Министерство образования и науки Республики Казахстан

Некоммерческое акционерное общество
«Алматинский университет энергетики и связи»

Е.А. Зуева

ОПЕРАЦИОННАЯ СИСТЕМА LINUX

Учебное пособие

Алматы
АУЭС
2017

УДК 004.451.9Linux(075.8)
ББК 32.973.26-018.2
З-93

Рецензенты:
доктор технических наук, профессор кафедры информатики КазНУ им. аль-Фараби
Шмыгалева Т.А.,
доктор PhD, асс. проф. КазНАУ, зав. каф. «Энергообеспечение и автоматика» КазНАУ
Шыныбай Ж.С.,
профессор кафедры «Информационные системы» АУЭС
Аренбаева Ж.Г.

Рекомендовано к изданию Ученым Советом Алматинского университета энергетики и связи (Протокол № 5 от 31.01.2017). Печатается по дополнительному плану выпуска ведомственной литературы АУЭС на 2017 год, позиция 2.

Зуева Е.А.

З-93 Операционная система Linux: Учебное пособие (для студентов высших учебных заведений специальности «Системы информационной безопасности»)/ Зуева Е.А. - Алматы: АУЭС, 2017. – 87 с.: табл. 1, ил. 46, библиогр. – 7 назв.

ISBN 978-601-7889-33-3

В представленном учебном пособии обобщены, систематизированы и представлены сведения о современной операционной системе Linux.

Учебное пособие предназначено для студентов, обучающихся по специальности «Системы информационной безопасности».

УДК 004.451.9Linux(075.8)
ББК 32.973.26-018.2

ISBN 978-601-7889-33-3

Содержание

Введение 4
1 Установка и базовая настройка операционных систем 5
2 Управление учетными записями пользователей 12
3 Работа с папками и файлами 14
4 Монтирование от физических носителей до виртуальных жестких дисков .. 19
5 Квотирование 28
6 Управление процессами. 33
7 Графические оболочки 36
8 Эмуляторы и виртуальные машины 38
9 Работа с загрузчиками 40
10 Шифрование и файервол 63
11 Режимы загрузки, взлом ОС, деактивация пользователей 65
12 Архивация 68
13 Восстановление 69
14 Алиасы 71
15 Реализация сценариев 73
16 Автозагрузка скриптов 85
Список литературы 87

Введение

В настоящее учебное пособие включены материалы, целью которых является изучение процессов установки и базовой настройки операционной системы и «азов» работы операционной системы Linux.

Знание внутреннего устройства операционной системы Linux, правил ведения безопасной работы в системе дает студенту уверенность и грамотный и эффективный подход к организации своих действий при работе в ней. Рассматриваются такие темы, как организация работы файловых систем, настройки оболочек систем, настройки загрузчиков, монтирование и размонтирование устройств, правильная организация сетей, запуск служб работы и диагностики ОС, управление учетными записями, работа с ОС как в консоли, так и в графическом меню, работа с файловыми системами, управление процессами и пакетами программ, работа с загрузчиками.

В учебном пособии изложены основные принципы безопасной работы с ОС различной степени сложности и назначения. Студенту важно понять, как платформы отличаются друг от друга разработанными (и запускаемыми) программными компонентами, как правильно администрировать ресурсы, проводить грамотную политику работы в операционных системах, а также знать принципы построения современных ОС и системного программного обеспечения. При этом студент начинает разбираться в основных принципах организации безопасного интерфейса пользователя с программной системой; в методах анализа, исследования и моделирования вычислительных и информационных процессов, связанных с функционированием объектов профессиональной деятельности и их компонентов; в принципах, методах и способах комплексирования аппаратных и программных средств при создании защищенных вычислительных систем и сетей.

Профессиональная подготовка бакалавра требует умения грамотно работать с ОС, а также с различными операционными средами, системами и оболочками. Предметом изучения в рассматриваемой дисциплине являются ОС и средства, расширяющие их возможности.

1 Установка и базовая настройка операционных систем

Операционная система (ОС) *Linux* называется так потому, что она строится на основе ядра операционной системы, которое называется *Linux*. Это название происходит от имени первого разработчика ядра - Линуса Торвальдса и слова *Unix*. Ядро разработано под руководством Линуса Торвальдса и распространяется свободно под лицензией *GNU GPL*. Ядро - это основная программа операционной системы, обеспечивающая управление аппаратной частью компьютера, файловой системой и выполнением процессов. Одним из следствий свободного распространения ПО для *Linux* явилось то, что большое число разных фирм и компаний, а также просто независимых групп разработчиков стали выпускать свои версии *Linux* - так называемые дистрибутивы.

Дистрибутив - набор пакетов программного обеспечения, включающий ядро системы и некоторую совокупность утилит и прикладных программ. Поскольку Линус Торвальдс продолжает координировать разработку ядра, версии ядра развиваются последовательно, а дистрибутивы множатся. Существует уже больше тысячи различных дистрибутивов *Linux*. Некоторые выпускаются фирмами на коммерческой основе, другие распространяются на условиях лицензии *GNU GPL*. Пользователь может выбрать себе понравившийся дистрибутив или (если может) сам собирать систему на свой вкус.

О дистрибутивах можно прочитать в Википедии или на http://www.distrowatch.com/. Там же приведен их сравнительный анализ.

Любой из дистрибутивов имеет модификацию, запускающуюся без установки на жесткий диск, называемую *Live CD*. Они созданы для ознакомления с выбранной ОС.

Установить *Linux* можно с:

- локального *CD* или *DVD-ROM* или *flash*-накопителя (с *USB*-порта);
- через сеть *FTP* или через *SMB* с удаленного компьютера (и из Интернет).

Установить ОС можно:

а) на виртуальную машину: выбирается и устанавливается в *Windows* виртуальная машина. Затем создается новая машина и в качестве образа ОС ставится выбранная ОС: файл вида *.iso; далее конфигурируются настройки ОС и запускается ее установка;

б) на раздел жесткого диска: при разбиении разделов можно использовать встроенные средства *Windows* (правая клавиша на «*Компьютер*» - «*Управление дисками*»). Или как вариант - программы *Partition Magic, Acronis disk director suite*. После создания раздела нужно перезагрузить компьютер и в начале его загрузки удерживать нажатой клавишу «*Delete*» – чтобы зайти в *BIOS* и выставить первичной загрузкой *CD* или *DVD-ROM* (если планируется установка с *CD* или *DVD*-диска) или *USB* (если планируется установка с *flash*-карты). После сохранения изменений и выхода из *BIOS*

автоматически начинает считываться сменный носитель и начинается установка.

И в том и в другом случае в процессе установки студенту для входа в систему нужно придумать логин (фамилия_имя), пароль и выбрать оболочку, которую система будет по умолчанию использовать.

В *Linux* приняты следующие имена накопителей:

– *hda* - первый диск (*master*), на канале *IDE1*;
– *hdb* - второй диск (*slave*), на канале *IDE1*;
– *hdc* - первый диск (*master*), на канале *IDE2*;
– *hdd* - второй диск (*slave*), на канале *IDE2*;
– *hde, hdf ...* - диски на дополнительных контроллерах *IDE*;
– *sda, sdb ...* - диски на *SCSI*-интерфейсе.

Многие *IDE*- и *SATA*-контроллеры распознаются в системе как *SCSI*-устройства, поэтому часто можно видеть название диска *sda*.

Очень важный вопрос при установке ОС – правильное распределение дискового пространства: зафиксируйте - каким образом это разделение вы производите в каждой из установленных ОС, пример подобной разбивки приведен на рисунке 1.

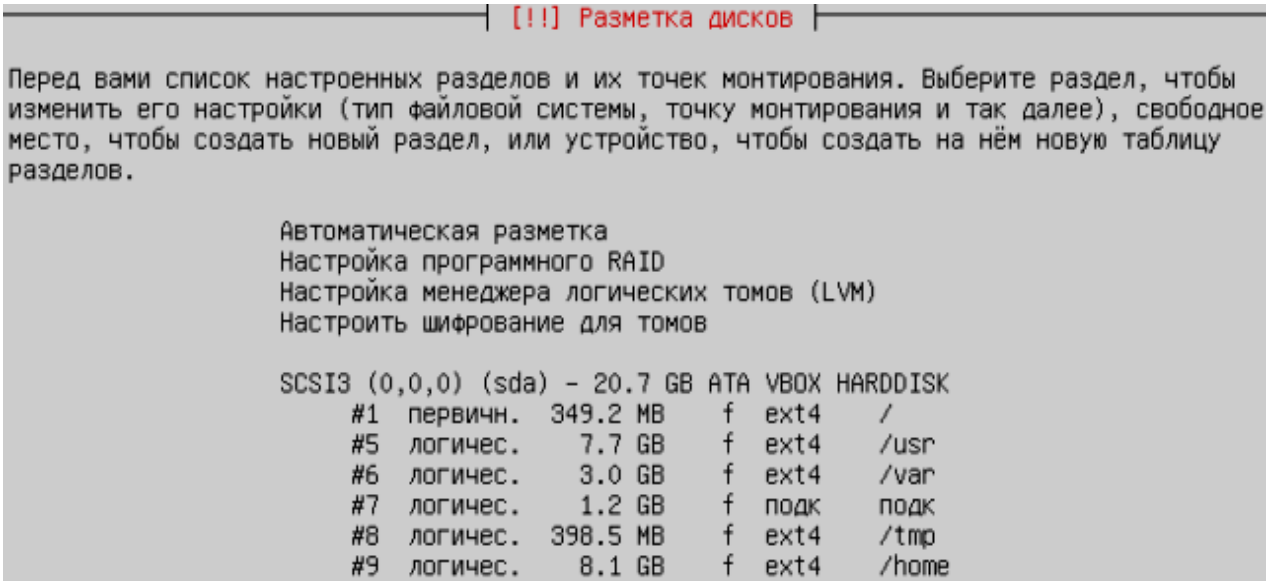

Рисунок 1 – Пример логического разбиения диска

Все собранные в одном месте каталоги (папки), подкаталоги, а также файлы есть данные файловой системы. Причем один из этих каталогов в *Linux*-системах является самым наиважнейшим, а именно «верхушкой» («корнем») файловой системы - в нём содержатся все остальные каталоги и файлы. На рисунке 2 представлены каталоги файловой системы *Linux* через файловый менеджер *Midnight Commander* (*MC*). Корень файловой системы (корневой каталог) указан красной стрелкой, к нему монтируются все другие файловые системы.

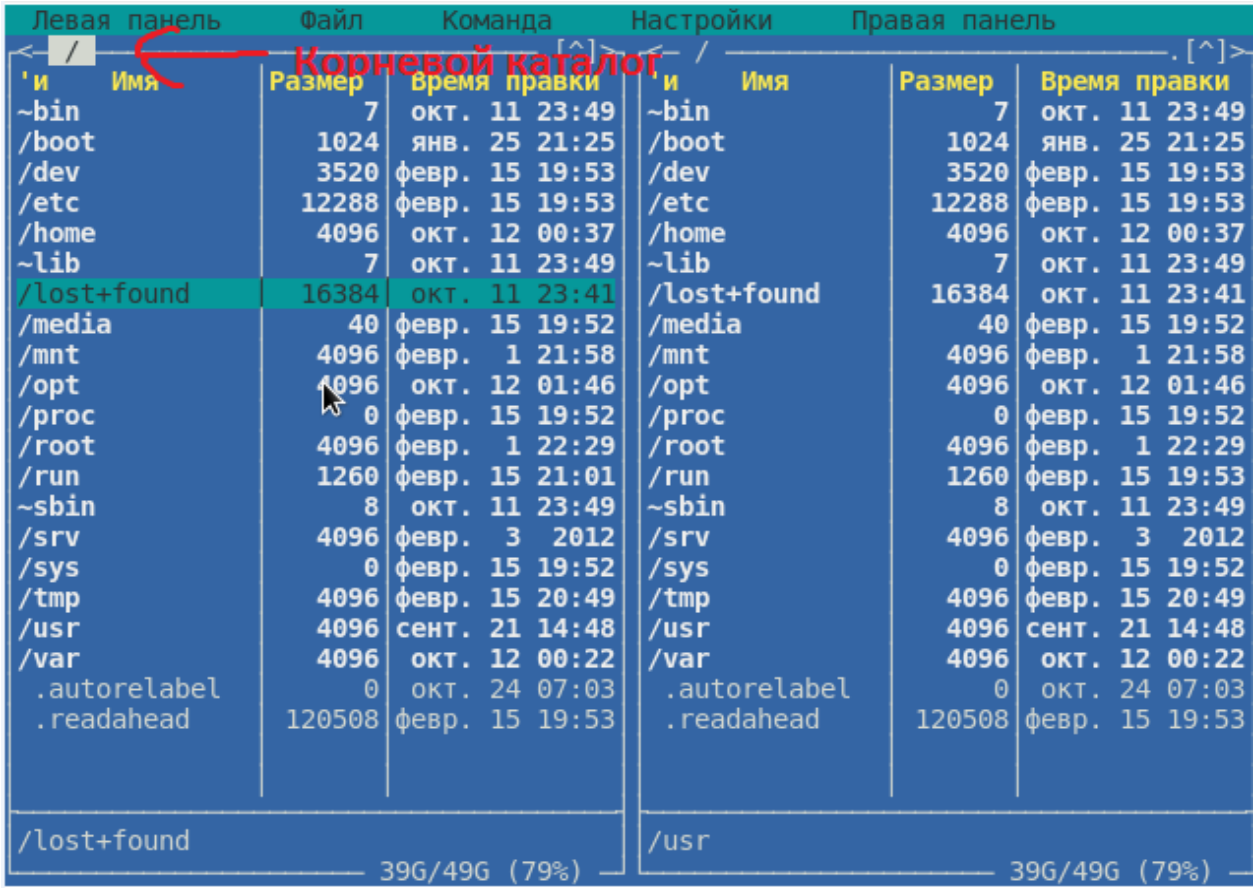

Рисунок 2 – Корень *Linux*

В разных дистрибутивах *Linux* могут содержаться разные каталоги: в некоторых есть дополнительные каталоги конфигурации, а некоторые файлы конфигурации, будут называться по-другому. Далее приведены общие папки:

/bin - основные бинарные файлы (в *Windows* такие файлы имеют расширение .exe), доступные всем пользователям, а также содержит символьные ссылки на исполняемые файлы. Символьная ссылка - специальный файл в файловой системе, внутри которого есть только одна-две строки с указанием полного пути, который должен быть открыт при попытке обратиться к данной ссылке (файлу).

/boot содержит конфигурационные файлы загрузчика (*GRUB* или *LILO*), образы ядра (по умолчанию хранятся последние 3), файлы Initrd (когда начинается начальная загрузка системы, сначала монтируется временный виртуальный диск, который содержит временную корневую файловую систему, с помощью которой, в свою очередь, осуществляется запуск ядра *Linux*; файл *Initrd* - образ этой временной корневой системы, который хранится на загрузочном устройстве).

/dev содержит файлы устройств.

/etc содержит конфигурационные файлы ОС и всех сетевых служб. Данный каталог можно сравнить с реестром *Windows*, но в *Windows* общесистемные настройки хранятся в одном большом бинарном файле, а в

Linux - в разных конфигурационных файлах, которые можно редактировать обычным текстовым редактором.

/*home* содержит домашние каталоги всех пользователей, которые зарегистрированы в системе. В домашних каталогах пользователей хранятся пользовательские файлы, а также пользовательские настройки различных программ. Такое разделение пользовательских каталогов и файлов операционной системы упрощает сохранение данных и повышает надежность самой операционной системы.

/*lib* различные библиотеки и модули ядра. В процессе установки различных программ в *Linux* (в том числе и драйверов) устанавливаются зависимости для корректной работы программы. Вот эти зависимости в большинстве случаев и есть библиотеки - набор собранных особым образом файлов, которые подключаются во время установки к устанавливаемой программе.

/*lost+found* нужен для хранения испорченных файлов при проблемах с файловой системой, которые были восстановлены после, например, некорректного размонтирования файловой системы. Удалить каталог не получится (он всё равно снова появится).

/*misc* может содержать все что угодно.

/*mnt* и /*media* содержат точки монтирования. В современных дистрибутивах *Linux* этот процесс обычно происходит автоматически. При этом в каталогах /*mnt* или /*media* создается подкаталог, имя которого совпадает с именем монтируемого тома.

/*opt* установленные программы, имеющие большой дисковый объем, или вспомогательные пакеты, например, *Adobe Reader* и *Google Chrome*. Рекомендуется размещать файлы (бинарники, библиотеки, страницы руководств и т.д.) для пакетов типа /opt/имя_пакета и их специфических конфигурационных файлов из /etc/opt.

/*proc* - каталог псевдофайловой системы *procfs*, которая используется для предоставления информации о процессах (по-другому это виртуальная файловая система, которая обеспечивает связь с ядром и монтируется в каталоге /*proc*). В *Linux* присутствует виртуальный файловый объект, именуемый каталогом /proc. Он существует только во время работы системы в оперативной памяти компьютера. Каталог представляет интерес и с точки зрения безопасности. Многие из утилит, выводящие информацию о системе, берут свои исходные данные именно из этого каталога.

/*root* - каталог пользователя *root*.

/*run* создан для хранения данных, которые были запущены приложениями, требующимися в процессе работы (это могут быть и службы, запускаемые самой системой, и программы, которые Вы запускаете сами). Сюда входят: идентификаторы процессов, информация о межпроцессорном взаимодействии, заблокированные файлы и другие данные, необходимые во время работы.

/*sbin* - набор утилит для системного администрирования, содержит исполняемые файлы, необходимые для загрузки системы и ее восстановления в различных щекотливых ситуациях. Запускать эти утилиты имеет право только root.

/*tmp* - каталог, в котором хранятся временные файлы. *Linux*, в отличие от *Windows*, следит за чистотой и регулярно очищает этот каталог.

/*usr* содержит пользовательские программы, документацию, исходные коды программ и ядра. По размеру это один из самых больших каталогов файловой системы. В этот каталог устанавливаются практически все программы. Его с большой натяжкой можно сравнить с каталогом *Program Files* в *Windows*.

/*var* содержит файлы, которые подвергаются наиболее частому изменению. Например, кэши различных программ; файлы блокировки для недопустимости одновременного использования одной программы несколькими пользователями; файлы системных журналов; временные файлы (при выключении компьютера содержимое очищается); информация о различных программах; общая информация о состоянии системы с момента последней загрузки, входа в систему и т.д.; очередь печати, факсов, а также входящие почтовые ящики пользователей и т.д.

Swap. Изначально под свопингом понималась выгрузка процесса из оперативной памяти целиком, в результате чего неактивные процессы могли полностью отсутствовать в ОЗУ. При наступлении условий активизации процесса диспетчер памяти загружал образ процесса обратно. Смысл термина изменился в 60-х годах, когда в ОС появилась поддержка виртуальной памяти: под свопингом стали понимать загрузку и выгрузку отдельных страниц. Подкачка страниц (*paging*; иногда используется *swapping* от *swap*) - один из механизмов виртуальной памяти, при котором отдельные фрагменты памяти (обычно неактивные) перемещаются из ОЗУ во вторичное хранилище, освобождая ОЗУ для загрузки других активных фрагментов памяти. Такими фрагментами в современных ЭВМ являются страницы памяти. Временно выгруженные из памяти страницы могут сохраняться на внешних запоминающих устройствах как в файле, так и в специальном разделе на жёстком диске (*partition*), называемые соответственно *swap*-файл и *swap*-раздел. В случае откачки страниц, соответствующих содержимому какого-либо файла (например, *memory-mapped files*), они могут удаляться. При запросе такой страницы она может быть считана из оригинального файла. Когда приложение обратится к откачанной странице, произойдет исключительная ситуация *PageFault*. Обработчик этого события должен проверить, была ли ранее откачана запрошенная страница, и, если она есть в *swap*-файле, загрузить ее обратно в память.

После установки системы обычно настраиваются персональные параметры, такие, как оболочка входа (обычно это *KDE* или *GNOME*), темы, количество рабочих столов, панель задач, рисунки рабочего стола и заставка,

шрифты, цветовая гамма, языковые раскладки, установка необходимого программного обеспечения и т.д.

Если на компьютере установлена одна ОС, то она загружается по умолчанию без вывода меню загрузчика. Ситуация меняется, если количество ОС больше одной. В этом случае загрузчик выводит меню для определения ОС для дальнейшей загрузки. Чтобы загрузчик в любом случае показывал свое меню при загрузке, необходимо удерживать нажатой клавишу *Shift*.

Открыть терминал в ОС можно несколькими способами. При работе с командной строкой (часто используются термины «терминал» или «консоль»), результат работы с объектами можно проверить в любом установленном оконном менеджере, например, *Midnight Commander*.

Командная строка начинается приглашением - подсказка, что система готова принимать команды пользователя (рисунок 3). Приглашение может быть оформлено по-разному, но чаще всего оно заканчивается символом “$”(права обычного пользователя) или “#”(суперпользователь).

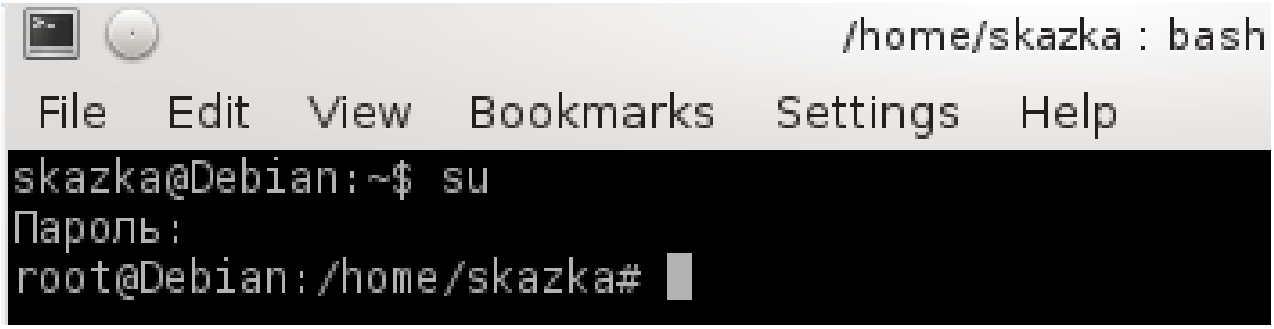

Рисунок 3 - Строка приглашения терминала, введенная команда su позволяет залогиниться под суперпользователем

Например, в конструкции *abc@def:gh* до ”@” - имя пользователя; между “@” и “:” - имя домена; после “:” - путь к папке, где находится текущий пользователь и в конце признак пользователя, например, запись *skazka@Debian:~$* означает: пользователь *skazka* сидит на машине с доменным именем *Debian*, ~ - находится в своей домашней директории (*/home/skazka*), являясь обычным пользователем. Или *root@Debian:/home/skazka#* означает: пользователь root сидит на машине с доменным именем *Debian*, находится в директории */home/skazka*, являясь суперпользователем.

Команды, необходимые для конфигурирования сети: *ping* – проверка работоспособности; *ifconfig* – для проводной сети; *iwconfig* – для беспроводной.

Репозитории. Все программы в дистрибутивах *Linux* - это отдельные проекты, которые развиваются сами по себе, то есть существуют отдельные пакеты с программным обеспечением, есть зависимости. У каждого дистрибутива есть свои разработчики (майнтейнеры). Эти люди занимаются тестированием различных пакетов программ на их нормальное

функционирование, взаимную совместимость, а также часто добавляют собственные усовершенствования или не успевшие войти в официальную сборку и, в конечном итоге, отвечающие за включение пакета в дистрибутив патчи, т.е. разработчики берут программы из открытых исходных кодов и начинают подгонять их друг к другу, упаковывая в пакеты и соблюдая все зависимости, тестируя и удаляя ошибки из этих самых программ. Результат работы - подогнанные друг к другу программы, библиотеки и оригинальные обои, упакованные в пакеты со всеми зависимостями – это и есть репозиторий дистрибутива (хранилище), откуда программы и устанавливаются на компьютер и ведется отслеживание новых версий и внутренних совместимостей пакетов для централизованного управления обновлением пакетов.

Часто компоненты, используемые различными программами, выделяют в отдельные пакеты и помечают, что для работы ПО, предоставленного пакетом А, необходимо установить пакет Б, то есть пакет А зависит от пакета Б или что между пакетами А и Б существует зависимость (обычно в роли зависимостей выступают какие-либо библиотеки, без которых программа не будет запускаться, поскольку использует функции этой библиотеки). Отслеживанием зависимостей между такими пакетами занимается менеджер пакетов, который ведёт базу данных установленных приложений и их версий, и всегда знает, какие файлы куда установлены, чтобы можно было поставить новые программы, удалить старые или обновить всю систему целиком без переустановки и удаления оставшихся файлов.

Какой бы *Linux* не была устойчивой, стабильной и устойчивой, всё же пользователь должен придерживаться определенной осторожности:

1) Не устанавливайте программы в обход менеджера пакетов простой компиляцией. Работать они будут, но пакетный менеджер ничего о них не будет знать, из-за чего при обновлении системы или программ можно получить много проблем. Устанавливайте программы только в виде пакетов.

2) Не подключайте те репозитории, о которых имеете совсем смутное представление. Например, не надо подключать репозитории со словами *testing, debug* и тому подобными терминами, ибо эти репозитории в первую очередь предназначены для самих разработчиков дистрибутивов и далеко не всегда стабильны.

3) Не подключайте подряд все доступные репозитории. Подключайте самые необходимые (один-два).

Для работы по установке пакетов есть свои команды в каждом из дистрибутивов – лучше всего их посмотреть на форумах, например, для систем на *RPM*-пакетах команда *rpm –qa* – показывает список установленных пакетов *RPM* в ОС; для систем на *DEB*-пакетах команда *dpkg -l |more* – показывает список установленных пакетов *DEB* в системе.

2 Управление учетными записями пользователей

В консоле команды для работы с пользователями и группами пользователей:

– *id skazka* - информацию по пользователю *skazka* (логин, *UID, GID*);

– *finger skazka* - показать информацию о пользователе *skazka*;

– *last* - действия последних зарегистрированных пользователей;

– *who* - показывает имя текущего пользователя и время входа;

– *whoami* - показывает к какой группе относится текущий пользователь;

– *useradd skazka* - добавление нового пользователя *skazka* (или *adduser*);

– *groupadd pritcha* - добавление группы *pritcha*;

– *usermod -a -G pritcha skazka* - добавляет в группу *pritcha* пользователя *skazka* (для *Debian*-подобных ОС);

– *passwd skazka* - задание пароля пользователю *skazka*;

– *userdel skazka* - удаление пользователя *skazka*;

– *groupdel pritcha* - удаление группы *pritcha*;

– *login skazka* - залогиниться под пользователем *skazka*;

– *exit* - завершение сеанса текущего пользователя;

– *su* (или *sudo su*) - войти под администраторской учетной записью;

– *sudo nano* - запустить приложение *nano* от имени администратора (*sudo= substitute user and do)* позволяет пользователю выполнять указанные программы с административными привилегиями без ввода пароля суперпользователя);

– *usermod -L skazka* - блокировать пользователя *skazka* (для *Debian*-подобных ОС);

– *usermod -U skazka* - разблокировать пользователя (для *Debian* ОС).

Все команды, введенные пользователем, сохраняются в истории команд (команда *history*), чтобы вызвать одну из команд и перейти от одной к другой, можно использовать стрелочки на клавиатуре ↑↓. Если необходимо вызвать справку по какой-либо команде, то можно написать «*man* имя_нужной_команды».

– !! - последняя команда;

– *uname -a* - показать версию ядра Linux;

– *lsb_release -a* - информация о версии ОС;

– *clear* - очистка экрана;

– *uptime* - текущее время и работу ОС без перезагрузки и выключения;

– *cal* – показ календаря;

– *date* – текущая дата;

– *exit* - завершить сеанс текущего пользователя;

– *shutdown -h now* или *poweroff* - выход из Linux;

– *halt* - аварийное завершение работы;

– *reboot* - перезагрузка системы;

– *last reboot* - статистика перезагрузок;

– *alias c=”color”* - псевдоним, чтобы одной командой можно было запустить более сложную комбинацию команд;

– *dmesg* – показывает *log*-файл загрузки ОС и нахождения новых устройств;

– *vmstat* 2 - статистика по использованию виртуальной памяти;

– *cat /proc/cpuinfo* - информация о модели процессора;

– *cat /proc/meminfo* - информация о занимаемой памяти;

– *grep SwapTotal /proc/meminfo* - размер раздела, выделенного под *swap*;

– *dmidecode* - информация о версии *BIOS* компьютера;

– *wall* и write - отправляет на терминалы других пользователей сообщения. *Write* позволяет общаться с другими пользователями путем копирования строк из Вашего терминала в их в режиме онлайн. Алгоритм действий: узнаем кто в данный момент находится в системе и к какому терминалу подключен с помощью команды *who* (рисунок 4), далее отправляем сообщение пользователю *test*, например: *write test pts/2*. Когда мы нажмем *Enter*, наше сообщение будет отправлено в его терминал, *Ctrl+D* чтобы прервать *write*.

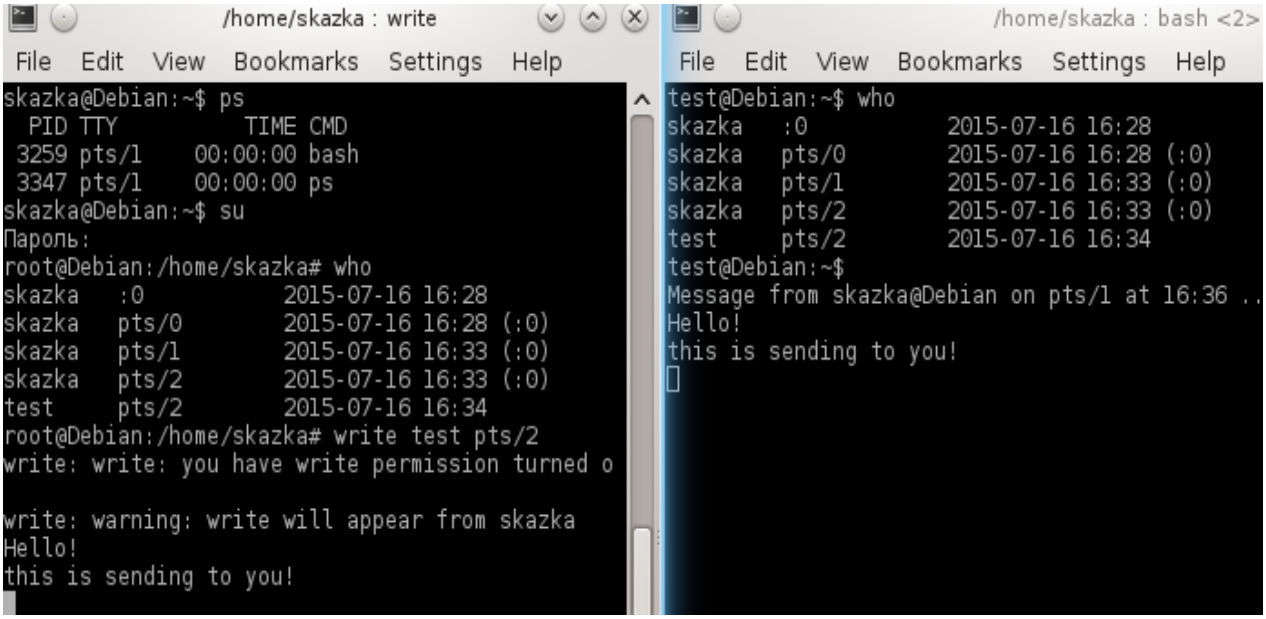

Рисунок 4 – Работа команды *write*

Для отправки широковещательного сообщения всем подключенным пользователям, используется команда *wall* (*wall = write to all*), формат как у команды *write*, сообщение будет отправлено после того как нажимаете *Ctrl+D* (рисунок 5).

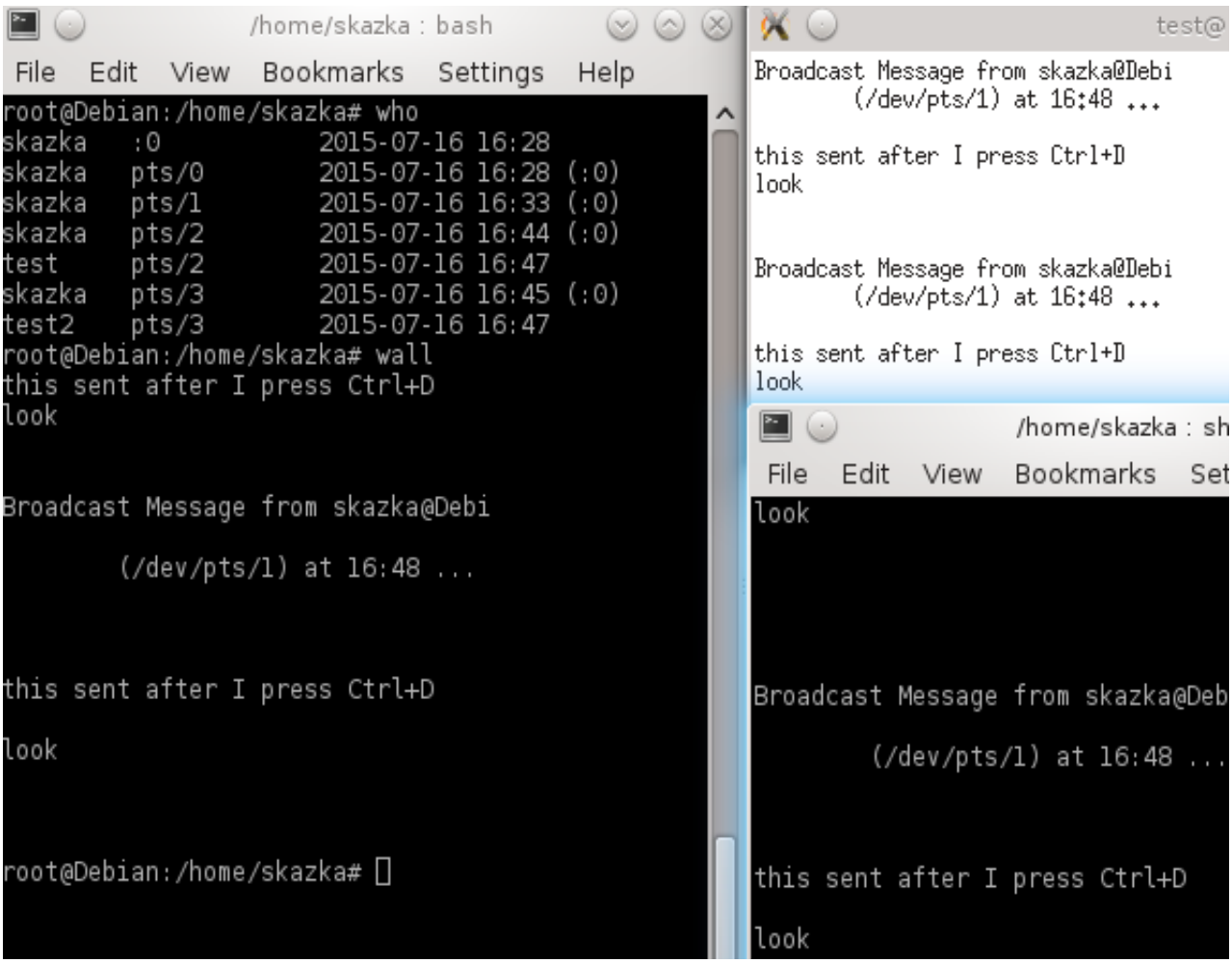

Рисунок 5 – Работа команды *wall*

3 Работа с папками и файлами

1. Для работы в *Linux* с файлами и директориями есть команды:

– *pwd* - выводит текущий путь;

– *ls* - выводит список файлов и каталогов по порядку (*ls -l* – наиболее полный вывод с атрибутами содержимого папки);

– *cd* - переход в домашнюю директорию;

– *cd /home* - переход в директорию /*home*;

– *cd ..* - переход на уровень выше;

– *cd* - - возврат в предыдущую директорию;

– *mkdir* 0 - создание директории с именем 0;

– *rmdir* 0 - удаление директории с именем 0;

– *touch* 2 – создание пустого файла;

– *cp* 1 2 – копирование файла 1 в 2;

– *nano* 1 (или *gedit* 1) – открытие файла 1 через текстового редактора;

– *cat* 1 - показать содержимое файла 1;

– *tac* 1 - показать содержимое файла 1 строками в обратном порядке;

– *echo «*Привет» – выводит на экран сообщение «Привет»;

– *echo* «Привет» > 1 – замена содержимого файла 1 строкой «Привет»;

– *echo* «Привет» >> 1 – дозапись в файл 1 строки «Привет»;

– *echo* «Привет» | *tee -a* 1 - добавление к концу файла 1 «Привет»;

– *mv* 1 2 – переименование/перемещение 1 в 2 (и для файла и для папки);

– *head* 1 – выводит начало файла (по умолчанию первые 10 строк);

– *tail* 1 - выводит конец файла (по умолчанию последние 10 строк);

– *sort* 1 – сортирует строки в файле 1;

– *find* 1 – поиск файла 1 в текущей папке;

– *grep* «23» 1 – поиск в файле 1 строки, содержащей «23»;

– *ln* 1 22 – создание ссылки 22 на файл 1 (полным копированием);

– *ln –s* 1 33 – создание символьной ссылки 33 на файл 1 (ярлык);

– *locate* 1 - поиск всех файлов по машине, содержащих в названии 1;

– *du -sh* 0 - размер заданной директории 0, подходит и для файлов;

– *wc* 1- подсчет количества символов, букв, байт в файле 1;

– *rm* 1 - удаление файла 1;

– *rm* -r - рекурсивное удаление;

– *more* 1 – постраничный просмотр файла 1;

– *yes abc* – бесконечно печатает *abc;*

– *nl 1* – нумерует строки файла 1.

Выше рассматривалась работа команд *write* и *wall* при передаче сообщений на терминалы других пользователей. При этом можно также передавать и файлы: *cat file.txt | write user pts/1* и *cat file.txt | wall.*

По *ls -l* выдается информация о текущей папке в полном объеме по своим атрибутам, дате и владельцу, его группе и правами. Объекту соответствует 9 битов разрешений, они формируют режим доступа и определяют, какие пользователи и группы имеют права на объект (рисунок 6).

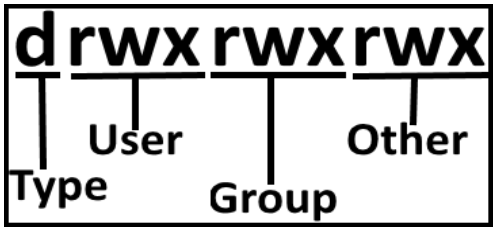

Рисунок 6 – Распределение прав владения объектом

После одного бита определения объекта:

- первые три бита определяют права для владельца файла;

- вторые три бита определяют права для группы, к которой принадлежит владелец данного файла;

- последние определяют права для всех остальных пользователей.

Изменить режим доступа для файлов может только владелец файла или суперпользователь. Код доступа к файлу задается одним из двух режимов:

– цифровой (1- выполнение, 2 - запись, 4 - чтение);

– символьный (x - выполнение, w - запись, r - чтение).

Обозначения для символьного режима: *u – user* (владелец файла), *g – group* (группа владельца), *o – other* (остальные), *a – all* (все). В цифровом виде для смешанных прав соответствующие числа складывают (рисунок 7).

Для работы в *Linux* с правами доступа на объекты есть команды:

– *chmod* 777 1 – изменить прав доступа файла 1, 0777 – разрешение на чтение/запись/исполнение для всех групп – полный доступ;

– *chmod -R* 777 0 - рекурсивное изменение прав доступа к директории 0, 777 – разрешение на чтение/запись/исполнение для всех групп, все вложенные директории и файлы будут иметь права 777;

– *chown skazka:test 1* - изменение владельца и группы владельца только для файла 1;

– *chgrp test* 1 - изменение группы владельца файла 1.

– Для передачи файла пользователю *test*: *cat file.txt | write test pts/1*. Для отправки файла всем пользователям: cat *file*.txt | wall.

– *history | tail -30* - показать последние 30 набранных команд.

Команды в терминале выводят информацию, можно использовать метасимволы для формирования сложных команд:

а) > - запись в файл. Например, существует файл *goa* с некоторым содержимым, набираем «*date>goa*» - содержимое файла стирается и в него записывается текущая дата;

б) >> - дозапись в конец существующего файла. Например, существует файл *goa* с некоторым содержимым, набираем «*date>>goa*» - содержимое файла остается + в конец файла *goa* добавляется строка с текущей датой;

в) | - программный канал - результат одной команды передается другой команде;

г) & - процесс выполняется в фоновом режиме;

д) ? – любой 1 символ;

е) * - любое количество любых символов;

ж) ; - перечисление, команды выполняются друг за другом;

и) && - при объединении команд: последующая команда выполняется только при нормальном завершении предыдущей;

к) || - при объединении команд: последующая команда выполняется только, если не выполнилась предыдущая команда;

л) () - группирование команд в скобки, как в арифметических расчетах;

м) { } - группирование команд с объединенным выводом;

н) [ ] - указание диапазона данных или перечисление данных (без запятых).

```
SIB:/tmp/PPAP# ls -l
итого 0
-rw-r--r-- 1 root root 0 Ноя 16 23:46 a.txt
SIB:/tmp/PPAP# chmod 102 a.txt
SIB:/tmp/PPAP# ls -l
итого 0
---x----w- 1 root root 0 Ноя 16 23:46 a.txt
SIB:/tmp/PPAP# (chmod 450 a.txt)&(ls -l)
[1] 5296
итого 0
-r--r-x--- 1 root root 0 Ноя 16 23:46 a.txt
[1]+  Done                    ( chmod 450 a.txt )
SIB:/tmp/PPAP# (chmod u+x a.txt)&(ls -l)
[1] 5319
итого 0
-r-xr-x--- 1 root root 0 Ноя 16 23:46 a.txt
[1]+  Done                    ( chmod u+x a.txt )
SIB:/tmp/PPAP# (chmod g-x a.txt)&(ls -l)
[1] 5321
итого 0
-r-xr----- 1 root root 0 Ноя 16 23:46 a.txt
[1]+  Done                    ( chmod g-x a.txt )
SIB:/tmp/PPAP# (chmod a+rw a.txt)&(ls -l)
[1] 5339
итого 0
-rwxrw-rw- 1 root root 0 Ноя 16 23:46 a.txt
[1]+  Done                    ( chmod a+rw a.txt )
SIB:/tmp/PPAP# (chmod o-rw a.txt)&(ls -l)
[1] 5346
итого 0
-rwxrw---- 1 root root 0 Ноя 16 23:46 a.txt
[1]+  Done                    ( chmod o-rw a.txt )
SIB:/tmp/PPAP# chown skazka:root a.txt
SIB:/tmp/PPAP# ls -l
итого 0
-rwxrw---- 1 skazka root 0 Ноя 16 23:46 a.txt
SIB:/tmp/PPAP# chown root a.txt
SIB:/tmp/PPAP# ls -l
итого 0
-rwxrw---- 1 root root 0 Ноя 16 23:46 a.txt
SIB:/tmp/PPAP# chown :skazka a.txt
SIB:/tmp/PPAP# ls -l
итого 0
-rwxrw---- 1 root skazka 0 Ноя 16 23:46 a.txt
```

Рисунок 7 – Примеры смены прав доступа на объект

Umask (от англ. *user file creation mode mask* - маска режима создания пользовательских файлов) - функция, изменяющая права доступа, которые присваиваются новым файлам и директориям по умолчанию. Права доступа файлов, созданных при конкретном значении *umask*, вычисляются при помощи следующих побитовых операций (*umask* обычно устанавливается в восьмеричной системе счисления): побитовое И между унарным дополнением аргумента (используя побитовое НЕ) и режимом полного доступа. Режим полного доступа для директорий - 0777, для файлов - 0666. *Umask* влияет на все дочерние процессы, исполняемые в текущей оболочке. Другими словами, *umask* используется на установкуобъекты начальных прав доступа к файлу или папки при их будущем создании, на уже созданные это не распространяется. Напомню: 1=x, 2=w, 4=r.

Umask-формулы расчета:

- для файла: 0776 – *umask* = устанавливаемые права на файл;
- для папки: 0777 – *umask* = устанавливаемые права на папку.

Разберем на примерах:

1) Для файлов.

Первоначальное значение в системе по умолчанию *umask* это 0022, что соответствует правам 644 (рисунок 8), *umask* 0666 означает что соответствует правам 000, то есть нет никаких прав; *umask* 0174 означает что соответствует правам 602;

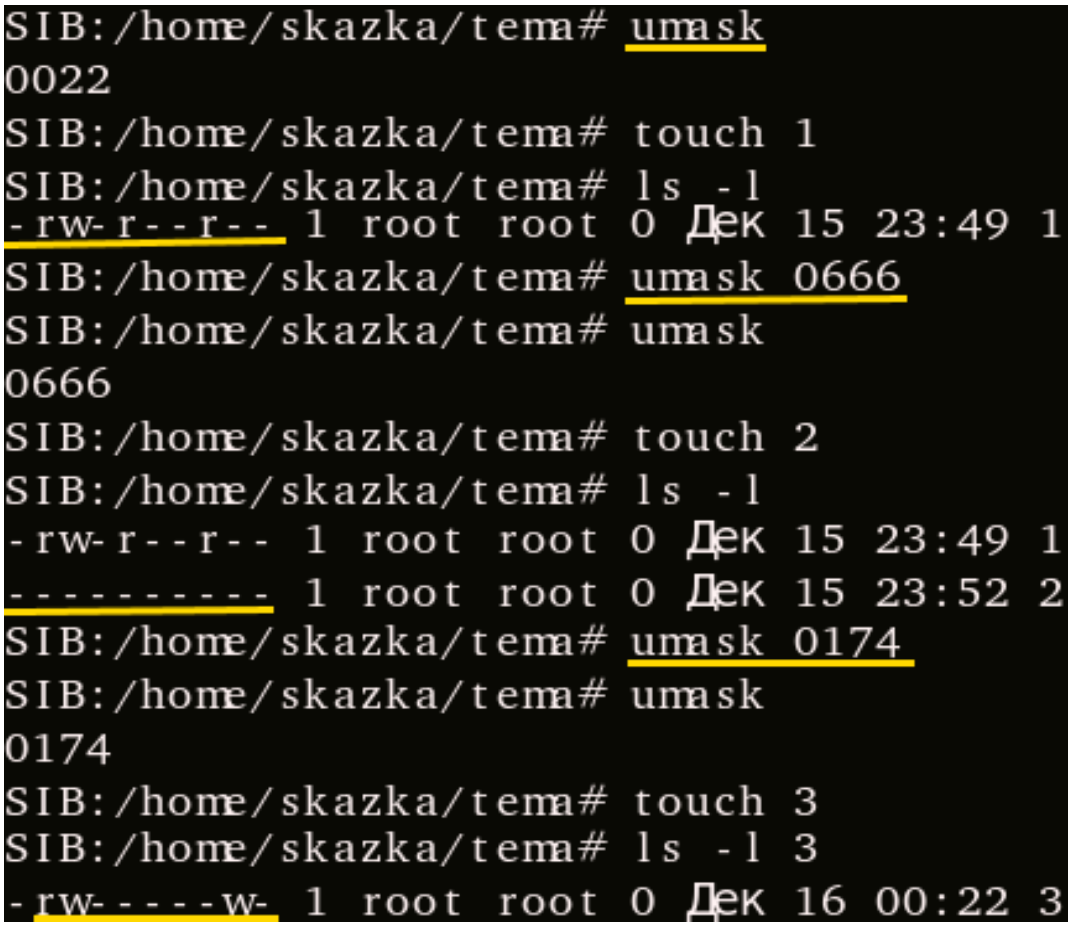

Рисунок 8 – Распределение прав по *umask* на файлы

2) Для директорий.

Umask 000 - это 0777, что соответствует наличию всех прав (рисунок 9), *umask* 0777 означает, что соответствует правам 000, то есть нет никаких прав; *umask* 0123 означает, что соответствует правам 654.

```
SIB:/home/skazka/tema# umask 0000
SIB:/home/skazka/tema# mkdir 0123
SIB:/home/skazka/tema# umask
0000
SIB:/home/skazka/tema# ls -l
drwxrwxrwx 2 root root 4096 Дек 16 00:07 0123
SIB:/home/skazka/tema# umask 0777
SIB:/home/skazka/tema# umask
0777
SIB:/home/skazka/tema# mkdir 2
SIB:/home/skazka/tema# ls -l
drwxrwxrwx 2 root root 4096 Дек 16 00:07 0123
d--------- 2 root root 4096 Дек 16 00:08 2
SIB:/home/skazka/tema# umask 0123
SIB:/home/skazka/tema# mkdir 3
SIB:/home/skazka/tema# ls -l
drwxrwxrwx 2 root root 4096 Дек 16 00:07 0123
d--------- 2 root root 4096 Дек 16 00:08 2
drw-r-xr-- 2 root root 4096 Дек 16 00:08 3
```

Рисунок 9 – Распределение прав по *umask* на файлы

4 Монтирование от физических носителей до виртуальных жестких дисков

В *Linux* при монтировании и форматировании существуют команды:

– *fdisk -l* - информация о всех подключенных жестких и сменных дисках;

– *blkid* - выводит *UUID* всех доступных накопителей в системе;

– *mount | column –t* - информация о примонтированных устройствах;

– *cat /proc/partitions* - только примонтированные разделы жесткого диска;

– *df* - показывает свободное место на разделах;

– *mount /dev/sda1 /mnt* - монтирует раздел */dev/sda1* в папку /*mnt*;

– *mkfs* – создание файловой системы на устройстве, на разделе диска;

– *umount /mnt* - отмонтирует раздел от точки монтирования /*mnt.*

Чтобы подключить носитель к файловой системе и получить доступ к нему, необходимо использовать команду *mount*. Эта команда берет из существующего файлового дерева каталог (называется точкой монтирования) и делает его корневым каталогом, присоединяемым к файловой системе. Таким образом, в системе *Linux* вся файловая система представлена как единое дерево каталогов.

Если ядро *Linux* опознало устройство-носитель данных, то оно должно предоставить какой-то внешний интерфейс пользователю для работы с устройством. Этим интерфейсом является создание файлов-устройств в каталоге */dev*, например:

– устройствам, подключённым к *IDE*, будут соответствовать файлы-устройства */dev/hda*, */dev/hdb* и т.д.;

– устройствам типа *SCSI*, а также близкие им *SATA*-устройства и *USB*-флеш, будут иметь файлы-устройства */dev/sda, /dev/sdb* и тому подобное.

Если на диске есть разделы, то цифра в имени файла-устройства будет соответствовать номеру раздела, например, если на *USB*-флеш есть два раздела, то первый будет называться */dev/sda1*, а второй */dev/sda2*.

Монтирование разделов - это объяснение системе, как добраться до ваших данных и сделать их доступными для использования. Системе нужно объяснить:

– какая файловая система на разделе;

– какое файл-устройство вам нужно;

– куда его подключить для просмотра - это точка монтирования.

Каталог, в котором будет просматриваться содержимое разделов флеш-накопителя, называется точкой монтирования (*mount point*). Поэтому нужно объяснить системе, что вы хотите смонтировать, куда и что за файловая система на этом разделе, например, так: *mount -t vfat /dev/hda3 /mnt/storage.*

Если нужно часто монтировать одни и те же разделы, занесите их в */etc/fstab* - это общесистемный конфигурационный файл, в котором указаны все необходимые разделы для монтирования. Редактировать этот файл может только *root*.

Файл */etc/fstab* состоит из колонок, разделителем является пустое пространство, оставленное табуляцией либо пробелами (в любых количествах). Вот пример содержимого файла */etc/fstab*:

```
# /etc/fstab: static file system information.
/dev/hda2     / reiserfs                notail,noatime                0 1
/dev/hdb      /mnt/cdrom  iso9660 ro,user,noauto                      0 0
/dev/sda1     /mnt/flash     vfat    iocharset=koi8-r,codepage=866,rw,user,auto    0 0
```

Формат колонок таков:

– монтируемое устройство (раздел, который надо примонтировать);

– точка монтирования (в какую папку монтировать);

– тип монтируемой файловой системы (*FAT32*, *NTFS* и т.п);

– опции монтирования (кодировка языка, необходимость монтирования при старте);

– *dump* (обычно 0);

– *pass* (обычно 0);

Необходимо вставлять разделители после каждой колонки при редактировании */etc/fstab*.

Ниже приводятся уточнения для каждого случая файловой системы, но идея монтирования разделов в Linux одна и та же, например, *mount -t vfat /dev/hda1 /home/username/tempdir* – примонтировать тип *FAT32* что и куда.

После всех операций раздел следует размонтировать, чтобы данные на него записались из памяти - точно так же, как при работе с *USB*-флеш в *Windows*. Для этого следует дать команду: *umount /mnt/XX,* где */mnt/XX* - точка монтирования. Если ни одно приложение не работает с разделом, то раздел размонтируется, отключится и данные на него полностью запишутся. После этого устройство можно безопасно извлечь.

Чтобы посмотреть, что можно смонтировать и увидеть информацию о разделах, можно использовать команду *fdisk –l.* На рисунке 10 показан пример работы этой команды.

```
root@Debian:/home/skazka# fdisk -l

Disk /dev/sda: 22.3 GB, 22325526528 bytes
255 heads, 63 sectors/track, 2714 cylinders, total 43604544 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00086149

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048    41732095    20865024   83  Linux
/dev/sda2        41734142    43603967      934913    5  Extended
/dev/sda5        41734144    43603967      934912   82  Linux swap / Solaris

Disk /dev/sdb: 2142 MB, 2142296064 bytes
255 heads, 63 sectors/track, 260 cylinders, total 4184172 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
```

Рисунок 10 – Пример вывода данных при вызове команды *fdisk –l*

Прежде, чем монтировать разделы, нужно создать точки монтирования. Точки монтирования следует создавать в каталогах /*mnt* или /*media*. То есть в этих папках создаем директорию *mkdir /mnt/win* и даём каталогу права доступа для обычных пользователей: *chmod 777 /mnt/win* - это позволит считывать и записывать данные не только суперпользователю, но и обычному

пользователю. Свою точку монтирования в директории /*mnt* нужно создать для каждого раздела.

Не бойтесь, если вы не знаете точно, как теперь называются ваши диски с *Windows C:* или *D:* и монтируйте разделы с указанием типа файловой системы *FAT* или *NTFS* - если ошибётесь, раздел не примонтируется и будет выскакивать ошибка вроде подобной:

mount: wrong fs type, bad option, bad superblock on /dev/sda2,
missing codepage or other error
In some cases useful info is found in syslog – try
dmesg | tail or so

и это означает, что примонтировать файловую систему не получилось.

Ниже приводятся описания нескольких типичных случаев монтирования разделов и файлов.

а) монтирование раздела с файловой системой *FAT32*.

Однократное монтирование:

$ sudo mount -t vfat /dev/XX /mnt/YY -o iocharset=utf8,codepage=866,rw

или

 # mount -t vfat /dev/XX /mnt/YY -o iocharset=utf8,codepage=866

например:

$ sudo mount -t vfat /dev/sda1 /mnt/flash -o iocharset=utf8,codepage= 866,rw - то есть монтируем раздел */dev/sda1* типа *vfat* в точку монтирования */mnt/flash*.

Постоянное монтирование (если нужно постоянно обращаться к этим разделам, в /*etc/fstab* нужно добавить строку):

/dev/XX /mnt/YY vfat iocharset=utf8,codepage=866,user,rw 0 0

например: */dev/sda1 /mnt/flash vfat iocharset=utf8,codepage=866,user,rw 0 0* - то есть монтируем раздел */dev/sda1* типа *vfat* в точку монтирования */mnt/flash.* После этого раздел *FAT32* в *Linux* будет доступен. Отмонтирование по команде *umount /mnt/flash*;

б) монтирование раздела с файловой системой *NTFS*.

Монтировать и записывать на *NTFS* возможно благодаря усилиям разработчиков, создавших драйвер *NTFS-3g*. Этот драйвер можно найти и установить в репозитории - пакет ntfs-3g.

Однократное монтирование:

mount -t ntfs-3g /dev/XX /mnt/YY -o umask=0,nls=utf8

например: монтирование раздела *Windows* с *NTFS - $ sudo mount -t ntfs-3g /dev/sda1 /mnt/flash -o umask=0,nls=utf8* - то есть монтируем раздел /*dev/sda1* типа *ntfs-3g* в точку монтирования */mnt/flash.*

Постоянное монтирование (если нужно постоянно обращаться к этим разделам, в */etc/fstab* нужно добавить строку):

/dev/XX /mnt/YY ntfs-3g umask=0,nls=utf8,user,auto,rw 0 0

например: */dev/sda1 /mnt/winds ntfs-3g umask=0,nls=utf8,exec,user,auto,rw 0 0.*

Иногда в /*etc/fstab* это уже есть с настройками вида: */dev/sda1 /mnt/winds ntfs-3g umask=0,nls=ru-RU.UTF-8 0 0* - при такой записи всё нормально, кроме одного - файлы и каталоги с русскими именами не отображались, будто их не

было вовсе. Чтобы это исправить, можно попробовать изменить на: */dev/sda1 /mnt/windows ntfs-3g exec,user,auto,rw 0 0* - то есть монтируем раздел */dev/sda1* типа *ntfs-3g* в точку монтирования /*mnt/winds*. После этого раздел *NTFS* в *Linux* будет доступен. Отмонтирование: *umount /mnt/winds*;

в) монтирование *CD/DVD* -диска.

Однократное монтирование: *mount -t iso9660 /dev/XX /mnt/YY,* например: монтирование *DVD*-диска: *mount -t iso9660 /dev/cdrom /mnt/dvd* - то есть монтируем раздел /*dev/cdrom* типа *iso9660* в точку монтирования */mnt/dvd*.

Постоянное монтирование (если нужно постоянно обращаться к этим разделам, в */etc/fstab* нужно добавить строку): */dev/XX /mnt/YY iso9660 user,ro 0 0*. Например, оптический диск в /*etc/fstab* выглядит так*: /dev/cdrom /mnt/cdrom iso9660 user,ro 0 0* - то есть монтируем раздел /*dev/cdrom* типа *iso9660* в точку монтирования /*mnt/cdrom*. После этого оптический диск в Linux будет доступен. Отмонтирование по команде *umount /mnt/cdrom*;

г) монтирование *ISO*-файла или другого файла.

Вы скачали *ISO*-файл и хотите просмотреть его содержимое. Для этого в *Windows* есть, например, программа *Alcohol*. В *Linux* такой программы нет и монтирование *ISO*-образа ничем для системы не отличается от монтирования *CD/DVD*-диска, с той разницей что нужно только передать один параметр *-o loop* чтобы сообщить системе, что монтировать надо на петлевое устройство (*loopback*): *mount -t iso9660 file.iso /mnt/YY -o loop,* например, *mount -t iso9660 file.iso /mnt/dvd -o loop* - то есть монтируем файл *file.iso* типа *iso9660* в точку монтирования /*mnt/dvd* на петлевое устройство *loop*. Отмонтирование по команде *umount /mnt/dvd*.

Точно так же можно монтировать любой *CD/DVD* образ, а также файл-образы сжатых\зашифрованных\экзотических файловых систем типа SquashFS;

д) монтирование раздела с файловой системой *ext2/ext3/Reiserfs/xfs/jfs*.

Монтирование нормальных файловых систем проблем не создаёт. Приведу пример с *ext3*, остальные файловые системы монтируются точно так же, только вместо *ext3* подставляется буквально: *ext2, reiserfs, xfs, jfs* в зависимости от нужной файловой системы.

Однократное монтирование: *mount -t ext3 /dev/XX /mnt/YY,* например: монтирование раздела с данными *mount -t ext3 /dev/sda1 /mnt/data* - то есть монтируем раздел /*dev/sda1* типа *ext3* в точку монтирования /*mnt/data.*

Постоянное монтирование (если нужно постоянно обращаться к этим разделам, в */etc/fstab* нужно добавить строку): */dev/XX /mnt/YY ext3 user,auto,rw 0 0*. Например, флеш в /*etc/fstab* так: */dev/sda1 /mnt/data ext3 user,auto,rw 0 0* - то есть монтируем раздел /*dev/sda1* типа *ext3* в точку монтирования /*mnt/data.* После этого раздел *ext3* в *Linux* доступен.

Для того, чтобы узнать тип файловой системы на диске, надо воспользоваться утилитой *file*. Так как в *UNIX/Linux/*BSD* исповедуется принцип «всё есть файл», то раздел с файловой системой - это тоже файл, поэтому команда *file* выдаст информацию о файловой системе дисковых

разделов - надо только указать программе воспринимать их как специальные файлы. Используем опцию *-s (–special-files)* для выдачи информации о типе файловой системы блочного устройства. Например: *file -s /dev/sda1* и получаем в ответ: */dev/sda1: ReiserFS V3.6 block size 4096 (mounted or unclean) num blocks 17920496 r5 hash* - то есть на разделе */dev/sda1* имеем файловую *ReiserFS V3.6.* Или, например: *file -s /dev/sda1* выдаст в ответ: */dev/sda1: x86 boot sector, code offset 0x58, OEM-ID "MSWIN4.1", sectors/cluster 64, reserved sectors 126, Media descriptor 0xf8, heads 255, hidden sectors 63, sectors 284237982 (volumes > 32 MB) , FAT (32 bit), sectors/FAT 34693, reserved3 0x800000, serial number 0x287d1dfa, unlabeled* - это поможет в определении того, какая файловая система (и намекнёт на то, какие данные там лежат) на данном разделе.

Вы пытаетесь размонтировать раздел с данными и возникает сообщение с ошибкой - устройство занято. Например, при выполнении команды *umount /mnt/windows* возникает сообщение вида: «*umount: /mnt/windows: device is busy».* Надо посмотреть, какой процесс держит раздел или файл и не даёт отмонтировать его. Дальше либо закрыть это приложение, либо убить его с помощью команды *kill*. Например, дается команда: *lsof | grep /mnt/windows.*

После этого вы получите ответ вида:

lsof: WARNING: can't stat() reiserfs file system /dev/.static/dev

Output information may be incomplete.

mc 14134 beast cwd DIR 8,1 4096 1 /mnt/windows,

значит раздел держит *MC (Midnight Commander*), который вы скорее всего открыли на другом рабочем столе и просто забыли о нём. Найдите это приложение, закройте его и отмонтируйте раздел снова - теперь проблема должна быть решена.

Форматирование - процесс создания файловой системы на устройстве. Алгоритм форматирования устройства: вставляем устройство; открываем терминал и смотрим имя устройства, которое хотим отформатировать командой *fdisk –l* или *df*. Если сработало автомонтирование, то отмонтируем диск с помощью *umount*. Запускаем форматирование: *mkfs.vfat -n 'Nazvanie' -I /dev/устройство*, при этом форматирует в *FAT32*, доступны и другие форматы: *mkfs.bfs, mkfs.ext2, mkfs.ext3, mkfs.ext4, mkfs.minix, mkfs.msdos, mkfs.vfat, mkfs.xfs, mkfs.xiafs* и т.п.; параметр *–n* задает имя (метку) для файловой системы, по умолчанию метка не создается, а имя раздела "*Nazvanie*"; *-I* - жесткие диски разбиваются таким образом по умолчанию – запрет создавать файловую систему на всем устройстве. Проверяем результат, запустив *df* после форматирования;

Для примера монтирования подключу виртуальный жесткий диск.

Изначально в ОС виртуальной машины имеется разбиение на разделы, представленное на рисунке 11. Необходимо остановить виртуальную машину, и в выключенном состоянии перейти в меню ее настроек, в пункте «Носители» добавить «Жесткий диск» (рисунок 12). Далее в разделе добавления виртуального жесткого диска к контроллеру *SATA* «Создать новый

диск», указать тип (по умолчанию *VDI*), указать имя и размер. После нажатия кнопки «*OK*» появится соответствующий диск (рисунок 13). Загружаем систему и проверяем наличие новых устройств и разделов, появился */dev/sdb* (рисунок 14). Создаем на нем файловую структуру (рисунки 15, 16).

```
root@debian:/home/skazka# fdisk -l

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders, total 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000138b1

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048      684031      340992   83  Linux
/dev/sda2          686078    16775167     8044545    5  Extended
/dev/sda5          686080     6537215     2925568   83  Linux
/dev/sda6         6539264     9408511     1434624   83  Linux
/dev/sda7         9410560    10356735      473088   82  Linux swap / Solaris
/dev/sda8        10358784    10852351      246784   83  Linux
/dev/sda9        10854400    16775167     2960384   83  Linux
root@debian:/home/skazka#
```

Рисунок 11 – Первоначальное разбиение в ОС

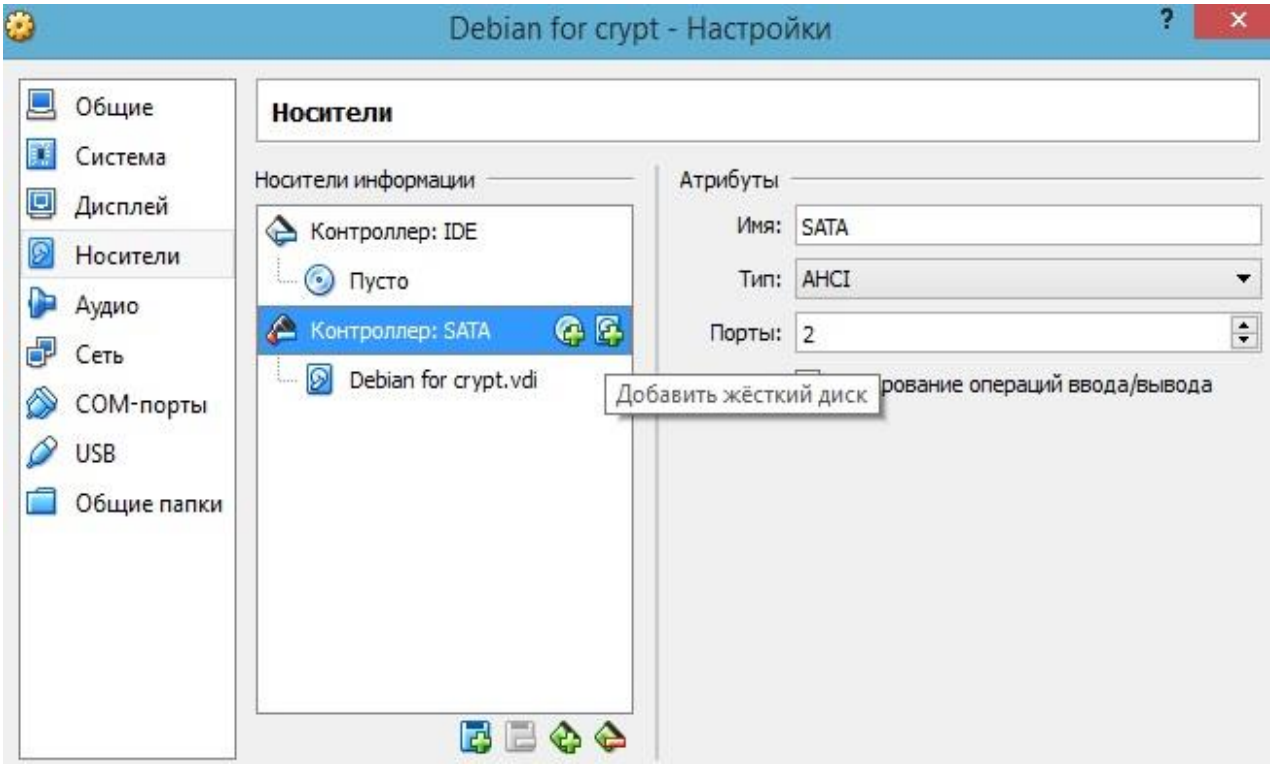

Рисунок 12 – Настройка и добавление диска

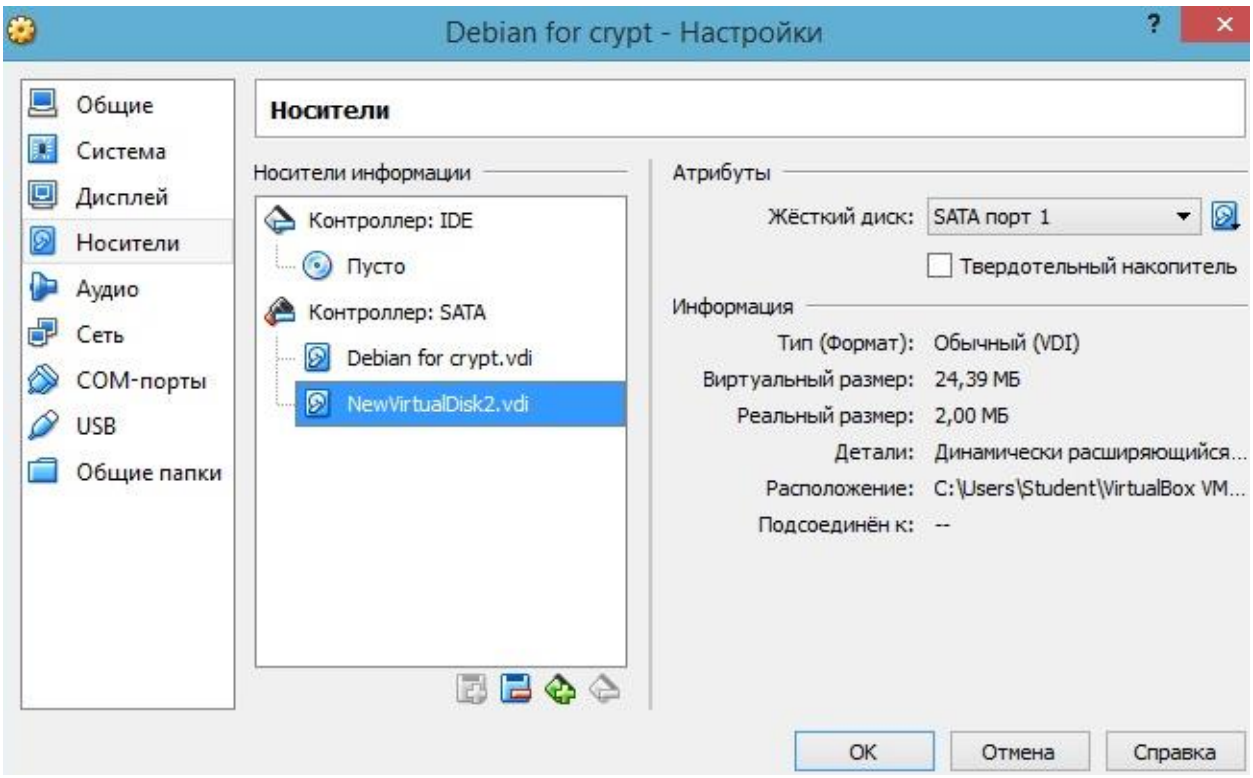

Рисунок 13 – Установка параметров виртуального жесткого диска

```
root@debian:/home/skazka# fdisk -l

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders, total 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000138b1

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048      684031      340992   83  Linux
/dev/sda2          686078    16775167     8044545    5  Extended
/dev/sda5          686080     6537215     2925568   83  Linux
/dev/sda6         6539264     9408511     1434624   83  Linux
/dev/sda7         9410560    10356735      473088   82  Linux swap / Solaris
/dev/sda8        10358784    10852351      246784   83  Linux
/dev/sda9        10854400    16775167     2960384   83  Linux

Disk /dev/sdb: 25 MB, 25570816 bytes
255 heads, 63 sectors/track, 3 cylinders, total 49943 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

Disk /dev/sdb doesn't contain a valid partition table
root@debian:/home/skazka#
```

Рисунок 14 – Новое устройство в *Linux*

```
root@debian:/home/skazka# fdisk /dev/sdb
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0xc428066e.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-49942, default 2048):
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-49942, default 49942):
Using default value 49942

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
root@debian:/home/skazka#
```

Рисунок 15 – Создание структуры на разделе

```
root@debian:/home/skazka# fdisk -l

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders, total 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000138b1

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048      684031      340992   83  Linux
/dev/sda2          686078    16775167     8044545    5  Extended
/dev/sda5          686080     6537215     2925568   83  Linux
/dev/sda6         6539264     9408511     1434624   83  Linux
/dev/sda7         9410560    10356735      473088   82  Linux swap / Solaris
/dev/sda8        10358784    10852351      246784   83  Linux
/dev/sda9        10854400    16775167     2960384   83  Linux

Disk /dev/sdb: 25 MB, 25570816 bytes
28 heads, 47 sectors/track, 37 cylinders, total 49943 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xc428066e

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            2048       49942       23947+  83  Linux
root@debian:/home/skazka#
```

Рисунок 16 – Смонтированный раздел

5 Квотирование

Квоты - это опциональная возможность ОС, которая позволяет ограничивать объем дискового пространства и/или количество файлов для конкретного пользователя или членов определенной группы в рамках одной файловой системы. Чаще всего эта возможность используется в системах разделения времени, когда желательно ограничить количество ресурсов, которые может использовать один пользователь или группа пользователей. Это позволит не допустить ситуации, когда один пользователь или группа пользователей заполняют всё доступное дисковое пространство.

Квотирование определяется, как для каждого пользователя, так и для каждой файловой системы. Если имеется несколько разделов, то квоты нужно определить для каждого раздела, в котором пользователь намеревается создавать файлы. Например, если вы входите в группу, которая превысила наложенное на нее ограничение, то вы не сможете использовать дисковое пространство, даже если вы все еще можете использовать его как пользователь. */etc/security/limits.conf* - файл для задания лимитов.

В *Linux* нет удобных утилит, позволяющих настраивать и управлять дисковыми квотами. Вам придётся выполнить ряд действий, некоторые из которых достаточно критичны в отношении системы (например, редактирование */etc/fstab*). Отнеситесь к этому очень внимательно. Делайте резервные копии всех файлов, которые будете редактировать, а изменения вносите очень внимательно, поскольку вы попросту можете оказаться у системы, неспособной загрузиться.

Шаги задания квот дискового пространства:

1) Разрешите определять квоты для каждой файловой системы = изменить файл */etc/fstab*.
2) Перемонтировать файловую систему.
3) Создать файлы базы данных квот и сгенерировать таблицу использования диска.
4) Назначить политики квот = сами дисковые ограничения.

Так как квота - это административная утилита для мониторинга и ограничения использования дискового пространства пользователями и группами на каждой файловой системе, то существует два возможных способа ограничений использования дисков. Первый - это число *inode*-ов (число файлов), которым может владеть пользователь или группа. Второй - число дисковых блоков (суммарное пространство в килобайтах), которое может выделяться в использование пользователю или группе.

При помощи квот системный администратор способствует возникновению ситуации, когда пользователь не расходует попусту неограниченный объем дискового пространства. Эта программа оперирует отдельно каждым пользователем и каждой файловой системой, поэтому для каждой файловой системы нужно определять квоты отдельно.

Алгоритм задания квот.

1) Создание ядра с поддержкой квот.

Если ядро *Linux* версии 2.2.14, то надо удостовериться, что вы ответили положительно ответили на вопрос *Filesystems*:

Quota support (CONFIG_QUOTA) [N/y/?] Y

2) Модификация файла /etc/fstab.

Файл */etc/fstab* содержит информацию обо всех файловых системах, инсталлированных на Linux-сервере. Квоты должны быть включены в нем, чтобы их можно было использовать. Так как квоты должны быть определены для каждой файловой системы независимо, и каждая файловая система описывается в файле */etc/fstab* в отдельной строке, то квота должна быть установлена для каждой строки, где вы хотите включить их поддержку. Используя программу «квота», в зависимости от нужд, вы можете включить квоты только для групп, пользователей или и тех, и других одновременно.

Для всех нижеприведенных примеров, я использую каталог */home*, размещенный на разделе */dev/sda6*.

Вариант 1.

Для включения квот для пользователей на определенной файловой системе отредактируйте */etc/fstab* файл (*nano /etc/fstab*) и добавьте опцию *usrquota* в четвертое поле после слова *defaults* или любой другой опции.

Например:

/dev/sda6 /home ext2 defaults 1 2 (как пример: слово *defaults*)

/dev/sda6 /home ext2 nosuid,nodev 1 2 (как пример: любая другая опция)

должен читаться:

/dev/sda6 /home ext2 defaults,usrquota 1 2

/dev/sda6 /home ext2 nosuid,nodev,usrquota 1 2

Вариант 2.

Для включения квот для групп на определенной файловой системы, отредактируйте /*etc/fstab* файл (*nano /etc/fstab*) и добавьте опцию *grpquota* в четвертое поле после слова *defaults* или любой другой опции. Например:

/dev/sda6 /home ext2 defaults 1 2 (как пример: слово *defaults*)

/dev/sda6 /home ext2 nosuid,nodev 1 2 (как пример: любая другая опция)

должен читаться:

/dev/sda6 /home ext2 defaults,grpquota 1 2

/dev/sda6 /home ext2 nosuid,nodev,grpquota 1 2

Вариант 3.

Для включения квот для пользователей и групп на определенной файловой системы, отредактируйте */etc/fstab* файл (*nano /etc/fstab*) и добавьте опции *usrquota, grpquota* в четвертое поле после слова *defaults* или любой другой опции. Например:

/*dev/sda6 /home ext2 defaults 1 2* (как пример: слово *defaults*)

/dev/sda6 /home ext2 nosuid,nodev 1 2 (как пример: любая другая опция)

должен читаться:

/dev/sda6 /home ext2 defaults,usrquota,grpquota 1 2

/dev/sda6 /home ext2 nosuid,nodev,usrquota,grpquota 1 2

3) Создание файлов quota.user и quota.group.

После модификации файла *e/tc/fstab*, чтобы квоты начали действовать, в корневой каталог файловой системы (например, /*home*) помещается файл *quota.user*, если вы хотите использовать пользовательские квоты, или *quota.group*, для групповых квот, или и тот и другой для комбинированных квот. Владельцем обоих файлов является суперпользователь. Объясню пошагово:

Шаг 1.

Для создания файлов *quota.user* и/или *quota.group*, как суперпользователь перейдите в корневой каталог раздела, где вы хотите активизировать квоты (например, /*home*), создайте *quota.user* и/или *quota.group*, для этого выполните следующие команды:

[root@deep /]# touch /home/quota.user
[root@deep /]# touch /home/quota.group
[root@deep /]# chmod 600 /home/quota.user
[root@deep /]# chmod 600 /home/quota.group

Команда *touch* будет создавать новые пустые файлы в каталоге /*home* с именами *quota.user* и *quota.group*. Команда *chmod* будет устанавливать права доступа к этим файлам в чтение-запись только для суперпользователя. При этом оба файла должны принадлежать суперпользователю с правами чтение-запись только для владельца.

Шаг 2.

Надо инициализировать файлы *quota.user* и *quota.group* в корневом каталоге файловой системы, чтобы не получать сообщений об ошибках о квотах во время перезагрузки сервера. Для инициализации файлов используйте следующие команды:

[root@deep /]# edquota -u sib
[root@deep /]# edquota -g sib

Вышеприведенные команды необходимы только для инициализации файлов *quota.user* и/или *quota.group*; команда *edquota (-u)* будет редактировать квоты для пользователя "*sib*" и *(-g*) будет редактировать квоты для группы. Вы должны редактировать существующие в системе *UID/GID*, чтобы инициализация файлов прошла успешно.

Шаг 3.

После того, как вы закончили устанавливать необходимые опции в файле */etc/fstab*, создали и инициализировали файлы *quota.users* и/или *quota.group*, вы должны перезагрузить систему, чтобы внесенные изменения в файлы */etc/fstab, quota.user* и/или *quota.group* вступили в силу ([*root@deep /]# reboot*).

4) Назначение квот для пользователей и групп.

После того, как система перезагрузилась, вы можете назначить квоты пользователям и группам пользователей. Это операция осуществляется при помощи команды *edquota. edquota (8).*

Edquota - это редактор квот, который создает временный файл с текущими дисковыми квотами, используемый суперпользователем для их установки для пользователей и групп пользователей. Нижеприведенный пример покажет, как установить квоты для пользователя и группы пользователей.

а) установка квоты для пользователя.

Предположим, для примера, что у вас есть пользователь с именем *sib*. Следующая команда вызывает редактор *nano*, чтобы изменить и установить квоты для пользователя *sib* на каждый раздел, где включены квоты:

Шаг 1.

Для редактирования и модификации квот для пользователя *sib* используйте следующую команду:

[root@deep /]# edquota -u sib
Quotas for user sib:
/dev/sda6: blocks in use: 6, limits (soft = 0, hard = 0)
inodes in use: 5, limits (soft = 0, hard = 0)

После выполнения этой команды, вы увидите на экране строки, связанные с пользователем *sib. blocks in use*: отображает общее число блоков (в килобайтах) расходуемых пользователем на разделе. *inodes in use*: отображает общее число файлов, которое имеет пользователь на разделе. Эти параметры (*blocks in use, and inodes in use*) контролируются и устанавливаются автоматически системой, и вы не можете установить или изменить их.

Шаг 2.

Для наглядности назначу 5MB квоту для пользователя *sib*, изменив следующие параметры в редакторе *nano*:

Quotas for user sib:
/dev/sda6: blocks in use: 6, limits (soft = 0, hard = 0)
inodes in use: 5, limits (soft = 0, hard = 0)

должна читаться:

Quotas for user sib:
/dev/sda6: blocks in use: 6, limits (soft = 5000, hard = 0)
inodes in use: 5, limits (soft = 0, hard = 0)

soft limit (soft =) определяет максимальное количество дискового пространства, которое пользователь может иметь. А *hard limit (hard =)* определяет абсолютное ограничение использования пользоватлем дискового пространства. Пользователь не может превзойти его. Следует заметить, что *hard limit* работает только когда установлен параметр *grace period*.

Параметр *grace period* позволяет установить время, прежде чем значение *soft limit* будет приведено в жизнь на файловой системе с включенными квотами. Например, этот параметр может быть использован для предупреждения пользователей о новой политике, которая установит дисковую квоту в 5 *MB* на их домашний каталог через 7 дней. Вы можете установить это значение в 0 дней (по умолчанию) для любого отрезка

времени. Чтобы изменить это, требуется два следующих шага (в моем примере я принимаю 7 дней).

Шаг 1. Редактирую значение по умолчанию параметра периода любезности *(grace period), используя следующую команду:*

[root@deep /]# edquota -t
Time units may be: days, hours, minutes, or seconds
Grace period before enforcing soft limits for users:
/dev/sda6: block grace period: 0 days, file grace period: 0 days

Шаг 2. Модифицирую период любезности (*grace period*) до 7 дней. Измените или установите следующие параметры в редакторе *nano*:
Time units may be: days, hours, minutes, or seconds
Grace period before enforcing soft limits for users:
/dev/sda6: block grace period: 0 days, file grace period: 0 days
должно читаться:
Time units may be: days, hours, minutes, or seconds
Grace period before enforcing soft limits for users:
*/dev/sda6: block grace period: **7** days, file grace period: **7** days*

Команда *edquota -t* редактирует параметр *soft time limits* для каждой файловой системы с включенными квотами;

б) назначение квот для отдельных групп.

Предположим, например, что есть группа с именем *webusers*. Следующая команда вызовет редактор *nano* для редактирования квот для группы *webusers* на каждой файловой системе, где квоты разрешены:

[root@deep /]# edquota -g webusers
Quotas for group webusers:
/dev/sda6: blocks in use: 6, limits (soft = 0, hard = 0)
inodes in use: 6, limits (soft = 0, hard = 0)

Процедура такая же, как и при назначении квот для пользователей; как описано выше, вы должны модифицировать параметр *soft* = и записать изменения;

в) назначение квот для групп пользователей с теми же значениями.

Программа *edquota* имеет специальную опцию (*-p*), которая назначает квоты для групп пользователей с некоторым значением, назначенным при инициализации пользователя. Допустим, вы хотите назначить пользователям *UID*-ы которых начинаются с 500 то же значение, что и для пользователя *sib*.Сначала мы редактируем квоты для пользователя *sib*, а затем выполняем следующую команду:

[root@deep /]# edquota -p sib `awk -F: '$3 > 499 {print $1}' /etc/passwd`

Программа *edquota* будет дуплицировать квоты, которые установлены для пользователя *sib*, на всех пользователей с *UID* больше 499 из файла */etc/passwd*.

6 Управление процессами

Все запущенные процессы имеют уникальные номера - *PID*. Команды *Linux* необходимые для мониторинга работы ОС (все показания выводятся на экран в реальном времени, а число, стоящее после команды, означает интервал между выводом информации):

- *top* - информация в реальном времени о процессах, потребление ОЗУ;
- *ps* – показать все процессы;
- *ps aux* - показать все загруженные процессы;
- *echo $$* - показать *PID* оболочки;
- *fuser -va 22/tcp* - показать *PID* процесса, использующий порт 22;
- *fuser -va /home* - показывает *PID* процесса, имеющего доступ к /*home*;
- *lsof /home* - список процессы, которые используют /*home*;
- *kill* 1234 - «убить» процесс с *PID* 1234;
- *killall* <имя_программы> - "убить" все процессы по имени программы;
- *nice* и *renice* – назначение и переназначение приоритетов процессам;
- *ps axu | grep test* - все процессы, запущенные от пользователя *test.*

Возможность для пользователя задавать значение приоритета его собственных процессов определяет отношение к другим пользователям системы, ведь процесс - это набор правил, которым руководствуется данная программа при использовании выделенного процессорного времени, памяти и ресурсов ввода/вывода. Каждый процесс, запущенный в системе имеет свой *ID (PID)*, по которому его можно отслеживать. Ядро позволяет собирать различную информацию о каждом процессе, которая включает, но не ограничивается:

- статус процесса (работает, спит, зомби или остановлен);
- приоритет выполнения процесса;
- информация об используемых ресурсах;
- владелец процесса;
- сетевые порты и файлы, открытые процессом и т.д.

Создаем процесс *yes* в терминале, перенаправив вывод в */dev/null:*

yes > /dev/null &

[1] 5997

Воспользуемся командой чтобы извлечь информацию о процессе: *ps -l*

```
F S   UID   PID  PPID  C  PRI  NI ADDR SZ WCHAN  TTY        TIME CMD
0 S  1000  5830  3283  0   80   0 -  6412 wait   pts/0    00:00:00 bash
0 R  1000  5997  5830 99  80   0 -  1757 -      pts/0    00:00:09 yes
0 R  1000  5998  5830  0   80   0 -  2399 -      pts/0    00:00:00 ps
```

Из этой таблицы можно узнать, что:

F - FLAG: процесс запущен без привилегий суперпользователя. В противном случае мы могли бы увидеть число 4 или сумму 1 и 4.

S - STATE: процесс в настоящее время работает.

UID - ID пользователя, инициализировавшего процесс.

PID - ID процесса нашей команды *yes* 5997.

PPID - Parent Process ID. Это *ID* родительского для нашей команды yes процесса. В нашем случае это bash с *PID* 5830.

C - загрузка процессора, выражается в %.

PRI - приоритет процесса, большее значение значит меньший приоритет.

NI - значение *Nice*, которое находится в диапазоне от -20 до 19. Большее значение означает меньший приоритет.

Принцип работы планировщика *Linux* (для ядра версии >= 2.6) вытесняющий, то есть способность ядра выбирать среди всех заданий то, которое имеет наивысший приоритет. Далее, ядро делит списки приоритета на задачи реального времени и пользовательские задания, ранжирующиеся от 1 - 100 и 101 - 140 соответственно. Далее, ядро выделяет задачам с более высоким приоритетом больший квант времени, а задачам с меньшим приоритетам - меньший квант времени, который в среднем составляет 200 и 10 мс соответственно. Другими словами, каждое задание допускается к выполнению только, если у него остается какая-либо часть времени. Поэтому меньший отрезок времени для выполнения означает, что процесс получает меньше времени в очереди выполнения и, соответственно, получает меньше ресурсов. Когда отрезок времени процесса заканчивается, он помещается в очередь выполнения с истекшим временем, затем его приоритет пересчитывается, и он снова помещается в активную очередь выполнения. Эта зависимость иллюстрируется приведенной здесь диаграммой. Важно помнить, что обе очереди выполнения содержат списки задач, отсортированных по их приоритету.

По умолчанию nice устанавливает значение приоритета 10:

$ nice yes > /dev/null &

[1] 5199

$ ps -l

```
F S   UID   PID   PPID C  PRI NI ADDR SZ WCHAN  TTY      TIME     CMD
0 S  1000  3383  3380  0   80   0 -  6445       wait     pts/0    00:00:00 bash
0 R  1000  5199  3383 99 90   10 -1757 -                pts/0    00:00:07 yes
0 R  1000  5200  3383  0  80   0 -  2399 -                pts/0    00:00:00 ps
```

Чтобы запустить процесс со значением *nice*, отличным от 10, можно использовать ключ *–n*: *$ nice -n 15 yes > /dev/null &*

[1] 5270

$ ps -l

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY      TIME     CMD
0 S  1000  3383  3380  0  80   0 -   6447 wait     pts/0    00:00:00 bash
0 R  1000  5270  3383 99  95 15 -   1757 -                pts/0    00:00:02 yes
0 R  1000  5271  3383  0  80   0 -   2399 -                pts/0    00:00:00 ps
```

Чтобы установить значение *nice* ниже нуля, требуются права суперпользователя. В противном случае будет установлено значение 0. Если попробовать задать значение *nice* -1 без прав *root*:

$ $ nice -n -1 yes > /dev/null &

[1] 5285

nice: cannot set niceness: Permission denied

$ ps -l

```
F S   UID   PID  PPID  C  PRI  NI ADDR SZ WCHAN  TTY        TIME  CMD
0 S  1000  3383  3380  0   80   0 -  6447 wait   pts/0    00:00:00 bash
0 R  1000  5285  3383 95  80   0 -  1757 -      pts/0    00:00:07 yes
0 R  1000  5295  3383  0   80   0 -  2399 -      pts/0    00:00:00 ps
```

Поэтому, чтобы задать значение *nice* меньше 0, необходимо запускать программу как *root*, или использовать *sudo*.

 # nice -n -1 yes > /dev/null &

[1] 5537

 # ps -l

```
F S   UID   PID  PPID C  PRI  NI ADDR SZ WCHAN  TTY        TIME  CMD
4 S     0  5428  3383  0  80   0 - 14430 wait   pts/0    00:00:00 su
0 S     0  5436  5428  1  80   0 -  7351 wait   pts/0    00:00:00 bash
4 R     0  5537  5436 87 79  -1 -  1757 -      pts/0    00:00:04 yes
4 R     0  5538  5436  0  80   0 -  2399 -      pts/0    00:00:00 ps
```

Пробуем изменить значение *nice* у запущенной программы с помощью команды *renice (*есть работающая программа *yes* со значением *nice* 10):

ps -l

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY        TIME  CMD
0 S  1000  3383  3380  0  80   0 -  6447 wait   pts/0    00:00:00 bash
0 R  1000  5645  3383 99  90  10 -  1757 -      pts/0    00:00:04 yes
0 R  1000  5646  3383  0  80   0 -  2399 -      pts/0    00:00:00 ps
```

Изменим значение *nice* на 15:

renice -n 15 -p 5645

5645 (process ID) old priority 10, new priority 15

ps -l

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY        TIME  CMD
0 S  1000  3383  3380  0  80   0 -  6447 wait   pts/0    00:00:00 bash
0 R  1000  5645  3383 99  95  15 -  1757 -      pts/0    00:00:31 yes
0 R  1000  5656  3383  0  80   0 -  2399 -      pts/0    00:00:00 ps
```

Согласно правилам, обычный пользователь может только увеличивать значение *nice* (уменьшая приоритет) любого процесса. Если попробовать изменить значение *nice* с 15 до 10, получим следующее сообщение об ошибке:

renice -n 10 -p 5645

renice: failed to set priority for 5645 (process ID): Permission denied

Команда *renice* позволяет суперпользователю изменять значение *nice* процессов любого пользователя. Это делается с помощью ключа *-u*. Следующая команда изменяет значение приоритета всех процессов пользователя на -19:

renice -n -19 -u lubos

1000 (user ID) old priority 0, new priority -19.

7 Графические оболочки

X Window System - оконная система, обеспечивающая стандартные инструменты и протоколы для построения графического интерфейса пользователя. Используется в *UNIX*-подобных ОС.

X Window System обеспечивает базовые функции графической среды: отрисовку и перемещение окон на экране, взаимодействие с устройствами ввода, такими как, например, мышь и клавиатура; но она не определяет деталей интерфейса пользователя - этим занимаются менеджеры окон, которых разработано множество. По этой причине внешний вид программ может очень сильно различаться в зависимости от возможностей и настроек конкретного оконного менеджера.

В *X Window System* предусмотрена сетевая прозрачность: графические приложения могут выполняться на другой машине в сети, а их интерфейс при этом будет передаваться по сети и отображаться на локальной машине пользователя (в случае, если это разрешено в настройках). В контексте *X Window System* термины «клиент» и «сервер» имеют непривычное для многих пользователей значение: «сервер» означает локальный дисплей пользователя (дисплейный сервер), а «клиент» - программу, которая этот дисплей использует (она может выполняться на удалённом компьютере).

X Window System часто называют *X11* или просто *X* (в разговорной речи «иксы́»).

На рисунке 17 *X* сервер принимает ввод с клавиатуры и мыши и производит вывод на экран. На пользовательской рабочей станции выполняются веб-браузер и эмулятор терминала. Программа обновления системы работает на удалённом сервере, но управляется с машины пользователя. Обратите внимание, что удалённое приложение работает так же, как если бы оно выполнялось локально.

X Window System использует клиент-серверную модель: *X* сервер обменивается сообщениями с различными клиентскими программами. Сервер принимает запросы на вывод графики (окон) и отправляет обратно пользовательский ввод (от клавиатуры, мыши или сенсорного экрана).

Протокол, с помощью которого общаются сервер и клиент, является прозрачным для сети: клиент и сервер могут находиться как на одной машине, так и на разных. В частности, они могут работать на различных архитектурах под управлением разных ОС - результат будет одинаковым. Клиент и сервер могут даже безопасно взаимодействовать через Интернет посредством туннелированного соединения сквозь зашифрованный сетевой сеанс.

Чтобы запустить удалённую клиентскую программу, выводящую графику на локальный X сервер, пользователь обычно открывает эмулятор терминала и подключается к удалённой машине при помощи *telnet* или *SSH*.

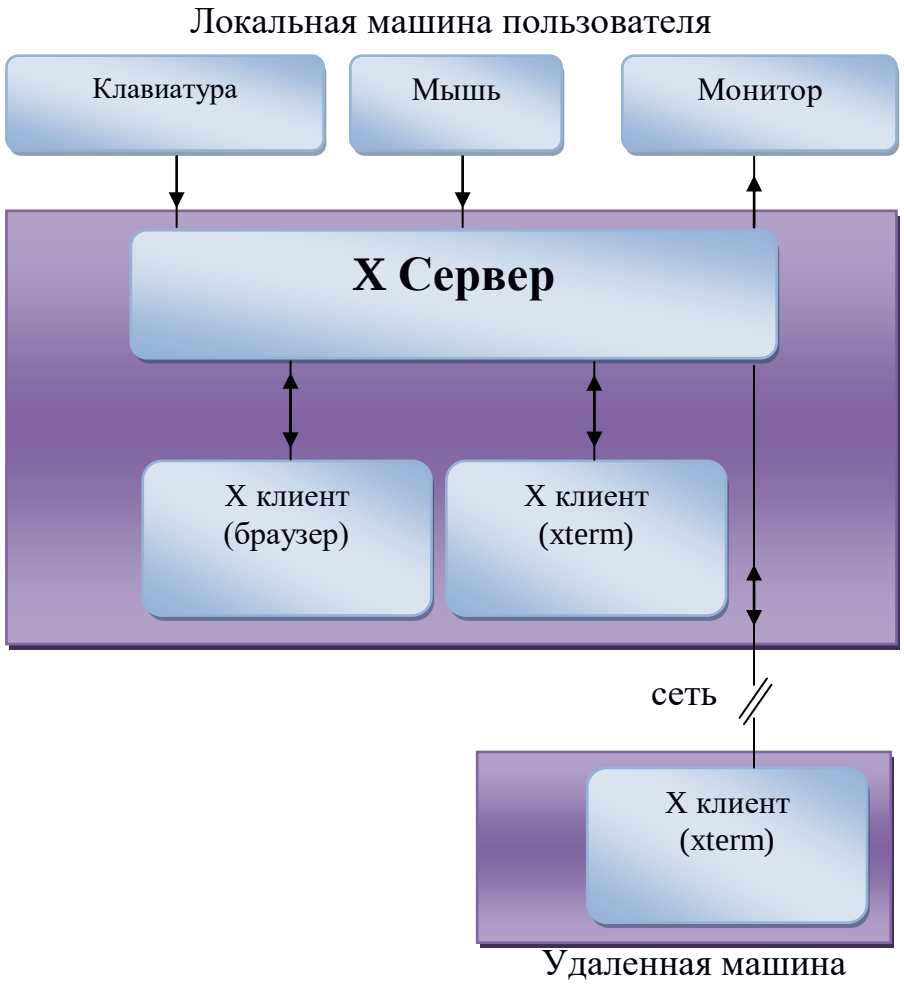

Рисунок 17 – Взаимодействие локальной и удаленной машин

Затем он отдаёт команду, указывающую дисплей, на который следует выводить графику (например, *export DISPLAY*=[имя компьютера пользователя]:0 при использовании *bash*). Наконец, пользователь запускает клиентскую программу. Она подключится к локальному X серверу и будет отображать графику на локальный экран и принимать ввод от локальных устройств ввода. Другой вариант - использовать небольшую вспомогательную программу, которая подключается к удалённой машине и запускает на ней нужное клиентское приложение.

Использование удалённых клиентов может быть полезно в ситуациях:

- графическое администрирование удалённой машины;
- выполнение интенсивных ресурсоёмких вычислений на удалённой *UNIX*-машине и вывод результатов на локальной *Windows*-машине;
- выполнение графических программ одновременно на нескольких машинах, с одним дисплеем, одной клавиатурой и одной мышью.

Самыми распространёнными интерфейсами пользователя в среде *Linux* являются: *KDE, Gnome, LXDE, Xfce* и т.п., их великое множество.

8 Эмуляторы и виртуальные машины

Wine - свободное программное обеспечение, позволяющее пользователям *UNIX*-подобных систем исполнять приложения *Microsoft Windows*. *Wine* также предоставляет программистам библиотеку программ *Winelib*, при помощи которой они могут компилировать *Windows*-приложения для портирования их в *UNIX*-подобные системы.

Название *Wine* является рекурсивным акронимом и расшифровывается «*Wine Is Not an Emulator*» - «*Wine* - не эмулятор» (имеется в виду, что *Wine* не является эмулятором компьютера, как, например, *qemu* или *VirtualBox, Wine* - это альтернативная реализация *Windows API*). *Wine* распространяется на условиях лицензии *GNU LGPL*.

При выборе настольной платформы виртуализации сегодня у пользователей по-сути всего два выбора - *VMware Workstation* и *Oracle VirtualBox*. Остальные аналоги настольных продуктов либо уже сняты с производства, либо откровенно не дотягивают до функционала этих двух платформ. При этом *VMware Workstation* является полноценным коммерческим продуктом с закрытым исходным кодом (исходный код открыт только у *VMware Player* - урезанной версии *Workstation*), а *Oracle VirtualBox* – *open-source* платформа, работающая поверх многих ОС. По отзывам пользователей оба продукта показывают одинаковую производительность в средних условиях (хотя бытует мнение, что *VirtualBox* быстрее).

У обеих платформ *VirtualBox* и *VMware Workstation* есть:

– понятный графический интерфейс;

– удобный редактор сетевого взаимодействия на хосте;

– диски виртуальных машин, растущие по мере наполнения их данными;

– технология мгновенных снимков (снапшотов);

– технология приложений в хостовой ОС из гостевой ОС в бесшовных окнах (то есть, приложение из виртуальной машины «выносится» в рабочую область хостовой системы, как будто оно в ней и работает;

– поддержка большого количества гостевых ОС, поддержка *Windows* и *Linux* в качестве гостевых ОС;

– поддержка 64-битных гостевых ОС;

– поддержка *Intel VT* и *AMD-V*;

– *USB* 2.0 устройства в виртуальных машинах;

– воспроизведение звука на устройствах хоста из виртуальной машины;

– буфер обмена между гостевой и хостовой ОС;

– поддержка *3D*-графики для игр и других приложений;

– поддержка импорта виртуальных модулей (*Virtual Appliances*);

– улучшенные драйверы в гостевой ОС: *VMware Tools* и *VirtualBox Guest Additions* (оба пакета обновляются автоматически);

– поддержка технологии *Memory Overcommit* (так называемый *Memory Ballooning* - перераспределение свободной физической памяти между гостевыми ОС виртуальных машин);

– поддержка многопроцессорных виртуальных машин (не менее 8 *vCPU*);

– расширение виртуальных дисков;

– копирование файлов между виртуальной машиной и ОС хоста;

– поддержка доступа к консоли виртуальной машины через *RDP*-сервер.

Что доступно в *VirtualBox*, но не хватает *VMware Workstation*:

– *VirtualBox* бесплатен;

– *VMware Workstation* работает только в хостовых ОС *Windows* и *Linux*, а *VirtualBox* поддерживает хосты *Windows*, *Linux*, *Mac OS X* и *Solaris*;

– технология «*Teleportation*», позволяющая переместить запущенную виртуальную машину на другой хост *VirtualBox*, без необходимости ее остановки. Данная функция отсутствует в *VMware Workstation*;

– *VirtualBox* имеет возможность работы не только с родным форматом *.VDI*, но и *.VMDK*, и *.VHD*. *VMware Workstation* имеет возможность исполнять виртуальные машины только из образов виртуальных дисков *VMDK* (хотя есть бесплатный продукт *VMware Converter* для импорта виртуальных машин из других форматов);

– *VirtualBox* имеет больше параметров для работы из командной строки (управление машиной, устройствами, снапшотами и многим другим);

– *VirtualBox* лучше поддерживает аудио для *Linux*-хостов (*Workstation* отключает звук в хостовой ОС, *VirtualBox* может играть параллельно);

– *VirtualBox* имеет возможность ограничения потребления ресурсов *CPU* и ввода-вывода, у *VMware Workstation* этого нет;

– *VirtualBox* имеет возможность регулировки видеопамяти.

Что доступно в *VMware Workstation*, но не хватает *VirtualBox:*

– *VMware Workstation* - коммерческий продукт, а значит можно рассчитывать на поддержку с определенным уровнем *SLA*;

– *VMware Workstation* имеет больше возможностей для поддержки 3D-графики, как то: *Windows Aero user interface*, *OpenGL* 2.1 и *Shader Model* 3.0. Сама *3D*-акселерация работает стабильней, чем в *VirtualBox*.

– *VMware Workstation* имеет драйвер универсальной печати.*ThinPrint* (не требуется установка драйверов в гостевую ОС);

– создание снапшотов через заданные интервалы времени (функции *AutoProtect*), что позволяет защитить виртуальные машины по аналогии с возможностью автосохранения (например, как в *Microsoft Word*);

– *Compact Virtual Disks* - сжатие виртуальных дисков для отдачи его под нужды других систем;

– *VMware Workstation* имеет более широкий функционал по работе с виртуальным сетевым взаимодействием - коммутаторы, *DHCP*, *NAT* и прочее

(хотя *VirtualBox* также имеет *NAT, Bridge Networking* - в *Workstation* это удобнее);

– *VMware Workstation* имеет функционал связанных клонов (*Linked Clones*) для виртуальных машин;

– запись активности виртуальной машины в видеоформате, а также в виде последовательности действий пользователя (*Guest Record* / *Replay*);

– *Workstation* имеет возможности интеграции со средами разработки и тестирования (например, *Eclipse*), а также специализированные функции для разработчиков ПО;

– защита виртуальных машин 256-битным шифрованием;

– в *Workstation* несколько приятных мелочей - типа ярлыков на приложения из меню «Пуск», *Pause a Virtual Machine (не suspend*) и т.п.

В довершении *VirtualBox* «кушает» существенно меньше ресурсов, однако, на нем весьма многие специализированные ОС не запускаются, также, зачастую, софт внутри виртуальной ОС не работает. Однако в *VMWare WKS* они прекрасно работают, так как *VMWare* гораздо более полно эмулирует железо, отсюда и требования к ресурсам.

Teamviewer – пакет программного обеспечения для удаленного контроля компьютеров, обмена файлами между управляющей и управляемой машинами, видеосвязи и веб-конференций. Работает на ОС: *Windows, Mac OS X, Linux, iOS, Android.*

Чтобы установить связь между компьютерами, клиент-оператор должен связаться с удаленным оператором и узнать его логин и пароль, а затем ввести их в клиент *Teamviewer.*

9 Работа с загрузчиками

Понимание процесса загрузки системы очень важно, так как позволяет находить и исправлять различные ошибки и сбои в работе системы, связанные с загрузкой ОС. Далее приводится последовательная схема загрузки ОС *Linux*.

Описание первого этапа (рисунок 18): *BIOS* из *MBR* (первые 512 байт диска, выбранного для загрузки) загружает *First Boot Loader*. *FSB* находит вторичный загрузчик, используя таблицу разделов, просматривая ее, обнаруживает активный раздел, после обнаружения этого раздела - загружает *SSB* в оперативную память и запускает его. Для корректной загрузки, активный раздел должен содержать каталог /*boot*, который должен находиться в начале диска и содержать *Second Stage Boot Loader*. В целом, *SSB* - это программа, которая выводит список вариантов загрузки (меню выбора загрузки ОС). Загрузчиком может быть *LILO* (более старый) или *GRUB*. Загрузчик берет свои настройки из конфигурационного файла (*/etc/lilo.conf* - для *LILO* и */boot/grub/grub.conf* - для *GRUB2* или */boot/grub/menu.lst* - для *GRUB Legacy*). Существуют, конечно, и другие загрузчики, такие как *syslinux, PXElinux, Isolinux, uBoot*. Исторически (до появления загрузчиков *LILO, GRUB*

и др. и когда образ ядра занимал объем не более 1,44 Мб) данного этапа не существовало, и загрузка происходила с дискеты без файловой системы, на которую был записан образ ядра *Linux*, который (образ) содержал в себе *MBR*, то есть *BIOS* загружал сразу образ ядра и передавал ему управление.

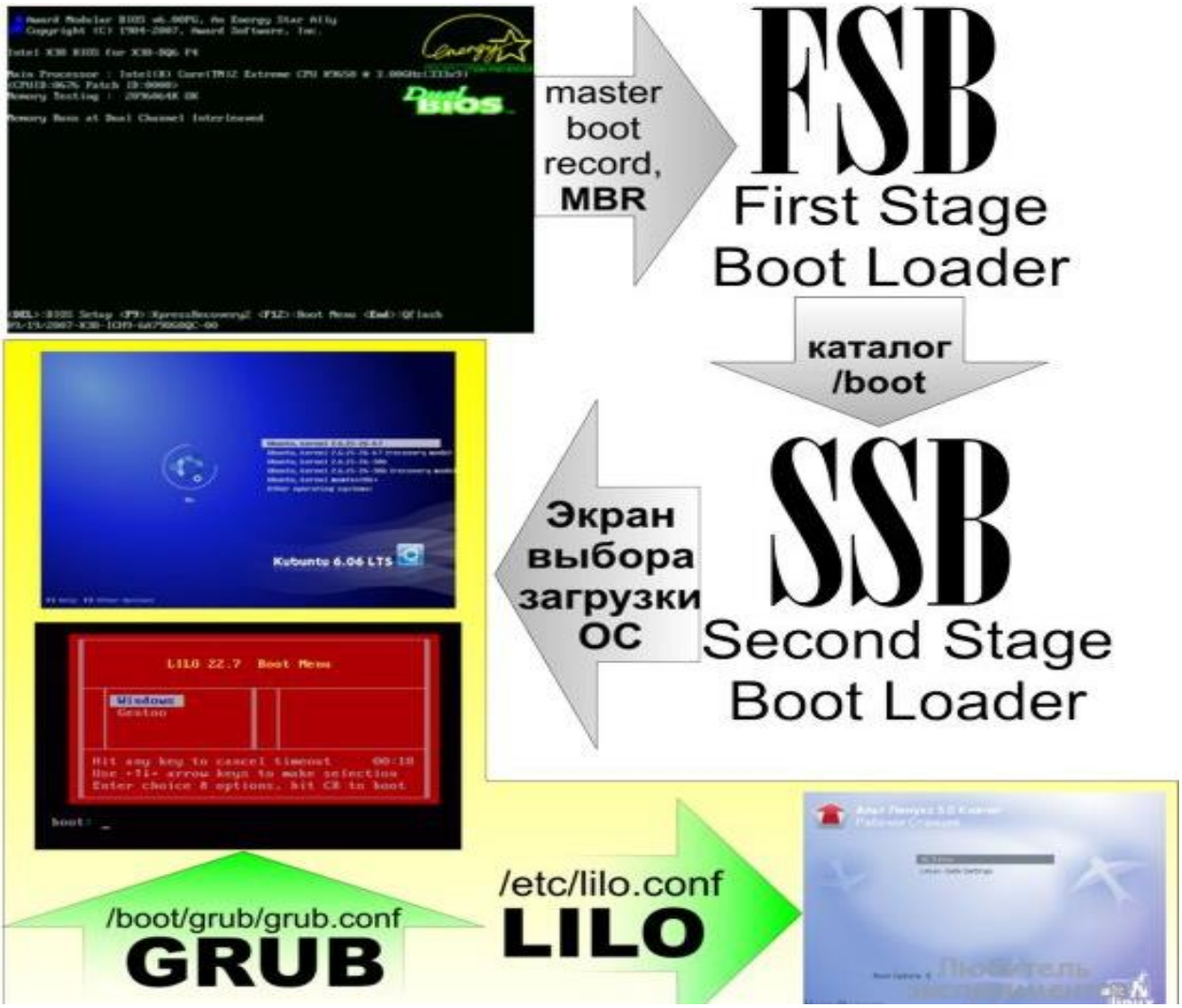

Рисунок 18 – Процесс загрузки ОС, этап 1

Описание второго этапа (рисунок 19): подготовка системы для запуска демонов (служб). При подготовке загрузчик загружает в память образ ядра из каталога /boot.

Рассмотрим пример образа ядра на примере ОС Debian 6:

```
boot@debian:~# file /boot/vmlinuz-2.6.32-5-686
/boot/vmlinuz-2.6.32-5-686: Linux kernel x86 boot
executable bzImage, \ 2.6.32-5-686 (unknown@Debian)
#, RO-rootFS, swap dev 0x2, Normal VGA
```

Команда *file* выводит информацию о файле образа ядра: это ядро Линукс (*Linux kernel*), 32-битной архитектуры (x86), содержащей возможность загрузки (*boot*), исполняемый (*executable*), в формате *bzImage* (то есть сжатое, бывают образы несжатые), далее указывается версия ядра и кое-какие другие

параметры образа. Данных файлов может быть несколько (в зависимости от количества установленных версий ядра) и для загрузки выбирается тот, который указан в настройках загрузчика. Образ ядра, инициализирует и подготавливает память, процессор, остальное оборудование, монтирует корневой раздел в режиме только для чтения для загрузки остальной системы (устройство и раздел на котором размещен корень системы должен быть указан в настройках загрузчика *GRUB* (*/boor/grub.conf*) или *LILO (/boor/lilo.conf)*) в виде параметра *root=*. При этом, выводится сообщение *VFS: Mounted root (ext2 filesystem) readonly*. Кроме того, ядро из файла конфигурации загрузчика получает параметры загрузки, такие как корневая файловая система, отображать сообщения ядра или нет и т.п.

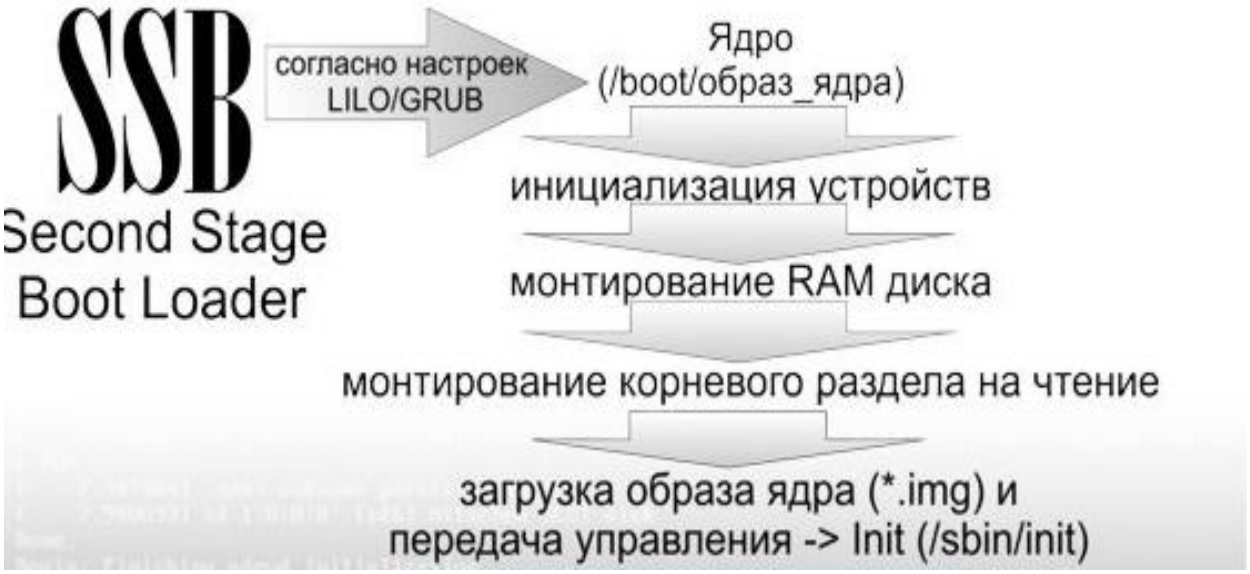

Рисунок 19 – Процесс загрузки ОС, этап 2

Параметры, переданные текущему загруженному ядру, можно посмотреть в файле */proc/cmdline*. Вот пример параметров все того же *Debian*:

```
boot@debian:~# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-2.6.32-5-686
root=UUID=6852e86c-b8f1-49d0-b1eb-9d10171083c3 ro quiet
```

Так как ядро Linux является модульным, то при загрузке может возникнуть необходимость подключить модуль ядра, который находится на еще не примонтированной файловой системе. Для решения данной проблемы при загрузке подгружается архив файловой системы (он же инициализационный *RAM* диск или *initrd*), содержащий в себе необходимый для загрузки набор модулей ядра. Так он выглядит для указанного выше ядра:

```
root@debian:~# file /boot/initrd.img-2.6.32-5-686
/boot/initrd.img-2.6.32-5-686: gzip compressed data
```

Какой архив initrd подгружать при загрузке указывается в GRUB:

```
boot@debian:~# grep initrd -B4 /boot/grub/menu.lst
title           Debian GNU/Linux, kernel 2.6.26-2-686
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.26-2-686 root=/dev/sda1
initrd          /boot/initrd.img-2.6.26-2-686
```

Так как стандартный вывод (вывод сообщений на дисплей) должен быть связан с каким-либо процессом, соответственно с идентификатором процесса, а у ядра нет идентификатора, оно помещает сообщения ядра (и модулей) в буфер кольца ядра и выводит на экран. Данный буфер еще называется *dmesg*. Его содержимое можно просмотреть, выполнив команду *dmesg*. После полной инициализации ядро передает управление процессу init (первому системному процессу с *PID*=1). На экран выводится сообщение *INIT: version 2.76 booting*. При этом, бинарный файл *init* последовательно ищется в корневом разделе в каталогах: /*sbin/init, /etc/init, /bin/init*, если в указанных местах не обнаружен файл, то ядро пытается запустить шелл /*bin/sh* (это, собственно, есть однопользовательский режим загрузки, он же режим восстановления). При этом не запускается ни один демон. Если не найден и шелл, то идет ошибка *Kernel panic: No init found. Try passing init= option*. Данная ситуация может возникнуть скорее всего, потому что неверно смонтирован корневой раздел.

Описание третьего этапа представлено на рисунке 20: до текущего момента процесс запуска любой *UNIX*-системы практически не отличался.

Третий этап загрузки может отличаться в зависимости от платформы, будь то *Linux, BSD, MacOS* и др. Далее будет рассмотрен процесс загрузки *Linux* с реализацией процесса запуска с помощью пакета *sysvinit* (так же именуемого *System V* – «систем 5»). В *Linux* в последнее время на смену *SysV* (а) внедряется разработанный «убунтологами» пакет *upstart* (б), в системах *BSD* происходит иное:

а) на третьем этапе загрузки *System V* происходит следующее:

после запуска, процесс *init*, согласно конфигурации в файле */etc/inittab* (а точнее строке, начинающийся на *si::sysinit:/etc/...)* первым делом выполняет скрипт */etc/rc.d/rc.sysinit* (для *RedHat*) или */etc/init.d/rcS* (для *Debian*), которые выполняют базовое конфигурирование системы (загрузка модулей, проверка корневой файловой системы и монтирование не чтение/запись, установка имени хоста, времени, монтирование оставшихся разделов, запуск сети, монтирование сетевых файловых систем и др.), а также данный скрипт с помощью утилиты *initlog* направляет сообщения о загрузке в */var/log/messages*. На данном этапе, нет уровня выполнения.

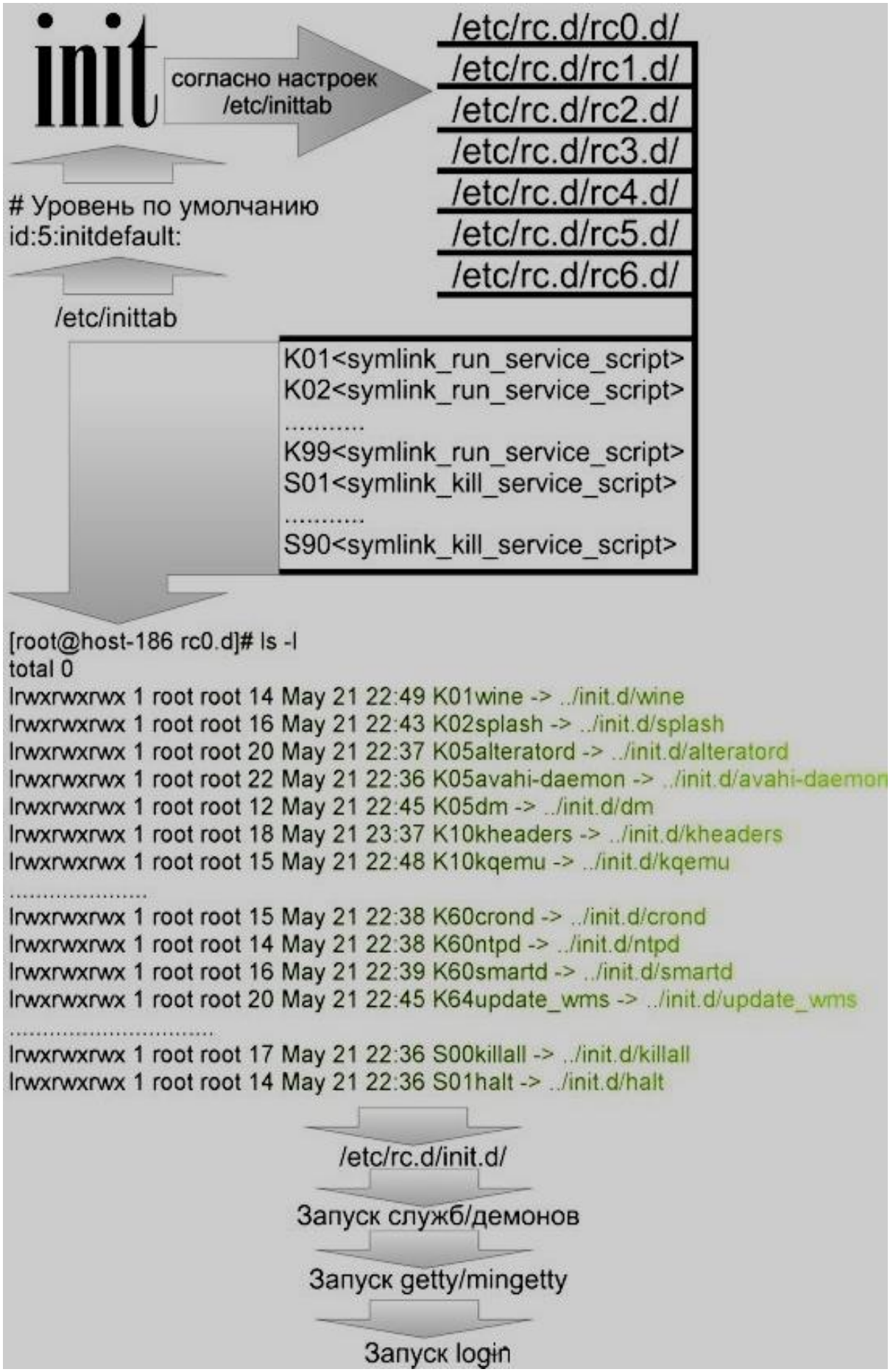

Рисунок 20 – Процесс загрузки ОС, этап 3

Далее, запускается скрипт инициализации */etc/rc.d/rc* (для *RedHat*) или */etc/init.d/rc* (для *Debian*), которому передается уровень запуска в виде параметра от 0 до 6 (в соответствии с настройками из файла */etc/inittab*, в котором указан уровень загрузки (выполнения) ОС по умолчанию и каталог */etc/rc*.d*, в котором расположены скрипты запуска демонов/служб для соответствующего уровня запуска), запускает скрипты из каталога, соответствующего текущему уровню запуска.

Далее, процесс *init* согласно уровню загрузки, просматривает каталог */etc/rc.d/rc0.d/* (в данном примере, цифра 0(ноль) в имени *rc0.d* соответствует уровню загрузки - нулевому), в котором содержатся симлинки (ссылки) на скрипты запуска системных служб, которые, в свою очередь, расположены в /etc/rc.d/init.d/ (для *Red Hat*) или в /etc/init.d/ (для *Debian*). Ссылки имеют следующий формат:

<*S*|*K*><число><имя_службы>, в котором: *S* - запуск службы (*Start), K* - остановка службы (*Kill*), <число> - число, определяющее последовательность запуска служб (00 - самая первая), <имя_службы> - имя запускаемой службы.

Уровни выполнения бывают следующие:

– 0: полная остановка машины;

– 1: *single-use*r (однопользовательский) режим; *(используется в случае серьезных проблем или для восстановления системы);*

– 2: *multi-user* (многопользовательский) режим, без поддержки сети;

– 3: *Multi-user* (многопользовательский) режим с поддержкой сети (используется преимущественно на серверных системах);

– 4: неиспользуемый;

– 5: *Multi-user* (многопользовательский) режим с поддержкой сети + графический интерфейс для входа в систему (*login*);

– 6: перезагрузка.

При этом в разных дистрибутивах возможны вариации уровней загрузки. Дистрибутив *Slackware* использует уровень выполнения 4 вместо 5 для полного запуска системы *X Window*. *Debian* использует один уровень выполнения для любого многопользовательского режима, обычно уровень 2.

После запуска всех демонов, содержащихся в каталоге */etc/rc.d/rc0.d/*, процесс *Init* запускает сценарий */etc/rc.d/rc.local* (для *RedHat*) или */etc/rc.local* (для *Debian*). В данном сценарии можно разместить свои настройки, которые вступят в силу после запуска всех демонов.

Далее, процесс *Init* запускает процесс *mingetty*, который запрашивает имя пользователя (о расположении *mingetty*, также сказано в файл */etc/inittab*). После ввода имени пользователя, mingetty передает введенную информацию процессу *login*. Процесс *login* просматривает файлы */etc/passwd* и */etc/shadow* на наличие указанного пользователя и выводит запрос пароля. После ввода пароля пользователя, процесс *login* сравнивает хеш пароля с данными в файле */etc/shadow* и в случае совпадения, запускает оболочку (шелл).

Указанный процесс актуален для входа в текстовую консоль, при входе в графическую оболочку, вместо *mingetty*, процесс *init* запускает процесс *gdm* (для *GNOME*) или *kdm* (для *KDE*), в зависимости от того, какой оконный менеджер установлен в системе. *gdm* (или *kdm*) запускает *X*-сервер и выводит окно аутентификации.

После успешной аутентификации, просматривается файл конфигурации */etc/group*, который определяет принадлежность пользователя к группам. А также запускается стартовый конфигурационный сценарий */etc/X11/gdm/PreSession* (для *X*-сервера) и конфигурационные стартовые скрипты (*/etc/profile* и др.) для bash, которые устанавливают настройки окружения пользователя.

Уровнем выполнения можно управлять с помощью команд;

б) на третьем этапе загрузки *Upstart* происходит следующее:

Затронем вопрос загрузки системы *Upstart*, которая введена в действие с *Ubuntu 7.10* и старше, *Fedora 9, RHEL 6*. Основное отличие *Upstart* от *System V* в том, что работа *Upstart* основана на обработке событий. Одной из основных задач внедрения Upstart была - уйти от последовательного запуска сервисов в *SysV*, тем самым ускорить загрузку ОС, сделав процесс запуска служб параллельным. Итак, система *Upstart* при запуске процессом *init* генерирует событие *startup* (запуск - одно из двух основных событий) - старт системы, а событие *shutdown* - при выключении системы. Другое ключевое событие - это *ctrlaltdel*, которое указывает, на то, что вы нажали клавиши *Ctrl-Alt-Delete*. В соответствии с событием генерируемым процессом *init*, обрабатываются конфигурационные файлы (**.conf*) из каталога */etc/init/* (в более старых версиях *Upstart* - каталог */etc/event.d/*). В этом, собственно, и заключается вся работа *Upstart*. Для обратной совместимости с *System V* существует файл */etc/init/rc.conf*, который запускает демоны, согласно *SysV*.

Рассмотрим содержимое файлов */etc/init/*.conf*. Каждый из файлов должен содержать *start on event* (по какому (*on*) событию (*event*) осуществлять запуск (*start*) службы), *stop on event* (по какому событию останавливать запуск службы), *exec* либо *script* (что, собственно, запустить по указанному выше событию). Наиболее часто применяемые события в конфигурационных файлах - это *startup, runlevel, stopped u started*. *Startup* - это событие запуска ОС, *runlevel* (с указанием уровня или нескольких уровней) - событие при смене уровня запуска, *stopped* - событие после остановки задания, *started* - событие после завершения старта задания. Просмотрев конфигурационные файлы в каталоге */etc/init/* можно найти и другие события, о которых можно почитать в *man upstart-events (7)*. В дополнение к стандартным записям *exec* и *script* существуют записи *pre-start* и *post-stop*, реализующие выполнение подготовительных действий до выполнения *exec* и завершающие после завершения *exec*.

Данной системой загрузки также можно управлять командами.

Для отключения автоматического запуска службы в *Upstart* надо:

– переименовать конфигурационный файл запуска службы в каталоге */etc/init* в файл без расширения «*.conf*»;

– или закомментировать строку «*start on»* с помощью символа '#';

в) на третьем этапе загрузки *BSD* происходит следующее:

По-умолчанию на 1 этапе там используется загрузчик *BSD Loader*, работа которого аналогична любому другому загрузчику и тоже разбита на подэтапы (загрузка *FSB, SSB* и т.д.). Файл конфигурации *BSD Loader* расположен в */boot/loader.conf*. Соответственно, после запуска *BSD Loader*, происходит передача управления ядру и запуск процесса */sbin/init*. Дальше запуск несколько отличается от *Linux*, так как в *BSD* нет понятия *runlevel*, то есть существует единый уровень исполнения. НО в *BSD* есть 2 режима загрузки - одно- (режим восстановления) и многопользовательский.

В однопользовательском режиме загрузка происходит:

– при выборе соответствующего пункта (*Boot in single user mode*) в меню начального загрузчика;

– при задании команды *boot -s* в командной строке загрузчика (после выбора пункта его меню *Escape to loader prompt*);

– при обнаружении серьезных (неустранимых автоматически) нарушений целостности файловой системы в ходе ее проверки на первой стадии инициализации.

При загрузке в однопользовательском режиме происходит запуск оболочки без пароля с правами суперпользователя и монтирование корневой файловой системы только на чтение. Запуска сценариев инициализации не происходит.

При загрузке в многопользовательском режиме, как и в *Linux*, выполняется 2 этап загрузки. На 3 этапе в *BSD* задача процесса *init* аналогична - это вызов и отработка сценариев инициализации или стартовых скриптов, собранных в каталоге */etc* и его подкаталогах и получение терминала (запуск процесса *getty*), установка его свойств и подготовка к аутентификации и авторизации (ввод логина/пароля) и последующее вытеснение *getty* процессом *login*. Отличие в том, что в *BSD* имеет свои стартовые скрипты, отличные от *Linux*.

Основной стартовый скрипт - это */erc/rc*, настройки, применяемые по умолчанию для скрипта */etc/rc* процесс *init* считывает из файла */etc/defaults/rc.conf*, а настройки, специфичные для текущей системы, из */etc/rc.conf*, после чего осуществляется монтирование файловых систем, перечисленных в файле */etc/fstab*, запуск сетевых служб, различных системных демонов и, наконец, выполнение скриптов запуска дополнительно установленных пакетов. Строки */etc/rc.conf* имеют вид *параметр=«значение»* (точнее *service_name_enable=«YES/NO»*). Соответственно, *service_name_ enable* - это имя службы, которое должно соответствовать имени скрипта запуска службы из каталога */etc/rc.d/** (*/etc/rc.d/service_name_enable*, например, */etc/rc.d/sshd*) или */usr/local/etc/rc.d/** - для программного

обеспечения, установленного из портов, значение *yes* или *no* соответственно включает или выключает службу.

Некоторые дополнительные системные сервисы могут быть не учтены в файле */etc/rc.conf*. Тогда для их запуска нужно прописать соответствующую команду в */etc/rc.local*

Не стоит записывать свои команды в */etc/rc.conf*. Для запуска демонов, или для выполнения нужной команды во время запуска надо записать скрипт в */usr/local/etc/rc.d.*

Процесс остановки системы *BSD*.

Во время контролируемого процесса остановки системы через утилиту *shutdown* программа *init* будет пытаться запустить скрипт */etc/rc.shutdown*, после чего будет посылать всем процессам сигнал *TERM*, а затем и *KILL* тем процессам, которые не завершили работу.

Итого, в *BSD* процесс загрузки прост - запуск скрипта */etc/rc* согласно настройкам */etc/rc.conf*, в котором прописано, какие скрипты запуска демонов из */etc/rc.d/** необходимо запустить, а какие - нет и финалом загрузки является скрипт */etc/rc.local.*

Загрузчик ОС – системное программное обеспечение, обеспечивающее загрузку ОС непосредственно после включения компьютера.

Популярным загрузчиком для *Linux*-систем является *GRUB*. Этот загрузчик до версии *GRUB* 0.9x имеет название *GRUB Legacy*, а после *GRUB2. GRUB 2* - следующая версия *GRUB Legacy*. Разработчики писали *GRUB 2* «с нуля», чтобы добиться переносимости и модульности. В связи с существованием *GRUB* 2 разработка *GRUB Legacy* прекращена, разработчики лишь принимают патчи, исправляющие ошибки.

GRUB является эталонной реализацией загрузчика, соответствующего спецификации *Multiboot* и может загрузить любую совместимую с ней ОС. Среди них: *Linux, FreeBSD, Solaris* и многие другие. Кроме того, *GRUB* умеет по цепочке передавать управление другому загрузчику, что позволяет ему загружать *Windows* (через загрузчик *NTLDR), MS-DOS, OS/2* и другие системы. После настройки *GRUB* пользователь при включении компьютера видит список операционных систем, которые установлены на его компьютер и которые можно загрузить, выбрав подходящую. *GRUB* позволяет пользователю при загрузке задавать произвольные параметры и передавать их в ядро *Multiboot*-совместимой ОС для дальнейшей обработки.

GRUB - популярный загрузчик в мире *Linux* и является загрузчиком по умолчанию в большинстве известных дистрибутивов. Менее популярным из-за скудности предоставления настроек системы является *LILO.*

При загрузке ОС не всегда меню загрузчика видно и потому, чтобы увидеть его, необходимо в процессе загрузки ОС удерживать клавишу *Shift.*

Перед загрузкой ОС удерживаем *Shift* для вызова *GRUB*, видим меню в первоначальном виде (рисунок 21), то есть на черном фоне белым шрифтом, а в выбранной записи белым шрифтом в голубом меню выведено содержимое, в

нем всего 2 загрузочные записи; при желании можно редактировать записи, нажав «e»; по умолчанию загружается первая запись.

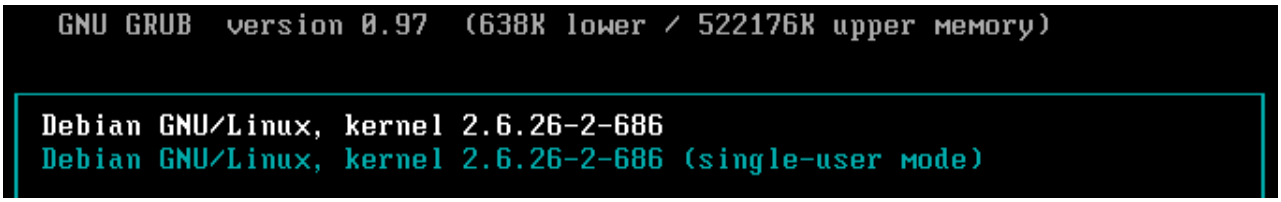

Рисунок 21 – Первоначальная цветовая схема загрузчика *Grub Legacy*

Алгоритм действий на процесс смены настроек:

– загрузить ОС;

– открыть консоль (терминал);

– залогиниться под суперпользователем, иначе конфигурационные файлы не сохранятся;

– перейти в папку /boot/grub, сделать резервную копию файла menu.lst и открыть его на редактирование (пример: gedit /boot/grub/menu.lst);

– задать внешнее оформление и параметры загрузчику;

GNU GRUB - это многосистемный загрузчик *(GRand Unified Bootloader)*. *GRUB Legacy* – это первая версия *GRUB*, сейчас очень популярен *GRUB2*. В данный момент *GRUB2* де-факто является стандартным загрузчиком *Linux*, но тем не менее *GRUB Legacy* на некоторых системах также продолжает существовать.

Часть 1. GRUB Legacy.

1) Найти строку в файле со словом «*color*» (ближе к началу), написать *color blue/red white/blue* #цветовая гамма поменялась, рисунок 22. Для проверки сохранить файл и перезагрузиться.

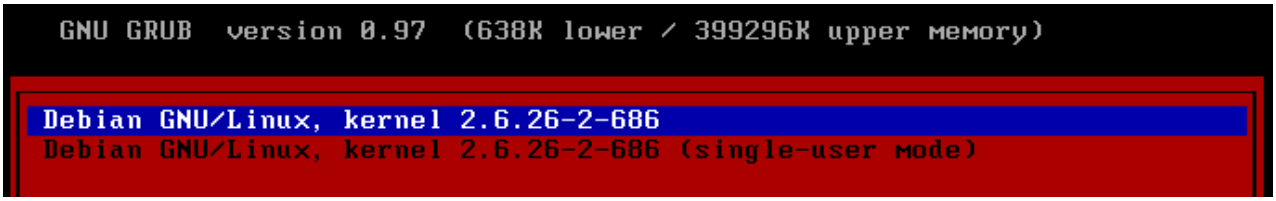

Рисунок 22 – Полученная цветовая схема загрузчика *Grub Legacy*

2) После строки с записью *color blue/red white/blue* надо дописать *hiddenmenu#* и тогда меню показывается при нажатии *Esc*, а пока оно будет таким, как показано на рисунке 23, далее сохранить файл, перезагрузиться.

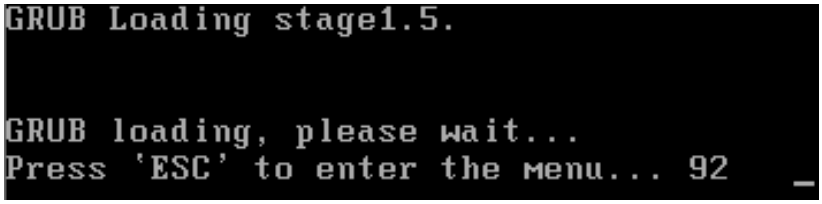

Рисунок 23 – Вид скрытого меню загрузчика *Grub Legacy*

3) После строки с *hiddenmenu* написать следующее*: timeout 30* # и по прошествии 30 секунд грузится загрузочная запись.

4) Почти в конце файла *menu.lst* найти блоки по 4 строчки, вида:

```
title           Debian GNU/Linux, kernel 2.6.26-2-686
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.26-2-686 root=/dev/sda1 ro quiet
initrd          /boot/initrd.img-2.6.26-2-686

title           Debian GNU/Linux, kernel 2.6.26-2-686 (single-user mode)
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.26-2-686 root=/dev/sda1 ro single
initrd          /boot/initrd.img-2.6.26-2-686
```

Тут видно, что доступны 2 записи (def*ault 0: Debian и default 1: Debian single-user mode*), при этом: *title* - название записи; *root* - *"hdX"* - указывает на жесткий диск, *X* указывает на первый раздел, разделы нумеруются с 0; *kernel* - путь к определенной версии ядра, которое будет загружаться; *initrd* - временная файловая система, используемая ядром при загрузке, пока реальная файловая система не примонтируется. Надо отформатировать строки так, добавляя новую загрузочную запись (№3):

```
title           -~-~-~-~-~- Debian  (obychnaya zagruzka)-~-~-~-~-~-
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.26-2-686 root=/dev/sda1 ro quiet splash
initrd          /boot/initrd.img-2.6.26-2-686

title           -~-~-~- Debian GNU/Linux (single-user mode) -~-~-~-
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.26-2-686 root=/dev/sda1 ro single
initrd          /boot/initrd.img-2.6.26-2-686

title           -~-~- Debian (zapusk komandnoi stroki root) -~-~-
root            (hd0,0)
kernel          /boot/vmlinuz-2.6.26-2-686 root=/dev/sda1 init=/bin/bash
initrd          /boot/initrd.img-2.6.26-2-686
```

Надо найти строку, в которой написано «*default*», поставить «*default 2#*» и по умолчанию будет загружаться загрузочная запись №3 (новая запись). После перезагрузки на экране появится скрин (рисунок 24).

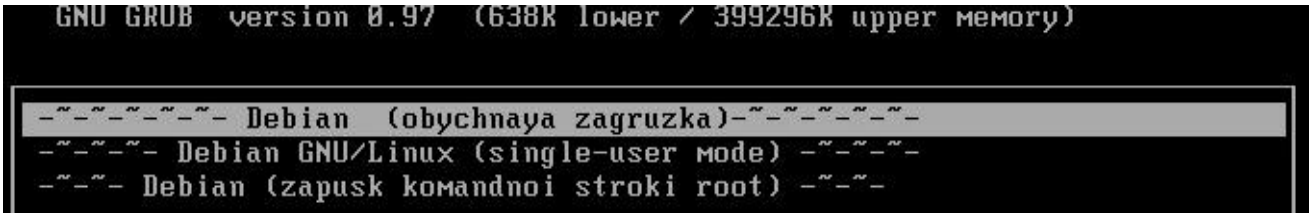

Рисунок 24 – Создание новых пунктов загрузчика

5) Необходимо установить пароль, например, «12345». Для этого в консоли под суперпользователем вводим команду «*grub-md5-crypt*» и компьютер выдает хеш пароля (12345): *grub-md5-crypt*

*Password: ******

*Retype password: ******

$E6eu5HUCvd/0$Qaxm2dFDGgr0 #это и есть полученный хеш «12345».

Далее надо скопировать из строки этот хеш в буфер обмена, открыть, например, с помощью *gedit* файл *menu.lst: gedit /boot/grub/menu.lst* и в нем найти строку «*#password topsecret*», раскомментировать ее и написать:

password topsecret

password --md5 $E6eu5HUCvd/0$Qaxm2dFDGgr0 #из буфера взять хеш.

После сохранения изменений в файле, перезагрузки это дает: редактировать меню *GRUB* нельзя, недоступны ранее «e», «c», «o» и т.д. Доступно лишь «p» (для ввода пароля, который задали ранее «12345») и тогда есть права на редактирование меню загрузчика.

6) Картинку - будущий фон с указанием своей фамилии, необходимо сделать самим в *Linux* (с помощью любого графического редактора), назовем его *1.xpm*. Далее архивируем его *gzip*. При этом картинка должна удовлетворять следующим требованиям: формат файла - *.xpm*; размер - 640x480; цветность - 14 цветов.

По шагам эта процедура выглядит так: например, при работе через *GIMP*, надо открыть меню «Файл» - «Новый» (размеры 640x480 -> «*ок*»), нарисовать картинку. Далее необходимо нажать «*Alt+i*» или «Изображение»->«Режим»->«индексированное», поставить 14 цветов для генерации палитры -> «*ок*», сохранить рисунок как *1.xpm*.

Далее открыть консоль под суперпользователем: «*gzip 1.xpm»,* затем скопировать *1.xpm.gz* в /*boot/grub*: «*cp 1.xpm.gz /boot/grub/1.xpm.gz*»**.**

Открыть файл *menu.lst* на редактирование *«gedit /boot/grub/menu.lst»* и дописать строку вида: «*splashimage=(hd0,1)boot/grub/1.xpm.gz»* как показано для примера на рисунке 25, при этом надо учесть *hdX* – ваш раздел для загрузки ОС. После сохранения и перезагрузки фон изменится (рисунок 26).

```
# Pretty colours
#color cyan/blue white/blue
splashimage=(hd0,0)/boot/grub/1.xpm.gz
```

Рисунок 25 – Изменение *menu.lst*

Часть 2. GRUB2.

Он более гибок чем *Legacy*, в нем появились возможности такие как, например, поддержка ext4; есть продвинутая поддержка таблицы разделов *GPT* и загрузка с *UEFI* (это нужно, если жёсткий диск имеет объём более 2ТБ). Если же вы ставите систему с нуля и у вас есть выбор загрузчика, то стоит отдать предпочтение *GRUB2* как более перспективной и уже стабильно работающей разработке.

Рисунок 26 – Измененный фон загрузчика *Grub Legacy*

Конфигурационный файл находится в */boot/grub/grub.cfg*, но его редактирование особого смысла не имеет. Дело в том, что этот файл создается автоматически утилитой *update-grub* на основе файла настроек */etc/default/grub* и скриптов, находящихся в /*etc/grub.d*/. Таким образом, если вы не хотите, чтобы ваши изменения терялись, например, при обновлении ядра, правильным способом настройки будет редактирование файла настроек, а при отсутствии требуемого параметра в файле настроек, редактирования конфигурационных скриптов. Чтобы изменения вступили в силу необходимо запустить команду после всех внесенных изменений «*update-grub*».

При настройке может понадобится выйти в командную строку *GRUB*. Для этого в меню выбора ОС необходимо нажать «c», для возврата – «*esc*». Командная строка поддерживает автодополнение по «*Tab*».

Чтобы в процессе работы в командной строке получить доступ к файлу, находящемуся не в корневой файловой системе необходимо знать наименование устройств и разделов. Например: (*hd0,*1) - означает первый жесткий диск (устройства нумеруются с 0) и на нем первый раздел (разделы нумеруются с единицы), что соответствует /*dev/sda1*.

Если при наборе команды, например: «*set root=(*» нажать *Tab*, то *GRUB* сам предложит список возможных устройств: *«possible devices are:*

hd0 fd0»

Если нажать «*Tab*» при набранной команде «*set root=(hd0*,», то будет предложен список разделов на первом жестком диске. Ну и для набранной команды «*set root=(hd0,1)/»,* будет предложен список файлов на данном разделе.

Также при настройке *GRUB* может оказаться полезной команда «*grub-mkconfig»*, которая позволяет посмотреть предварительный вариант файла «*grub.cfg*» без изменения реального конфигурационного файла. Собственно, *update-grub* выполняет *grub-mkconfig* с параметром вывода в файл */boot/grub/grub.cfg*. *GRUB* использует *UNICODE*, т.е. создание пункта меню на русском языке проблем вызвать не должно.

Пример настроек. При открытии файла «*/etc/default/grub*» можно увидеть что-то подобное:

```
GRUB_DEFAULT=0
GRUB_TIMEOUT=3
#GRUB_TIMEOUT=0
#GRUB_HIDDEN_TIMEOUT=1
#GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
GRUB_CMDLINE_LINUX=""
GRUB_GFXMODE=1280x768
GRUB_GFXPAYLOAD_LINUX=keep
GRUB_BACKGROUND="/root/picture.jpeg"
```

Первые два параметра отвечают за пункт меню по умолчанию и время ожидания выбора в секундах. Если у вас установлена всего одна ОС, то можно вместо «*GRUB_TIMEOUT=3*» раскомментировать следующие три строчки - в этом случае выбор меню *GRUB2* отображаться не будет, если не нажать «*Esc*»; на это будет столько секунд, сколько задается параметром «*GRUB_HIDEN_TIMEOUT*». После изменения этих настроек надо, как было выше описано, выполнить «*update-grub*».

Если надо изменить имя ОС в меню выбора, можно воспользоваться для этого параметром «*GRUB_DISTRIBUTOR*».

Для управления списком параметров, передаваемых ядру *Linux*, *GRUB2* предоставляет команды «*GRUB_CMDLINE_LINUX»* и «*GRUB_CMDLINE_ LINUX_DEFAULT*».

Последние три параметра отвечают за настройку внешнего вида *GRUB2*. Например, надо поменять разрешение экрана и чтобы узнать текущее разрешение, в командной строке *GRUB2* надо ввести команду «*set*», далее надо посмотреть на параметр «*gfxmode»*. Список поддерживаемых разрешений можно получить командой «*vbeinfo*». Далее надо загрузиться в ОС, найти в конфигурационном файле «*/etc/default/grub»* параметр

«*GRUB_GFXMODE»*, раскомментировать его если он закомментирован и добавить если его нет так: «*GRUB_GFXMODE=1280x768*» или «*GRUB_GFXMODE=1280x768x32»* (цветность можно не указывать). Для того, чтобы разрешение экрана до запуска иксов оставалось таким же, как в *GRUB*, можно установить параметр: «*GRUB_GFXPAYLOAD_LINUX=true*».

Если надо поставить картинку в качестве фона, для начала можно посмотреть, как она будет выглядеть: в консоли *GRUB* сначала подгружаем нужные модули командой: «*insmode jpeg» (*или *png, tga*) и можно устанавливать фон: «*background_image /path/to/picture.jpeg*». Тут надо помнить, что смонтирован только корневой раздел, до остальных можно добраться, указав идентификатор раздела, например, (*hd0,1)/path/to/picture .jpeg.*

Установить картинку в качестве фона можно несколькими способами, далее идет описание двух из них. Самый простой - скопировать картинку в /*boot/grub*/ и не забыть запустить «*update-grub*». Второй способ (он является более приоритетным) - установить параметр: *GRUB_BACKGROUND= "/path/to/picture.jpeg"».*

К сожалению, поменять цвет шрифта в файле настроек нельзя напрямую, поэтому придется открыть настроечные скрипты. Файл «/*etc/grub.d/40_custom*» предназначен для пользовательской настройки. Надо добавить в конец:

set menu_color_normal=red/black
set menu_color_highlight=yellow/black
set color_normal=yellow/black

Примеры измененного интерфейса представлены на рисунках 27-31.

Рисунок 27 – Пример изменения внешнего вида загрузчика

Рисунок 28 – Пример изменения внешнего вида загрузчика

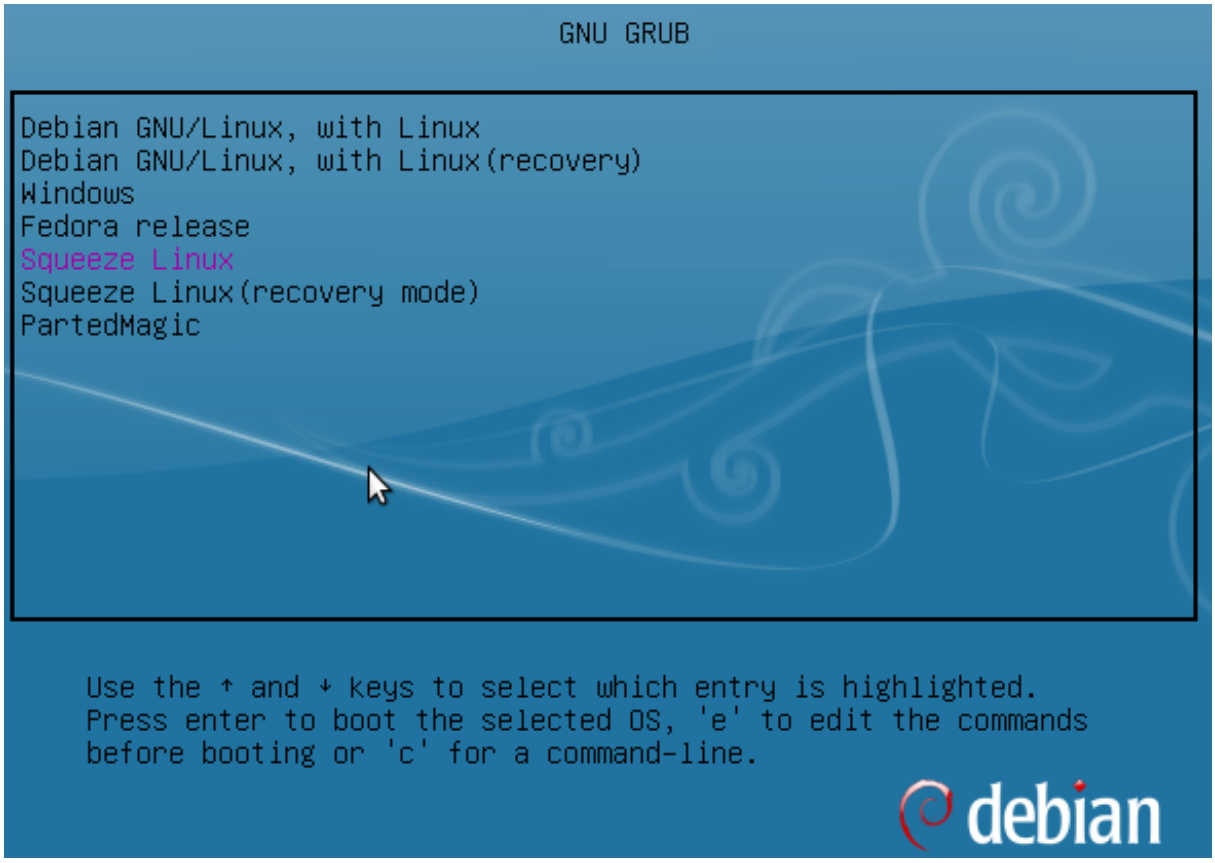

Рисунок 29 – Пример изменения внешнего вида загрузчика

Рисунок 30 – Пример изменения внешнего вида загрузчика

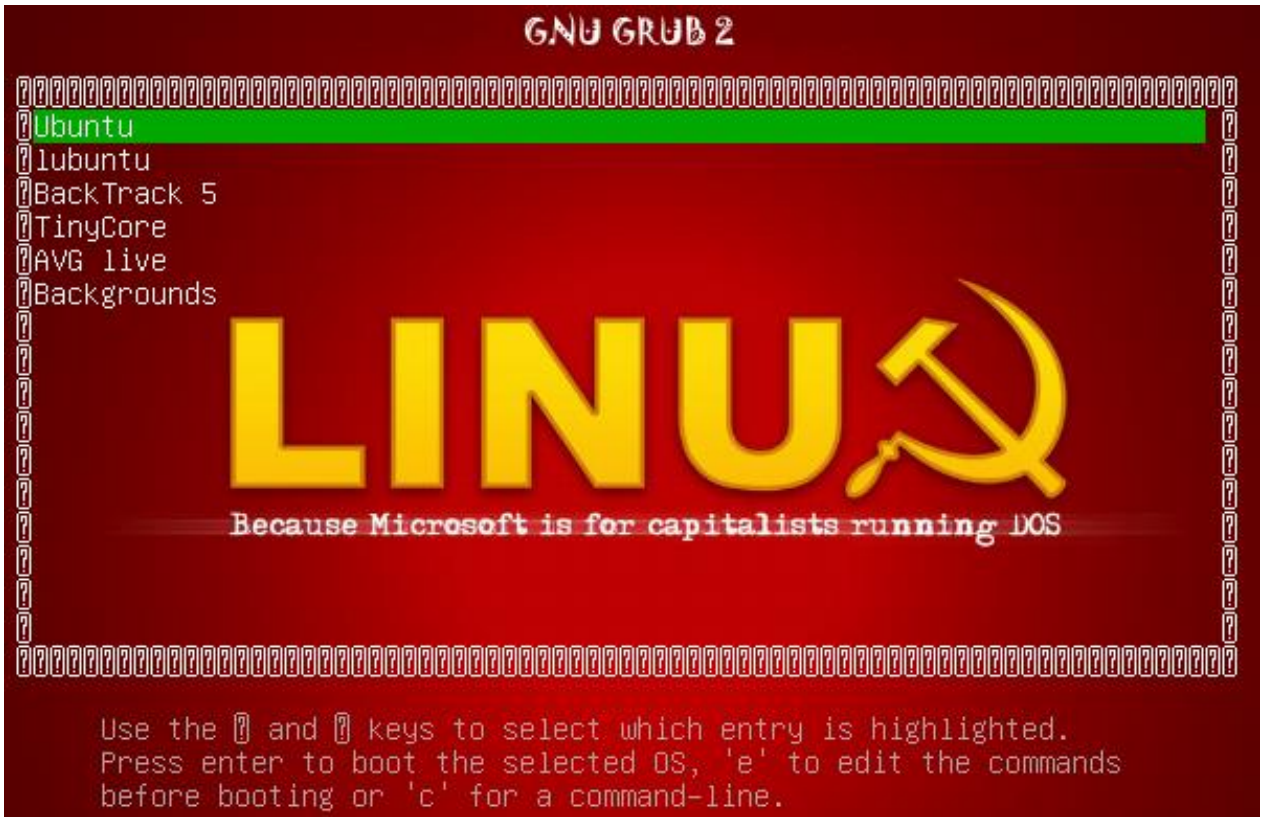

Рисунок 31 – Пример изменения внешнего вида загрузчика

Установить можно 4 параметра: «*color_highlight», «color_normal», «menu_color_highlight», «menu_color_normal»*. Все они должны быть вида color-foreground/color-background, соответственно, цвет текста и цвет фона. Допустимы цвета: *blue, green, cyan, red, magenta, brown, light-gray, dark-gray, light-blue, light-green, light-cyan, light-red, light-magenta, yellow, white,,black.* Если в качестве фона установлен *black*, то фон будет прозрачным. Как и в случае с фоновым изображением, цвета текста можно предварительно опробовать, используя командную строку *GRUB*. Достаточно ввести команду «*set one_of_4_color_parametrs first-color/second-color».*

При редактировании скриптов удобно пользоваться командой «*grub-mkconfig*», чтобы не портить существующий файл конфигурации. При создании конфигурационного файла используются все скрипты из директории *«/etc/grub.d/»,* а запускаются в алфавитном порядке. Там находятся в Debian:

– «*00_header*» отвечает за загрузку основных параметров и библиотек;

– «*05_debian_theme*» отвечает за оформление внешнего вида;

– «*10_linux*» - формирование записей в меню, соответствующих ОС, в которой стоит *GRUB*;

– «*20_linux_xen*» - аналогично предыдущему;

– «*30_os-prober*» обнаружение и включение в список остальных ОС;

– «*40_custom*» и «*41_custom*» - пользовательские скрипты.

Большинство параметров, которыми можно задать настройку *GRUB2*:

GRUB_DEFAULT - какой пункт загружать по умолчанию.

GRUB_SAVEDEFAULT=true в паре с *GRUB_DEFAULT=keep* указывает *GRUB*, что по умолчанию нужно ставить предыдущий выбор.

GRUB_TIMEOUT - время в секундах, в течение которого *GRUB* ждет выбора, прежде чем загрузить значение по умолчанию. Можно установить равным 0, чтобы GRUB не показывал экран выбора ОС, или любое отрицательное значение для отключения таймера.

GRUB_HIDDEN_TIMEOUT указывает сколько секунд ждать перед тем, как показать экран загрузки, если не нажата клавиша. Учитывается, только когда не установлены другие ОС. Если *GRUB_HIDDEN_TIMEOUT_ QUIET=true*, то во время ожидания не показывается таймер.

GRUB_DISTRIBUTOR - название текущего дистрибутива, используется при формировании соответствующей записи в меню выбора ОС.

GRUB_CMDLINE_LINUX, GRUB_CMDLINE_LINUX_DEFAULT - список параметров, используемых при запуске ядра. Параметры *GRUB_CMDLINE_ LINUX* используются как для обычного режима загрузки, так и для recovery. Параметры *GRUB_CMDLINE_LINUX_DEFAULT* используются только обычным режимом. Аналогично используются параметры *GRUB_CMDLINE_ NETBSD, GRUB_CMDLINE_NETBSD_DEFAULT GRUB_CMDLINE_XEN, GRUB_CMDLINE_XEN_DEFAULT* для систем *NetBSD* и *Xen*.

GRUB_DISABLE_RECOVERY=true - тип загрузки recovery не будет создаваться.

GRUB_DISABLE_OS_PROBER=true - убрать проверку наличия других установленных ОС.

GRUB_GFXMODE=1280x768x32 - установить разрешение экрана.

GRUB_GFXPAYLOAD_LINUX=keep - разрешение экрана будет сохранено при передаче загрузки ядру *Linux.*

GRUB_BACKGROUND="/path/to/picture.jpeg" - установить картинку на фон.

GRUB_THEME="path/to/theme.file" - установить файл тем.

Кроме того, *GRUB2* имеет ряд специфических возможностей:

– возможность отключить использование *GRUB'ом* специальной системы идентификаторов для определения корневой файловой системы (*GRUB_DISABLE_LINUX_UUID*);

– возможность исключать поврежденные блоки памяти (*GRUB_BADRAM*);

– возможность загружать дополнительные модули (*GRUB_PRELOAD_MODULES*);

– возможность управлять терминалами ввода и вывода (*GRUB_TERMINAL_INPUT, GRUB_TERMINAL_OUTPUT, GRUB_TERMINAL, GRUB_SERIAL_COMMAND*);

– возможность воспроизводить звуковой сигнал при запуске (*GRUB_INIT_TUNE*).

GRUB предоставляет возможность разграничения прав доступа. Можно создавать обычных пользователей, которые могут загружать разрешенные им ОС, а также суперпользователей, которые могут загружать любую ОС, редактировать существующие записи и вызывать командную строку. Список суперпользователей задается командой: «*set superusers="root"»*. Создать обычного пользователя и задать ему не зашифрованный пароль можно командой: «*password user 123*». Той же командой можно создать пароль суперпользователю. Для создания зашифрованного пароля нужно воспользоваться программой *grub-mkpasswd-pbkdf2*, а в скриптах прописать команду: «*password_pbkdf2 root grub.pbkdf2.sha512.10000.mnogobukv*». Все эти команды лучше всего дописать в скрипт *40_custom*.

Для того, чтобы определенный пользователь мог запускать определенную ОС, он должен быть прописан в списке в соответствующем пункте меню в ключе *--users: «menuentry "Debian" --users "user user1 user2"».*

Восстановление GRUB2.

После установки *Windows* на тот же жесткий диск, на котором стоит *Linux*, можно столкнуться с тем, что при загрузке будет отображаться только меню загрузки *Windows*, что приведет к невозможности загрузить что-либо кроме *Windows*. Такое происходит из-за того, что *Windows* при установке затирает загрузочную область жесткого диска (*MBR*-раздел), удаляя оттуда запись загрузчика *GRUB2*. Для восстановления понадобится *LiveCD* современной ОС, включающей *GRUB2* в свой дистрибутив. Архитектура *LiveCD* должна соответствовать архитектуре восстанавливаемой системы, а

узнать свою архитектуру можно с помощью команды «*arch*» или «*uname -m*»: *i686* соответствует 32-битной архитектуре, *amd64* - 64-битной.

Способ 1. Восстановление *GRUB2* с *LiveCD*.

Надо загрузиться с *LiveCD* и запустить консоль, предварительно узнав на каком диске и на каком разделе установлена нужная восстанавливаемая ОС с помощью команды «*sudo fdisk –l»*.

Для работы понадобится /-раздел, надо примонтировать его командой *sudo mount /dev/sdaX /mnt*, где *X* – номер нужного раздела. Если раздел /*boot* был сделан отдельным разделом, его также надо будет примонтировать: *sudo mount /dev/sdaY /mnt/boot;* также надо смонтировать папку /*dev live*-системы как /*dev* нашего *root*-раздела командой *sudo mount --bind /dev /mnt/dev* и папку /*proc* как /*proc root*-раздела командой: *sudo mount –bind /proc /mnt/proc* и вот теперь надо выполнить команду: *sudo chroot /mnt /bin/bash* – то есть теперь вы *root*-пользователь в системе, корневым разделом которой считается /*mnt.*

Это означает готовность обновления *MBR*-раздела жесткого диска, но для начала нужно определиться с какого физического диска загружается компьютер. Если у вас один жесткий диск, разбитый на разделы - он будет именоваться *sda*, если несколько - первый из них - *sda*, второй - *sdb*, и так далее. Определившись с загрузочным диском (для примера возьмем *sda*), ставим на него *GRUB2*: *grub-install /dev/sda*. Если вы столкнетесь с какими-либо ошибками - попробуйте перезапустить команду с ключом *--recheck*: *grub-install /dev/sda –recheck /dev/sda.* Обратите внимание: мы устанавливаем *GRUB2* на физический диск (*sda, sdb,...*), а не на раздел (*sda3, sdb1,...*). Если все прошло успешно, выходим из *chroot* командой *exit.*

Далее отмонтировать /*dev* нашей *live*-системы: *sudo umount /mnt/dev; /proc live*-системы: *sudo umount /mnt/proc; boot*-раздел, если он монтировался отдельно: *sudo umount /mnt/boot;* и корневой раздел*:sudo umount /mnt.*

Далее надо перезагрузиться и восстановить порядок загрузки в *BIOS* (ставим жесткий диск на первое место) и смотрим результат.

Способ 2. Восстановление *GRUB2* с *LiveCD* (без *chroot*).

Надо загрузиться с выбранного *LiveCD*, запустить консоль, но вместо использования *chroot* надо воспользоваться ключом «*--root-directory»,* но для начала надо убедиться, что эта опция поддерживается *live*-системой: «*grub-install –help»,* если в описании опций присутствует вышеупомянутый ключ «*--root-directory»,* то все в порядке.

Далее надо примонтировать корневой раздел системы, если вы не помните, на каком разделе стоит система, то поможет «*sudo fdisk –l».*

Определившись с корневым разделом, надо смонтировать его (для примера *sda5*): «*sudo mount /dev/sda5 /mnt»*. Если был выделен отдельный *boot*-раздел, надо примонтировать и его (например, это *sda2*): *«sudo mount /dev/sda2 /mnt/boot»*; далее надо смонтировать корневой раздел в /*mnt*, надо выполнить: «*sudo grub-install --root-directory=/mnt /dev/sda*» (как и в предыдущем способе *GRUB2* устанавливается на физический диск, а не на раздел). В качестве диска нужно указать тот диск, который установлен

загрузочным в *BIOS*. Если все пройдет успешно, установщик выведет сообщение об успешном завершении и список обнаруженных жестких дисков, которые были добавлены в "*device.map*".

Если все прошло успешно, то надо отмонтировать диски и перезагрузиться. Если будет выведен неполный список дисков – надо отредактировать файл "*device.map*" в корневой папке установленной системы (в описанном примере – «*/mnt/boot/grub/device.map»*) и поправить его, добавив остальные диски и поправив нумерацию. Файл должен иметь вид:

(hd0) /dev/sda (hd1) /dev/sdb и так далее для всех жестких дисков. Надо сохранить файл и повторно выполнить команду «*grub-install*», как описанно выше и теперь должен отобразиться правильный список дисков.

Надо отмонтировать диски и перезагрузиться.

GRUB-customizer – программное обеспечение, которое в графическом режиме помогает настраивать внешний вид, содержимое, поведение загрузчика. Примеры интерфейса программы и результат настройки показаны на рисунках 32-34.

Загрузчиком, набирающим популярность, сейчас является загрузчик *BURG*, что означает «*Brand-new Universal loadeR from GRUB*», который основан на загрузчике *GRUB* (интерфейс *BURG* и результаты его настройки показан на рисунках 35-37).

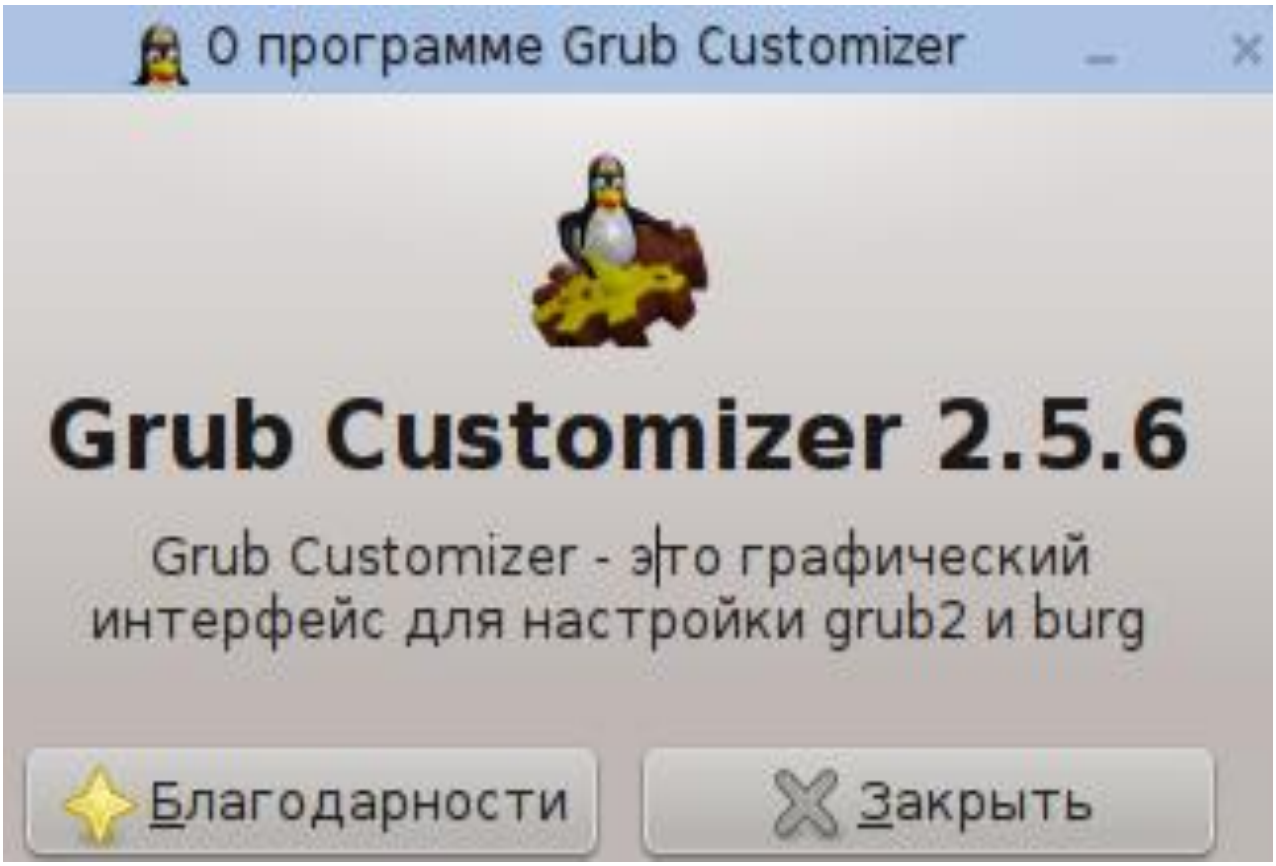

Рисунок 32 - Интерфейс *GRUB-customizer*

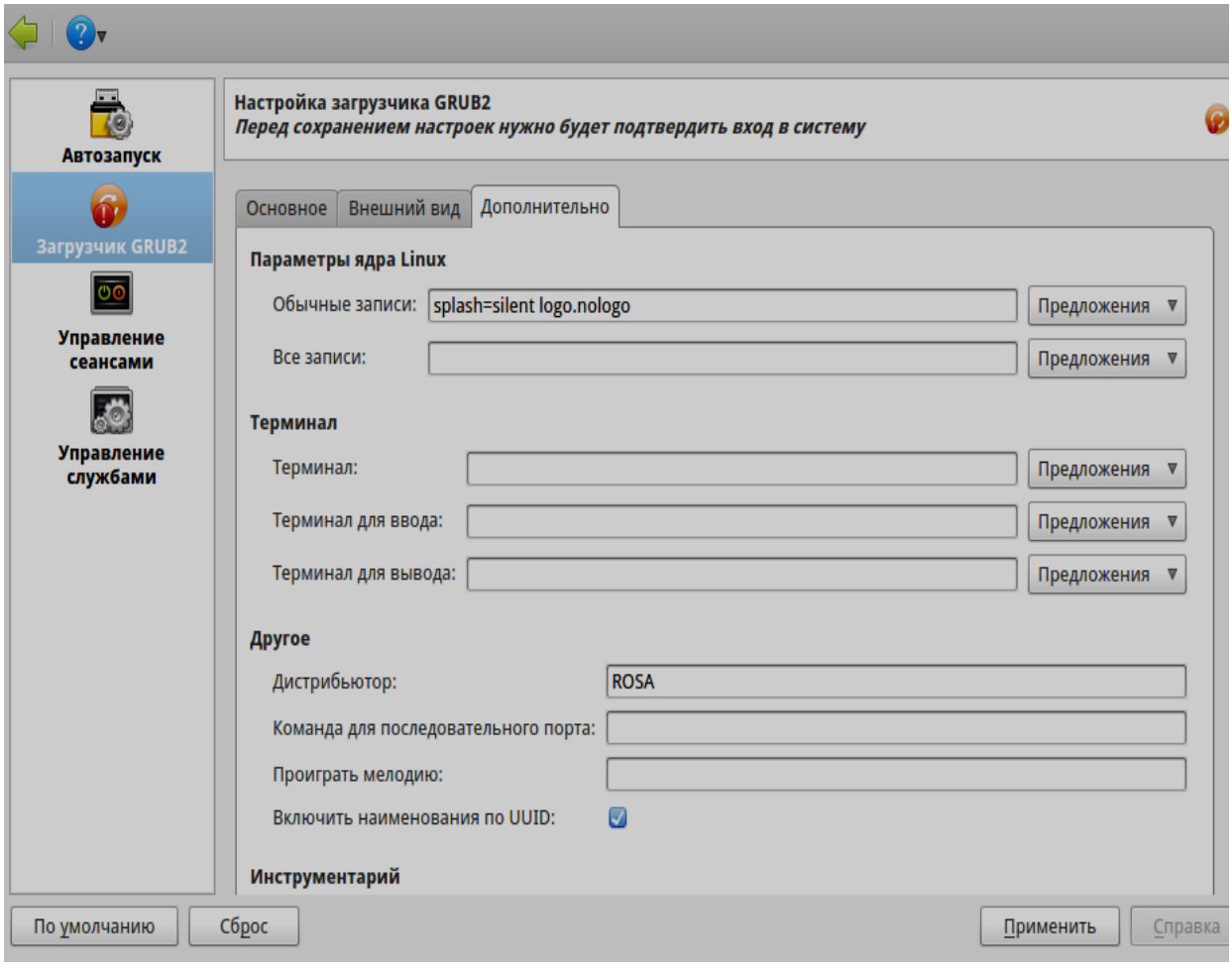

Рисунок 33 - Интерфейс *GRUB-customizer*

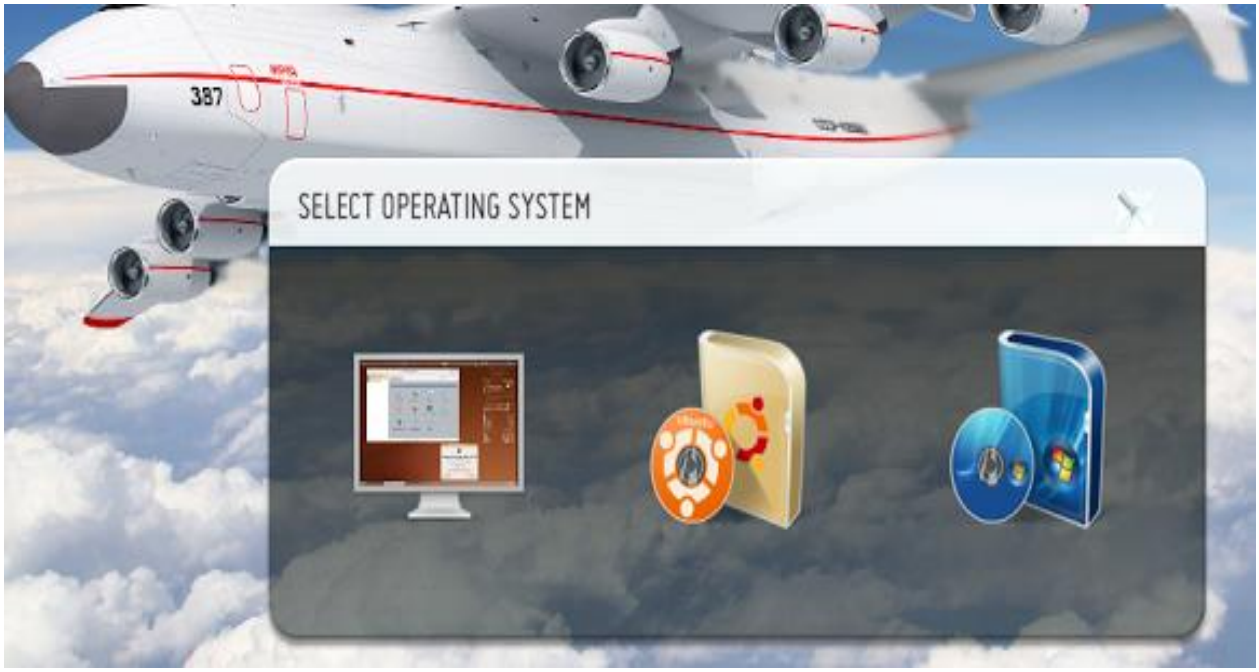

Рисунок 34 – Результат настройки через *GRUB-customizer*

Рисунок 35 – Интерфейс *BURG*

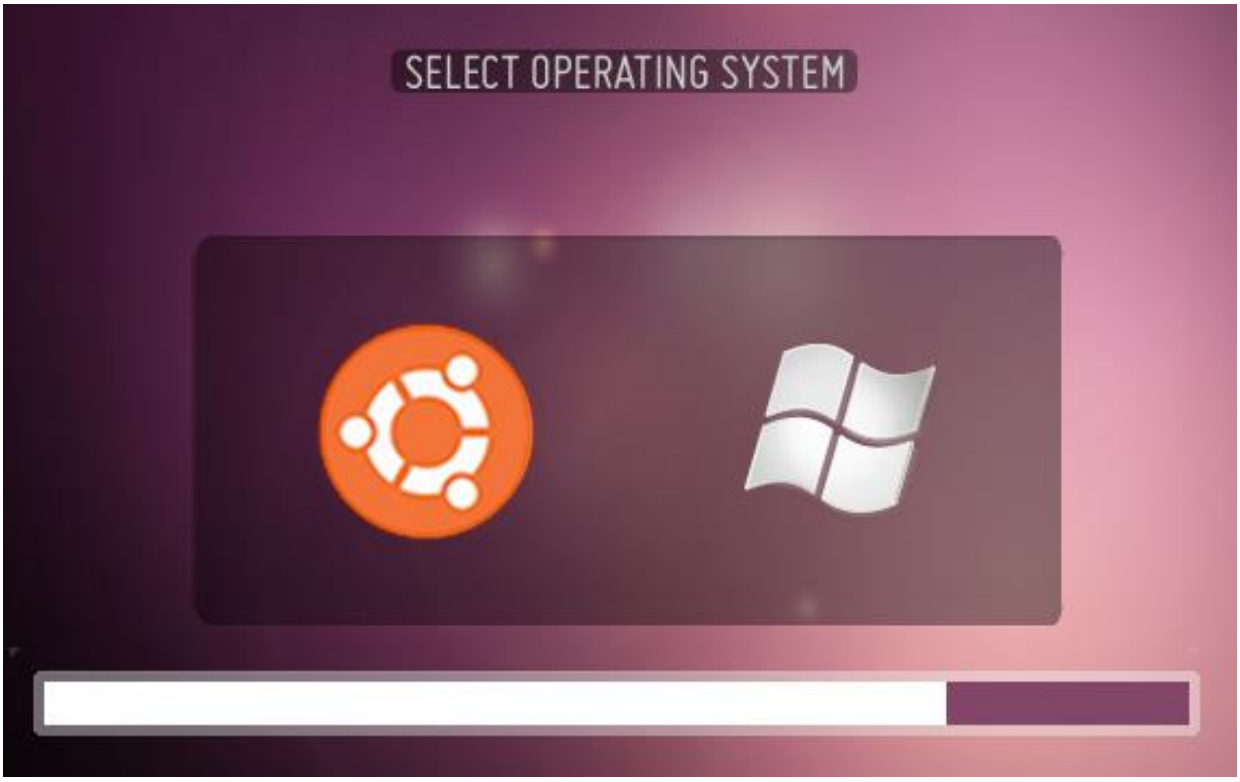

Рисунок 36 – Результат настройки через *BURG*

Рисунок 37 – Результат настройки *BURG*

10 Шифрование и файервол

С появлением в 2.6-х ядрах низкоуровневого драйвера логических томов - *Device mapper* стала возможной работа не только с петлевыми устройствами, но и непосредственно с дисковыми разделами. Более того, благодаря модульной структуре *Device mapper*, ничто не мешает, комбинируя различные цели, организовать шифрование данных на *RAID*-массиве или создать зашифрованный своп. Цель для *Device mapper*, отвечающая за шифрование данных, называется *dm-crypt*.

Результат шифрования файла с помощью встроенной утилиты, а именно: *OpenSSL* представлен на рисунке 38. Команда в консоли, использованная при шифровании файла *file1-Zuyeva.txt*, находящегося в */home/skazka/Crypt* будет следующей:

openssl enc -e -aes-256-cbc -in /home/skazka/Crypt/file1-Zuyeva.txt -out file3-Zuyeva.txt.encoded

После этого пароль вводится 2 раза для шифрования, опция *-enc* значит, что шифрование идет с помощью алгоритма (*AES256); -e* указывает на процедуру шифрования; *-in* указывает входящий файл; *-out* - выходящий файл.

Для дешифрования команда (параметр *-d* указывает на дешифровку):

openssl enc -d -aes-256-cbc -in /home/skazka/Crypt/file3-Zuyeva.txt.encoded -out file5-Zuyeva.txt.decrypt

Рисунок 38 – Результат шифрования *file1-Zuyeva.txt* с помощью *ssh*

Межсетевой экран, файрвол или брандмауэр - комплекс аппаратных и программных средств в компьютерной сети, осуществляющий контроль и фильтрацию проходящих через него сетевых пакетов в соответствии с заданными правилами.

Во все ядра *Linux*, начиная с 2.0, встроено средство для фильтрации сетевых пакетов. В 2.0 это *ipfwadm*, в 2.2 - *ipchains*, а в ядрах, начиная с 2.4 - *iptables*.

Фаервол, встроенный в ядро *Linux*, называется *Netfilter*, а *iptables* - утилита для управления этим фаерволом. Многие ошибочно полагают, что фаервол называется *iptables*. Это не так.

Принцип фильтрации такой: когда через ядро проходит пакет, он проверяется на совпадение с одним или несколькими правилами. При этом в зависимости от этих правил он может быть пропущен (*ACCEPT*), отброшен (*DROP*) или отклонен (*REJECT*). Кроме того, он может быть отправлен на проверку в следующую цепочку правил. Здесь же можно указать, что факт прохождения пакета, подходящего под определенное правило, должен быть отмечен в *syslog*.

Правила могут включать в себя проверку адреса. Для изменения используемого набора правил используется программа *iptables*. Все правила хранятся в памяти ядра и при перезагрузке сбрасываются. Поэтому необходимо создать файл конфигурации, из которого правила фильтрации будут считываться при загрузке машины. Обычно это */etc/rc.d/rc.firewall*. Это обычный скрипт оболочки, который вызывает */sbin/iptables* с определенными параметрами, соответствующими составленным правилам. Поэтому в большинстве дистрибутивов для изменения конфигурации *iptables* необходимо отредактировать указанный файл и запустить его (этот файл, как правило, автоматически выполняется при загрузке машины). После чего можно посмотреть обновленную таблицу правил командой *iptables -L*.

Пример настройки правил файервола на машине:

```
[root@Linux_router root]# iptables -A FORWARD -p tcp --dport 80 -s 192.168.30.0
/24 -d 192.168.10.100 -j ACCEPT
[root@Linux_router root]# iptables -A FORWARD -s 192.168.30.0/24 -j DROP
[root@Linux_router root]# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  192.168.30.0/24      192.168.10.100     tcp dpt:80
DROP       all  --  192.168.30.0/24      0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

11 Режимы загрузки, взлом ОС, деактивация пользователей

Для запуска ОС в минимальном окружении: надо выполнить следующие действия:

1) Перезагрузить систему и войти в меню загрузчика *GRUB2*.

2) Изменить параметры запуска ядра, дописав параметр *init=/bin/bash* в конец строки параметров ядра (*kernel*) (рисунок 39). Выполнить загрузку ядра с данным параметром и дождаться приглашения терминала.

3) Попробовать создать файл «*proba.txt*» в корневом каталоге: *«touch /proba.txt»*. Ответом будет комментарий об ошибке о том, что корневая файловая система доступна только для чтения (рисунок 40).

4) Перемонтировать корневой раздел в режиме чтения-записи: *«mount -o remount, rw /»*, где *remount, rw* - опции перемонтирования в режим чтения-записи (только для чтения будет *ro - read-only*). А слеш - корневой раздел (рисунок 41).

5) Попробовать создать файл «*proba.txt*» в корневом каталоге: *«touch proba.txt».* Файл успешно создан, рисунок 42.

6) Выполнить команду «*exec /sbin/init*» для начала нормальной загрузки ОС. Загрузка системы в минимальном окружении может быть полезна в различных случаях, особенно, если происходит восстановление удаленных случайно данных, когда любое воздействие на диск (запись) нежелательны. При желании можно перемонтировать корневой раздел в режиме чтения-записи и продолжить реанимацию системы. Как только операции по восстановлению-диагностике будут сделаны, можно продолжить нормальную загрузку системы.

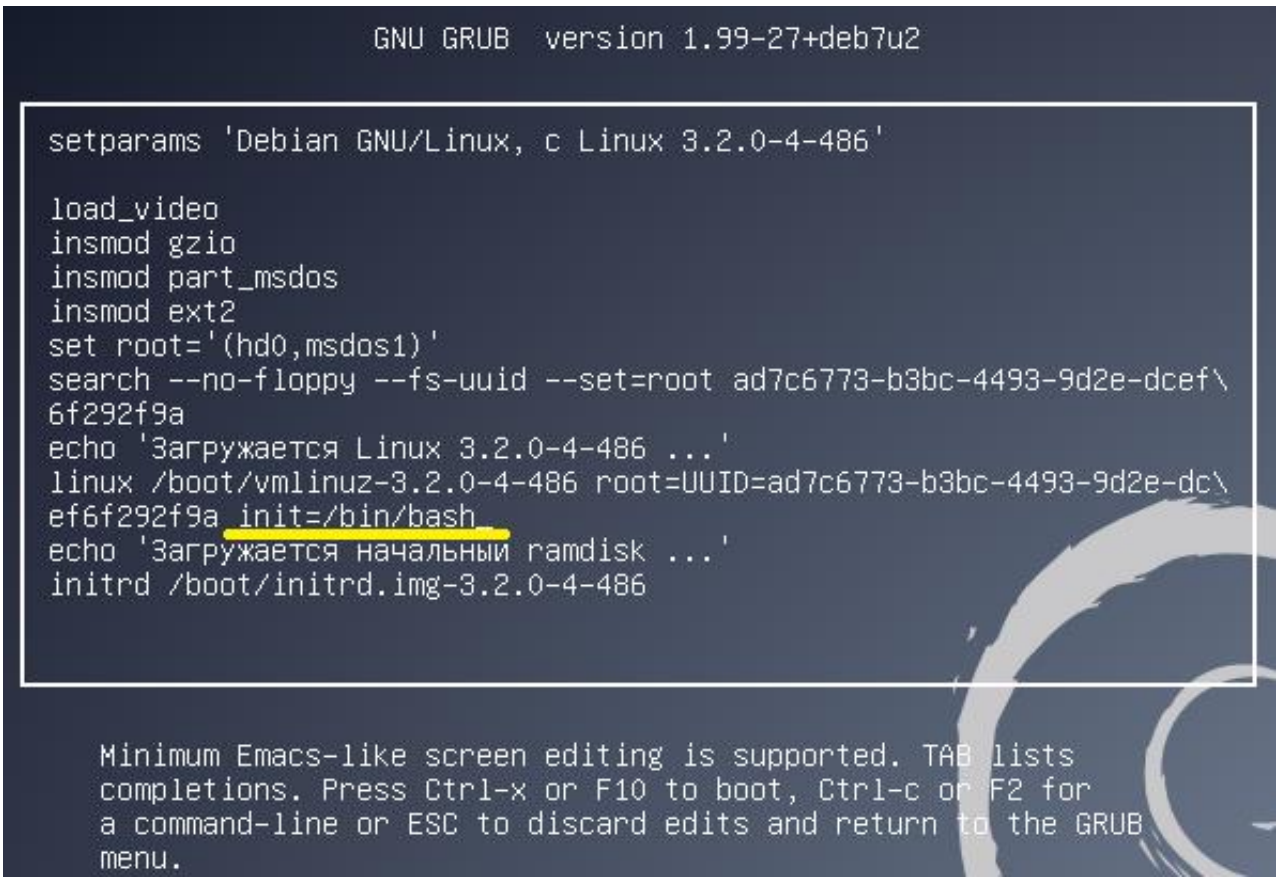

Рисунок 39 – Изменения режима загрузки текущей записи

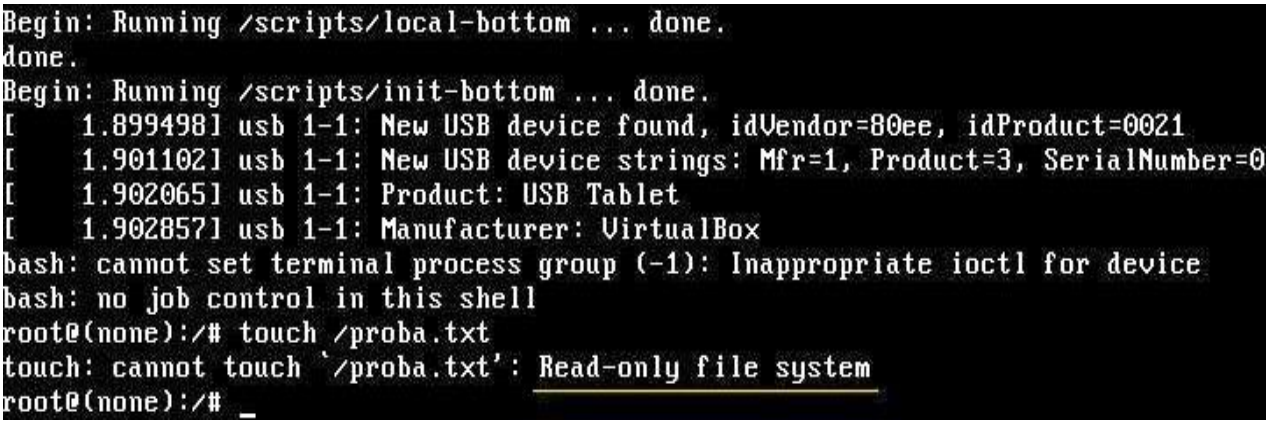

Рисунок 40 – Сообщение об ошибке

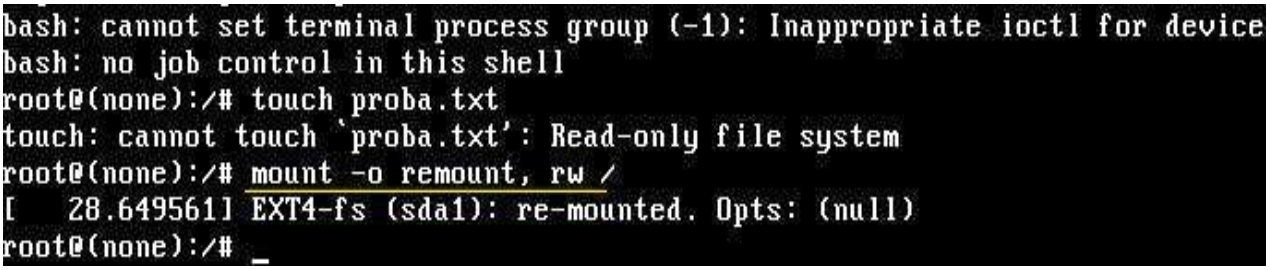

Рисунок 41 – Команда перемонтирования

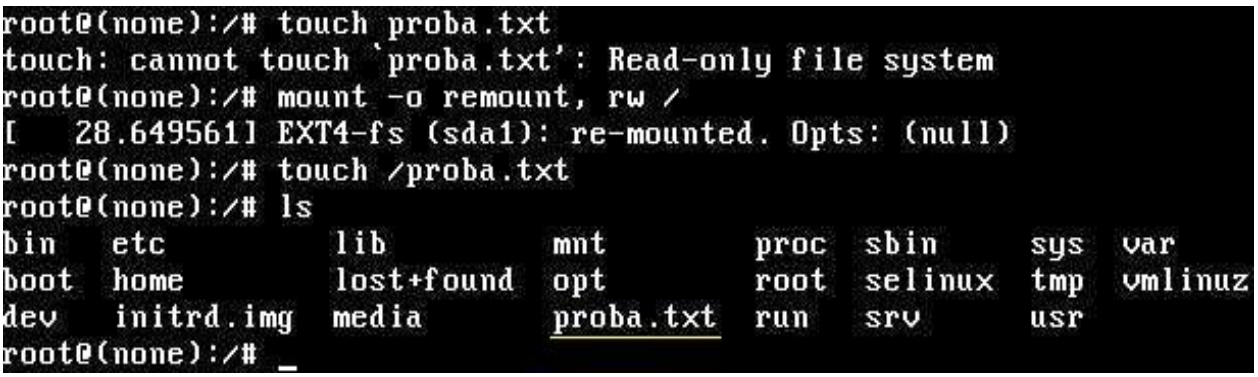

Рисунок 42 - Создание файла *proba.txt* в ОС

Деактивация пользователей.

2. Способ 1. Редактирование файла */etc/shadow*. В данном файле хранятся зашифрованные пароли из /*etc/passwd*. К примеру, вот так выглядит зашифрованный пароль в файле *shadow*:

user1:6KYriHdKR$Yku3LWgJmomsynpcle9BCA:15711:0:4444:5:::

для отключения аккаунта надо перед зашифрованным паролем добавить символы ! или *:

user:!6KYriHdKR$Yku3LWgJmomsynpcle9BCA:15711:0:4444:5:::

или:

user:6KYriHdKR$Yku3LWgJmomsynpcle9BCA:15711:0:4444:5:::*

После чего пользователь *user1* не сможет себя авторизовать, так как системе не удастся расшифровать пароль. Чтобы вернуть, надо удалить из файла /*etc/shadow* символы ! и *, которые добавили раньше.

Способ 2. Оболочка *nologin.* Существует еще способ деактивации аккаунта пользователя - это пользовательская оболочка *nologin*, которая находится по адресу: /*usr/sbin/nologin*. В файле *passwd* изменить по примеру:

user2:x:1001:1001:Test,User,,:/home/user:/bin/bash

на

user2:x:1001:1001:Test,User,,:/home/user:/usr/sbin/nologin

После чего пользователь *user2* не сможет авторизоваться, несмотря на ввод правильного пароля.

Способ 3. То же самое можно сделать с помощью команды: «*usermod -L user3*». Любой метод авторизации, использующий для аутентификации пользователя файл /*etc/shadow*, больше не будет работать, так как расшифровать пароль будет невозможно:

$ su user3

Password:

su: Authentication failure

Для активации аккаунта можно ввести команду: «*usermod -U user3*».

Взлом.

Чтобы осуществить взлом с помощью *LiveCD* необходимо загрузиться с *Live*-диска, подмонтировать корневой раздел, отредактировать /*etc/shadow*, который состоит из строк, каждая из которых имеет 9 полей, разделенных

двоеточиями. В первом поле хранится имя пользователя, во втором - хеш пароля. Надо удалить содержимое второго поля нужной строки, сохранить файл, отмонтировать файловую систему, перезагрузиться, и загрузившись с жесткого диска, войти в систему как *root*, без пароля.

12 Архивация

Архиватор нужен для уменьшения размеров файлов, упаковывания каталогов в единые объекты для более быстрой передачи по каналам связи и компактного хранения.

В *Linux* есть три различных метода упаковки: *tar, gzip* и *bzip2* Команды для работы:

tar cf 2.tar 1.txt - создать *tar*-архив с именем 2.*tar*, содержащий 1.*txt*;

tar czf 2.tar.gz 1.txt - создать *tar*-архив с сжатием *Gzip* по имени 2.*tar.gz*;

tar cjf 2.tar.bz21.txt - создать *tar*-архив с сжатием *Bzip2* по имени 2.*tar.bz*;

tar xf 2.tar - распаковать архив 2.*tar* в текущую папку;

tar xzf 2.tar.gz - распаковать *tar*-архив с *Gzip*;

tar xjf 2.tar.bz - распаковать *tar*-архив с *Bzip2*.

tar --exclude='/home/user/' -cvzf archive.tar.gz /home/** - архивировать всю директорию /*home*, но исключить из архива папку /*home/user*.

После создания и просмотра содержимого файлов посмотрим содержимое каталога (рисунок 43).

```
root@debian:/home/skazka# ((uname -a)>5-Zyueva.txt) & ((id root)>6-Zyueva.txt)
[4] 3547
root@debian:/home/skazka# cat 5-Zyueva.txt & cat 6-Zyueva.txt
[5] 3549
uid=0(root) gid=0(root) группы=0(root)
[4]   Done                    ( ( uname -a ) > 5-Zyueva.txt )
root@debian:/home/skazka# Linux debian 3.2.0-4-486 #1 Debian 3.2.57-3 i686 GNU/
root@debian:/home/skazka# ls -l
итого 8
-rw-r--r-- 1 root root 59 Июл 27 19:24 5-Zyueva.txt
-rw-r--r-- 1 root root 45 Июл 27 19:24 6-Zyueva.txt
root@debian:/home/skazka#
```

Рисунок 43 – Создание и просмотр содержимого файлов

Команда «*xxd -l 100 5-Zyueva.txt*» выводит первые 100 байт файла. Чтобы упаковать все это в архив 7.*tar* надо воспользоваться одной из команд выше. Далее посмотреть размер архива: *«ls -lh 7.tar»*, при этом получается, что исходные файлы были даже меньше по размеру, а в архиве стали весить больше так как сам по себе *tar* не сжимает файлы, а просто упаковывает их в своего рода контейнер, и уже этот контейнер можно «скормить» утилитам-упаковщикам *gzip* или *bzip2* (их можно использовать в *tar*, с помощью ключей *z* и *j* соответственно).

Сожмём теперь этот файл при помощи *gzip: «gzip 7.tar*». Нетрудно заметить, что к расширению файла теперь припишется *gz* автоматически и размер файла уменьшился (большой уровень сжатия).

Распаковка из резервной копии ведется похоже. Есть тут два пути:

а) последовательный: распакуем архив *.gz*: «*gzip -d 7.tar.gz*», ключ «*-d*» означает «*decompress*», в папке появился файл 7.*tar* прежнего размера. Далее: «*tar -xvf 7.tar*» - распакует содержимое этого контейнера.

Ввиду большой популярности архиваторов *gzip* (и *bzip2*) их поддержка уже включена во многие утилиты. В частности в *tar*. Для *gzip* прибавляется опция *z*, для *bzip2 – j*;

б) можно обойтись одной командой: «*tar -xvzf ~/backup.tar.gz*» и декомпрессировать и разжать сразу.

Если архиватор не установлен: «*apt-get install rar*». Для архивации всех файлов и папок в директории /home/skazka/lab9/z1-5 и директории /home/skazka/lab9/z1-3 надо использовать «*rar a -r -m5 myarchive.rar /home/skazka/lab9/z1-5/* /home/skazka/lab9/z1-3/*»,* где *«*-a» - добавить данные (*add*); «-m5» - степень сжатия от 0 до 5. 0 - без сжатия; *myarchive.rar* - имя архива; остальное - архивируемые директории через пробел; «*-r*» - опция, которая добавляет информацию для восстановления. Опция «*-hp*» позволяет задать пароль на архив.

Распаковать архив: «*rar e myarchive.rar*» в текущую директорию, проверить архив*: «rar t myarchive.rar»*; восстановить *«rar myarchive.rar».*

13 Восстановление

Представим, что нам надо сделать резервную копию системы, показать пример удачного восстановления из образа системы.

Резервное копирование системы (*Backup*) является одной из важных профилактических мер по поддержание стабильности работы системы. Для резервного копирования понадобится утилита по работе с архивами – *tar*.

Сделать резервную копию работающей системы: «*su -l root*». Надо посмотреть сколько системой использовано, сколько места свободно (*backup* надо сжать в архив, так-что размер будет меньше, чем текущее использование системой): выполнение команды «*root@debian:~# df -h*» выдает такую информацию:

ФС	Размер	Исп-но	Дост	Испол-%	Смонтировано в
/dev/sda2	73G	2,1G	67G	3%	/
tmpfs	5,0M	0	5,0M	0%	/lib/init/rw
tmpfs	152M	1,4M	151M	1%	/run
udev	753M	0	753M	0%	/dev
tmpfs	303M	0	303M	0%	/run/shm
/dev/sdb1	147G	26G	114G	19%	/Datas

В рассматриваемом примере вся система установлена на раздел /*dev/sda2* и занимает 2.1G, в корень этого раздела и надо копировать дамп, т.к. доступно ещё 67G.

Надо перейти в корень системы «*cd /*». Будет произведено копирование работающей системы, поэтому надо исключить из копирования разделы /*proc, /lost+found, /sys* и сам архив */backup.tgz* и для данного примера исключить раздел /*Datas*. Для чистоты рекомендуется почистить логи в */var/log* и удалить кеш архивов *apt-get clean*. Команда: «*tar cvpzf backup.tgz -exclude=/proc -exclude=/lost+found -exclude=/backup.tgz -exclude=/mnt -exclude=/sys -exclude=/Datas /».*

Далее при просмотре атрибутов «*ls –alh*» выводится информация:

- rw- r-- r-- 1 *root root* 607M Июл 28 13:28 *backup.tgz*

А затем надо поместить в надежное место архив *backup.tgz* на случай сбоев работы, чтобы с лёгкостью восстановить систему в короткие сроки.

Восстановление системы из созданного бэкапа: если делать восстановление системы на то же оборудование, с теми же разделами жестких дисков, то процесс займёт буквально несколько минут. Шаги:

1) Загрузиться с *LiveCD,* скопировать *backup* системы в корень.
2) Распаковать архив в корень раздела: «*tar xvpfz backup.tgz* /».
3) Прописать загрузочную область (если делали разметку утилитой *GParted*, то в начале диска обязательно надо оставить несколько не задействованных мегабайт, иначе *GRUB2* не установится): «*grub-install -root-directory=/mnt/ /dev/sda2»,* (*-root-directory=/mnt*/ в данном случае указывает, что для корня считать точку /*mnt*, т.к. туда у нас временно смонтирован раздел *sda2*).
4) Создать пустые каталоги */proc /sys*. Перезагрузиться и надо отслеживать логи, которые выводит система при загрузке и соответственно реагировать.

Если же ваша система переносится на другое устройство, то здесь будет немного сложнее:

1) Распаковать архив.
2) Посмотреть разделы жёстких дисков, определившиеся в *LiveCD.*
3) Перезарузка, в меню загрузчика редактируем конфигурацию загрузки в соответствии с менами разделов жёстких дисков.
4) Если система не может запуститься из-за отсутствия файловой системы, значит нужно пересобирать *initrd*-загрузчик (это загрузчик, который определяет всё оборудование, а потом далее передаёт ядру ОС управление и дальнейшую загрузку системы) с необходимыми модулями; выполнять это можно загрузившись с *LiveCD*, примонтировав разделы /*proc* и /**sys** к системе, в которую надо компилировать */mnt/proc /mnt/sys*, а далее авторизоваться, как будто бы работает система *chroot /mnt.*

Описанный метод - самый простой способ для резервного копирования рабочей системы.

14 Алиасы

Алиасы - это очень удобный инструмент для тех, кто часто работает в командной строке и хочет сократить написание команд до минимума. При правильных установках она поможет быстрее закончить писать стандартные конструкции, то есть переназвать что-то длинное более кратким.

Команда *alias* позволяет пользователю создавать простые псевдонимы для команд любой сложности (вместе с опциями, аргументами, перенаправлениями и программными каналами). Псевдонимами заменяются команды или группы команд, которые долго или неудобно набирать на клавиатуре, их применение ускоряет и упрощает работу в командной строке.

Но особенность такова, что если алиасы были заданы через командную строку, то они действительны для текущей сессии текущего пользователя, после выхода или перезагрузки они исчезнут.

Для того, чтобы они были действительны навсегда, рекомендуется записать их в файл *bashrc* (для удобства их можно записать в конец файла).

Универсальным является метод с использованием файла *bashrc*. Сначала нужно проверить наличие файла *bashrc* в системе: «*locate bashrc*». В зависимости от наличия файлов типа *bashrc* (*.bashrc, bash.bashrc* и т.п.) в различных директориях, возможно несколько вариантов:

а) чтобы создать постоянные псевдонимы для данного пользователя, если в домашней директории есть файл *.bashrc* (скрытый), то нужно просто вписать в конец этого файла нужные псевдонимы по одному на строку. Например:

alias c='clear'

alias grep='grep --color'

и так далее, а если в домашней директории не имеется файла *.bashrc*, то нужно создать текстовый файл вписать туда нужные псевдонимы как показано выше.

Заработают созданные псевдонимы после перезагрузки;

б) чтобы создать постоянные псевдонимы для пользователя *root*: если существует файл */root/.bashrc* (скрытый), то вписать нужные псевдонимы в этот файл. Если такового файла нет, то следует создать его и вписать псевдонимы. А вписав новые псевдонимы, не забудьте перезагрузиться.

Заработают созданные псевдонимы после перезагрузки;

в) */etc/bash.bashrc* (если используется оболочка *bash*). Все то же самое как и описано выше.

Команда *alias* необычна тем, что имеет всего одну опцию *-p,* чтобы «одним махом» создать псевдоним и просмотреть список уже созданных. Например: «*alias -p p='pwd'*» показывает созданные алиасы до этого момента и создает при этом следующий (не показывая его в общем списке).

Чтобы изменить поведение команды по умолчанию, например, введя псевдоним: «*alias ls='ls –a'*» можно быть уверенным, что команда *ls* покажет также скрытые файлы, чего по умолчанию она не делает.

Если взять команду *df*, которая выводит информацию обо всех разделах, смонтированных в системе, то можно увидеть, что информация не совсем доходчива, так как единицей размера раздела по умолчанию выбран однокилобайтный блок (это тянется с тех дней, когда килобайт считался большим количеством). Существует опция *-h*, которая использует в качестве единиц размера мегабайты и гигабайты, это намного нагляднее, поэтому имеет смысл ввести псевдоним «*alias df='df –h'*».

Полезным также будет псевдоним «*alias grep='grep –color'*», который сделает вывод команды grep цветным (рисунок 44).

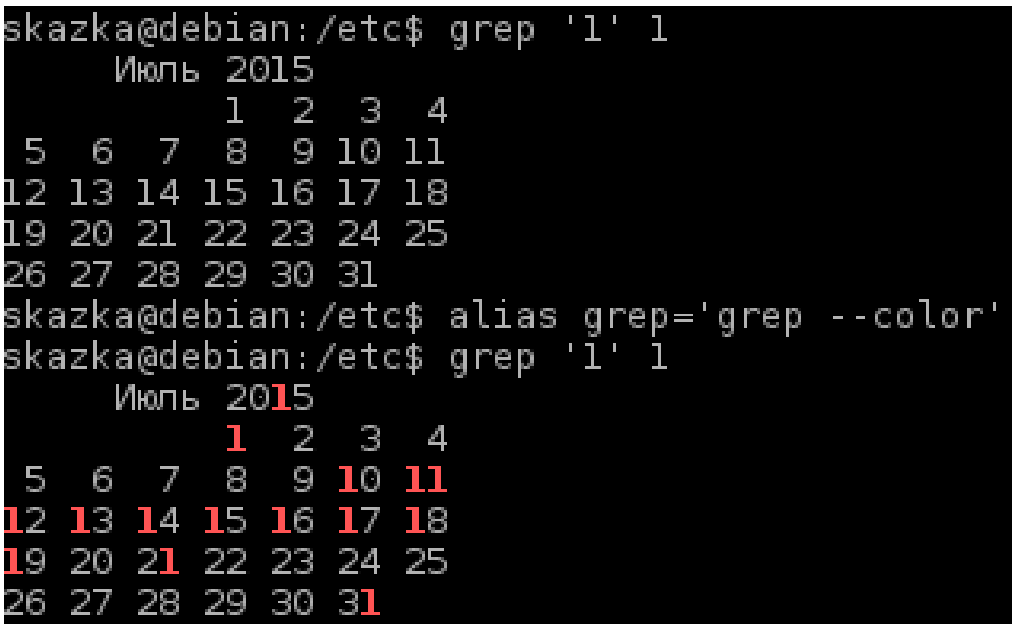

Рисунок 44 – Изменение параметров вывода команды через алиас

Чтобы избежать последствий неправильного набора команд, например, если пользователь постоянно ошибается и постоянно печатает *pdw* вместо *pwd*, он может создать псевдоним: «*alias pdw='pwd'*» и больше не задумываться о том, правильно ли он ввел команду.

Чтобы повысить безопасность системы, сделав некоторые «опасные» команды интерактивными: это заставит пользователя подтверждать свои действия. Например, команда *rm* удаляет файлы и директории без возможности восстановления, поэтому имеет смысл создать для нее псевдоним: «*alias rm='rm –i'*». В интерактивном варианте команда не столь опасна. Или взять команду *cp*, копирующую содержимое одного файла в другой. Если по ошибке указать в качестве аргумента существующий файл, то команда сотрет его содержимое и перезапишет новым. Избежать этого поможет псевдоним: «*alias cp='cp -i'*», который заставит подтвердить операцию копирования, снизив тем самым риск ошибки.

Удаление псевдонимов: «unalias имя_псевдонима», эта команда удаляет не только созданные для текущей сессии псевдонимы, но и постоянные, прописанные в конфигурационном файле. Опция «*-a*» позволяет удалить все псевдонимы для данного пользователя и данной сессии: «*unalias -a*».

15 Реализация сценариев

Скрипт - это программа или программный файл сценария, который автоматизируют некоторую задачу, которую пользователь делал бы вручную, используя интерфейс программы.

BASH - Bourne-Again SHell (Снова шелл Борна (создателя *sh*)), самый популярный командный интерпретатор в *GNU/Linux*.

Список некоторых встроенных команд, которые минимально необходимы начинающему программировать пользователю:

- *break* - выход из цикла;
- *for, while* или *until continue* - выполнение следующей итерации цикла;
- *for, while* или *until echo* - вывод аргументов, разделенных пробелами, на стандартное устройство вывода;
- *exit* - выход из оболочки;
- *export* - отмечает аргументы как переменные для передачи в дочерние процессы в среде;
- *hash* - запоминает полные имена путей команд, указанных в качестве аргументов, чтобы не искать их при следующем обращении;
- *kill* - посылает сигнал завершения процессу;
- *read* - читает строку из ввода оболочки и использует ее для присвоения значений указанным переменным;
- *return* - заставляет функцию оболочки выйти с указанным значением;
- *shift* - перемещает позиционные параметры налево;
- *test* - вычисляет условное выражение;
- *times* - выводит имя пользователя и системное время, использованное оболочкой и ее потомками;
- *trap* - указывает команды, которые должны выполняться при получении оболочкой сигнала;
- *unset* - вызывает уничтожение переменных оболочки;
- *wait* - ждет выхода из дочернего процесса и сообщает выходное состояние.
- Помимо этого, зарезервированные переменные:
- $*DIRSTACK* - содержимое вершины стека каталогов;
- $*EDITOR* - текстовый редактор по умолчанию;
- $*EUID* - эффективный *UID*. Если вы использовали программу su для выполнения команд от другого пользователя, то эта переменная содержит *UID* этого пользователя
- $*UID* - содержит реальный идентификатор, который устанавливается только при логине;
- $*FUNCNAME* - имя текущей функции в скрипте;
- $*GROUPS* - массив групп, к которым принадлежит текущий пользователь;

– $*HOME* - домашний каталог пользователя;
– $*HOSTNAME* – имя хост-машины;
– $*HOSTTYPE* - архитектура хост-машины;
– $*LC_CTYPE* - внутренняя переменная, которая определяет кодировку символов;
– $*OLDPWD* - прежний рабочий каталог;
– $*OSTYPE* - тип ОС;
– $*PATH* - путь поиска программ;
– $*PPID* - идентификатор родительского процесса;
– $*SECONDS* - время работы скрипта в секундах;
– $# - общее количество параметров, переданных скрипту;
– $* - все аргументы, переданные скрипту, выводятся в строку;
– $@ - тоже самое, что и предыдущий, но параметры выводятся в столбик;
– $! - *PID* последнего запущенного в фоне процесса;
– $$ - *PID* самого скрипта.

Условные операторы.

Структура *if-then-else* используется следующим образом:

if <команда или набор команд возвращающих код возврата (0 или 1)>
then
<если выражение после *if* истино, то выполняется этот блок>
else
<если выражение после *if* ложно, тот этот>

В качестве команд, возвращающих код возврата могут выступать структуры [[, [, *test*, (( )) или любая другая (или несколько) *Linux*-команда.:

test - для логического сравнения, после выражения необходима закрывающая скобка "]";

[- синоним команды *test;*

[[- расширенная версия "[" (начиная с версии 2.02), внутри которой могут быть использованы || (или), & (и). Долна иметь закрывающую скобку "]]";

(( )) - математическое сравнение.

Для краткости и читаемости кода, можно использовать структуру:

if
then
elif
then
elif

Общие настройки для скриптов.

Хорошей идеей было бы создать отдельную директорию на жестком диске ~/scripts, в которой будут находиться ваши скрипты. Взаимодействие с командным интерпретатором *Shell* осуществляется с помощью командной строки. Командный файл или скрипт содержит одну или несколько

выполняемых команд или процедур. Скрипт целесообразно делать тогда, когда используется одна и та же последовательность команд, записав которую, можно вызывать на выполнение многократно. По правилам хорошего тона программирования в ОС скрипт должен иметь расширение *sh*, чтобы люди отличали простые файлы от исполняемых, но это правило не всегда используется.

Если скрипты в ОС не запускаются, и система выдает ошибку, то необходимо проверить в файле */etc/fstab* напротив нужного раздела системы стоит ли параметр exec к файлам (исполняемость файлов). Перед написанием скрипта стоит зайти в */bin* и посмотреть какая оболочка установлена (команда «*find *sh**»), в дальнейшем, например, будем пользоваться *bash*. Работа со скриптами ведется через терминал.

Алгоритм создания скрипта:

– в нужной папке создать файл и заполнить его привычным для скрипта содержанием - командами, начиная с указания компилятора (это *bash*);
– изменить права к файлу-скрипту: добавить x – исполняемость;
– запустить файл в терминале.

Демонстрация результата работы скриптов.

Выше приведено много команд и значений системных переменных и лучше всего их работу я поясню на примерах.

Пример 1. Вывести в сообщении название скрипта и его автора, календарь, просмотреть текущую директорию.

Выбираем папку, например, */home*, заходим, создаем скрипт *cat>primer1.sh* с содержимым, представленным на рисунке 45, при этом надо учитывать, что «#» - решетка с пробелом это комментарий; а без пробела - системный символ. Готовый файл нужно сделать запускаемым в терминале *chmod 777 primer1.sh*. Запустить на выполнение: «*./1.sh*» или «*bash primer1.sh*» (результат представлен на рисунке 46).

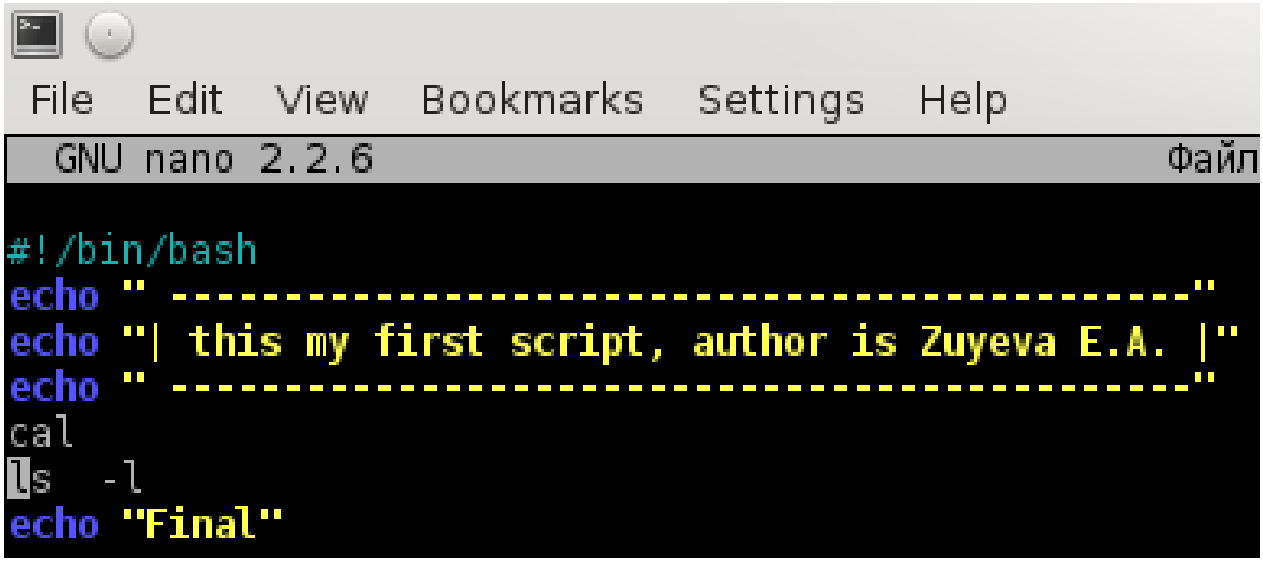

Рисунок 45 – Листинг скрипта в файле

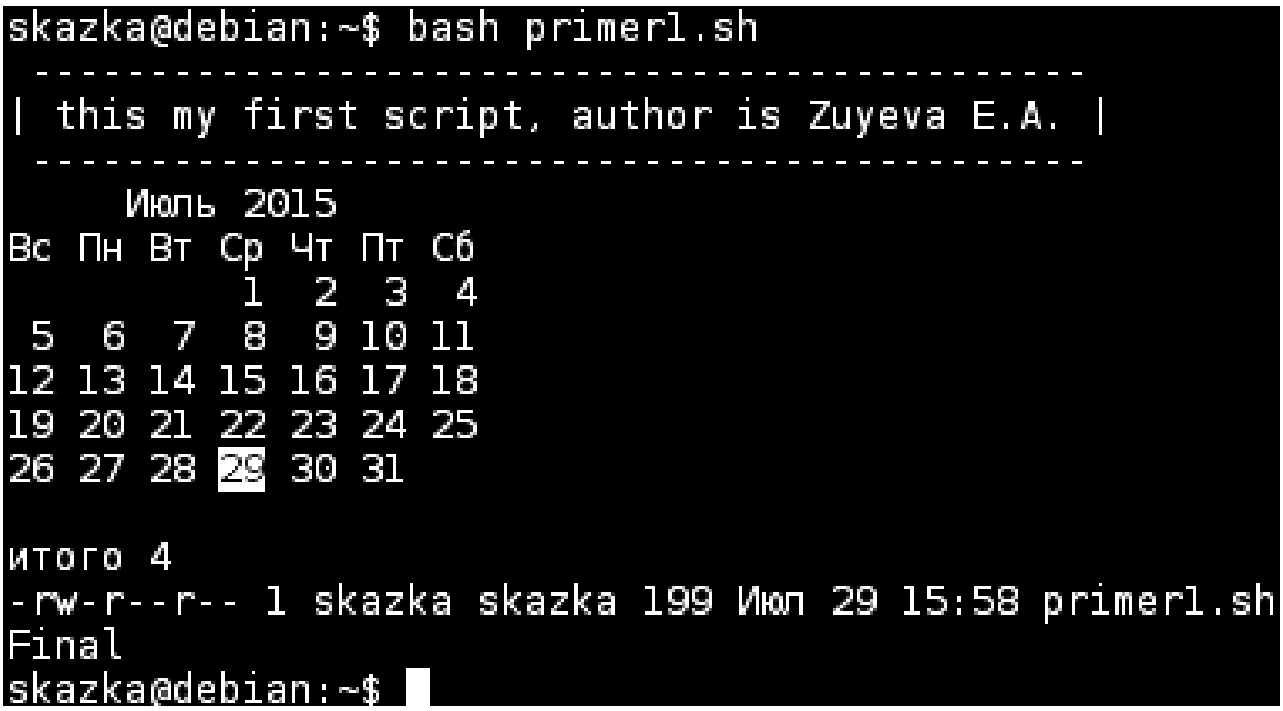

Рисунок 46 - Результат выполнения скрипта

Пример 2. Написать скрипт, выводящий текущую дату, создать файл и показать атрибуты этого файла до и после их смены.

Через консоль создаем файл (*primer2.sh*), в который пишем:

#!/bin/bash
date
touch file_for_primer2.txt
echo "byl sozdan fail, vot ego prava:"
ls –l file_for_primer2.txt
chmod 777 file_for_primer2.txt
echo "a vot prava posle izmenenii:"
ls -l file_for_primer2.txt

Файл в консоле *primer2.sh* надо сделать исполняемым, то есть ввести «*chmod primer2.sh*»

Запустить на выполнение: *./primer2.sh* или *bash primer2.sh*.

Пример 3. Для обозначения переменных используются правила:

– последовательность букв, цифр и символов подчеркивания;

– переменные не могут начинаться с цифры;

– присваивание значений проводится с использованием «=», например, *PS2* = '<';

– для обращения к значению переменной перед ее именем ставится знак $.

Переменные можно разделить на следующие группы:

- позиционные переменные вида *$n*, где *n* - целое число;

- простые переменные, значения которых может задавать пользователь или они могут устанавливаться интерпретатором;

- специальные переменные # ? - ! $ устанавливаются интерпретатором и позволяют получить информацию о числе позиционных переменных, коде завершения последней команды, идентификационном номере текущего и фонового процессов, о текущих флагах интерпретатора Shell.

Например, надо переменной x присвоить значение переменной z. Листинг:

```
z=1000
x= $z
echo $x
```

Пример 4. Присвоить переменным *x, y, z* значения 10, 100, 200, вывести *x2, y2, z2 (x2=x, y2=y, z2=z):*

```
x=10; y=100; z=200
x2=$x && y2=$y && z2=$z
echo x2=$x2,y2=$y2, z2=$z2
```

Пример 5. Организовать циклический показ информации по процессам и очистку *экрана (Ctrl+Z – выход):*

```
while clear
do
ps -a
done
```

Пример 6. Организовать циклический просмотр списка файлов и выдачу сообщения (est) при появлении заданного имени (*primer6.sh*) в списке:

```
while ls
do
        find primer6.sh && echo est
done
```

Пример 7. Подсчитать число символов в цепочках символов. Операция выполняется с использованием функции *length* в команде *expr,* а *«`»* - находится на клавише с буквой «ё»:

```
x="The program is written"
y=` expr length "$x"`
echo $y
```

Пример 8. Вывод сообщения и считывание информации с клавиатуры:

```
echo -n "Please write down your name:"
read x
echo x=$x
```

Формат условного оператора *if*:

```
if  <условие>
    then
            list1
    else
            list2
fi
```

Оператор цикла с условием *while true* и *while false*. Команда *while* (пока) формирует циклы, выполняющиеся до тех пор, пока команда *while* определяет значение следующего за ним выражения как *true* или *false*. Формат оператора цикла с условием *while true*:

```
while  list1
      do
         list2
          done
```

Здесь *list1* и *list2* - списки команд. *While* проверяет код возврата списка команд, стоящих после *while*, и если его значение равно 0, то выполняется команды, стоящие между *do* и *done*.

Оператор цикла с условием *while false* имеет формат:

```
until    list1
 do
list2
     done
```

В отличие от предыдущего случая условием выполнения команд между *do* и *done* является ненулевое значение возврата. Программный цикл может быть размещен внутри другого цикла (вложенный цикл). Оператор *break* прерывает ближайший к нему цикл. Если ввести оператор *break* с уровнем 2 (*break* 2), то это обеспечит выход за пределы двух циклов и завершение программы.

Оператор цикла с перечислением *for*:

```
   for  name    in    [wordlist]
do
     list
done,
```

где *name* - переменная; *wordlist* - последовательность слов; *list* - список команд. Переменная *name* получает значение первого слова последовательности *wordlist,* после этого выполняется список команд, стоящий между *do* и *done*. Затем name получает значение второго слова *wordlist* и снова выполняется список *list*. Выполнение прекращается после того, как кончится список *wordlist*.

Переменные начинаются с символа $. Чтобы выполнить команду и присвоить вывод переменной нужно заключить команду в апострофы(«`»). Сравните вывод двух скриптов:

```
#!/bin/bash
x=ls
echo $x
```

и

```
#!/bin/sh
x=`ls`
echo $x
```

В таблице 1 представлены листинги скриптов и результат выполнения.

Таблица 1 – Листинги скриптов с выполнением

Листинг	Результат выполнения
вывод на экран #!/bin/bash echo "Hello World"	Hello World
вывод на экран через переменную #!/bin/bash peremennaya="HELLO WORLD!!!" echo $peremennaya	HELLO WORLD!!!

глобальные и локальные переменные
#!/bin/bash
VAR="global variable"
function bash {
local VAR="local variable"
echo $VAR
}
echo $VAR
bash
echo $VAR
	global variable
	local variable
	global variable

выполнение в скрипте команд оболочки #!/bin/bash echo `uname -o` echo uname -o	GNU/Linux uname -o
чтение/ввод переменных #!/bin/bash echo -e "Type 1 word: \c " read a echo "The word you entered is: $a" echo -e "Enter two words: " read b c echo "Here is your input: \"$b\" \"$c\"" echo "How do you feel about bash scripting? " read echo "You said $REPLY, I'm glad to hear" echo "What are your favorite colours (3 colors)?" read -a colours echo "My favorite colours are also ${colours[0]}, ${colours[1]} and ${colours[2]}:-)"	Type 1 word: eferfref The word you entered is: eferfref Enter two words: rvreverfvefverver rv Here is your input: "rvreverfvefverver" "rv" How do you feel about bash scripting? fevfeverver You said fevfeverver, I'm glad to hear What are your favorite colours (3 colors) ? e3 4 4 My favorite colours are also e3, 4 and 4:-)

арифметические операции a=7 b=7 r=$(($a + $b)) echo $r echo r=$((${a} + ${b})) echo $r	14 14

циклы
for a in `seq 1 5`; do
  echo "Proshlo $a sec from Start"
  sleep 1;
done
echo "Final"
	Proshlo 1 sec from Start
	Proshlo 2 sec from Start
	Proshlo 3 sec from Start
	Proshlo 4 sec from Start
	Proshlo 5 sec from Start
	Final

массивы #!/bin/bash ARRAY=('Pervyi Element' 'Vtoroi' 'Tretii' '4-yi') ELEMENTS=${#ARRAY[@]} for ((i=0;i<$ELEMENTS;i++)); do echo ${ARRAY[${i}]} done	Pervyi Element Vtoroi Tretii 4-yi

чтение из файла 1
#!/bin/bash
declare -a ARRAY
# exec <filename с клавы stdin в файл 	
exec 10<1
let count=0
while read LINE <&10; do
  ARRAY[$count]=$LINE
  ((count++))
done
echo Number of elements:
${#ARRAY[@]}
echo ${ARRAY[@]}
# закрываем файл 	
exec 10>&-
	Number of elements: 13
	#!/bin/bash declare -a ARRAY # exec <filename с клавы stdin в файл exec 10<1 let count=0 while read LINE <&10; do ARRAY[$count]=$LINE ((count++)) done echo Number of elements: ${#ARRAY[@]} echo ${ARRAY[@]} # закрываем файл exec 10>&-

условие если-иначе #!/bin/bash a=4 echo "1. Bash" echo "2. Scripting"	1. Bash 2. Scripting 3. Tutorial Please choose number [1,2 or 3] 5 Please make a choice between 1-3 !

echo "3. Tutorial"
echo -n "Please choose number [1,2 or 3] "
while [$a -eq 4]; do
read a
if [$a -eq 1] ; then
echo "You have chosen: Bash"
else
if [$a -eq 2] ; then
echo "You have chosen: Scripting"
else
if [$a -eq 3] ; then
echo "You have chosen: Tutorial"
else
echo "Please make a choice between 1-3 !"
echo "1. Bash"
echo "2. Scripting"
echo "3. Tutorial"
echo -n "Please choose a word [1,2 or 3]? "
a=4
fi
fi
fi
done
	1. Bash
	2. Scripting
	3. Tutorial
	Please choose a word [1,2 or 3]? 3
	You have chosen: Tutorial

арифметические сравнения
-lt <
-gt >>br>-le <=
-ge >=
-eq ==
-ne !=
#!/bin/bash
a=2
b=2
if [$a -eq $b]; then
echo "Both Values are equal"
else
echo "Values are NOT equal"
fi
	Both Values are equal

сравнение переменных #!/bin/bash a=2 b=1 if [$a -eq $b]; then	2 is greater then 1

echo "Both Values are equal" elif [$a -gt $b]; then echo "$a is greater then $b" else echo "$b is greater then $a" fi	

символьно-текстовые сравнения
= одинаковые
!= неодинаковые
< меньше чем
> больше чем
-n s1 переменная s1 не пустая
-z s1 переменная s1 пустая
#!/bin/bash
a="Bash"
b="Scripting"
if [$a = $b]; then
echo "Both Strings are equal"
else
echo "Strings are NOT equal"
fi
	Strings are NOT equal

цикл for
#!/bin/bash
for f in $(ls /var/); do
echo $f
done
 	
или
for ((value=1 ; value<5; value++)) do
echo “Like it”
done
	backups
	cache
	lib
	local
	log
	mail
	opt
	run
	spool
	tmp
	www

цикл while #!/bin/bash a=6 while [$a -gt 0]; do echo Value of count is: $a let a=a-1 done	Value of count is: 6 Value of count is: 5 Value of count is: 4 Value of count is: 3 Value of count is: 2 Value of count is: 1
until цикл #!/bin/bash a=0	Value of count is: 0 Value of count is: 1 Value of count is: 2

until [$a -lt 5]; do echo Value of count is: $a let a=a+1 done	Value of count is: 3 Value of count is: 4 Value of count is: 5
оператор выбора select #!/bin/bash a='Choose one word: ' select b in "linux" "bash" "scripting" "tutorial" do echo "The word you have selected is: $b" break done exit 0	1) linux 2) bash 3) scripting 4) tutorial #? 2 The word you have selected is: bash
оператор выбора case #!/bin/bash echo "What is your lovely programming language?" echo "1) bash" echo "2) perl" echo "3) python" echo "4) c++" echo "5) nothing" read a; case $a in 1) echo "You selected bash";; 2) echo "You selected perl";; 3) echo "You selected python";; 4) echo "You selected c++";; 5) exit esac	What is your lovely programming language? 1) bash 2) perl 3) python 4) c++ 5) nothing 5 What is your lovely programming language? 1) bash 2) perl 3) python 4) c++ 5) nothing 3 You selected phyton
#!/bin/bash echo "Выберите редатор для запуска:" echo "1 Запуск программы nano" echo "2 Запуск программы vi" echo "3 Запуск программы emacs" echo "4 Выход" read doing #здесь мы читаем в переменную $doing со стандартного ввода case $doing in	Выберите редатор для запуска: 1 Запуск программы nano 2 Запуск программы vi 3 Запуск программы emacs 4 Выход После выбор цифры и нажатия Enter запуститься тот редактор, который вы выбрали (если конечно все пути указаны правильно, и у вас

1) /usr/bin/nano # если $doing содержит 1, то запустить nano ;; 2) /usr/bin/vi # если $doing содержит 2, то запустить vi ;; 3) /usr/bin/emacs # если $doing содержит 3, то запустить emacs ;; 4) exit 0 ;; *) #если введено с клавиатуры то, что в case не описывается, выполнять следующее: echo "Введено неправильное действие" esac #окончание оператора case.	установлены эти редакторы
координаты #!/bin/bash echo -en "\E[3;3f Hello, world!" #курсора координаты (3;3), а затем выведен текст	Hello, world!
временное ожидание ввода #!/bin/bash echo "I waiting you: print me 2 simbols now" read -n 2 B echo echo "I waiting you: print me 3 simbols during 5 sec" read -t 5 -n 3 B	I waiting you: print me 2 simbols now df I waiting you: print me 3 simbols during 5 sec dff
работа с объектами, аналог ls -l для .txt файлов в текущей папке #!/bin/bash ls -l | grep "\.txt$" далее уже не в файле, а в консоле alias z=”bash 1”	-rwxr-xr-x 1 root root 35 Mar 16 18:58 1.txt -rwxr-xr-x 1 root root 35 Mar 16 18:58 2.txt

16 Автозагрузка скриптов

Автозагрузка скрипта - это ситуация, в которой скрипт запускается не вручную, а автоматически по времени или по факту наступления события.

Рассмотрим процес организации автозагрузки в системе. Чтобы правильно запустить/остановить сервис, необходимо описать сценарий с командами для запуска/остановки. Содержимое каталога /*etc/init.d* содержит скрипты, которые управляют загрузками/остановками сервисов в ОС. Значит, первый, но не последний пункт успешной настройки - наличие скрипта в /*etc/init.d*. В скрипте не описывается, когда должен выполняться тот или иной сценарий. Он лишь может принимать параметры *start, stop, restart* и так далее. ОС знает, когда необходимо вызвать скрипт, так как в каталогах хранятся символические ссылки на скрипты из */etc/init.d /etc/rcN.d,* где *N* - это цифра от 0 до 6. Что означает каждый каталог:

- *rc0.d* - выполнение скрипта при выключении системы;
- *rc1.d* - выполнение скрипта при запуске системы в однопользовательском режиме;
- *rc2.d* - выполнение скрипта при запуске системы в многопользовательском режиме;
- *rc3.d* - *rc5.d* - зарезервировано;
- *rc6.d* - выполнение скрипта при перезагрузке системы.

Например, если происходит перезагрузка, то будут выполнены все скрипты из каталога /*etc/rc6.d*, при выключении из */etc/rc0.d* и так далее. Цифра в названии каталога называется уровнем запуска. То есть каталог /*etc/rc0.d* будет называться нулевым уровнем запуска и так далее. Есть очередность выполнения скриптов из каталогов *rcN.d*. Ведь для правильной организации запуска/остановки работы может потребоваться запускать/останавливать сервисы в определенном порядке. Этот момент решается специальным именованием файлов в каталогах уровней запуска. Файлы имеют следующие имена: [*S*|*K*]*NN*[имя], где [*S*|*K*] - это один символ («*S*» означает, что скрипт запускает сервис, «*K*» – останавливает), *NN* – порядковый номер, [имя] – имя файла. Символ «*S*» или «*K*» самостоятельно выбирать не придется, так как все скрипты в каталогах *rc1.d-rc5.d* должны начинаться с символа *«S*», а в каталогах *rc0.d* и *rc6.d* – с символа «*K*». Число «*NN*» определяет очередность запуска скриптов, который производится от меньшего к большему. Чем меньше число у скрипта для запуска, тем раньше он будет запущен при старте системы; чем больше число у скрипта остановки сервиса, тем позже он будет выполнен.

При необходимости запуска какой-либо службы или приложения до или после конкретного существующего сервиса надо посмотреть его порядковый номер в соответствующей директории *rcN.d* и учитывать при выборе порядкового номера для своего скрипта.

В каталоге */etc/init.d* находится пример скрипта для управления запуском/остановкой сервисов. Это файл /*etc/init.d/skeleton*, а в примере ниже

он будет упрощен. Для создания нового скрипта необходимо сделать копию примера и отредактировать его. Далее надо воспользоваться командой: «*sudo cp /etc/init.d/skeleton /etc/init.d/myscript && nano /etc/init.d/myscript*». При создании нового скрипта надо дать ему права на выполнение: «*chmod +x /etc/init.d/myscript»*.

Можно воспользоваться специализированной утилитой *update-rc.d*, с её помощью можно добавить новый скрипт в любой уровень загрузки, удалить существующий и так далее. Вот пример использования: «*sudo update-rc.d myscript start 99 2 3 4 5 . stop 01 0 1 6*» - добавит новый скрипт «*myscript*» во все уровни загрузки. Будет выполнен запуск сервиса на уровнях со 2 по 5 с приоритетом 99 (в последнюю очередь) и остановка сервиса на 0, 1 и 6 уровнях с приоритетом 01 (самым первым). Чтобы удалить скрипт из автозагрузки: «*sudo update-rc.d -f myscript remove*».

Cron - планировщик *Linux*, он выполняет задания по расписанию. *CronTab* – утилита, позволяющая автоматически запускать программы в определенное время, в том числе и периодически, например, раз в час, каждую пятницу и т.д.

Структура файла с заданиями для *CronTab*:

* * * * * command

— — — — —

| | | | |

| | | | +—— День недели (0 - 6) (Sunday=0)

| | | +——- Месяц года (1 - 12)

| | +——— День месяца (1 - 31)

| +———— Час дня запуска (0 - 23)

+————- Минута часа для запуска (0 - 59)

command - запускаемая программа или скрипт; значок * задаёт параметр (день, год, месяц, час).

Пример:

01 * * * * *xclock* - запуск программы xclock каждый час в одну минуту.

0 6 * * * *script* - запуск скрипта каждый день в 6 часов утра.

Значения могут быть числом, трехбуквенным названием, а также диапазоном, например, запись «1-5» в поле day будет означать «с понедельника по пятницу». Значения могут отделяться запятыми: «1,15,31» в поле *day* будет запускать указанную команду 1-го, 15-го и 31-го числа каждого месяца.

Все пять полей времени допускают использование символа «*», который обозначает «использовать любое допустимое значение» для этого поля.

Для создания задания можно использовать команды:

– *crontab -e* - изменит *crontab* файл или создаст новый;

– *crontab -l* - отобразит содержимое существующего *crontab* файла;

– *crontab -r* - удалит *crontab* файл;

– *crontab -v* –отображает, когда в последний раз изменяли свой *crontab*.

Список литературы

Основная

1 Таненбаум Э. Современные операционные системы. - СПб: Питер, 2013. – 1120 с.

2 Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы: Уч-к для вузов. - СПб.: Питер, 2013. - 943 с.

3 Дейтел Харвин М. Операционные системы. Основы и принципы. - Т. 1-3-е изд.- М.: БИНОМ, 2011. - 1024 с.

4 Эви Немет, Гарт Снайдер и др. UNIX и Linux. Руководство системного администратора. - Киев: Вильямс, 2012. – 1312 с.

Дополнительная

5 Национальный открытый университет ИНТУИТ. Основы операционных систем http://www.intuit.ru/studies/courses/1088/322/info

6 Национальный открытый университет ИНТУИТ. Операционные среды, системы и оболочки http://www.intuit.ru/studies/courses/492/348/info

7 Национальный открытый университет ИНТУИТ. Основы работы в ОС Linux http://www.intuit.ru/studies/courses/91/91/info

Зуева Екатерина Александровна

ОПЕРАЦИОННАЯ СИСТЕМА LINUX

Учебное пособие

Редактор Н.М. Голева

Подписано в печать 23.02.2017
Тираж 100 экз. Формат 60x84 1/16

Бумага типографская №2
Уч.-изд.л. 5.44 Заказ №___
Цена 2720 тенге

Некоммерческое АО «АУЭС»
Алматы, ул. Байтурсынова, 126

Копировально-множительное бюро
некоммерческого акционерного общества
«Алматинский университет энергетики и связи»
Алматы, ул. Байтурсынова, 126