
- Каким образом можно найти блоки данных, ранее принадлежавшие удаленному файлу (предполагаем, что файл был текстовым и его содержание приблизительно известно).
- Каким образом можно «спасти» остатки удаленного файла, если его индексный дескриптор уже занят?
- Как скопировать данные с поврежденного машинного носителя?
- Какие требования предъявляются к гарантированному удалению конфиденциальной информации?
- Какие программные средства гарантированного удаления данных имеются в операционных системах Linux?
- Восстановите поврежденный или логически удаленный файл, предложенный преподавателем.

ЛАБОРАТОРНАЯ РАБОТА № 4
«Администрирование операционной системы Linux»

Подготовка к работе

1. Зарегистрируйтесь в системе в консольном режиме с правами **root.** Пароль администратора будет сообщен вам преподавателем.

Работа с учетными записями пользователей

2. Используя команду **cat** с правами **root,** просмотрите содержимое файла **/etc/passwd.** *Файл представляет собой таблицу, каждая строка которой представляет отдельную учетную запись, состоящую 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** вместо пароля указывает на то, что хэшированный пароль находится в другом файле - **/etc/shadow**.*
3. Аналогично изучите содержимое файла **/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 1.01.70 г. до дня последнего изменения пароля,*
   - *минимальное число дней действия пароля со дня его последнего изменения,*
   - *максимальное число дней действия пароля,*
   - *за сколько дней до устаревания пароля 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.

4. 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 гри-

фом. 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 работать 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 модифицировать 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.

5. 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?
6. 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`).**
7. Создайте учетные записи для вышеуказанных десяти новых пользователей. Регистрационные данные (кроме паролей и групп) сведены в таблицу. 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 схему, поясняющую разграничение доступа сотрудников подразделения к компьютерной информации.
8. Пять первых пользователей (**w_gromov, n_kalinina, e_ivanova, r_klinova, b_rebrov**) зарегистрируйте с помощью команды **useradd.** Синтаксис команды: **`useradd -u UID -g group_name -d dir_home -m -p password -e date_del_user user_name.`** Например, **`useradd -u 501 -g sigma -d /home/n_kalinina -p v5g7K2S4 -e 2004-01-07 n_kalinina`**. Параметр **-m** обеспечивает создание домашнего каталога пользователя, если он еще не существует. Прочие параметры команды можно не указывать. *Помните, имя пользователя не должно начинаться с цифры и содержать заглавных и русских букв, символов типа *#%^.... Идентификаторы пользователей **UID** назначаются, начиная с 500.* Дата удаления учетной записи пользователя вводится в формате ГГГГ-ММ-ДД.
9. 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 пара-

метра по умолчанию вводить **Enter**. *В некоторых дистрибутивах ОС **Linux** командный файл **adduser** отсутствует, и в этом случае регистрацию остальных пользователей следует произвести аналогично пункту 8. При использовании **adduser** символ подчеркивания в имени пользователя не воспринимается. Если это так, замените этот символ в именах пользователей на символ дефиса.*

Таблица 1

Пользователи	**UID**	**Пароль**	**Группа**	**Домашний каталог**	**Дата удаления учетной записи**
i_frolov	**501**			/home/i_frolov	T+ 10 дней
m_kruglov	**502**			/home/m_kruglov	T+ 30 дней
b_rebrov	**503**			/home/b_rebrov	T+ 12 дней
d_lavrov	**504**			/home/d_lavrov	T+ 60 дней
e_ivanova	**505**			/home/e_ivanova	T+ 30 дней
t_uporov	**506**			/home/t_uporov	T+ 15 дней
k_beglov	**507**			/home/k_beglov	T+ 45 дней
n_kalinina	**508**			/home/n_kalinina	T+ 30 дней
r_klinova	**509**			/home/r_klinova	T+ 90 дней
w_gromov	**510**			/home/w_gromov	не удалять

10. Переключаясь во вторую консоль, отслеживайте изменения, происходящие в файле **/etc/passwd** по мере ввода новых учетных записей.
11. Попробуйте с правами пользователя посмотреть файл **/etc/shadow**. Повторите попытку просмотра с консоли суперпользователя. Почему поля, отведенные для хэшированных паролей у пользователей **e_ivanova и r_klinova** различаются? *Обратите внимание на то, что во многих дистрибутивах ОС утилита **useradd** работает некорректно и записывает пароли в файл 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. Администратору в этом случае рекомендуется в ответ на соответствующий вопрос **useradd** ввести пустой пароль (в файле **/etc/passwd** в поле признака пароля при этом будет стоять восклицательный знак), а затем устанавить пароль пользователю с помощью команды **passwd.***
12. 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? (В различных дистрибутивах Linux возврат полномочий администратора организован различным образом).
13. Запустите оболочку **Midnight Commander** в режиме редактирования (F4) файла паролей **/etc/passwd** и удалите в учетной записи пользователя **n_kalinina** символ признака пароля (между первым и вторым двоеточием), включая пробел. Сохраните изменения в файле, завершите сеанс в **Midnight**

Commander, с помощью **Ctrl+Alt+F2** (**Alt+F2**) откройте второй текстовый терминал и зарегистрируйтесь пользователем **n_kalinina**, но теперь с «пустым» паролем. Сделайте вывод относительно опасности предоставления прав на запись в этот файл. Завершите сеанс для пользователя **n_kalinina** с помощью команды **exit**.

14. Пользователь **d_lavrov** уволен за дисциплинарный проступок. С помощью команды **`userdel –r user_name`** удалите его учетную запись вместе с домашним каталогом. В реальных условиях необходимо вначале скопировать в другую директорию файлы пользователя, представляющие ценность для организации.
15. Зарегистрируйте вместо уволенного пользователя нового сотрудника **f_mironov** с предоставлением ему аналогичных прав (пароль должен быть новым!).
16. Пользователь **r_klinova** убыла в командировку сроком на две недели. Заблокируйте ее учетную запись, для чего с правами администратора войдите в режим редактирования файла паролей и вставьте во второе поле (между первым и вторым двоеточием) любой символ, который не разрешено использовать для пароля. Попытайтесь зарегистрироваться во второй консоли с правами **r_klinova** и убедитесь в том, что для этого пользователя система недоступна.
17. Зарегистрируйтесь во второй консоли с правами пользователя **k_beglov,** вызовите команду **passwd** и измените свой пароль. В качестве нового пароля введите **qwerty**.
18. 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?
19. Просмотрите электронную справку по файлу **/etc/sudoers.** Отредактируйте его таким образом, чтобы предоставить следующим пользователям дополнительные права за счет использования команды **sudo:**

- пользователю **e_ivanova** - право монтировать файловые системы,
- пользователю **b_rebrov** - право изменения владельца файлов.

Ответьте, чем отличается предоставление прав пользователям с помощью **sudo** от использования эффективных идентификаторов **SUID**?

20. Из второй консоли с правами пользователя **f_mironov** создайте файл **`cal 2004 > /home/f_mironov/cal2004.`** 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?

21. Просмотрите с правами администратора системные журналы в каталоге **var/log** и убедитесь, что система зафиксировала факты присвоения полномочий администратора.

Контрольные вопросы

- Достаточно ли трех базовых прав доступа к файлам для реализации в ОС **Linux** требуемой политики безопасности?
- Какие изъяны вы усматриваете в использованных утилитах регистрации учетных записей пользователей?
- 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?
- Существуют ли способы ограничения доступа суперпользователя к некоторым конфиденциальным файлам?
- 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 правами **root.**
2. Комбинацией клавиш **<Ctrl+Alt+F2>** (**<Alt+F2>**) откройте второй текстовый терминал и зарегистрируйтесь как **user1**.
3. С правами **root** создайте в каталоге **/mnt** подкаталог **floppy**. Смонтируйте предоставленную преподавателем дискету к указанному подкаталогу с помощью команды **`mount -t auto /dev/fd0 /mnt/floppy`**. Скопируйте с дискеты один исполняемый файл с именем **signorer** в каталог **/bin.** Размонтируйте дискету командой **`umount /mnt/floppy`** (до размонтирования не извлекать!) и верните ее преподавателю.

Наблюдение за файловой системой /proc

4. Выведите на экран содержимое файловой системы **/proc** (**`ls -la /proc`**). Обратите внимание на то, что в строках большей части таблицы выводится информация не о файлах, а об исполняемых на данный момент процессах. Поэтому размер этих “файлов” равен нулю, а в последнем столбце таблицы вместо имени файлов указаны идентификаторы процессов **PID**. Для каждого работающего в системе процесса в каталоге **/proc** существует подкаталог, имя которого совпадает с идентификатором этого процесса.
5. Откройте несколько таких подкаталогов подряд. Обратите внимание на то, что все подкаталоги содержат одни и те же «файлы» (см. табл. 1).
6. 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?

Изучение и анализ отображаемой информации о процессах

7. 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),
- по состоянию **S:** (исполняющиеся - **R или O**, ожидающие записи на диск - **D**, ожидающие событий - **S**, приостановленные - **T**, зомби - **Z** и т.д.),
- по текущему динамическому приоритету **PRI** (наименьшее значение у высокоприоритетных процессов),
- по относительному приоритету NI.

8. Комбинацией клавиш **<Ctrl+Alt+F3>** (**<Alt+F3>**) откройте третий текстовый терминал и зарегистрируйтесь в нем как суперпользователь. В этой кон-

соли запустите утилиту **top** для текущего контроля процессов. Утилита позволяет отобразить наиболее активные процессы (столько, сколько их помещается на экран) с достаточно полной информацией о них (для пользователя утилита представляет ограниченный набор выводимых параметров).

9. Из первой консоли создайте процесс **`od /dev/zero > /dev/null.`** В соответствии с введенной командой утилита **od** читает и выводит непрерывный поток нулевых байт 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**.
10. 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.
11. 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?
12. С консоли пользователя **user1** измените приоритет одного из принадлежащих ему процессов. Для этого воспользуйтесь командой **`renice –10 PID`**. Изменился ли относительный приоритет процесса?
13. Повторите предыдущий пункт с правами администратора.
14. Переключитесь в консоль пользователя и измените приоритет одного из принадлежащих ему процессов командой **`renice 5 PID.`** Произошло ли изменение приоритета?
15. Проконтролируйте из третьей консоли изменение приоритетов запущенных процессов.
16. Удалите созданные процессы командой **kill.**

Управление процессами

17. 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!
do
echo HELLO!
done
Ctrl+d
```

Используя команду **chmod**, присвойте пользователю полные права на чтение, запись и исполнение данного сценария. Запустите сценарий на исполнение (на экран должны непрерывно выводиться приветствия **HELLO!**)

18. 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**.
19. Нажатием **Ctrl+C** из второй консоли остановите процесс. Как изменилось при этом состояние процесса?
20. Повторно запустите из второй консоли процесс, перейдите в первую консоль и отправьте "зависшему" процессу сигнал на останов (команда **`kill -15 PID_process`**).
21. 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?
22. 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.
23. 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?`**
24. 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?
25. Попробуйте запустить несколько утилит из второй консоли с правами пользователя (например, **`renice -10 PID, date -s 0`**). Как реагирует система на даши попытки?
26. Перейдите в первую консоль и повторите запуск утилит с правами суперпользователя.
27. 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 запуска (в ядре операционной системы, в командной оболочке, в самой утилите?). Ответ обоснуйте.
28. 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?

29. 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 каналами

30. Создайте неименованный канал. Для этого можно использовать команду **`od /dev/zero | tr . 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?
31. Остановите запущенный процесс комбинацией клавиш **<Ctrl-C>.**
32. Изучите принцип 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`**

Контрольные вопросы:

- Чем отличаются системные процессы, «демоны» и пользовательские процессы?
- Какую информацию можно извлечь из каталога **/proc**?
- Как система распределяет процессорное время между конкурирующими процессами? Как на это влияет приоритет процессов?

- Почему пользователю не разрешается повышать приоритет процессов? Усматриваете ли вы какую-либо опасность в возможности понижения приоритета пользовательских процессов?
- В консоли, с которой вы работаете, произошло «зависание», вызванное неправильным исполнением одного из ваших процессов. Каким образом можно повлиять на возникшую ситуацию?
- Какие угрозы безопасности связаны с использованием эффективного идентификатора **SUID**? Почему администратор должен учитывать и контролировать процессы с таким идентификатором?
- В чем различие между именованным и неименованным каналами? Для чего они используются?

Библиографический список

1. UNIX: руководство системного администратора. Для профессионалов. 3-е изд. /Э.Немет, Г.Снайдер, С.Сибасс, Т.Хейн. СПб.: Питер; Киев: Издательская группа BHV, 2003. 925 с.
2. Бэндл Д. Защита и безопасность в сетях Linux. Для профессионалов / Д. Бэндл. СПб.: Питер, 2002. 480 с.
3. Робачевский А.М. Операционная система UNIX / А. М. Робачевский. СПб.: БХВ-Санкт-Петербург, 2000. 528 с.
4. Костромин В. А. ОС Linux на вашем персональном компьютере / В. А. Костромин (электронный вариант книги).
5. Митчелл М. Программирование для Linux. Профессиональный подход: пер. с англ. / М. Митчелл, Дж. Оулдем, А. Самьюэл. М.: Издательский дом «Вильямс», 2003. 288 с.
6. Таненбаум Э. Современные операционные системы. 2-е изд. / Э. Таненбаум. СПб.: Питер, 2002. 1040 с.
7. Мак-Клар С. Секреты хакеров. Безопасность сетей готовые решения.: пер. с англ. 2-е изд. / С. Мак-Клар, С. Скембрей, Д. Курц. М.: Изд.дом “Вильямс”, 2001. 656 с.
8. Фридл Дж. Регулярные выражения. Библиотека программиста / Дж. Фридл. СПб.: Питер, 2001. 352 с.
9. Glover Robin (31 Jan 1996), HOW-TO : Undelete linux files (ext2fs/debugfs), comp.os.linux.misc Usenet posting.
10. Peek Jerry, Tim O'Reilly, Mike Loukides et al (1993), UNIX Power Tools O'Reilly and Associates, Inc./Random House, Inc., ISBN: 0-679-79073-X. Second edition, 1998.
11. Режим доступа: news://comp.os.linux.misc/
12. Режим доступа: www.kiev.epos.us

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 восьмиричного представления файла. Обычно используется для двоичных файлов.

"Навигация" по файловой системе

cd [dir] – изменение текущего каталога. Варианты: **cd** – переход в «домашний» каталог, **cd /** - переход в корневой каталог, **cd ..** – переход 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 файлами, каталогами и ссылками

mkdir [arg] [dir] – создание нового каталога. Атрибут –m mode задает права доступа к каталогу. Пример: **mkdir –m 1555 /home/user1**.
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, **–u** – без этого аргумента удаление не происходит, **–n** раз – число повторов.
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 каталога 1 в каталог 2
[**–a** – сохранение атрибутов файла, **–p** – сохранение режима доступа к файлу, его принадлежности и временных отметок (без параметра файл переходит в собственность копирующего)].

mv [arg] file1 file2 – изменение имени файла.
mv [arg] file_name [dir] – перемещение указанного файла в другой каталог.
ln [arg] [file_name] [link] – создание ссылки на файл (**-s** – создание символической ссылки).
chattr +(-)[arg] file_name – установка дополнительных атрибутов файла (**-i** – блокирование любых изменений файла, **-a** – запрет любых операций, кроме добавления данных, **-c** – автоматическое сжатие и декомпрессия при записи/чтении, **-s** – гарантированное стирание блоков данных при удалении файла. Знак "+" означает присвоение атрибута, знак "-" - его удаление).
ls [arg] [dir] – вывод списка файлов в директории (-**l** – подробная информация, **-a** – все файлы и подкаталоги, **-i** - inode).
ls [arg] -li - получение подробной информации о конкретном файле.
ls /dev/hd* - получение информации о логических и физических IDE-дисках.
lsattr *[arg]* [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 -l device – вывод информации о логических разделах на физическом диске.

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.
dd if=/dev/fd0 of=file_name skip=n count=m block=b conv=noerror - поблочное чтение носителя с игнорированием ошибок чтения (где **if –** откуда**, of –** куда**, n -** число пропущенных блоков**, m -** число копируемых блоков, **b** – размер блока в байтах (по умолчанию **b** = 512 байт))**.**

Изменение прав доступа

chmod mode filename – изменение прав доступа к объекту.
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 указанных прав вместо имеющихся,
- вместо ***p*** - подставляется символ, обозначающий соответствующее право: **r** (чтение), **w** (запись), **x** (выполнение), **s** (бит SUID - только для **user**), **t** (sticky bit - только для **all**). 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 filename – передача прав на файл другому владельцу.

Работа с процессами

ps [arg] (process status)– вывод информации о существующих процессах **(-a** или **-e** - все процессы, -**f** - полный листинг, **-l** - расширенный листинг).

kill [sign] pid – сигнал процессу с указанным идентификатором процесса **pid (sign = 15 -** послать сигнал о завершении**, 9 -** "убить" процесс**). Kill -9 0** - остановка всех активных процессов пользователя.

Работа с пользователями и паролями

groupadd -g GID group_name – создание новой группы пользователей.

groupdel group_name – удаление зарегистрированной группы

id (identifier**)** – вывод имен и числовых идентификаторов текущего пользователя и его групп.

su (substitute user) - подстановка пользователя (с последующим запросом и вводом пароля).

su - попытка приобрести полномочия суперпользователя (администратора).

su -user_name - смена пользователя со сменой пользовательского окружения.

su user_name - смена пользователя без смены окружения. Если команду вводит **root**, система пароль не запрашивает.

passwd - смена пароля (будет предложено ввести старый, а затем новый пароль). Администратор может указать команду с именем конкретного пользователя и изменить ему пароль.

chage (change aging**)** – получение или изменение информации об устаревании пароля.

adduser - добавить учетную запись пользователя (обычно ввод данных производится в интерактивном режиме).

useradd -u UID -g group_name -G add_group_name -d dir_home -m -p password -e date_del_user user_name – создание новой учетной записи пользователя**.**

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).

Монтирование и размонтирование файловых систем

mount –t type –o option <устройство> <точка_ монтирования>
type - тип монтируемой системы (**ext2, ext3, msdos, vfat, ntfs** и т.д.). Тип **auto** - предоставление системе возможности автоматически (по записи в таблице /etc/fstab) определить монтируемую файловую систему.
option - дополнительные опции (**ro** – только чтение, **rw** – чтение и запись, и т.д.)
<устройство> - имя специального файла устройства, например **/dev/hda2**;
<точка_монтирования> - имя каталога, к которому будет монтироваться файловая система (например, **/mnt/floppy** или **/mnt/ntfs**).
mount –t auto /dev/fdo /mnt/floppy - монтирование дискеты (после монтирования к дискете нельзя обращаться как к устройству).
umount <устройство> или **umount <точка_монтирования>** - размонтирование файловой системы.
umount /mnt/floppy - размонтирование дискеты.

"Горячие" клавиши и общие команды

Ctrl+Alt+Fn = Alt+Fn - переключение на другой терминал.
Ctrl+D или **logout** или **exit** - завершить работу 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 из 1000 команд).
↑ - возврат к предыдущей команде.
man name_command - вызов электронной справки по нужной команде. Выход из электронного справочника по **q**.
command_name --help - вызов краткой справки о команде.
shutdown –h +t - полная остановка через t минут.
shutdown –h 0 или **halt** или **Ctrl+Alt+Del** - немедленная полная остановка.
shutdown –r или **reboot** - перезагрузка операционной системы.

СПРАВКА ПО ОТЛАДЧИКУ DEBUGFS

DebugFS является самой известной утилитой, предназначенной для работы с файловыми системами EXT2FS и EXT3FS. Для ReiserFS эта утилита не годится, и для нее в современных дистрибутивах Linux предусмотрен свой отладчик.

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 тоже 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 оболочка Bash воспринимает как команды перенаправления ввода/вывода.

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

`debugfs –R stats /dev/hdc3` - выводит достаточно полную информацию о суперблоке и всех группах блоков (файл устройства указан для конкретного случая). Фрагмент, выведенный такой командой, приведен в листинге 8.

а) В интерактивном режиме вход в среду отладчика производится командой

`debugfs`

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/hdc3` (указывается конкретный логический раздел блочного устройства). Можно сразу воспользоваться командой **`debugfs –w /dev/hdc3`**.

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 печальным 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 (manual – руководство). Рассмотрим эти команды (они сгруппированы по выполняемым функциям или последовательности применения):

`clri filt_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 – полезная 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 также 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. Гораздо лучше воспользоваться аналогичной командой редактора Lde,

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> - очень нужная 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.

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] 0
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.

В некоторых случаях гораздо удобнее пользоваться редактором **Lde** непосредственно из командной строки оболочки. В обиходе вполне достаточно двух команд:

`lde -i inode_number device` – вывести информацию, содержащуюся в 128-байтном фрагменте inode.

`lde -b block_number device | more -` вывести поэкранный дамп логического блока с указанным номером. Обычно логические блоки нумеруются шестнадцатеричными числами, в этом случае номер блока указывать в виде **0x12345678**.

Учебное издание

Валентин Викторович Бакланов

Администрирование и безопасность операционных систем Linux

Редактор *Н. П. Кубыщенко*

Компьютерная верстка автора

Подписано к печати 16.06.2005			Формат 60x84 1/16
Бумага типографская	Офсетная печать		Усл. печ. л. 5,41
Уч. - изд. л. 6,0	Тираж	Заказ	Цена “С”

Редакционно-издательский отдел ГОУ ВПО УГТУ-УПИ
620002, Екатеринбург, ул. Мира, 19
Ризография НИЧ ГОУ ВПО УГТУ-УПИ
620002, Екатеринбург, ул. Мира, 19