МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования

«Дальневосточный федеральный университет»

(ДВФУ)

ШКОЛА ЕСТЕСТВЕННЫХ НАУК ДВФУ

«СОГЛАСОВАНО»
Руководитель ОП
д.ф.-м.н., профессор, академик РАН, Гузев М.А.

_____
(подпись) (Ф.И.О. рук. ОП)
«23» июня 2017 г.

«УТВЕРЖДАЮ»
Заведующая (ий) кафедрой
информатики, математического и компьютерного моделирования
(название кафедры)

_____ Чеботарев А.Ю.
(подпись) (Ф.И.О. зав. каф.)
«23» июня 2017 г.

РАБОЧАЯ ПРОГРАММА УЧЕБНОЙ ДИСЦИПЛИНЫ (РПУД)
Операционные системы

09.03.03 Прикладная информатика

Форма подготовки очная

Школа естественных наук ДВФУ
Кафедра информатики, математического и компьютерного моделирования
Курс 3 семестр 2
лекции 26 (час)
практические занятия
семинарские занятия
лабораторные работы 72 (час.)
консультации
всего часов аудиторной нагрузки 98 (час.)
самостоятельная работа 40 (час.)
контрольные работы
зачет 6 семестр

Рабочая программа составлена в соответствии с требованиями образовательного стандарта, самостоятельно установленного ДВФУ, принятого решением Ученого совета Дальневосточного федерального университета, протокол от 28.01.2016 № 01-16, и введенного в действие приказом ректора ДВФУ от 18.02.2016 № 12-13-235.
Рабочая программа обсуждена на заседании кафедры информатики, математического и компьютерного моделирования, протокол №22 «23» июня 2017 г..

Заведующий кафедрой _____ А.Ю. Чеботарев
Составитель ведущий инженер-программист кафедры информатики, математического и компьютерного моделирования
_____ И.С. Олейников

Аннотация дисциплины

«Операционные системы»

Операционные системы возникли в 60-х 80-х годах XX века как попытка вынести логику работы из вычислительного устройства в отдельный модуль, являющийся независимым от остального устройства частью. При этом, изначально, на подобные системы были наложены только требования работы с устройствами памяти (перфокарты, перфоленты, магнитные диски), поскольку от них требовалось лишь обработать данные пользователя и выдать результат. Постепенно требования расширялись и к 90-м годам XX столетия произошла революция в сознании множества людей, с появлением такого устройства как персональный компьютер. Понимание того, что компьютер может присутствовать не только в сложных математических и физических расчетах, но и в доме у каждого человека, для выполнения его повседневной работы, игры и контроля безопасности изменило мир.

Работа подобных систем уже не укладывалсь в простую схему ввод данных –> обработка данных –> вывод информации. Появилась необходимость интерактивного взаимодействия между человеком и вычислительной машиной. Такую нагрузку и взяли на себя появившиеся в то время Операционные системы. Если процессор современного компьютера является его мозгом, то, можно считать, что современные ОС являются сознанием машины.

Учебная программа по дисциплине Операционные системы в полной мере отражает важность понимания устройства этого непростого объекта в мире современных компьютерных технологий. Полное понимание данного аспекта важно, как для программистов, составляющих компьютерные программы, так и для системных администраторов, настраивающих машины и базы данных и выполняющих работы по оптимизации их скорости работы.

Программа дисциплины соответствует требованиям для специальности «09.03.03 Прикладная информатика».

Изучаемая дисциплина формирует основные компетенции специалиста в области Операционных систем современных компьютеров и других вычислительных устройств.

Дисциплина разрабатывалась с рассчетом на свободное ПО и большинство заданий в курсе может быть выполнено без использования проприетарного программного обеспечения. Однако с целью полного охвата всей темы в ней рассмотрены такие ОС как Windows, являющейся основной используемой на персональных компьютерах в России и ближнем зарубежье.

Целью изучения дисциплины «Операционные системы» является изучение принципов организации современных операционных систем, их состава и схемы работы, принципов управления ресурсами и методов организации файловых систем. Ознакомление с принципами сетевого взаимодействия операционных систем, а также основными методами разработки программного обеспечения для них.

Для успешного освоения дисциплины требуется освоение студентами следующих курсов: «Программирование на C++», «Технологии программирования», «Язык Ассемблера». Данный курс может изучаться параллельно, либо быть предшествующим, с курсом «Параллельное программирование».

Курс «Программирование на C++» является обязательным, поскольку дает основополагающие знания о языке ядра операционных систем Unix подобного типа и значительно упрощает выполнение практических заданий, связанных с Windows API. Курс «Технологии программирования» дает представление о сетевом взаимодействии программ средствами операционных систем и необходим для успешного изучения Сетевых и распределенных ОС. Курс «Язык Ассемблера» дает общее представление о работе базовых частей ядра различных ОС. Дисциплина «Параллельное программирование».

Пересекается с описываемым курсом в части работы сетевых операционных систем.

В результате освоения дисциплины обучающийся должен:

– Знать: принципы организации, состав и схемы работы операционных систем, принципы управления ресурсами, методы организации файловых систем, принципы построения сетевого взаимодействия ОС, основные стандарты POSIX.

– Уметь: работать на различных типах ЭВМ, использующих различные ОС, такие как Ubuntu Linux, Windows 7 и т.д. а также составлять для этих операционных систем прикладные программы с использованием функций ядра ОС и стандартной библиотеки.

– Владеть навыками работы с: Unix подобными ОС, включая вызовы стандартных библиотек и прикладных программных интерфейсов (WinAPI, POSIX).

– Владеть навыками работы с ОС типа Windows и ее программными эмуляторами, например Wine.

ОК-5 способность использовать современные методы и технологии (в том числе информационные) в профессиональной деятельности	знает	современные методы и технологии (в том числе информационные)
	умеет	использовать современные методы и технологии (в том числе информационные) в профессиональной деятельности
	владеет	навыками использования современных методов и технологий (в том числе информационных)

ПК-8 Способностью программировать приложения и создавать программные прототипы решения прикладных задач	знает	понятия информатики: данные, информация, знания, информационные системы и технологии; методы структурного и объектно-ориентированного программирования.
	умеет	разрабатывать и отлаживать эффективные алгоритмы и программы с использованием современных технологий программирования
	владеет	навыками моделирования прикладных задач; численными методами; навыками программирования в современных средах.

ПК-13 Способностью проводить тестирование компонентов программного обеспечения	знает	принципы организации проектирования и содержание этапов процесса разработки программных комплексов;
	умеет	формулировать требования к создаваемым программным комплексам;
	владеет	работы в современной программно-технической среде в различных операционных системах; разработки программных комплексов для решения прикладных задач, оценки сложности алгоритмов и программ, использования современных технологий программирования, тестирования и документирования программных комплексов работы с инструментальными средствами моделирования предметной области, прикладных и информационных процессов;

ПК-14 Способностью осуществлять инсталляцию и настройку параметров программного обеспечения информационных систем	знает	теоретические основы построения и функционирования операционных систем, их назначение и функции;
	умеет	использовать различные операционные системы;
	владеет	работы в современной программно-технической среде в различных операционных системах; разработки программных комплексов для решения прикладных задач, оценки сложности алгоритмов и программ, использования современных технологий программирования, тестирования и документирования программных комплексов работы с инструментальными средствами моделирования предметной области, прикладных и информационных процессов;

1. Цели освоения дисциплины

В результате освоения данной дисциплины бакалавр приобретает знания, умения и навыки, обеспечивающие достижение целей основной образовательной программы «Прикладная математика и информатика».

Целями освоения дисциплины операционные системы являются изучение принципов организации современных операционных систем, их состава и схемы работы, принципов управления ресурсами и методов организации файловых систем. Ознакомление с принципами сетевого взаимодействия операционных систем, а также основными методами разработки программного обеспечения для них.

2.Место дисциплины в структуре ОП бакалавриата

Курс «Операционные системы» (ОС) является предметом цикла дисциплины специализации (ДС) по направлению подготовки «Прикладная математика и информатика», т.к. дает специализированные знания в области операционных систем различных вычислительных машин, в том числе, в самом распространенном варианте – персональных компьютеров.

Для успешного освоения дисциплины требуется освоение студентами следующих курсов: «Программирование на C++», «Технологии программирования», «Язык Ассемблера». Данный курс может изучаться параллельно, либо быть предшествующим, с курсом «Параллельное программирование».

Курс «Программирование на C++» является обязательным, поскольку дает основополагающие знания о языке ядра операционных систем Unix подобного типа и значительно упрощает выполнение практических заданий, связанных с Windows API. Курс «Технологии программирования» дает

представление о сетевом взаимодействии программ средствами операционных систем и необходим для успешного изучения Сетевых и распределенных ОС. Курс «Язык Ассемблера» дает общее представление о работе базовых частей ядра различных ОС. Дисциплина «Параллельное программирование». Пересекается с описываемым курсом в части работы сетевых операционных систем.

3.Компетенции выпускника ОП бакалавриата, формируемые в результате освоения данной ОП ВПО.

В результате освоения дисциплины обучающийся должен:

– Знать: принципы организации, состав и схемы работы операционных систем, принципы управления ресурсами, методы организации файловых систем, принципы построения сетевого взаимодействия ОС, основные стандарты POSIX.

– Уметь: работать на различных типах ЭВМ, использующих различные ОС, такие как Ubuntu Linux, Windows 7 и т.д. а также составлять для этих операционных систем прикладные программы с использованием функций ядра ОС и стандартной библиотеки.

– Владеть навыками работы с: Unix подобными ОС, включая вызовы стандартных библиотек и прикладных программных интерфейсов (WinAPI, POSIX).

– Владеть навыками работы с ОС типа Windows и ее программными эмуляторами, например Wine.

4. СТРУКТУРА И СОДЕРЖАНИЕ КУРСА

5. Таблица 1. Общее содержание дисциплины

№	Раздел дисциплины	Семестр	Неделя семестра	Виды учебной работы, включая самостоятельную работу студентов и трудоемкость (в часах)			Формы текущего контроля успеваемости *(по неделям семестра)* Форма промежуточной аттестации *(по семестрам)*
				Лекции	Лабораторные	Самостоя-тельная работа	
1	История развития операционных систем	9	1	2		10	Реферат (последняя неделя семестра)
2	Unix подобные ОС на примере Ubuntu Linux	9	2,3	8	8	16	Результаты лабораторных работ: 2, 3, 4 недели семестра
3	Windows подобные ОС на примере Windows 7	9	4,5,	10	14	8	Результаты лабораторных работ: 5, 6, 7 недели семестра

4	Операционные системы реального времени (Общие принципы работы)	9	7,8	2	2	4	Результаты лабораторных работ: 8, 9 недели семестра
5	Сетевые и распределенные ОС	9	9	4	2	2	Результаты лабораторных работ: 10 неделя семестра

6. Таблица 2. Теоретический курс

№ раздела	Тема лекции	Количество часов освоения темы	
		Лекции	Сам. работа
1	Операционная система. Определение. История развития ОС. Функции ОС. Классификация операционных систем по особенностям алгоритмов управления ресурсами, особенностям аппаратных платформ, особенностям областей использования. Типы архитектур ядра операционных систем. Архитектура микроядра. Архитектура монолитного ядра.	2	10 (написание реферата)
2	Linux. Краткая история разработки. Различия Linux и Unix. Процессы и нити Unix систем. Состояния процесса в Linux. Системные функции управления процессами. Сигналы.	2	4

	Планирование процессов. Стандарты POSIX.		
2	Linux. Организация памяти. Свопинг и подкачка по запросу. Алгоритмы выделения памяти. Системные вызовы управления памятью. Вирутальное адресное простарнство.	2	4
2	Linux. Фундаментальные концепции безопасности в Linux. Организация доступа. Распределение прав доступа. Назначение прав пользователя. Реализация безопасности Linux.	1	2
2	Linux. Ввод/Вывод в Unix системах. Работа с сетью. Работа с файлами. Системные вызовы Ввода/Вывода. Файловые системы Unix. Виртуальная файловая система VFS. Файловые системы Ext2, Ext3.	2	4
2	Linux. Установка прикладных программ. Репозитории. Политика распространения исходных кодов программ. Оболочки Unix. Программное окружение Unix. Оболочки - C-shell, Bourne-shell, Korn-shell, bash.	1	2
3	Windows. История Windows. Структура операционной системы. Уровень абстрагирования оборудования HAL. Уровень ядра. Уровень пользовательских приложений. Асинхронные вызовы процедур. Процессы и потоки в Windows. Приоритеты процессов. WinAPI для работы с процессами.	2	
3	Windows. Управление памятью. Файл подкачки. Адресация памяти. Виртуальная	2	

	память. WinAPI для управления памятью. Алгоритм замещения страниц. Кэширование.		
3	Windows. Безопасность в Windows. Функции WinAPI для разграничения доступа. Дескрипторы безопасности.	1	2
3	Windows. Реестр Windows. Организация реестра. Кусты, узлы и из соответствие системным файлам.	1	2
3	Windows. Файловая система Windows. FAT12. FAT16. FAT32. NTFS. Ввод/Вывод в Windows. WinAPI для ввода/вывода. Драйверы устройств.	2	2
3	Windows. Прикладные программы Windows. Оболочки. Командная строка. PowerShell. Команды PowerShell, принцип работы, написание скриптов.	2	2
4	RTOS. Дополнения к Ubuntu для запуска ее в режиме реального времени. Требования к системам жесткого и мягкого реального времени.	2	4
5	Распределенные ОС. Мультипроцессоры и мультикомпьютеры. Распределение памяти. UMA. NUMA	2	
5	Распределенные ОС. Сетевые операционные системы. Соединение компьютеров через Internet. Облачные вычисления. Grid-системы. Globus Toolkit	2	2

6. Учебно-методическое обеспечение самостоятельной работы студентов. Оценочные средства для текущего контроля успеваемости, промежуточной аттестации по итогам освоения дисциплины.

Для контроля успеваемости студентов используется набор практических заданий и реферат. По факту выполнения определенного процента заданий студенту выставляется та или иная оценка. Страница курса расположена и поддерживается в системе учета студенческих работ AWorks (Разработчик А. С. Кленин) по адресу http://imcs.dvgu.ru/works/course_view?id=16635 .

Реферат

В рамках курса каждый студент должен написать и защитить реферат по выбранной им, или оставшейся после распределения всех остальных, теме. Каждый студент получает индивидуальную тему реферата. Варианты тем представлены в таблице 3. Каждая тема формируется как пара из первого и второго столбцов, например: «Ядро и стандартная библиотека MacOS» или «Файловая система Linux». Таким образом из таблицы 3 получается 20 тем рефератов. В случае необходимости столбец операционные системы разрешается дополнять другими ОС, не включенными в список.

Таблица 3. Темы предлагаемых рефератов.

Тема реферата	**Название реферируемой ОС**
Ядро и стандартная библиотека	Windows
Управление памятью	Linux
Файловая система	MacOS
Технологии безопасности	Android, QNX, и т.д.

Реферат должен содержать не более 5 страниц в содержательной части реферата и в сжатой форме отражать уникальные особенности выбранной

части выбранной операционной системы. Дополнение реферата историей развития ОС допускается только в случае крайней необходимости описания эволюции ОС, приведшей к какой-либо уникальной особенности. Например: сохранение в WinAPI в целях совместимости различных устаревших функций из старых версий ОС, что привело к огромному количеству функций в стандартной библиотеке API Windows, а также к тому, что многие новые функции получают в названии префикс “w_” или суффикс “NT”.

Прием и защита реферата осуществляется в два этапа. После передачи готового реферата преподавателю дается неделя на проверку и подготовку вопросов к сданной работе. После передачи вопросов студенту дается неделя на подготовку ответов на поставленные вопросы. Вопросы обязаны быть по теме содержания реферата (не по оформлению) и только по неуказанным в реферате, но необходимым для понимания темы, или неправильно указанным моментам.

Практические задания

Практическое задание 1. Linux.

Задание находится на пересечении тем «Файловая система Linux», «Команды оболочки Linux» и «Процессы и потоки в Linux».

Имеется файл, в который программа производит запись. Переместить его командой mv

а) на тот же физический носитель

б) на другой физический носитель (можно использовать флэшку)

Объяснить произошедшее.

Варианты решения:

В данной задаче происходящее будет сильно зависеть от дистрибутива Linux, на котором выполняется задание, от версии этого дистрибутива, а также оттого, насколько непрерывно программа производит запись в файл.

Если программа пишет в файл, не закрывая его после каждой записи, и, перерыв между записями командой sleep, wait и подобными не сделан, то в случае (А) программа продолжит писать в перенесенный файл. Это произойдет по причине сохранения inode для данного файла в пределах физического носителя и VFS. В случае (Б) в зависимости от дистрибутива ОС программа может либо перенести на другой физический носитель то, что было в файле на момент вызова команды, а записанное потом удалить в /dev/null, либо скопировать уже записанное содержимое и вызвать ошибку в программе производящей запись, либо продолжить писать по полного заполнения другого носителя.

В случае если запись производится скриптом на bash, sh и т.д. и утилита закрывает файл после каждой операции записи то в случае (А) будет перенесено уже записанное, а для последующей записи создан новый файл, вместо перенесенного. В случае (Б) возможен вариант с вызовом ошибки в записывающей программе, либо, более вероятно, полное повторение эффектов случая (А).

Практическое задание 2. Linux.

Задание по теме «Команды оболочки Linux»

В каталоге имеется значительное количество файлов с именами вида abcd20021211.res где 2002 - год, 12- месяц, 11 - день в месяце. Недавно выяснилось, что в именах файлов был перепутан год, т.е. проставленный

является на 1 меньшим, чем реальный год создания файла. А буквы перед годом являются аббревиатурой и должны быть записаны прописными буквами.

Задача: написать скрипт на одной из linux shell, который приведет имена файлов в порядок.

Варианты решения:

В данной задаче требуется написать либо скрипт с циклом и разбиением названия файла на компоненты на linux shell, либо в случае разрешения использовать perl скрипты, возможно обойтись одной командой prename с соответствующим регулярным выражением в качестве аргумента.

Небольшую сложность здесь представляет то, что год должен содержать ровно 4 знака и в случае недостатка цифр дополнятся справа необходимым количеством нулей.

Практическое задание 3. Linux.

Задание находится на пересечении тем «Файловая система Linux», «Безопасность Linux» и «Процессы и потоки в Linux».

Профессор хранит журнал с оценками студентов по его предмету в виде файла на общем кафедральном компьютере. Причем права доступа к файлу установлены таким способом, что профессор может записывать в этот файл и читать из него, а все остальные могут только читать файл (права 700). В одно прекрасное утро профессор случайно установил права на запись в этот файл всем желающим (права 707 или 777). К вечеру того же дня он спохватился и изменил права доступа к файлу на прежние (себе - чтение и запись, остальным - только чтение). После этого профессор

проверил, что файл не изменился за день (сверил с копией на своем ноутбуке), и, спокойный, ушел домой. Утром профессор обнаружил, что некоторые студенты выставили себе дополнительные оценки за несданные еще задания в этот файл. Задача – воспроизвести ситуацию.

Варианты решения:

Решение задачи основано на системе разрешения доступа процесса к файлу. В том случае, если файл открывается каким-либо процессом, то он сохраняет права, даже если потом права доступа к файлу изменились. Некоторые ошибки могут заключаться в том, что если использовать оператор fopen языка Си, то некоторые компиляторы вставляют проверку прав доступа при каждом обращении к файлу. Желательно при открытии и доступе к файлу использовать функции стандартной библиотеки Linux Open, которая предназначена для чтения данных как с файлов, так и с устройств. Тогда предварительное открытие файла, перед сменой прав доступа гарантированно сработает.

Практическое задание 4. Linux.

Задание по теме «Linux. Установка прикладных программ. Репозитории»

Установить программу NCView (http://meteora.ucsd.edu/~pierce/ncview_home_page.html), включая библиотеку для чтения файлов формата HDF5 и библиотеку работы с единицами измерения Udunits.

Варианты решения:

В данной работе предлагается два варината решения. Поскольку в стандартном репозитории любого дистрибутива Linux данной программы нет, то возможна как установка из исходных кодов, так и добавление стороннего репозитория к списку используемых. В любом случае установка библиотек NetCDF, HDF5 и Udunits. Потребует дополнительной установки и встраивания в систему. Поскольку программа предназначена для просмотра срезов полей данных различной размерности (до 7 мерных варинатов) проверка осуществляется открытием в установленной программе файлов соответствующих форматов. Примеры имеются на сайте программы.

Практическое задание 5. Windows.

Задание по теме «Windows. Работа с реестром»

Используя инструмент regedit произвести экспорт реестра windows во внешний файл, подождать некоторое время и еще раз экспортировать реестр в другой файл. сравнить полученные файлы, объяснить результаты.

Варианты решения:

В данной задаче изменения реестра будут связанны с фоновыми процессами, происходящими в операционной системе, либо с работой запущенных пользователем программ. При поиске изменений и их комментировании (включает указание внесшей их программы) студент учится определять как именно операционная система Windows использует реестр.

Практическое задание 6. Windows.

Задание находится на пересечении тем «Безопасность Windows» и «Windows. Работа с реестром»

ВНИМАНИЕ!!! Данное задание необходимо выполнять с использованием виртуальной машины (VirtualBox, VMWare, VirtualPC и т.д.). В противном случае за утерю данных и порчу программ при выполнении задания никто ответственности не несет!!! Найти на просторах интернета, либо взять из вирусария преподавателя (за самостоятельный поиск начисляются дополнительные баллы) локализовать и обезвредить вирус из разряда Windows-блокера, изменяющий записи реестра windows для загрузки самого себя вместо explorer.exe, userinit и им подобных загрузочных записей, либо вместе с ними и не дающий выполнять какие либо операции с системой.

Советы:

В первую очередь рекомендуются к просмотру следующие ветки реестра: HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon и HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options

Варианты решения:

Универсальным вариантом решения (без использования антивирусов) является подключение жесткого диска зараженной виртуальной машины (настоятельно рекомендуется заражать именно чистую виртуальную машину, не содержащую ценных данных) к другому компьютеру и проверка реестра через regedit. Проверяются ветки, указанные в советах к задаче. Сравнение рекомендуется производить с аналогичными ветками чистой виртуальной машины. Все несовпадения удаляются, а пути к несовпадающим программам проверяются на предмет наличия испольняемого файла вируса, который и сдается в качестве локализованного (в архиве под паролем). Вирусарий прилагается в УМКД. Все вирусы в нем локализуются описанным выше способом.

Практическое задание 7. Windows.

Задание по теме «Файловая система Windows»

Создать в Windows пустой файл (размером 0 байт), при удалении которого будет освобождено более 500 Мб места на диске.

Варианты решения:

Решение задачи основано на использовании в Windows файловой системы NTFS, позволяющей представлять файлы в виде набора потоков, при этом содержимое файла находится в стандартном потоке :$DATA. При проверке размера файла операционной системой указывается количество байт именно в этом потоке, все остальные потоки игнорируются, однако копируются и удаляются только совместно с файлом. Для выполнения задания достаточно стандартной Windows утилиты echo. Тогда команда выглядит следующим образом echo bigfile.file > zerofile.file:hiddenstream. После выполнения этой команды в поток hiddenstream файла zerofile.file будет записано содержимое большого файла, которое удалится вместе с файлом zerofile.file, размер которого будет 0 байт.

Практическое задание 8. Windows.

Задание находится на пересечении тем «Файловая система Windows» и «стандартная библиотека WinAPI»

Написать программу, выводящую список всех файловых потоков, прикрепленых к заданному файлу.

Варианты решения:

Возможно два варианта решения. Первый состоит в использовании недокументированной функции WinAPI “NtQueryInformationFile”, второй использует функции FileBackup создавая резервную копию файла со всеми его потоками. После создания резервной копии в заголовке ее будут перичислены названия всех потоков.

Практическое задание 9. Windows.

Задание находится на пересечении тем «Файловая система Windows» и «прикладные программы Windows»

Создать в файловой системе NTFS файл, у которого параметр "Размер" будет больше размера диска, на котором находится файл. (Не путать с параметром "Размер на диске").

Варианты решения:

Очевидно, файл должен быть “прореженным” (sparce). Этого можно добиться при помощи команды fsutil sparse setflag file. После этого файл заполняется нолями на небольшое количество байт, например 256. Затем создается его резервная копия с помощью процедур WInAPI из задания 8, либо сторонней утилитой , например NTFS BackupRead Dumper (http://hex.pp.ua/backupread-util.php). Полученный файл открывается любым HEX редактором (например встроенный в оболочку Far) и в нем исправляются последние 2 байта, отвечающие за размер файла. После этого файл восстанавливается из резервной копии. Восстановленный файл может иметь размер больше размера жесткого диска на котором он размещается. Это происходит из-за хранения разреженный файлов не целиком, а кусками с пропусканием пустых кусков и их заполнению по требованию.

Практическое задание 10. Windows.

Задание по теме «Оболочки Windows»

В каталоге имеется значительное количество файлов с именами вида abcd20021211.res где 2002 - год, 12- месяц, 11 - день в месяце. Недавно выяснилось, что в именах файлов был перепутан год, т.е. проставленный является на 1 меньшим, чем реальный год создания файла. А буквы перед годом являются аббревиатурой и должны быть записаны прописными буквами.

Задача: написать скрипт на Windows PowerShell, который приведет имена файлов в порядок. Запрещено использовать конструкции циклов.

Варианты решения:

Та же задача, что и задание 2, только в Windows. Усложнение с запретом циклов используется, чтобы обязать написать данный скрипт в стиле функционального программирования, т.е. вместо циклов необходимо создать в PowerShell список файлов в каталоге при помощи команды Get-Item, а затем обработать весь список регулярными выражениями, выделяющими нужные компоненты названия файла, как один файл.

Практическое задание 11. Сетевые и распределенные ОС.

Задание по теме «Распределение нагрузок»

Реализовать алгоритм разбиения графа на два подграфа таким образом, чтобы потоки информации между подграфами были минимальны или близки к минимальным, используя эвристику Кернигана - Лина.

Варианты решения:

Реализация стандартного алгоритма планирования распределения вычислительной нагрузки в сети или кластере при помощи эвристики, придуманной Б. Керниганом и С. Лином.

7. Учебно-методическое и информационное обеспечение дисциплины (модуля)

Основная литература:

1.1 Таненбаум Э. Современные операционные системы. СПб.: Питер, 2002. 1040 с.

1.2 Олифер В.Г., Олифер Н.А. Сетевые операционные с истемы. СПб.: Питер, 2001. 544 с.

Дополнительная литература:

2.1 Колисниченко Д. Н., Аллен П. LINUX: полное руководс тво. - СПб: Наука и Техника, 2006. - 784 с .

2.2 Столлингс В. Операционные системы. М.: Вильямс , 2002. 848 с.

Программное обеспечение и Интернет-ресурсы

3.1 Сайт курса http://imcs.dvgu.ru/works/course_view?id=16635

8. Материально-техническое обеспечение дисциплины (модуля)

1) Ubuntu Linux версии не ниже 10.4
2) Виртуальная машина Virtual Box (можно использовать VMware или Virtual PC)

3) Желательно Windows версии не ниже XP (можно использовать эмулятор Wine for Linux)
4) Исходные коды программы NCView для выполнения практического задания №4 (*http://meteora.ucsd.edu/~pierce/ncview_home_page.html*)
5) Набор вирусов, для выполнения практического задания №6 (прилагается в электронном виде)
6) Программа-оболочка Far 2.0
7) Программа –оболочка PowerShell

Программа составлена в соответствии с требованиями ФГОС ВПО с учетом рекомендаций и ПрООП ВПО по направлению подготовки 010400.62 «Прикладная математика и информатика».

Автор ведущий инженер-программист _____ Олейников И.С.

Программа одобрена на заседании_____

(Наименование уполномоченного органа вуза (УМК, НМС, Ученый совет)

от _____ года, протокол № _____.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДВФУ)

<ШКОЛА ЕСТЕСТВЕННЫХ НАУК>

КОНСПЕКТЫ ЛЕКЦИЙ

по дисциплине «Операционные системы»

Направление— 230700.62 «прикладная информатика»

г. Владивосток

Операционная система Linux

Загрузка ОС Linux

1: BIOS

Когда вы включаете питание компьютера, в его оперативной памяти еще нет никакой программы, поэтому управление компьютером может осуществляться только аппаратным обеспечением. На Intel-овских платформах начальная загрузка операционной системы осуществляется посредством так называемой “базовой системы ввода/вывода” или BIOS. Я не буду приводить здесь это описание полностью, перечислю только кратко основные этапы.

После включения компьютера блок питания проверяет все необходимые уровни напряжений. Если все уровни напряжений соответствуют номинальным, то на материнскую плату поступит сигнал PowerGood. До появления этого сигнала на вход процессора подается сигнал RESET, который удерживает процессор в сброшенном состоянии. Но после получения сигнала PowerGood от блока питания сигнал RESET будет снят и процессор начнет выполнять свои первые инструкции. При этом процессор стартует от вполне известного состояния: командный регистр CS содержит 0xFFFF, указатель команд (регистр IP) содержит 0, сегментные регистры данных и стека содержат 0. Таким образом, после снятия RESET процессор в реальном режиме выполняет инструкции, размещающиеся в области ROM BIOS, начинающейся с адреса FFFF:0000 (физический адрес, соответственно, - 0xFFFF0). Размер этой области, очевидно, составляет 16 байт, вплоть до конца максимально адресуемого адресного пространства в реальном режиме - 0xFFFFF. По этому адресу располагается инструкция перехода на реально исполняемый код BIOS.

По соображениям снижения стоимости код BIOS в современных материнских платах хранится в постоянной памяти (ПЗУ) в сжатом виде. Только небольшая его часть, используемая на самых первых этапах загрузки, является непосредственно исполняемой. Поэтому первая задача, которая решается сразу после включения питания, заключается в том, чтобы инициализировать контроллер DRAM, декомпрессировать основной код BIOS и загрузить его в ту область оперативной памяти (RAM), которую именуют "теневой" (shadow RAM). Эта область затем защищается от записи и управление передается на записанный в нее исполняемый код BIOS. Теневая память в ходе дальнейшей работы отдана в полное владение чипсета материнской платы; операционная система к ней доступа не имеет. Но аппаратными средствами обеспечивается отображение теневой памяти на те

области, которые в реальном режиме работы доступны для старых операционных систем типа MS-DOS, так что последние обнаруживают код BIOS именно там, где ожидают его найти.

Исполняемый код BIOS вначале реализует функцию начального самотестирования (POST - Power-On Self Test). При этом тестируются процессор, память и системные средства ввода/вывода, а также производится конфигурирование программно-управляемых аппаратных средств компьютера. Кроме того производится поиск и обнаружение периферийных устройств. При этом производится сравнение установок, записанных в CMOS (Complementary Metal Oxide Semiconductor) с тем, что реально обнаружено в системе. Некоторые несовпадения, например, различие типов флоппи-дисковода, могут быть допустимы и процесс загрузки продолжится. Другие ошибки, например, отсутствие видеокарты, приводят к невозможности дальнейшей загрузки. Но все сообщения о выявленных на этом этапе ошибках сводятся только к тому, что раздастся несколько коротких звуковых сигналов.

Можно еще отметить, что некоторые типы периферийных устройств могут содержать расширения BIOS в собственных ПЗУ. В таком случае устанавливаются соответствующие ссылки на эти расширения. Основная причина, по которой это необходимо, заключается в том, что по историческим причинам размер первичного загрузчика (загрузчика первого этапа, как мы его будем дальше называть) на персональных компьютерах ограничен величиной 446 байт. Этого явно недостаточно для того, чтобы включить в этот загрузчик драйверы всех периферийных устройств (например, дисплея и устройств хранения данных), которые могут понадобиться на этапе начальной загрузки системы. Тем более, что драйверы могут различаться для однотипных устройств от разных производителей. Поэтому функция загрузки некоторых драйверов возлагается на BIOS.

В составе исполняемого кода BIOS имеется утилита Setup, которая позволяет выполнить некоторые действия по конфигурированию аппаратных средств компьютера. Утилиту Setup обычно можно вызвать, если во время процесса самотестирования нажать указанную на экране клавишу. Параметры конфигурирования, установленные с помощью этой утилиты, запоминаются в энергонезависимой памяти, питаемой от миниатюрного аккумулятора, размещенного на материнской плате. Утилиту Setup можно использовать для некоторых настроек, но мы на этом не задерживаемся. После завершения процедуры самотестирования та часть кода BIOS, которая реализует процедуры самотестирования (POST), удаляется из оперативной памяти за ненадобностью. Оставшаяся часть BIOS,

реализующая функции BIOS (runtime services), остается в ОП. Она будет доступна в последующем для загруженной операционой системы.

Первой задачей той части BIOS, которая осталась в ОП, является поиск активного загрузочного устройства. Список устройств, которые могут являться загрузочными, хранится в энергонезависимой памяти компьютера (CMOS), а порядок просмотра этого списка является одним из настраиваемых параметров BIOS. Загрузочным устройством может быть дискета, CD-ROM, раздел жесткого диска, сетевое устройство или даже USB-устройство (флеш-диск). Поиск загрузочного устройства осуществляется путем вызова прерывания INT19h BIOS.

Процедура обработки прерывания INT19h состоит в том, что считывается сектор с координатами Cylinder:0 Head:0 Sector:1 на очередном устройстве, его содержимое помещается в ОП по адресу 0000:7C00h, после чего осуществляется проверка, является ли этот сектор загрузочным, то есть содержит ли он код первичного загрузчика. Загрузочные сектора помечаются "волшебным" числом 0x55AA в позиции 0x1FE = 510. Это последние два байта сектора. Наличие (или отсутствие) такого кода в последних байтах сектора позволяет программе BIOS решить, является ли данное устройство загрузочным.

Как только программа обработки прерывания обнаружит, что загруженный в память сектор содержит это самое "магическое число" 0x55AA, управление передается на начало этого сектора (по абсолютному адресу 0000:7C00h). Дальнейшие события зависят от того, где обнаружен загрузочный сектор - на жестком диске или на одном из других устройств: дискете, CD или flash-диске (на всех этих устройствах обычно создается образ загрузочной дискеты).

Если при проверке загрузочный сектор не обнаружен ни на одном устройстве, вызывается прерывание INT18h. Когда-то (в первых персональных компьютерах, производимых компанией IBM) это прерывание служило для вызова интерпретатора ROM-BASIC, который дальше управлял работой компьютера. Клоны IBM-PC не имеют BASIC в ROM-памяти и теперь это прерывание используют для организации загрузки по сети. Но мы не будем рассматривать эту ветку развития событий и вернемся к тому случаю, когда прерывание INT19h обнаружило загрузочный диск и передало управление находящейся в нем программе.

Примечание: Как следует из приведенного выше описания, BIOS выполняет массу работы по тестированию системы. Как мы увидим чуть позже, ядро Linux потом повторно проделывает всю эту работу. Как правило, после загрузки ядра большинство функций BIOS

не используется (хотя есть некоторые исключения) и, тем не менее, этот уже бесполезный код BIOS сохраняется в "теневой" памяти компьютера, отнимая часть драгоценного ресурса у работающей системы.

2: Master Boot Record и Boot Record

Итак, BIOS нашел загрузочное устройство и передал управление программе, которая находилась в самом первом секторе этого диска или дискеты (физический адрес: цилиндр 0, головка 0, сектор 1). Теперь эта программа загружена в память и именно она управляет ходом дальнейшей загрузки. Со времен MS-DOS эту программу принято называть загрузочной записью (Boot Record), а первый сектор любого диска или дискеты - загрузочным сектором (Boot Sector).

Примечание: На всех стандартно отформатированных дискетах, даже на тех, которые не являются загрузочными, Boot Sector содержит исполняемый код. Этот код необходим хотя бы для выдачи на экран информационого сообщения. Текст этого сообщения различается в зависимости от того, в какой операционной системе форматировалась дискета. В MS-DOS это было сообщение "Non-system disk or disk error" (вероятно знакомое вам). Жесткий диск, отформатированный в NT (несистемный), сообщал "NTLDR is missing".

Как известно, размер сектора на устройствах долговременного хранения данных для IBM-совместимых компьютеров равен всего-навсего 512 байтам. В принципе существует возможность записать в эти 512 байт программу, обеспечивающую загрузку ядра операционной системы. Как сказано в книге Т.Айвазяна, существует три варианта загрузки ядра Linux:

- с помощью загрузочного сектора Linux, загружающего непосредственно ядро
- с помощью специального загрузчика типа LILO или GRUB;
- с помощью программ, загружающих Linux из другой ОС (например, MS-DOS или Windows), типа loadlin и т.п.

Мы будем далее рассматривать только второй вариант, как наиболее распространенный.

С появлением жестких дисков большого объема, которые стали разбивать на разделы, небольшой загрузчик, размещаемый в загрузочном секторе и загружающий непосредственно ядро, перестал справляться с возросшим объемом задач. Ведь теперь надо было не просто загрузить файл с определенного физического адреса, а вначале найти загрузочный раздел. К тому же часть места, отведенного для загрузочной записи, отняла

таблица разделов жесткого диска. Поэтому старая загрузочная запись была перенесена в первый сектор так называемого "активного" раздела, а в самый первый сектор на жестком диске стали записывать другую программу, задачей которой было найти "активный" раздел и загрузить программу из этого раздела. Первый сектор жесткого диска стали называть главным загрузочным сектором (а соответствующую программу - главной загрузочной записью или Master Boot Record, MBR). На жёстком диске MBR находится по тому же физическому адресу, что и BOOT-сектор на дискете (цилиндр 0, сторона 0, сектор 1). Его структура представлена в табл. 1

***Таблица 1**. Структура главного загрузочного сектора.*

Смещение	**Размер**	**Содержание**
0x000	446 байт	Главная загрузочная запись (Master Boot Record)
0x1BE	64 байта	Таблица разбиения диска
0x1FE	2 байта	"Магическое число" (0x55AA)

"Магическое число" 0x55AA, как мы уже знаем, является признаком того, что диск является загрузочным. Содержащаяся в MBR такого диска таблица разбиения определяет 4 первичных раздела жесткого диска. Первые 446 байт MBR содержат небольшую программу (которая нас в данный момент интересует), а также текст сообщений об ошибках, которые могут возникнуть в ходе ее выполнения.

Основная задача главной загрузочной записи состоит в том, чтобы найти и загрузить в оперативную память собственно загрузчик операционной системы. MBR сканирует таблицу разделов (partition table) в поисках первого (обычно он и единственный) активного раздела (раздела, помеченного как "загрузочный"). Если в таблице разделов активный раздел не обнаружен или хотя бы один раздел содержит неправильную метку, а также если несколько разделов помечены как активные, выдаётся соответствующее сообщение об ошибке. Когда активный раздел найден, программа считывает в оперативную память первый сектор активного раздела.

Примечания: 1. Ход дальнейших действий несколько различается в зависимости от того, с какого устройства считан этот сектор. Мы сейчас будем рассматривать только "основную" ветку - когда первым обнаруженным загрузочным устройством оказался жесткий диск и

загрузка производится с него. Варианты загрузки с дискеты, CD-ROM или flash-диска пока не рассмотриваются. Если будет время и желание, я вернусь к этому вопросу позже, в отдельном разделе.

2. Задание таблицы разделов в MBR (как и использование стандартных BIOS) уже давно ограничивает возможности операцтонных систем. Поэтому фирма Intel разработала Extensible Firmware Interface (EFI, Расширяемый Микропрограммный Интерфейс) - стандарт, призванный придти на смену отжившего свой век PC BIOS. Частью этого стандарта является GPT призван стать стандартным форматом размещения таблицы разделов на физическом жестком диске. Частью этого стандарта является GUID Partition Table (GPT) - новый формат таблицы разделов жесткого диска. GPT призван стать стандартным форматом размещения таблицы разделов на физическом жестком диске.

3. Поскольку загрузочная запись каждого диска содержит исполняемый код, сохраняется возможность заражения загрузочными вирусами компьютера, на котором установлена ОС Linux. Главная загрузочная запись имеет один и тот же формат для всех PC-совместимых компьютеров. И если эта программа содержит код, который считывает самый последний сектор жесткого диска и выполняет содержащийся в нем код, а затем заставляет систему выполнить код из первого сектора активного раздела, вы вряд ли заметите сразу, что что-то идет не так, как надо. Вирусы, которые ведут себя подобным образом, принято называть "загрузочными", именно потому, что они размещаются в загрузочной записи и могут нанести какой-то вред вашей системе только в том, случае, если вы попытаетесь загрузиться с зараженного диска. Этот тип вируса может воздействовать на любой PC-совместимый компьютер. Некоторые типы компьютеров (точнее, некоторые типы BIOS) обеспечивают защиту от данного вида вирусов тем, что запись в загрузочный сектор может быть запрещена. Однако эта мера позволяет защититься только от старых типов загрузочных вирусов, поскольку более новые научились изменять CMOS, разрешая запись в MBR. Что касается Linux-компьютеров, то опасность "подцепить" загрузочный вирус существует только в том случае, если по какой-то причине начинается загрузка с дискеты (скорее всего, случайно оставленной в дисководе). То есть, необходимо быть внимательным и при включении компьютера позаботиться о том, чтоб в дисководе не было дискет.

Практически все загрузчики современных операционных систем состоят из двух частей: загрузчика первого этапа (или первичного загрузчика), который имеет достаточно малый размер, чтобы разместиться в загрузочном секторе, и значительно большего по объему загрузчика 2-го этапа (или вторичного загрузчика), который может храниться уже где

угодно на загрузочном носителе, обычно в разделе, содержащем корневую файловую систему. Загрузчик первого этапа может быть размещен как в главном загрузочном секторе диска, так и в загрузочном секторе активного раздела. Если вам приходилось устанавливать Linux, вы знаете, что программа инсталляции предоставляет пользователю такой выбор. Будем пока для определенности считать, что загрузчик 1-го этапа мы поместили в первый сектор активного раздела.

Итак, главная загрузочная запись отыскала активный раздел и загрузила из него в память загрузчик первого этапа, который теперь отвечает за продолжение процесса загрузки. Загрузчик первого этапа имеет такой же небольшой размер, как и код загрузочной записи диска, то есть не более 446 байт. Поэтому и сделать он может не больше, а именно - только загрузить основной загрузчик. Еще более затрудняет ситуацию то, код загрузчика первого этапа пока не имеет доступа к файловой системе и, следовательно, определяет расположение программ на диске, используя только информацию о физических секторах и низкоуровневые вызовы BIOS. Поэтому эта крохотная программа имеет единственной целью своего существования загрузку и запуск на выполнение загрузчика второго этапа.

Дополнение. В случае, когда первичный загрузчик от Linux установлен не в MBR, а в загрузочный сектор одного из разделов, в MBR, скорее всего осталась программа первичной загрузки, создаваемая программой FDISK от Microsoft. Поэтому приведу в качестве дополнения описание того, как работает первичный загрузчик Windows-систем, которое помогает понять, как происходит процесс загрузки.

Код, содержащийся в главной загрузочной записи, выполняет следующие действия:

- сканирует таблицу разделов (partition table) в поисках первого (обычно он и единственный) активного раздела (раздела, помеченного как "загрузочный"). Если в таблице разделов активный раздел не обнаружен или хотя бы один раздел содержит неправильную метку, а также если несколько разделов помечены как активные, выдаётся соответствующее сообщение об ошибке;
- когда активный раздел найден, программа запоминает его физическое размещение на диске, используя запись в соответствующей строке таблицы разделов диска;
- затем программа первичного загрузчика перемещает свой код по адресам от 0600 до 07FF (если вы помните, первоначально она размещалась по смещению 7C00) и продолжает выполняться оттуда;
- код загрузочного сектора раздела (это уже код загрузчика файловой системы, размещенной в этом разделе) загружается в память по смещению 7C00 (до 7DFF).

- управление передается загрузчику файловой системы (по адресу 7C00).

В MS DOS программа из загрузочного сектора активного раздела просматривает блок параметров BIOS в поисках расположения корневого каталога, а затем копирует из него в память системный файл IO.SYS (который, по сути, является частью DOS) и передаёт ему управление.

В системе Windows NT код из загрузочного сектора активного раздела ищет расположение на диске файла NTLDR, делая это следующим образом:

- Ищет блок параметров BIOS (the BIOS Parameter Block) и расширенный блок параметров BIOS (Extended BIOS Parameter block). Использует эти данные для того, чтобы найти NTLDR на *первом* ("загрузочном") диске.
- Загружает NTLDR в память.
- Запускает его.

Если NTLDR не найден, выдается сообщение "Could not find NTLDR".

Обратите внимание! Местоположение NTLDR ищется на первом диске, даже если код загрузочного сектора хранится на другом диске (что может иметь место в тех случаях, когда используется MBR не от Microsoft). Хотя это выглядит достаточно просто, следствием того факта, что NTLDR ищется на первом диске, является то, что если вы пытаетесь установить NT на второй диск и процедура инсталляции не находит на первом диске свободного места для установки загрузочного сектора, файла NTLDR и других файлов (что может иметь место, если на первом диске уже установлена другая ОС, которую NT не может распознать), будет выдано примерно такое сообщение об ошибке: "xxxx MB disk0 at id0 on bus0 on atapi does not contain a partition suitable for starting Windows NT". (Для SCSI-дисков слова немного отличаются, но суть та же). Это сообщение можно перевести на более понятный язык примерно следующим образом: "Были использованы данные из блока параметров BIOS, но загрузка с того раздела диска, на который указывает ссылка, невозможна".

3: Загрузчик 2 этапа операционной системы

3.1. Задачи, решаемые загрузчиком

Итак, отработали маленькие программы, которые расположены в главной загрузочной записи диска (MBR) и в первом секторе активного раздела (Boot Record), то есть первичный загрузчик ОС. Код первичного загрузчика уже различен в разных операционных системах и

является, как уже было сказано, частью загрузчика этой ОС (ну, или какого-то универсального загрузчика, типа GRUB). В некоторых дистрибутивах используются собственные загрузчики, например, в ASPLinux это ASPLoader. Однако рассказ о всех видах загрузчиков не является нашей целью, поэтому мы кратко рассмотрим только два из них - LILO и GRUB. LILO (LInux LOader) - это "родной" загрузчик операционной системы Linux, который, однако в последнее время все более вытесняется универсальным загрузчиком GRUB (GRand Unified Bootloader). Поскольку нас интересует не создание собственного загрузчика, а всего лишь настройка процедур загрузки системы, мы рассмотрим эту проблему на примере двух названных загрузчиков.

Главная задача, которая стоит перед загрузчиком любой операционной системы, заключается в том, чтобы перенести в память ядро операционной системы и передать этому ядру управление дальнейшим функционированием компьютера. При этом современные загрузчики позволяют выбрать ядро из нескольких возможных вариантов как одной и той же операционной системы, так и загрузить ядра разных ОС. Поскольку загрузчик второго этапа представляет собой уже досточно большую программу, ему это вполне под силу. Более того, некоторые загрузчики (в частности, GRUB) предоставляют пользователю возможность выполнения некоторых команд, то есть они представляют собой уже некоторое подобие командного процессора.

Для того, чтобы загрузить в оперативную память ядро, загрузчик второго этапа должен решить несколько сопутствующих проблем. Первая состоит в том, что код ядра размещается в виде файла на одном из физических носителей. В работающей системе доступ к файлу предоставляется через драйвер файловой системы. А как загрузчику получить доступ к этому файлу? LILO и GRUB решают эту задачу по-разному. Эти решения будут рассмотрены чуть позже, а пока вернемся к перечню проблем, стоящих перед загрузчиком.

Одна из основных задач ядра – управление тем «железом», которое составляет аппаратную часть компьютера. Поэтому ядро после запуска производит опрос «железа» с целью определить, какие устройства присутствуют в системе. Если драйвер найденного устройства вкомпилирован в ядро, устройство инициализируется. Однако маловероятно, что стандартное ядро, заготовленное разработчиком дистрибутива, содержит драйверы для всех устройств, имеющихся в вашем компьютере. Для этого разработчикам пришлось бы создать ядро огромного размера, причем большая часть кода такого ядра была бы простым балластом, поскольку в реальных системах никогда бы не использовалась. Значит, драйверы устройств, отсутствующие в ядре, надо загрузить из соответствующих файлов.

После загрузки ядро должно запустить процесс **init**, который, как известно, является родональником всех остальных процесов в системе. Код программы **init** лежит где-то на носителе, доступ к которому тоже предоставляет файловая система.

Но файловая система не является частью ядра, драйвер файловой системы надо загрузить с того же диска. Когда-то вариантов выбора для корневой файловой системы в Linux, можно сказать, не было. И размещалась она либо на дискете, либо на одном из разделов жесткого диска. Поэтому достаточно было указать загрузчику местоположение корневой файловой системы (что достигается заданием параметра типа "root=/dev/hda2") и проблема решалась. В наше время загрузочным устройством может оказаться устройство самого разного вида: SCSI, SATA, RAID-массивы, CD- или DVD-ROM, USB (причем USB-портов может быть несколько, надо еще выбрать, которое из этих устройств загрузочное). Корневая файловая система на загрузочном устройстве может быть самого разного типа (fat, ext2, reiserfs и так далее), может быть сжата, зашифрована и так далее. Возможно, что она даже расположена на сервере сети, ядру требуется провести присвоение адреса по DHCP, произвести DNS поиск и установить соединение с удаленным сервером (с именем и паролем), все это до того, как ядро сможет найти и выполнить первую программу пространства пользователя. Ситуация усугубляется еще и тем, что драйверы некоторых устройств могут конфликтовать друг с другом, так что загрузить их в память одновременно (например, если они включены в одно и то же ядро) невозможно.

Получается замкнутый круг: чтобы подключить загрузочное устройство, надо получить доступ к файловой систем, а для этого надо подключить загрузочное устройство.

Решение этой проблемы было найдено за счет временного подключения виртуальной файловой системы, образ которой хранится в файле рядом с образом ядра. При этом минимальное число самых необходимых драйверов включается в само ядро, а все остальные размещаются на виртуальном диске, в виде подгружаемых модулей. Когда осуществляется загрузка (в частности, на неизвестном оборудовании) ядро опрашивает аппаратуру и загружает только те модули, которые соответствуют обнаруженной конфигурации. Впервые такое решение было применено в ядре версии 1.3.73, и с некоторыми модификациями используется до настоящего времени как для запуска процедуры инсталляции, так и в процессе штатной загрузки системы. В частности, и LILO и GRUB создают в оперативной памяти виртуальный диск и разворачивают на нем временную корневую файловую систему, содержащую необходимые ядру драйверы устройств и служебные файлы. Сжатый образ этой файловой системы обычно располагается в одном каталоге с образом ядра в виде файла, содержащего дополнительные

компоненты, например, драйверы некоторых устройств. Только способ организации виртуального диска в ядрах серий 2.4.x и 2.6.x различен. Но эти различия мы рассмотрим в следующем подразделе.

Еще одной задачей, выполняемой загрузчиком второго этапа, является передача ядру значений аргументов, которые пользователь пожелал переопределить. К числу таких аргументов относятся, например, уровень выполнения или имя программы, которую ядро должно запустить вместо запускаемого по умолчанию процесса **init**. Вообще говоря, обычно передавать ядру какие-либо параметры и не требуется - принятые по умолчанию значения хорошо срабатывают в большинстве случаев. Изменять эти значения приходится тем пользователям, которые вынуждены оптимизировать ядро под их конкретную машину, или тем, кто создает свое собственное ядро для поддержки редкого оборудования, автоматическое определение которого по тем или иным причинам невозможно.

Чтобы понять, каким образом загрузчик решает перечисленные проблемы, давайте заглянем в каталог **/boot** в дереве каталогов вашей файловой системы. Вы знаете, что этот каталог содержит файлы, используемые в процессе загрузки. Можете вы объяснить назначение всех файлов, находящихся в этом каталоге? Если вы ответите «да», то вам не стоит тратить время на чтение моих заметок. А всех остальных я приглашаю просмотреть содержимое этого каталога. Оно может несколько отличаться о того, что я увидел, например, на компьютере с ASP Linux 11:

Листинг 1. Содержимое каталога /boot

```
 [kos@trend /boot] # ls -l
total 3499
-rw-r--r--  1 root root   62293 Jan 23  2006 config-2.6.14-1.1653.1asp
drwxr-xr-x  2 root root    1024 Aug 31  2006 grub
-rw-r--r--  1 root root 1101106 Apr 29  2006 initrd-2.6.14-1.1653.1asp.img
drwx------  2 root root   12288 Apr 29  2006 lost+found
lrwxrwxrwx  1 root root      29 Apr 29  2006 System.map -> System.map-2.6.14-1.1653.1asp
-rw-r--r--  1 root root  806603 Jan 23  2006 System.map-2.6.14-1.1653.1asp
lrwxrwxrwx  1 root root      26 Apr 29  2006 vmlinuz -> vmlinuz-2.6.14-1.1653.1asp
-rw-r--r--  1 root root 1569945 Jan 23  2006 vmlinuz-2.6.14-1.1653.1asp
```

Как видите, тут только 4 файла (подкаталоги и символические ссылки не считаем): **config-2.6.14-1.1653.1asp**, **initrd-2.6.14-1.1653.1asp.img**, **System.map-2.6.14-1.1653.1asp** и **vmlinuz-2.6.14-1.1653.1asp**.

Файл vmlinuz-2.6.14-1.1653.1asp представляет собой образ ядра, которое загрузчик должен перенести в оперативную память. В принципе здесь может размещаться несколько разных файлов с образами ядер. Но это в том случае, если вы проводили обновление системы или самостоятельно ставили какое-то ядро. Символическая ссылка с именем vmlinuz указывает на то ядро, которое будет запускаться.

Файл config-2.6.14-1.1653.1asp для нас пока интереса не представляет - в нем сохранены параметры и опции, которые были использованы при компиляции этого ядра. Он может пригодиться, если вы вздумаете перекомпилировать ядро под свое оборудование. В таком случае этот файл может несколько облегчить процедуру компиляции.

А вот два оставшихся файла непосредственно используются в процессе загрузки, поэтому мы должны поговорить о них подробнее. И начнем с рассмотрения файла **/boot/initrd-2.x.yy-что-то-там.img**.

3.2. Файл /boot/initrd и его назначение

Как было сказано в предыдущем разделе, в процессе начальной загрузки вначале монтируется временный виртуальный диск, содержащий корневую файловую систему (очевидно, тоже временную), с помощью которой осуществляется запуск на выполнение ядра операционной системы. Образ этой корневой системы хранится на загрузочном устройстве в каталоге **/boot** и обычно носит имя **initrd-2.x.yy-zzzz** (точное наименование файла смотрите в вашей системе, далее он будет именоваться просто **initrd**). Загрузчик переносит образ корневой файловой системы из файла **initrd** в оперативную пямять одновременно с образом ядра, ядро преобразует **initrd** в "нормальный" RAM диск, монтирует временную корневую файловую систему, извлекает из нее нужные драйверы и другие служебные файлы, с помощью которых инициализирует все необходимые устройства. После этого происходит перемонтирование корневой файловой системы, временная ФС заменяется на постоянную, размещающуюся уже на долговременном носителе и процесс загрузки системы продолжается.

После того, как будет смонтирована корневая файловая система с жесткого диска, файл **initrd** уже никак не используется и место, занимаемое им в памяти, тоже освобождается. А теперь давайте подробнее рассмотрим как устроен этот файл. При этом

надо сразу сказать, что внутреннее устройство **initrd** и его применение в ядрах версий 2.6 было изменено по сравнению с ядрами версий 2.4, так что придется рассмотреть эти случаи по отдельности.

3.2.1. Внутреннее устройство initrd

В дистрибутиве Fedora Core 3 (и его предшественниках) файл **initrd** создавался с использованием виртуального loop-устройства. В этом случае сделать содержимое файла **initrd** доступным для просмотра можно с помощью следующих команд (напомню, что имя вашего образа **initrd** может отличаться):

Листинг 2. Просматриваем **initrd** в варианте loop-устройства (до FC3)

```
# mkdir temp ; cd temp
# cp /boot/initrd.img.gz .
# gunzip initrd.img.gz
# mount -t ext -o loop initrd.img /mnt/initrd
# ls -la /mnt/initrd
```

После этого вы можете посмотреть содержимое **initrd** в директории /mnt/initrd.

Заметьте что даже если имя вашего образа **initrd** не заканчивается на ".gz", это все равно сжатый файл и добавив к концу имени суффикс ".gz", вы сможете воспользоваться командой **gunzip**, чтобы его распаковать.

В таких дистрибутивах, как Fedora Core 4, SUSE 10, ... файл **initrd** представляет собой cpio-архив, сжатый упаковщиком gzip. Чтобы просмотреть его содержимое, надо выполнить команды, приведенные в листинге 3

Листинг 3. Просматриваем **initrd** в варианте cpio-архива (FC3 и позже)

```
mkdir ~/tmp
cd ~/tmp
cp /boot/initrd-2.6.16.13-4-default initrd-2.6.16.13-4-default.gz
gzip -d .gz
mv initrd-2.6.16.13-4-default initrd-2.6.16.13-4-default.cpio
cpio -i < initrd-2.6.16.13-4-default
```

После этого в каталоге ~/tmp вы найдете каталоговую структуру, аналогичную структуре каталогов обычной Linux-системы. В ней вы обнаружите обычные для корневого каталога подкаталоги, а также файл **linuxrc**. Могут там оказаться и другие файлы, например, файл **bootsplash**(судя по названию он содержит картинку, которая отображается при запуске).

Познакомившись с внутренним содержанием файла **initrd** можно вернуться к рассмотрению его роли в процессе загрузки.

3.2.2. Загрузка системы в случае использования initrd (ядра версий 2.4.x)

1. Загрузчик загружает ядро и содержимое **initrd** в память *(очевидно, по вполне определенным адресам ???)*, после чего передает управление ядру.

2. Ядро содержит в себе небольшую несжатую часть, которая вначале разархивирует само ядро. Видимо как раз на этом этапе на экране монитора отображается сообщение "Uncompressing Linux... Ok, booting the kernel."

3. Ядро инициализирует устройства, создает файловую систему устройств **/dev**, разархивирует **initrd** и копирует его содержимое на устройство **/dev/ram0**, а затем освобождает память, занятую **initrd**.

4. Ядро монтирует устройство **/dev/ram0** для чтения и записи в качестве начальной корневой файловой системы.

5. Если в начальной корневой файловой системе находится исполняемый файл **/linuxrc**, он исполняется с uid 0. Этот файл должен иметь разрешения на исполнение, он может быть как обычным исполняемым файлом, так и просто скриптом оболочки, но в последнем случае должен быть и интерпретатор скриптов.

6. Скрипт **/linuxrc** монтирует нормальную корневую файловую систему.

7. Корневая файловая система помещается в корневую директорию. Смена корневого устройства выполняется системным вызовом pivot_root, который также доступен через утилиту pivot_root (см. pivot_root(8); pivot_root распространяется в составе util-linux версии не ниже 2.10h). Смена корневой директории не включает её демонтирование. Следовательно, при выполнении данной процедуры можно сохранить процессы, запущенные с файловой системы **initrd**.

8. Если нормальная корневая файловая система имеет каталог **/initrd**, то устройство **/dev/ram0** перемещается из **/** в **/initrd**. Иначе, если каталог **/initrd** не существует, устройство **/dev/ram0** размонтируется.

9. На нормальной корневой файловой системе следует обычная процедура загрузки (например, вызов **/sbin/init**).

10. Теперь можно демонтировать **/dev/ram0** и освободить память, занятую RAM диском.

3.2.3. Initramfs - новая модель инициализации (ядра версий 2.6.x)

Все сказанное выше относительно процедур загрузки и использования файла **initrd** справедливо для ядер версии 2.4.xx и более ранних. В ядрах версии 2.6 (а точнее, начиная с экспериментального ядра версии 2.5.46) разработчики решили реализовать иной механизм. Причиной послужили несколько недостатков ранее использовавшегося варианта реализации временной корневой файловой системы.

Во-первых, виртуальный диск, как и все блочные устройства, требует драйвер файловой системы для интерпретации данных во время выполнения. Этот драйвер приходилось включать в ядро.

Во-вторых, неэфективно используется оперативная память, так как размер виртуального диска фиксирован и не может изменяться во время работы без его переформатирования (даже если диск не заполнен, невозможно отдать эту память под другие нужды).

В третьих, в Linux осуществляется кэширование всех файлов и записей каталогов, прочитанных или записанных на блочное устройство. Виртуальный диск тоже кешируется, как и обычные диски, то есть часть данных будет храниться не только на RAM диске, но и в страничном кэше "page cache" (для файловых данных) и в кэше для записей каталогов "dentry cache", что еще больше снижает эффективность использования памяти.

Для устраненеия этих недостатков Линус Торвальдс предложил монтировать кэш ядра Линукс как особую файловую систему, работающую полностью в кэше ядра. При этом нет лишнего дублирования информации между блочным устройством и кэшем, так как блочного устройства попросту нет. Файловая система, реализующая эти идеи, была разработана и получила название **initramfs**. Ее использование имеет следующие преимущества:

- Система, использующая **initramfs** в качестве корневой файловой системы, более не нуждается в соответствующем драйвере файловой системы, встроенном в ядро, так как нет блочных устройств для интерпретации файловых систем.
- Размер этой файловой системы автоматически изменяется в соответствии с обьёмом данных, которые она содержит. При добавлении новых файлов (как и при

расширении существующих) автоматически выделяется память, при удалении или уменьшении файла происходит высвобождение памяти.

- Достоинством **initramfs** является также то, что это не новый код, а новое применение уже существующего кода кэширования ядра Линукс, что практически не влечёт увеличение размера ядра, выполнение будет очень простым и основано на чрезвычайно хорошо протестированной инфраструктуре.
- Исчезают некоторые проблемы загрузки с SATA-дисков.
- **Initramfs** загружается немного быстрее, чем **initrd**.

Формат **initramfs** используется по умолчанию для всех ядер, начиная с версии 2.6.15.

В каталоге **/boot** такая файловая система хранится, по-прежнему, в файле в виде сжатого cpio-архива. Только имя включаемого в этот архив скрипта изменилось с **linuxrc** на **init**. Например, в системе ASP Linux 11 после разархивирования файла initrd-2.6.14-1.1653.1asp.img по приведенному в листинге 3 алгоритму я увидел скрипт **init** и следующую структуру каталогов:

/bin

/dev

/etc

/lib

/loopfs

/proc

/sbin -> bin

/sys

/sysroot

В подкаталоге /bin обнаружилось всего 4 исполняемых файла и две ссылки, в подкаталоге /dev - 8 файлов устройств, в /etc - конфигурационный файл udev/udev.conf, в каталоге lib - два файла: ext3.ko и jbd.ko. Остальные подкаталоги - пустые. Файл **init** оказался скриптом оболочки следующего вида:

Листинг 4. Содержимое файла **init** в дистрибутиве ASP Linux версии 11.

```
#!/bin/nash
```

```
mount -t proc /proc /proc
setquiet
echo Mounted /proc filesystem
echo Mounting sysfs
mount -t sysfs /sys /sys
echo Creating /dev
mount -o mode=0755 -t tmpfs /dev /dev
mknod /dev/console c 5 1
mknod /dev/null c 1 3
mknod /dev/zero c 1 5
mkdir /dev/pts
mkdir /dev/shm
echo Starting udev
/sbin/udevstart
echo -n "/sbin/hotplug" > /proc/sys/kernel/hotplug
echo "Loading jbd.ko module"
insmod /lib/jbd.ko
echo "Loading ext3.ko module"
insmod /lib/ext3.ko
/sbin/udevstart
echo Creating root device
mkrootdev /dev/root
echo Mounting root filesystem
mount -o defaults,errors=remount-ro --ro -t ext3 /dev/root /sysroot
echo Switching to new root
```

```
switchroot --movedev /sysroot
```

Обратите внимание на первую строку этого файла: это скрипт для оболочки **nash**, которая является сильно урезанной версией командного процессора и тоже включена в состав **initramfs**. Другие, более мощные оболочки на этом этапе еще недоступны.

Как видно из листинга 4, скрипт **init** монтирует файловые системы /sys и /dev, инициализирует несколько устройств, загружает модули из подкаталога /lib и монтирует корневую файловую систему

Образ виртуальной файловой системы **initramfs** хранится на диске по-прежнему в файле **/boot/initrd-version** и создается такой файл как и ранее командой **mkinitrd**. Перечень модулей (иначе говоря, драйверов), которые будут включены в эту файловую систему, приведен в переменой **INITRD_MODULES** в файле **/etc/sysconfig/kernel**. Если вам требуется включить в **/boot/initrd-version** какие-то дополнительные модули (например, в случае изменения состава оборудования), вы можете скорректировать соответствующим образом список модулей в файле **/etc/sysconfig/kernel**, после чего заново создать файл **/boot/initrd-version**.

3.3. Загрузчик GRUB

Как уже говорилось выше, в последнее время вместо LILO все чаще используется GRUB (the GRand Unified Bootloader). Его первоначальная версия была разработана Э.Болейном (Erich Boleyn), а широкая популярность пришла к нему после того, как к разработке подключились Gordon Matzigkeit и Okuji Yoshinori (не решаюсь написать их имена в русской транскрипции). В настоящее время GRUB является частью проекта GNU.

Преимущества GRUB обусловлены тем, что он реализует спецификацию многовариантной загрузки (Multiboot Specification) и изначально (в отличие от LILO) разрабатывался с ориентацией на загрузку различных операционных систем. Однажды установив GRUB на жесткий диск вашего компьютера вы уже не должны будете переустанавливать его после замены версии ядра или внесения изменений в существующее ядро. Изменение конфигурации системы, добавление новых вариантов загрузки осуществляется просто редактированием конфигурационного файла. При этом GRUB умеет работать с некоторыми самыми распространенными файловыми системами, что позволяет обеспечить загрузку ядра по имени файла, не прибегая к указанию физического адреса ядра на диске. Но самое существенное - это означает, что для GRUB не существует проблемы больших дисков. Кроме того, GRUB предоставляет пользователю небольшую bash-подобную оболочку, в которой можно выполнить ряд команд, не загружая систему полностью (впрочем, эту

оболочку можно вызвать и после загрузки системы из командной строки bash). Как и LILO последних версий, GRUB может работать через текстовый интерфейс, либо использовать интерфейс в виде меню.

Поскольку нас интересует в основном настройка процедур загрузки, я не буду здесь рассказывать о том, как установить GRUB (скорее всего он и так уже установлен в вашей системе), а перейду сразу к вопросу его конфигурирования. Однако вначале придется остановиться на вопросе о том, как GRUB именует диски и другие устройства, поскольку его подход к этому вопросу отличается от системы имен, принятой в Linux (я слышал, что в будущих версиях планируется изменить систему именования устройств, но пока точных данных на этот счет не имею).

Как GRUB именует диски?

GRUB использует следующий формат имен дисковых разделов:

(hd***n***,***p***)

где ***n*** - это номер диска (нумерация начинается с 0), а ***p*** - номер раздела на диске (опять же, начиная с 0). Например, второй раздел на третьем жестком диске будет обозначаться как:

(hd2,1)

Для операционных систем, которые используют разбиение разделов диска на подразделы (как, например, во FreeBSD) добавляется еще третье поле - буквенное обозначение подраздела:

(hd0,0,a)

Обратите внимание на то, что запятая и скобки обязательны - это неотъемлемая часть названия раздела жесткого диска в GRUB. Как правило, IDE диски в нумерации предшествуют SCSI-дискам, то есть под нулевым номером идет первый диск (Master) на первом IDE-контроллере, далее - второй диск (Slave) на первом контроллере и так далее. SCSI-диски следуют по порядку за последним IDE-диском.

Где размещается GRUB?

Если все файлы загрузчика LILO обычно располагаются в каталоге **/boot**, то GRUB размещается "этажом ниже", в каталоге **/boot/grub**. Давайте заглянем в этот каталог и посмотрим, что там находится.

Первым делом обратите внимание на файлы **stage1** и **stage2**. Это и есть две основных части загрузчика, о которых мы говорили выше. Файл **stage1** - это первичный загрузчик, который

записывается в загрузочный сектор (проверьте - размер это файла равен 512 байтам), а**stage2** - вторичный загрузчик (посмотрите размер этого файла), задача которого - загрузить ядро и файл **initrd**.

Далее следует обратить внимание на группу файлов, названия которых оканчиваются на **stage1_5**:

reiserfs_stage1_5

e2fs_stage1_5

iso9660_stage1_5

ufs2_stage1_5

fat_stage1_5

jfs_stage1_5

vstafs_stage1_5

ffs_stage1_5

minix_stage1_5

xfs_stage1_5

Эти файлы - "изюминка" GRUB. Насколько я понимаю, в каждом из них содержится что-то вроде драйвера одной из файловых систем. Их называют загрузчиками полуторного этапа. Их назначение состоит в том, чтобы обеспечить загрузчику второго этапа возможность читать последующие данные средствами, предоставляемыми файловыми системами. Именно полуторный загрузчик обеспечивает основные преимущества GRUB:

- реконфигурация загрузчика не требует перезаписи главной загрузочной записи;
- предназначенные для загрузки ядра адресуются обычным для файловых систем способом;
- физические операции над содержимым диска (дефрагментация, перенос файлов), никак не отражаются на работе загрузчика;
- логическая адресация объектов загрузки делает возможным управление ими не только по заранее определённому сценарию, но и в интерактивном режиме. И возможность эта реализована в виде командного режима работы GRUB.

Кроме основных модулей **stage1** и **stage2** может присутствовать еще несколько, более специфических: **stage2_eltorito** для установки GRUB на компакт-диск, **nbgrub** и **pxegrub** для сетевой загрузки.

Задача **stage1** в том, чтобы загрузить **stage2** или **stage1_5** с диска. При этом, размещение этих файлов кодируется в терминах цилиндр/головка/сектор, и значит на этом этапе еще не требуется распознавание типа файловой системы.

После загрузки **stage1_5** и **stage2** GRUB получает доступ к файловой системе на диске, а значит, получает возможность не только загрузить ядро, но и прочитать конфигурационный файл, найти и вывести фоновую картинку-заставку (она хранится в файле splash.xpm.gz), может выполнить команды своей оболочки.

Стоит отметить, что если запись **stage1_5** по каким-либо причинам недоступна, то вторичный загрузчик может найти ядро по его физическому адресу. То есть, в этом случае GRUB использует методологию своих предшественников типа LILO.

В каталоге **/boot/grub** находится еще файл **device.map**, который содержит соответствия между именами дисков, применяемыми GRUB, и именами дисков, которые понятны операционной системе. Например, у меня этот файл состоит из одной строки

(hd0) /dev/hda

Необходимость наличия этого файла обусловлена тем, что в BIOS имеется возможность поменять последовательность загрузки между IDE и SCSI-дисками. Вы можете при необходимости редактировать этот файл, а также добавить в него свои комментарии, поскольку при обращении к нему оболочка GRUB игнорирует любую строку, начинающуюся символом #.

Последний из файлов, расположенных в каталоге **/boot/grub/**, - это файл **/boot/grub/grub.conf** (или **menu.lst** для некоторых версий), с помощью которого осуществляется конфигурирование GRUB. О том, как осуществляется настройка загрузки посредством этого файла мы сейчас и поговорим.

Конфигурирование GRUB

В листинге 7 приведен пример файла **/boot/grub/grub.conf**, с помощью которого мы и разберем приемы настройки GRUB.

Листинг 7. Пример файла /boot/grub/grub.conf

```
default=0
```

```
timeout=10
splashimage=/grub/splash.xpm.gz

title ASPLinux-2.6.14
        kernel /vmlinuz-2.6.14-1.1653.1asp root=/dev/hda3 reboot=b pci=noacpi pci=usepirqmask quiet rhgb resume=/dev/hda1
        initrd /initrd-2.6.14-1.1653.1asp.img
        boot
        root (hd0,1)
        setup (hd0)

title Windows
        map (hd0,0) (hd0,2)
        map (hd0,2) (hd0,0)
        rootnoverify (hd0,2)
        chainloader +1
```

Вначале обратите внимание на две секции этого файла, начинающиеся со слова "title". Каждая из этих секций предназначена для организации загрузки какой-то из операционных систем, имеющихся на ваших дисках. После слова "title" (в той же строке) стоит название этого варианта (вы можете задать его произвольным образом). Эти названия появятся в виде пунктов меню загрузки.

Вначале рассмотрим первую из этих секций, которая обеспечивает загрузку Linux. Строка этой секции, начинающаяся со слова "root", указывает, какой раздел и на каком диске содержит образ ядра Linux (это может быть как корневой раздел файловой системы Linux, так и любой другой раздел или диск). В нашем случае строка "root (hd0,1)" сообщает GRUB, что файл ядра находится на втором разделе диска /dev/hda. Еще раз обратите внимание на систему именования дисков в GRUB, в которой числа 0-3 используются для обозначения первичных разделов диска, а числа с 4 и больше - для обозначения логических разделов. В данном случае применяется именование дисков способом, принятым в GRUB.

Все ссылки на каталоги и файлы, встречающиеся далее в той же секции конфигурационного файла, являются относительными к разделу, указанному строкой "root (hd0,1)". Поэтому не стоит удивляться, когда в строке, начинающейся словом "kernel" (эта строка сообщает GRUB, где находится ядро) записано

kernel /vmlinuz-2.6.14-1.1653.1asp

хотя в работающей системе вы найдете файл ядра в каталоге "/boot". Просто эти строки могут иметь различный вид в зависимости от того, имеется или нет у вас отдельный раздел для каталога /boot. В моем случае каталог "boot" вынесен в отдельный раздел. Поэтому образ ядра расположен в корне этого раздела. То же самое относится и к строке "initrd", которая содержит указание на местоположение файла образа виртуального диска.

Кроме местоположения ядра в строке, начинающейся словом "kernel", заданы некоторые параметры начальной загрузки, о которых мы поговорим в следующем разделе. Пока что обратите внимание на параметр "root=", имеющийся в этой строке. Как мы увидим ниже, этот параметр задает местоположение коневой файловой системы Linux. В отличие от команды "root (hd0,1)" конфигурационного файла здесь уже используется метод именования дисков, принятый в Linux.

Секция, обеспечивающая загрузку Windows, нас не очень интересует, поэтому скажем о ней очень кратко. Строки, начинающиеся словом "map", в секции, задающей загрузку Windows 98, заставляют Windows полагать, что она установлена в первый раздел первого диска. Строка "rootnoverify" сообщает GRUB, что требуется загрузить систему с Windows-раздела, не пытаясь его смонтировать, а строка "chainloader +1" требует от GRUB передать управление загрузчику Windows.

Теперь давайте вернемся к началу конфигурационного файла, где находятся несколько строк, которые определяют поведение GRUB на этапе выдачи пользователю начального меню загрузки. Эти строки можно назвать секцией глобальных настроек, как это было в случае с LILO.

Команда "default=0" в этой секции определяет вариант загрузки, выбираемый по умолчанию, то есть в том случае, когда пользователь не выбрал (с помощью клавиатуры) какой-то другой вариант. Возможные варианты задаются секциями **title** (смотри выше), причем нумерация начинается с нуля. В примере из листинга 7 имеются только две секции "title" и, следовательно, возможны только два варианта загрузки по умолчанию - "default=0" и "default=1".

Команда "timeout=10" определяет период времени (в секундах) в течение которого GRUB ожидает, пока пользователь выберет нужный вариант загрузки, введет дополнительные команды или задаст нужные параметры загрузки. Если по истечении этого времени пользователь не нажал ни одной клавиши, будет автоматически выбран вариант загрузки, определяемый командой "default=n".

Число различных параметров, которые можно задать в конфигурационном файле GRUB, довольно велико, так что приводить здесь описания каждого из них не имеется возможности. Тем более, что необходимые сведения можно легко найти в многочисленных статьях, доступных в Интернет. Кроме того, самое полное руководство по GRUB (но на английском языке) вы найдете на сайте GNU.ORG. Отметим только еще раз, что в отличие от LILO, GRUB не требует переустановки после внесения изменений в свой конфигурационный файл.

Как происходит загрузка с GRUB

При инсталляции GRUB полуторный загрузчик из файла **fsname_stage1_5** (естественно, соответствующий файловой системе диска) записывается в последовательные сектора жесткого диска, следующие за загрузочным сектором на нулевой дорожке диска. Размер этой области (в байтах) равен числу секторов на дорожке минус 1, умноженному на 512, но полуторный загрузчик достаточно мал (посмотрите размер файла **fsname_stage1_5** на своем диске), чтобы здесь разместиться.

В процессе загрузки системы первичный загрузчик загружает только первый сектор полуторного загрузчика, задача которого состоит в том, чтобы развернуть весь полуторный загрузчик, обеспечив тем самым доступ к файловой системе жесткого диска. После получения управления полуторный загрузчик, в свою очередь, загружает в память вторичный загрузчик.

Как уже упоминалось выше, если полуторный загрузчик не найден, может быть сразу загружен вторичный загрузчик. В таком случае он может самостоятельно подгрузить полуторный загрузчик. Выбор полуторного загрузчика, соответствующего файловой системе диска, осуществляется путем обращения к предопределенному адресу на том диске или разделе, который указан как загрузочный. По этому адресу прописан признак, по которому вторичный загрузчик и выбирает нужный полуторный загрузчик.

Прежде чем перейти к этапу загрузки ядра Linux, надо сказать несколько слов о параметрах начальной загрузки, которые могут быть переданы загрузчику, а через него - ядру и процессу **init**.

4.1. Загрузка и инициализация ядра

В сети можно найти несколько статей, содержащих довольно подробное описание процесса загрузки ядра. Хотя в этих статьях рассматривается ядро версии 2.4 (и даже версии 1.0), я полагаю, что с переходом к ядру версии 2.6 в этом процессе изменилось не многое. Попытаюсь составить из этих источников описание основных действий, выполняемых ядром на этапе его загрузки и инициализации.

Во время загрузки Linux на консоль выводится большое количество сообщений, в том числе и сообщений о действиях, выполняемых ядром при его инициализации. Эти сообщения обычно быстро проскакивают, и вы, вероятно, не сможете прочитать их, несмотря на то, что в некоторые моменты возникают задержки на несколько секунд. А если эти сообщения скрыты за какой-то красивой заставкой, эти сообщения могут накладываться на графический фон или могут быть вообще скрыты. И хотя у вас обычно есть возможность отключить графическую заставку (как это сделать, было рассказано в предыдущем разделе), тем не менее прочитать сообщения на этапе загрузки в любом случае проблематично.

К счастью, все эти сообщения сохраняются и их можно увидеть после завершения загрузки, воспользовавшись командой **dmesg**. Протоколирование этих сообщений осуществляется демоном протоколирования klogd (он запускается как один из потоков ядра). Klogd сохраняет эти сообщения в специальном буфере ядра, содержимое которого и выдается по команде **dmesg**. Кроме того, klogd передает эти сообщения демону системного протоколирования syslogd, который записывает их в файл **/var/log/messages**. Благодаря этому они и доступны для просмотра и анализа после завершения запуска системы. Я попытаюсь дать описание процесса загрузки ядра, используя эти сообщения в качестве ориентиров на пути развития данного процесса (смотрите листинги этих сообщений, взятые с моего домашнего компьютера).

Поскольку демон протоколирования klogd просто не может быть запущен в течение некоторого промежутка времени после передачи управления ядру, самый начальный этап работы ядра не отражен в протоколе.

4.1.1. Получение системной информации от BIOS и разархивация ядра

Ядро хранится на диске в сжатом виде. Только его первая часть не сжата. Когда загрузчик перенесет ядро в память, эта несжатая часть ядра выполняет декомпрессию оставшейся части. Несжатая часть кода, которая содержится в начале ядра, написана на ассемблере, ее можно найти в файлах arch/i386/boot/setup.S, arch/i386/boot/video.S и arch/i386/kernel/head.S.

Код, хранящийся в файлах arch/i386/boot/setup.S и arch/i386/boot/video.S, отвечает за получение системной информации от BIOS и размещение ее в подходящих местах системной памяти. Кратко перечислим основные действия, которые выполняет эта часть ядра:

- Чтение DASD-типа второго жесткого диска (*не знаю, что такое DASD*).
- Проверка того, что ядро загружено правильно. Осуществляется проверкой сигнатуры в конце кода запуска. Если она не найдена, то копируются секторы с запускающим кодом и сигнатура ищется снова. Если опять безуспешно, то выдается сообщение "No setup signature found ...".
- Проверка на возможность работы с верхней памятью. Если образ ядра слишком большой (и поэтому загружен в верхнюю память), а код несжатой части ядра не может работать с образами, расположенными в верхней памяти, то загрузка прекращается и выдачей сообщения "Wrong loader, giving up...".
- Выяснение объема памяти системы. Для этого используется вызов трех различных функций BIOS: E820h, которая позволяет собрать карту памяти, затем E801h, которая вернет 32-битный размер памяти и последней вызывается 88h, которая вернет размер в диапазоне 0-64 МБ. Полученные значения сохраняются для дальнейшего использования.
- Частота повторов клавиатуры устанавливается в максимальное значение при помощи BIOS int 0x16.
- Настройка режима видеоадаптера. Возможность выбора видеорежима зависит от конфигурации ядра. Можно указать специфичный (предопределенный) режим, который будет использован в процессе загрузки ядра, либо запросить меню со списком режимов, из которого пользователь выберет режим по своему желанию.
- Считываются (путем вызова прерываний 0x41 и 0x46) и сохраняются параметры жестких дисков hd0 и hd1 (если есть).
- Считывается информация о шине Micro Channel.
- Проверяется наличие PS/2 мыши при помощи прерывания BIOS 0x11.

- Если ядро сконфигурировано с поддержкой APM (Advanced Power Management - улучшенное управление питанием, реализуемая BIOS спецификация управления питанием для персональных компьютеров), проверяется поддержка APM в BIOS .
- Осуществляется подготовка к переключению в защищенный режим: ядро перемещается в подходящее место (если это необходимо), подправляются адреса и значения регистров.
- Задействуется адресная линия A20 (что-то связанное с клавиатурой, точно не знаю).
- Производится сброс математического сопроцессора.
- Маскируются все прерывания, кроме IRQ2.
- Производится переключение первого процессора в защищенный 32-битный режим.
- Код загрузки в "linux/arch/i386/boot/setup.S" перемещает выполнение к началу кода в "linux/arch/i386/kernel/head.S" (помеченному "startup_32:"). Чтобы эта точка стала доступной, небольшая разжатая функция ядра разжимает оставшийся сжатый ядерный образ и затем переходит к полученному коду.

4.1.2. Задание начальных установок процессора(ов)

Далее начинает работать код из файла arch/i386/kernel/head.S. Этот код выполняет следующие действия (регистры обозначаются как '%регистр', константы записываются как номера с или без '$' впереди):

- Устанавливаются сегментные регистры (%ds = %es = %fs = %gs = __KERNEL_DS = 0x18).
- Для систем с SMP (Symmetric MultiProcessing) производится проверка того, что работа происходит на загружающем процессоре (BSP - Bootstrap Processor). Если это не так, то инициализация таблицы страниц пропускается, осуществляется переход к пункту "Включение управления страницами".
- Инициализируются таблицы страниц.
- Включение управления страницами. Разрешается листание страниц установкой бита PG в %cr0 в состояние "страничная организация задействована".
- Обнуляется BSS (Block Started by Symbol - блок, начинающийся с символа; неинициализированный сегмент данных), для SMP это действие выполняет только первый CPU.
- Параметры загрузки заносятся в первые 2 килобайта по адресу _empty_zero_page, а содержимое командной строки (kernel commandline) - в следующие 2 килобайта.

- Проверяется тип CPU, используя EFLAGS и, если возможно, cpuid, позволяющие обнаружить процессор 386 и выше. Также проверяется наличие сопроцесора 80287 или 80387.
- Загружается дескриптор таблицы указателя регистров и осуществляется переход к __KERNEL_CS:%eip. Теперь процессор работает в защищенном режиме.
- Запуск других процессоров. Вторичный(е) процессор(ы) инициализируется (ются) и переводится(ятся) в режим простоя до тех пор, пока процессы его(их) не займут.
- Первый CPU вызывает функцию start_kernel().

4.1.3. Инициализация ядра

Функция init/main.c:start_kernel() написана на C и выполняет следующие действия:

- Выполняется глобальная блокировка (BKL: big kernel lock - большая блокировка ядра), необходимая для того, чтобы через процесс инициализации проходил только один CPU. Прерывания в это время еще запрещены.
- Вывод "баннера" ядра, который содержит версию, компилятор, использованные при сборке, и пр., в кольцевой буфер для сообщений. Текст "баннера" задается в переменной linux_banner, определенной в init/version.c. Текст этот можно вывести на экран командой **cat /proc/version**. Это фактически первое сообщение, которое выдается ядром. Пример таких сообщений с двух моих компьютеров:

```
Linux version 2.6.14-1.1653.1asp (build@amd64.asplinux.com.ua) (gcc version 4.0.2 20051125 (Red Hat 4.0.2-8)) #1 Mon Jan 23 19:24:56 EET 2006
```

```
Linux version 2.6.18-8.SEL5 (buildsys@pceddyn.internal.startcom.org) (gcc version 4.1.1 20070105 (StartCom Linux 4.1.1-52)) #1 SMP Fri Mar 23 16:08:07 EDT 2007
```

Замечание: хотя выше и написано про вывод баннера, фактически на консоль еще ничего не выводится, поскольку консоль еще не зарегистрирована в системе. Вывод идет в кольцевой буфер. Как только консоль подключится, сообщения из буфера будут выданы на консоль. Это замечание нужно иметь в виду и всюду ниже, где говорится о выводе каких-либо сообщений.

- Вызывается функция setup_arch(&command_line) (см. в "linux/arch/i386/kernel/setup.c"), которая выполняет архитектурно-специфичные действия по инициализации, а именно:

- Системные параметры из 16-битного реального режима копируются и преобразуются в 32-битный запускающий код. Для конфигураций с включенным RAMdisk (CONFIG_BLK_DEV_RAM) инициализируются rd_image_start, rd_prompt и rd_doload из параметров реального режима.
- Карта памяти от BIOS используется для установки областей памяти.

Вероятно именно в этот момент выводится сообщение вида

```
BIOS-provided physical RAM map:
 BIOS-e820: 0000000000000000 - 000000000009fc00 (usable)
 BIOS-e820: 000000000009fc00 - 00000000000a0000 (reserved)
 BIOS-e820: 00000000000e0000 - 0000000000100000 (reserved)
 BIOS-e820: 0000000000100000 - 00000000177f0000 (usable)
 BIOS-e820: 00000000177f0000 - 0000000017800000 (reserved)
 BIOS-e820: 00000000fec00000 - 0000000100000000 (reserved)
```

- Выставление границ памяти. Устанавливаются значения для начала кода ядра, конца кода ядра, конца данных ядра и "_end" (конец кода ядра = адресу "brk").
- Устанавливаются значения для начала и конца code_resource и начала и конца data_resource.
- Разбираются и запоминаются параметры "mem=" командной строки ядра.
- Установка страничных кадров. Используя карту памяти от BIOS, устанавливаются страничные кадры. Регистрируются доступные страницы нижней RAM в распределителе bootmem. Резервируется физическая страница 0: "это особенная страница BIOS во многих системах, включающая чистые перезагрузки, SMP-операции и laptop-функции."

```
0MB HIGHMEM available.
375MB LOWMEM available.
Using x86 segment limits to approximate NX protection
On node 0 totalpages: 96240
  DMA zone: 4096 pages, LIFO batch:1
  Normal zone: 92144 pages, LIFO batch:31
  HighMem zone: 0 pages, LIFO batch:1
```

Адресуемая обычным образом область памяти (располагающаяся в пределах первых 896 МБ) называется нижней памятью (low memory). Подпрограмма распределения памяти ядра, kmalloc(), отвечает за выделение памяти из этой области. Память за пределами 896 MB

(называемая верхней памятью - high memory) доступна только путем применения специальных приемов (special mappings).

В процессе загрузки ядро определяет и выводит общее число страниц, присутствующих в каждой из этих зон. После загрузки системы эта информация доступна через /proc/meminfo.

- Обработка конфигураций SMP и IO APIC (APIC: Advanced Programmable Interrupt Controller - улучшенный программируемый контроллер прерываний).
  Для CONFIG_SMP резервируем страницу, следующую за страницей 0 для стэка и запуска, затем вызываем smp_alloc_memory() для выделения нижней памяти под запускающий (trampoline) код реального режима для AP процессора(ов).
  Для конфигураций с CONFIG_X86_IO_APIC вызываем find_smp_config() чтобы найти и зарезервировать любую память с SMP-конфигурационной информацией времени загрузки вроде таблицы данных MP (Multi Processor) от BIOS.
- paging_init() устанавливает страничные таблицы (первые 8 МБ уже зарезервированы в head.S). Эта процедура также освобождает страницу по виртуальному адресу ядра 0, что позволяет вылавливать в ядре надоедливые ошибки с нулевыми ссылками (pesky NULL-reference errors).
- Сохранение SMP-конфигурации времени загрузки. Для конфигураций с CONFIG_X86_IO_APIC вызываем get_smp_config() для чтения и записи данных конфигурации MP-таблицы маршрутизации прерываний IO APIC.
  Для конфигураций с CONFIG_X86_LOCAL_APIC вызываем init_apic_mappings().

```
Local APIC disabled by BIOS -- you can enable it with "lapic"
mapped APIC to ffffd000 (0138a000)
```

- Резервирование памяти для INITRD. Для конфигураций с CONFIG_BLK_DEV_INITRD при наличии достаточного кол-ва памяти под начальный RamDisk вызываем reserve_bootmem() чтобы зарезервировать RAM для него.
- Поиск и работа с ROM. Вызываем probe_roms() и если находим корректные ресурсы ROM, то резервируем их. Это делается для образа ROM стандартной видеоBIOS, любых других найденных ROM и для расширения ROM системной платы.
- Резервирование системных ресурсов. Вызываем request_resource() для резервирования видеоRAM. Вызываем request_resource() для резервирования всех стандартных ресурсов ввода-вывода системной платы PC.

```
 ACPI: RSDP (v000 IntelR                    ) @ 0x000f7490
ACPI: RSDT (v001 IntelR AWRDACPI 0x42302e31 AWRD 0x00000000) @ 0x1bff3000
ACPI: FADT (v001 IntelR AWRDACPI 0x42302e31 AWRD 0x00000000) @ 0x1bff3040
ACPI: DSDT (v001 INTELR AWRDACPI 0x00001000 MSFT 0x0100000c) @ 0x00000000
ACPI: PM-Timer IO Port: 0x4008
Allocating PCI resources starting at 20000000 (gap: 1c000000:e3b00000)
Detected 1205.234 MHz processor.
Built 1 zonelists.  Total pages: 114672
```

- Выводится содержимое командной строки ядра.

```
Kernel command line: ro root=/dev/system/LogVol00 rhgb quiet
```

- Между прочим командную строку запуска ядра вы всегда можете в работающей системе просмотреть с помощью команды **cat /proc/cmdline**.
- Разбор параметров командной строки с помощью функции parse_options(command_line). Эта функция анализирует командную строку и соответственно инициализирует аргументы и окружение init'a (который поток). Каждый аргумент командной строки рассматривается как переменная окружения, если он содержит знак "=". Она также ищет опции, предназначенные для ядра, при помощи вызова checksetup(), который проверяет командную строку на параметры ядра - при их наличии они объявляются при помощи "__setup", например:

  __setup("debug", debug_kernel);
  Если при наличии такого объявления обнаружится строка "debug", то будет вызвана функция debug_kernel(). Список параметров ядра можно найти в "linux/Documentation/kernel-parameters.txt".
  Эти опции не передаются init'у - они предназначены для использования исключительно внутри ядра. Список аргументов по умолчанию для init'a есть {"init", NULL}, максимум с 8-ю аргументами командной строки. Установки окружения по умолчанию для потока init'a есть {"HOME=/", "TERM=linux", NULL}, с максимум 8-ю установками переменных окружения в командной строке. Если LILO грузит нас с командной строкой по умолчанию, то он помещает "auto" перед всей строкой, благодаря чему шелл думает, что он должен выполнить скрипт с таким именем. Так что мы игнорируем все аргументы, введенные _перед_ init=... [MJ]

- Выделение ресурсов для APIC. Улучшенный программируемый контроллер прерываний (APIC, the Advanced Programmable Interrupt Controller) - это современная реализация программируемого контроллера прерываний 8259. APIC предоставляет большее количество прерываний. Локальный APIC - это часть архитектуры APIC, относящаяся к каждому конкретному ЦПУ системы. I/O APIC - это другая часть архитектуры, обрабатывающая прерывания от устройств ввода-вывода и доставляющая их локальным APIC. Локальные APIC играют существенную роль в мультипроцессорных системах. Разделение прерываний с использованием традиционного PIC - это сложная задача. APIC сокращает необходимость в разделении (совместном использовании) прерываний. Кроме того, APIC обрабатывает прерывания быстрее, чем PIC. Некоторые BIOS-ы отключают APIC. Вы можете задействовать APIC, добавив в командную строку ядра опцию lapic.

```
Local APIC disabled by BIOS -- you can enable it with "lapic"
mapped APIC to ffffd000 (0138a000)
```

- Вызывается подпрограмма trap_init (из linux/arch/i386/kernel/traps.c), которая:
  - устанавливает обработчики исключений для основных процессорных исключений, (это не обработчики аппаратных прерываний).
  - устанавливает обработчик системного вызова программного прерывания.
  - вызывает cpu_init() чтобы:

  - инициализировать каждый процессор
  - перезагрузить GDT и IDT
  - демаскировать бит NT (Nested Task) регистра eflags
  - установить и загрузить для каждого процессора TSS (Task State Segment - сегмент состояния задачи, i386-специфичная структура данных задачи) и LDT (Local Descriptor Table - локальная дескрипторная таблица, i386-специфичная таблица управления памятью, которая используется для описания памяти для каждого неядерного процесса)
  - очистить 6 отладочных регистров (0, 1, 2, 3, 6 и 7)
  - stts(): установить бит 0x08 (TS: Task Switched) в CR0 для включения задержанной (lazy) записи в регистры при контекстных переключениях.

```
Initializing CPU#0
```

- Вызывается подпрограмма init_IRQ (в linux/arch/i386/kernel/i8259.c), которая:
  - вызывает init_ISA_irqs() чтобы инициализировать контроллер прерываний 8259A и установить дефолтовые обработчики прерываний для ISA.

- устанавливает вход прерывания (interrupt gate - ?) для всех неиспользованных векторов прерывания.

```
CPU 0 irqstacks, hard=c03fd000 soft=c03fc000
```

- для конфигураций с CONFIG_SMP выставляет IRQ 0 заранее, потому как оно используется перед установкой IO APIC.
- для CONFIG_SMP устанавливает обработчик прерывания для CPU-to-CPU IPI, которые используются для "reschedule helper."
- для CONFIG_SMP устанавливает обработчик прерывания для IPI, который используется для сброса (здесь - to invalidate) TLB.
- для CONFIG_SMP устанавливает обработчик прерывания для IPI, который используется для вызовов основных функций.
- для конфигураций с CONFIG_X86_LOCAL_APIC устанавливает обработчик прерывания для IPI от независимо работающего локального таймера APIC.
- для конфигураций с CONFIG_X86_LOCAL_APIC устанавливает обработчики прерываний для ложных или ошибочных прерываний.
- настраивает микросхему системных часов для генерации прерываний каждые HZ герц.
- если система содержит внешний FPU, то устанавливает обработчик IRQ 13 для обработки исключений при операциях с плавающей точкой.

- Отрабатывает подпрограмма инициализации данных для планировщика (sched_init, находится в linux/kernel/sched.c)
- Устанавливаем ID процессора для структур init_task
- Очищаем таблицу pidhash (хешей PID'ов).

```
PID hash table entries: 2048 (order: 11, 32768 bytes)
```

- Вызываем init_timervecs()
- Вызываем init_bh() для инициализации "верхней половины" (bottom half) очередей для timer_bh, tqueue_bh и immediate_bh.

- Вызывается подпрограмма инициализации данных хранения времени (time_init в linux/arch/i386/kernel/time.c)
- Выставляем текущее время системы (xtime) из CMOS.
- Устанавливаем обработчик прерывания irq0 для тиков от таймера.

```
Using pmtmr for high-res timesource
```

- Инициализация подсистемы программных прерываний (softirq_init в linux/kernel/softirq.c)
- Инициализация консоли (console_init в linux/drivers/char/tty_io.c)

```
Console: colour VGA+ 80x25
```

- Если ядро было скомпилировано с поддержкой загружаемых модулей, инициализируется подсистема динамической загрузки модулей (init_modules в linux/kernel/module.c). Тем самым выставляется размер (или кол-во символов) таблицы символов ядра.
- Установка профилирования. Если профилирование включено ("profile=#" в командной строке ядра): вычисляем размер "сегмента" текста (кода) профиля ядра; вычисляем размер буфера профиля в страницах (с округлением); выделяем буфер для профиля: prof_buffer = alloc_bootmem(size);
- kmem_cache_init (в linux/mm/slab.c), начало инициализации менеджера памяти.

```
Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
```

- Вызывается mem_init() (в linux/arch/i386/mm/init.c), которая подсчитывает объем доступной оперативной памяти и сравнивает его с объемом памяти, необходимой для работы ядра:
  - Очищаем empty_zero_page.
  - Вызываем free_all_bootmem() и добавляем всю освобожденную память к totalram_pages.
  - Подсчитываем количество зарезервированных страниц RAM.
  - Выводим размер системной памяти (свободно/всего), размер кода ядра, объем зарезервированной памяти, размер данных ядра, "начальный" размер ядра и объем верхней памяти.

```
Memory: 376448k/384960k available (2125k kernel code, 7956k reserved, 732k data, 172k i
0k highmem)
```

  - Для CONFIG_SMP вызываем zap_low_mappings().

get_free_pages() можно использовать после mem_init().

- Завершение инициализации менеджера памяти (kmem_cache_sizes_init из linux/mm/slab.c)
  Устанавливаем оставшиеся внутренние и основные кэши.
- Вызывается функция calibrate_delay(), определенная в init/calibrate.c. Эта функция подсчитывает, сколько раз в течение интервала времени между двумя отсчетами таймера процессор выполнит внутренний цикл задержки. Результат сохраняется во внутренней переменной ядра "loops_per_jiffy". Эта переменная используется для генерации задержки на заданное время, например, для генерации задежки на 1 микросекунду нужно выполнить пустой цикл (HZ/1000000)*loops_per_jiffy раз, где HZ - частота микропроцессора в герцах. Значение loops_per_jiffy используется для вычисления условной меры быстродействия процессора, известной как BogoMIPS. Значение BogoMIPS можно применять для сравнения производительности разных процессоров. Например, для лэптопа на процессоре Pentium M с частотой 1.6 GHz значение loops_per_jiffy оказалось равным 6385059, а значение BogoMIPS - 3192.52.

```
Calibrating delay using timer specific routine.. 3591.60 BogoMIPS (lpj=7183205)
```

- Инициализация SELinux

```
Security Framework v1.0.0 initialized
SELinux: Initializing.
SELinux: Starting in permissive mode
selinux_register_security: Registering secondary module capability
```

- Разрешаются прерывания (sti).
- Инициализация структур данных для procfs (proc_root_init из linux/fs/proc/root.c)
  Для конфигураций с CONFIG_PROC_FS:
  - вызываем proc_misc_init()
  - mkdir /proc/net
  - для CONFIG_SYSVIPC выполняем mkdir /proc/sysvipc
  - для CONFIG_SYSCTL выполняем mkdir /proc/sys
  - mkdir /proc/fs
  - mkdir /proc/driver
  - вызываем proc_tty_init()
  - mkdir /proc/bus
- mempages = num_physpages;

- Устанавливаем максимальное кол-во потоков по умолчанию в безопасное значение: потоковые структуры могут использовать максимум половину памяти.
  fork_init(mempages) (в linux/kernel/fork.c) создает uid_cache, инициализируется max_threads исходя из объема доступной памяти и конфигурируется RLIMIT_NPROC для init_task как max_threads/2.
- Создаются различные кэши для VFS, VM, кэш буфера и пр..
  - proc_caches_init() (в linux/kernel/fork.c)
    Вызываем kmem_cache_create() для создания отдельных кэшей для signal_act (работа с сигналами), files_cache (files_struct), fs_cache (fs_struct), vm_area_struct и mm_struct.
  - vfs_caches_init(mempages) (в linux/fs/dcache.c)
    Вызываем kmem_cache_create() для создания отдельных кэшей для buffer_head, names_cache, filp и для CONFIG_QUOTA, dquot.
    Вызываем dcache_init() чтобы создать dentry_cache и dentry_hashtable.
  - buffer_init(mempages) (в linux/fs/buffer.c)
    Выделяем память под хэш-таблицу буферного кэша и создаем пустой список. Используем get_free_pages() для хэш-таблицы чтобы уменьшить TLB-промахи; используем SLAB-кэш для заголовков буфера. Устанавливаем цепочки хэша, пустые списки и списки LRU.
  - page_cache_init(mempages) (в linux/mm/filemap.c)
    Выделяем и очищаем память под хэш-таблицу страничного кэша.
  - kiobuf_setup() (в linux/fs/iobuf.c)
    Вызываем kmem_cache_create() чтобы создать кэш буфера ввода-вывода ядра.
  - signals_init() (в linux/kernel/signal.c)
    Вызываем kmem_cache_create() чтобы создать SLAB-кэш "sigqueue" (очереди заданий).
  - bdev_init() (в linux/fs/block_dev.c)
    Инициализируем заголовки списка bdev_hashtable.
    Вызываем kmem_cache_create() чтобы создать SLAB-кэш "bdev_cache".
  - inode_init(mempages) (в linux/fs/inode.c)
    - Выделяем память под inode_hashtable.
    - Инициализируем заголовки списка inode_hashtable.
    - Вызываем kmem_cache_create() SLAB-кэш инодов (inodes).
- Если имеется поддержка System V IPC, то инициализируется подсистема IPC (**ipc_init()** в linux/ipc/util.c). Инициализируются различные ресурсы System V IPC (семафоры, сообщения и разделяемая память). Обратите внимание, что для System V shm, это включает монтирование внутреннего (in-kernel) экземпляра файловой системы shmfs.

- Если включена поддержка квот (quota), создается и инициализируется специальный кэш (**dquot_init_hash()** в linux/fs/dquot.c)

```
VFS: Disk quotas dquot_6.5.1
Dquot-cache hash table entries: 1024 (order 0, 4096 bytes)
```

- Выполняется платформо-зависимая "проверка ошибок" (**check_bugs()** в linux/include/asm-i386/bugs.h) и, если это возможно, активируется обработка ошибок процессора/шины/проч.:
  - identify_cpu()
  - Для не-CONFIG_SMP конфигураций вызываем print_cpu_info()

```
CPU: After generic identify, caps: 0383f9ff 00000000 00000000 00000000 00000000 00000000
CPU: After vendor identify, caps: 0383f9ff 00000000 00000000 00000000 00000000 00000000
CPU: L1 I cache: 16K, L1 D cache: 16K
  CPU: L2 cache: 256K
CPU: After all inits, caps: 0383f1ff 00000000 00000000 00000040 00000000 00000000
Intel machine check architecture supported.
Intel machine check reporting enabled on CPU#0.
```

- - check_config()
  - check_fpu()
  - check_hlt()

```
Checking 'hlt' instruction... OK.
```

- Инструкция процессора HLT, поддерживаемая процессорами x86, переводит ЦПУ в спящий режим с низким энергопотреблением. Выход из этого режима происходит после поступления следующего аппаратного прерывания. Ядро использует инструкцию HLT для перевода ЦПУ в состояние бездействия (in the idle state). Функция cpu_idle() определена в arch/i386/kernel/process.c.
  Можно отключить использование инструкции HLT, если задать опцию командной строки

ядра no-hlt. Если эта опция задана, ядро оставляет ЦПУ в активном состоянии в периоды ожидания вместо того, чтобы переводить его в состояние HLT.

- - check_popad()
  - Корректируем system_utsname.machine{байт 1} в соотв-вии с boot_cpu_data.x86
- В случае SMP запускаем другие процессоры.

  Выполнение smp_init(), в зависимости от конфигурации ядра, возможно тремя способами.
  Для однопроцессорных (UP) систем без IO APIC (CONFIG_X86_IO_APIC не определена), smp_init() пуста - соответственно ничего не происходит.
  Для однопроцессорных систем с IO APIC для маршрутизации прерываний она вызывает IO_APIC_init_uniprocessor().

```
SMP alternatives: switching to UP code
Freeing SMP alternatives: 16k freed
SMP motherboard not detected.
Local APIC not detected. Using dummy APIC emulation.
Brought up 1 CPUs
```

- Для многопроцессорных (SMP) систем основная задача smp_init() заключается в вызове архитектурно-специфичной функции "smp_boot_cpus()", которая выполняет следующее:
  - Для ядер с CONFIG_MTRR вызывает mtrr_init_boot_cpu(), которая должна отработать до того, как загрузятся другие процессоры.
  - Сохраняет и выводит информацию о BSP CPU.
  - Сохраняет ID BSP APIC и BSP ID логического CPU (последний равен 0).
  - Если не найдена таблица маршрутизации прерываний MP BIOS, то возвращается к использованию только одного CPU и завершается.
  - Проверяет существование локального APIC для BSP.
  - Если использована опция загрузки "maxcpus" со значением 1 (без SMP), то игнорирует таблицу маршрутизации прерываний MP BIOS.
  - Переключает систему из PIC-режима в режим прерывания симметричного ввода-вывода.
  - Устанавливает локальный APIC BSP.
  - Использует карту наличия (presence map) CPU для последовательной загрузки AP. Ожидает загрузки предыдущего AP перед запуском загрузки следующего.

- В случае использования IO APIC {что справедливо всегда кроме случаев с опцией загрузки "noapic"} устанавливает IO APIC (или каждый, если их несколько).

- Инициализация INITRD (INITial Ram-Disk)

```
checking if image is initramfs... it is
Freeing initrd memory: 2693k freed
```

- О том, что такое виртуальный диск initrd и для чего предназначен, было рассказано в разделе 3.2. После инициализации RAM-диска ядро освобождает ту область памяти, в которую загрузчик перенес образ этого диска (в нашем случае - 2693 КБайт). Освободившиеся страницы памяти раздаются другим частям ядра, которым потребуется память.
- Инициализация сетевых сокетов.

```
NET: Registered protocol family 16
NET: Registered protocol family 2
NET: Registered protocol family 1
```

- *(эти сообщения следуют в потоке сообщений, выдаваемых в процессе загрузки, не группой, а с некоторым разбросом)*
  Уровень сетевых сокетов (socket layer) в Линукс - это унифицированный интерфейс, с помощью которого пользовательские приложения получают доступ к различным сетевым протоколам. Каждый протокол регистрируется в socket layer, используя назначенный ему номер семейства (определен в include/linux/socket.h). Так, например, номер 2 назначен AF_INET, семейству протоколов Internet IP.
  Номер 16 в приведенном листинге соответствует семейству AF_NETLINK (family 16). Сокеты сетевых соединений - это специальный механизм для организации взаимодействия ядра с процессами пользовательского уровня, используя механизм API сокетов. С помощью сетевых сокетов обеспечивается доступ к таблицам маршрутизации и таблицам протокола разрешения адресов (Address Resolution Protocol, ARP), полный список возможностей, обеспечиваемых сетевыми сокетами, можно найти в include/linux/netlink.h. Сетевые соединения более подходящи для таких задач, чем системные вызовы, поскольку они асинхронны, более просты в применении и могут быть связаны динамически.
  Еще одним широко используемым семейством протоколов является AF_UNIX (семейство с номером 1). Доменные сокеты Unix (Unix domain sockets) используются такими программами как X Window System для организации взаимодействия между процессами

одной и той же системы. Они используются даже в том случае, если машина не подключена к сети.

- На следующем этапе загрузки опробуются и инициализируются шины ввода-вывода и контроллеры периферийных устройств. Ядро опробует аппаратное обеспечение путем опроса шины PCI,

```
PCI: Probing PCI hardware (bus 00)
PCI quirk: region 4000-407f claimed by ICH4 ACPI/GPIO/TCO
PCI quirk: region 4080-40bf claimed by ICH4 GPIO
Boot video device is 0000:01:00.0
PCI: Transparent bridge - 0000:00:1e.0
```

- после чего опрашивает другие подсистемы:
- видео чип (part of the 855 North-bridge chipset in this case),

```
Linux agpgart interface v0.101 (c) Dave Jones
agpgart: Detected an Intel i815 Chipset.
agpgart: AGP aperture is 64M @ 0xd0000000
```

- - последовательный порт ,

```
Serial: 8250/16550 driver $Revision: 1.90 $ 4 ports, IRQ sharing enabled
serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
serial8250: ttyS1 at I/O 0x2f8 (irq = 3) is a 16550A
00:08: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
00:09: ttyS1 at I/O 0x2f8 (irq = 3) is a 16550A
```

- - виртуальный диск,

```
RAMDISK driver initialized: 16 RAM disks of 16384K size 4096 blocksize
```

- - IDE-контроллер (в данном случае - часть чипсета южного моста ICH2),

```
Uniform Multi-Platform E-IDE driver Revision: 7.00alpha2
ide: Assuming 33MHz system bus speed for PIO modes; override with idebus=xx
ICH2: IDE controller at PCI slot 0000:00:1f.1
```

```
ICH2: chipset revision 5
ICH2: not 100% native mode: will probe irqs later
    ide0: BM-DMA at 0xf000-0xf007, BIOS settings: hda:DMA, hdb:DMA
      ide1: BM-DMA at 0xf008-0xf00f, BIOS settings: hdc:DMA, hdd:DMA
Probing IDE interface ide0...
hda: ST340810A, ATA DISK drive
hdb: ST3160023A, ATA DISK drive
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
Probing IDE interface ide1...
hdc: FX810T4, ATAPI CD/DVD-ROM drive
hdd: _NEC DVD_RW ND-3500AG, ATAPI CD/DVD-ROM drive
ide1 at 0x170-0x177,0x376 on irq 15
hda: max request size: 128KiB
hda: 78165360 sectors (40020 MB) w/2048KiB Cache, CHS=65535/16/63, UDMA(100)
hda: cache flushes not supported
 hda: hda1 hda2 hda3
hdb: max request size: 512KiB
hdb: 312581808 sectors (160041 MB) w/8192KiB Cache, CHS=19457/255/63, UDMA(100)
hdb: cache flushes supported
 hdb: hdb1 hdb2 hdb3
```

- - дисковод гибких дисков,

```
ide-floppy driver 0.99.newide
```

- - USB-контроллер,

```
usbcore: registered new driver hiddev
usbcore: registered new driver usbhid
drivers/usb/input/hid-core.c: v2.6:USB HID core driver
```

- - контроллер PS/2 с клавиатурой и мышью,

```
PNP: PS/2 Controller [PNP0303:PS2K,PNP0f13:PS2M] at 0x60,0x64 irq 1,12
serio: i8042 AUX port at 0x60,0x64 irq 12
```

```
serio: i8042 KBD port at 0x60,0x64 irq 1
mice: PS/2 mouse device common for all mice
```

- - SCSI,

```
SCSI subsystem initialized
```

- - обратную петлю (the loopback device),
  - контроллер Ethernet
  - контроллер PCMCIA

  и другие устройства. Нужно иметь в виду, что в случае, когда некоторые драйверы сконфигурированы как загружаемые модули, соответствующие сообщения могут появляться немного позже и в другом порядке.
- Монтирование файловых систем

```
EXT3 FS on hda3, internal journal
kjournald starting. Commit interval 5 seconds
EXT3 FS on hda4, internal journal
EXT3-fs: mounted filesystem with ordered data mode.
Adding 1050800k swap on /dev/hda1. Priority:-1 extents:1 across:1050800k
```

- Файловая система Ext3 постепенно становится стандартом де-факто для корневой файловой системы Linux. Она добавляет свойство журналируемости к файловой системе Ext2; журналируемость заключается в том, что каждая файловая операция вначале фиксируется в журнале и только после этого производятся действительные изменения на диске. Это позволяет быстро восстановить файловую систему в случае каких-то сбоев. Ext3 использует поток ядра kjournald для поддержки журналирования. После того, как Ext3 будет готова к работе, ядро может смонтировать корневую файловую систему.
- Устанавливается флаг, указывающий на то, что планировщик должен быть вызван "при первой возможности" и создается поток ядра init(), который выполняет команду, указанную в параметре "init=" , если она имеется среди параметров командной строки, или пытается запустить **/sbin/init**, **/etc/init**, **/bin/init**, **/bin/sh** в указанном порядке; если не удается ни один из запусков, то ядро "впадает в панику" с "предложением" задать параметр "init=".
  Как и бездействующие потоки (idlers), init является неблокируемым потоком ядра, который делает системные вызовы (и поэтому не может быть блокируемым).
- **unlock_kernel()** Снимает глобальную блокировку ядра (BKL).

- **current->need_resched = 1;**
- Переход **cpu_idle()** в фоновый поток с pid=0 Эта функция остается как процесс номер 0. Ее цель заключается в использовании пустых циклов CPU. Если ядро собрано с поддержкой APM или ACPI, то cpu_idle() вызывает поддерживаемые энергосберегающие свойства этих спецификаций. В противном случае она просто выполняет инструкцию "hlt".

Процесс init и файл /etc/inittab

Как было показано в предыдущем разделе, если в параметре начальной загрузки "init=" не задан запуск какой-то другой программы, после монтирования корневой файловой системы в режиме "только для чтения" ядро запускает процесс init, который, как известно, является родоначальником всех других процессов в Linux. Сам по себе init в принципе ничем не отличается от других программ в системе Linux, просто это первая (и единственная) программа, которая запускается непосредственно ядром, все остальные процессы являются потомками процесса init. Файл программы init вы можете найти в каталоге /sbin среди других исполняемых файлов.

Init отвечает за продолжение процедуры загрузки, и перевод системы от начального состояния, возникающего после загрузки ядра, в стандартное состояние обработки запросов многих пользователей. Основная задача, которая стоит перед init, заключается в том, чтобы запускать в определенной последовательности другие программы в процессе загрузки системы и останавливать процессы в случае переключения уровня выполнения (в частности, при остановке системы). Init выполняет еще массу различных операций, необходимых для дальнейшей работы системы: проверку и монтирование файловых систем, запуск различных служб (демонов), запуск процедур логирования, оболочек пользователей на различных терминалах и т.д. Но, прежде чем рассматривать работу процесса init более детально, нужно сказать несколько слов о стилях загрузки и так называемых "уровнях выполнения".

5.1. Уровни выполнения

Уровни выполнения (run levels) – это просто несколько стандартных вариантов загрузки системы, каждый из которых определяет перечень действий, выполняемых процессом init, и состояние системы после загрузки, т. е. конфигурацию запущенных процессов. Уровень выполнения идентифицируется одним символом. В большинстве дистрибутивов ОС Linux

используется 6 основных уровней выполнения. В следующей таблице показано, как эти уровни используются в дистрибутивах Slackware и Red Hat:

Уровень	Slackware	Red Hat
0	остановка системы	остановка системы
1	однопользовательский режим	однопользовательский режим
2	использование не регламентировано	многопользовательский режим без NFS (то же, что и 3, если компьютер не работает с сетью)
3	многопользовательский режим	полный многопользовательский режим
4	запуск системы в графическом режиме (X11 с KDM/GDM/XDM)	использование не регламентировано
5	использование не регламентировано	запуск системы в графическом режиме
6	перезагрузка системы	перезагрузка системы;

В некоторых дистрибутивах (например, в Debian), кроме того, используется дополнительный уровень S (или s) — примерно то же, что и однопользовательский режим, но S и s используются в основном в скриптах.

Как видите, уровни 0, 1 и 6 зарезервированы для особых случаев. Относительно того, как использовать уровни со 2 по 5, единого мнения не существует. Некоторые системные администраторы используют разные уровни для того, чтобы задать разные варианты работы, например, на одном уровне запускается графический режим, на другом работают в сети и т. д. Вы можете сами решить, как использовать разные уровни для создания разных вариантов загрузки. Но для начала проще всего воспользоваться тем способом определения разных уровней, который был задан при установке.

Уровень выполнения может быть задан как одна из опций, передаваемых ядру загрузчиком. Обычно единственной причиной, по которой уровень загрузки может быть задан как аргумент при загрузке, является необходимость запуска системы в однопользовательском режиме (уровень выполнения 1) для выполнения каких-то административных задач или в случае повреждения диска. Но если уровень выполнения не задан как опция загрузки, то **init** будет загружать систему на уровень, заданный в файле /etc/inittab.

Если вам нужно узнать, на каком уровне выполнения работает ваша система, то можно воспользоваться командой **runlevel**. Будучи запущенной без параметров, она сообщает предыдущий и текущий уровень выполнения в виде двух цифр (лучше сказать символов, потому что существуют уровни S и s), разделенных пробелом. Если уровень выполнения не изменялся, то первый символ (идентифицирующий предыдущий уровень) примет значение N.

Изменять уровень выполнения в в работающей системе можно с помощью команды **telinit**. Естественно, что для ее запуска нужно иметь права суперпользователя. Но об этой команде мы поговорим чуть позже, а пока давайте посмотрим что из себя представляет файл /etc/inittab.

5.2. Файл inittab

Конфигурационный файл /etc/inittab состоит из отдельных строк. Если строка начинается со знака # или пуста, то она игнорируется. Все остальные строки состоят из 4 полей, разделенных двоеточиями:

id:runlevels:action:process

где:

- id — идентификатор строки. Это произвольная комбинация, содержащая от 1 до 4 символов. В файле inittab не может быть двух строк с одинаковыми идентификаторами;
- runlevels — уровни выполнения, на которых эта строка будет задействована. Уровни задаются цифрами или буквами без разделителей, например, 345;
- process — процесс, который должен запускаться на указанных уровнях. Другими словами в этом поле указывается имя программы, вызываемой при переходе на указанные уровни выполнения;
- action — действие.

В поле action стоит ключевое слово, которое определяет дополнительные условия выполнения команды, заданной полем process. Допустимые значения поля action:

- respawn — перезапустить процесс в случае завершения его работы;

Кодовое слово respawn заставляет init запустить команду, которая указана в этой строке и, если вызванная программа завершает работу, запустить ее снова. Возьмем, например, следующую строку в файле inittab:

1:2345:respawn:/sbin/mingetty tty1

Программа getty обеспечивает открытие виртуальной консоли (в данном случае - первой) и запуск в ней программы login, выводящей приглашение ко входу пользователя в систему. Ключевое слово respawn в приведенной строке означает, что после того, как пользователь выйдет из системы, приглашение login будет выведено на первую виртуальную консоль снова.

- once — выполнить процесс только один раз при переходе на указанный уровень;
- wait — процесс будет запущен один раз при переходе на указанный уровень и init будет ожидать завершения работы этого процесса, прежде, чем продолжать работу; Например, появление этого слова в строке

  l5:5:wait:/etc/rc.d/rc 5

  означает, что init при переходе на 5-ый уровень запустит на выполнение команду /etc/rc.d/rc 5 и будет ожидать окончания работы этой программы прежде чем приступить к чему-либо другому.

- sysinit — это ключевое слово обозначает самые первые действия, выполняемые процессом init еще до перехода на какой-либо уровень выполнения (поле id игнорируется). Процессы, помеченные этим словом, запускаются до процессов, помеченных словами boot и bootwait;
- boot — процесс будет запущен на этапе загрузки системы независимо от уровня выполнения;
- bootwait — процесс будет запущен на этапе загрузки системы независимо от уровня выполнения, и init будет дожидаться его завершения;
- initdefault — строка, в которой это слово стоит в поле action, определяет уровень выполнения, на который система переходит по умолчанию. Поле process в этой строке игнорируется. Если уровень выполнения, используемый по умолчанию, не задан, то процесс init будет ждать, пока пользователь, запускающий систему, не введет его с консоли;
- off — игнорировать данный элемент;

- powerwait — позволяет процессу init остановить систему, когда пропало питание. Использование этого слова предполагает, что имеется источник бесперебойного питания (UPS) и программное обеспечение, которое отслеживает состояние UPS и информирует init о том, что питание отключилось;
- ctrlaltdel — разрешает init перезагрузить систему, когда пользователь нажимает комбинацию клавиш <Ctrl>+<Alt>+<Del> на клавиатуре. Обратите внимание на то, что системный администратор может определить действия по комбинации клавиш <Ctrl>+<Alt>+<Del>, например игнорировать нажатие этой комбинации (что вполне разумно в системе, где много пользователей).

Этот список не является исчерпывающим. Более подробно о файле inittab можно узнать из man-страниц init (8), inittab (5) и getty (8).

5.3. Стили начальной загрузки

А теперь давайте вспомним, что существует два разных стиля начальной загрузки операционной системы типа UNIX, происхождение которых уходит корями в историю развития UNIX-систем: так называемый стиль BSD (используемый также в таких системах как FreeBSD, NetBSD и OpenBSD), и стиль System V (или стиль ATT). Различие между ними проявляется в организации и размещении стартовых сценариев (скриптов), обеспечивающих управление процессами загрузки системы. В классических BSD-системах эти файлы хранятся в каталоге /etc и их имена начинаются с префикса "rc". В системах семейства System V файлы сценариев располагаются в каталоге /etc/init.d, а ссылки на них созданы в каталогах /etc/rc0.d, /etc/rc1.d и т.д. Отличие стиля BSD от стиля System V состоит в организации скриптов **rc**: если в системах стиля System V вызывается один и тот скрипт **rc**, только ему передается параметр, задающий уровень выполнения, то для систем BSD-стиля характерно наличие отдельного скрипта для каждого из уровней выполнения. Вариант организации в стиле System V является более четким и позволяет аккуратнее выполнять останов системы.

Большая часть дистрибутивов Linux использует на этапе загрузки стиль System V. К этому классу относятся Debian, все клоны Red Hat, включая Mandrake и российские дистрибутивы ASPlinux и ALT Linux. В стиле BSD организована загрузка в дистрибутивах Gentoo, Slackware. Однако тот или иной стиль сценариев начальной загрузки выдерживается не очень четко. Поскольку стиль System V взят за основу при создании стандарта LSB (Linux Standart Base), дистрибутивы, ранее использовавшие стиль BSD, в последнее время

заботятся о совместимости с System V. Slackware обеспечивает такую совместимость начиная с версии 7.0.

Структура каталогов, в которых хранятся инициализационные скрипты, в основных дистрибутивах существенно различается:

Листинг 8.

Fedora (Red Hat) и Mandriva 2007	**/etc/init.d/**-----символьная ссылка на **/etc/rc.d/init.d/** **/etc/rc.d/**------скрипты **rc, rc.sysinit** и **rc.local** | **/init.d/**-----множество файлов-скриптов **/rc0.d/**-----символьные ссылки **@K*.*** и **@S*.*** **/rc1.d/**-----символьные ссылки **@K*.*** и **@S*.*** ..... **/rc6.d/**-----символьные ссылки **@K*.*** и **@S*.*** **/rcS.d/**-----символьные ссылки **@K*.*** и **@S*.***
Knoppix (клон **Debian**)	**/etc/init.d/**-----скрипты **rc, rcS, reboot** и множество других, **/etc/rc0.d/**-----символьные ссылки **@K*.*** и **@S*.*** **/rc1.d/**-----символьные ссылки **@K*.*** и **@S*.*** ..... **/rc6.d/**-----символьные ссылки **@K*.*** и **@S*.*** **/rcS.d/**-----символьные ссылки **@K*.*** и **@S*.***
Slackware	**/etc/rc.d/**--скрипты **rc.0, rc.4, rc.6, rc.K, rc.M, rc.S, rc.local,** **rc.syslog, rc.nfsd, rc.sysvinit, rc.netdevice, rc.keymap** и другие скрипты, имена которых имеют вид **rc.***.
Gentoo	**/etc/init.d/**--скрипты с самыми разными именами.
SuSE	**/etc/rc.d/**------символьная ссылка на **/etc/init.d/** **/etc/init.d/**----множество файлов с разными именами, в частности **boot.*** **/boot.d/**----символьные ссылки **@K*.*** и **@S*.*** **/rc0.d/**-----символьные ссылки **@K*.*** и **@S*.*** **/rc1.d/**-----символьные ссылки **@K*.*** и **@S*.*** .....

	/rc6.d/-----символьные ссылки **@K*.*** и **@S*.*** **/rcS.d/**-----символьные ссылки **@K*.*** и **@S*.***

Как видите, даже в Red Hat и Debian, которые следуют стилю System V, структура каталогов несколько отличается. А SuSE, хотя и происходит от Slackware, движется в сторону стиля System V, по крайней мере в организации структуры каталогов. А вот Red Hat, наоборот, завел скрипт rc.local, напоминающий одноименный сценарий во FreeBSD. В процессе загрузки он выполняется последним. Правда, в последних версиях Red Hat и Fedora Core этот скрипт "пустой", то есть практически не используется.

Но структура каталогов и названия инициализационных скриптов, может быть, и не самое главное. Более существенно то, что в этих скриптах содержится и как они используются. В листинге 9 приведены выдержки из файла /etc/inittab для нескольких дистрибутивов. Приводится не полный текст файла /etc/inittab, а только по 4 группы самых важных строк, определяющих процесс загрузки. Во всех дистрибутивах кроме приведенных инструкций определяются еще действия по комбинации клавиш <Ctrl>+<Alt>+<Del> и выполняется запуск виртуальных терминалов (только в Fedora Core и SuSE для этого вызываются процессы mingetty, в Gentoo и Slackware - agetty, в Knoppix - /bin/bash -login).

Листинг 9.

Fedora Core **(Red Hat)**	id:3:initdefault: # System initialization. si::sysinit:/etc/rc.d/rc.sysinit l0:0:wait:/etc/rc.d/rc 0 l1:1:wait:/etc/rc.d/rc 1 l2:2:wait:/etc/rc.d/rc 2 l3:3:wait:/etc/rc.d/rc 3 l4:4:wait:/etc/rc.d/rc 4 l5:5:wait:/etc/rc.d/rc 5 l6:6:wait:/etc/rc.d/rc 6 x:5:respawn:/etc/X11/prefdm -nodaemon

Knoppix (клон **Debian**)	id:5:initdefault: # Boot-time system configuration/initialization script. si::sysinit:/etc/init.d/rcS # What to do in single-user mode. ~~:S:respawn:/bin/bash -login >/dev/tty1 2>&1 </dev/tty1 l0:0:wait:/etc/init.d/knoppix-halt l1:1:wait:/etc/init.d/rc 1 l2:2:wait:/etc/init.d/rc 2 l3:3:wait:/etc/init.d/rc 3 l4:4:wait:/etc/init.d/rc 4 l5:5:wait:/etc/init.d/rc 5 l6:6:wait:/etc/init.d/knoppix-reboot # Run X Window session from CDROM in runlevel 5 w5:5:wait:/bin/sleep 2 x5:5:wait:/etc/init.d/xsession start
Slackware	id:3:initdefault: # System initialization (runs when system boots). si:S:sysinit:/etc/rc.d/rc.S su:1S:wait:/etc/rc.d/rc.K rc:2345:wait:/etc/rc.d/rc.M l0:0:wait:/etc/rc.d/rc.0 l6:6:wait:/etc/rc.d/rc.6 # Runlevel 4 used to be for an X window only system x1:4:wait:/etc/rc.d/rc.4
Gentoo	id:3:initdefault: # System initialization (runs when system boots).

	si:S:sysinit:/sbin/rc boot ~~:S:wait:/sbin/sulogin l0:0:wait:/sbin/rc shutdown l1:1:wait:/sbin/rc single l2:2:wait:/sbin/rc nonetwork l3:3:wait:/sbin/rc default l4:4:wait:/sbin/rc default l5:5:wait:/sbin/rc default l6:6:wait:/sbin/rc reboot z6:6:respawn:/sbin/sulogin # Used by /etc/init.d/xdm to control DM startup. x:a:once:/etc/X11/startDM.sh
SuSE	# The default runlevel is defined here id:5:initdefault: # First script to be executed, if not booting in emergency (-b) mode si::bootwait:/etc/init.d/boot # /etc/init.d/rc takes care of runlevel handling l0:0:wait:/etc/init.d/rc 0 l1:1:wait:/etc/init.d/rc 1 l2:2:wait:/etc/init.d/rc 2 l3:3:wait:/etc/init.d/rc 3 #l4:4:wait:/etc/init.d/rc 4 l5:5:wait:/etc/init.d/rc 5 l6:6:wait:/etc/init.d/rc 6 # what to do in single-user mode ls:S:wait:/etc/init.d/rc S ~~:S:respawn:/sbin/sulogin

Как видите, во всех случаях порядок действий примерно одинаков:

- Вначале задается уровень выполнения (runlevel).
- Затем выполняются действия начальной инициализации системы, не зависящие от заданного уровня. В разных дистрибутивах для этого вызываются разные скрипты:
  - в Fedora Core - /etc/rc.d/rc.sysinit
  - в Knoppix - /etc/init.d/rcS
  - в Slackware - /etc/rc.d/rc.S (то есть скрипт перехода в Single User mode):
  - в Gentoo - /sbin/rc boot
  - в SuSe - /etc/init.d/boot
- Затем выполняется скрипт перехода на заданный уровень выполнения. При этом, если в Fedora Core и Gentoo для каждого уровня выполнения задана своя строка, то в Slackware для уровней со 2-го по 5-ый выполняются одни и те же действия.
- Наконец, во всех дистрибутивах, кроме SuSE, отдельная строка задает процедуры запуска графической оболочки.

При использовании стиля System V используется один и тот же скрипт перехода на заданный уровень, а то, что он делает, определяется содержимым каталога /etc/rc.d/rcN.d. Этот каталог содержит перечень ссылок на скрипты запуска тех системных сервисов, которые должны работать на уровне N. Сами скрипты размещаются в каталоге /etc/init.d или /etc/rc.d/init.d.

В отличие от стиля System V в BSD-стиле каждому уровню загрузки соответствует свой сценарий. И сначала всегда происходит переход на уровень S (однопользовательский), а затем уже переход на заданный уровень. Стиль BSD в наиболее чистом виде (из рассматриваемых примеров) представлен в дистрибутиве Slackware.

Поскольку стиль System V взят за основу при создании стандарта LSB (Linux Standart Base), дистрибутивы, ранее использовавшие стиль BSD, в последнее время заботятся о совместимости с System V. Slackware обеспечивает такую совместимость начиная с версии 7.0. Достигается это путем использования сценария rc.sysinit, который производит поиск всех сценариев стиля System V в каталоге /etc/rc.d и выполнит их, если уровень загрузки соответствующий. Это полезно, если вы пользуетесь коммерческим программным обеспечением, которое ориентируется на стиль System V. В то же время, вы можете пользоваться и BSD сценариями.

Можно по-разному оценивать, какой из вариантов организации инициализационных скритпов лучше или хуже.

- "В сценариях Red Hat непросто разобраться."
- "Сценарии запуска системы - это та область, в которой SUSE превосходит другие дистрибутивы Linux. Эти сценарии четко организованы, надежны и хорошо документированы."
- "Сценарии Debian ненадежны, недокументированы и невероятно противоречивы. Печально, но отсутствие стандартных правил организации сценариев запуска привело в данном случае к хаосу."

Как бы то ни было, разобраться в сценариях запуска возможно, что мы и попытаемся сделать в следующем разделе. А пока рассмотрим в целом, какие задачи выполняет процесс **init**.

5.4. Действия, выполняемые процессом init

Для примера возьмем файл inittab из дистрибутива ASP Linux 11, то есть будет рассматриваться процедура начальной загрузки в стиле System V.

Листинг 10. Файл /etc/inittab системы ASP Linux 11.

```
#
# inittab       This file describes how the INIT process should set up
#               the system in a certain run-level.
#
# Author:       Miquel van Smoorenburg,
#               Modified for RHS Linux by Marc Ewing and Donnie Barnes
#

# Default runlevel. The runlevels used by RHS are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:

# System initialization.
```

```
si::sysinit:/etc/rc.d/rc.sysinit

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# When our UPS tells us power has failed, assume we have a few minutes
# of power left.  Schedule a shutdown for 2 minutes from now.
# This does, of course, assume you have powerd installed and your
# UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"

# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"



# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
#4:2345:respawn:/sbin/mingetty tty4
#5:2345:respawn:/sbin/mingetty tty5
#6:2345:respawn:/sbin/mingetty tty6

# Run xdm in runlevel 5
x:5:once:/etc/X11/prefdm -nodaemon
```

Обработка файла /etc/inittab процессом init начинается в однопользовательском режиме (уровень 1), в котором единственным пользователем является root, работающий с консоли. Первым делом init находит строку, которая определяет, какой уровень выполнения запускается по умолчанию:

id:3:initdefault:

Это и будет тот уровень, в котором запустится и будет работать система после загрузки, поэтому естественно, что нельзя указывать в строке initdefault уровни 0 и 6.

Далее init выполняет команды, указанные в строке с ключевым словом sysinit. В нашем примере здесь выполняется скрипт rc.sysinit из каталога /etc/rc.d:

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

После этого процесс init просматривает файл /etc/inittab и выполняет скрипты, соответствующие однопользовательскому уровню (1 во втором поле строки):

l1:1:wait:/etc/rc.d/rc 1

всем уровням (строки с пустым вторым полем):

ca::ctrlaltdel:/sbin/shutdown -t3 -r now
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"

и уровню, заданному по умолчанию:

l3:3:wait:/etc/rc.d/rc 3
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3

Как видим, вначале вызывается скрипт rc из каталога /etc/rc.d. Какой бы уровень выполнения не был задан, вызывается один и тот скрипт, только в зависимости от уровня выполнения ему передается соответствующее значение параметра вызова, так что, например, для 3-го уровня вызов скрипта осуществляется с параметром 3.

Следующая важная функция, которую выполняет этот процесс (на уровнях со 2 по 5) — запуск шести виртуальных консолей (процессов getty), чтобы предоставить пользователям возможность регистрироваться в системе с терминалов.

Функции, выполняемые скриптами rc.sysinit и rc, а также процесс запуска виртуальных консолей мы подробнее рассмотрим ниже, а сейчас вернемся к краткому обзору действий процесса init.

После завершения загрузки init продолжает работать в фоновом режиме, отслеживая изменения в состоянии системы. Например, если будет подана команда telinit, позволяющая изменить уровень выполнения, процесс init обеспечит выполнение команд, заданных для нового уровня файлом /etc/inittab. Этот файл прочитывается заново и в случае поступления сигнала HUP; эта особенность избавляет от необходимости перезагружать систему для того, чтобы сделать изменения в начальной конфигурации.

Таким образом, процесс начальной загрузки init постоянно находится в оперативной памяти и при получении соответствующих сигналов повторно выполняет цикл чтения из файла /etc/inittab инструкций о том, что нужно делать, причем этот набор инструкций различен для разных уровней выполнения.

Когда суперпользователь останавливает систему (командой shutdown), именно init завершает все другие исполняющиеся процессы, размонтирует все файловые системы и останавливает процессор.

К вопросу о том, что делает процесс init после завершения процесса загрузки системы, нам, видимо, еще придется вернуться, а пока обратимся к разбору инициализационных скриптов.

6: Инициализационный скрипт rc.sysinit

В этом и следующем разделах мы рассмотрим инициализационные скрипты, организованные в стиле Red Hat. Для примера будут использоваться скрипты из системы ASP Linux 11 и Mandriva Free 2007.

Если вы внимательно прочитали предыдущий раздел, то представляете, что в обычной ситуации одним из первых действий, выполняемых процессом init является запуск на выполнение скрипта **rc.sysinit** из каталога **/etc/rc.d**. О том, какие действия выполняет этот скрипт, можно понять, просмотрев и поранализировав его содержимое. В принципе, это не так уж и сложно, чтобы самому разобраться в этом файле, нужно только немного владеть языком команд командной оболочки **bash**. Разбор инициализационных скриптов - неплохая практика в освоении языка интерпретатора команд. Однако не стоит думать, что это совсем

уж просто. Главная сложность не в том, чтобы понять назначение отдельных команд или фрагментов этого файла. Гораздо сложнее понять логику его работы в целом, поскольку чаще всего сначала определяются значения каких-то переменных, выполняются различные подготовительные действия, и только где-то потом, значительно позже, эти заготовки используются.

Вторым источником данных о том, что делают инициализационные скрипты, могут служить те сообщения, которые выдаются на консоль в процессе загрузки системы. Однако, в отличие от сообщений, сохраняемых в кольцевом буфере ядра, мне не удалось найти полную запись этих сообщений после загрузки системы.

6.1. Обзор действий, выполняемых скриптом rc.sysinit

Очень кратко, перечень действий, выполняемых скриптом **rc.sysinit**, выглядит примерно так:

1. Задаются начальные значения общесистемных переменных PATH, HOSTNAME, HOSTTYPE и т.д., а также значения нескольких переменных, которые будут использоваться на дальнейших этапах загрузки.
2. Определяется, используя содержимое файла **/etc/sysconfig/network**, должна ли данная система подключаться к какой-то локальной сети. В этом файле просто задается значение переменной NETWORKING (да или нет).
3. Аналогичным образом (то есть путем считывания установок из нескольких файлов, размещаемых в каталоге ,**/etc/sysconfig**, и задания каких-то значений переменных) задается необходимость использования usb-устройств, уровень безопасности и т.д.
4. Считываются из файла **/etc/init.d/functions** определения функций, которые будут использоваться скриптами из каталога **/etc/init.d** на следующих этапах загрузки (а, возможно, и в работающей системе).
5. Монтируются файловые системы **/proc** (файловая система, используемая в Linux для определения состояния различных процессов) и **/sys**.
6. Утилита **udev** вызывается для создания файлов блоковых устройств в каталоге **/dev**.
7. Задается системный шрифт.
8. Содержимое командной строки загрузки ядра записывается в переменную **cmdline**.
9. Выдается сообщение с ссобщением о загружаемой систем и информацией о возможности продолжения загрузки в интерактивном режиме:
10. Добро пожаловать в Mandriva Linux 2007.1

Нажмите 'I' для перехода в интерактивную загрузку.

11. Перенастраиваются параметры ядра путем запуска команды **sysctl**. Эта операция выполняется в скрипте **rc.sysinit** неоднократно по мере изменения каких-то параметров.
12. Восстанавливается системное время и другие временные установки (установку часового пояса и т.д.), ориентируясь на показания датчика времени в BIOS и значения параметров, заданные при инсталляции системы.
13. Загружаются модули, заданные пользователем в **/etc/sysconfig/modules/*.modules**.
14. Задается сетевое имя компьютера (host name), используемое в процедурах идентификации, таких как NIS (*Network Information Service*), NIS+ (улучшенная версия NIS) и так далее.
15. Задаются настройки, необходимые для подключения устройств Firewire, USB, RAID-массивов, менеджера логических томов и шифрования диска. Например, проверяется существует ли **/proc/bus/usb** и является ли он каталогом. Если этот каталог существует, но не указан в файле **/proc/mounts**, то монтируется файловая система типа usbfs.
16. Проверяется корневая файловая система и, в случае отсутствия проблем, она монтируется в режиме чтения-записи.
17. Задействуется виртуальная память, активизируется и монтируется swap-раздел, указанный в файле **/etc/fstab**.
18. Проверяются другие файловые системы, перечисленные в **/etc/fstab**.
19. Монтируются все файловые системы, перечисленные в файле **/etc/fstab**.
20. Проверяются квоты на использование дискового пространства.
21. Идентифицируется и распознается установленное оборудование, конфигурируются устройства Plug'n'Play, активизируются другие устройства, например, звуковая плата.
22. Проверяет состояние специальных дисковых устройств, таких как RAID (*Redundant Array of Inexpensive Disks*).
23. Загружаются раскладки клавиатуры. Задается клавиша переключения раскладок.
24. Инициализируется генератор (псевдо)случайных чисел.
25. Если обнаружен ленточный накопитель или в командной строке запуска присутствует параметр "hdd=ide-scsi", загружаются необходимые модули.
26. Запускается оптимизация жесткого диска (вызовом команды **hdparm**).

27. Обновляется ссылка на файл /boot/System.map (чтобы этот файл соответствовал загружаемому ядру).
28. Содержимое кольцевого буфера ядра выгружается в файл /var/log/dmesg.
29. Производится удаление всех временных файлов, каталогов и переменных, созданных для целей загрузки.
30. Если в командной строке не задан выбор графической оболочки, то для нее задается установка по умолчанию.

    Последнее сообщение, выдаваемое в процессе работы скрипта **rc.sysinit**, относится, по-видимому, к инициализации свопа:

Включается пространство для свопинга: [OK]
Если я правильно понимаю, дальнейшие сообщения относятся уже к работе скриптов из **/etc/init.d**. Но об этих скриптах будем говорить в следующем разделе.

6.2. Конфигурационные файлы, вызываемые из rc.sysinit

Как было сказано в разделе 6.1, скрипт **rc.sysinit** вызывает (или считывает) ряд конфигурационных файлов, размещаемых в каталоге **/etc** и его подкаталогах, в особености в подкаталоге **/etc/sysconfig**,. Давайте кратко охарактеризуем эти файлы, рассмотрим, какие задачи они выполняют и что можно с их помощью настроить. В приводимой ниже таблице каталоги выделены жирным шрифтом.

Файл	Назначение или содержимое
/etc/mandriva-release	Файл, содержащий название данного релиза системы. Используется для вывода баннера
/etc/modprobe.conf /etc/modprobe.preload /etc/modules **/etc/modprobe.d** **/etc/sysconfig/modules** /etc/modprobe.d/ldetect-lst.conf **/etc/modprobe.preload.d** /etc/modprobe.preload.d/floppy	Эти файлы и каталоги, видимо, используются на этапе загрузки модулей, заданных пользователем в **/etc/sysconfig/modules/*.modules** и в файле **/etc/modprobe.preload**, который содержит имена модулей ядра, загружаемых в процессе начальной загрузки системы. Этот файл используется с ядрами 2.5 и выше, для старых ядер аналогичную роль

	выполняет файл /etc/modules.
/etc/fstab	Назначение этого файла известно - он задает список монитруемых файловых систем и параметров монтирования для каждой из них
/etc/locale/ru **/etc/locale/ru_RU.UTF-8** /etc/sysconfig/i18n	Эти два каталога и файл **/etc/sysconfig/i18n** содержат переменные, которые задают региональные установки системы (локаль), то есть язык, форматы представления даты, времени, денежной единицы и т.д
/etc/sysconfig/init	В этом файле определяются переменные, задающие способ отображения сообщений, поступающих от скриптов запуска.
/etc/sysconfig/syslog	Этот файл задает опции для демонов протоколирования syslogd и klogd.
/etc/sysconfig/system	В моей системе имеет вид: SECURITY=3 CLASS=beginner LIBSAFE=no META_CLASS=download
/etc/sysconfig/harddisks	Задает опции для оптимизации работы с жесткими дисками
/etc/sysconfig/mouse	Задает тип мыши.
/etc/sysconfig/keyboard	Задает тип клавиатуры (ru, en), таблицу раскладки и переключатель (GRP_TOGGLE=alt_shift_toggle)
/etc/sysconfig/network	У меня содежит всего одну строку: "NETWORKING=yes"
/etc/sysconfig/clock	Задает переменные, определяющие временные установки (часовой пояс).

	В мой Mandriva имеет следующий вид: UTC=false ARC=false ZONE=Europe/Moscow
/etc/sysconfig/console /etc/sysconfig/console/consolefonts /etc/sysconfig/console/consoletrans /etc/sysconfig/console/default.kmap /etc/sysconfig/console/consolefonts/UniCyr_8x16.psf.gz	Задание консольного шрифта
/etc/sysconfig/network-scripts /etc/sysconfig/network-scripts/ifdown-post /etc/sysconfig/network-scripts/ifdown-ppp /etc/sysconfig/network-scripts/ifdown-tunnel /etc/sysconfig/network-scripts/ifdown-sit /etc/sysconfig/network-scripts/ifdown-sl /etc/sysconfig/network-scripts/ifdown-ipv6 /etc/sysconfig/network-scripts/ifup-aliases /etc/sysconfig/network-scripts/ifup-wireless /etc/sysconfig/network-scripts/ifup-bnep /etc/sysconfig/network-scripts/ifdown-eth /etc/sysconfig/network-scripts/ifup-ipv6 /etc/sysconfig/network-scripts/ifup-ipx /etc/sysconfig/network-scripts/ifup-plip /etc/sysconfig/network-scripts/ifup-plusb /etc/sysconfig/network-scripts/ifup-post /etc/sysconfig/network-scripts/ifup-ppp /etc/sysconfig/network-scripts/ifup-routes /etc/sysconfig/network-scripts/ifup-sit /etc/sysconfig/network-scripts/ifup-sl /etc/sysconfig/network-scripts/ifup-tunnel /etc/sysconfig/network-scripts/ifdown-bnep /etc/sysconfig/network-scripts/network-functions-ipv6 /etc/sysconfig/network-scripts/init.ipv6-global	Каталог **/etc/sysconfig/network-scripts** содержат скрипты, используемые при подключении/отключении комьютера к/от сети (в том числе на этапе загрузки).

/etc/sysconfig/network-scripts/network-functions /etc/sysconfig/network-scripts/ifdown-routes /etc/sysconfig/network-scripts/ifup-eth /etc/sysconfig/network-scripts/ifcfg-eth0 /etc/sysconfig/network-scripts/hostname.d/s2u /etc/sysconfig/network-scripts/ifdown.d/vpn /etc/sysconfig/network-scripts/ifup.d/vpn /etc/sysconfig/network-scripts/ifup.d/netprofile	
/etc/sysconfig/networking /etc/sysconfig/networking/ifcfg-lo	Файл **ifcfg-lo** задает настройки петлевого интерфейса.
/etc/sysconfig /etc/sysconfig/autofsck /etc/sysconfig/readonly-root	
/etc/sysconfig/usb	Этот файл определяет, надо ли искать в системе usb-устройства
/etc/udev /etc/udev/udev.conf /etc/udev/links.conf /etc/udev/agents.d/usb/usbcam /etc/udev/conf.d/mouse.conf /etc/udev/rules.d/60-persistent-input.rules /etc/udev/rules.d/05-udev-early.rules /etc/udev/rules.d/50-mdk.rules /etc/udev/rules.d/60-cdrom_id.rules /etc/udev/rules.d/61-net_config.rules /etc/udev/rules.d/60-persistent-storage.rules /etc/udev/rules.d/62-create_persistent.rules /etc/udev/rules.d/62-net.rules /etc/udev/rules.d/70-hotplug_map.rules /etc/udev/rules.d/95-udev-late.rules /etc/udev/rules.d/alsa.rules /etc/udev/rules.d/90-hal.rules /etc/udev/rules.d/60-dynamic.rules /etc/udev/rules.d/70-libsane.rules	Утилита **udev** вызывается из **rc.sysinit** для создания nodes блоковых устройств

/etc/udev/rules.d/70-libgphoto2.rules /etc/udev/rules.d/nomad.rules /etc/udev/rules.d/70-hplj10xx.rules /etc/udev/rules.d/61-block_config.rules /etc/udev/scripts/dvb.sh /etc/udev/scripts/floppy-extra-devs.sh /etc/udev/scripts/ide-model.sh	

7: Скрипт rc и запуск системных сервисов

Как мы видели в предыдущем разделе, скрипт **rc.sysinit** выполняет те задачи по начальной настройке системы, которые не зависят от уровня выполнения. Скрипт **/etc/rc.d/rc**, который запускается следующим, должен уже произвести перевод системы на тот уровень выполнения, который задан в файле **inittab** (или в командной строке). Напомню, что в файле **inittab** присутствует отдельная строка для каждого уровня выполнения. В этих строках вызывается один и тот же скрипт, и строки отличаются только аргументом вызова этого скрипта. Этот аргумент (или параметр) и задает уровень выполнения. Но, прежде чем рассматривать функции, выполняемые скриптом **rc**, надо сказать несколько слов о каталоге **/etc/rc.d.**

7.1. Структура каталога /etc/rc.d/

Этот каталог играет важную роль в процессе загрузки, поскольку он содержит основные скрипты (программы на языке командного процессора shell), служащие для организации процесса загрузки.

Каталог **rc.d** содержит следующий набор подкаталогов:

- rc0.d
- rc1.d
- rc2.d
- rc3.d
- rc4.d
- rc5.d
- rc6.d
- init.d

Если вы просмотрите (например, с помощью команды ls -l) содержимое подкаталогов **rcZ.d**, то увидите, что в этих подкаталогах содержатся не файлы, а только ссылки на файлы скриптов, находящиеся в других каталогах, а именно (за редким исключением), в каталоге **/etc/rc.d/init.d**. Для примера в листинге 13 приведен перечень файлов каталога **rc3.d** из системы ASPLinux 11.

Листинг 13. Файл /etc/rc.d/rc системы ASP Linux 11. Часть 1.

```
K01yum -> ../init.d/yum
K05saslauthd -> ../init.d/saslauthd
K10lirc -> ../init.d/lirc
K35winbind -> ../init.d/winbind
K38freshclam -> ../init.d/freshclam
K39clamd -> ../init.d/clamd
K50snmpd -> ../init.d/snmpd
K50snmptrapd -> ../init.d/snmptrapd
K66mDNSResponder -> ../init.d/mDNSResponder
K67nifd -> ../init.d/nifd
K68rpcidmapd -> ../init.d/rpcidmapd
K69rpcgssd -> ../init.d/rpcgssd
K74ntpd -> ../init.d/ntpd
K87named -> ../init.d/named
K89netplugd -> ../init.d/netplugd
K89rdisc -> ../init.d/rdisc

S05kudzu -> ../init.d/kudzu
S08iptables -> ../init.d/iptables
S10network -> ../init.d/network
S12syslog -> ../init.d/syslog
S13portmap -> ../init.d/portmap
S14nfslock -> ../init.d/nfslock
S25netfs -> ../init.d/netfs
S26lm_sensors -> ../init.d/lm_sensors
S50hplip -> ../init.d/hplip
S55cups -> ../init.d/cups
```

```
S56xinetd -> ../init.d/xinetd
S60nfs -> ../init.d/nfs
S85gpm -> ../init.d/gpm
S85httpd -> ../init.d/httpd
S90xfs -> ../init.d/xfs
S91smb -> ../init.d/smb
S97messagebus -> ../init.d/messagebus
S98haldaemon -> ../init.d/haldaemon
S99local -> ../rc.local
S99webmin -> /etc/init.d/webmin
```

Подкаталог **init.d** содержит уже не ссылки, а скрипты, управляющие работой для тех служб, которые обычно запускаются в системе (NFS, sendmail, cron, syslog, httpd и т. п.).

Рассмотрим, для примера, один из этих скриптов, /etc/init.d/network, который управляет запуском сетевых служб.

Если вы запустите его с опцией stop - /etc/init.d/network stop , работа с сетью будет остановлена. Если же выполнить его в следующей форме /etc/init.d/network start сеть будет снова запущена.

Различные скрипты из каталога **init.d** воспринимают различное число опций (или параметров запуска), но все они понимают опции stop, start и restart. Полный список допустимых опций запуска скрипта можно получить, запустив его на выполнение без аргументов:

Листинг 14. Вывод списка допустимых параметров вызова скрипта.

```
[root]# /etc/init.d/network
Usage: /etc/init.d/network {start|stop|restart|reload|status}
[root]# /etc/init.d/httpd
Usage: /etc/init.d/httpd
{start|stop|restart|condrestart|reload|status|fullstatus|graceful|help|configtest}
```

А можно просто заглянуть в текст скрипта и просмотреть, какие варианты заданы в команде **case**. В листинге 15 приведен для примера полный текст скрипта **/etc/rc.d/init.d/webmin** из системы ASP Linux 11. Выбор именно этого скрипта обусловлен

только тем, что он имеет относительно маленький объем. Но общие принципы построения таких скриптов он позволяет проиллюстрировать.

Листинг 15. Текст скрипта запуска-останова службы webmin

```
#!/bin/sh
# chkconfig: 235 99 10
# description: Start or stop the Webmin server
#
### BEGIN INIT INFO
# Provides: webmin
# Required-Start: $network $syslog
# Required-Stop: $network
# Default-Start: 2 3 5
# Default-Stop: 0 1 6
# Description: Start or stop the Webmin server
### END INIT INFO

start=/etc/webmin/start
stop=/etc/webmin/stop
lockfile=/var/lock/subsys/webmin
confFile=/etc/webmin/miniserv.conf
pidFile=/var/webmin/miniserv.pid
name='Webmin'

case "$1" in
'start')
        $start >/dev/null 2>&1 /dev/null 2>&1
        fi
        ;;
'stop')
        $stop
        RETVAL=$?
        if [ "$RETVAL" = "0" ]; then
                rm -f $lockfile
```

```
        fi
        ;;
'status')
        pidfile=`grep "^pidfile=" $confFile | sed -e 's/pidfile=//g'`
        if [ "$pidfile" = "" ]; then
                pidfile=$pidFile
        fi
        if [ -s $pidfile ]; then
                pid=`cat $pidfile`
                kill -0 $pid >/dev/null 2>&1
                if [ "$?" = "0" ]; then
                        echo "$name (pid $pid) is running"
                        RETVAL=0
                else
                        echo "$name is stopped"
                        RETVAL=1
                fi
        else
                echo "$name is stopped"
                RETVAL=1
        fi
        ;;
'restart')
        $stop && $start
        RETVAL=$?
        ;;
*)
        echo "Usage: $0 { start | stop | restart }"
        RETVAL=1
        ;;
esac
exit $RETVAL
```

Как видите, в начале этого скрипта задаются значения нескольких переменных (две первых переменных снова просто задают пути к скриптам запуска и останова данной службы), а затем вызывается команда выбора варианта **case**.

7.2. Скрипт /etc/rc.d/rc

Теперь, когда вы знаете, как организована структура объектов, используемых скриптом **/etc/rc.d/rc**, можно перейти к рассмотрению самого этого скрипта. В качестве примера ниже будет использоваться скрипт **/etc/rc.d/rc** из системы Mandriva Free 2007.1

При рассмотрении этого скрипта надо иметь в виду, что в общем случае он предназначен для перевода системы с одного уровня выполнения на другой. В процессе начальной загрузки этот скрипт переводит систему из однопользовательского режима на уровень, задаваемый по умолчанию. Поэтому, общий прицип работы этого скрипта заключается в следующем. Вначале он останавливает те службы, которые не нужны на новом уровне выполнения, для чего последовательно (в порядке присвоенных номеров NN) вызывает программы, на которые указывают ссылки с именами вида **KNNname** из каталога **/etc/rc.d/rcZ.d**, где Z — номер уровня выполнения. При этом программы вызываются с аргументом stop, т. е. соответствующие службы останавливаются. Затем так же последовательно перебираются ссылки с именами **SNNname** и соответствующие программы вызываются с параметром start. Из сказанного ясно, что буквы (символы) S и K, с которых начинаются имена ссылок в подкаталогах **rcX.d,** происходят от start и kill, соответственно. Отметим еще раз, что двузначные номера NN в именах ссылок определяют порядок запуска скриптов в каталоге, а name обычно является именем соответствующей программы (это имя приводится, скорее всего, просто для удобства администрирования, его отсутствие ничего бы не изменило).

Если вы хотите остановить или наоборот запустить какую-то из стандартных системных служб, вы можете сделать это, вызвав соответствующий скрипт, например,

```
[root]# /etc/rc.d/rc5.d/S10network start
или
[root]# /etc/rc.d/init.d/network start
```

В некоторых дистрибутивах существует удобная утилита service, которая позволяет сделать то же самое, не набирая в командной строке полный путь в соответствующему скрипту:

[root]# service network start

Однако, очевидно, что эта утилита будет корректно работать только в том случае, если скрипт вызова службы находится в стандартном каталоге. Впрочем, сама утилита **/sbin/service** представляет собой скрипт оболочки, в одной из начальных строк которого этот каталог и задается.

Если вы сами установили в систему какую-то программу и хотите, чтобы она всегда запускалась при старте системы, не требуя от вас каких-то дополнительных телодвижений, вы можете создать соответствующий управляющий скрипт, который разместить в стандартном каталоге **/etc/rc.d/init.d** (в некоторых дистрибутивах это будет **/etc/init.d/**, хотя в большинстве случаев один из этих каталогов является просто ссылкой на другой). После этого нужно просто создать ссылку вида **SNNnewprogram** в каталогах **/etc/rc.d/rcZ.d** для тех уровней выполнения, на которых вы желаете запускать вашу программу. Все это нетрудно проделать "вручную", хотя существуют специальные утилиты, с помощью которых можно легко скорректировать состав служб, запускаемых на разных уровнях. В Red Hat такая утилита называется **redhat-config-services**, в Debian - **rcconf**. Существуют и графические варианты таких утилит, например, **system-config-services**. Но, если вы желаете проделать все эти операции вручную, то теперь вы знаете, как это сделать.

Одна из последних ссылок вида **SXXname** (обычно это S99local), используемых скриптом **rc** на уровнях 2–5, является ссылка на скрипт **/etc/rc.d/rc.local**. Как сказано в самом этом файле, этот скрипт выполняется после всех других скриптов в процессе инициализации системы, поэтому если вы хотите, чтобы в процессе загрузки были выполнены какие-то дополнительные команды или ваши персональные настройки, то их целесообразно поместить именно сюда.

В последних версиях дистрибутивов скрипт **/etc/rc.d/rc.local** либо вообще пуст (хотя еще и сохраняется), либо выполняет очень ограниченные задачи. И некоторые авторы вообще не рекомендуют им пользоваться.

7.3. Системные сервисы (демоны)

Как вы могли видеть из приведенного описания работы скрипта **rc**, большое значение для результирующей конфигурации системы имеют программы, вызываемые из этого скрипта, размещенные по большей части в каталоге **/etc/rc.d/init.d**. Я буду называть программы

этого класса системными сервисами, поскольку они выполняют в системе служебные функции и во многом определяют функциональность системы.

8: Запуск процессов getty и login

Теперь давайте вспомним содержание раздела 5 настоящих заметок, где было рассказано, в каком порядке процесс **init** выполняет инструкции из файла **/etc/inittab** и приведен пример такого файла. Обратите внимание на то, что строки, в которых объявляется вызов скрипта **rc** помечены ключевым словом **wait** в третьем поле (поле action). Это означает, что **init** будет дожидаться, пока не завершиться выполнение скрипта **rc**, не предпринимая других действий.

Когда выполнение **rc N** завершится, init будет выполнять команды, заданные в строках с ключевым словом respawn для заданного уровня выполнения. Программы, запущенные с условием respawn будут повторно запускаться после завершения.

Например, в Red Hat для пятого уровня выполнения в файле **/etc/inittab** обычно прописаны следующие строки:

```
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
x:5:respawn:/etc/X11/prefdm -nodaemon
```

Первые 6 из этих строк обеспечивают запуск шести виртуальных консолей, доступ к которым (или переключение между которыми) пользователь Linux получает с помощью комбинаций клавиш ALT-F1 .... ALT-F6. Для этого init порождает процессы, именуемые getty-процессами (от "get tty" — получить терминал), и следит за тем, какой из процессов открывает какой терминал. Каждый getty-процесс устанавливает свою группу процессов, используя вызов системной функции setpgrp, открывает отдельную терминальную линию и обычно приостанавливается во время выполнения функции open до тех пор, пока машина не получит аппаратную связь с терминалом.

Когда функция open возвращает управление, getty-процесс (демон getty) отображает на экране содержимое файла **etc/issue** и запускает программу login (регистрации в системе), которая требует от пользователей, чтобы они идентифицировали себя указанием регистрационного имени и пароля.

Если пользователь не смог успешно зарегистрироваться, программа регистрации через определенный промежуток времени завершается, закрывая открытую терминальную линию, а процесс init порождает для этой линии следующий getty-процесс, открывающий терминал, вместо прекратившего существование (помните - ключевое слово respawn).

Последняя строка в приведенном выше примере, обозначенная идентификатором x, задает регистрацию пользователя в графическом режиме. Заметьте, что эта строка работает только на 5-ом уровне выполнения, поскольку другие уровни в Red Hat не предполагают использования графики. Но рассмотрение варианта загрузки в графическом режиме мы отложим до отдельного раздела настоящих заметок (или, скорее, отдельного приложения).

Как уже отмечалось где-то выше, вовсе не обязательно открывать 6 терминальных линий. Большинству пользователей более чем достаточно трех таких виртуальных терминалов. Тем более, надо иметь в виду, что запуск каждого лишнего процесса - это расходование системных ресурсов, в первую очередь - памяти. Чтобы сократить число открываемых терминальных линий, достаточно закомментировать "лишние" строки в файле **/etc/inittab**, как это было показано в листниге 10.

9: Старт оболочки Bash

Итак, демон getty запустил программу **login**, которая выводит предложение ввести имя пользователя (рис.3). Получив это имя, **login** обращается к файлу **/etc/passwd** за получением необходимых данных об этом пользователе, в частности, о его идентификаторе, идентификаторе группы, домашнем каталоге и о том, какую оболочку для него запускать. Одновременно выводится запрос на ввод пароля пользователя.
Когда пользователь введет пароль, программа **login** считывает его, криптует и сравнивает результат с тем, что лежит в соответствующей строке файла **etc/shadow**.

Если пользователь ввел правильный пароль, программа login, наконец, запускает командный процессор (оболочку). Как уже было сказано, какую именно оболочку запускать, определяется соответствующим полем в файле **/etc/passwd**.

Поскольку наиболее часто применяемым вариантом командной оболочки является **bash**, в дальнейшем будем рассматривать именно процесс загрузки этой программы (могу честно признаться, что главным основанием выбора этого варианта является то, что у меня используется именно эта оболочка и с другими я не работал).

9.1. Регистрационный shell

Теперь необходимо пояснить, что оболочка **bash** может запускаться в нескольких различных режимах. Рассказывать здесь полностью о том, чем режимы различаются, нет возможности, для этого загляните в man bash. Здесь же скажем только, что тот режим, который используется при запуске через программу **login**, называется режимом *"интерактивного регистрационного shell или login shell"* . Интерактивность в данном случае означает, что оболочка взаимодействует со стандартными потоками ввода-вывода, то есть стандартный входной и выходной потоки оболочки подключены к терминалу (просто существует и *неинтерактивный* режим запуска оболочки, например, для выполнения какого-то скрипта в фоновом режиме).

В случае запуска в режиме "login shell" процесс, в котором запускается оболочка, имеет тот же идентификатор, что и начальный getty-процесс, и является процессом, возглавляющим группу процессов. Чтобы убедиться в этом (вы же не обязаны верить мне на слово), выполните команду **ps –axf**, затем перейдите в свободную виртуальную консоль, запустите там программу Midnight Commander, вернитесь в ту консоль, где вы работали и снова запустите **ps –axf**. Сравнив вывод этой команды в первом и втором случае, вы увидите, что еще один процесс **getty** заменился на **shell**, причем из этой оболочки запущена команда **mc**.

В зависимости от режима запуска **bash** использует немного различные настроечные файлы. Какие именно из этих файлов используются в том или ином режиме запуска, рассказано в **man bash**. Я не буду приводить здесь описание всех вариантов, ограничусь только тем, который нас в данном случае интересует.

9.2. Запуск регистрационного экземпляра оболочки bash

При вызове **bash** в режиме интерактивного регистрационного интерпретатора команд, **bash** сначала читает и выполняет команды из файла **/etc/profile**, если этот файл существует. После прочтения этого файла, он последовательно ищет файлы **~/.bash_profile**,

~/.bash_login и **~/.profile**, читает и выполняет команды из первого же из них, который существует и доступен на чтение.

Давайте рассмотрим эти файлы из системы Mandriva Free 2007.1 и на этом примере разберем процесс загрузки оболочки **bash** в варианте регистрационного интерпретатора команд ("login shell").

Листинг 16. Файл /etc/profile системы Mandriva Free 2007.1.

```
# /etc/profile -*- Mode: shell-script -*-
# (c) MandrakeSoft, Chmouel Boudjnah

loginsh=1

if [ "$UID" -ge 500 ] && ! echo ${PATH} |grep -q /usr/games ; then
    PATH=$PATH:/usr/games
fi

umask 022

USER=`id -un`
LOGNAME=$USER
MAIL="/var/spool/mail/$USER"
HISTCONTROL=ignoredups
HOSTNAME=`/bin/hostname`
HISTSIZE=1000

if [ -z "$INPUTRC" -a ! -f "$HOME/.inputrc" ]; then
    INPUTRC=/etc/inputrc
fi

# some old programs still use it (eg: "man"), and it is also
# required for level1 compliance for LI18NUX2000
NLSPATH=/usr/share/locale/%l/%N

export PATH PS1 USER LOGNAME MAIL HOSTNAME INPUTRC NLSPATH
export HISTCONTROL HISTSIZE

for i in /etc/profile.d/*.sh ; do
        if [ -r $i ]; then
                . $i
```

```
        fi
done


unset i
```

Как видите,

- вначале задается переменная loginsh, которая должна свидетельствовать о том что это именно "login sheell".
- Далее проверяется, что пользователь не является привилегированным и если это так, то в число путей поиска добаляется каталог **/usr/games** (простому пользователю разрешается поиграть!).
- Переменной umask присваивается значение 022, что означает что создаваемые данным пользователем файлы по умолчанию будут заданы права доступа ???
- Задаются значения переменных:
- USER=`id -un`
- LOGNAME=$USER # переменным USER и LOGNAME присваивается имя пользователя;
- MAIL="/var/spool/mail/$USER" # переменной MAIL присваивается имя каталога,
- # где будет храниться почта данного пользователя;
- HISTCONTROL=ignoredups # если я правильно понимаю английский, будем
- # игнорировать повторы в истории команд;
- HOSTNAME=`/bin/hostname` # мне почему-то кажется, что где-то раньше мы эту
- # переменную уже задавали
- HISTSIZE=1000 # задаем число команд, которые будут храниться в истории команд.
- Если переменная INPUTRC имеет нулевую длину и (-a) файл $HOME/.inputrc не является простым файлом (то ли его не существует, то ли это сылка), то переменной INPUTRC присваивается значение /etc/inputrc.
- Для некоторых старых программ (например, man) требуется задать переменную NLSPATH=/usr/share/locale/%l/%N
- Экспортируются значения всех заданных переменных, чтобы они были доступны для всех процессов, запускаемых из данного экземпляра оболочки.

- Ну, и наконец, запускаются на выполнение все скрипты (файлы с расширением .sh) из каталога /etc/profile.d/, если только такие файлы существуют и для них установлено право на чтение.

Скрипты из **/etc/profile.d/** выполняют следующие задачи:

- - устанавливаются параметры локализации (скрипт 10lang.sh);
- - задаются общесистемные алиасы (скрипт alias.sh); причем вначале проверяется, не отказался ли пользователь от использования общесистемных алиасов; между прочим, среди этих общесистемных алиасов есть очень интересные, загляните в этот список в вашей системе и, возможно, вы будете использовать сокращенные написания некоторых часто употребляемых команд;
- - настройки клавиатуры немного изменяются в зависимости от аппаратной архитектуры (скрипт configure_keyboard.sh);
- - задаются установки, связанные с безопасностью (скрипт msec.sh);
- - создается директория для временных файлов (если ее не было) и указание на нее запоминается в переменной TMPDIR (скрипт tmpdir.sh);
- - корректируется поведение клавиши NumLock (скрипт numLock.sh);
- - значения многих переменных сразу после их задания экспортируются.

и так далее.

Как было сказано в начале этого раздела, после общесистемного файла **/etc/profile** оболочка **bash** последовательно ищет файлы **~/.bash_profile**, **~/.bash_login** и **~/.profile** и выполняет команды из этих файлов. Файлов ~/.bash_login и ~/.profile в моей системе не нашлось, а файл **~/.bash_profile** приведен в листинге 36:

Листинг 36. Файл ~/.bash_profile системы Mandriva Free 2007.1.

```
# .bash_profile

# Initialize keychain if needed
if [ -r $HOME/.ssh/identity -o -r $HOME/.ssh/id_dsa -o -r $HOME/.ssh/id_rsa ]; then
        if [ ! -d $HOME/.keychain ]; then
                keychain
    fi
```

```
fi

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH
unset USERNAME
```

Как видите, в этом скрипте корректируется переменная PATH (в список путей для поиска добавляется каталог ~/bin) и вызывается на выполнение еще один скрипт - **~/.bashrc**, причем комментарий к этому вызову сообщает, что тем самым снова задаются алиасы и функции. Впрочем, как видно из листинга 37, в самом вызываемом скрипте мало что задается - он всего лишь содержит вызов общесистемного скрипта **/etc/bashrc**.

Листинг 37. Файл ~/.bashrc системы Mandriva Free 2007.1.

```
# .bashrc

# User specific aliases and functions

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
```

Листинг 38. Файл /etc/bashrc системы Mandriva Free 2007.1.

```
# /etc/bashrc
```

```
# System wide functions and aliases
# Environment stuff goes in /etc/profile

# by default, we want this to get set.
# Even for non-interactive, non-login shells.
if [ "`id -gn`" = "`id -un`" -a `id -u` -gt 99 ]; then
        umask 002
else
        umask 022
fi

# are we an interactive shell?
if [ "$PS1" ]; then
   case $TERM in
        xterm*)
           PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME}:
${PWD}\007"'
           ;;
        *)
           ;;
   esac
   [ "$PS1" = "\\s-\\v\\$ " ] && PS1="[\u@\h \W]\\$ "

   if [ -z "$loginsh" ]; then # We're not a login shell
        # Not all scripts in profile.d are compatible with other shells
        # TODO: make the scripts compatible or check the running shell by
        # themselves.
        if [ -n "${BASH_VERSION}${KSH_VERSION}${ZSH_VERSION}" ]; then
        for i in /etc/profile.d/*.sh; do
            if [ -x $i ]; then
               . $i
            fi
           done
```

```
        fi
    fi
fi

unset loginsh
```

На этом загрузка интерактивного регистрационной оболочки завершена и пользователь может работать в системе. При этом он получает окружение, сформированное перечисленными файлами скриптов.

Отмечу, что если в Mandriva файла **~/.profile** не оказалось, то в системе OpenSUSE 10.2 от существует. В комментариях внутри этого файла говорится, что этот файл прочитывается, то есть вызывается на выполнение каждый раз, когда стартует регистрационный шелл. Все другие интерактивные оболочки вызывают только файл **~/.bashrc**.

Еще раз приведем список файлов, оказывающих влияние на то, какую именно конфигурацию оболочки мы получим после запуска. Этот список выглядит примерно следующим образом:

/etc/passwd

Глобальный файл, содержащий регистрационную информацию о пользователе

/etc/shadow

Глобальный файл паролей;

/etc/profile

Общесистемный файл инициализации (файл профилей), выполняется начальными командными интерпретаторами. Устанавливает общесистемную переменную $PATH и другие важнейшие переменные; заглянув в него, вы увидите, что в нем вызываются все файлы из подкаталога /etc/profile.d, в частности, файл, задающий параметры локализации системы;

/etc/bashrc

Глобальный файл конфигурации bash, устанавливает синонимы (алиасы) и функции, и т.п.;

/etc/issue

Содержит сообщение, выдаваемое на терминал перед входом в систему (перед запросом имени и пароля); однако редактировать этот файл с целью изменения

текста сообщения не стоит, потому что сам он формируется инициализационным скриптом /etc/rc.d/rc.local;

/etc/motd

Устанавливает сообщение, выдаваемое пользователю после входа в систему (после правильного ввода пароля).

~/.bash_profile

Личный файл инициализации (файл личного профиля), выполняется начальными командными интерпретаторами

~/.bashrc

Отдельный файл начального запуска для интерактивных командных интерпретаторов

~/.inputrc

Отдельный файл инициализации библиотеки **readline**

~/.profile

Личный файл инициализации (файл личного профиля), выполняется начальными командными интерпретаторами

Если вы что-то хотите поменять в той конфигурации, которую получаете при входе в систему, то, очевидно, имеет смысл подкорректировать ваш личный файл профиля **~/.bash_profile** или файл **~/.bashrc**.

Модель управления памятью в ОС Linux

В x86-архитектуре память разделяется на три типа адресов:

· **Логический адрес** - адрес расположения ячейки памяти, который может быть (а может и нет) связан непосредственно с физическим расположением. Логический адрес обычно используется при запросе информации из контроллера.

· **Линейный адрес** (или линейное адресное пространство) - это память, адресация которой начинается с 0. В Intel-архитектурах используется сегментированное адресное пространство, в котором память разделяется на сегменты размером 64KB, а сегментный регистр всегда указывает на базовый адрес адресуемого сегмента. 32-битный режим в этой архитектуре рассматривается как линейное адресное пространство, но в нем тоже используются сегменты.

· **Физический адрес** - адрес, представленный битами физической адресной шины. Физический адрес может отличаться от логического; в этом случае модуль управления памятью транслирует логический адрес в физический.

CPU использует два модуля для преобразования логического адреса в его физический эквивалент. Первый называется **модулем сегментации (segmented unit)**, а второй - **модулем разделения на страницы (paging unit)**.

Сущность модели сегментации состоит в том, что память управляется при помощи набора сегментов. Каждый сегмент имеет отдельное адресное пространство и состоит из двух компонентов:

· Базовый адрес - адрес некоторого места физической памяти;

· Значение длины, указывающее длину сегмента.

Сегментированный адрес тоже состоит из двух компонентов - **селектора сегмента** и **смещения в сегменте**. Селектор сегмента указывает на используемый сегмент (то есть, значения базового адреса и длины), а компонент смещения указывает смещение от базового адреса для реального доступа к памяти. Физический адрес реального месторасположения памяти является суммой значений смещения и базового адреса. Если смещение превышает длину сегмента, система генерирует нарушение защиты.

Модуль управления страницами преобразует линейные адреса в физические. Набор линейных адресов группируется вместе, образуя страницы. Эти линейные адреса непрерывны по своей природе - модуль управления страницами отображает эти наборы непрерывной памяти в соответствующий набор непрерывных физических адресов, называемых **страничными фреймами**. Модуль управления страницами представляет RAM разделенным на страничные фреймы фиксированного размера.

По этой причине разбиение на страницы имеет следующие преимущества:

· Права доступа, определенные для страницы, будут действительны для группы линейных адресов, формирующих страницу;

· Длина страницы равна длине страничного фрейма

Базовые концепции модуля сегментации в Linux

В Linux все сегментные регистры указывают на один и тот же диапазон адресов сегментов - другими словами, каждый использует один и тот же набор линейных адресов. Этопозволяет Linux использовать ограниченное число дескрипторов сегментов, то есть, все дескрипторы

могут храниться в **GDT (глобальной таблица дескрипторов)**. Двумя преимуществами этой модели являются:

· Управление памятью проще при использовании всеми процессами одинаковых значений сегментных регистров (когда они совместно используют одинаковый набор линейных адресов).

· Может быть достигнута совместимость с большинством архитектур. Некоторые RISC-процессоры тоже поддерживают такой ограниченный метод сегментирования.

Linux использует модель сегментирования на минимальном уровне, поэтому рассмотрим только базовые ее концепции. У каждого процесса в системе Linux есть адресное пространство, состоящее из трех логических сегментов: текста, данных и стека.

Текстовый сегмент (text segment) содержит машинные команды, образующие исполняемый код программы. Как правило, текстовый сегмент доступен только для чтения. **Сегмент данных (data segment)** содержит переменные, строки, массивы и другие данные программы. Он состоит из двух частей: инициализированные данные и неинициализированные данные. По историческим причинам вторая часть называется **BSS (Block Started by Symbol)**. Инициализированная часть сегмента данных содержит переменные и константы компилятора, значения которых должны быть заданы при запуске программы. Все переменные в BSS должны быть инициализированы в нуль после загрузки. Неинициализированные данные необходимы лишь с точки зрения оптимизации. На практике большинство глобальных переменных не инициализируются, и, таким образом, их начальное значение равно 0 (согласно семантике языка С). В целях экономии места в исполняемых файлах неинициализированные переменные собираются вместе после инициализированных, так что компилятору нужно только записать в заголовок слово, содержащее количество подлежащих выделению байтов.

Для того чтобы избежать выделения полной нулей физической страницы, во время инициализации Linux выделяет статическую нулевую страницу (защищенную от записи страницу, заполненную нулями). Когда процесс загружается, указатель на область его неинициализированных данных устанавливается на эту нулевую страницу. Когда процесс пытается писать в эту область, то вмешивается механизм «копирования при записи» и процессу выделяется настоящая страница.

В отличие от текстового сегмента (который не может изменяться), сегмент данных изменяться может. Более того, многим программам требуется динамическое выделение памяти во время выполнения. Для этого операционная система Linux разрешает сегменту

данных расти при выделении памяти и уменьшаться при освобождении памяти. Дескриптор адресного пространства процесса содержит информацию о диапазоне динамически выделенных областей памяти процесса (который обычно называется кучей —*heap*).

Третий сегмент —это **сегмент стека (stack segment)**. На большинстве компьютеров он начинается около старших адресов виртуального адресного пространства и растет вниз к 0. Если указатель стека оказывается ниже нижней границы сегмента стека, то происходит аппаратное прерывание, при котором операционная система понижает границу сегмента стека на одну страницу. Программы не управляют явно размером сегмента стека. Когда программа запускается, ее стек содержит все переменные окружения (оболочки), а также командную строку, введенную в оболочке для вызова этой программы. Таким образом, программа может узнать параметры, с которыми она была запущена.

Большинством систем Linux поддерживаются **текстовые сегменты совместного использования (shared text segemts)**. Означает это возможную компоновку физической памяти, когда оба процесса совместно используют один и тот же фрагмент текста. Отображение выполняется аппаратным обеспечением виртуальной памяти. Сегменты данных и стека никогда не бывают общими, кроме как после выполнения системного вызова *fork* — и то только те страницы, которые не модифицируются. Если размер одного из сегментов должен быть увеличен, то отсутствие свободного места в соседних страницах памяти не является проблемой, поскольку соседние виртуальные страницы памяти не обязаны отображаться на соседние физические страницы.

На некоторых компьютерах аппаратное обеспечение поддерживает раздельные адресные пространства для команд и для данных. Если такая возможность есть, то система Linux может ее использовать.

Реализация управления памятью в Linux

Каждый процесс системы Linux на 32-разрядной машине обычно получает 3 Гбайт виртуального адресного пространства для себя, а оставшийся 1 Гбайт памяти резервируется для его страничных таблиц и других данных ядра. 1 Гбайт ядра не виден в пользовательском режиме, но становится доступным, когда процесс переключается в режим ядра.

Память ядра обычно находится в нижних физических адресах, но отображается в верхний гигабайт виртуального адресного пространства процесса (между адресами 0xC0000000 и 0xFFFFFFFF, это диапазон от 3 до 4 Гбайт). Адресное пространство создается при создании

процесса и перезаписывается системным вызовом *exec*. Чтобы несколько процессов могли совместно использовать физическую память, Linux выполняет следующие действия:

· Отслеживает использование физической памяти;

· Выделяет при необходимости дополнительную память пользовательским процессам и компонентам ядра;

· Динамически отображает области физической памяти на адресное пространство разных процессов;

· Динамически доставляет в память и убирает из нее исполняемые программы, файлы и прочую информацию состояния (по мере необходимости*).*

Управление физической памятью. Страничная организация

Вследствие различных аппаратных ограничений, имеющихся на многих системах, не вся их физическая память одинакова — особенно в отношении ввода-вывода и виртуальной памяти. Linux различает три зоны памяти:

· **ZONE_DMA** — это страницы, которые можно использовать для операций **DMA (Direct Memory Access)**;

· **ZONE_NORMAL** —это нормальные отображаемые страницы;

· **ZONE_HIGHMEM** — это страницы с адресами в верхней области памяти, которые не имеют постоянного отображения.

Точные границы и компоновка этих зон памяти зависит от архитектуры. На платформах x86 некоторые устройства могут выполнять операции DMA только в первых 16 Мбайт адресного пространства, следовательно **ZONE_DMA** находится в диапазоне от 0 до 16 Мбайт. Кроме того, оборудование не может напрямую отображать адреса памяти выше 896 Мбайт, поэтому **ZONE_HIGHMEM** — это все, что находится выше этой отметки. **ZONE_NORMAL** — это все, что находится между ними. Поэтому на платформах x86 первые 896 Мбайт адресного пространства Linux отображаются напрямую, а остальные 128 Мбайт адресного пространства ядра используются для доступа к верхним областям памяти. Ядро поддерживает структуру *zone* для каждой из этих трех зон и может выполнять выделение памяти для этих трех зон по отдельности. Основная память в Linux состоит из трех частей. Первые две части — ядро и карта памяти — прикреплены в памяти (то есть, никогда не вытесняются). Остальная память разделена на страничные блоки, каждый из которых может содержать страницу текста, данных или стека, страницу с таблицей страниц

или находится в списке свободных. Единицей управления памятью является страница, и почти все компоненты управления памятью работают с точностью до страниц.

Ядро поддерживает карту памяти, которая содержит всю информацию об использовании физической памяти системы (такую как зоны, свободные страничные блоки и т. д.).

Эта информация организована следующим образом.

Прежде всего, Linux поддерживает массив **дескрипторов страниц (page descriptors)** типа *page* для каждого физического страничного блока системы (он называется *mem_map*). Каждый дескриптор страницы содержит:

· Указатель на адресное пространство, к которому принадлежит страница (если она не свободна);

· Пару указателей, которые позволяют ему сформировать дважды связанный список с другими дескрипторами (например, чтобы собрать вместе все свободные страничные блоки);

· Несколько прочих полей.

Размер дескриптора страницы равен 32 байтам, поэтому вся mem_map может занимать менее 1% физической памяти (при размере страничного блока в 4 Кбайт). Поскольку физическая память разделена на зоны, то для каждой зоны Linux поддерживает **дескриптор зоны (zone descriptor)**. Этот дескриптор содержит информацию об использовании памяти в зоне, такую как:

· Количество активных и неактивных страниц;

· Нижний и верхний пределы для алгоритма замещения страниц (который будет описан далее в этой же главе);

· Много других полей.

Кроме того, дескриптор зоны содержит массив свободных областей, *i*-й элемент этого массива указывает первый дескриптор страницы первого блока из 2*i* свободных страниц. Поскольку в наличии может иметься много блоков из 2*i* свободных страниц, то Linux использует в каждом элементе *page* пару указателей на дескрипторы страниц (чтобы связать их вместе). Эта информация используется в операциях выделения памяти в Linux. Linux является переносимой на архитектуру **NUMA (Non-Uniform Memory Access** схема реализации компьютерной памяти, используемая в мультипроцессорных системах, когда время доступа к памяти определяется её расположением по отношению к процессору), где разные физические адреса имеют очень сильно отличающиеся времена доступа. Поэтому

для различения физической памяти различных узлов (и во избежание выделения структур данных в чужих узлах) Linux использует **дескриптор узла (node descriptor)**.

Каждый дескриптор узла содержит информацию об использовании памяти и зонах данного конкретного узла. На платформах UMA (**Uniform Memory Access** - архитектура многопроцессорных компьютеров с общей памятью) система Linux описывает всю память припомощи одного дескриптора узла. Первые несколько битов каждого дескриптора страниц используются для идентификации узла и зоны, к которой принадлежит данный страничный блок. Чтобы механизм подкачки был эффективен на архитектурах x32 и x64, система Linux использует **четырехуровневую страничную организацию**. Трехуровневая схема была реализована в системе для процессора Alpha, она была расширена после версии Linux 2.6.10, и начиная с версии 2.6.11 используется четырехуровневая схема. Каждый виртуальный адрес разбивается на пять полей. Поля каталогов используются как индекс в соответствующем каталоге страниц (каждый процесс имеет свой приватный каталог). Обнаруженное значение является указателем на один из каталогов следующего уровня, которые тоже проиндексированы полем из виртуального адреса. Выбранный элемент среднего каталога страниц указывает на окончательную таблицу страниц, проиндексированную полем страницы из виртуального адреса. Найденный здесь элемент содержит указатель на нужную страницу. На компьютерах с процессором Pentium используется только двухуровневая организация страниц. В этом случае каждый из верхних и средних каталогов страниц содержит только одну запись. Таким образом, элемент глобального каталога фактически указывает на таблицу страниц. При необходимости может использоваться и трехуровневая страничная организация, для этого размер поля верхнего каталога страниц устанавливается в нуль.

Физическая память используется для различных целей. Само ядро жестко фиксировано — ни одна его часть никогда не выгружается на диск. Остальная часть памяти доступна для страниц пользователей, страничного кэша и других задач. Страничный кэш содержит страницы с блоками файлов, которые были недавно считаны или были считаны заранее (в надежде на то, что они скоро могут понадобиться), либо страницы с блоками файлов, которые надо записать на диск (например, созданные процессами пользовательского режима, которые были выгружены на диск). Его размер динамически меняется, причем он состязается за один и тот же пул страниц с пользовательскими процессами. Страничный кэш в действительности не является настоящим отдельным кэшем, а представляет собой набор страниц пользователя, которые более не нужны и ожидают выгрузки на диск. Если страница, находящаяся в страничном кэше, потребуется снова (прежде, чем она будет

удалена из памяти), то ее можно быстро получить обратно.Кроме того, операционная система Linux поддерживает динамически загружаемые модули, в основном драйверы устройств. Они могут быть произвольного размера и каждому из них должен быть выделен непрерывный участок в памяти ядра. Для выполнения этих требований система Linux управляет физической памятью таким образом, что она может получить по желанию участок памяти произвольного размера. Для этого используется алгоритм, известный как **«приятельский» алгоритм (buddy algorithm)**. Он описан ниже.

Алгоритм выделения памяти

Linux поддерживает несколько механизмов выделения памяти. Главным механизмом для выделения новых страничных блоков физической памяти является **распределитель страниц (page allocator)**, который работает при помощи широко известного «приятельского» алгоритма.

Основная идея управления блоками памяти заключается в следующем. Изначально память состоит из единого непрерывного участка. Представим, что размер этого участка равен 64 страницам. Когда поступает запрос на выделение памяти, он сначала округляется до степени двух, например до 8 страниц. Затем весь блок памяти делится пополам. Если получившиеся в результате этого деления участки памяти все еще слишком велики, нижняя половина делится пополам еще и еще и так до тех пор, пока не получается участок памяти нужного размера, после чего он предоставляется вызвавшему процессу. Теперь предположим, что приходит второй запрос на 8 страниц. Он может быть удовлетворен немедленно. Следом поступает запрос на 4 страницы. Наименьший из имеющихся участков делится надвое и выделяется половина. Затем освобождается второй 8-страничный участок. Наконец, освобождается другой 8-страничный участок. Поскольку эти два смежных только что освободившихся участка были «приятелями» (то есть они вышли из одного 16 страничного блока), то они снова объединяются в 16-страничный блок. Операционная система Linux управляет памятью при помощи этого «приятельского» алгоритма. К нему дополнительно имеется массив, в котором первый элемент представляет собой начало списка блоков размером в 1 единицу, второй элемент является началом списка блоков размером в 2 единицы, третий элемент — началом списка блоков размером в 4 единицы и т. д. Таким образом, можно быстро найти любой блок размером, кратным степени 2.

Этот алгоритм приводит к существенной внутренней фрагментации, так как, если вам нужен 65-страничный участок, то вы должны будете запросить и получите 128 страничный блок. Чтобы решить эту проблему, в системе Linux есть второй механизм выделения памяти

— **распределитель фрагментов (slab allocator)**, выбирающий блоки памяти при помощи «приятельского» алгоритма, а затем нарезающий из этих блоков более мелкие куски и управляющий этими более мелкими кусками по отдельности. Поскольку ядро часто создает и уничтожает объекты определенных типов (например, task_struct), то оно зависит от так называемых **кэшей объектов (object caches)**. Эти КЭШи состоят из указателей на один или несколько кусков, в которых может храниться несколько объектов одного типа. Каждый из этих кусков может быть полным, частично заполненным или пустым. Например, когда ядру нужно выделить новый дескриптор процесса (то есть новую **task_struct**), оно ищет в кэше объектов структуры задач и сначала пытается найти частично заполненный кусок и выделить новую task_struct в нем. Если такого куска нет, то оно просматривает список пустых кусков. Наконец (при необходимости) оно выделит новый кусок, поместит в него новую структуру задач и свяжет этот кусок с кэшем объектов структур задач. Служба ядра **kmalloc**, которая выделяет физически смежные области памяти в адресном пространстве ядра, фактически построена поверх интерфейса кусков и кэша объектов (описанного здесь).

Есть также и третий механизм выделения памяти — **vmalloc**, — который используется в тех случаях, когда запрошенная память должна быть смежной только в виртуальном пространстве, а не в физической памяти. На практике это справедливо для большей части запрашиваемой памяти. Одним из исключений являются устройства, которые живут на другой стороне от шины памяти и блока управления памятью (и поэтому не понимают виртуальных адресов). Однако использование vmalloc приводит к некоторому падению производительности, поэтому он применяется в основном для выделения больших количеств непрерывного виртуального адресного пространства (например, для динамической вставки модулей ядра). Все эти механизмы выделения памяти ведут свое происхождение от SystemV.

Представление виртуального адресного пространства

Виртуальное адресное пространство делится на однородные, непрерывные и выровненные по границам страниц области. То есть каждая область состоит из участкасмежных страниц с одинаковой защитой и страничной организацией. Текстовый сегмент и отображенные файлы являются примерами таких областей. Между областями виртуального адресного пространства могут быть дыры. Любая ссылка на дыру приводит к фатальной страничной ошибке. Размер страницы фиксирован, например для Pentium он равен 4 Кбайт, а для Alpha он равен 8 Кбайт. Начиная с Pentium (который поддерживает страничные блоки размером в 4 Мбайт), Linux умеет поддерживать увеличенные страничные блоки размером по 4 Мбайт. Кроме того, в режиме расширения физических адресов (PAE, Physical Address Extension),

который используется на некоторых 32-битных архитектурах для увеличения адресного пространства процессов сверх 4 Гбайт, поддерживается также и размер страниц в 2 Мбайт. Каждая область описывается в ядре элементом **vm_area_struct**. Все эти элементы (для одного процесса) связываются вместе в список, отсортированный по виртуальным адресам (чтобы все страницы можно было найти). Когда список становится слишком длинным (более 32 элементов), для ускорения по нему поиска создается дерево. В элементе vm_area_struct перечислены свойства области. Эти свойства включают:

· Режим защиты (например, «только для чтения» или «чтение/запись»);

· Информацию о том, закреплен ли он в памяти (не подкачивается) и в каком направлении он растет (для сегментов данных —вверх, для стеков — вниз);

· Информацию о том, является ли область приватной для процесса или используется совместно с одним или несколькими другими процессами.

После выполнения вызова *fork* система Linux делает копию списка областей для дочернего процесса, но настраивает указатели в родительском и дочернем процессах на одни и те же таблицы страниц. Области помечаются как «чтение/запись», но страницы помечаются как «только для чтения». Если какой-то процесс пытается сделать запись в страницу, то происходит ошибка защиты и ядро видит, что область логически доступна для записи, а страница — нет. Тогда ядро дает процессу копию страницы и помечает ее как «чтение/запись». С помощью этого механизма реализовано «копирование при записи».

В структуре vm_area_struct также записано, имеет ли область резервное хранение надиске, — и если имеет, то где. Текстовые сегменты используют в качестве резервного хранения исполняемый двоичный файл, а отображаемые на память файлы — дисковый файл.

Другие области (такие, как стек) не имеют резервного хранения (до момента вытеснения в файл подкачки).

Дескриптор памяти верхнего уровня mm_struct собирает:

· Информацию обо всех областях виртуальной памяти (принадлежащих адресному пространству);

· Информацию о различных сегментах (текста, данных, стека);

· Информацию пользователях, совместно использующих это адресное пространство и т. д.

Ко всем элементам адресного пространства в mm_struct можно обращаться через их дескриптор памяти (двумя способами). Во-первых, они организованы в связанные списки, упорядоченные по адресам виртуальной памяти. Этот способ полезен тогда, когда нужно

обращаться ко всем областям виртуальной памяти, или тогда, когда ядро ищет для выделения область виртуальной памяти определенного размера. Кроме того, элементы структуры vm_area_struct организованы в бинарное дерево (это оптимизированная для быстрого поиска структура). Этот метод используется, когда нужно обратиться к определенной виртуальной памяти. Обеспечивая доступ к элементам адресного пространства процессов этими двумя способами, Linux использует больше памяти на процесс, но позволяет различным операциям ядра использовать тот метод доступа, который более эффективен длятекущей задачи.

Подкачка в Linux

В ранних системах UNIX использовался **процесс подкачки (swapper process)**, который перемещал процессы целиком между памятью и диском (когда все активные процессы не помещались в физической памяти). Linux (подобно другим современным версиям UNIX) больше не перемещает процессы целиком. Подсистема подкачки работает с точностью до страниц. Основная идея подкачки в Linux проста: процессу не обязательно находиться целиком в памяти для того, чтобы выполняться. Все, что нужно — это пользовательская структура и таблицы страниц. Если они подкачаны в память, то процесс считается находящимся в памяти и может планироваться для выполнения. Страницы сегментов текста, данных и стека подкачиваются динамически (по одной) по мере появления ссылок на них. Если пользовательская структура и таблица страниц не находятся в памяти, то процесс не может выполняться до тех пор, пока процесс подкачки не доставит их в память.

Подкачка реализована частично ядром и частично новым процессом, называемым **демоном страниц (page daemon)**. Демон страниц — это процесс 2 (процесс 0 — это процесс idle — традиционно называемый **своппером**, а процесс 1 — это init). Как и все демоны, демон страниц работает периодически. После пробуждения он осматривается — есть ли для него работа. Если он видит, что количество страниц в списке свободных слишком мало, то он начинает освобождать страницы.

Операционная система Linux является системой с подкачкой страниц по требованию (без упреждающей подкачки) и без концепции рабочего набора (хотя в ней есть системный вызов для указания пользователем страницы, которая ему может скоро понадобиться). Текстовые сегменты и отображаемые на адресное пространство памяти файлы подгружаются из соответствующих им файлов на диске. Все остальное выгружается либо в раздел подкачки (если он присутствует), либо в один из файлов подкачки (фиксированной длины), которые называются **областью подкачки (swap area)**. Файлы подкачки могут динамически добавляться и удаляться, и у каждого есть свой приоритет. Подкачка страниц

из отдельного раздела диска, доступ к которому осуществляется как к отдельному устройству, не содержащему файловой системы, более эффективна (чем подкачка из файла) по нескольким причинам. Во-первых, не требуется отображение блоков файла в блоки диска.

Во-вторых, физическая запись может иметь любой размер, а не только размер блока файла. В-третьих, страница всегда пишется на диск в виде единого непрерывного участка, а при записи в файл подкачки это может быть и не так. Страницы на устройстве подкачки или разделе подкачки не выделяются до тех пор, пока они не потребуются. Каждое устройство или файл подкачки начинается с битового массива, в котором сообщается, какие страницы свободны. Когда страница, у которой нет резервного хранения на диске, должна быть удалена из памяти, то из разделов (или файлов) подкачки, в которых еще есть свободное место, выбирается раздел (или файл) с наивысшим приоритетом, и в нем выделяется страница. Как правило, раздел подкачки (если таковой имеется) имеет более высокий приоритет, чем любой файл подкачки. Таблица страниц обновляется, чтобы отразить тот факт, что страница больше не присутствует в памяти (то есть устанавливается бит «страница отсутствует») и ее местоположение на диске записывается в элемент таблицы страниц.

Управление памятью - это большой, комплексный и трудоемкий набор задач, к которому, такие компоненты как планирование, разбиение на страницы и взаимодействие процессов, предъявляют серьезные требования.

Используемая в Linux модель памяти довольно проста, что должно обеспечить переносимость программ, а также реализацию операционной системы Linux на машинах с сильно отличающимися блоками управления памятью, варьирующимися от элементарных до сложного оборудования со страничной организацией.

Операционная система Windows

Файловая система ОС Windows

Вся информация, предназначенная для долговременного использования, хранится в файлах, реестрах и базах данных. Файл (file) – совокупность данных, хранящаяся на каком-либо носителе информации, доступ к которой осуществляется по ее имени. Файл, таким образом, противопоставляется другим объектам, доступ к которым осуществляется по их адресу [1].

Обычно единственным способом работы с файлами является применение системы управления файлами или иначе – файловой системы (ФС).

Файловая система – это часть операционной системы, включающая:

- совокупность всех файлов на носителе информации (магнитном или оптическом диске, магнитной ленте и др.);
- наборы структур данных, используемых для управления файлами (каталоги и дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске и др.);
- комплекс системных программных средств, реализующих различные операции над файлами (создание, уничтожение, чтение, запись) [2].

Задачи, решаемые ФС, во многом определяются способом организации вычислительного процесса (наиболее простые – в однопрограммных и однопользовательских ОС, наиболее сложные – в сетевых ОС.).

В мультипрограммных, многопользовательских ОС задачами файловой системы являются [3]:

• соответствие требованиям управления данными и требованиям состороны пользователей, включающим возможность хранениянданных и выполнения операций с ними;

• гарантирование корректности данных (Уровень корректной файловой системы означает, что все файлы на диске сохранены в корректном виде, то есть все транзакции, связанные с файлами, были завершены корректно перед запуском резервного копирования.Однако несохраненные данные приложений могут быть утеряны);

• оптимизация производительности, как с точки зрения системы(пропускная способность), так и с точки зрения пользователя(время отклика);

• поддержка ввода-вывода для различных типов устройств хранения информации;

• минимизация возможных потерь или повреждений данных:

Рассмотрим пример для NTFS:NTFS обеспечивает восстановление файловой системы на основе концепции атомарной транзакции (atomictransaction). Атомарные транзакции — это метод обработки изменений в БД, при котором сбои в работе системы не нарушают корректности или целостности БД. Суть: некоторые операции над БД(транзакции) выполняются по принципу «все или ничего».Отдельные изменения на диске, составляющие транзакцию, выполняются атомарно: в ходе транзакции на диск должны быть внесены все требуемые изменения. Если транзакция прервана аварией системы, часть изменений, уже внесенных к этому моменту, нужно отменить. Такая

отмена – откат (rollback). После отката БД возвращается в исходное согласованное состояние, в котором она была до начала транзакции.

Защита пользовательских данных на основе технологии обработки транзакций предусматривается в большинстве СУБД для Windows, например в Microsoft SQL Server. Microsoft не стала реализовать восстановление пользовательских данных на уровне ФС, так как приложения обычно поддерживают свои схемы восстановления, оптимизированные под тот тип данных, с которыми они работают[9];

• защита файлов от несанкционированного доступа (безопасность обеспечиваетсяза счет использования разрешений и системы шифрования файлов EFS (EncryptionFileSystem – шифрованная ФС) для ограничения доступа конкретных пользователей к определенным файлам);

• обеспечение поддержки совместного использования файлов несколькими пользователями (в том числе средства блокировки файла и его частей, исключение тупиков, согласование копий и т.п.).

Рассмотрим подробнее ситуацию совместного доступа к файлу. Операция открытия файла влечет создание объекта "открытый файл". Его специфика в том, что он содержит лишь уникальные данные (например, указатель текущей позиции), тогда как собственно файл - совместно используемые данные. Поэтому, если два раза осуществить операцию открытия одного и того же файла, то система создаст два объекта "файл". Очевидно, что потоки должны синхронизировать доступ к совместно используемым файлам или каталогам, чтобы получить предсказуемый результат. Между двумя операциями read одного потока другой поток может модифицировать данные, что для многих приложений неприемлемо. ОС Windows предлагает стандартное решение данной проблемы на уровне пользователя - предоставить возможность одному из потоков захватить часть файла между двумя записями для монопольного доступа. Для этого используются Win32-функции LockFile и UnlockFile [8];

• обеспечение стандартизированного набора подпрограмм интерфейса ввода-вывода.

Минимальным набором требований к файлам системы со стороныпользователя диалоговой системы общего назначения можно считать следующую совокупность возможностей, предоставляемую пользователю:

1) создание, удаление, чтение и изменение файлов;

2) контролируемый доступ к файлам других пользователей;

3) управление доступом к своим файлам;

4) реструктурирование файлов в соответствии с решаемой задачей;

5) перемещение данных между файлами;

6) резервирование и восстановление файлов в случае повреждения;

7) доступ к файлам по символьным именам.

Система Windows знает о расширении имен файлов и назначает каждому расширению определенное значение. Пользователи могут регистрировать расширения в ОС, указывая программу, которая станет их «владельцем». При двойном щелчке мыши на имени файла запускается программа, назначенная этому расширению, с именем файла в качестве параметра [4].

В Windows существуют следующие файловые системы: FAT, FAT32, NTFS илиReFS.Для выбора ФС нужно учесть, для чего будет использована ФС (будь то сервер или рабочая станция); количество дисков; требования к безопасности.

Файловая система FAT

Файловая система FAT (FileAllocationTable — таблица размещения файлов) получиласвое название благодаря простой таблице, в которой указываются:

- непосредственно адресуемые участки логического диска, отведенные для размещенияв них файлов или их фрагментов;
- свободные области дискового пространства;
- дефектные области диска (эти области содержат дефектные участки и не гарантируютчтение и запись данных без ошибок) [5]

Файловая система FAT может использоваться с WindowsNT/2000, Windows 9x, WindowsforWorkgroups, MS-DOS и OS/2.

FAT32

Данная система обеспечивает оптимальный доступ к жестким дискам, повышая скорость и производительность всех операций ввода/вывода. FAT32 представляет собой усовершенствованную версию файловой системы FAT, предназначенную для использования на томах, объем которых превышает 2 Гбайт. Windows 2000 продолжает поддерживать файловую систему FAT, а также добавляет дополнительную поддержку для FAT32.

Возможности файловой системы FAT32 намного превышают возможности файловой системы FAT16. Так, эта файловая система поддерживает жесткие диски, размер которых может достигать теоретического предела 2 терабайта.

NTFS

С точки зрения пользователя ФС NTFS организует файлы по каталогам и сортирует их.Но на диске нет специальных объектов и отсутствует зависимость от особенностей установленного оборудования. Кроме того, на диске отсутствуют специальные хранилища данных (таблицы FAT и суперблокиHPFS).

ЦельNTFS:

- обеспечение надежности, имеющей большое значение для высокопроизводительных систем и файловых серверов.
- предоставление платформы дополнительной функциональности.
- поддержка требований POSIX.
- устранение ограничений, характерных для файловых систем FAT и HPFS.
- надежность

Для обеспечения надежности NTFS особое внимание было уделено трем основным вопросам:

1) способности к восстановлению;

Для обеспечения способности к восстановлению NTFS отслеживает все транзакции в отношении файловой системы. Выполнение команды CHKDSK в файловой системе FAT или HPFS служит для проверки последовательности указателей в пределах каталога, размещения и таблицы файлов. Файловая система NTFS хранит журнал операций с этими компонентами. Таким образом, для восстановления связности системы необходимо с помощью команды CHKDSK выполнить «откат» транзакций до последней точки фиксации.

2) устранению неустранимых ошибок одного сектора;

При использовании FAT или HPFS сбой сектора, в котором хранится один из специальных объектов файловой системы, приводит к возникновению неустранимой ошибки одного сектора. В NTFS эта проблема решается двумя способами. Во-первых, специальные объекты не используются, а все имеющиеся на диске объекты отслеживаются и защищаются. Во-вторых, существует несколько копий (число зависит от размера тома) основной таблицы файлов.

3) экстренному исправлению.

Подобно версиям HPFS для OS/2, NTFS поддерживает экстренное исправление.

Дополнительные функции:

- Основное предназначение конфигурации операционной системы WindowsNT на любом уровне является обеспечение платформы, которую можно использовать в

качестве модуля при построении других систем, и NTFS не является исключением. Эта файловая система представляет собой гибкую платформу с широкими функциональными возможностями, которую могут использовать другие файловые системы. Кроме того, в NTFS полностью реализована модель безопасности WindowsNT и поддержка нескольких потоков данных. Файл данных перестал быть отдельным потоком данных. Кроме того, пользователи могут добавлять собственные атрибуты файлов.

- Поддержка POSIX

Из всех поддерживаемых файловых систем NTFS наиболее полно соответствует POSIX.1. В ней реализована поддержка следующих требований POSIX.1:

- назначениеимен с учетом регистра;

(согласно POSIX.README.TXT, Readme.txt и readme.txt – это разные файлы)

- дополнительная отметка времени;

- дополнительный штамп времени для сохранения времени последнего доступа к файлу;

- жесткие связи.

(Жесткой связью называется такая связь, когда два различных имени файла (которые могут быть расположены в разных каталогах) указывают на одни и те же данные)

- Устранение ограничений

Во-первых, в NTFS значительно – до 2^64 байт (16 экзабайт или 18 446 744 073 709 551 616 байт) – увеличен допустимый раздел файлов и томов. В NTFS для решения проблемы фиксированного размера сектора снова применена концепция кластеров, ранее использованная в файловой системе FAT. Это было сделано для улучшения аппаратной независимости операционной системы WindowsNT при ее использовании с жесткими дисками, изготовленными по другой технологии. Таким образом, была принята точка зрения, что деление диска на секторы размером 512 не всегда является оптимальным. Размер кластера определяется кратным числом единичных блоков жесткого диска. Кроме того, для задания имен файлов используется кодировка Юникод и наряду с длинными именами обеспечена поддержка формата 8.3. [11]

NTFS лучше всего подходит для использования с томами размером более 400 МБ. С увеличением размера тома производительность файловой системы NTFS не падает, как у FAT.

Благодаря способности к восстановлению в NTFS отсутствует необходимость использования каких-либо программ восстановления диска[6].

ReFS

ReFS (Resilientfilesystem) — локальная файловая система, используемая в WindowsServer 2012, WindowsServer 2012 R2, бета-версиях Windows 8, Windows 8.1. Является дальнейшим развитием NTFS.Не поддерживается Windows 7 и более ранними системами.

Особенности:

Улучшенная надёжность хранения информации на диске структур. ReFS использует B+ деревья (принцип, сходный с хранением данных в реляционных СУБД) как для метаданных, так и для содержимого файлов. Свободное место на диске описывается 3 отдельными иерархическими таблицами для малых, средних и больших фрагментов свободного пространства. Имена файлов и длина пути ограничена 32 кибибайтами, для их хранения используется Unicode.

Поддержка стратегии Copy-on-write (копирование при записи, выделение при записи) для метаданных, при которой любые транзакции ФС не перезаписывают старые метаданные, а записываются в новый блок и организуются в пачки. Для всех метаданных в ReFS используются 64-битные контрольные суммы, хранящиеся независимо. Данные файлов могут иметь контрольную сумму в отдельном потоке (атрибут «integrity»). В случае, если содержимое файлов или метаданных не соответствует контрольным суммам, не требуется отключение файловой системы для удаления или восстановления таких данных. За счет встроенных проверок ReFS не требует регулярного использования утилит проверки диска типа CHKDSK.

Совместимость со старымиAPI, поддержка многих особенностей NTFS, например, шифрование BitLocker, AccessControlLists, USNJournal, уведомления об изменениях, символьные ссылки, junctionpoint, точки монтирования, reparsepoint, идентификаторы файлов, NTFSoplock.

Многие возможности NTFS не поддерживаются в ReFS, включая именованные потоки файлов, NTFSDLT, короткие имена файлов, сжатие файлов, шифрование на уровне файлов EncryptingFileSystem, TransactionalNTFS, жёсткие ссылки, extendedattributes, и дисковые квоты. Разреженные файлы (Sparsefiles) поддерживаются в RTM.

В WindowsServer 2012 не поддерживается загрузка с ReFS. Ввиду отсутствия поддержки именованных потоков, ReFS не может быть использована для размещения экземпляров MSSQL, включая версию 2012 [10]

Логика файловой системы

Логически файловая система FreeBSD (как и любой Unix-системы) организована по древовидному принципу: в основании ее лежит корень (корневой каталог, обозначаемый

символом / и именуемый также root-каталогом; последнее не должно путать с каталогом /root, который выполняет роль домашнего для суперпользователя).

От корневого каталога, который можно уподобить скорее стволу дерева, отходят ветви - вложенные в него подкаталоги, и побеги - обычные файлы. Последних, правда, немного: в версиях FreeBSD это некий энтропийный файл (/entropy) и файл с описанием авторских прав на систему /COPYRIGHT.

Наименьшая единица, которую FreeBSD использует для обращения к файлам, это имя файла. Имена файлов чувствительны к регистру, поэтому readme.txt и README.TXT — два разных файла.

FreeBSD не использует расширение файла (.txt) для определения программа это, документ или другой тип данных. Файлы хранятся в каталогах.

Обращение к файлам происходит путем задания имени файла или каталога, дополняемого прямым слэшем /, за которым может следовать имя другого каталога. Если есть каталог foo, содержащий каталог bar, который содержит файл readme.txt, полное имя, или *путь* к файлу будет foo/bar/readme.txt.[3]

Однако Unix-системах всякий файл (в том числе и каталог) опознается системой не по ее имени, а по уникальному идентификатору его записи в таблице **inodes**. Существуют средства для того, чтобы эти файловые идентификаторы просмотреть. Одно из них - команда ls с опцией -i, которая выведет идентификаторы каждого именованного файла.[3]

Рассмотрим еще некоторые основные понятия.

Блоки – это сегменты диска, содержащие информацию. По умолчанию в системе FreeBSD блок имеет размер 16 Кбайт. Не все файлы имеют размеры, кратные 16 Кбайт, поэтому в файловой системе FFS для хранения остаточных кусков файлов используются *фрагменты*. Стандартный размер фрагмента составляет одну восьмую блока, или 2 Кбайт.[1]

Загрузочные блоки

UFS1 резервирует 8 Кб в начале раздела под загрузочные блоки. Хотя это пространство и выглядит огромным в сравнении с 1 Кб, который выделяли ФС предыдущего поколения, с другой стороны современные загрузчики требуют все больше и больше места. Поэтому разработчики решил и пересмотреть размер загрузочного кода в UFS2. Теперь boot-сектор раздела может иметь размер 0, 8, 64 (по умолчанию) и 256 Кб. [5]

Индексные дескрипторы – это специальные блоки, которые содержат базовую информацию о файлах, включая права доступа, размер и список блоков, занятых файлом.

Данные, хранящиеся в индексных дескрипторах, называются *метаданными*. Это данные о данных (metadata). Такой способ хранения данных не уникален для FFS – другие файловые системы, такие как NTFS, также используют блоки для хранения данных и индексные дескрипторы. Однако в каждой файловой системе используется свой, уникальный принцип индексации. Любая файловая система имеет определенное число индексных дескрипторов, число которых пропорционально размеру файловой системы. Современные диски могут иметь сотни тысяч индексных дескрипторов на каждом разделе, что необходимо для обеспечения возможности хранения сотен тысяч файлов. Если у вас имеется огромное число очень маленьких файлов, возможно, вам потребуется перестроить свою файловую систему, чтобы добавить в нее дополнительные индексные дескрипторы. Увидеть количество свободных индексных дескрипторов, доступных в файловой системе, можно с помощью команды du –i.[1]

Soft Updates и функция журналирования в FFS

Soft Updates – это технология организации и упорядочения операций записи на диск, при использовании которой метаданные файловой системы на диске остаются непротиворечивыми, производительность близка к производительности при асинхронном монтировании, а надежность соответствует надежности синхронного монтирования. Это не означает, что на диск будут записаны все данные – неполадки с электропитанием по-прежнему могут повредить данные. Однако Soft Updates предотвращают значительную часть трудностей. В случае отказа операционной системы, функция soft updates может продолжить работу со своей файловой системой и выполнить ее проверку.

Начиная с версии FreeBSD 7.0, файловая система FFS обзавелась поддержкой *журналирования*. Журналируемая файловая система записывает все изменения, производимые в файловой системе, на отдельный раздел диска, поэтому в случае неожиданного завершения работы
системы измененные данные могут быть восстановлены автоматически во время загрузки. Благодаря этому отпадает необходимость запускать утилиту fsck(8). Любая журналируемая файловая система использует порядка 1 Гбайт для хранения журнала, что делает ее слишком расточительной для применения на небольших файловых системах. Однако принцип ведения журнала в системе FreeBSD отличается от аналогичных принципов, принятых в других журналируемых файловых системах, тем, что ведение журнала поддерживается на уровне диска, ниже уровня самой файловой системы. Такой способ ведения журнала является новинкой в FreeBSD и по-прежнему находится на экспериментальной стадии.[1]

Система безопасности ОС Windows.

Система Windows Vista включает в себя несколько новых и усовершенствованных технологий, которые обеспечивают повышенный уровень защиты от вредоносных программ:

- Контроль учетных записей (UAC);
- Защитник Windows;
- Брандмауэр Windows;
- Центр обеспечения безопасности Windows;
- Средство удаления вредоносных программ;
- Политики ограниченного использования программ.

Переопределение пользовательских режимов. Начиная с выпуска системы Windows 2000, способность вызывать приложения в качестве администратора и работать с ними в качестве стандартного пользователя была встроена в архитектуру Windows; при этом администраторами использовались такие функции как «Запуск от имени». Функция «Запуск от имени» предоставляет средство командной строки, которое администраторы могут использовать, чтобы запускать инструменты и программы с разрешениями, отличными от тех, которые предоставлены текущему зарегистрированному пользователю.

Контроль учетных записей. Система Windows Vista включает в себя функцию контроля учетных записей (UAC), которая предоставляет способ разделить привилегии и задачи обычного пользователя и администратора. Функция контроля учетных записей повышает уровень безопасности, улучшая работу пользователя при использовании учетной записи обычного пользователя. Теперь пользователи могут выполнять больше задач и пользоваться повышенной совместимостью приложений без необходимости входить в систему с привилегиями администратора. В системе Windows Vista обычные пользователи могут теперь выполнять множество задач, которые ранее требовали прав администратора, но не влияли на безопасность отрицательным образом. Например: изменение параметров часового пояса, подключение к защищенной беспроводной сети.

Используя контроль учетных записей пользователей, администраторы смогут запускать большинство приложений, компонентов и процессов с ограниченными привилегиями, но при этом иметь возможность предоставлять расширенные права для выполнения определенных административных задач и функций приложений. И наоборот,

когда пользователи запускают системную задачу, для которой требуются привилегии администратора, например программу установки приложения, Windows Vista уведомит об этом пользователя и будет требовать разрешения администратора. Этот тип запроса предупреждает случайные модификации пользователями своих настольных компьютеров. Это также поможет устранить опасность того, что вредоносная программа будет использовать привилегии администратора без уведомления об этом пользователя.

Защитник Windows. Защитник Windows — это программа, входящая в систему Windows Vista, также доступная для загрузки в системе Windows XP. Она позволяет защитить компьютеры от всплывающих окон и угроз безопасности, вызванных программами-шпионами и иным нежелательным программным обеспечением. Защитник Windows осуществляет наблюдение в режиме реального времени за важными контрольными точками операционной системы Windows Vista, которые являются целью подобного нежелательного программного обеспечения, например, за папкой "Автозагрузка" и записями автозагрузки в реестре.

Брандмауэр Windows. Персональный брандмауэр является важнейшей линией защиты от многих видов вредоносных программ. Брандмауэр Windows в системе Windows Vista включает фильтрацию входящего и исходящего трафика для защиты пользователей от вредоносного ПО. Брандмауэр также интегрируется с функцией осведомленности о состоянии сети системы Windows Vista, что позволяет применять специализированные правила в зависимости от местонахождения клиентского компьютера.

Кроме того, впервые для операционной системы Windows система Windows Vista интегрирует управление брандмауэром с IPsec (Internet Protocol security). В системе Windows Vista IPsec и управление брандмауэром интегрированы в одну консоль — консоль брандмауэра Windows в режиме повышенной безопасности. Эта консоль позволяет централизованно управлять фильтрацией входящего и исходящего трафика и параметрами изоляции сервера и домена IPsec с помощью пользовательского интерфейса для упрощения настройки и уменьшения количества конфликтов политик.

Центр обеспечения безопасности Windows. Центр обеспечения безопасности Windows (WSC) работает в фоновом режиме. В системе Windows Vista эта функция постоянно проверяет и отображает состояние четырех важных категорий безопасности:

- Брандмауэр;
- Функция автоматического обновления;
- Защита от вредоносных программ;

- Другие параметры безопасности.

Корпорация Майкрософт улучшила центр обеспечения безопасности Windows в системе Windows Vista, включив в него категорию "Другие параметры безопасности". Эта категория отображает состояние параметров безопасности обозревателя Internet Explorer и функции управления учетными записями пользователей. Другая новая категория в системе Windows Vista — это "Защита от вредоносных программ", которая включает наблюдение за антивирусным и антишпионским программным обеспечением. Помимо защиты, обеспечиваемой системой Windows Vista по умолчанию, центр обеспечения безопасности Windows позволяет осуществлять наблюдение за решениями различных производителей по обеспечению безопасности для брандмауэра Windows, а также антивирусным и антишпионским программным обеспечением, запущенном на компьютере, и отображать, какие из решений включены и обновлены.

Средство удаления вредоносных программ. Средство удаления вредоносных программ Microsoft Windows предназначено для удаления вредоносных программ с зараженных компьютеров. Это средство сканирует компьютер в фоновом режиме и создает отчет, если находит какое-либо заражение. Оно не устанавливается в проверяемую операционную систему.

Политики ограниченного использования программ. Политики ограниченного использования программ дают возможность администраторам определять приложения и контролировать возможность их запуска на локальных компьютерах. Эта функция способствует защите компьютеров от известных конфликтов, а также помогает обезопасить их от вредоносного программного обеспечения. Политики ограниченного использования программ полностью интегрируются со службой каталогов Active Directory и групповой политикой. Эту функцию также можно использовать и на автономных компьютерах.

Защита конфиденциальных данных.

Шифрование диска BitLocker. Шифрование диска BitLocker позволяет защитить данные на клиентском компьютере. Весь том Windows шифруется для предотвращения нарушения защиты файлов Windows и системы, а также автономного просмотра злоумышленниками информации на защищенном диске. В самом начале загрузки BitLocker проверяет целостность системы и оборудования компьютера. При обнаружении попытки несанкционированного доступа к любым системным файлам или данным загрузка компьютера прекратится. BitLocker не позволяет злоумышленникам, которые используют другую операционную систему или запускают средства для атаки, обходить защиту файлов

и системы Windows Vista или автономно просматривать файлы, хранящиеся на защищенном диске. Шифрование диска BitLocker позволяет заблокировать нормальную последовательность загрузки до ввода пользователем персонального идентификационного номера (ПИН) или вставки флэш-накопителя USB с соответствующими ключами дешифрования. Технология BitLocker доступна в выпусках Windows Vista Enterprise Edition и Ultimate Edition для клиентских компьютеров.

Шифрованная файловая система. Шифрованную файловую систему (EFS) можно использовать для шифрования файлов и папок, чтобы обеспечить защиту данных от несанкционированного доступа. Файловая система EFS интегрируется с файловой системой NTFS, а ее работа полностью прозрачна для приложений. Когда пользователь или программа пытается получить доступ к зашифрованному файлу, операционная система автоматически получает ключ для расшифровки содержимого, а затем без подтверждения выполняет шифрование и расшифровку от лица пользователя. Пользователи, имеющие разрешенные ключи, смогут получать доступ к зашифрованным файлам и работать с ними точно так же, как с любыми другими файлами, при этом другим пользователям отказывается в доступе. Сами ключи хранятся в альтернативном потоке $EFS файловой системы NTFS.

Служба управления правами. Служба управления правами (RMS) обеспечивает применение политик использования и безопасности к конфиденциальным сообщениям электронной почты, документам, веб-содержимому и другим типам данных. RMS обеспечивает защиту данных благодаря постоянному шифрованию информации. Если файл или сообщение электронной почты передается внутри предприятия или через Интернет, к ним могут получать доступ только прошедшие проверку подлинности пользователи, которым предоставлены соответствующие права.

Управление устройствами. Способность пользователей добавлять новые самонастраиваемые устройства на клиентские компьютеры, такие как флэш-накопители USB или другие съемные носители, создает для администраторов значительные проблемы, связанные с безопасностью. Такие типы устройств не только усложняют обслуживание клиентских компьютеров, когда пользователи устанавливают неподдерживаемое оборудование, но и могут ставить под угрозу безопасность данных. Злоумышленник потенциально может использовать съемные носители для хищения интеллектуальной собственности организации. Он также может использовать съемный носитель для автономной установки вредоносной программы на клиентском компьютере.

Windows Vista позволяет администраторам использовать групповую политику для управления неподдерживаемыми или запрещенными устройствами. Например, можно разрешить пользователям устанавливать целые классы устройств (таких как принтеры), но запретить все виды съемных носителей. Администратор может иметь право переопределить эти политики и установить оборудование. Тем не менее, важно понимать, что устройство на компьютере устанавливается не для отдельного пользователя. После установки устройства пользователем оно обычно доступно всем пользователям данного компьютера. Windows Vista поддерживает управление доступом к установленным устройствам на чтение и запись на уровне пользователя. Например, можно разрешить полный доступ на чтение и запись к установленным устройствам, таким как флэш-накопитель USB, одной учетной записи пользователя, но предоставить только доступ на чтение другой учетной записи пользователя на том же компьютере.

Улучшенные средства безопасности в Windows 7.

Безопасность Windows 7 основана на безопасности Windows Vista и процессах и технологиях разработки, которые сделали Windows Vista наиболее надежной версией Windows на сегодняшний день. Фундаментальные функции безопасности, такие как: защита от исправлений ядра, ограниченный режим работы служб, предотвращение выполнения данных, рандомизация загрузки адресного пространства и необходимые уровни целостности, продолжают предоставлять улучшенную защиту от вредоносных программ и атак.

Улучшенный аудит. Windows 7 предоставляет улучшенные возможности аудита. Улучшения аудита начинаются с упрощенного управления настройками аудита, а заканчиваются большей прозрачностью того, что происходит в организации. Например, Windows 7 предоставляет более информации о том, почему пользователь получил или не получил доступ к определенной информации, а также об изменениях, сделанных определенными пользователями или группами пользователей.

Оптимизированный контроль учетных записей. Windows 7 продолжает улучшение контроля учетных записей и вносит изменения для улучшения работы пользователей. Эти изменения включают в себя уменьшение количества задач и приложений операционной системы, для которых необходимы права администратора, и предоставление гибкого поведения запроса согласия для пользователей, которые продолжают работать с правами администратора. В результате стандартные пользователи могут сделать гораздо больше, чем раньше, а все остальные пользователи получат меньше запросов.

AppLocker. AppLocker предоставляет простую и мощную структуру благодаря трем типам правил: «разрешить», «запретить» и «исключение». Разрешающие правила ограничивают выполнение приложений известными и надежными приложениями и блокируют все остальное. Запрещающие правила используют другой подход и разрешают выполнение всех приложений за исключением тех, которые находятся в списке «известных ненадежных» приложений. Правила исключений исключают файлы из разрешающего/запрещающего правила, которые обычно включаются. С помощью исключений можно, например, создать правила для «разрешения выполнения всего в операционной системе Windows за исключением встроенных игр». Использование разрешающих правил с исключениями обеспечивает надежный способ создания «известного надежного списка» приложений без необходимости создания слишком большого количества правил.

BitLocker и BitLocker To Go. В Windows 7 BitLocker добавлена поддержка агента восстановления данных (DRA – Data Recovery Agent) для всех томов. Поддержка DRA позволяет обеспечивать шифрование всех томов, защищенных с помощью BitLocker, (операционная система, фиксированные тома и новые портативные тома) соответствующим DRA. DRA – это новое ключевое средство защиты, записываемое на каждый том данных, чтобы авторизованные администраторы всегда имели доступ к томам, защищенным с помощью BitLocker. BitLocker To Go расширяет поддержку BitLocker для съемных устройств хранения, включая флэш-накопители USB и портативные диски. BitLocker To Go также предоставляет администраторам возможность следить за тем, как используются съемные носители в производственной среде, и контролировать уровень защиты, необходимый для этих носителей.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДВФУ)

<ШКОЛА ЕСТЕСТВЕННЫХ НАУК>

МАТЕРИАЛЫ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

по дисциплине «Операционные системы»

Направление— 230700.62 «прикладная информатика»

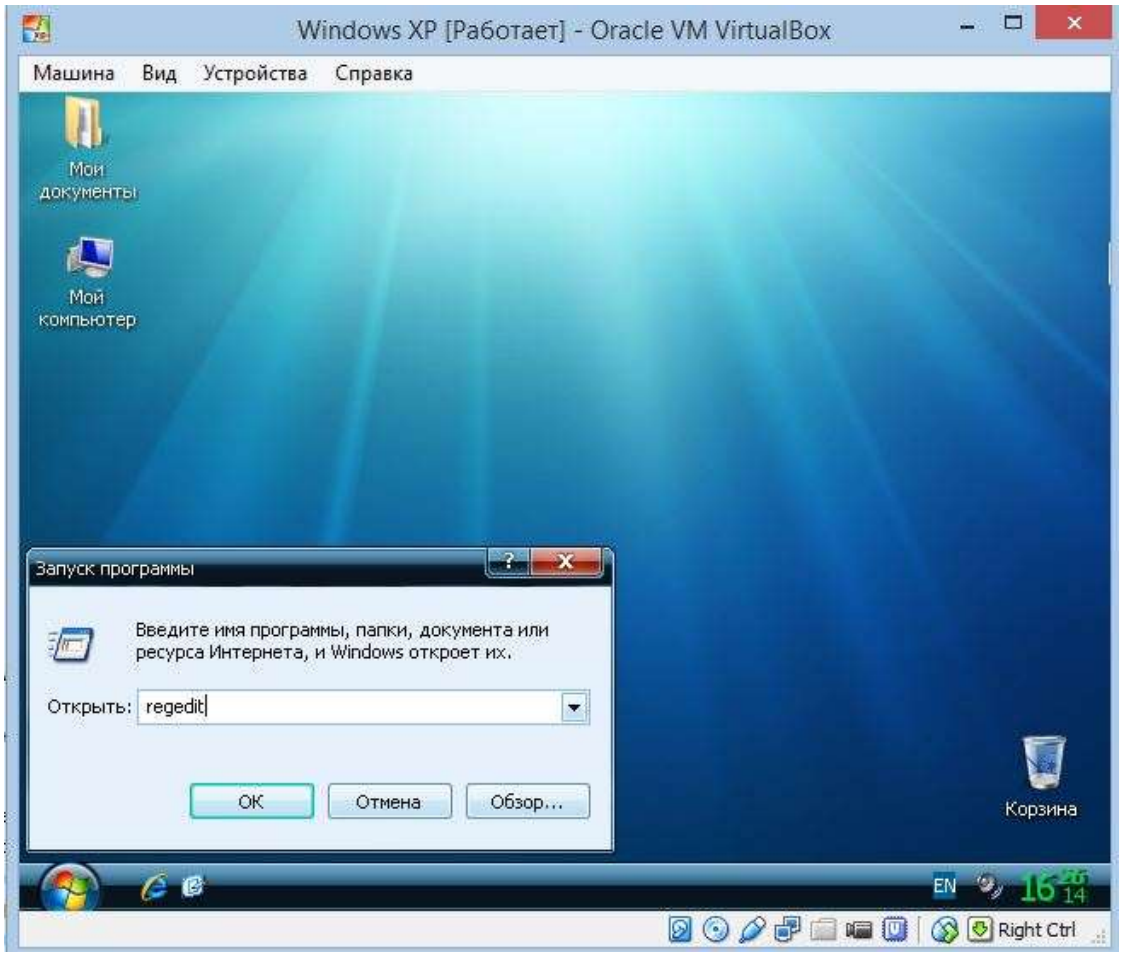
г. Владивосток

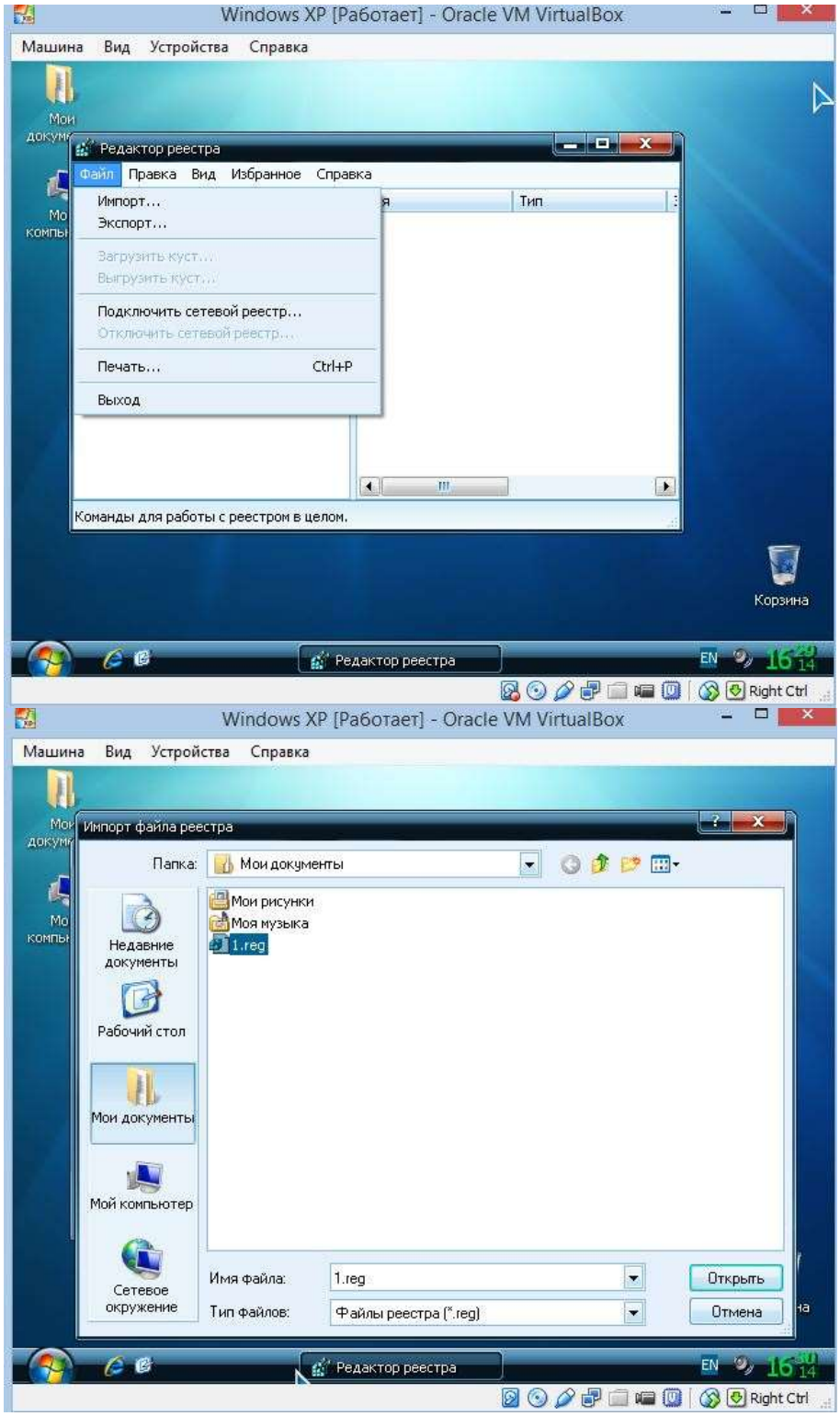
Задание 8. «Windows. Реестр»

URL: http://imcs.dvfu.ru/works/task_view?id=31708

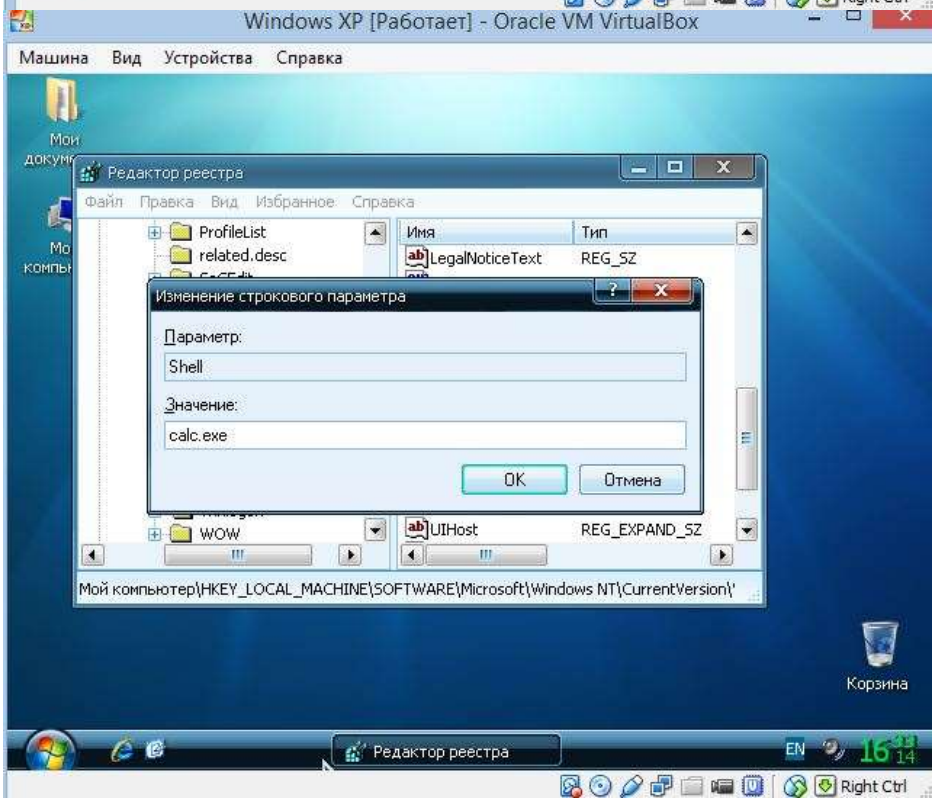
Используя инструмент regedit подключить к реестру одной windows машины реестр другой, затем внести изменения вреестр HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell. Попробовать загрузить систему с этими изменениями. Система должна загружаться.

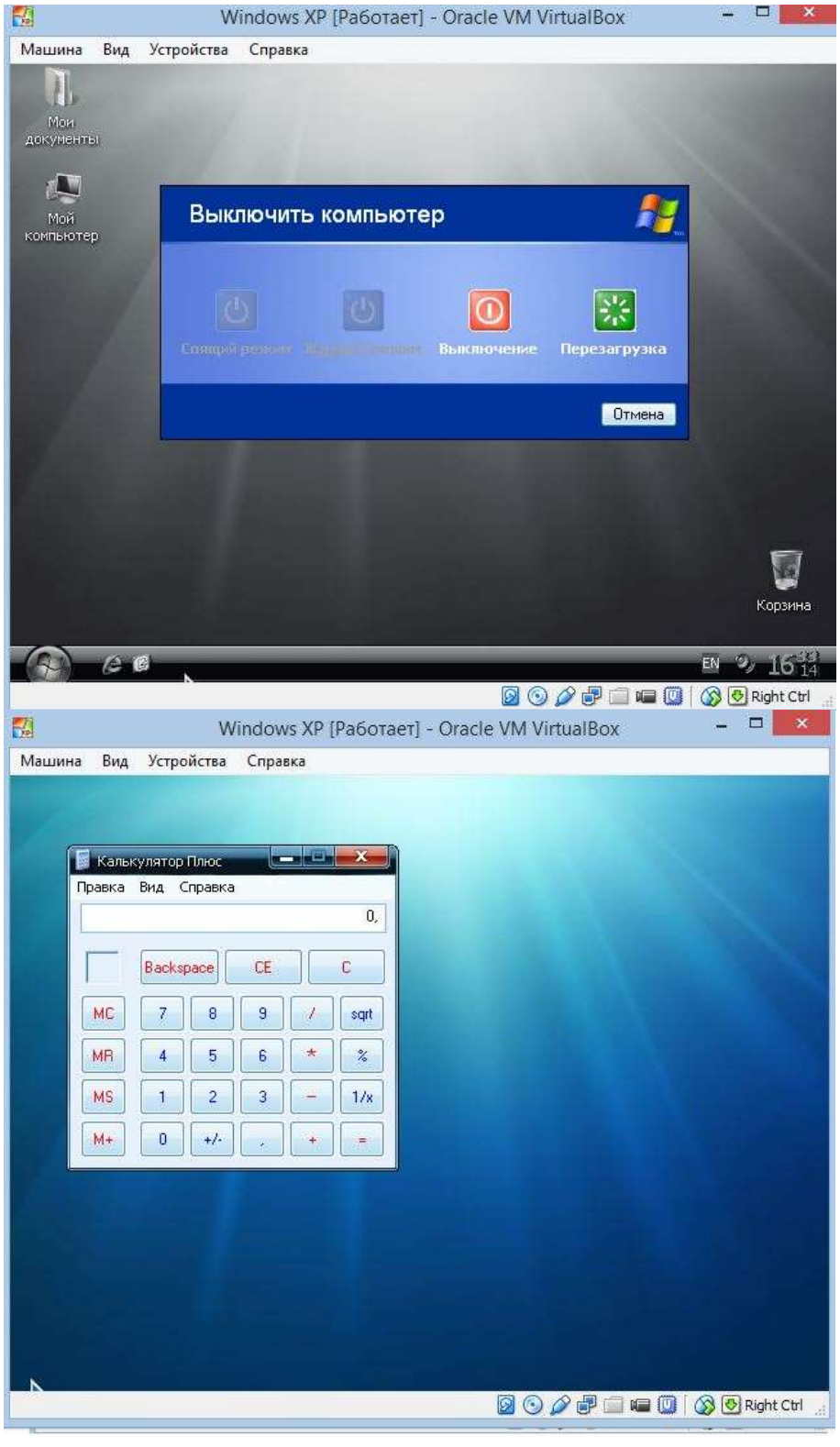
Решение




Windows XP [Работает] - Oracle VM VirtualBox
Машина Вид Устройства Справка
Редактор реестра
Файл Правка Вид Избранное Справка
Импорт...
Экспорт...
Загрузить куст...
Выгрузить куст...
Подключить сетевой реестр...
Отключить сетевой реестр...
Печать... Ctrl+P
Выход
Тип
Команды для работы с реестром в целом.
Корзина
Редактор реестра
Right Ctrl
Windows XP [Работает] - Oracle VM VirtualBox
Машина Вид Устройства Справка
Импорт файла реестра
Папка: Мои документы
Мои рисунки
Моя музыка
1.reg
Недавние документы
Рабочий стол
Мои документы
Мой компьютер
Сетевое окружение
Имя файла: 1.reg
Тип файлов: Файлы реестра (*.reg)
Открыть
Отмена
Редактор реестра
Right Ctrl


Windows XP [Работает] - Oracle VM VirtualBox
Машина Вид Устройства Справка
Редактор реестра
Файл Правка Вид Избранное Справка
ProfileList
related.desc
SeCEdit
Setup
SvcHost
SystemRestore
Terminal Server
Time Zones
Tracing
Type 1 Installer
Userinstallable.drivers
Windows
Winlogon
WOW
Имя
Тип
LegalNoticeText REG_SZ
LogonType REG_DWORD
passwordexpiryw... REG_DWORD
PowerdownAfter... REG_SZ
ReportBootOk REG_SZ
scremoveoption REG_SZ
SfcQuota REG_DWORD
Shell REG_SZ
ShowLogonOptions REG_DWORD
ShutdownWithou... REG_SZ
System REG_SZ
UIHost REG_EXPAND_SZ
Мой компьютер\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\'
Корзина
Редактор реестра
EN
Right Ctrl


Windows XP [Работает] - Oracle VM VirtualBox
Машина Вид Устройства Справка
Редактор реестра
Файл Правка Вид Избранное Справка
ProfileList
related.desc
Имя
Тип
LegalNoticeText REG_SZ
Изменение строкового параметра
Параметр:
Shell
Значение:
calc.exe
OK
Отмена
WOW
UIHost REG_EXPAND_SZ
Мой компьютер\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\'
Корзина
Редактор реестра
EN
Right Ctrl

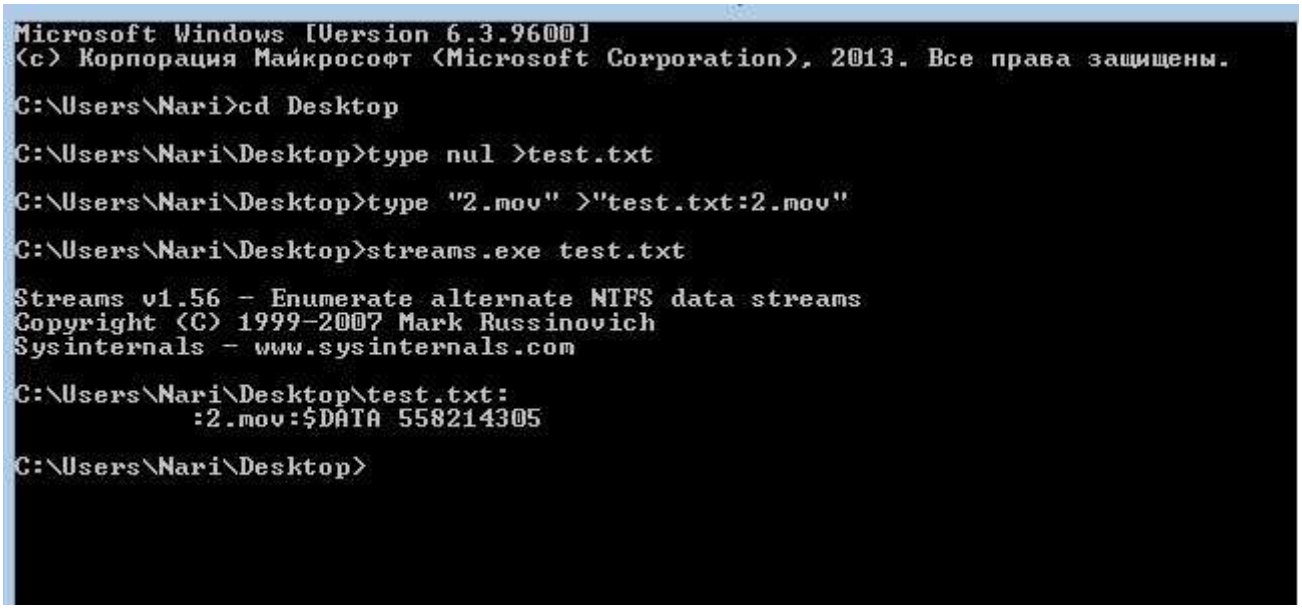


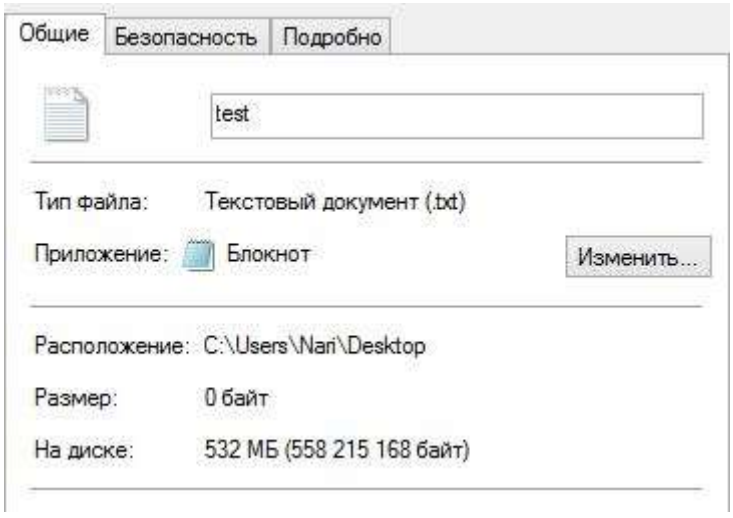
Задание 10. «Windows. NTFS. Файловые потоки»

URL: http://imcs.dvfu.ru/works/task_view?id=31710

1) Создать в Windows пустой файл, при удалении которого будет освобождено более 100 Мб места на диске. 2) Вывести список всех файловых потоков, прикрепленных к заданному файлу.

Решение





Поток данных (англ. stream) в программировании — абстракция, используемая для чтения или записи файлов, сокетов и т. п. в единой манере.

Потоки являются удобным унифицированным программным интерфейсом для чтения или записи файлов (в том числе специальных и, в частности, связанных с устройствами), сокетов и передачи данных между процессами.

Поддержка потоков включена в большинство языков программирования и едва ли не во все

современные (на 2008 год) операционные системы. При запуске процесса ему предоставляются предопределённые стандартные потоки. Возможность перенаправления потоков позволяет связывать различные программы. Команда **type** осуществляет вывод содержимого файла.

Задание 11. «Windows. NTFS. Прореженные файлы»

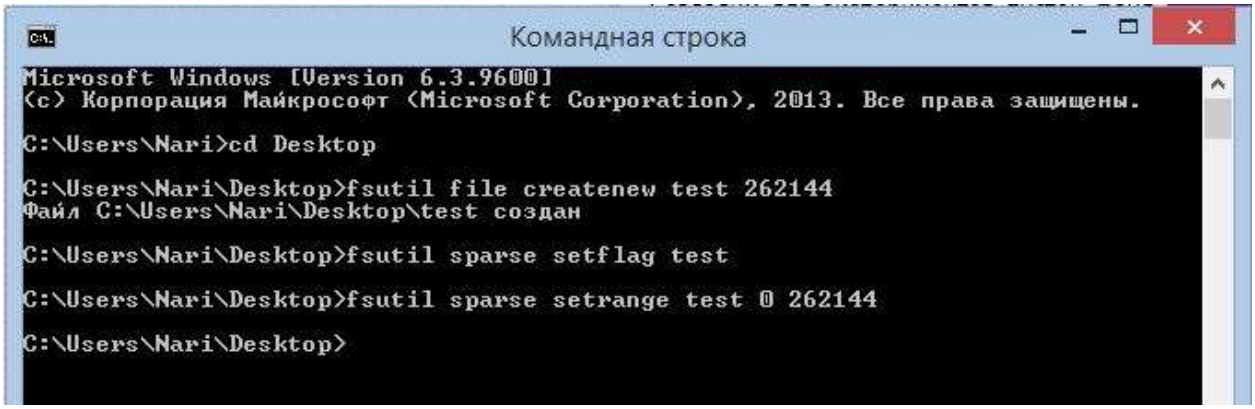
URL: http://imcs.dvfu.ru/works/task_view?id=31709

Создать в файловой системе NTFS файл, у которого параметр "Размер" будет больше размера диска, на котором находится файл. (Не путать с параметром "Размер на диске")

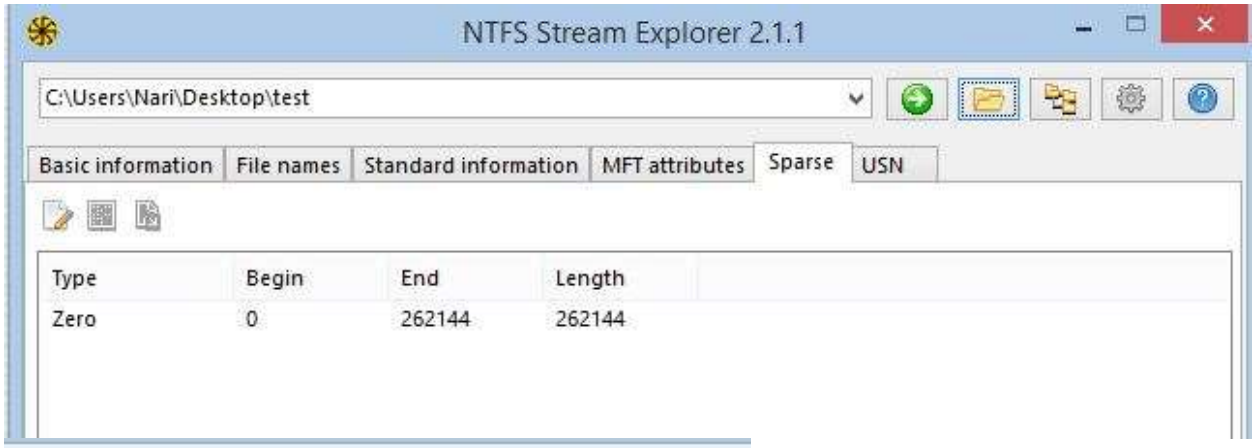
Решение

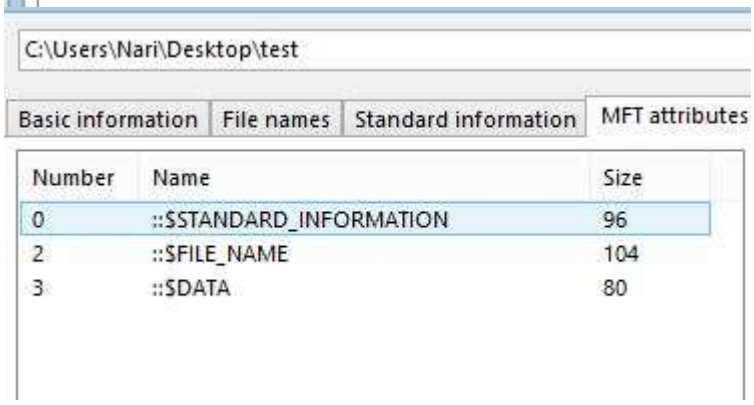
В NTFS есть поддержка **разреженных файлов** (sparsefiles). Это такие файлы, которые занимают меньше дискового пространства, чем их собственный размер. Данная технология не имеет отношения к встроенной в NTFS поддержке компрессии файлов, так как экономия места на диске в sparse-файлах основана на другом принципе. Никакого сжатия данных не осуществляется. Вместо этого, в файле высвобождаются области, занятые одними лишь нулями (0x00). Приложение, читающее разреженный файл, дойдя до области с нулями, прочитает нули, но реального чтения с диска не произойдёт.

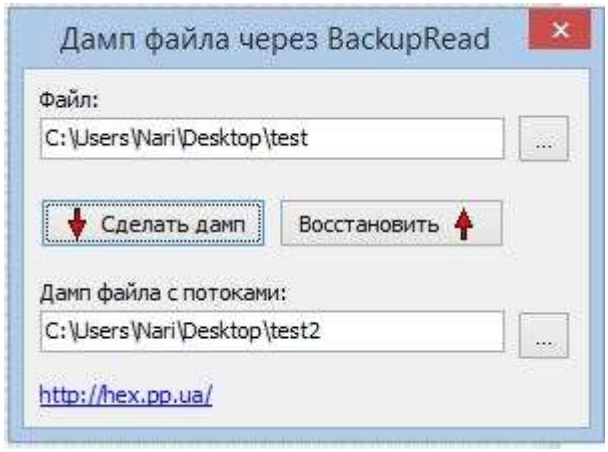
Таким образом, можно создавать файлы гигантского размера, состоящие из нулей, но на диске они могут занимать всего лишь несколько килобайт. Реальное дисковое пространство выделяется тогда, когда вместо 0x00 записываются какие-то другие данные. Разреженность поможет сэкономить дисковое пространство только в таких файлах, в которых есть действительно большие пустые области.

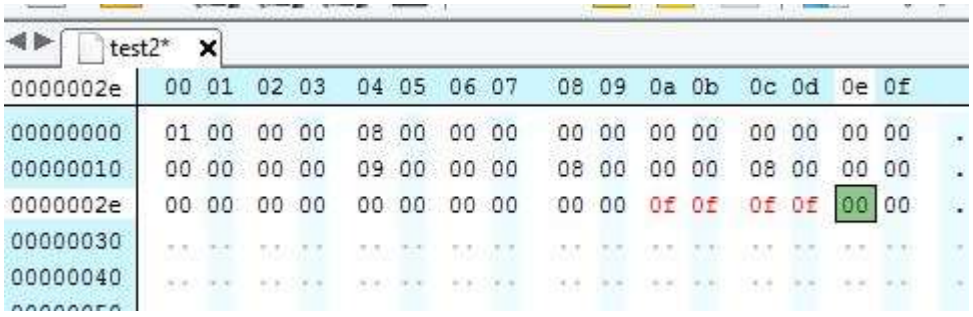


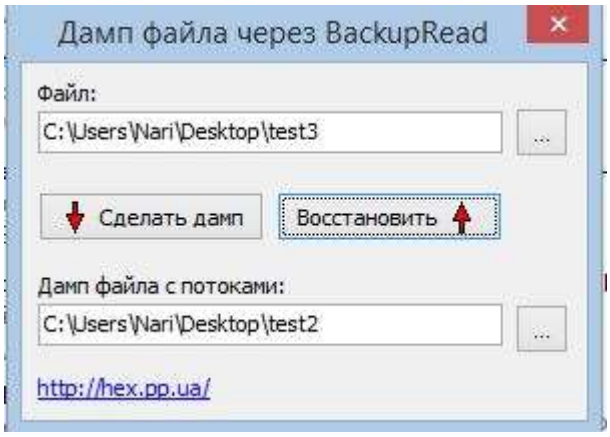
Выше произведено создание файла объемом 256 КБ, установлен флаг «разреженности», указанный диапазон был заполнен нулями.











Задание 12. «Windows. PowerShell информация о процессах»

URL: http://imcs.dvfu.ru/works/task_view?id=31840

Используя WindowsPowerShell выдать всю информацию о работающих в данный момент процессах и файлах из которых они были запущены. Формат вывода - два столбца 1) имя процесса 2) имя файла из которого он был запущен.

Решение

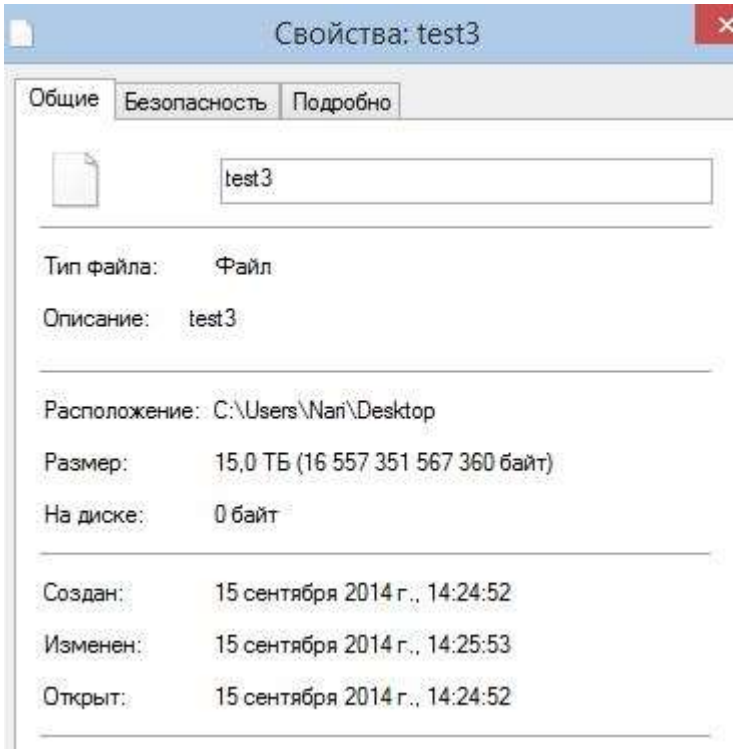
get-process | format-table Name, Path

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation), 2013. Все права защищены.

PS C:\Users\Nari> get-process | format-table Name, Path

Name                                                  Path
----                                                  ----
AppleMobileDeviceService
armsvc
Ath_CoexAgent
BtwRSupportService
chrome                                                C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
chrome                                                C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
chrome                                                C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
ChsIME                                                C:\Windows\System32\InputMethod\CHS\ChsIME.exe
conhost                                               C:\WINDOWS\system32\conhost.exe
csrss
csrss
dasHost
dllhost
dwm
ekrn
explorer                                              C:\WINDOWS\Explorer.EXE
GoogleUpdate
hkcmd                                                 C:\Windows\System32\hkcmd.exe
Idle
igfxpers                                              C:\Windows\System32\igfxpers.exe
igfxtray                                              C:\Windows\System32\igfxtray.exe
iPodService
iTunesHelper                                          C:\Program Files (x86)\iTunes\iTunesHelper.exe
LingvoCOMHelper                                       C:\Program Files (x86)\ABBYY Lingvo x5\LingvoCOMHelper.exe
lsass
mDNSResponder
NetworkLicenseServer
novapdfs
nvvsvc
nvvsvc
nvxdsync
```

Примечание: путь не указан у служб/системных процессов

Иной способ вывода:

```
C:\Users\Nari> get-process | format-table Id, Name, Path

   Id Name                                    Path
   -- ----                                    ----
 1916 AppleMobileDeviceService
 1900 armsvc
 2012 Ath_CoexAgent
  516 BtwRSupportService
 3996 chrome                                  C:\Program Files (x86)\Google\Chrome...
 4400 chrome                                  C:\Program Files (x86)\Google\Chrome...
 4564 chrome                                  C:\Program Files (x86)\Google\Chrome...
 4956 chrome                                  C:\Program Files (x86)\Google\Chrome...
 4960 chrome                                  C:\Program Files (x86)\Google\Chrome...
 1440 ChsIME                                  C:\Windows\System32\InputMethod\CHS\...
 3976 conhost                                 C:\WINDOWS\system32\conhost.exe
  504 csrss
  580 csrss
 1784 dasHost
 3828 dllhost
  884 dwm
 1704 ekrn
 2488 explorer                                C:\WINDOWS\Explorer.EXE
 2984 GoogleUpdate
 3924 hkcmd                                   C:\Windows\System32\hkcmd.exe
    0 Idle
 5240 igfxpers                                C:\Windows\System32\igfxpers.exe
 5048 igfxtray                                C:\Windows\System32\igfxtray.exe
 5700 iPodService
 5344 iTunesHelper                            C:\Program Files (x86)\iTunes\iTunes...
 4876 LingvoCOMHelper                         C:\Program Files (x86)\ABBYY Lingvo ...
  680 lsass
```



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДВФУ)

<ШКОЛА ЕСТЕСТВЕННЫХ НАУК>

Материалы для организации самостоятельной работы студентов

по дисциплине «Операционные системы»

Направление— 230700.62 «прикладная информатика»

г. Владивосток

Задания на самостоятельную работу

Самостоятельное задание 1. Поиск и сортировка сообщений.

Задание по теме «Команды оболочки Linux»

В каталоге журналируемой файловой системы ext4 в некой директории скопи(рова)лось множество файлов сообщений из разных источников. Все сообщения добавлялись по мере поступления и им давались имена в виде

"<Буквоцифры>_<цифры>.msg".

Внутри файла сообщение выглядит примерно так

*****<номер>*****
<хэш сообщения>
<непечатаемый спецсимвол с кодом 001E>DD<номер отправителя>[P|S]
<данные сообщения>

Всего в директории накопилось множество сообщений (>75000 штук).

Задание: Используя bash или csh написать скрипт, который выберет все сообщения от отправителя с заданным номером и склеит их в один файл, причем сообщения должны быть отсортированы по дате создания файлов (либо по дате их изменения, на выбор). Оценка зависит от объема скрипта

1) <=65 символов - отлично (10)

2) > 65 символов - хорошо (8)

P.S. Perl запрещен!

Варианты решения:

1) Написать скрипт с циклами, условными переходами и прочими командами длиной более 65 символов – 8 баллов

2) Написать в командной строке cat *.msg | grep <номер отправителя> 1>out.file

Самостоятельное задание 2. Загрузчик.

Задание находится на пересечении тем «Файловая система Linux», «Загрузка ОС».

Написать программу, загружающуюся с помощью загрузчика GRUB и выводящую на экран надпись "Hello World!" путем отображения на кусок видеопамяти. Задание проверяется на виртуальной машине, либо на машине без установленной на ней ОС.

Варианты решения:

Написать на языке Си с элементами ассемблера и некоторых стандартных процедур GRUB. Проще говоря, встроиться в GRUB на этапе загрузке.

Самостоятельное задание 3. RAID.

Задание по теме «RAID и загрузка»

Установленная по умолчанию, в институте Запуска Операционных Генераторов, операционная система Linux, вдруг начала давать сбои в работе. При диагностике было выяснено, что сбоит жесткий диск в виртуальной машине, на который установлена ОС. Поэтому на ученом совете института, было решено перенести информацию на программный RAID_1 (Зеркало). Для этого в институте было закуплено два одинаковых виртуальных жестких диска для виртуальной машины.

Задание: Перенести имеющуюся виртуальную машину с Ubuntu Linux (которую вы ставили в рамках подготовки к курсу ОС на программный RAID_1, с сохранением всех необходимых данных. Перенесенная система должна успешно загружаться с программного raid'a.

Варианты решения:

Честно претащить ОС Ubuntu на программный RAID 1. Согласно следующему мануалу http://help.ubuntu.ru/wiki/migrate-to-raid

Самостоятельное задание 4. Оболочка.

Задание находится на пересечении тем «Файловая система Linux», «Команды оболочки».

Реализовать программу-оболочку для Linux. Необходимые возможности:

1) создание дочерних процессов с использованием вызова имеющихся программ (5 баллов);

2) передача данных процессам через конвейер «|» (8 баллов);

3) исполнение файлов-скриптов (10 баллов)

4) реализация ожидания окончания выполнения/не ожидания, символ & в bash (символ & в конце строки, ПРИ ЭТОМ ОН НЕ ОБЯЗАН НАХОДИТСЯ ИМЕННО В КОНЦЕ СТРОКИ! дает выполнить всю предыдущую команду в виде отдельной задачи и позволяя вводить другие команды в оболочку без ожидания завершения выполнения предыдущей) (8 баллов)

5) хранение внутренних переменных среды, передача их в запущенные программы в окружение и текстовая подстановка при запуске с помощью символа «$» подробнее теорию читать в http://www.opennet.ru/docs/RUS/zlp/004.html (9 баллов)

Варианты решения:

Написать, выполнив по пунктам все перечисленное в задании на каком-либо языке программирования.

Самостоятельное задание 5. Установка программ в Linux.

Задание находится на пересечении тем «Репозитории и исходные коды», «Переменные среды Linux».

Произвести запуск программы ncl с помощью написанной ранее Оболочки.

Варианты решения:

1) Зарегистрироваться на сайте http://ncl.ucar.edu
2) Скачать исходный код
3) Откомпилировать
4) Установить программу
5) Прописать переменные среды
6) Настроить оболочку на эти переменные

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДВФУ)

<ШКОЛА ЕСТЕСТВЕННЫХ НАУК>

Контрольно-измерительные материалы

по дисциплине «Операционные системы»

Направление— 230700.62 «прикладная информатика»

г. Владивосток

Программа теоретического зачета по дисциплине «Операционные системы»

1. История развития операционных систем. Multics, Unix, System V, BSD, DOS, Windows, и др.
2. Классификация операционных систем по особенностям алгоритмов управления ресурсами, особенностям аппаратных платформ.
3. Типы архитектур ядра операционных систем. Архитектура микроядра. Архитектура монолитного ядра.
4. Процессы и нити Unix систем. Состояния процесса в Linux. Системные функции управления процессами.
5. Сигналы. Планирование процессов. Стандарты POSIX.
6. Организация памяти. Свопинг и подкачка по запросу. Алгоритмы выделения памяти. Системные вызовы управления памятью.
7. Вирутальное адресное простарнство. Отображение виртуальной памяти на реальные устройства, на файловые системы.
8. Ввод/Вывод в Unix системах. Работа с сетью. Работа с файлами. Системные вызовы Ввода/Вывода.
9. Файловые системы Unix. Виртуальная файловая система VFS. Файловые системы Ext2, Ext3, Ext4.
10. Установка прикладных программ. Репозитории. Политика распространения исходных кодов программ.
11. Оболочки Unix. Программное окружение Unix. Оболочки - C-shell, Bourne-shell, Korn-shell, bash.
12. Windows. Уровень абстрагирования оборудования HAL. Уровень ядра. Уровень пользовательских приложений.
13. Асинхронные вызовы процедур. Процессы и потоки в Windows. Приоритеты процессов. Существуют ли синхронные вызовы процедур и почему да или почему нет.
14. WinAPI
15. Подкачка и свопинг. Алгоритм замещения страниц. Кэширование.
16. Реестр Windows. Организация реестра. Кусты, узлы и из соответствие системным файлам
17. Файловая система Windows. FAT12. FAT16. FAT32. NTFS. Ввод/Вывод в Windows. WinAPI для ввода/вывода.
18. Распределенные ОС. Мультипроцессоры и мультикомпьютеры. Распределение памяти. UMA. NUMA

19. Распределенные ОС. Сетевые операционные системы. Соединение компьютеров через Internet. Облачные вычисления.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДВФУ)

<ШКОЛА ЕСТЕСТВЕННЫХ НАУК>

Список литературы

по дисциплине «Операционные системы»

Направление— 230700.62 «прикладная информатика»

г. Владивосток

Основная

1. Таненбаум Э. – Современные операционные системы. 3-е изд. – Спб.: Питер, 2010. – 1120 с.: ил. – (Серия «Классика ComputerScience»)
2. Олифер В.Г., Олифер Н.А. Сетевые операционные с истемы. СПб.: Питер, 2001. 544 с.
3. Иртегов Д.В. – Введение в операционные системы, – 2-е изд., перераб. и доп. – Спб.: БХВ-Петербург, 2008. – 1040 с.: с ил. – (Учебное пособие)
4. Назаров С.В. – Современные операционные системы: учебное пособие / С.В. Назаров, А.И. Широков. – М.: Национальный Открытый Университет «ИНТУИТ», 2012. – 376 с: ил., табл. – (Основы информационных технологий)
5. Назаров С.В. Операционные среды, системы и оболочки. Основыструктурной и функциональной организации: Учеб.пособие. – М.:КУДИЦ-ПРЕСС, 2007.- 504 с.
6. Гордеев А.В. – Операционные системы: учебник для вузов, – 2-е изд., перераб. и доп., – Спб.: Питер, 2004. – 416 с.

Дополнительная

1. Лукас М. FreeBSD. Подробное руководство, 2_е издание. – Пер. с англ. – СПб.: Символ_Плюс, 2009. – 864 с.[Электронный ресурс] //. –Режим доступа: http://sferon.dlinkddns.com/Pub/%D0%9B%D0%B8%D1%82%D0%B5%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%B0/FreeBSDLucas2009.pdf
2. Tom Rhodes, Глава 19. Поддержка файловых систем http://www.freebsd.org/doc/ru/books/handbook/filesystems.html
3. Федорчук А. Файловая система FreeBSD: иерархия и монтирование.[Электронный ресурс] : http://citkit.ru/articles/161

4. Глава 4. Основы UNIX.Организация дисков. Гл 4. http://www.freebsd.org/doc/ru/books/handbook/disk-organization.html
5. Пешеходов А. П. Файловая система FreeBSD http://www.opennet.ru/soft/fs/freebsd_fs.pdf

Интернет ресурсы

1. Электронный ресурс: http://support.microsoft.com/kb/100108/ru
2. Электронный ресурс: http://windows.microsoft.com/ru-ru/windows7/comparing-ntfs-and-fat32-file-systems
3. Электронный ресурс: http://www.intuit.ru/studies/educational_groups/942/courses/217/lecture/5607?page=2
4. Электронный ресурс: http://www.uhlib.ru/kompyutery_i_internet/4_vnutrennee_ustroistvo_windows_gl_12_14/p1.php
5. Электронный ресурс: http://en.wikipedia.org/wiki/ReFS
6. Электронный ресурс: http://en.wikipedia.org/wiki/NTFS
7. Электронный ресурс: The Linux file systems http://swift.siphos.be/linux_sea/linuxfs.html
8. Электронный ресурс: http://rus-linux.net/kos.php?name=/papers/boot/index.html В.А.Костромин, Исследуем процесс загрузки Linux, 2007 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Дальневосточный федеральный университет»
(ДВФУ)

<ШКОЛА ЕСТЕСТВЕННЫХ НАУК>

Глоссарий
по дисциплине «Операционные системы»

Направление— 230700.62 «прикладная информатика»

г. Владивосток

ОС – Операционная система

Unix – ОС, образовавшаяся как урезанная MULTICS для персональных компьютеров

Linux – ОС, образовавшаяся на базе Unix. Первоначально была написана как курсовой проект студентом Линусом Торвальдсом под руководством Эндрю Таненбаума (реализовал операционную систему MINIX и хороший курс в университете по ОС)

DOS – Disc Operation System – операционная система, разработанная Биллом Гейтсом (перекуплена и доработана из QDOS, автора QDOS история не запомнила) для первых компьютеров IBM серии PC

Windows – Операционная система, разработанная компанией Microsoft изначально как оболочка для DOS, но выросшая в отдельную ОС.

Оболочка – программа, интегрированная в ОС и предоставляющая дополнительные возможности по управлению данными и программами в самой ОС. Иногда для настройки ОС. Обычно применяется как более удобный пользовательский/административный интерфейс

Загрузчик – программа выполняющая начальную инициализацию оборудования и запускающая ОС

GRUB – GRand Unified Bootloader, стандартный загрузчик операционных систем серии Linux

Менеджер памяти – программа, определяющая принципы использования памяти другими программами в ОС

Свопинг – Процесс обмена данными между памятью и подкачкой на жестком диске или другом хранилище постоянной памяти.

Подкачка – Место на физическом носителе для временного хранения там оперативной памяти или ее части.

Виртуальная память – система организации физической памяти

ОС реального времени – группа ОС, отличающаяся возможностями обеспечения полной синхронности в работе множетсва систем. Все задачи в ней выпоняются и рассчитываются к выполнению со строго нормированным временем работы, при этом могут быть строго синхронизированны. Применяются в условиях необходимости точного контроля времени при выполнении операции (начинка самолетов, автомобилей, поездов, прочей техники которая взаимодействует напрямую с реальным миром)

Распределение задач – процесс определения порядка выполнения группы задач в ОС. Также может быть пременено физичское разделение по разным процессорам (исполнителям)

Ядро – часть ОС, выполняющая функции соединения прикладных программ и аппаратуры, также выступает в качестве контроллера критических процессов, предоставляет API для работы с устройствами.

API – Applied Programming Interface, прикладной интерфейс программирования для доступа к ресурсам, которые обычно контролируются ОС

POSIX – стандарт API, также известный как ISO/IEC 9945 или IEEE 1003

Файловые системы – система организации памяти на постоянном носителе в виде кусков, именуемых файлами.