Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

В.Г. КОБЫЛЯНСКИЙ

ОПЕРАЦИОННЫЕ СИСТЕМЫ, СРЕДЫ И ОБОЛОЧКИ

Часть 1

Методические указания к выполнению лабораторных работ для студентов 2 курса направления 01.03.02 «Прикладная математика и информатика» дневного отделения факультета прикладной математики и информатики

Новосибирск

2021

СОДЕРЖАНИЕ

Организация лабораторного практикума ... 3
Лабораторная работа № 1. Командный интерфейс Linux ... 5
Лабораторная работа № 2. Текстовый редактор Vim ... 24
Лабораторная работа № 3. Командные сценарии Linux ... 32
Лабораторная работа № 4. Файловые системы ОС Linux ... 43
Лабораторная работа № 5. Файловые системы ОС Windows ... 62
Лабораторная работа № 6. Инструментальные средства разработки программ ... 85
Лабораторная работа № 7. Технология виртуализации ... 107
Список источников ... 117

Организация лабораторного практикума

Целью лабораторного практикума по курсу «Операционные системы» является получение практических навыков работы с операционными системами (ОС) Linux и Windows и углубленное изучение вопросов, связанных с командным режимом, файловыми системами и прохождением программ в среде ОС.

Основная часть практикума выполняется в среде ОС Linux, установленной на сервере ФПМИ (*students.ami.nstu.ru* или *fpm2.ami.nstu.ru*). Доступ к серверу с компьютеров, установленных в компьютерных классах факультета, проводится с помощью клиентской Windows-программы **putty**, эмулирующей удаленный терминал Linux. Задачей этой программы является отправка вводимых с клавиатуры символов серверу и прием полученной от сервера информации.

На сайте разработчика https://putty.org.ru/download.html присутствует два варианта дистрибутива – инсталлируемый в систему и портируемый, для которого установка не требуется. Рекомендуется использовать второй вариант. Для подключения к серверу в поле «Host Name» необходимо указать имя сервера, а для корректной работы с символами кириллицы в поле «Remote Caracter Set» меню Translation указать кодировку UTF-8. Соединение с сервером желательно проводить по защищенному протоколу ssh.

Несмотря на то, что Linux дает возможность смены пароля, изменять пароли, выданные администратором системы, студентам не рекомендуется.

Для выполнения лабораторных работ студенты могут объединяться в бригады составом не более двух человек. Каждая бригада имеет свой логин и пароль для входа в систему, а также свой домашний каталог, имя которого совпадает с логином.

Порядок выполнения каждой лабораторной работы:

- изучение теоретического материала;
- выполнение практического задания;
- оформление отчета;
- получение допуска к защите в форме устной беседы с преподавателем (допуск подтверждается подписью преподавателя на отчете);
- защита работы в форме теста.

Допускается оформление одного отчета на бригаду в случае одновременного получения допуска к защите всех членов бригады, иначе требуется отчет на каждого

члена бригады. Подписанный отчет сдается преподавателю непосредственно перед защитой, сданные отчеты не возвращаются.

Отчет должен содержать цель работы, описание хода выполнения работы и выводы. В разделе описания необходимо по каждому пункту выполнения привести задание, результаты выполнения задания и анализ этих результатов. Если, например, при выполнении какой-либо команды выводятся сообщения об ошибках, то необходимо не только привести скриншот результата, но и пояснить причину возникновения ошибки.

Для того, чтобы начать работу ОС Linux, необходимо:

1. войти в систему, установленную на вашем рабочем компьютере в компьютерном классе (Windows), используя бригадный логин и пароль;

2. запустить клиента putty, в поле HostName ввести имя сервера и нажать кнопку Open;

3. после установки соединения и появления приглашения **login:** необходимо набрать имя пользователя (например, pm8101) и нажать ENTER;

4. после появления приглашения на ввод пароля набрать пароль и нажать клавишу ENTER. Обратите внимание: при вводе пароля курсор на экране не перемещается;

5. если пароль введен корректно, то на экране появляется системное приглашение, например [pm8101@students ~]$, и Linux готов принимать команды. При некорректном вводе пароля более трех раз Linux блокирует подключение пользователя на период времени, заданный администратором системы.

Завершение сеанса работы с Linux проводится с помощью команд **logout** или **exit**

Лабораторная работа № 1. Командный интерфейс Linux

1. Цель работы

Основным интерфейсом для любой операционной системы (ОС) является командный интерфейс, представленный набором команд. Количество команд ОС может быть достаточно большим, каждая команда реализует одно действие над заданным ресурсом, а основным устройством управления при этом является клавиатура. Графический интерфейс является надстройкой над командным интерфейсом, т.е. любое действие, заданное мышью, выполняется с помощью соответствующей команды ОС.

Командный интерфейс требует от пользователя более глубоких знаний устройства компьютера и ОС, поэтому он прежде всего предназначен для применения IT-специалистами. Целью лабораторной работы является приобретение практических навыков работы с интерфейсом командной строки ОС Linux.

2. Методические указания

Диалог пользователя с ОС осуществляется в форме команд. Операционная система готова к диалогу, если на экране имеется системное приглашение в виде строки, содержащей логин пользователя, имя компьютера, имя текущего каталога и символы $ или # для обычного пользователя или для администратора соответственно.

Команда набирается строчными латинскими буквами и завершается нажатием клавиши <ENTER>. Формат командной строки следующий:

имя_команды [ключи] [аргументы]

Здесь *имя_команды* указывает действие; *аргументы* - имена объектов, над которыми выполняется действие, а *ключи (опции)* – уточняют действие команды. Ключи и аргументы не являются обязательными для всех команд, т.е. возможны команды, не имеющие параметров (например, **clear**). Все элементы командной строки разделяются пробелами, ключи начинаются с символа ‘-‘ (дефис).

Команды анализируются и исполняются командным интерпретатором Shell (/bin/sh) и бывают двух типов: внутренние и внешние. Внутренние команды выполняются непосредственно командным интерпретатором, а внешние команды реализуются программами, поставляемыми вместе с ОС в виде отдельных файлов. Для

определения типа команд можно использовать команду **type.**

Командный интерпретатор распознает элементы командной строки, выделяя последовательности символов между пробелами, и пытается найти функцию, имя которой совпадает с именем команды, в своей внутренней таблице. Если такая функция там зарегистрирована, то она исполняется с аргументами и ключами, выделенными из командной строки, а такая команда называется внутренней. Если во внутренней таблице имя команды отсутствует, то интерпретатор будет проводить поиск одноименного исполняемого файла в каталогах файловой системы, имена которых указаны в глобальной переменной PATH. В случае удачи соответствующая программа будет запущена, иначе на экране появится сообщение *command not found…*

После запуска команды интерпретатор ждет завершения соответствующей программы и после этого выводит на экран системную подсказку, сообщая пользователю о готовности к приему новой команды. Возможен запуск программ в фоновом режиме, когда интерпретатор возвращает управлению пользователю без ожидания их завершения, например при выполнении длительных по времени действий. Для запуска в фоновом режиме после команды необходимо указать символ '&'

Для ОС семейства UNIX разработано несколько различных командных интерпретаторов (/bin/sh, /bin/csh, /bin/ksh, /bin/bash и др.), которые могут незначительно отличаться друг от друга набором поддерживаемых команд.

В одной командной строке можно записывать несколько команд, при этом порядок их исполнения будет зависеть от используемых символов – разделителей команд (таблица 1).

Таблица 1

Разделитель команд	**Порядок исполнения**	**Пример**
;	Первая команда, затем вторая команда	cd /home; ls /home выполняется переход в каталог cd /home и затем выводится содержимое этого каталога
&&	Вторая команда будет выполнена только при успешном завершении первой команды	cd /home && ls /home содержимое каталога выводится только при успешном входе в него (в случае отсутствия прав доступа команда ls не выполняется
||	Вторая команда будет выполнена только в случае невыполнения первой команды.	cd /home || echo error в случае отсутствия прав доступа к каталогу на экран выводится сообщение об ошибке
|	Выполняется первая команда и ее выходные данные подаются на вход второй команды	ls /home | wc –l содержимое каталога /home подается на вход команды wc, которая выводит количество файлов в этом каталоге

Если в командной строке необходимо использовать символы, которые командный интерпретатор понимает как специальные, то перед таким символом необходимо поставить символ экранирования ‘\’. Например, в Вашем каталоге есть файл с именем “tes*t” и Вы должны вывести его содержимое на экран. По команде ***cat tes*t*** на экран буде выведено содержимое всех файлов, имена которых начинаются на ‘tes’ и заканчиваются символом ‘t’, т.к. ‘*’ является метасимволом и заменяет любое число любых символов. Чтобы вывести содержимое заданного файла, необходимо применить экранирование спецсимвола: ***cat tes*t***

2.1 Общие сведения о файловой системе Linux

Основным отличием ОС Linux от Windows с точки зрения пользователя является организация файловой системы. В Linux файловая система представляет собой единую иерархическую структуру, отображаемую в виде дерева, корню которого соответствует основной (корневой) каталог, а листьям – файлы. Корневой каталог имеет имя «/» и содержит системные файлы и подкаталоги.

Под файлом в Linux понимается не только поименованная совокупность информации на ВЗУ, но и любое устройство, которое может хранить, поставлять или потреблять информацию. При этом устройство подключается (монтируется) к существующему дереву файловой системы в указанной пользователем точке с помощью команды **mount,** после чего пользователь может обращаться к любым доступным файлам, при этом в имени никак не отражается имя устройства, на котором файл находится или создается.

Linux поддерживает следующие типы файлов: *обычные, каталоги, каналы, специальные (блочные или символьные), ссылки.* Понятие обычного файла в Linux и Windows близки, но в Linux отсутствует деление на текстовые и бинарные файлы, т.е. все обычные файлы представляются в виде последовательность байтов без какой-либо дополнительной структуры.

Любой файл в файловой системе Linux имеет так называемый индексный дескриптор, который и хранит всю необходимую информацию о файле. Для каждого файла номер индексного дескриптора уникальный, а имя файла является всего лишь ссылкой на данный дескриптор. Каталоги в Linux устанавливают соответствие между именем и номером дескриптора файла. Возможна ситуация, когда несколько

элементов каталога ссылаются на один номер дескриптора, в этом случае мы будем иметь несколько ссылок на один и тот же файл, т.е. в Linux имеется возможность присвоить различные имена одному файлу. Ссылки могут быть жесткими и мягкими (символьными). Жесткая ссылка ссылается на дескриптор оригинального файла, а мягкая – на его имя.

Такое отклонение от древовидной структуры весьма полезно, т.к. позволяет любому файлу дать любое имя, не дублируя информацию. При удалении файла поле номера дескриптора в элементе каталога обнуляется, поэтому при удалении всех ссылок на файл его восстановление невозможно.

Канал - это временный файл ограниченного размера, информация из которого после чтения исчезает. Каналы используются для организации обмена данными между различными процессами.

Специальные файлы соответствуют устройствам (блочные - дискам, символьные - всем прочим). Введение этого понятия позволяет единообразно организовать обмен информацией с любым источником или приемником информации. Например, для чтения флэш-накопителя достаточно обычным образом открыть файл /dev/usb, позиционироваться в нем и прочесть нужное число блоков. Т.е. прикладные программы одинаково обмениваются информацией с обычным файлом, каналом или устройством.

Полное имя показывает местоположение файла или каталога в файловой системе. Существует два типа имени: *абсолютное* и *относительное*. Абсолютное всегда начинается с символа «/», обозначающего корневой каталог. Кроме того, этот символ используется и в абсолютном, и в относительном имени в качестве разделителя имен каталогов и файлов (например, /home/NSTU/pm6101). Абсолютное имя показывает положение файла по отношению ко всей иерархической структуре, а относительное имя - по отношению к текущему или домашнему каталогу.

Имена каталогов и файлов в Linux являются регистрозависимыми, т.е. имена newfile и NEWFILE обозначают разные файлы. Если имя файла начинается с символа «.» (точка), такой файл называется скрытым и его характеристики не будут выводиться при просмотре содержимого каталогов.

Возможны три категории пользователей файлов и каталогов – владелец (u), группа пользователей, в которую входит владелец (g) и остальные пользователи (o), для каждой из этих категорий можно устанавливать собственные права доступа, которые определяют, кто и что может делать с содержимым файла. Права могут быть стандартными и специальными, к первой группе относятся права на чтение (r) , запись (w) и выполнение (x), а специальными правами являются права на выполнение от имени владельца (s) и запрет удаления чужих файлов (t).

Права доступа хранятся в дескрипторе файла и отображаются при просмотре каталогов в виде строки из 10 символов. Первый символ строки обозначает тип файла (дефис – это обычный файл, d – каталог, l –мягкая ссылка, b – блочный специальный файл, c – символьный специальный файл). Далее в строке идут три группы, состоящие из трех символов каждая: первая группа определяет права владельца файла, вторая – права группы пользователей, третья - права остальных пользователей. В каждой из этих групп первый символ определяет право доступа для чтения, второй – для записи, третий – для выполнения файла. Если в какой – либо позиции в этих группах выводится символ «-», то соответствующее право доступа отключено. В таблице 2 представлена система определения прав доступа.

Например,

а) **-rw-r--r--** владелец имеет право читать и изменять файл, члены группы и остальные пользователи могут только читать файл;

б) **drwxr-x--x** владелец может просматривать, изменять и входить в каталог, члены группы могут входить и просматривать его, все остальные – только входить

Таблица 2

Право	Обозначение	Файл	Каталог
Чтение	r	Файл можно посмотреть и скопировать	Можно посмотреть список входящих файлов
Запись	w	Файл можно изменить и переименовать	Можно создавать и удалять файлы
Выполнение	x	Файл можно запустить на выполнение (скрипты и программы)	Можно входить, делать текущим
Выполнение от имени владельца	s	Файл можно запустить на выполнение с правами его владельца	Все объекты, создаваемые внутри каталога, наследуют имя его группы

Запрет удаления чужих файлов	t	Не применяется	Можно удалять только собственные файлы

2.2 Основные команды ОС Linux.

Обратите внимание:

– при указании имен файлов в командах можно использовать метасимволы (‘*’ - заменяет любое количество символов, в том числе ни одного, ‘?’ - заменяет любой одиночный символ, [список символов] - заменяет одиночный символ из указанного списка или диапазона символов);

– все ключи в командной строке Linux являются регистрозависимыми;

2.2.1 Команды для работы с каталогами

2.2.1.1 Команда: **mkdir**

Назначение: создать каталог

Формат: mkdir имя_каталога1 [имя_каталога2...]

Пример: mkdir abc mykat

Комментарий: для того, чтобы создать каталог, должны быть правильно установлены права доступа.

2.2.1.2 Команда **ls.**

Назначение: просмотр каталога.

Формат: ls [-опции] [путь]

Пример**:** ls abc

Комментарий: Чтобы увидеть имена скрытых файлов, используйте опцию **a**. Для получения информации о типах файлов (каталог, исполняемый файл, ссылка), используйте опцию **F**. При использовании этой опции в поле имени выводится символ, который определяет тип файла (табл. 3).

Для получения подробной информации о файлах и каталогах используйте опцию ***l*.** При этом о каждом файле и каталоге вы получите следующую информацию: тип файла, права доступа, число ссылок, имя владельца, имя группы, размер, дата последнего изменения, имя файла или каталога.

Таблица 3

Тип файла	Каталог	Исполняемый файл	Мягкая ссылка	Обычный файл
Символ	/	*	@	отсутствует

2.2.1.3 Команда **cd.**

Назначение: переход в указанный каталог.

Формат: cd [имя_каталога]

Пример: cd /u/home/bpi

Комментарий: для перехода в домашний каталог используйте команду cd без параметров. Для перемещения по файловой системе можно использовать сокращенные имена каталогов, приведенные в табл. 4.

Таблица 4

Тип каталога	Домашний	Текущий	Родительский
Сокращенное имя	~	.	..

2.2.1.4 Команда **pwd**.

Назначение: вывод абсолютного имени текущего каталога.

Пример**:** pwd

2.2.1.5 Команда **rmdir**.

Назначение: удаление каталогов.

Формат: rmdir [-опции] имя_каталога

Пример: rmdir kat1 kat 2

Комментарий: для удаления каталога, содержащего файлы, используйте опцию -r, без указания этой опции команда не будет выполняться. Для этой цели можно также применить команду rm –rf имя_каталога.

2.2.2 Команды для работы с файлами

2.2.2.1 Команда **cat**.

Назначение: просмотр текстовых файлов.

Формат: cat имя_файла

Пример: cat file1

Комментарий: для просмотра больших файлов используйте команду **more**, так как она позволяет осуществлять постраничный просмотр файлов.

2.2.2.2 Команда **more**.

Назначение: постраничный просмотр текстовых файлов.

Формат: more имя_файла

Пример: more /etc/passwd

Комментарий: для управления просмотром используются следующие управляющие клавиши: пробел – переход на следующую страницу, b – переход на предыдущую страницу, q – выход.

2.2.2.3 Команда **head**.

Назначение: просмотр указанного числа начальных строк файла.

Формат: head [-n] имя_файла.

Параметр: n - количество выводимых строк (по умолчанию выводится 10 строк)

Пример: head /etc/passwd

2.2.2.4 Команда **tail**.

Назначение: просмотр указанного числа конечных строк файла .

Формат: tail [-n] имя_файла

Параметр: n - количество выводимых строк (по умолчанию выводится 10 строк)

Пример: tail /etc/passwd

2.2.2.5 Команда **cp**.

Назначение: копирование файлов и каталогов.

Формат: cp [-опции] исходный_файл целевой_файл

Примеры:

а) cp abc1 ./tmp/November – копирование файла;

б) cp –a kat_1 kat_2 – копирование каталога kat_1 в каталог kat_2.

Комментарий: команда **cp** с опцией **r** позволяет копировать каталоги вместе с входящими в них файлами и каталогами.

2.2.2.6 Команда **mv**.

Назначение: перемещение и переименование файлов и каталогов.

Формат: mv [-опции] старое_имя новое_имя

Пример: mv file1 file2

2.2.2.7 Команда **rm**.

Назначение: удаление файлов.

Формат: rm [-опции] имя_файла

Пример: rm file1 file2

Комментарий: если Вы хотите, чтобы команда запрашивала подтверждение на удаление файла, то используйте опцию i.

2.2.2.8 Команда **touch**.

Назначение: создание пустого файла.

Формат: touch [-опции] имя_файла

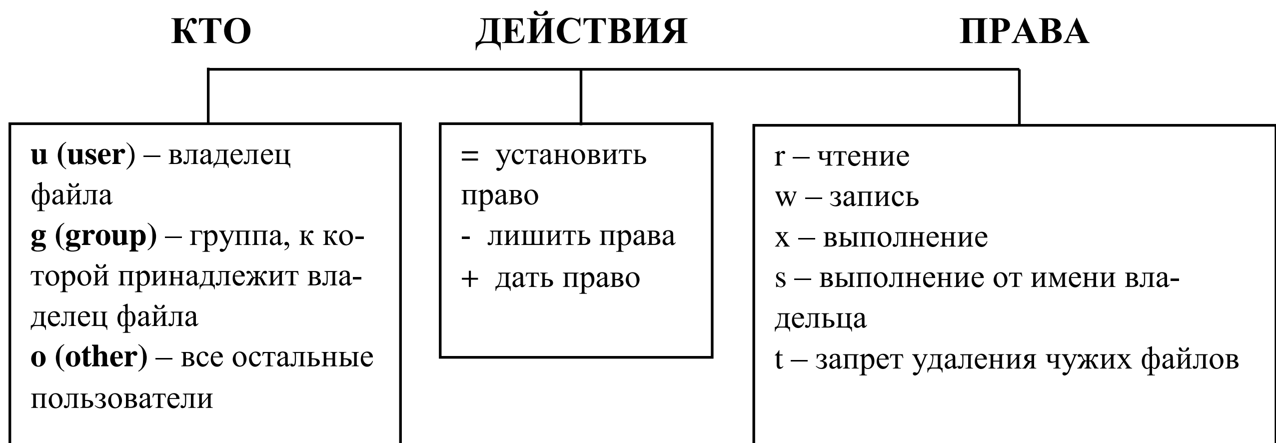
Пример: touch myfile

2.2.2.9 Команда **chmod**.

Назначение: изменение прав доступа к файлу или каталогу.

Формат: chmod режим имя_файла

Здесь «режим» имеет следующую структуру и способ записи:



Права доступа могут быть заданы в команде не только в символьном виде, но и в цифровой форме (восьмеричное значение). Связь между цифровой и символьной формами приведена в табл. 5.

Таблица 5

ЦИФРОВАЯ ФОРМА		СИМВОЛЬНАЯ ФОРМА
двоичная	восьмеричная	
111	7	rwx
110	6	rw-
101	5	r-x
100	4	r--
011	3	-wx
010	2	-w-
001	1	- -x
000	0	- - -

Установка специальных прав доступа в цифровой форме реализуется путем добавления дополнительных битов следующим образом:

chmod 1nnn имя каталога – запрет удаления чужих файлов;

chmod 2nnn имя файла – разрешение на запуск от имени группы, с которой связан файл;

chmod 4nnn имя файла – разрешение на запуск от имени владельца файла.

Отключение специальных прав выполняется следующим образом:

chmod 0nnn имя файла или каталога – для запрета на удаление;

chmod g-s имя файла – для разрешения на запуск от имени группы;

chmod u-s имя файла – для разрешения на запуск от имени владельца;

Здесь *nnn* – стандартный набор прав доступа к файлу или каталогу.

Примеры:
а) chmod g-x june – лишить членов группы права на выполнение файла ./june;
б) chmod 774 october – дать все права доступа к файлу ./october владельцу и членам группы и право чтения для остальных пользователей системы;
в) chmod g+rw,o+r april или chmod 764 april – добавить права чтения и записи в файл ./april членам группы пользователей и права чтения для всех остальных пользователей;
г) chmod +t mykatalog – добавить запрет на удаление чужих файлов в каталоге ./mykatalog;
д) chmod 4rwx myfile – разрешить запуск файла от имени владельца.

2.2.2.10 Команда **find** .

Назначение: используется для поиска и вывода имен файлов, соответствующих заданной строке символов.

Формат: find начальная_точка_поиска [-опции],

где «начальная_точка_поиска» определяет каталог, начиная с которого по всем подкаталогам будет вестись поиск.

Примеры:

а) вывести на экран имена файлов из домашнего каталога и его подкаталогов, начинающихся на f:

find ~ -name "f*" –print, где

-name - после этой опции указывается имя файла, который нужно найти,

"f*" - строка символов, определяющая имя файла,

-print - опция, задающая вывод результатов поиска на экран.

б) вывести на экран имена файлов в каталоге /etc, начинающихся с символа p:

find /etc -name "p*" -print

в) найти в домашнем каталоге файлы, имена которых заканчиваются символом “%” и удалить их:

```
find ~  -name "*%"  -exec rm {} \;
```

Здесь опция -exec rm {}; задает применение команды rm ко всем файлам, имена которых соответствуют указанной после опции -name строке символов.

Комментарий: команда пытается обратиться к каждому из найденных файлов, поэтому в случае отсутствия соответствующих прав доступа на экране рядом с именем найденного файла будет выводиться сообщение об отсутствии прав доступа. Для того, чтобы отфильтровать сообщения об таких ошибках, необходимо направить выходной поток ошибок на фиктивное устройство **null**:

```
find  /etc  -name "p*"  –print  2>/dev/null
```

2.2.2.11 Команда **wc**.

Назначение: вывод числа символов или строк в файле.

Формат: wc [-опции] имя_файла

Примеры:

а) wc –c file1 - выводит число символов в файле file1,

б) wc –l file2 - выводит число строк в файле file1.

2.2.2.12 Команда **ln**.

Назначение: создание ссылки на файл.

Формат: ln [-опции] имя_файла имя_ссылки

Пример: ln folder1/file1 intro - создает в текущем каталоге жесткую ссылку с именем intro на файл folder1/file1. Для создания мягкой ссылки необходимо использовать опцию –s.

2.2.2.13 Команда **mount**.

Назначение: подключение (монтирование) нового устройства к файловой системе.

Формат: mount [-опции] имя_файла имя_ссылки

Пример: mount –t vfat /dev/fd0 /tc – монтирует файловую систему из раздела fd0 в каталог /tc

2.2.2.14 Команда **grep**.

Назначение: поиск заданной символьной строке в указанном файле

Формат: grep строка имя_файла

Примеры:

а) показать строки во всех файлах домашнего каталога с именами, начинающимися на ‘f’, в которых есть слово ‘super’:

grep super f*

б) вывести на экран полную информацию о файлах текущего каталога, которые были изменены 10 февраля:

ls –l | grep "feb 10"

2.2.3 Команды для управления сеансом работы пользователя

2.2.3.1 Команда **who**.

Назначение: - вывод списка активных пользователей.

Формат: who

Комментарий: для вывода имени текущего пользователя существует команда **whoami.**

2.2.3.2 Команда **type**.

Назначение: определение типа команды (внутренняя или внешняя).

Формат: type команда

Пример: type find

Комментарий: для внутренних команд на экран выводится сообщение о том, что команда является встроенной в командный процессор, для внешних - имя каталога, в котором находится соответствующая программа**.**

2.2.3.3 Команда **man**.

Назначение: вывод справочной информации по указанной команде.

Формат: man команда

Пример: man find

Комментарий: для управления просмотром можно использовать следующие клавиши: **пробел**   - перемещение по документу на одну страницу вперед; **ENTER** – перемещение по документу на одну строку вперед; **b (-1) –** возврат на одну страницу; **q** - выход из режима просмотра описания.

2.2.3.4 Команда **history**.

Назначение: выводит на экран список ранее выполненных команд из буфера интерпретатора команд.

Формат: history

Комментарий: информацию, полученную с помощью команды **history**, можно использовать для вызова ранее выполненных команд:

!<номер_команды> - повторное выполнение команды с заданным номером из буфера команд. Например, по команде **!5** будет повторно выполнена пятая команда из буфера.

! <номер-команды>:s/<что_меняем>/<на_что_меняем> - повторное выполнение команды с заданным номером и модификацией командной строки. Например, по команде **!5:s/a/F** будет повторно выполнена пятая команда из буфера с заменой ключа **'a'** на '**F**'.

2.2.3.5 Команда **alias**.

Назначение: назначает псевдоним любой команде Linux.

Формат: alias новое_имя=”команда”

Комментарий: используется для удобства ввода длинных команд. Например, команда alias ll=”ls –a –l” задает новую команду ll, эквивалентную команде ls –a –l Чтобы псевдонимы были доступны при входе в систему, они должны быть описаны в файле ~/.bash_profile. Команда без аргументов выводит список всех псевдонимов.

2.2.3.6 Команда **uname**.

Назначение: вывод информации о версии операционной системы.

Формат: uname

Пример: uname -a

2.2.4 Команды переназначения ввода и вывода

Linux имеет средства переназначения потоков ввода и вывода данных для любых программ, работающих под ее управлением. Например, если какая-либо программа по умолчанию выводит результаты на экран монитора, то можно этот вывод направить в определенный файл на диске. Для этого имеются специальные команды:

'<' - переназначить ввод данных;

'>' - переназначить вывод с замещением данных;

'>>' - переназначить вывод с добавлением данных к уже существующим.

Примеры:

ls>catalog – вывод содержимого текущего каталога в файл catalog;

sort<catalog – сортировка данных из файла catalog.

Краткое описание некоторых полезных команд приведено в таблице 6.

Таблица 6

hostname	Сетевое имя машины
whoami	Имя текущего пользователя
uname -m	Покажет архитектуру машины
uname -r	Версия ядра
cat > имя_файла	В текущем каталоге создаст файл и запишет в него информацию вводимую с клавиатуры. Для завершения ввода с клавиатуры необходимо ввести признак конца файла (CTRL+D)
cat /etc/passwd	Информация об учетных записях пользователей
cat /etc/shells	Информация о доступных командных интерпретаторах
cat /proc/cpuinfo	Информация о процессоре
lscpu	Информация о процессоре
cat /etc/fstab	Информация о смонтированных файловых системах
cat /proc/interrupts	Информация о прерываниях
cat /proc/meminfo	Информация о памяти
cat /proc/swaps	Информация об области свопинга
cat /proc/version	Информация о версии ядра и другая информация
cat /proc/net/dev	Информация о сетевых интерфейсах и их статистика
cat /proc/partitions	Информация о доступных разделах дисковой памяти
cat /proc/modules	Информация о загруженных модулях ядра
date	Текущая дата
cal	Календарь на текущий месяц
echo	Вывести на экран аргументы команды
w	Показывает пользователей в системе, и что они делают
whereis имя	Показать путь к указанной программе (имя)

Справочник по командам ОС Linux можно найти по адресу *//hpc.icc.ru/documentation/cmnds.pdf.*

2.3 Примеры использования команд управления файлами и каталогами

2.3.1. Копирование файла в домашнем каталоге. Скопировать файл ~/abc1 в файл april и в файл may:

```
cd
cp abc1 april
cp abc1 may
```

2.3.2. Копирование нескольких файлов в каталог. Скопировать файлы april и may в каталог monthly :

```
mkdir monthly
cp april may monthly
```

2.3.3. Копирование файлов в произвольном каталоге. Скопировать файл monthly/may в файл с именем june:

```
cp monthly/may monthly/june
ls monthly
```

2.3.4. Копирование каталогов в текущем каталоге. Скопировать каталог monthly в каталог monthly.21:

```
cp -r monthly monthly.21
```

2.3.5. Копирование каталогов в произвольном каталоге. Скопировать каталог monthly.21 в каталог /tmp

```
cp -r   monthly.21 /tmp
```

2.3.6. Переименование файлов в текущем каталоге. Изменить название файла april на july в вашем домашнем каталоге:

```
cd
mv april july
```

2.3.7. Перемещение файлов в другой каталог. Переместить файл july в каталог monthly.21

```
mv july  monthly.21
ls monthly.21
```

2.3.8. Переименование каталогов в текущем каталоге. Переименовать каталог monthly.21 в monthly.05

```
mv  monthly.21 monthly.05
```

2.3.9. Перемещение каталога в другой каталог. Переместить каталог monthly.05 в каталог reports:

```
mkdir reports
mv monthly.05 reports
```

2.3.10. Переименование каталога, не являющегося текущим. Переименовать каталог reports/monthly.05 в reports/monthly:

```
mv reports/monthly.05 reports/monthly
```

2.3.11. Установка и отмена прав доступа:

- установить владельцу файла ./may права на выполнение:

```
chmod u+x may
```

- лишить владельца файла ./may права на выполнение:

```
chmod u-x may
```

- отменить права на чтение каталога monthly для членов группы и всех остальных пользователей:

chmod g-r, o-r monthly

- установить права на запись в файл ./abc1 членам группы:

chmod g+w abc1

2.3.12. Поиск файлов и каталогов:

- найти в домашнем каталоге и подкаталогах файлы с именем may и удалить их:

find ~ -name "may" -exec rm {} \;

- найти все файлы с именем, содержащим строку ‘fstab’:

locate fstab

- найти все файлы с именем fstab:

whereis fstab

Замечание: команда **find** при выполнении поиска пытается войти во все каталоги, начиная с указанной начальной точки поиска, поэтому при отсутствии соответствующих прав доступа на экран часто будет выводиться сообщение об ошибке доступа. Для того, чтобы отфильтровать эти сообщения, рекомендуется отправить их на фиктивное null-устройство следующим образом: find / -name "memo" 2>/dev/null

Здесь проводится поиск файла **memo**, а для фильтрации поток данных с диагностическими сообщениями переназначается на фиктивное устройство. Напомним, что по системным соглашениям любой процесс имеет следующие стандартные потоки данных: 0 –ввод, 1 –вывод, 2 - диагностические сообщения.

3. Порядок выполнения работы

1. Осуществить вход в систему, используя в качестве имени пользователя (pmi-byykk), где yy - номер группы (например, 81,82...), а kk - номер бригады (например, 01,02...). Полезно одновременно открыть два сеанса работы с Linux, что позволит в одном окне знакомиться с документацией по команде, а в другом –выполнять соответствующее задание.

2. Определите полное имя вашего домашнего каталога. Этот каталог будет считаться текущим в последующих упражнениях.

3. Постройте иерархическую структуру файловой системы, чтобы она имела следующий вид, показанный на рисунке 1. Здесь yy – номер группы, kk – номер бригады. Обратите внимание: что часть этой иерархической структуры уже существует, а Вы должны достроить только недостающую часть.

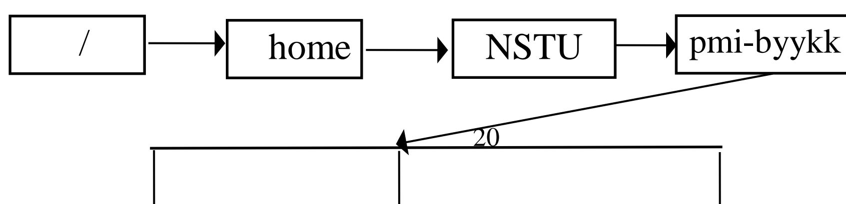


Рис. 1

4. Перейдите в каталог **trash_kk**.

5. Посмотрите содержимое каталога **trash_kk**, используя в команде **ls** различные опции.

6. Проверьте, существует ли в корневом каталоге файл с именем **unix**.

7. Существует ли в каталоге **/var/spool** подкаталог с именем **cron**? Если существует, то докажите, что это именно каталог.

8. Посмотрите содержимое вашего домашнего каталога. Кто является владельцем его файлов и подкаталогов?

9. Создайте с помощью команды **cat** в домашнем каталоге файл *abc1*, в который запишите свою фамилию, имя, отчество, наименование учебной группы и номер бригады.

10. Выполните все примеры, приведенные в разделе 2.3. Результаты выполнения команд занесите в отчет.

11. Создайте три новых каталога с именами **letters, memos, misk** в вашем домашнем каталоге одной командой.

12. Удалите эти каталоги одной командой.

13. Попробуйте удалить каталог **~/tmp_kk** командой **rm**. Что получилось? Как можно удалить этот каталог ?

14. С помощью команды **man** определите, какая опция команды **ls** позволяет просматривать не только содержимое указанного каталога. но и подкаталогов, входящих в него. Проверьте работу этой опции.

15. Определите при помощи команды **man**, какой набор опций команды **ls** позволяет отсортировать список с развернутым описанием файлов по времени последнего изменения. Создайте псевдоним для этой команды с именем “l_kk”, проверьте его работоспособность.

16. Определите типы команд cd, pwd, find, grep, vim.

17. Определите, используя конвейер команд **who** и **wc**, количество пользователей, подключенных к серверу в данный момент времени.

18. Получите информацию по оборудованию и операционной системе сервера fpm2.ami.nstu.ru. Сведения о процессоре получите с помощью команд **cat** и **lscpu,** сравните их результаты (модель и количество процессоров, число ядер, тактовая частота, размер кэш-памяти). Сведения о параметрах памяти можно получить командой **cat**, об установленной ОС – командой **uname**.

19. Создайте в домашнем каталоге жесткую и мягкую ссылки на файл *abc1* с именами *hard_abc1* и *soft_abc1* соответственно, с помощью команды **ls –l** убедитесь, что ссылки созданы. Подтвердите работоспособность ссылок.

20. Удалите файл *abc1* и проверьте работоспособность ссылок.

21. Используя информацию, полученную командой **history**, выполните модификацию и исполнение нескольких команд из буфера команд (по указанию преподавателя).

22. Сохраните в файле **history_kk** все команды, которые Вы выполнили в лабораторной работе. Включите в отчет фрагмент этого файла.

4. Контрольные вопросы

1. Дайте определение командной строки. Приведите примеры.
2. Как определить абсолютное имя текущей директории?
3. Проведите сравнение понятия файла в MS Windows и Linux.
4. Используется ли понятие устройства при обращении к файлу в ОС Linux?
5. Как восстановить удаленные файлы в ОС Linux?
6. Назовите и дайте характеристику основным типам файлов в ОС Linux.
7. Как определить только тип файлов и их имена в текущем каталоге?
8. Какие файлы считаются скрытыми? Как получить информацию о скрытых файлах?
9. Как удалить файл и каталог?
10. Как определить, какие команды выполнил пользователь в сеансе работы? Какие проблемы при этом могут возникнуть?
11. Каким образом можно исправить и запустить на выполнение команду, которую пользователь уже использовал в сеансе работы?

12. Можно ли в одной строке записать несколько команд? Если да, то как?

13. Что такое символ экранирования? Приведите примеры использования этого символа.

14. Какая информация выводится на экран о файлах и каталогах, если используется опция **l** в команде **ls**?

15. Что такое относительное имя файла? Приведите примеры.

16. Назовите и дайте характеристику командам, которые позволяют просмотреть текстовые файлы.

17. Назовите и дайте характеристику командам перемещения и переименования файлов и каталогов.

18. Что такое права доступа, как они могут быть изменены?

19. Назовите и дайте характеристику командам поиска файлов. Приведите примеры использования этих команд.

Лабораторная работа № 2. Текстовый редактор Vim

1. Цель работы

Целью работы является освоение основных возможностей экранного редактора **vim,** который является расширением стандартного UNIX-редактора **vi**.

2. Методические указания

2.1 Общие сведения

Особенностью ОС семейства UNIX является широкое использование текстовых файлов для хранения системных данных, поэтому текстовый редактор является обязательной и важнейшей программой, поставляемой в дистрибутиве ОС. Именно с помощью редактора настраивается система, редактируются общесистемные и пользовательские конфигурационные файлы, пишутся скрипты и сценарии.

Все текстовые редакторы POSIX-совместимых ОС делятся на два класса - редакторы командного стиля и меню - ориентированные редакторы. В первых навигация по тексту и его обработка осуществляется путем ввода команд, например, перейти на пять слов вперед, заменить строку номер десять и т.д. Примерами командных редакторов являются редакторы **vi** и **vim**.

Действия в меню - ориентированных редакторах, осуществляются путем выбора одного из предлагаемых в меню вариантов действий. Примерами таких редакторов являются Jed и редактор Midnight Commander.

Файлы, создаваемые текстовым редактором, имеют кодировку ASCII, и могут быть прочитаны в любой среде, не требуя специальных конвертеров. Это особенно важно для документов с символами кириллицы. С помощью редакторов можно готовить html-страницы, осуществлять верстку документов, править конфигурационные файлы, создавать программы и т.д.

Редактор **vi** (или какой-либо из его клонов) - непременный атрибут всех Unix-систем, и потому любой их пользователь должен иметь о нем представление. Это интерактивный экранный редактор, который используется для создания и редактирования текстовых файлов. Все действия **vi** производит в буфере. Произведенные изменения могут быть записаны на диск или отменены. Редактор **vi** имеет три режима: командный, вставки/ввода и последняя строка.

Командный режим позволяет управлять курсором и вводить команды редактирования. *Режим вставки* допускает производить ввод текста, при этом текст не будет восприниматься, как команды редактирования. *Режим последней строки* позволяет производить запись файла на диск, завершать работу с редактором, а также вводить дополнительные команды редактирования текста.

Вызов редактора **vi** осуществляется с помощью команды: **vi имя_файла**

Если файла с указанным именем не существует, то он будет создан, иначе файл с указанным именем будет загружен для редактирования.

Редактирование файла осуществляется с помощью команд редактирования и позиционирования. Нажатие клавиши <ESC> всегда переводит **vi** в командный режим (это удобно, когда Вы точно не помните в каком режиме находитесь). Если Вы нажмете клавишу <ESC>, находясь в командном режиме, редактор напомнит вам об этом, подав звуковой сигнал.

Для выхода из **vi** в командном режиме необходимо нажать символ **‘:’**, который переводит редактор в режим *последней строки*. В этой строке необходимо набрать символы **wq** для записи изменений в файл на диске и выхода из редактора. Если необходимо закрыть **vi** без сохранения выполненных изменений, необходимо в последней строке набрать символ q (или q!). Редактор различает прописные и строчные буквы, поэтому при использовании команд обращайте внимание на их правильное употребление.

2.2.Основные группы команд редактора

2.2.1 Команды управления курсором

Команды управления курсором приведены в табл. 7.

Таблица 7

Курсор влево	**Курсор вправо**	**Курсор вверх**	**Курсор вниз**
backspace	Space		ENTER
h	l	k	j
←	→	↑	↓

2.2.2 Команды позиционирования**:**

0 (ноль) - перейти в начало строки;

$ - перейти в конец строки;

G - перейти в конец файла;

nG или :n - перейти на строку номер n;

2.2.3 Команды перемещения по файлу:

Ctrl + d - переместиться на 1/2 экрана вперед;

Ctrl + u - переместиться на 1/2 экрана назад;

Ctrl + f - переместиться на страницу вперед;

Ctrl + b - переместиться на страницу назад.

2.2.4 Команды перемещения по словам:

W или w - переместиться на слово вперед;

nW или nw - переместиться на n слов вперед;

b или B - переместиться на слово назад;

nb или nB - переместиться на n слов назад.

Примечание:

а) при использовании прописных W и B под разделителями понимаются только пробел, табуляция и возврат каретки.

б) при использовании строчных w и b под разделителями понимаются также любые знаки пунктуации.

2.2.5 Команды редактирования

Добавление / вставка текста:

a - добавить текст после курсора;

A - добавить текст в конец строки;

i - вставить текст перед курсором;

ni - вставить текст n раз (например, напечатав 10i, затем privet, затем ESC, Вы можете вставить слово “privet” 10 раз;

I - вставить текст в начало строки.

Вставка строки:

o - вставить строку под курсором;

O - вставить строку над курсором.

Удаление текста:

x - удалить один символ в буфер;

dw - удалить одно слово в буфер;

d$ - удалить в буфер текст от курсора до конца строки;

d0 (ноль) - удалить в буфер текст от начала строки до позиции курсора;

dd - удалить в буфер одну строку;

10dd - удалить в буфер 10 строк.

Отмена и повтор произведенных изменений:

u - отменить последнее изменение;

. - повторить последнее изменение.

Копирование текста в буфер:

v – перейти в визуальный режим;

стрелки – выделение блока текста в визуальном режиме;

y – копировать выделенный блок текста в буфер;

c - вырезать выделенный блок текста в буфер;

Y - скопировать строку в буфер;

nY - скопировать n строк в буфер;

yw - скопировать слово в буфер.

Вставка текста из буфера:

p - вставить текст из буфера после курсора;

P - вставить текст из буфера перед курсором.

Замена текста:

cw - заменить слово;

ncw - заменить **n** слов;

c$ - заменить текст от курсора до конца строки;

r - заменить символ;

R - заменить текст.

Поиск и замена текста:

/текст - произвести поиск указанной строки вперед по тексту;

?текст - произвести поиск указанной строки назад по тексту;

:s/текст - произвести поиск указанной строки по всему тексту;

2.2.6 Команды редактирования в режиме последней строки

Копирование и перемещение текста:

:i,jd - удалить строки с i по j, например **:3,8d**

:i,jm k - переместить строки с i по j, начиная со строки k, например :4,9m 12

:i,jt k - копировать строки с i по j, начиная со строки k, например :2,5 t 13

:i,jw имя_файла - записать строки с i по j в файл с именем **имя_файла**, например :5,9 test.txt

Поиск и замена текста:

:s/текст1/текст2 – найти и заменить первое появление символов **текст1** в текущей строке на **текст2**;

:%s/текст1/текст2/g - найти и заменить каждое появление символов **текст1** на **текст2** в файле;

:i,js/текст1/текст2/g - найти и заменить каждое появление символов **текст1** на **текст2** в диапазоне строк [i,j];

Запись в файл и выход из редактора:

:w - записать измененный текст в файл на диске без выхода из **vi** ;

:w newfile - записать измененный текст в новый файл с именем newfile;

:w! имя_файла - записать измененный текст в файл с именем **имя_файла**;

:wq - записать изменения в файл и выйти из **vi**;

:q - выйти из редактора **vi** ;

:q! - выйти из редактора без записи;

:e! - вернуться в командный режим, отменив все изменения, произведенные со времени последней записи.

Примечание. Обязательно укажите имя файла при выходе из **vi**, если при запуске редактора вы этого не сделали.

2.3 Опции редактора.

Рабочая среда редактора vi настраивается с помощью нескольких десятков опций, для задания которых используется команда **set** в режиме последней строки, например:

:set all - вывести полный список опций;

:set nu - вывести номера строк;

:set list - вывести невидимые символы;

:set ic - включить регистронезависимый поиск символов.

Примечание. Если вы хотите отказаться от использования опции, то в команде **set** перед именем опции надо поставить '**no'**, например **:set nonu** прекращает вывод номеров строк.

3. Порядок выполнения работы

Задание 1. Создание нового файла

1. Создайте каталог с именем **practice**;

2. Перейдите во вновь созданный каталог.

3. Вызовите **vi** и создайте файл с именем 'memo' (**vi memo**).

4. Нажмите клавишу **i** и введите следующий текст:

```
@REM AUTOEXEC.BAT DTK 386/40
ECHO OFF
Path c:\dos;c:\stacker;c:\Util;c:\NC;C:\MOUSE
SET PROMPT=$P$G
SET TMP=C:\TEMP
LH C:\UTIL\RKEGA
goto %config%
:student1
C:\DOS\SMARTDRV.EXE C+ 2048 1024
goto nc
:student2
APPEND E:\tc\bgi
:teacher
PATH %path%E:\windows;e:\tc;e:\tc\bin;e:\foxpro;
goto win
:onc
PATH %path%G:\pctcp;
SET TZ=GMT
goto nc
:nc
nc.exe
goto end
win.com
:end
```

5. Нажмите клавишу ESC для перехода в командный режим после завершения ввода текста.

6. Нажмите '**:**' (двоеточие) для перехода в режим последней строки.

7. Нажмите **wq** (записать и выйти), а затем нажмите клавишу ENTER для сохранения вашего текста и завершения работы.

Задание 2. Редактирование существующего файла

1. Вызовите **vi** для редактирования файла **memo.**

2. Установите курсор на начало слова 'DTK' в первой строке.

3. Перейдите в режим вставки и наберите '1-203'; теперь текст будет выглядеть так:

@ REM AUTOEXEC.BAT 1-203 DTK 386/40

Нажмите ESC для возврата в командный режим.

4. Установите курсор на третью строку и сотрите слово C:\MOUSE.

5. Перейдите в режим вставки, наберите следующий текст: C:\GMOUSE и нажмите ESC, чтобы вернуться в командный режим.

6. Установите курсор на последней строке файла. Вставьте строку, содержащую следующий текст: extention 287

7. Замените слово **extention** на **x**.

8. Удалите последнюю строку.

9. Введите команду отмены изменений **u** для отмены последней команды.

10. Установите курсор на строку 5, вставьте перед ним пустую строку и введите следующий текст: @REM this is a comment

Оставьте пустую строку между новым параграфом и следующим за ним. Нажмите ESC для возврата в командный режим.

11. Введите символ ‘:’ (двоеточие) для перехода в режим последней строки, запишите произведенные изменения на диск и выйдите из редактора.

Задание 3. Заключительное упражнение

1. Вернитесь в ваш домашний каталог.

2. Скопируйте в каталог **~/practice** файл **testcase.c,** предварительно осуществив его поиск во внешней памяти сервера.

3. Перейдите в каталог **~/practice** и загрузите **vi** для редактирования файла **testcase.c**

4. Включите отображение номеров строк. Сколько строк в данном файле?

5. Вернитесь в начало файла.

6. Найдите первое слово **WORD** и замените его на **IGNORE**.

7. Найдите слово **Reset** и замените его на **set**.

8. Найдите слово **input** и замените его на **output.**

9. Вставьте строку, заполненную вопросительными знаками **‘?’** под строкой: **state** = **WORD**

10. Скопируйте строки с 16 по 29 в файл **printwords.c**

11. Перейдите в конец файла и удалите две последние строки.

12. Вернитесь в начало файла и перенесите фрагмент текста, начинающийся словами **/*Manifests** .... в конец файла.

13. Запишите произведенные изменения на диск в файл **testvi.c** и выйдите из редактора.

4. Контрольные вопросы

1. Дайте краткую характеристику режимам работы редактора **vi**.

2. Как выйти из редактора, не сохраняя произведенные изменения?

3. Назовите и дайте краткую характеристику командам позиционирования.

4. Что для редактора **vi** является словом?

5. Каким образом из любого места редактируемого файла перейти в начало или в конец файла?

6. Назовите и дайте краткую характеристику основным группам команд редактирования?

7. Необходимо заполнить строку символами **‘$’** - ваши действия?

8. Как отменить некорректное действие, связанное с процессом редактирования?

9. Назовите и дайте характеристику основным группам команд режима последней строки.

10. Выполните анализ опций редактора **vi** (сколько их, как узнать их назначение и т.д.).

11. Как определить режим работы редактора **vi**?

12. Постройте граф взаимосвязи режимов работы редактора **vi**.

13. Назовите основные типы текстовых редакторов POSIX - совместимых ОС. Дайте им краткую характеристику.

14. Почему знание основных возможностей редактора **vi** необходимо для пользователей POSIX – систем?

Лабораторная работа № 3. Командные сценарии Linux

1. Цель работы

Целью работы является изучение расширенных возможностей командного языка Shell.

2. Методические указания

2.1 Общие сведения

Командный сценарий (скрипт, командный файл) – это исполняемый текстовый файл, состоящий из команд интерпретатора. Сценарий часто называют shell–программой и используют для администрирования и управления ОС. Язык командного интерпретатора семейства UNIX, в отличие от интерпретатора Windows, является языком высокого уровня, поддерживающим ввод-вывод переменных, ветвления (if-then-else, case), циклы (for, while, until), работу с функциями, пошаговый режим отладки и т.д.

Запуск сценария может проводиться несколькими способами:

- запустить интерпретатор, указав ему в качестве аргумента имя файла сценария, например: ***sh myscript.sh***
- выполнить сценарий в текущем экземпляре shell, например:

  . myscript.sh или ***source myscript.sh***
- установить для сценария файловый атрибут «исполняемый» и запустить как обычную программу: ***chmod 755 myscript.sh; ./myscript.sh***

При написании сценария необходимо соблюдать следующие требования:

- в первой строке обычно указывается имя интерпретатора, который будет исполнять сценарий, например #!/bin/bash;
- в конце сценария обязательно вводится символ перевода строки (нажатие ENTER).

В дистрибутив Linux обычно включаются несколько интерпретаторов команд, которые хранятся в каталогах /bin и /usr/bin. Список доступных интепретаторов можно просмотреть в файле /etc/shells (например, sh, bash, csh и др.), имя текущего интерпретатора хранится в глобальной переменной SHELL.

2.2 Переменные сценария

Каждый запущенный экземпляр командного интерпретатора shell имеет определенный набор стандартных глобальных переменных, значения которых можно просмотреть командами **env** или **printenv**. В таблице 8 приведены назначения некоторых переменных.

Таблица 8

Переменная	Значение
?	Код завершения последней команды (0 – нормальное завершение)
$	PID текущего процесса shell
!	PID последнего фонового процесса
#	Число параметров, переданных в shell
*	Список параметров shell в виде одной строки
@	Список параметров shell в виде набора слов
-	Флаги, передаваемые в shell
HOME	Имя домашнего каталога
PATH	Пути поиска исполняемых файлов
PS1	Вид системной подсказки
SHELL	Полное имя файла текущего командного интерпретатора
USER	Имя учетной записи текущего пользователя

Пользователь может использовать в сценариях глобальные переменные интерпретатора и собственные локальные переменные. Имена переменных представляют собой последовательность букв, цифр и символов подчеркивания, имя должно начинаться с буквы или символа подчеркивания. Переменные всегда имеют строковый тип и задаются сразу со значением (возможно с пустым).

При объявлении переменной ее имя и значение должны быть записаны без пробелов относительно символа равенства. Например, создадим две переменных, присвоив переменной VAR_1 значение “245”, а переменной VAR_2 пустое значение:

```
VAR_1=245
VAR_2=
```

Если для переменной задается значение, содержащее пробелы, то нужно заключать его в кавычки:

```
VAR_3=”program files”
```

В качестве значения переменной можно присвоить результат выполнения команды Linux, указав имя команды в обратных кавычках (обычно этот символ находится на клавише для ввода символов «~» и «ё»):

```
VAR_2=`pwd`
```

Значение переменным можно задать в диалоговом режиме, например, команда

read A B C

присваивает значения переменным A, B и C.

Вывод значений переменных на экран проводится командой ***echo*** с указанием символа “$” перед именем переменной:

echo $VAR_3

При необходимости выполнения конкатенации значения переменной и произвольной символьной строки используются фигурные скобки:

echo ${VAR_3}binary

Язык SHELL поддерживает не только простые переменные, но и массивы. Присвоение значений элементам массива можно проводить путем общего описания массива или индивидуально. Например, строка

steps=(5 10 15)

создает массив, состоящий из трех элементов и присваивает им соответствущие значения. Такой же результат можно получить другим способом:

steps[1]=5; steps[2]=10; steps[3]=15

Обращение к элементам массива проводится с использованием фигурных скобок. Например, для чтения второго элемента описанного выше массива в переменную *step* надо записать:

step=${steps[2]}

Командный интерпретатор с помощью команды ***expr*** имеет возможность обработки переменных как целых чисел, выполняя операции сложения (+), вычитания (-), умножения (*), целочисленного деления (/) и получения остатка от деления (%):

a=`expr $x + $y`; b=`expr $x - $y`

c=`expr $x / $y`; d=`expr $x % $y`; e=`expr $x “*” $y`

Замечание: символ умножения должен быть взят в кавычки, т.к. это служебный символ интерпретатора, а имена переменных и знаки операций всегда разделяются пробелами.

Аналогичные результаты можно получить с помощью команды ***let***, которая сама реализует преобразование строковых значений в числовые:

let “a=x+y”; let “b=x-y”; let “c=x/y”; let “d=x%y”; let “e=x*y”

Удаление переменных выполняется командой ***unset***, например unset VAR_3.

Все локальные переменные доступны только в том процессе shell, в котором они были объявлены. Для того, чтобы переменная стала глобальной и к ней можно

было обращаться из дочерних процессов, ее следует экспортировать командой **export**. Возможны два способа экспорта:

- при создании переменной (например, export VAR_4=myfolder);
- после создания переменной (например, export VAR_4)

Команда ***export*** без аргументов выводит список всех экспортированных переменных интерпретатора.

2.3 Передача данных в командный файл

Часто возникает необходимость создания сценария, который должен обрабатывать различные наборы входных данных. В этом случае возможны два способа передачи данных в сценарий: а) с помощью списка параметров, б) с помощью глобальных переменных.

При использовании первого способа сценарий может принимать до 10 аргументов, используя механизм формальных и фактических параметров. Для описания формальных параметров в программе используются специальные переменные, имена которых состоят из одной цифры в диапазоне от 0 до 9, перед которой стоит символ '$'.

При запуске сценария в командной строке указываются значения фактических параметров, которые подставляются на место соответствующих формальных параметров. Формальный параметр $0 содержит имя сценария, параметр $1 – значение первого фактического параметра, параметр $2 – второго фактического параметра и т.д. Общее число параметров содержится в переменной # .

Если число фактических параметров превышает число формальных параметров, то в сценарии можно использовать команду **shift n**, которая сдвигает список формальных параметров относительно списка фактических параметров на **n** позиций (по умолчанию n=1). <u>Обратите внимание</u>: при использовании команды **shift** значение параметра $0 не изменяется.

Второй способ (глобальные переменные) удобно использовать при обмене данными между сценариями. При этом в родительском сценарии с помощью команды **export** необходимо создать соответствующие глобальные переменные и присвоить им заданные значения, после чего можно использовать эти переменные в дочерних сценариях.

2.4 Окружение сценария

Для выполнения командного сценария Linux создает отдельный экземпляр командного интерпретатора, поэтому глобальные переменные ‘*’, ‘#’ и ‘@’ имеют собственные значения для каждого запущенного сценария. Все переменные, известные сценарию во время выполнения, образуют его окружение, включающее переменные, которые сценарий получает путем наследования от родительского процесса (от интерпретатора shell) и собственные локальные переменные.

При загрузке системы командный интерпретатор Shell выполняет команды, содержащиеся в файле **/etc/profile**, затем команды, содержащиеся в файле **$HOME /.profile**. Для того, чтобы при входе в систему автоматически устанавливались необходимые переменные, следует откорректировать файл **.profile**, находящийся в Вашем домашнем каталоге.

2.5 Операторы языка Shell

2.5.1 Оператор TEST

Оператор test проверяет выполнение заданного условия и возвращает логическое значение «истина» (0) или «ложь» (1).

Общий формат оператора:

test *условие* или [*условие*]

Замечание: вторая форма записи оператора предусматривает наличие обязательных пробелов между квадратными скобками и выражением условия. Если строка условия в команде test пустая, то возвращается значение 1.

Возможны условия нескольких видов:

а) проверка файлов или переменных

формат команды: test ключ имя_файла | имя_переменной

ключ команды задает режим проверки и может принимать следующие значения

-f - указанный файл является обычным файлом;

-d - указанный файл является обычным каталогом;

-c - указанный файл является символьным устройством;

-r - указанный файл доступен для чтения;

-w - указанный файл доступен для записи;

-s - указанный файл не является пустым;

-n - указанная переменная имеет значение (не пустая);

-z - указанная переменная не имеет значения.

Примеры команды:

test –f /usr/user1/file_1   возвращает 0, если файл /usr/user1/file_1 является обычным файлом;

test –w /usr/user1/file_1   возвращает 0, если файл /usr/user1/file_1 доступен для записи;

test –n $ALFA   возвращает 0, если переменная ALFA имеет значение.

2) сравнение строк

формат команды: test строка1 отношение строка2

операции отношения строк: = (равно)   или   != (не равно) .

Примеры команд:

test $A = $B   возвращает 0, если значение переменной A равно значению переменной B;

test $A != $B возвращает 0, если значение переменной A не равно значению переменной B.

Замечание: операция отношения обязательно с двух сторон окружается пробелами.

3) сравнение целых чисел.

формат команды: test число1 отношение число2

операции отношения чисел:

-eq - равно;   -ne - не равно;

-gt - больше;   -ge - больше или равно;

-lt - меньше;   -le - меньше или равно.

2.5.2 Оператор IF

Условный оператор IF имеет следующий формат:

if *условие* then
    список_операторов
  [elif *условие* then
      список_операторов]
  [else *список_операторов*]
fi

Здесь *условие* – любой оператор, возвращающий значение true или false (обычно используется оператор *test*);

список_операторов - операторы, разделенные точкой с запятой (;).

Пример оператора:

```
if [ -f  /tmp/myfile ] then
    cksum  /tmp/myfile
  else
```

```
ls /tmp > /tmp/myfile;  cksum  /tmp/myfile
fi
```

2.5.3 Оператор CASE

Оператор множественного выбора CASE имеет следующий формат:

```
Case   строка   in
   шаблон_1)      список_операторов;;
   шаблон_2)      список_операторов;;
   шаблон_n)      список_операторов;;
esac
```

Пример оператора:

```
echo  –n ‘please enter symbol from A to C:  ‘
read ENTRY
case $ENTRY in
    A|a)  echo ‘Apple’;;
    B|b) echo ‘Baby’;;
    C|c)  echo ‘Coke’;;
    *) echo ‘Please type A, B or C ! ’;;
esac
```

2.5.4 Оператор FOR

Оператор предопределенного цикла FOR имеет следующий формат:

```
for имя  [in список]
 do
  список_операторов
done
```

Если список значений для переменной не задан, то в качестве списка используется значение внутренней переменной @, содержащей список аргументов сценария (см. табл. 7). Пример оператора:

```
list=”word 1 word2 word3 word4”
for   VAL in $LIST do
  echo $VAL
done
```

2.5.5 Операторы WHILE и UNTIL

Операторы WHILE и UNTIL предназначены для создания циклов с неопределенным числом исполнения списка операторов. *Список_операторов* оператора

WHILE исполняется только тогда, когда *условие* истинно, а для оператора UNTIL – когда условие ложно. Выход из циклов проводится путем контроля значения *условия* или принудительно по команде break.

Форматы операторов:

while *условие*	until *условие*
do	do
список_операторов	*список_операторов*
done	done

2.5.6 Использование функций в сценариях

Функции Shell описываются следующим образом:

```
имя_функции ()
(
список_операторов
)
```

Функции поддерживают механизм формальных и фактических параметров. Формальные параметры являются локальными и имеют имена от 0 до 9. Если функция должна вернуть код завершения, то используется оператор *return*

2.6 Создание сценария

Файл сценария можно создать несколькими способами. Первый способ основан на команде ***echo*** и может использоваться для создания только очень простых сценариев. Если файл должен содержать несколько строк, то в команде ***echo*** необходимо с помощью ключа **-e** разрешить обработку специальных символов и для перевода строки применять символы '\n', например:

```
echo -e "ls -l \nman echo" > echo_primer.sh
```

Здесь в файл echo_primer.sh будут записаны две строки:

```
ls –l
man echo
```

Второй способ использует команду cat:

```
cat >cat_primer.sh
< команды >
нажатие ENTER
Ctrl+D
```

Третий способ основан на применении текстового редактора **vi.** Редактор запускается следующим образом: **vi имя_файла**. Если файл с именем **имя_файла** не существует, то он будет создан.

Обратите внимание: командный интерпретатор работает с кодировкой ASCII, поэтому скрипты Linux должны создаваться только в этой кодировке.

3. Порядок выполнения работы

1. Посмотреть справку по команде ***echo***, ознакомиться с ее ключами.
2. Освоить способы создания сценария, описанные в разделе 2.6
3. Разработать сценарий для создания файла данных с информацией согласно таблицы 9. Количество записей в файле – не менее 10, в скобках указано смещение поля относительно начала записи. Привести пример запуска сценария и результат его работы. Данные в сценарий передавать через параметры, форматирование полей записи проводить путем циклического добавления необходимого числа пробелов, для чего возможны два варианта.

Первый вариант предполагает организацию отдельного цикла для каждого параметра. Например, запись значения переменной fam, для которой выделено 20 позиций в выходной строке, можно провести следующим образом:

```
echo –n  $fam >>$file_dat
len_fam=`expr length $fam`
 [ $len_fam  –lt  20 ]
do
  #записываем пробел без перевода строки
  echo –n  “  “ >>$file_dat
  len_fam=`expr $len_fam  + 1`
done
```

Здесь *len_fam* – переменная, в которой храниться длина введенного пользователем значения поля, для определения длины используется функция *length*.

Обратите внимание: функция *length* определяет размер указанной переменной в байтах, поэтому для корректного форматирования файла данных необходимо все значения фактических параметров сценария вводить с использованием латиницы, т.к. символы кириллицы в кодировках UNICODE и UTF-8 занимают 2 байта.

Второй вариант организует единый цикл для всех передаваемых в сценарий параметров:

```
for i in $1 $2 $3
do
```

```
echo –n $i >> $file_dat
len_param=`expr length $i`
 while [[ $len_ param  –lt  ${steps[$k]} ]]
    do
     echo –n  “  “ >>$file_dat
     let “len_param=len_param+1”
    done
 let “k=k+1”
done
```

Таблица 9

№ варианта	Предметная область	Название полей и смещение относительно начала записи (номер начальной позиции)	Способ обмена данными (для п.6 задания)
1	Студенческая группа	Фамилия (0), год рождения (20), место рождения (25), средний балл (40).	Список параметров
2	Расписание поездов	Номер поезда (0), пункт назначения (5), время отправления (30), время в пути (40).	Глобальные переменные
3	Туристическая фирма	Номер тура (0), страна (5), город (20), продолжительность тура (40), дата отправления (45)	Список параметров
4	Расписание самолетов	Номер рейса (0), пункт назначения (10), время отправления (30), время в пути (40)	Глобальные переменные
5	Склад	Код товара (0), группа товара (5), наименование товара (20), дата поступления (40), цена (50)	Список параметров
6	Отдел кадров	Фамилия (0), год рождения (20), место рождения (25), должность (45), адрес, телефон (55)	Глобальные переменные
7	Успеваемость студентов	Группа (0), фамилия (10), номер семестра (30), название дисциплины (32), оценка (50).	Список параметров
8	Договорной отдел	Шифр договора (0), заказчик (10), источник финансирования (30), сумма договора (50), дата начала (60), дата окончания (70)	Глобальные переменные
9	Бухгалтерия	Табельный номер (0), фамилия (7), оклад (30), надбавка (40), наличие налогового вычета (45),	Список параметров
10	ГИБДД	Госномер автомобиля(0), модель (10), год выпуска (25), тип двигателя (30), адрес регистрации (40), дата регистрации (65)	Глобальные переменные

4. Разработать сценарий для поиска в файле данных заданной пользователем строки символов. Данные в сценарий передавать через параметры, привести пример запуска сценария и результат его работы.

5. Разработать сценарий для сортировки файла данных по заданному пользователем полю записи. Данные в сценарий передавать через параметры, привести пример запуска сценария и результат его работы.

6. Разработать сценарий, формирующий меню для опроса пользователя (пункты меню – добавление данных, поиск данных, сортировка данных, просмотр файла данных, выход) и выполнение этого меню с помощью вызова соответствующих сценариев, разработанных в п. 3–5 задания. В сценарий необходимо передать в виде параметров номер Вашей группы и номер бригады; при работе сценарий должен в первой строке выводить текущую дату и значения принятых параметров, а затем – строки меню. Передачу данных в дочерние сценарии провести заданным в табл. 9 способом. Привести пример запуска сценария и результаты его работы по каждому пункту меню.

4. Контрольные вопросы

1. Командный сценарий: определение, назначение.
2. Способы запуска командного сценария.
3. Способы передачи данных в командный сценарий.
4. Переменные в командном сценарии: типы, область видимости, допустимые операции.
5. Какие виды циклов могут использоваться в сценарии.
6. Какие виды ветвлений могут использоваться в сценарии.
7. Поясните алгоритм форматирования записей при выполнении п. 3 лабораторной работы.

Лабораторная работа № 4. Файловые системы ОС Linux

1. Цель работы

Целью работы является изучение файловой системы ОС Linux и приобретение практических навыков применения команд для анализа файловой системы, управления файлами и процессами.

2. Методические указания

При изучении внешней памяти следует различать два понятия – *устройство внешней памяти* и *файловая система*. Устройство представляет собой физический носитель, на который записываются данные, а файловая система определяет способ хранения и доступа к этим данным. Устройства внешней памяти современных компьютеров очень разнообразны. К ним относятся различные типы дисков (магнитные, оптические, магнито-оптические), твердотельные накопители (SSD), USB флэш-накопители, карты памяти и т.д. Наиболее широко применяются магнитные диски, поэтому в данной работе будет изучаться именно этот класс устройств.

Напомним, что основные отличия файловых систем семейства ОС UNIX от файловых систем семейства ОС Windows с точки зрения пользователя сформулированы в описании лабораторной работы № 1.

2.1 Модель внешней памяти

2.1.1 Физическая модель диска

На физическом уровне работа с устройством внешней памяти всегда проводится через соответствующий контроллер. Именно контроллер знает физические параметры устройства, поэтому физическая модель магнитных дисков не зависит от операционной системы (Linux или Windows).

Информация на диске размещается вдоль концентрических окружностей, называемых *дорожками (*tracks*)*. Дорожки с одинаковыми номерами на различных *поверхностях* диска образуют *цилиндр*, для записи-чтения дорожек на каждой поверхности диска выделяется отдельная магнитная *головка*. Дорожка содержит определенное количество *секторов* – это участок дорожки, хранящий минимальную порцию информации (обычно 512 байтов), которая может быть считана или

записана за одно обращение к диску. Дорожки нумеруются в пределах поверхности, а сектора – в пределах дорожки.

Возможны два способа разбивки дорожек на сектора – с постоянным количеством секторов на дорожке и с переменным количеством секторов на дорожке. В первом способе каждая дорожка диска имеет одинаковое число секторов, что обеспечивается изменением плотности записи при переходе с одной дорожки на другую. Максимальная плотность записи используется на дорожках с минимальным радиусом. Для получения доступа к произвольному сектору диска необходимо указать его *физический адрес*, который включает номер цилиндра (cylinder), номер головки (head) и номер сектора (sector). Такой способ адресации называется *CHS* и может использоваться только для дисков небольшого объема (до 500 Мбайт).

Современные диски большого объема используют второй способ разбиения, при котором дорожки на каждой поверхности диска группируются в зоны. Все дорожки в одной зоне имеют одинаковое количество секторов, дорожки в зонах с меньшим радиусом имеют меньше секторов, чем зоны с большим радиусом, а плотность записи остается постоянной. Таким образом, без изменения технологии производства увеличивается общее число секторов на диске и, следовательно, его объем. При этом используется линейная адресация всех секторов диска (LBA). В настоящее время для задания LBA-номера сектора используется 6 байтов, что даёт возможность адресовать на диске 2^{48} секторов.

Таким образом, физическая модель диска включает следующие понятия: дорожка, цилиндр, поверхность и сектор. Все диски отличаются друг от друга по набору этих параметров. Физическая модель диска используется его контроллером.

Синонимом термина «физический диск» является термин «физический том».

2.1.2 Логическая модель диска

С логической точки зрения все адресное пространство диска представляет собой набор последовательно пронумерованных секторов. Небольшая часть этих секторов выделяется для хранения служебной информации и называется системной областью диска (областью метаданных), остальные сектора предназначены для хранения файлов и каталогов и образуют область данных. С логической моделью диска работает операционная система.

Единица размещения дисковой памяти называется блоком (в ОС Windows –кластером), который состоит из одного или нескольких секторов. Например, в ОС Linux обычно размер блока по умолчанию равен 1Кбайт, то есть блок включает два сектора.

Все линейное дисковое пространство обычно делится на несколько частей – разделов (partitions). В один раздел объединяется группа смежных цилиндров, для каждого раздела следует хранить информацию о его начале и конце, т.е. номера первого и последнего из задействованных в нем цилиндров. Разделение всего дискового пространства на разделы полезно по нескольким причинам:

- уменьшаются «дальние перемещения» головок чтения/записи и увеличивается скорость выполнения операций чтения и записи;
- появляется возможность структурирования данных (например, можно отделить файлы пользователя от файлов ОС);
- на одном диске можно установить несколько операционных систем;
- на одном диске можно хранить информацию в разных файловых системах, или в одинаковых файловых системах, но с разным размером кластера.

Структура данных, хранящая информацию о логической организации диска вместе с небольшой программой, с помощью которой можно найти и загрузить в ОЗУ программу загрузки ОС, называется главной загрузочной записью (Master Boot Record, MBR), и располагается в самом первом секторе, т.е. в секторе с физическим адресом **0-0-1.**

В MBR расположена также таблица разбиения диска на разделы (*partition table*), которая содержит четыре записи по 16 байтов. Каждая запись может хранить информацию по одному разделу, который называется первичным. Таким образом число первичных разделов на диске ограничено, их не может быть более четырех. По мере увеличения объемов дисковой памяти стало ясно, что четырех разделов мало, и были предложены логические разделы**.** Для этого один из первичных разделов объявляется расширенным и в нем создаются логические разделы.

Расширенные разделы сами по себе не используются, они могут лишь хранить информацию о логических разделах. Первый сектор расширенного раздела хранит таблицу разделов с четырьмя записями: одна (первая) используется для логического

раздела, другая (вторая) для еще одного расширенного раздела, а две другие не используются. Таким образом, каждый расширенный раздел имеет свою таблицу разбиения, в которой используются только две записи, задающие один логический и один расширенный, то есть получается цепочка из таблиц разделов. Очевидно, что в последнем расширенном разделе из четырех записей таблицы разбиения используется только первая. На рис. 8 показан пример разбиения диска, имеющего три первичных и два логических раздела.

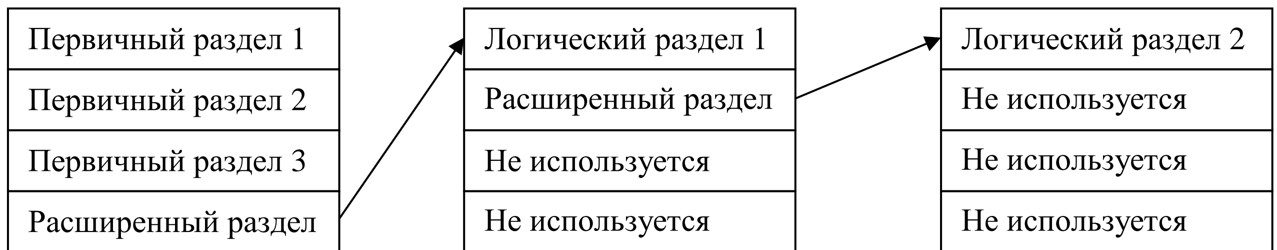


Рис.8

В семействе ОС Unix диски и их разделы представляются пользователю в виде файлов устройств, зарегистрированных в каталоге /dev**.** Имена этих файлов формируются по определенным правилам. Так, до недавнего времени наиболее распространенные IDE – диски с интерфейсом Parallel ATA, PATA именуются **hda**, **hdb** и т.д. Современные диски с интерфейсом Serial ATA (SATA) именуются **sda**, **sdb** и т.д.

Дисковые разделы идентифицируются порядковыми номерами. Цифры с 1 по 4 отведены под первичные разделы, а логические разделы нумеруются, начиная с цифры 5.

Пример. Если на SATA – диске имеются два первичных раздела, второй из которых определен как расширенный и поделен на три логических раздела, то соответствующие им устройства будут именоваться так:

/dev/sda1 – первичный раздел;

/dev/sda2 – первичный раздел, определенный в качестве расширенного;

/dev/sda5, /dev/sda6, /dev/sda7 – логические разделы.

Логические разделы могут создаваться не только через расширенный раздел, но и с помощью менеджера логических томов (LVM) – специального компонента Linux, реализованного с помощью подсистемы device mapper (dm). LVM позволяющий создавать виртуальные (логические) адресные пространстсва внешней памяти. Таким образом, например, можно один раздел физического диска, размеченного с помощью MBR, разделить на несколько логических разделов, управляемых LVM.

Дисковые разделы могут создаваться не только для разделения одной области диска, но и для объединения, в результате которого физические диски или их отдельные части видны опреационной системе как непрерывное дисковое пространство, на котором может быть создана единая файловая система. Существует два основных способа объединения – организация дисковых массивов и применение менеджера логических томов.

Технология дисковых массивов (RAID) используется для объединения нескольких дисков в один логический объект с целью избыточности или повышения производительности. Существует более 10 различных спецификаций RAID. Например, RAID уровня 1 предназначен для увеличения надежности хранения данных и реализуется путем зеркалирования (дублирования) дисков.

Менеджер логических томов позволяет использовать разные области одного физического жёсткого диска или области различных жёстких дисков как один логический том. В этом случае LVM позволяет:

- иметь файловую систему, которая превышает размер наибольшего диска;
- добавлять диски или разделы в дисковую группу и расширять существующие файловые системы «на лету»;
- заменить два жестких диска размером 250 ГБ одним диском на 500 ГБ без необходимости выключения компьютера для переноса системы или ручного перемещения данных между дисками;
- уменьшить размеры файловых систем и удалить диски из дисковой группы, когда их емкость больше не требуется;
- создавать резервные копии на основе мгновенных копий файловой системы.

В LVM используются понятия физического тома (physical volume – PV), группы томов (volume groups – VG) и логического тома (logical volume – LV). Физические тома являются физическими дисками или разделами физических дисков, группы томов объединяют физические тома. Группа томов может быть логически разделена на логические тома, которые для пользователей представляются физическими дисковыми разделами.

На рис. 9 показан принцип отображения физической структуры томов в логическую. Здесь 4 раздела физического диска hda и два физических диска (hdb и hdd)

отображаются в группу томов VG0, в которой создан логический том LV0 и оставлено свободное место для других логических томов или для последующего роста LV0.

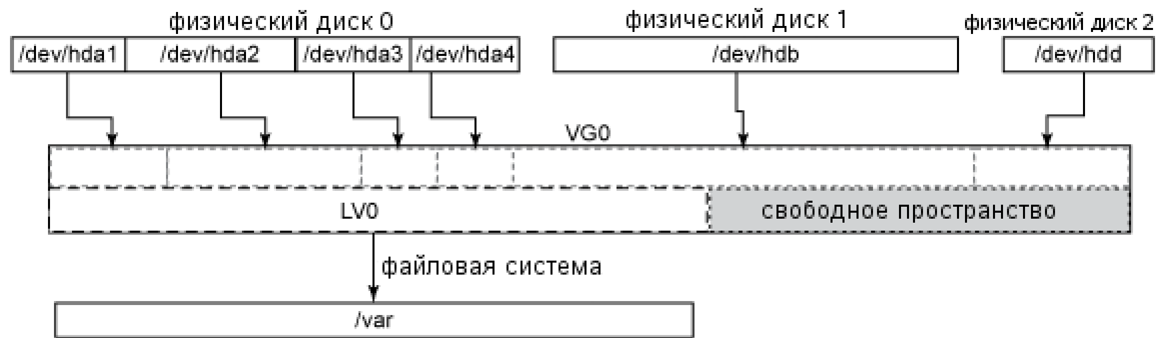


Рис. 9

Файлы блочных логических томов хранятся в каталоге /dev/mapper.

Каждый раздел дисковой памяти, включая логические тома, имеет собственную файловую систему и характеризуется следующими основными параметрами:

- имя, используемое ядром Linux, например dm-1;
- тип (первичный, расширенный, логический);
- тип установленной файловой системы;
- точка монтирования – имя каталога, к которому подключается раздел;
- имя устройства – имя, под которым раздел представлен пользователю, например centos-tmp;
- номер устройства, состоящий из двух частей: старший номер (major) идентифицирует драйвер, ассоциированный с устройством, а младший номер (minor) указывает номер устройства. Например, диск sda может идентифицироваться парой номеров 8:0, а разделы sda1 и sda2 – номерами 8:1 и 8:2 соответственно;
- тип устройства, например Virtual CD/ROM.

В последнее время вместо устаревшего разбиения на основе MBR начинает использоваться разбиение на разделы на основе таблицы GPT (GUID Partition Table, таблица с глобальными идентификаторами). Если диск имеет таблицу разбиения GPT, то на нем по умолчанию зарезервировано место под 128 разделов, каждый из которых является первичным.

2.2 Файловая система SYSTEM V

Файловая система SYSTEM V (s5) была одной из первых файловых систем, используемых в семействе ОС Unix. Система управляет всеми блоками раздела, разделяя его на две области – область метаданных и область данных. В области метаданных расположены три объекта – загрузочный блок, суперблок и индексные дескрипторы (рис. 10).

Загрузочный блок	Суперблок	Индексные дескрипторы (i-узлы)	Блоки данных (файлы и каталоги)

Рис 10.

Загрузочный блок используется для загрузки ОС и занимает первый блок файловой системы в каждом разделе. Однако активным является только один загрузочный блок, находящийся в корневой файловой системе.

Суперблок располагается непосредственно за загрузочным блоком и содержит самую общую информацию о файловой системе (общий размер, размер области индексных дескрипторов, их число, список свободных блоков и индексных дескрипторов и т. д.). При монтировании файловой системы в оперативной памяти создается копия ее суперблока. Все последующие операции по созданию и удалению файлов влекут изменения копии суперблока в оперативной памяти, эта копия периодически записывается на магнитный диск. Часто причиной повреждения файловой системы является отключение электропитания или зависание ОС в тот момент, когда система производит копирование суперблока из оперативной памяти на магнитный диск.

Область индексных дескрипторов содержит дескрипторы файлов (i - узлы), в англоязычной литературе называемые inode. С каждым файлом связан один i - узел, но одному i - узлу могут соответствовать несколько файлов. В i - узле размером 64 байта хранится вся информация о файле, кроме его имени. Область индексных дескрипторов имеет фиксированный формат и располагается непосредственно за суперблоком. Общее число дескрипторов и, следовательно, максимальное число файлов раздела, задается в момент создания файловой системы.

Дескрипторы нумеруются натуральными числами. Первый дескриптор используется ОС для описания специального файла, содержащего информацию о сбойных блоках раздела, которые рассматриваются ОС, как принадлежащие специальному файлу, и поэтому считаются занятыми. Второй дескриптор содержит описание корневого каталога файловой системы.

В области данных расположены как обычные файлы, так и файлы каталогов (в том числе корневой каталог). Специальные файлы представлены в файловой системе только записями в соответствующих каталогах и индексными дескрипторами специального формата, т. е. места в области памяти не занимают.

Повышение быстродействия файловых операций осуществлялось дроблением файловых систем на относительно независимые части, получившие название групп цилиндров (файловая система ufs в UNIX BSD) или групп блоков (файловая система ext2 в Linux). В результате этого появилась возможность размещения логически связанной информации в физически смежных областях диска, что минимизировало перемещение дисковых головок, способствуя тем самым скорости доступа к данным.

Расчленение файловых систем также способствовало росту их надежности, т.к. давало возможность в разных ее частях дублировать критически важную информацию – суперблок, утрата которого (например, из-за физического повреждения дисковой поверхности) ранее влекла за собой невозможность доступа к данным вообще.

2.3 Файловые системы ОС Linux

Файловые системы ОС Linux делятся на два типа – локальные и распределённые (сетевые). Локальные файловые системы могут располагаться во внешней памяти (ext2, ext3, ext4 и др.) или в оперативной памяти (псевдо-файловые системы). К последней группе относятся:

- /proc – используется в качестве интерфейса к структурам данных в ядре; большинство расположенных в ней файлов доступны только для чтения, но в некоторые файлы можно записывать данные, что позволяет изменить переменные ядра;
- /tmpfs – позволяет не записывать на физические диски временные файлы, которые формируются в оперативной памяти, а затем удаляются; поддерживает работу с виртуальной памятью и создается для каждого пользователя на время проведения сенса работы;
- /devfs – предназначена для управления устройствами;
- /sysfs – используется для получения информации о всех устройствах и драйверах.

Распределенные файловые системы предназначены для объединения на логическом уровне файловых систем отдельных компьютеров в единое целое. В Linux такой системой является nfs (Network File System).

Основной файловой системой ОС Linux на начальном этапе была система ext2, структура которой показана на рис. 11. Она позволяла хранить длинные имена файлов (до 255 символов) и обеспечить высокую производительность. В первом блоке файловой системы ext2 располагается загрузчик, а все остальное пространство делится на блоки равного размера (обычно 1 Кбайт). Блоки объединяются в группы, каждая из которых имеет суперблок, описатель группы, два битовых массива (для блоков и для i-узлов) и блоки для хранения данных.

Суперблок хранит информацию о количестве i-узлов и блоков данных в группе, о размере группы и т.д. Описатель группы содержит информацию о расположении битовых массивов, количестве свободных блоков и i-узлов в группе, а также о количестве каталогов в группе. Битовые массивы предназначены для учета свободных блоков и i-узлов, для хранения каждого массива выделяется один блок. При размере блока 1Кбайт размер группы равен 8192 блока и количество i-узлов равно 8192.

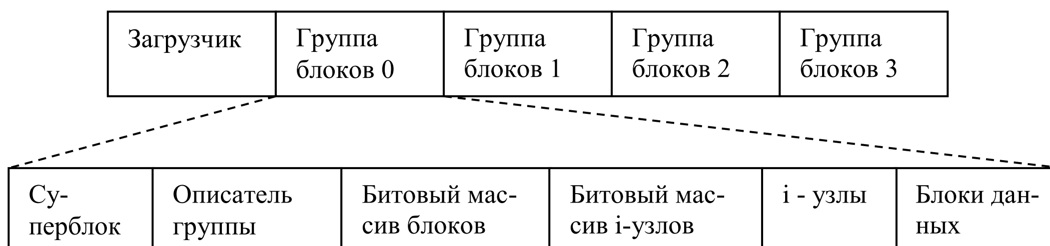


Рис. 11

Далее находятся i-узлы, структура которых показана в таблице 17. Размер каждого узла составляет 128 байт.

Таблица 17

Режим	Счетчик связей	UID	GID	Размер	Метки времени	Адреса блоков	Однократный косвенный блок	Двукратный косвенный блок	Трехкратный косвенный блок

Здесь обозначено:

- режим – тип файла, биты защиты;
- счетчик связей – число записей каталогов, указывающих на этот i-узел;
- UID – идентификатор владельца файла;
- GID – идентификатор группы владельца;

- размер – размер файла в байтах;
- метки времени – времена последнего доступа и последнего изменения файла, а также время последнего изменения i-узла;
- адреса блоков – адреса первых 12 блоков файла, размер адреса – 4 байта;
- однократный косвенный блок – адрес одинарного косвенного блока, который содержит адреса 256 дополнительных блоков файла;
- двукратный косвенный блок – содержит адреса 256 одинарных косвенных блоков, каждый из которых содержит адреса 256 блоков данных;
- трехкратный косвенный блок – содержит адреса 256 двукратных косвенных блоков.

Часть информации из i-узла можно посмотреть командой

stat ***имя_файла***

Достоинством файловой системы ext2 является ее быстродействие, а недостатком – отсутствие средств журналирования, что ухудшает надежность хранения данных. Этот недостаток исправлен в современных файловых системах ext3 и ext4. Кроме того, в ext4 адресация блоков увеличена до 6 байтов, а размер i-узлов – до 256 байтов, что позволило увеличить максимальный размер одного файла до 16 Тбайт.

Файловая система ext2 до сих пор активно используется на твердотельных накопителях, например на SSD-дисках, где ячейки памяти необратимо изнашиваются после определённого количества операций записи. Отсутствие журналирования дает возможность значительно уменьшить количество перезаписей одного и того же сектора диска и продлить срок службы устройства.

2.4. Команды анализа файловой системы

2.4.1 Команда **df**

Назначение: показывает список файловых систем на смонтированных устройствах, их размер, занятое и свободное пространство, а также точки монтирования.

Формат: df [-опции][имя_файла]

Примеры:

а) df -T – вывод списка файловых систем с указанием их типа;

б) df ~ – вывод информации по файловой системе, в которой находится домашний каталог.

Комментарий: для того, чтобы вместо информации об использовании блоков выводить информацию об использовании индексных дескрипторов, надо применить опцию –i.

2.4.2 Команда **du**

Назначение: показывает использование дисковой памяти файлами.

Формат: du [-опции][имя_файла]

Примеры:

а) du – вывод размера всех подкаталогов текущего каталога;

б) du -ah – вывод размера (в байтах) всех файлов и подкаталогов текущего каталога;

Комментарий: размер файлов по умолчанию выводится в блоках, для вывода в байтах надо применить опцию –h.

2.4.3 Команда **lsblk**

Назначение: показывает список блочных устройств.

Формат: lsblk [-опции][имя_устройства]

Примеры:

а) lsblk – вывод списка активных устройств;

б) lsblk -f – вывод списка активных устройств с указанием типа файловой системы;

в) lsblk –a – вывод списка всех устройств;

г) lsblk –p – вывод списка активных устройств с указанием путей доступа

д) lsblk –o KNAME,FSTYPE,SIZE,MODEL,MOUNTPOINT

Комментарий: опция –o дает возможность самостоятельного формирования списка вывода, например, при задании параметров KNAME, FSTYPE, SIZE, MODEL, MOUNTPOINT будут выводиться имя устройства, тип его файловой системы, размер, модель и точка монтирования.

2.4.4 Команда **cat /proc/partitions**

Назначение: вывод файла, который содержит общую информацию о разделах файловой системы.

Формат: cat /proc/partitions

Комментарий: по каждому разделу выводится имя, размер и номер устройства, на котором расположен раздел.

2.4.5 Команда **cat /etc/fstab**

Назначение: вывод файла, который содержит общую информацию об используемых файловых системах.

Формат: cat /etc/fstab

Комментарий: выводится список смонтированных файловых систем.

2.4.6 Команда **ls /dev/mapper**

Назначение: отображение каталога, в котором содержатся файлы блочных логических томов.

Формат: ls /dev/mapper

Комментарий: логические тома используются при установке в Linux системы LVM, предназначенной для создания абстрактной логической внешней памяти вместо физического управления дисками.

2.5. Команды управления процессами

Для выполнения любой программы ОС создает процесс, имеющий уникальный идентификатор (process ID, PID). Процессы могут запускаться как с терминала пользователя, так и из другого (родительского) процесса. Пользователь Linux всегда может получать информацию о работающих процессах и управлять их выполнением.

На рис.12 приведены возможные состояния процессов и диаграмма переходов из одного состояния в другое. В состоянии порождения для задачи создаются системные структуры данных и выделяются все требуемые ресурсы за исключением процессора, после чего она регистрируется в очереди диспетчера и переходит в состояние готовности, ожидая выделения процессорного времени.

Когда диспетчер выделяет процессу процессорное время, он переходит в состояние выполнения и называется активным, а переход из состояния готовности в состояние выполнение называется запуском.

Из состояния выполнения возможны три перехода:

- в состояние завершения, если процесс успел завершиться в течение кванта, при этом ОС освобождает все выделенные ей ресурсы;
- в состояние готовности, если процесс не успел завершиться в течение кванта;

- в состояние блокирования, если процессу во время выполнения требуется подождать наступления какого-либо события, например, завершения чтения очередного блока данных из внешней памяти, освобождения какого-либо ресурса или завершения чтения нужной страницы памяти из файла подкачки виртуальной памяти. При наступления ожидаемого события процесс из состояния блокирования переходит в состояние готовности, этот переход называется пробуждением.

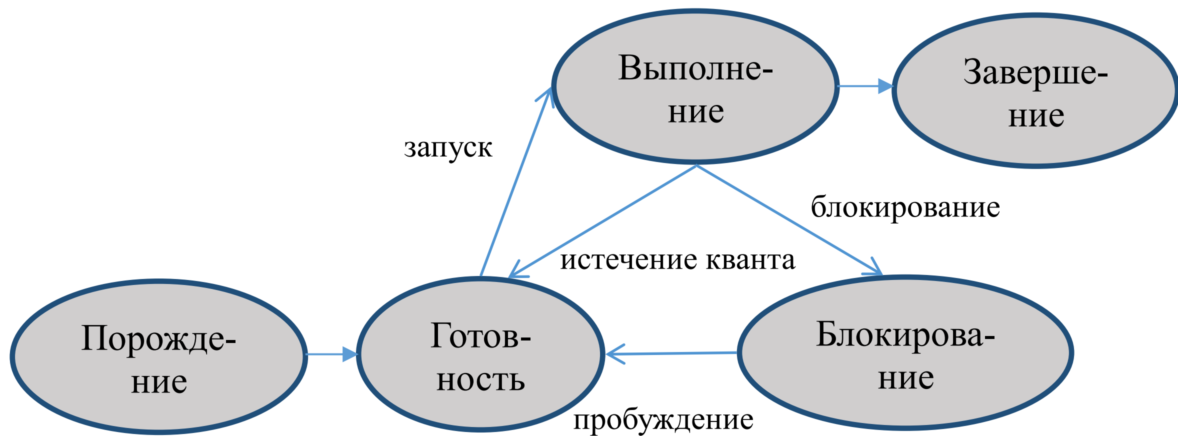


Рис. 12

Возможны ситуации, когда какая-то программа зависает, т.е. во время работы перестает реагировать на действия пользователя или на сообщения от других программ. В этом случае дочерний процесс, связанный с этой программой, не может сообщить своему родительскому процессу о своем завершении, а запись о нем остается в списке процессов ОС. Аналогичная ситуация возникает в случае, если некоторое приложение запустило дочерний процесс, а затем было уничтожено или неожиданно завершено, оставив после себя зависший дочрний процесс. Такое состояние дочернего процесса называется *зомби*.

2.5.1 Команда **ps**

Назначение: вывод информации о процессах.

Формат: ps [опции]

Комментарий: вывод команды содержит информацию, представленную в таблице 18, где состояние процесса обозначается следующим образом:

R – процесс в режиме выполнения;

D – процесс находится в ожидании дисковой операции;

I – процесс в режиме ожидания более 20 секунд;

S – процесс в режиме ожидания менее 20 секунд;

T – процесс остановлен;

Z – мертвый (зомби) процесс.

Таблица 18

Название столбцов	Значение
USER	Идентификатор (имя) пользователя, запустившего процесс
PID	Идентификатор процесса
%CPU	Интенсивность использования процессора
%MEM	Интенсивность использования памяти
STAT	Состояние процесса
STIME	Время старта процесса
TTY	Терминал, с которого был запущен процесс
TIME	Использованное время процессора
COMAND	Выполняемая команда

Примеры:

а) ps – вывод краткой информации о процессах, запущенных пользователем;

б) ps -ux – вывод подробной информации о процессах, запущенных пользователем;

в) ps -e – вывод краткой информации о всех процессах, запущенных в системе;

а) ps -ef – вывод подробной информации о всех процессах, запущенных в системе;

д) ps -a – вывод краткой информации о всех процессах, запущенных пользователями;

На рисунке 13 приведен пример вывода команды **ps**, из которого видно, что пользователь kvg открыл два сеанса работы с сервером с терминалов pts/0 и pts/1. В первом сеансе открыты интерпретатор **bash** и два экземпляра команды **top**, во втором сеансе работают второй экземпляр интерпретатора **bash** и команда **ps**.

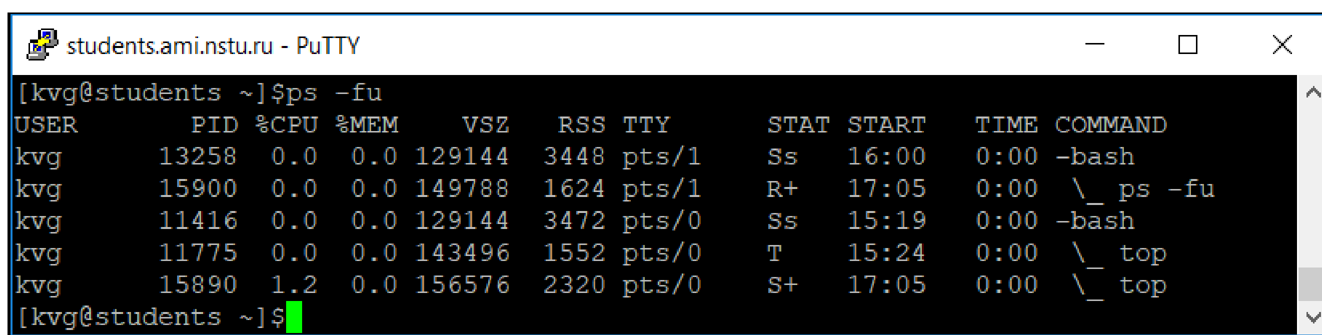


Рис. 13

2.5.2 Команда **top**

Назначение: периодический мониторинг состояния активных процессов.

Формат: top [опции]

Команда выводит на экран информацию по общей загрузке системы и по всем запущенным процессам, обновляя данные с определенным периодом времени (по

умолчанию 3 сек.). Она часто используется системными администраторами для диагностики серверов в случае возникновения каких-либо проблем.

Основные опции программы:

-d число – задает период обновления данных о процессах, сек;

-n число –максимальное число периодов обновления данных, выводимых программой;

-u имя_пользователя – отображать процессы указанного пользователя;

Завершение команды выполняется нажатием Ctrl+C, временная приостановка мониторинга – нажатием Ctrl+Z. Для продолжения работы приостановленной программы необходимо ввести команду **fg**;

Пример:

top -u user_1 -n 5 – включить мониторинг процессов, запущенных пользователем user_1 с периодом 3 сек, вывести 5 итераций.

Вывод команды состоит из двух частей – заголовка, в который выводится общая информация о загрузке системы, и списка процессов (рис. 14).

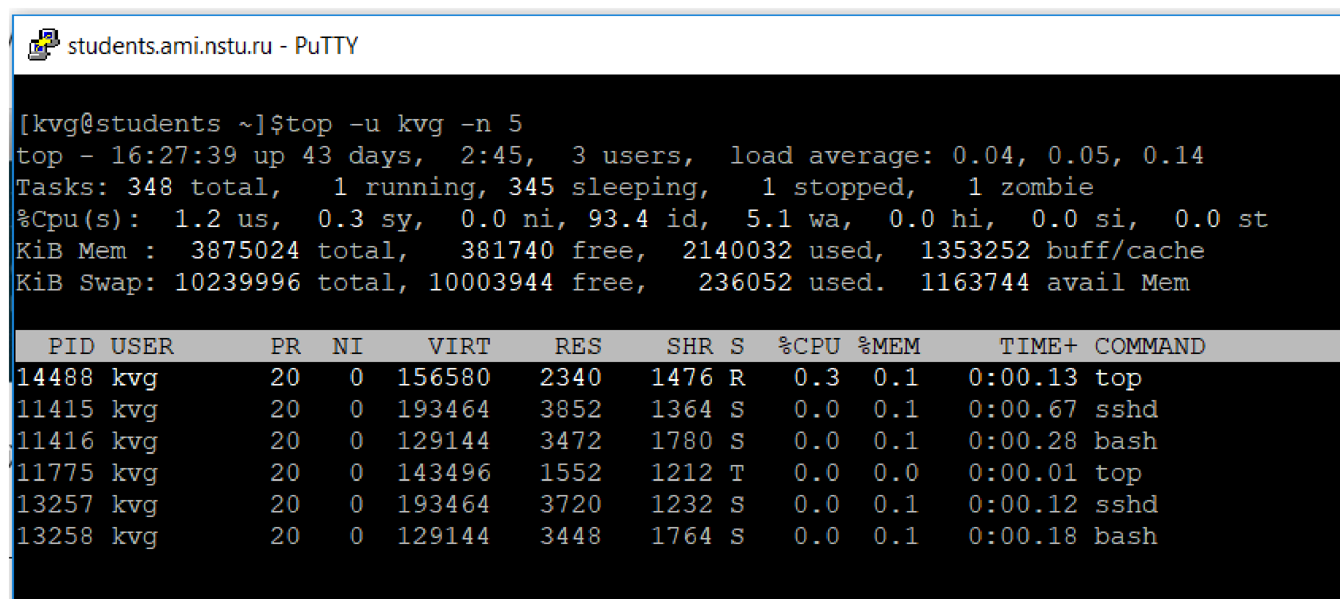


Рис. 14

В первой строке заголовка выводится текущее системное время, общее время работы системы, число подключенных пользователей и средняя загрузка в течение последних 1 минуты, 5 минут и 15 минут. Вторая строка содержит данные об общем числе процессов в системе (total), а также о числе процессов, находящихся в состоянии выполнения (running), блокирования или готовности (sleeping), завершения (stopped) и зомби (zombie). Остальные строки заголовка содержат информацию о загрузке процессора, оперативной памяти и виртуальной памяти.

Список процессов по умолчанию сортируется по степени загрузки ЦП и содержит следующие поля:

- PID - идентификатор процесса;
- USER - имя пользователя, который является владельцем процесса;
- PR - приоритет процесса;
- NI - значение "NICE", влияющие на приоритет процесса;
- VIRT - объем виртуальной памяти, используемый процессом;
- RES - объем физической памяти, используемый процессом;
- SHR - объем разделяемой памяти процесса;
- S - указывает на статус процесса: S (блокирован), R (работает), Z (зомби);
- %CPU - процент использования центрального процессора данным процессом;
- %MEM - процент использования оперативной памяти данным процессом;
- TIME+ - общее время использования процессора;
- COMMAND – имя процесса.

2.5.3 Команда **kill**

Назначение: формирование сигнала для уничтожения процесса.

Формат: kill [опции] PID

Примеры:

а) kill –l – вывод списка всех доступных сигналов;

б) kill –9 521 – принудительно завершить процесс с идентификатором 521.

Комментарий: команда используется в случае некорректной работы или зависания процесса для формирования сигнала уничтожения процесса с указанным идентификатором PID. Наиболее часто используются сигналы SIGTERM (15), который генерируется по умолчанию, и SIGKILL (9), который используется в случае неудачи с сигналом SIGTERM.

3. Порядок выполнения работы

1. Осуществите вход в систему, используя соответствующее имя пользователя.
2. Создайте в домашнем каталоге нижеперечисленные объекты файловой системы, где kk – номер Вашей бригады, и задайте им указанные права доступа:

drwxr--r-- ... australia_kk

drwx--x--x ... play_kk

-r-xr--r-- ... my_os_kk

-rw-rw-r-- ... feathers_kk

3. Проделайте приведенные ниже упражнения, записывая в отчет используемые при этом команды:

3.1. Просмотрите содержимое файла /etc/passwd с использованием команды постраничного просмотра. Сколько пользователей имеют учетные записи на сервере? Приведите в отчете структуру учетной записи.

3.2. Скопируйте файл ./feathers_kk в файл ./file.old

3.3. Переместите файл ./file.old в каталог ./play_kk

3.4. Скопируйте каталог ./play_kk в каталог ./australia_kk .

3.5. Лишите владельца файла ./feathers_kk права на чтение.

3.6. Что произойдет, если вы попытаетесь просмотреть файл ./feathers_kk командой **cat**?

3.7. Что произойдет, если вы попытаетесь скопировать файл ./feathers_kk?

3.8. Дайте владельцу файла ./feathers_kk право на чтение и выполните п. 3.6

3.9. Лишите владельца каталога ./play_kk права на выполнение.

3.10. Перейдите в каталог ./play_kk. Что произошло?

3.11. Дайте владельцу каталога ./play_kk право на выполнение и выполните п. 3.10.

4. Откройте второе соединение с сервером, в котором командой **top** включите мониторинг Ваших процессов и определите:

- число подключенных к системе пользователей,
- общее количество процессов в системе и их состояние,
- количество Ваших процессов,
- загрузку процессора и памяти.

Дальнейшие действия выполняйте в первом соединении, а во втором соединении фиксируйте соответствущие изменения. Команду **top** использовать в режиме фильтрации по имени пользователя.

5. Создайте в файле **loop** следующий сценарий, реализующий бесконечный цикл и запустите его в фоновом режиме командой **./loop &**

```
while true
do
```

```
    true
  done
```

6. С помощью команды **ps -efu** посмотрите список Ваших активных процессов и занесите его в отчет. Посмотрите изменения в результатах, выводимых командой **top** в втором окне, занесите их в отчет и поясните результаты.

7. В основном окне выполните следующие действия:

- запустите программу **mc** и отключите вывод на экран ее окон;
- в командной строке mc повторно запустите в фоновом режиме сценарий **loop** командой **sh;**
- посмотрите список Ваших активных процессов и сравните результаты с полученными в п.6; занесите в отчет идентификаторы и имена новых процессов.
- посмотрите изменения в результатах, выводимых командой **top** в втором окне;
- выполните принудительное прерывание всех процессов, запущенных в п.5 и п.7 и убедитесь, что все процессы уничтожены;
- в отчете поясните полученные результаты.

8. Посмотрите с помощью команды **stat** и занесите в отчет информацию из индексного дескриптора файла ~/.bash_history.

9. С помощью команд **lsblk** и **df** определите основные характеристики разделов внешней памяти сервера (имя и номер устройства, имя и тип раздела, размер, тип файловой системы, коэффициент использования памяти). Результаты представьте в виде следующей таблицы:

№ п/п	Имя устройства	Имя раздела	Тип раздела	Размер раздела	Тип ФС	Номер драйвера устройства	Коэф-т использования
1	sda	sda1	первичный	500M	xfs	8:1	64%

10. С помощью команд **df** и **du** определите типы файловых систем, используемых на сервере, а также в каком из имеющихся разделов расположен ваш домашний каталог и размер домашнего каталога. Поясните назначение каждой из файловых систем. Сравните размер домашнего каталога, полученный из команды **du** и аналогичный размер, полученный в MC при выполнении предыдущей лабораторной работы.

11. Посмотрите и занесите в отчет содержимое файлов /proc/partitions и /etc/fstab, сопоставьте их с результатами, полученными в п. 9 и п. 10.

12. Прервите во втором окне выполнение команды top и закройте оба соединения.

4. Контрольные вопросы

1. Дайте характеристику физической модели диска.
2. Дайте характеристику логической модели диска.
3. Дайте характеристику каждой файловой системе, существующей в компьютере, на котором Вы выполняли лабораторную работу.
4. Приведите общую структуру файловой системы ext2 и дайте характеристику каждому элементу этой структуры.
5. Опишите действия системы при обращении к некоторому файлу с запросом на чтение.
6. Как определить объем свободной памяти на жестком диске и объем вашего домашнего каталога?
7. Каким образом вы можете получить информацию о процессах в системе?
8. Как удалить “зависший“ процесс?
9. Что такое MBR ?
10. Что такое первичный раздел? Какое максимальное число первичных разделов может быть на диске? Каким образом нумеруются первичные разделы?
11. Что такое расширенный и логический разделы, как они нумеруются?
12. Будет ли информация о расширенном разделе в распечатке после выполнения команды **df** ? Если нет, то почему?
13. Поясните принцип работы LVM.
14. Назовите средства мониторинга процессов в Linux и их возможности.

Лабораторная работа № 5. Файловые системы ОС Windows

1. Цель работы

Целью работы является приобретение навыков анализа физической и логической структуры магнитных дисков и закрепление знаний по файловым системам FAT и NTFS.

2. Методические указания

На физическом уровне работа с устройством внешней памяти всегда проводится через соответствующий контроллер. Именно контроллер знает физические параметры устройства, поэтому физическая модель магнитных дисков не зависит от операционной системы (Linux или Windows) .

Диск Windows, также как и Linux, может быть разбит на несколько разделов, которые могут быть первичными, расширенными или логическими. Параметры разделов хранятся в таблице разделов диска (*partition table*). В отличие от Linux, разделы диска называются логическими дисками и обозначаются латинскими буквами, за которыми следует двоеточие – A:, B:, C:, D: и т.д. Файловые системы каждого раздела, в отличие от Linux, не связаны между собой и функционируют отдельно друг от друга.

Небольшая часть адресного пространства каждого раздела выделяется под системную область, основная часть – под область данных. Единицей дисковой памяти в области данных является кластер, размер которого кратен размеру сектора и может достигать 64 Кбайт. Все кластеры имеют сквозную нумерацию, причем первый допустимый номер кластера равен 2. Файлы на диске могут храниться в несмежных кластерах, т.е. в различных частях диска.

Файл Windows - это поименованная совокупность информации, хранящаяся на ВЗУ. В виде файлов на диске хранятся программы и данные. Каталог – файл специального формата, предназначенный для хранения метаданных о зарегистрированных в этом каталоге файлах и подкаталогах (имя, расширение, атрибуты, размер, дата и время создания или последнего изменения, адрес). Каталоги каждого логического диска организованы в единую древовидную структуру. Имена файлов и каталогов, в отличие от Linux, регистронезависимы.

Структура системной области диска зависит от типа файловой системы, которая определяет способы доступа к информации в области данных. Наиболее известными файловыми системами для Windows являются FAT и NTFS.

2.1 Файловая система FAT

На рис. 15 приведена логическая модель диска с файловой системой FAT. В системной области находятся загрузочная запись, таблица размещения файлов и корневой каталог. Загрузочная запись, иногда называемая начальным загрузчиком, имеет размер 512 байт, всегда хранится в нулевом секторе и используется в процессе загрузки операционной системы.

Загрузочная запись	Таблица размещения файлов (2 копии)	Корневой каталог	Кластеры данных (файлы и каталоги)

Рис. 15

Таблица размещения файлов, в оригинальной литературе называемая FAT (File Allocation Table), содержит информацию о размещении файлов в области данных. Она всегда занимает сектора, начиная с первого. На любом диске для обеспечения надежного доступа к данным всегда хранится две копии FAT, которые обновляются одновременно.

Корневой каталог – главный каталог диска, который занимает сектора, следующие за FAT. Фиксированное число элементов и размещение в системной области корневого каталога являются принципиальным отличием от прочих каталогов.

2.1.1 Таблица размещения файлов

Таблица размещения файлов содержит информацию о номерах кластеров, выделенных для хранения каждого файла. Она представляет собой карту (образ) области данных, в которой описывается состояние каждого кластера диска. Размер таблицы зависит от объема диска. Номер начального кластера, выделенного файлу, записывается в элемент каталога этого файла.

Каждый элемент таблицы соответствует одному кластеру в области данных. Дефектные кластеры помечаются как "bad". Если кластер свободен, то соответствующий ему элемент FAT имеет значение "0". Если кластер выделен для какого-либо файла, то возможны два варианта:

а) элемент содержит признак конца файла "EOF", если этот кластер является последним кластером, выделенным файлу;

б) элемент содержит значение номера следующего кластера, выделенного файлу.

Пример. Файл Ist.dat размером 14150 байтов занимает на диске кластеры 3, 4, 5 и 6. Приведем элемент каталога, в котором зарегистрирован этот файл, и фрагмент FAT, а также связь между ними:

элемент каталога:

Имя	Тип	Атрибут	Резерв	Время созд.	Дата созд.	Номер нач. кла-стера	Размер файла
IST	dat	r		12-50	01.10.06	3	14150

фрагмент FAT:

2	3	4	5	6	7	8	9	10	11
EOF	4	5	6	EOF	EOF	0	0	0	0

Таким образом элементы FAT, выделенные одному файлу, связываются в цепочки, позволяющие получить доступ к информации даже в том случае, если файл при записи разбивается на несколько фрагментов, хранящихся в несмежных кластерах диска. Элементы FAT могут быть 16- и 32-разрядными, в зависимости от этого файловые системы имеют названия FAT-16 и FAT-32 и работают с 16-разрядными и 32-разрядными дисковыми адресами соответственно.

Логическое разбиение области данных на кластеры, как совокупность секторов, взамен использования одиночных секторов имеет следующий смысл:

- уменьшается размер FAT;
- уменьшается возможная фрагментация файлов;
- ускоряется доступ к файлу, т.к. в несколько раз сокращается длина цепочек фрагментов дискового пространства, выделенных для него.

Следует иметь ввиду, что при увеличении размера кластера ухудшается коэффициент использования дисковой памяти за счет увеличения внутренней фрагментации. Минимальный размер кластера на диске с файловой системой FAT зависит об объема диска ($V_{диска}$) и разрядности элемента FAT (r):

$$V_{кл_min} = V_{диска} / 2^{r}$$

2.1.2 Корневой каталог диска

Записи корневого каталога имеют длину 32 байта, структура записей представлена в таблице 19. Если файл не имеет расширения, то в соответствующем поле хранятся пробелы. Дата и время используются в виде четырехбайтового значения в операциях сравнения. Номер начального кластера определяет точку входа в FAT для данного файла и одновременно дисковый адрес собственно файла.

Таблица 19

Номер поля	Длина (байт)	Назначение поля
1	8	имя файла
2	3	расширение имени
3	1	атрибуты
4	10	резерв
5	2	время создания/модификации
6	2	дата создания/модификации
7	2	номер начального кластера
8	4	размер файла

Байт атрибутов файла задает его статус в соответствии с табл. 20. Если байт атрибута равен 8, то метка тома хранится в полях имени и расширения файла элемента корневого каталога. Если элемент каталога указывает на подкаталог, то используются все поля элемента каталога, при этом последнее поле (размер файла) имеет нулевое значение.

Таблица 20

Значение байта атрибутов	Характеристика файла
1	только чтение
2	скрытый файл
4	системный файл
8	элемент каталога хранит метку тома (11 байтов)
16	элемент каталога указывает на подкаталог

2.1.3 Каталог нижнего уровня

Все подкаталоги имеют структуру, аналогичную корневому каталогу, только в отличие от него они не имеют фиксированного размера и фиксированного дискового адреса, т.е. хранятся на диске как обычные файлы. Для того, чтобы в процессе работы существовала возможность перемещения по дереву каталогов как вниз, так и вверх, в подкаталогах предусмотрено существование двух элементов с именами "." и "..". Первый элемент каталога является указателем на данный каталог, а второй – на родительский каталог. Соответственно поле «Номер начального кластера» этих элементов содержит дисковые адреса этих каталогов.

Если это поле содержит нуль, то это означает, что каталог-родитель является корневым каталогом диска.

2.1.4 Хранение длинных имен файлов

Классическая файловая система FAT может хранить файлы с именами до 8 символов, что следует из таблицы 19.. Современная версия файловой системы FAT может хранить файлы с именами до 255 символов и называется VFAT. При этом для каждого файла и подкаталога хранятся два имени – длинное (до 255 символов) и короткое (в формате 8.3).

Хранение длинных имен файлов организуется в специально отформатированных записях каталога, у которых байт атрибутов равен **0F**. Экспериментальным путем было определено, что такие записи каталога не видны для старых программ, работающих только с короткими именами. Структура записи каталога для хранения длинных имен приведена в таблице 21, откуда видно, что одна запись может хранить до 13 символов в кодировке Unicode.

Для регистрации файла с длинным именем в каталоге выделяется необходимое количество специальных записей, а также одна стандартная запись для хранения короткого имени. Блок специальных записей всегда располагается в каталоге перед стандартной записью, поэтому если к каталогу обращается 16-разрядная DOS программа, то она будет видеть только короткое имя файла, а 32-разрядные Windows-приложения могут работать с длинными именами.

Таблица 21

Номер поля	Длина (байт)	Назначение поля
1	1	порядок следования
2	10	первые 5 символов имени
3	1	Атрибуты (0F)
4	1	Указатель типа
5	1	Контрольная сумма
6	12	Следующие 6 символов имени
7	2	номер начального кластера (всегда 0)
8	4	последние 2 символа в имени

Короткое имя образуется из длинного следующим образом: оставляется 6 символов длинного имени, к которому дописываются знак “~” (тильда) и порядковый номер в пределах каталога для файлов, у которых первые 6 символов имени совпадают. Например, для регистрации файла с именем «Курсовой проект.doc» в каталоге будут выделены две специальные записи и одна стандартная, которая будет хранить имя “Курсов~1”.

2.1.5 Удаление файлов

При удалении файла обычно выполняются следующие действия:

- в таблице размещения файлов обнуляются все элементы, выделенные для описания этого файла;
- в соответствующем элементе каталога изменяется имя файла – вместо первого символа в поле имени записывается символ «x».

Остальные характеристики файла в элементе каталога, а также содержимое файла в кластерах диска, не изменяются, поэтому всегда есть возможность полностью или частично восстановить удаленный файл. Полное восстановление возможно, если:

- не перезаписан соответствующий элемент каталога;
- имеется доступ к каталогу;
- кластеры, ранее занимаемые файлом, не выделены другим файлам или каталогам;
- удаленный файл был нефрагментированным.

При несоблюдении последнего условия полное восстановление не гарантируется, т.к. не всегда возможно извлечь данные о том, какие кластеры были выделены файлу.

Современные операционные системы обычно проводят удаление файлов в специальный скрытый каталог, который называется корзиной. Размер корзины может устанавливаться пользователем. Корзина обслуживается специальной программой, что делает восстановление ошибочно удаленных файлов удобным и быстрым.

2.2 Файловая система NTFS

2.2.1 Организация раздела NTFS

Каждый раздел NTFS организован в виде последовательности кластеров размером до 64 Кбайт (по умолчанию обычно размер кластера равен 4 Кбайт) и содержит каталоги, файлы, битовые массивы и другие структуры данных. Основным отличием NTFS от файловой системы FAT является хранение основных системных структур данных в виде обычных файлов. Например, таким образом хранятся корневой каталог, битовый массив использованных блоков, определения атрибутов файлов и т.д. Имена системных файлов начинаются с символа “$”.

На рис. 16 показана организация раздела NTFS. В первом блоке раздела находится загрузочная запись ($Boot), в которой содержится программа загрузки и информация о разделе (тип файловой системы и адреса основных системных файлов). Загрузочная запись занимает обычно 8 КБ (16 первых секторов).

Рис. 16

Основной структурой данных в NTFS является главная таблица файлов (Master File Table, MFT), которая хранится в файле $MFT и представляет собой главный каталог, в котором регистрируются все файлы раздела, включая системные файлы. Для MFT резервируется 12% от общего объема раздела в виде непрерывной последовательности блоков, которая называется MFT-зоной. Запись файлов и каталогов в эту зону не проводится, а ее адрес хранится в загрузочной записи.

MFT состоит из множества записей размером 1 Кбайт о файлах, расположенных на томе. В записи MFT хранится вся информация о файле (имя, дата и время создания, размер, положение на диске отдельных фрагментов, и т.д). Если не хватает одной записи MFT, то используются несколько, причем не обязательно подряд. При этом первая запись называется базовой. Каждая запись MFT имеет уникальный номер – индекс, общее количество записей – до 2^{48}.

Структура файла $MFT показана на рис. 17. Первые 16 записей выделены для хранения информации о системных файлах. Самая первая запись в MFT – это запись о самом файле $ MFT. Во второй записи содержится информация о зеркальной копии MFT (файл $MFTMirr), в которой дублируются первые 4 записи таблицы MFT. В случае возникновения сбоя, если MFT окажется недоступным, информация о системных файлах будет считываться из файла $MFTMirr, адрес которого также имеется в загрузочной записи.

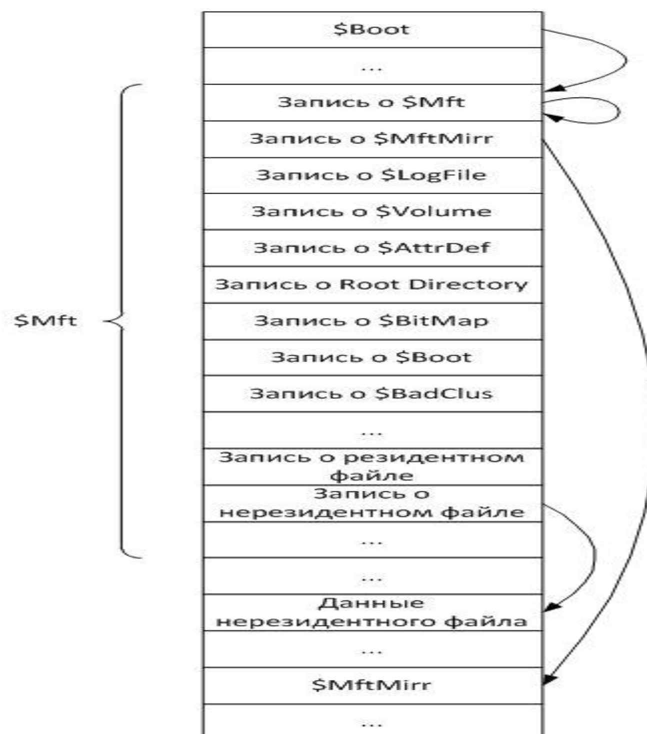


Рис. 17

Ниже приведено назначение некоторых системных файлов NTFS:

- $LogFile – файл журнала, в котором записывается информация о всех операциях, изменяющих структуру раздела NTFS, например, создание файлов и каталогов. Файл журнала используется при восстановлении тома NTFS после сбоев;
- $Volume – файл информации о томе, в котором содержатся имя тома (Volume label), версия NTFS и набор флагов состояния тома, например, флаг, установка которого означает, что том был поврежден и требует восстановления при помощи системной утилиты Chkdsk;
- $AttrDef – таблица определения атрибутов, содержащая возможные на данном томе типы атрибутов файлов;
- Root Directory – файл с информацией о корневом каталоге тома. В нем хранятся ссылки на файлы и каталоги, содержащиеся в корневом каталоге;
- $BitMap – файл битовой карты, каждый бит в которой соответствует одному кластеру: единичное значение бита соответствует занятому кластеру, нулевое – свободному;
- $Boot – файл загрузочной записи тома;
- $BadClus – файл плохих кластеров, содержащий информацию обо всех кластерах, имеющих сбойные секторы.

2.2.2 Структура записи MFT

Файловая запись MFT, структура которой приведена на рис. 18, всегда располагается в начале сектора; ее первые байты кодируют слово "FILE" (ASCII-коды 46 49 4C 45), а конец записи определяется 4-байтовой последовательностью FF FF FF FF. Запись состоит из заголовка и набора атрибутов. В заголовке содержится служебная информация о записи (например, её тип и размер), а все данные, относящиеся непосредственно к файлу, хранятся в виде атрибутов.

Каждый атрибут имеет заголовок, определяющий тип атрибута и его свойства, и тело, содержащее основную информацию атрибута. Названия атрибутов, как и системных файлов, начинаются с "$", например, имя файла ($FILE NAME), информация о его свойствах ($STANDARD_INFORMATION), данные файла ($DATA). Физически атрибут файла хранится в виде простой последовательности байтов.

Рис. 18

По расположению относительно MFT атрибуты бывают резидентные и нерезидентные. Резидентные атрибуты полностью помещаются в файловую запись MFT, нерезидентные атрибуты хранятся вне MFT. Область, в которой расположен нерезидентный атрибут, называется группой. Поскольку нерезидентных атрибутов в файле может быть несколько, то и групп бывает тоже несколько. Множество групп файла называется списком групп (RunList). Файловая запись при наличии нерезидентных атрибутов содержит ссылку на расположение группы на диске (см. рис. 17).

Некоторые поля заголовка файловой записи, а также резидентных и нерезидентных атрибутов представлены на рис.19. На том же рисунке справа показан

пример файловой записи с конкретными значениями рассматриваемых полей. Числа слева от полей записи обозначают шестнадцатеричное смещение поля от начала записи.

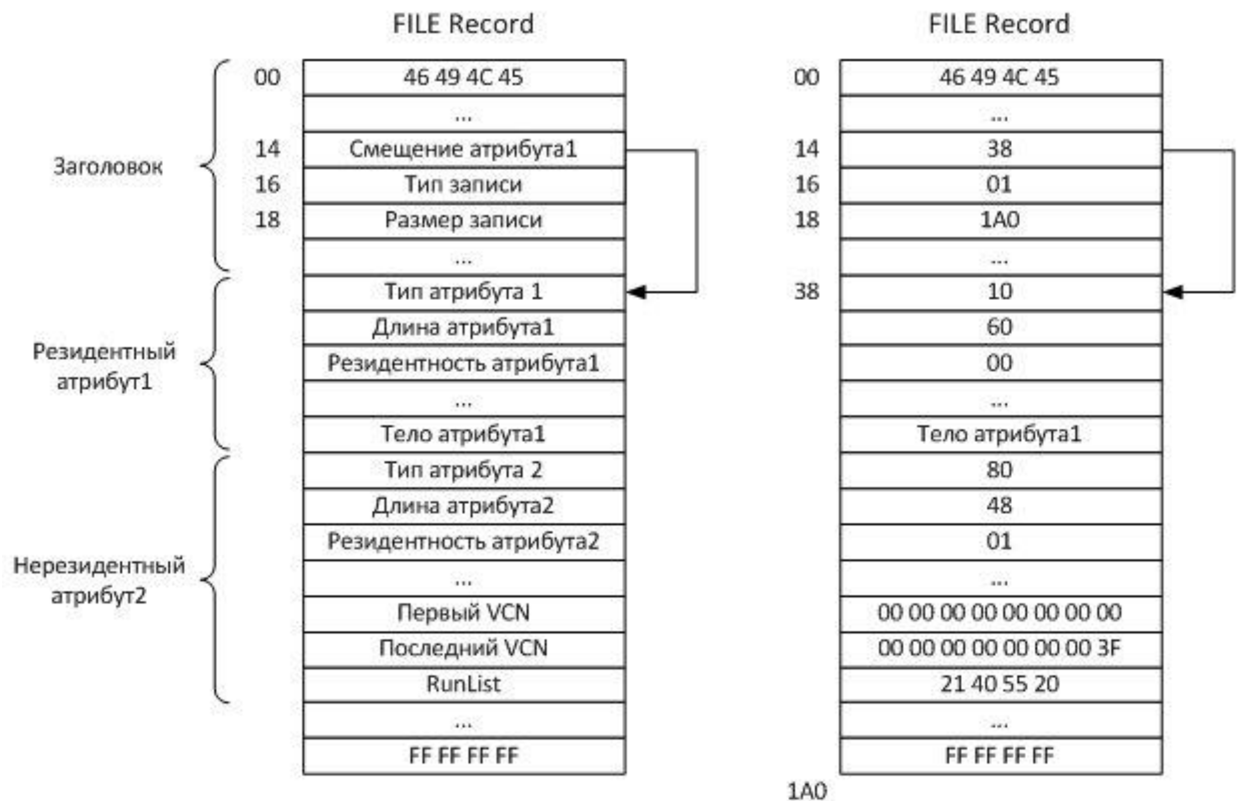


Рис. 19

В начале файловой записи находится признак её начала – слово "FILE" (46 49 4C 45). По смещению 0x14 расположено двухбайтовое поле, в котором записано смещение первого атрибута относительно начала файловой записи. В примере в этом поле записано 38, т. е. первый атрибут расположен по смещению 38.

В следующем поле хранится тип файловой записи: значение 01 обозначает файл, 02 – каталог (directory). В примере файловая запись соответствует файлу (значение 01 по смещению 16). Ещё одно поле в заголовке содержит размер всей записи. В примере на рис. 15 в этом поле записано 1A0, т. е. размер записи составляет 416 байт.

Каждый атрибут имеет поля, указывающие тип, длину и резидентность атрибута. Все типы атрибутов имеют свои численные значения, например, атрибуту $FILE_NAME соответствует значение 0x30, атрибуту $STANDARD_ INFORMATION – 0x10, атрибуту $DATA – 0x80.

Если атрибут резидентный, то в поле резидентности записывается 0x00, иначе – 0x01. В случае нерезидентного атрибута предусмотрены поля для хранения номеров кластеров, в которых располагается группа или несколько групп, выделенных для размещения файла.

В примере на рис 19 показаны два атрибута. Первый атрибут имеет тип $STANDARD_INFORMATION (значение 10), длина атрибута 96 байт ($60_{16} = 96$), атрибут является резидентным (00). У второго атрибута тип $DATA (80), длина – 72 байта ($48_{16} = 72$), атрибут является нерезидентным (01).

Для нумерации кластеров используются два типа номеров – логический номер кластера (LCN) и виртуальный номер кластера (VCN). Логический номер является номером кластера в пределах всего диска и используется для поиска начального кластера группы. Виртуальный номер обозначает порядковый номер кластера внутри группы.

В случае нерезидентных атрибутов в заголовке атрибута содержатся следующие поля: номер *VCN* первого кластера группы (обычно равен 0x00), номер *VCN* последнего кластера группы и список групп (RunList), описывающий расположение групп на диске.

Рассмотрим пример описания расположения групп, приведенный на рис.16 справа. В этом примере значения полей следующие:

- первый VCN = 0x00;
- последний VCN = 0x3F;
- список групп (RunList) = 0x21 40 55 20 00.

Расположение кластеров для данного примера приведено на рис. 20.

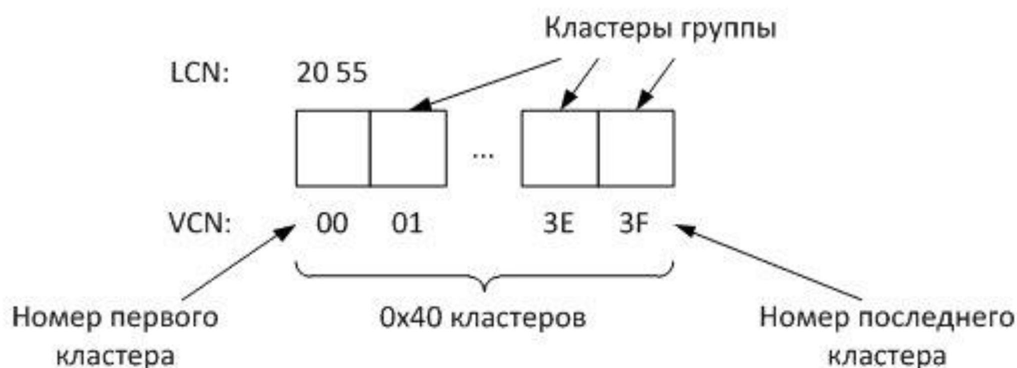


Рис. 20

В этом примере значение для списка групп 0x21 40 55 20 00 обозначает следующее:

- 0x21 – первый байт кодирует размер двух полей, которые за ним следуют:

  - младший полубайт обозначает размер поля (в байтах), в котором хранится длина группы в кластерах; в данном случае значение 1 указывает, что на длину группы отводится один байт;

  - старший полубайт обозначает размер поля (в байтах), в котором расположен номер LCN первого кластера группы; в данном случае значение 2 указывает на двухбайтовое поле;

- 0x40 – длина группы. Поскольку в первом байте размер поля длины группы определен в один байт, в качестве длины группы рассматриваем однобайтовое поле; в данном примере оно равно 0x40 (64 кластера);

- 0x2055 – LCN номер первого кластера. В первом байте размер поля номер первого кластера определен в два байта, поэтому в качестве LCN номера первого кластера рассматривается двухбайтовое поле, которое в примере равно 0x2055 (обратите внимание, что байты на диске записываются в обратном порядке: сначала младшие – 55, затем старшие – 20);

- 0x00 – признак окончания описания списка групп.

Указанные обозначения проиллюстрированы на рис. 21.

Рис. 21

В рассмотренном примере нерезидентный атрибут содержится всего в одной группе, но в общем случае групп может быть несколько.

2.2.3 Альтернативные потоки NTFS

Каждый файл в NTFS представляет набор потоков, в которых хранятся данные. По умолчанию содержимое файла записывается в основной поток, но при необходимости к файлу можно добавлять дополнительные, альтернативные потоки данных (ADS), в которых можно хранить, например, иконки и другую информацию о файле.

Потоки данных файла описываются атрибутом $DATA, одним из свойств которого является имя потока. Основной поток является неименованным, а каждый альтернативный поток должен иметь собственное имя. Альтернативные потоки скрыты от пользователя и не отображаются большинством стандартных программ.Они могут содержать любой тип информации – текстовый, графический, видео и т.д. В альтернативный поток можно даже записать программу, что иногда используется для распространения вредоносного ПО.

Пример. Пусть имеется файл ***primer.txt*** размером 15 байтов, содержащий строку*: ”Hello,students!”*

1.Создаем в файле именованный поток с именем **potok1.txt**:

echo This is second stream> primer.txt:potok1.txt

Размер файла, выводимый командой **dir** или Проводником, при этом не изменится.

2. Просмотрим содержимое основного и альтернативного потоков:

type primer.txt (выводится строка «Hello,students!»)

type primer.txt:potok1.txt (ошибка, команда не видит второй поток !)

more< primer.txt:potok1.txt (выводится строка «This is second stream»)

notepad primer.txt: potok1.txt (Блокнот также видит альтернативный поток)

3. Создадим в файле второй альтернативный поток с именем **potok2.jpg**, содержащий графический файл:

type foto.jpg > primer.txt:potok2.jpg

Размер файла, выводимый командой **dir** или Проводником, при этом также не изменится.

4. Извлечем графические данные из потока с помощью графического редактора:

mspaint primer.txt:potok2.jpg

5. Создадим в файле третий альтернативный поток с именем **calcul.exe**, содержащий программу Калькулятор:

type c:\windows\system32\ calc.exe > primer.txt:calcul.exe

Размер файла, выводимый командой **dir** или Проводником, при этом опять не изменится.

6. Запустим Калькулятор из текстового файла:

s*tart .\ primer.txt:calcul.exe*

2. 3 Программное обеспечение для выполнения работы

Основным инструментом исследования файловой системы магнитных дисков является специальная программа - дисковый редактор. Современные дисковые редакторы обладают большим набором возможностей: просмотр и редактирование системной области диска; просмотр и редактирование директорий и файлов; восстановление удаленных директорий и файлов; доступ к любому участку диска по

номеру сектора или кластера; работа с образом диска; создание загрузочных дисков и т.д.

Для выполнении лабораторной работы можно использовать дисковые редакторы Acronis Disk Editor (http://www.acronis.com/ru-ru/), DiskExplorerForNTFS (http://www.runtime.org/diskexplorer.htm) или DMDE (http://dmde.ru). Редактор DMDE имеет свободно распространяемую версию, которую можно скачать с сайта разработчика, поэтому рекомендуем работать именно с этой программой. Выполнение данной лабораторной работы требует прямого обращения к дискам, что несет потенциальную опасность для вычислительной системы, поэтому для работы необходимо использовать дисковый редактор **только в режиме чтения**.

Если Вы выполняете работу на домашнем компьютере, то можно работать с реальными дисками или флэш – накопителями. В компьютерном классе в целях безопасности администратор системы может отключить возможность работы с дисками, в этом случае можно работать с заранее подготовленным образом диска или использовать виртуальную машину, работающую в любой среде виртуализации (Oracle Virtual Box, Microsoft Hiper-V и т.д.).

В классах ФПМИ лабораторная работа выполняется на виртуальной машине Hiper-V с операционной системой Windows Server 2012, имеющей три логических диска: системный диск с файловой системой NTFS (C:) и пользовательские диски с файловыми системами NTFS (D:) и FAT32 (F:). Удаленное подключение к виртуальной машине проводится с помощью бригадной учетной записи командой:

mstsc -v pmi-os-lab

После загрузки редактора необходимо выбрать тип диска – физический или логический. При выборе физического диска открывается таблица разделов, в которой хранится список логических дисков с указанием типа файловой системы, объема и границ каждого логического диска (рис. 22). Для отображения имени разделов диска можно нажать кнопку «Меню» и выбрать пункт «Показать буквы томов».

Индикаторы показывают наличие соответствующих структур:

- T – таблица разделов
- E – элемент таблицы разделов
- B – загрузочный сектор тома

- C – копия загрузочного сектора
- F – основные структуры ФС (например, начальная запись MFT для NTFS);

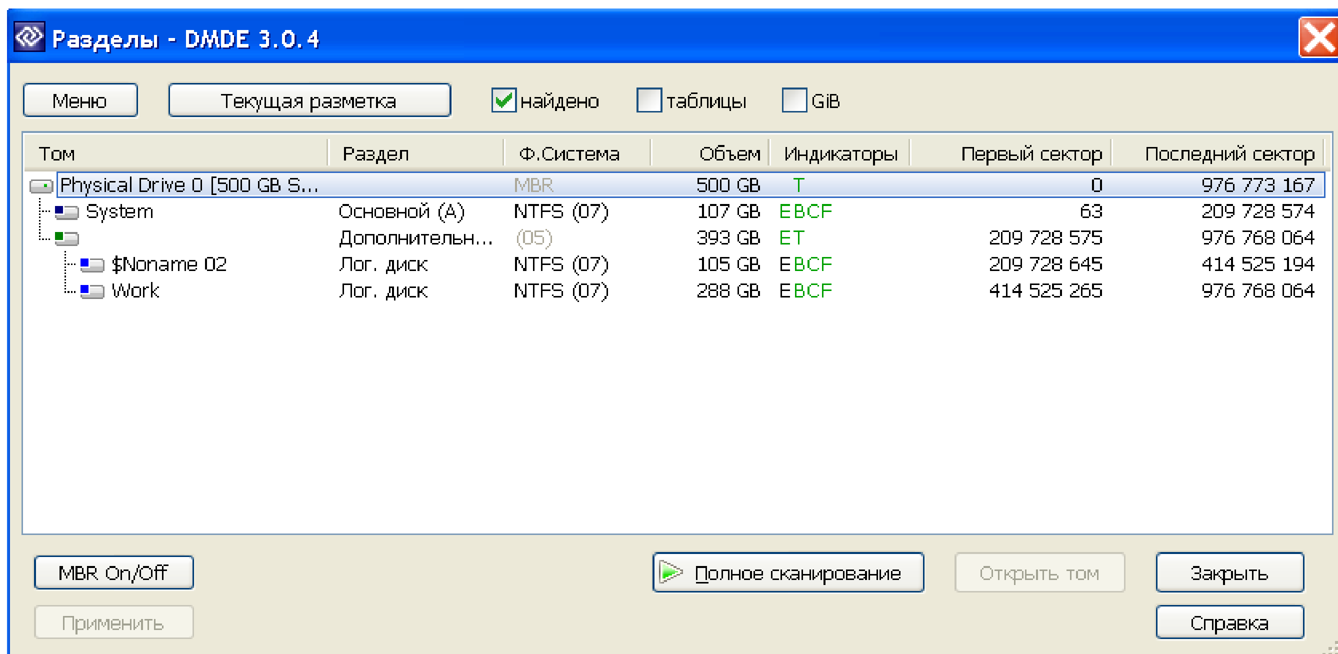


Рис. 22

С помощью контекстного меню для каждого логического диска можно выполнить следующие действия: открыть, удалить или создать образ. Образ представляет собой файл, содержащий снимок диска, т.е. его точную физическую копию, которую можно использовать для восстановления диска в случае повреждения. После открытия логического диска редактор выводит его параметры, набор которых зависит от типа установленной файловой системы: размеры сектора и кластера, число элементов корневого каталога или расположение файла MFT и т.д. (рис. 23).

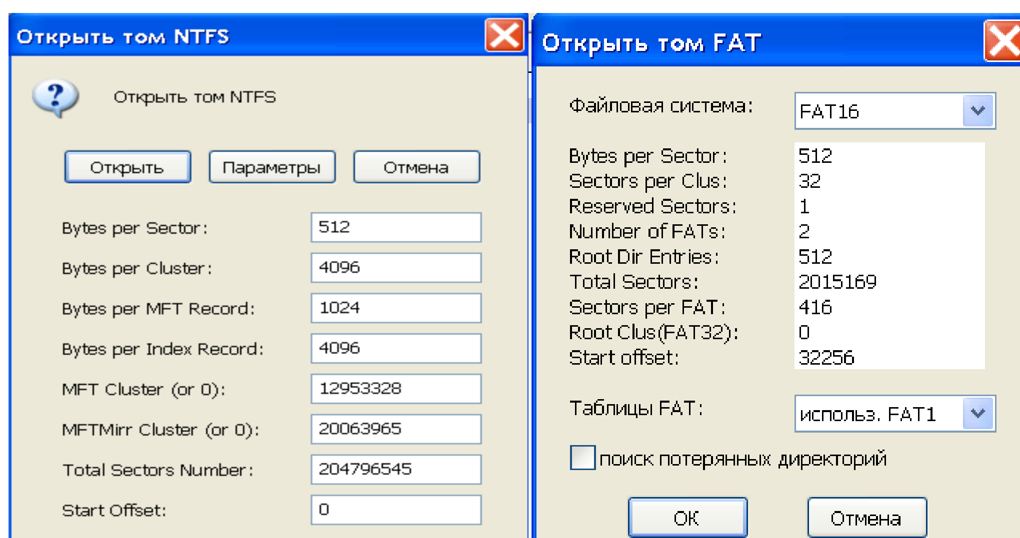


Рис. 23

Нажатие кнопки «Открыть» переводит редактор в режим просмотра, в котором имеется три панели (просмотр папок, просмотр файлов и панель редактора),

показанных на рис. 24. В панель редактора можно выводить содержимое системной области и области данных диска. Управление панелью редактора проводится через меню Редактор.

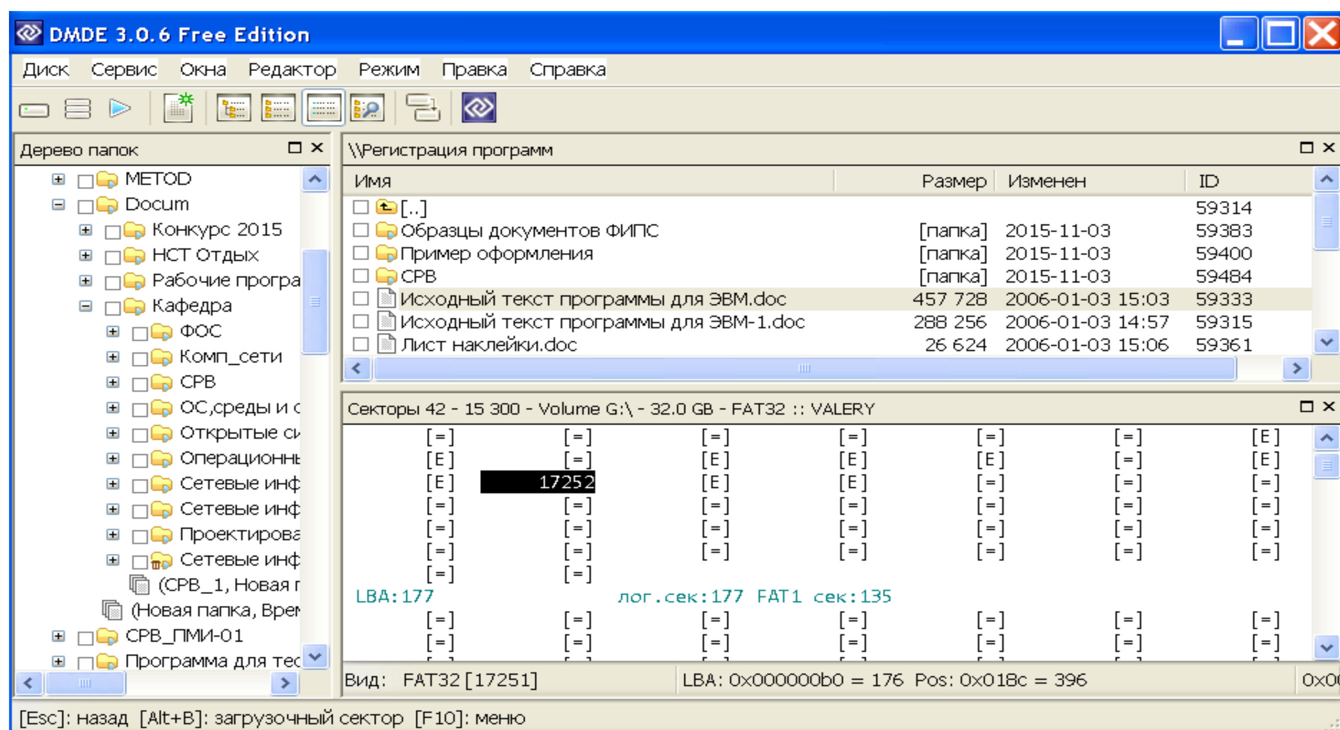


Рис. 24

2.3.1 Работа с файловой системой FAT32

Для FAT32 в панель редактора из системной области можно выводить загрузочную запись, таблицу FAT и корневой каталог, а из области данных – каталоги и файлы.

При просмотре таблицы FAT элементы, соответствующие свободным кластерам, выводятся символом «0», занятым кластерам – символом «=», а занятым последним кластерам – символом «E». Реальные значения элементов FAT выводятся при установке курсора на элемент, при этом в строке статуса отображается название файловой системы и номер кластера, который соответствует текущему элементу FAT (например, FAT32 [17251]). По каждому каталогу и файлу выводится имя, расширение, размер, номер начального кластера, атрибуты и даты создания и изменения (см. рис. 24).

Просмотр содержимого файла, которое выводится в шестнадцатиричном и символьном виде, проводится двойным щелчком мыши по имени файла; изменение кодировки символов проводится в меню Режим/Кодировка. В этом режиме

можно также посмотреть цепочку кластеров, выделенных данному файлу (меню Редактор/ Карта кластеров), как показано на рис. 25.

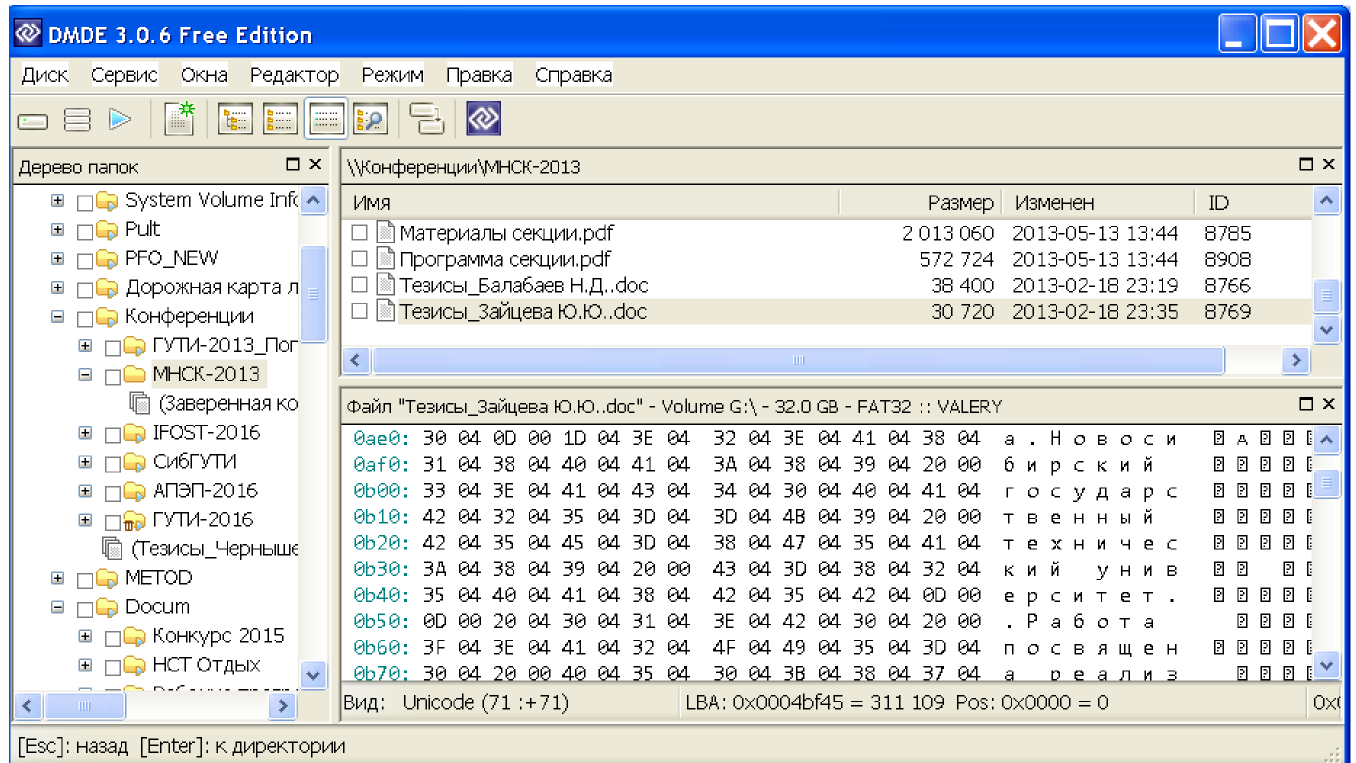


Рис. 25

Элементы каталога, имена которых начинаются с символа «х» соответствуют удаленным файлам. Если в поле имени стоят цифры или символы «е0», то этот элемент предназначен для хранения длинного имени файла (рис. 26).

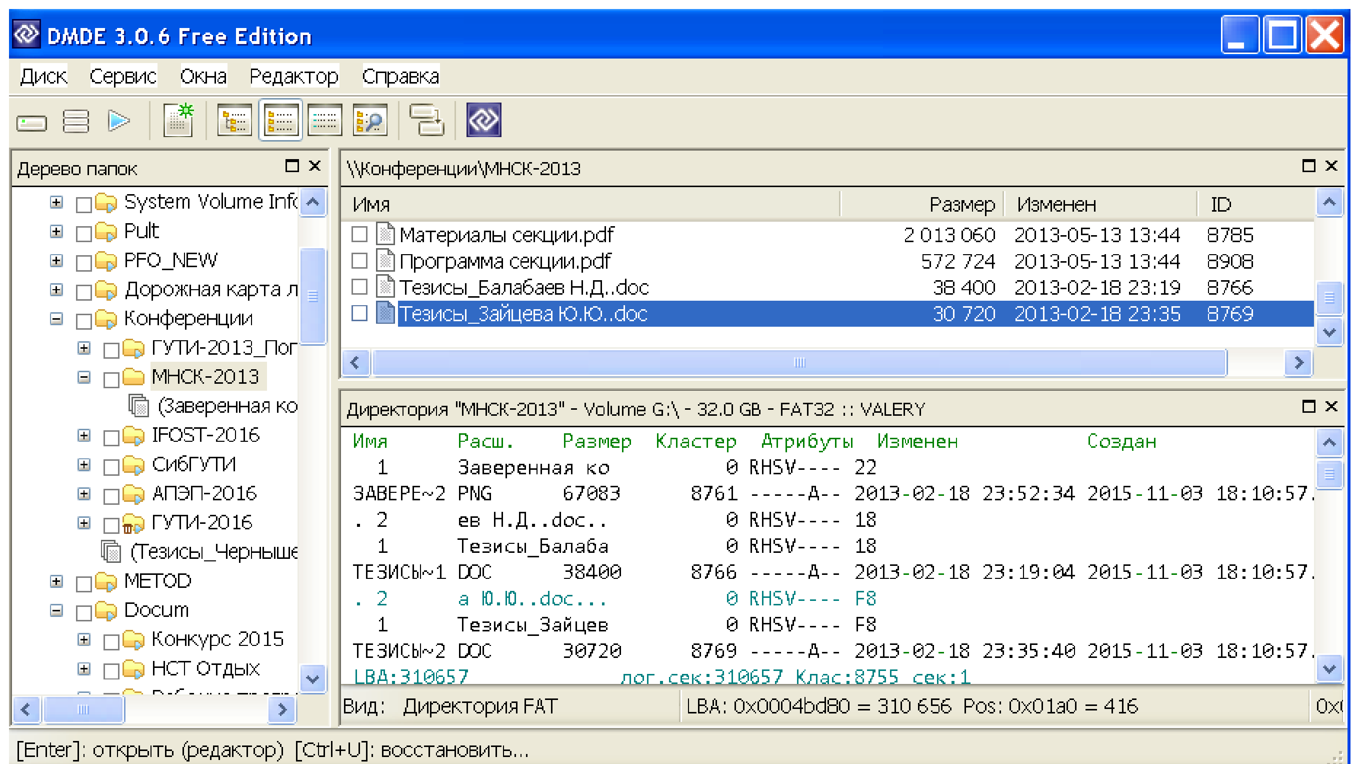


Рис. 26

DMDE позволяет осуществить быстрый переход на заданный кластер или сектор диска по их номеру (меню Редактор/Кластер или меню Редактор/Сектор тома), а также восстановить удаленные файлы.

Для восстановления необходимо отметить на панели нужные файлы, выбрать в контекстном меню пункт «Восстановить объект…» и указать каталог, в который надо провести восстановление. Для того, чтобы не испортить файл-оригинал, восстановление желательно проводить на другой логический диск.

2.3.1 Работа с файловой системой NTFS

После выбора логического диска в окне редактора будет выведено содержимое файла $MFT(рис. 27).

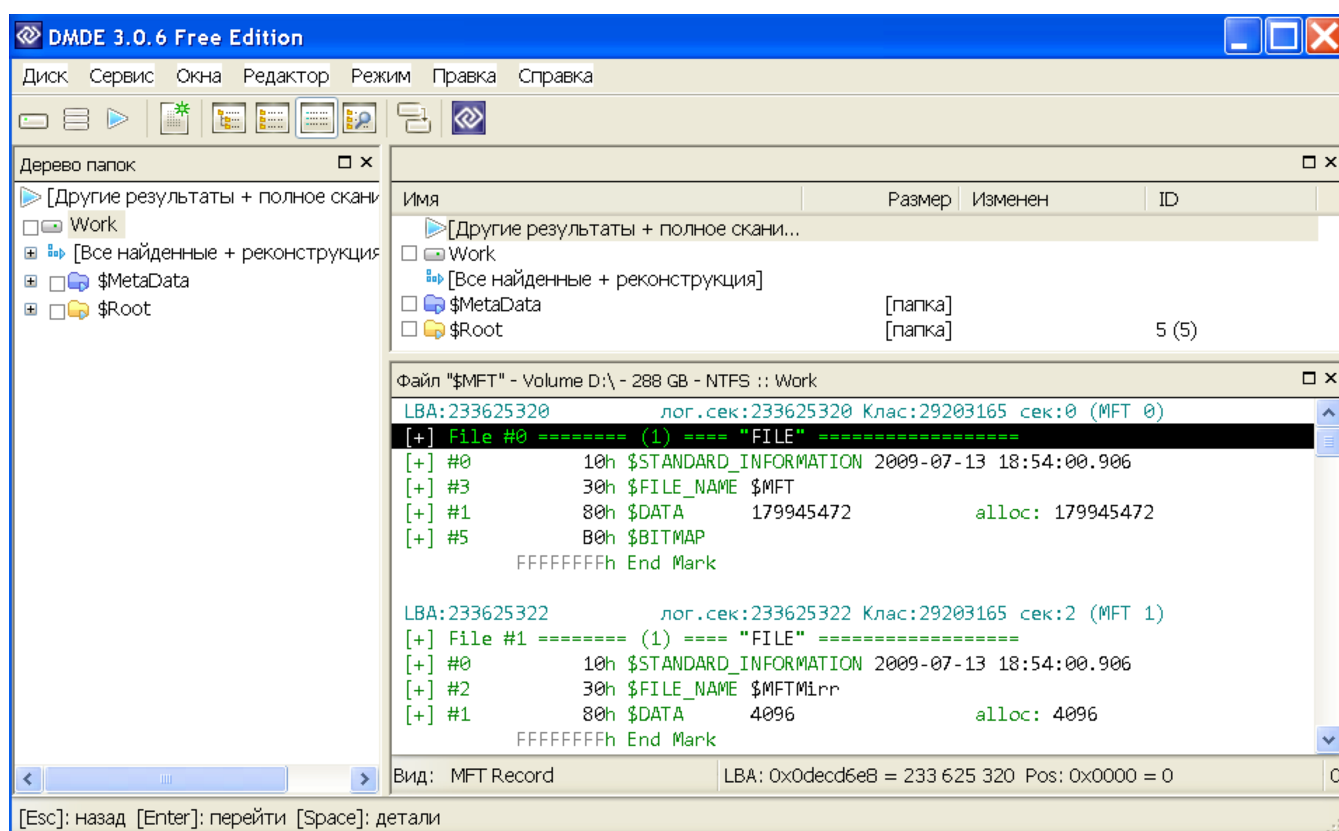


Рис. 27

Первая запись описывает сам файл $MFT, а вторая – копию его первых четырех записей ($MFTMirr). Для каждой записи выводится ее адрес на диске (номера кластера и сектора), граничные метки, внутренний номер (индекс) и набор атрибутов. Минимальный набор включает атрибуты $STAN-DART INFORMATION, $FILE NAME и $DATA. Для просмотра содержимого каждого атрибута необходимо в его строке сделать щелчок мыши на символе ‘+’.

На рис. 28 показано содержимое атрибута $DATA, указывающего на расположение данных одного из файлов.

Анализ рисунка позволяет сделать следующие выводы:

- индекс файла в MFT – 172808;
- данные файла находятся на диске, т.к. атрибут является нерезидентным;
- данные занимают 8 кластеров (start vcn=0, end vcn=7) или 32768 байта, файл не фрагментирован;
- номер начального кластера файла – 18986720;
- длина атрибута – 72 байта.

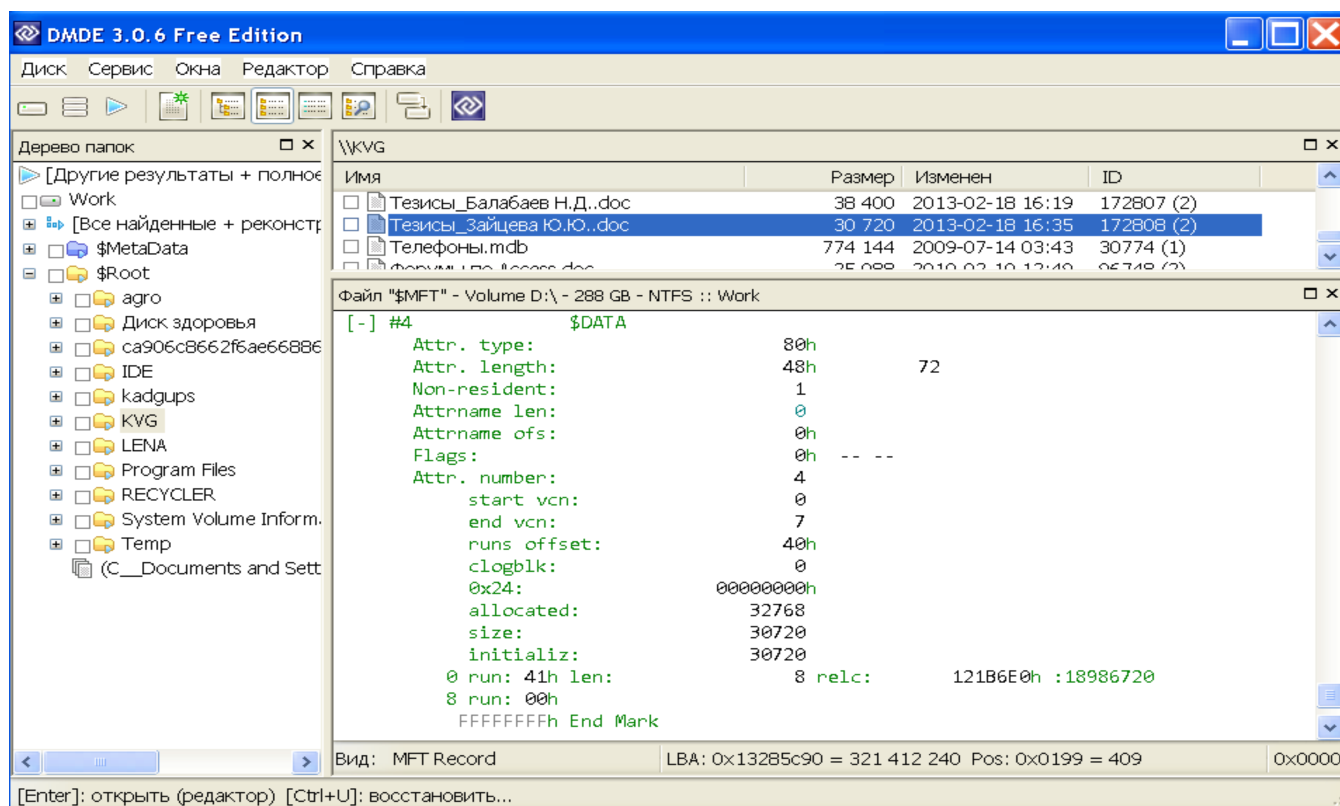


Рис. 28

На рис. 29 этот файл показан в режиме просмотра данных.

3 Порядок выполнения работы

1. Войдите в среду Windows на рабочем компьютере с помощью бригадной учетной записи и подключитесь к виртуальной машине **pmi-os-lab**.
2. Откройте дисковый редактор DMDE и определите параметры виртуального диска: общий объем, число и типы разделов, тип файловой установленной файловой системы. Для FAT - раздела определите размеры сектора и кластера; число секторов, выделенных для таблицы FAT и размер корневого каталога. Для NTFS - раз-

дела определите размеры сектора и кластера, размер файла $MFT и его адрес, размеры записи MFT и индексной записи. Занесите все параметры в отчет, подтверждая их скриншотами.

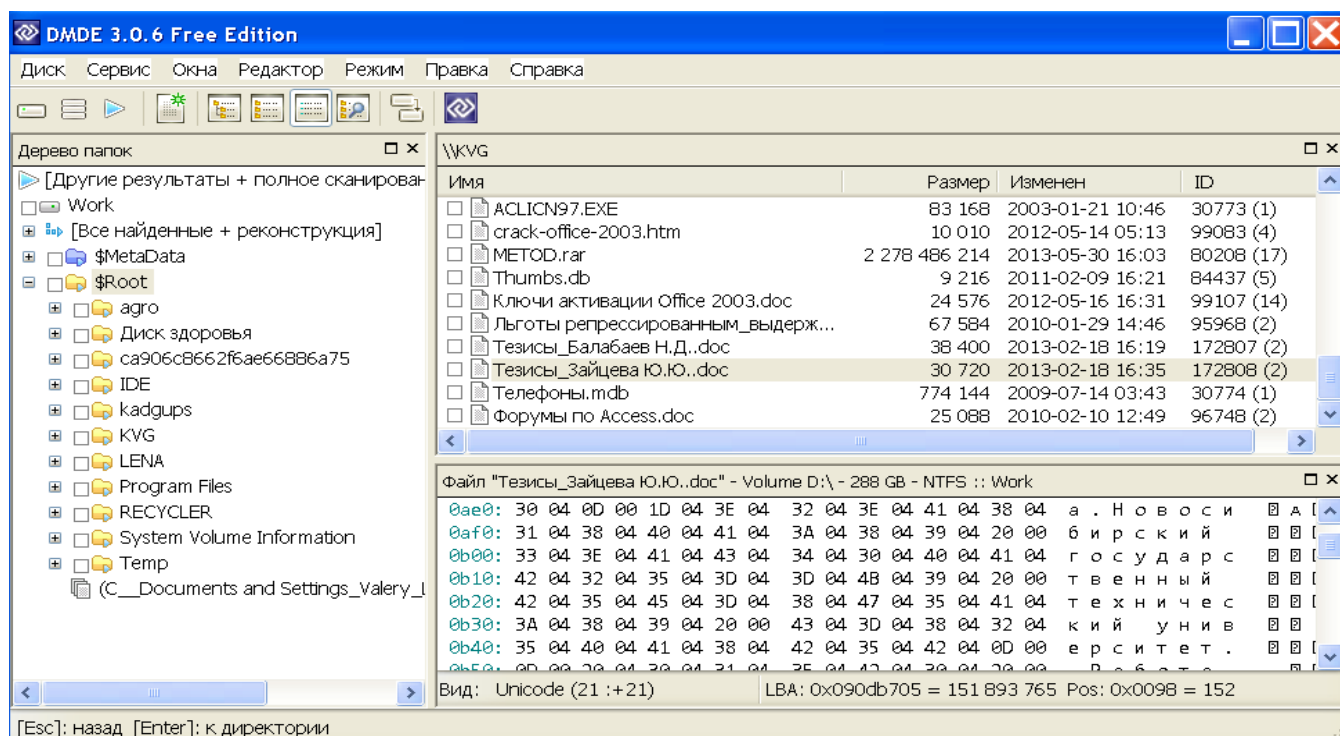


Рис. 29

3. Откройте логический диск с файловой системой FAT32 и выполните следующие действия, подтверждая их скриншотами.

3.1 Создайте на диске каталог с именем, соответствующим Вашей учетной записи и в нем создайте структуру каталогов согласно заданию лабораторной работы № 1 (см. рис. 1).

3.2. В каталог *abc_kk* запишите три файла размером 40 – 60 Кбайт, имеющих форматы *.txt*, *.doc* и *.docx* , имена файлов должны содержать не менее 15 символов, например, *Лабораторная работа № 6*. Содержимое файлов должно быть записано с использованием кириллицы. Если виртуальная машина **pmi-os-lab** не поддерживает работу с кириллицей, то необходимо добавить поддержку русского языка через панель управления Windows (Control Panel/Clock,Langauge and Region/ Langauge/Add a Langauge).

3.3. Для файла *Лабораторная работа № 6.txt* выполните следующие действия:

- определите число элементов каталога, выделенных для хранения информации по файлу;

- занесите в таблицу 22 содержимое элемента, предназначенного для хранения короткого имени;

Таблица 22

Наименование поля	Значение поля
имя файла	
расширение имени	
атрибуты	
время создания	
дата создания	
номер начального кластера	
размер файла	

- просмотрите содержимое и коды первых 16 байтов, занесите их в отчет;

- определите используемую кодировку символов путем сравнения с кодировочными таблицами редактора;

- определите список кластеров этого файла, результаты занесите в таблицу 23;

Таблица 23

Логический номер кластера в файле	1	2	3	…	n
Номер кластера на диске					
Значение элемента FAT					

3.4. С помощью программы *Проводник* скопируйте файл *Лабораторная работа № 6.txt* в каталог *trash_kk*.

3.5. Удалите файл *Лабораторная работа № 6.txt* из каталога *abc_kk*, проведите анализ изменений в FAT и в каталоге *abc_kk*, результаты занесите в отчет в виде таблиц 22 и 23. Посмотрите содержимое начального кластера удаленного файла, результат занесите в отчет.

3.6. Восстановите удаленный файл *Лабораторная работа № 6.txt.*

3.7. Определите используемую кодировку символов для файлов *Лабораторная работа № 6.doc* и *Лабораторная работа № 6.docx*, результаты занесите в отчет.

Обратите внимание: файлы, созданные редактором MS Word имеют следующие особенности:

- содержимое файла начинается с 5-го сектора начального кластера, выделенного этому файлу;
- формат файлов .docx, используемый по умолчанию в современных версиях редактора, начиная с Office Word 2007, основан на XML. Это дает возможность получить файлы меньшего размера по сравнению со стандартным форматом .doc, но не позволяет просмотреть их содержимое с помощью редактора DMDE.

4. Откройте логический диск с файловой системой NTFS и выполните действия, подтверждая их скриншотами..

4.1. Создайте на диске структуру каталогов и файлов согласно п.3.1 и п.3.2.

4.2. Определите характеристики файла $MFT (начальный адрес, число записей, размер в байтах и кластерах).

4.3. Определите число записей в файле $MFTmirr.

4.4. Проведите полный анализ записи MFT, соответствующей файлу *Лабораторная работа № 6.txt* и занесите в отчет описания всех атрибутов, включая расположение файла на диске.

4.5. Удалите файл *Лабораторная работа № 6.txt,* проведите анализ изменений в MFT и в области данных. Результаты занесите в отчет.

4.6. Восстановите удаленный файл.

4.7. С помощью программы Блокнот создайте текстовый файл **primer.txt**, записав в него фразу «Very good weather today!». Проведите анализ соответствующей записи MFT, определить адрес этого файла на диске.

4.8. Запишите в файл **primer.txt** второй поток данных, используя для этого, например, любой текстовый файл размером не менее 50 Кбайт. Проведите анализ соответствующей записи MFT и определите расположение данных этого потока на диске. Определите размер файла, сравните с предыдущим пунктом.

4.9. Запишите в файл **primer.txt** третий поток данных, используя для этого любой графический файл (например, фотографию). Проведите анализ соответствующей записи MFT и определите расположение данных этого потока на диске. Определите размер файла, сравните с предыдущим пунктом.

4 Контрольные вопросы

1. Каким образом поддерживается древовидная многоуровневая система каталогов в Windows?

4. Какова структура FAT, в чем отличия для жестких и гибких дисков?

5. Какова структура каталогов файловой системы FAT ? В чем отличие корневого и прочих каталогов ?

6. Какие действия выполняются файловой системой при удалении файла в файловых системах FAT и NTFS ?

7. Поясните действия файловой системы FAT при поиске файла по имени:

а) файл находится в корневом каталоге;

б) файл расположен в подкаталоге.

8. Поясните механизм выделения дисковой памяти файловой системы FAT при записи нового файла на диск.

9. Какие компоненты компьютера используют физическую и логическую модели магнитного диска ?

10. Чем определяется число элементов каталога, выделяемых для хранения метаданных файла в файловой системе FAT?

11. Назовите основные различия файловых систем FAT и NTFS.

12. Какова структура файла MFT ?

13. Поясните структуру файловой записи MFT.

14. Алгоритмы восстановления файлов в FAT и NTFS.

15. Резидентные и нерезидентные атрибуты записи MFT.

16. Каким образом в NTFS увеличена скорость доступа к файлам по сравнению с FAT ?

Лабораторная работа № 6. Инструментальные средства разработки программ

1. Цель работы

Целью работы является изучение основных этапов разработки и отладки приложений в ОС Linux, а также приобретение практических навыков по использованию инструментальных средств фонда свободного программного обеспечения при компиляции исходного кода, сборке, отладке и тестировании программ, написанных на языке Си.

2. Методические указания

В настоящее время для разработки программного обеспечения часто используются интегрированные системы программирования, включающие компилятор, компоновщик, текстовый редактор, отладчик и другие программы. Такие системы скрывают от программиста многие детали, связанные с подготовкой программ к выполнению и не позволяют студентам в полной мере понять такие важные вопросы, как методы сборки и отладки исполняемого модуля, классификация ошибок и т.д. При выполнении лабораторной работы необходимо освоить все операции по подготовке программ к исполнению в командном режиме.

ОС Linux имеет полный набор инструментов для разработки приложений, поддерживающий все основные этапы разработки:

- создание исходного кода (текста) программы;
- сохранение различных вариантов исходного текста;
- компиляция исходного кода;
- компоновка программы (сборка);
- отладка программы;
- сохранение всех изменений, выполняемых при тестировании и отладке.

Технология подготовки программы к исполнению зависит от способа преобразования исходного кода в машинный код. Существует два таких способа – компиляция и интерпретация. *Компиляция* основана на том, что специальная программа, называемая компилятором, просматривает и обрабатывает весь текст исходной программы в набор машинных команд, который записывается в отдельный объектный файл. Далее независимо откомпилированные объектные файлы с помощью

программы, которая называется компоновщиком, собираются в единый исполняемый файл. *Интерпретация* заключается в том, что каждая строка исходной программы отдельно переводится в набор машинных команд, который сразу загружается в ОЗУ и выполняется. Для реализации этого способа вместо компилятора используется другая программа – интерпретатор. Наиболее часто используется компилирующий способ преобразования, поэтому далее будет рассматриваться именно этот способ.

На рисунке 30 приведена общая схема прохождения программы через среду разработки и операционную систему, где ЦП – центральный процессор, ОЗУ – оперативная память.

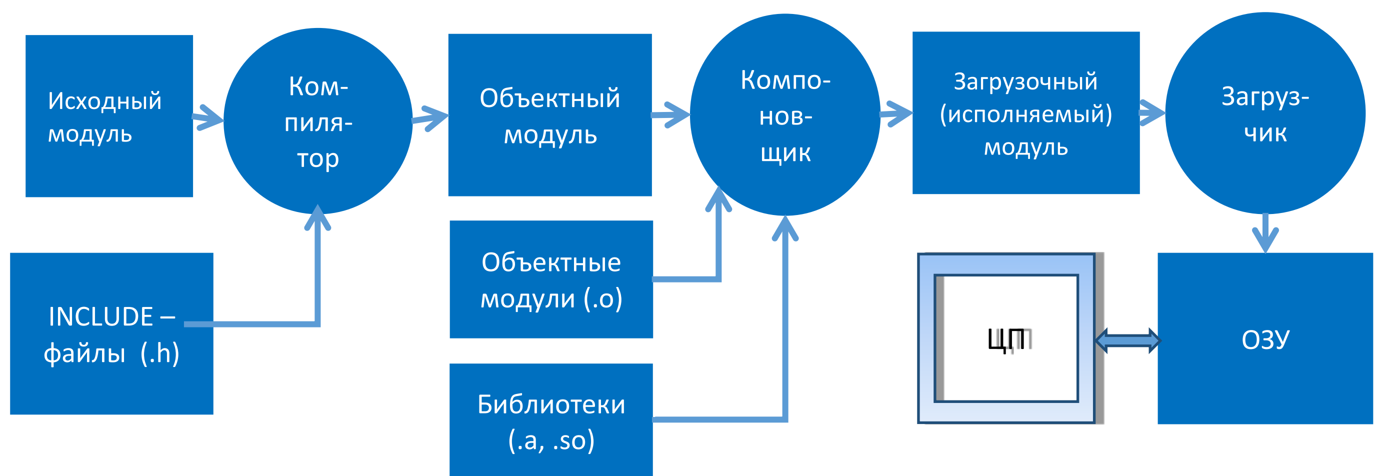


Рис.30

Компилятор преобразует исходную программу (исходный модуль) в набор машинных двоичных команд (объектный модуль) с относительной адресацией.

Компоновщик подключает к объектному модулю, сформированному компилятором, другие объектные модули, требуемые для его выполнения, в результате чего получается загрузочный (исполняемый) модуль, также имеющий относительную адресацию. Дополнительные объектные модули могут находиться как в виде последовательных файлов с расширением .o, так и в библиотечных файлах, имеющих расширение .a или .so. Здесь расширения имен файлов указаны для ОС Linux, в других ОС они могут быть отличаться.

Загрузчик получает от операционной системы физический адрес выделенного для загрузки программы участка и преобразует относительную адресацию исполняемого модуля в абсолютную (физическую) с учетом используемого алгоритма

распределения памяти, загружает этот модуль в выделенные участки памяти и сообщает центральному процессору адрес первой команды программы.

В лабораторной работе будут использованы компилятор **gcc,** компоновщик **ld** и отладчик **gdb**.

2.1 Компиляция исходного модуля

Компиляция – процесс перевода программы с алгоритмического языка на язык машинных команд, при этом исходный модуль программы преобразуется в объектный модуль. В общем случае обработка исходной программы компилятором выполняется в три этапа:

- препроцессорная обработка;
- преобразование результата работы препроцессора в ассемблерный код;
- преобразование ассемблерного кода в набор машинных команд.

На этапе препроцессорной обработки в исходную программу включается содержимое всех заголовочных файлов, указанных в директивах **#include**. В заголовочных файлах обычно находятся объявления функций, используемых в исходной программе, но не определённых в ее тексте. Их определения находятся в других файлах с исходным кодом или в бинарных библиотеках. Результатом препроцессорной обработки является объединенный исходный код программы.

На этапе формирования ассемблерного кода проводятся лексический и синтаксический анализ объединенного исходного кода, а также перевод операторов языка программирования высокого уровня в набор ассемблерных команд. На заключительном этапе каждая команда ассемблера переводится в машинную команду.

Компилятор также включает в состав объектного модуля дополнительную служебную информацию, необходимую для компоновки – словарь внешних символов ESD и словарь перемещений RLD.

Компилятор **gcc** способен по суффиксам файлов определять их типы и действие, которое необходимо с ними выполнить (см. табл. 24).

Таблица 24

Имя файла	Содержимое и действие
file.c	исходный код на С, который нуждается в препроцессорной обработке
file.cpp	исходный код на С++, который нуждается в препроцессорной обработке
file.i	исходный код на С, который не нуждается в препроцессорной обработке
file.ii	исходный код на С++, который не нуждается в препроцессорной обработке
file.h	заголовочный файл С

file.S	ассемблерный код, который нуждается в препроцессорной обработке
file.s	ассемблерный код
file.o	объектный код

Компилятор имеет множество различных опций, используемых для управления процессом компиляции. Для выполнения лабораторной работы потребуются следующие опции:

-g – включить в исполняемый файл дополнительную отладочную информацию, используемую отладчиком gdb;

-c – отключить автоматический вызов компоновщика, при этом на выходе компилятора формируется объектный файл для каждого исходного входного файла;

-S – сформировать файл с ассемблерным кодом для каждого не ассемблерного входного файла, последующую обработку прекратить;

-E – остановиться после этапа препроцессорной обработки, на выходе формируется исходный файл с подключенными заголовочными файлами;

-o file – поместить результат в файл с указанным именем (по умолчанию результат записывается в файл a.out);

-I*каталог* – добавить указанный каталог в список каталогов для поиска заголовочных файлов;

-L*каталог* – добавить указанный каталог в список каталогов для поиска библиотек;

При обработке исходной программы компилятор может выводить два типа диагностических сообщений – предупреждение (warning) и ошибка (error). *Предупреждение* выводится в случае обнаружения ситуации, которая не влияет на корректность работы программы (например, в программе описана переменная, но она нигде не используется). *Ошибка* является следствием нарушения синтаксиса языка программирования и делает невозможным формирование объектного модуля.

В сообщениях компилятор указывает место обнаружения ошибки в исходном коде. Часто возникает ситуация, когда число сообщений об ошибках превышает число реальных ошибок в программе. Это объясняется тем, что компилятор просматривает всю программу целиком и реальная ошибка в одном операторе может приводить к неправильным синтаксическим конструкциям в нескольких последующих операторах программы, что приводит к генерации «наведенных» сообщений об ошибках.

После успешного формирования объектного модуля компилятор **gcc** без дополнительных команд со стороны пользователя вызывает компоновщик, который формирует исполняемый (загрузочный) модуль. Если при вызове компилятора была указана опция -c, то компоновщик не вызывается.

Для проведения отладки программы в состав исполняемого модуля рекомендуется добавить дополнительную отладочную информацию, указав компилятору ключ –g. В этом случае компилятор добавляет в исполняемый код исходный текст программы и сохраняет связь каждой строки исходного текста с машинным кодом, за счет чего становится доступным его пошаговое выполнение.

Отладочная информация значительно увеличивает размер исполняемого файла, поэтому после отладки рекомендуется её удалить с помощью программы **strip**. Для того, чтобы полностью очистить файл от отладочной информации, необходимо указать опцию -s: **strip -s a.out**

Примеры:

gcc -o abcd –g abcd.c – компиляция программы **abcd.c** и формирование исполняемого модуля **abcd** с включением в него отладочной информации;

gcc -c abcd.c – компиляция программы **abcd.c** и формирование объектного модуля **abcd.o**;

gcc -E abcd.c – препроцессорная обработка программы **abcd.c** и формирование результата на стандартный вывод.

2.2 Компоновка программы

Компоновка программы – процесс организации межпрограммных связей, позволяющий объединить независимо компилированные объектные модули в единый исполняемый (загрузочный) модуль. Компоновщик формирует из набора объектных модулей с относительной адресацией загрузочный модуль, имеющий единую относительную адресацию. Такой модуль является перемещаемым, что позволяет операционной системе загружать его в любой свободный участок оперативной памяти.

Компоновщик (или редактор связей) последовательно разрешает все внешние ссылки, получая информацию о них из словарей внешних символов каждого объектного модуля, включаемого в формируемый исполняемый модуль. Процесс разрешения внешней ссылки заключается в поиске модуля с именем, указанным в ESD, и включении его в состав исполняемого модуля, причем поиск может проводиться только в файлах, указанных пользователем, или в известных компоновщику библиотеках.

Компоновщик **ld** имеет большое число управляющих опций и позволяет обеспечить максимально возможный контроль за процессом сборки исполняемого модуля. В лабораторной работе могут потребоваться следующие опции, которые могут находиться в командной строке в любом порядке:

-l*имя* – добавляет в список файлов для компоновки статическую библиотеку с именем **libимя.a**; например, опция -lm добавляет в список библиотеку математических функций libm.a;

-L*каталог* – добавляет указанный каталог в список каталогов для поиска библиотек;

-Map=*имя* – записывает в файл с указанным именем структуру исполняемого модуля, содержащую информацию об относительных адресах, именах и размерах всех объектных модулей, включенных в состав исполняемого модуля, а также диагностические сообщения, возникшие на этапе компоновки; этот файл называется картой памяти.

Компоновщик поддерживает два метода компоновки - статический и динамический. Статическая компоновка проводится с использованием библиотечных файлов с расширением **.a** путем включения в состав исполняемого модуля всех необходимых функций из библиотек. Динамическая компоновка основана на том, что в состав исполняемого модуля включаются только ссылки на библиотечные модули, а непосредственное их извлечение из библиотек и загрузка в оперативную память проводятся на этапе выполнения программы. Для этого метода можно использовать только специальные библиотеки динамической компоновки, находящиеся в файлах типа **.so**.

Компоновщик может вызываться неявным или явным способами. Неявный вызов проводится с помощью компилятора и является наиболее простым способом компоновки. Если компилятор во время обработки исходной программы не обнаружил ошибок, то он сам вызывает компоновщик и передает ему на вход сформированный объектный модуль.

Применение явного вызова позволяет максимально полно использовать все возможности компоновщика, например, сформировать карту памяти, но в этом случае необходимо указать имена всех дополнительных модулей, создающих среду выполнения программы.

Примеры:

а) неявный вызов

```
gcc func1.o main.o -o myprogram
```

б) явный вызов:

```
ld  -dynamic-linker  /lib64/ld-linux-x86-64.so.2  -o myprogram  -Map=abcdmap
/usr/lib64/crt1.o  /usr/lib64/crti.o  /usr/lib/gcc/x86_64-redhat-linux/4.8.5/crtbegin.o  main.o  func1.o  -lc
/usr/lib/gcc/x86_64-redhat-linux/4.8.5/crtend.o  /usr/lib64/crtn.o
```

Здесь в первом случае вызывается компилятор **gcc** и на его вход подаются два объектных файла – func1.o и main.o, результат записывается в файл **myprogram** и для компоновки используется стандартная библиотека **libc.a**, а во втором – вызывается компоновщик **ld**, которому на вход подаются файлы пользователя func1.o и main.o и служебные файлы (crt1.o, crti.o, crtbegin.o, crtend.o, crtn.o), а на выходе формируются исполняемый файл **myprogram** и карта памяти **abcdmap** .

Назначение служебных модулей:

crt1.o – содержит код, инициализирующий среду исполнения языка С и вызывающий определенную пользователем функцию main;

crti.o – содержит пролог функции _init, помещаемый в начало секции .init;

crtn.o – содержит эпилог функции _init, помещаемый в конец секции .init;

crtbegin.o, crtend.o – обрабатывают глобальные конструкторы и деструкторы языка С++;

Для того, чтобы не указывать абсолютный путь к служебным файлам, при выполнении лабораторной работы рекомендуется скопировать все эти файлы в Ваш рабочий каталог. В этом случае в командной строке можно указывать только имена файлов. Обратите внимание: в имени файла crt1.o используется символ «единица» !

На рис. 31 приведен пример фрагмента карты памяти, содержащий план размещения машинного кода исполняемого модуля программы. Секция кода имеет стандартное имя .text, каждая запись содержит имя секции, ее относительный адрес, размер и имя файла с программной функцией, расположенный по этому адресу. Все адреса и размеры выводятся в шестнадцатиричной системе счисления.

```
 .text          0x00000000004004f0      0x2c /home/NSTU/staff/kvg/obj/crt1.o
                0x00000000004004f0               _start
 .text          0x000000000040051c       0x0 /home/NSTU/staff/kvg/obj/crti.o
 *fill*         0x000000000040051c       0x4
 .text          0x0000000000400520      0xbd /home/NSTU/staff/kvg/obj/crtbegin.o
 *fill*         0x00000000004005dd       0x3
 .text          0x00000000004005e0     0x16b abcd2.o
                0x00000000004005e0               GetWords
                0x00000000004006b7               main
 *fill*         0x000000000040074b       0x1
 .text          0x000000000040074c      0x82 printwords.o
                0x000000000040074c               PrintWords
 *fill*         0x00000000004007ce       0x2
 .text          0x00000000004007d0      0x72 /usr/lib64/libc_nonshared.a(elf-init.oS)
                0x00000000004007d0               __libc_csu_init
                0x0000000000400840               __libc_csu_fini
 *fill*         0x0000000000400842       0x2
 .text          0x0000000000400844       0x0 /home/NSTU/staff/kvg/obj/crtend.o
 .text          0x0000000000400844       0x0 /home/NSTU/staff/kvg/obj/crtn.o
 *(.gnu.warning)
```

Рис. 31

Например, по адресу 0x4005E0 находится код объектного модуля **abcd2.o**, имеющий размер 363байта, а по адресу 0x40074C – код модуля **printwords.o,** занимающий в памяти 130 байтов.

2.3 Утилита make

При разработке большой программы, состоящей из нескольких исходных файлов, приходится постоянно следить за файлами, которые требуют перекомпиляции после внесения изменений. Программа **make** освобождает пользователя от такой рутинной работы и служит для документирования взаимосвязей между файлами. Описание взаимосвязей и соответствующих действий хранится в текстовом файле, который должен иметь имя **makefile** или **Makefile**.

В общем случае **makefile** является списком правил. Каждое правило начинается с имени, после которого стоит двоеточие, а далее через пробел указываются зависимости – имена файлов, от которых зависит достижение цели. После зависимостей идут команды Linux, каждая из которых должна находиться на отдельной строке и начинаться с символа табуляции. Структура правила может быть сложной и включать ветвления, циклы и другие конструкции языка shell. Строки, которые начинаются с символа «#», являются комментариями.

Например, make-файл для программы, хранящейся в файле **abcd.c,** может иметь вид:

```
# Makefile for abcd.c
#  Compile abcd.c normaly
 abcd: abcd.c
```

```
        gcc -o abcd abcd.c
# Compile abcd.c with debugging
 testabcd: abcd.c
        gcc -o testabcd -g abcd.c
# End Makefile
```

Этот файл включает два правила компиляции и построения исполняемого модуля: первое предусматривает обычную компиляцию с построением исполняемого модуля с именем **abcd**, второе позволяет включить в исполняемый модуль **testabcd** информацию для отладки на уровне исходного текста.

В качестве имени правила может быть указана любая символьная строка, но рекомендуется использовать имена файлов или действие. Например, для сборки программы, состоящей из трех модулей, фрагмент make-файла может выглядеть следующим образом:

```
iResult: main.o func1.o func2.o
        gcc  -o iResult main.o func1.o func2.o
```

Здесь целью является файл iResult (исполняемый файл программы), а правило описывает, как можно получить новую версию файла этого файла (скомпоновать из перечисленных объектных файлов).

Make-файл должен находиться в одном каталоге со всеми файлами программы, запуск утилиты **make** проводится следующим образом: **make имя_правила,** например, **make abcd.** Если не указывать какой-либо цели в командной строке, то **make** выбирает по умолчанию первое правило make-файла.

2.4 Отладка и тестирование

Отладка – этап разработки, связанный с обнаружением, локализацией и устранением ошибок, целью отладки является обеспечение полного выполнения всех требований, предъявляемых к программе. Тестирование – это процесс исследования и испытания программы, имеющий две цели: продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям, и выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим требованиям.

Ошибки, возникающие при разработке программ, могут быть синтаксическими, сборочными и семантическими. Синтаксические ошибки связаны с нарушением формальной грамматики языка программирования и обнаруживаются компилятором. Ошибки сборки могут возникать на этапе компоновки из-за отсутствия

необходимых файлов или библиотек, они обнаруживаются компоновщиком. При наличии этих типов ошибок формирование исполняемого модуля невозможно.

Семантические ошибки не обнаруживаются компилятором и компоновщиком, но приводят к неправильной работе программы, когда результат ее работы не соответствует требованиям. Обнаружить такие ошибки можно только с помощью специальных программ – отладчиков. Для использования отладчика в программу необходимо включить отладочную информацию на этапе компиляции, выполнив для нашего примера команду **make testabcd**.

Запуск отладчика **gdb** проводится командой **gdb testabcd**, при этом на экран будет выведена информация об отладчике и появится приглашение для ввода его команд**,** основными из которых являются:

backtrace – выводит весь путь к текущей точке останова, то есть названия всех функций, начиная с main();

break *параметр* – устанавливает точку останова, параметром может быть номер строки или название функции;

continue – продолжает выполнение программы от текущей точки до конца;

display – добавляет выражение в список выражений, значения которых отображаются каждый раз при остановке программы;

finish – выполняет программу до выхода из текущей функции; отображает возвращаемое значение, если такое имеется;

help [*команда*] **–** информация о команде или общая информация об использовании отладчика gdb;

list – пролистывает 10 строк вниз, начиная с текущей;

next – пошаговое выполнение программы, но, в отличие от команды step, не выполняет пошагово вызываемые функции;

print *выражение* – выводит значение какого-либо выражения;

run – запускает программу на выполнение;

step – пошаговое выполнение программы;

quit – выход из отладчика.

2.5 Система управления версиями

Современные сложные программы разрабатываются коллективом программистов и состоят из большого числа программных модулей, представленных в виде набора отдельных файлов, каждый из которых компилируется и отлаживается независимо от других модулей. После отладки всех модулей проводится сборка программы в единый исполняемый файл.

Для организации коллективной работы над программным проектом используется специальное программное обеспечение, которое называется системой управления версиями. При разработке сложных программ необходимо отслеживать изменения исходного кода (кто вносил изменения, когда и какова причина изменений). Если несколько человек работают над проектом программы, важно, чтобы они не делали изменений одного и того же кода одновременно. Классическим примером такой системы является система CVS (Concurrent Versions System), входящая в комплект поставки практически всех дистрибутивов Linux.

2.5.1 Общие принципы работы CVS

Система CVS хранит историю изменений определённого набора текстовых файлов (например, исходного кода программ). Все файлы располагаются в специальном хранилище – репозитории, который обычно размещается на отдельном выделенном сервере, но может находиться и на локальном компьютере пользователя, т.к. учет версий может быть удобен и в случае однопользовательской работы. CVS реализован на основе архитектуры клиент-сервер, причем серверная часть управляет репозиторием, а клиентские части устанавливаются на локальные компьютеры пользователей.

Объектом управления CVS является модуль, под которым понимается отдельный блок информации, который весь целиком может быть запрошен пользователем. Обычно для каждого проекта или группы проектов заводится свой CVS-модуль, который хранится в одном или нескольких каталогах репозитория.

В репозитории хранится только последняя версия модуля и история всех внесенных в него изменений, начиная с начальной версии. Поэтому в случае необходимости всегда есть возможность вернуться к любой предыдущей версии проекта. Регистрация изменений от лица конкретного пользователя хранится с точностью до строки. Имена версий файлов хранятся в виде *имя_файла номер_версии*, например *myfile.cpp 1.1*, *myfile.cpp 1.2* и т.д.

Для работы с текущей версией исходной программы на локальном компьютере создается рабочий каталог, который не должен находиться в одном каталоге с репозиторием. Рабочий каталог создается отдельно для каждого программного проекта. Локальная копия модуля, полученная из репозитория в рабочий каталог с помощью CVS- клиента, называется *рабочей копией*.

Упрощенная схема системы работы CVS включает следующие основные шаги:

- создание хранилища (репозитория);
- создание рабочего каталога и размещение в нем текущей версии отлаживаемой программы;
- сохранение текущей версии исходного файла в репозитории.

Отладка и изменение исходного файла должны выполняться в рабочем каталоге. После внесения изменений текущая версия передается в хранилище с соответствующим комментарием. После передачи в хранилище соответствующий проект может быть удален из рабочего каталога, чтобы исключить возможность внесения в него изменений без поддержки CVS.

Имя каталога, в котором располагается хранилище, можно записать в глобальную переменную $CVSROOT для того, чтобы его не указывать в каждой команде CVS.

2.5.2 Основные команды CVS

Общий синтаксис команд выглядит следующим образом:

cvs [опции_cvs] команда [опции _команды] [аргументы]

Здесь *опции_cvs* используются для управления режимами CVS, *опции _команды* уточняют действие конкретной команды, *аргументы* задают имена объектов, над которыми выполняется команда. В таблице 25 приведены основные команды CVS.

Таблица 25

Команда	Назначение	Синтаксис
init	создать репозиторий	cvs init имя_каталога
add	добавить в репозиторий новый файл или каталог	cvs add имя_файла cvs commit -m "[комментарий]" имя_файла
checkout	извлечь из репозитория копию исходного текста в рабочий каталог проекта	cvs checkout имя_проекта
commit	сохранить в репозитории изменения, внесенные в локальную копию	cvs commit -m ["комментарий"]

release	удалить файлы проекта из локального каталога	cvs release -d имя_проекта
remove	удалить файл из репозитория	cvs remove -f имя_файла cvs commit -m "[комментарий]" имя_файла
log	получить историю работы с файлом	cvs log [имя_файла]
diff, rdiff	просмотреть изменения файла от версии к версии	cvs diff -r версия_1 [-r версия_2] имя_файла
update	1. получить указанную версию файла 2. синхронизировать файлы в локальном каталоге и репозитории	1. cvs update -r номер_версии имя_файла 2. cvs update [имя_файла]

2.5.3 Пример работы с CVS

Исходные данные: имя домашнего каталога – /home/brigades/ pmi7501, имя файла – file.c.

1. Создание хранилища:

а) в домашнем каталоге создаем каталог **cvsroot**.

б) в каталоге **cvsroot** создаем хранилище:

```
cvs -d /home/brigades/pmi7501/cvsroot  init
```

2. Создание рабочего каталога:

а) в домашнем каталоге создаем каталог **workdir**

б) в каталоге **workdir** создаем каталог **project**, в котором будут находиться файлы проекта;

в) в каталог **project** запишем файл **file.c**.

3. Связывание рабочего каталога с хранилищем:

а) сделать рабочий каталог текущим;

б) выполнить команду

```
cvs  -d /home/brigades/ pmi7501/cvsroot checkout -l .
```

4. Передача проекта (каталога **project**) и файла **file.c** в хранилище следующими командами:

```
cvs  -d /home/brigades/ pmi7501/cvsroot add project
cvs  -d /home/brigades/ pmi7501/cvsroot add project /file.c
cvs  -d /home/brigades/ pmi7501/cvsroot commit
```

При выполнении команды **commit** будет вызван редактор **vi** для ввода комментария, например: “Пользователь pmi7501передал файл file.c под управление CVS”. После ввода комментария выполняется стандартные действия: ESC,”:”, “wq”. Далее для записи новых версий файла необходимо использовать только команду **commit**,

а команду **add** использовать не надо, так как там уже есть директория **project** и файл **file.c**.

5. Работа в рабочем каталоге (отладка файла **file.c**):

а) получаем исполняемый код путем вызова компилятора **gcc**;

б) исполняемый код запускаем на выполнение; если работает, то переходим на пункт е);

в) получаем исполняемый код для отладчика (**gcc** с ключом -**g**);

г) запускаем отладчик, находим ошибку и выходим из отладчика;

д) запускаем редактор **vi** и исправляем ошибку;

е) исправленную версию записываем в хранилище;

ж) переходим в пункт а).

6. Вывести обзор исправлений в программе **file.c**

```
cvs  -d  ~/cvsroot rdiff  -r 1.1 .
```

7. Вывести журнал изменений версий файла **file.c**

```
cvs  -d  ~/cvsroot log  file.c
```

Упрощенная схема работы CVS представлена на рис. 32.

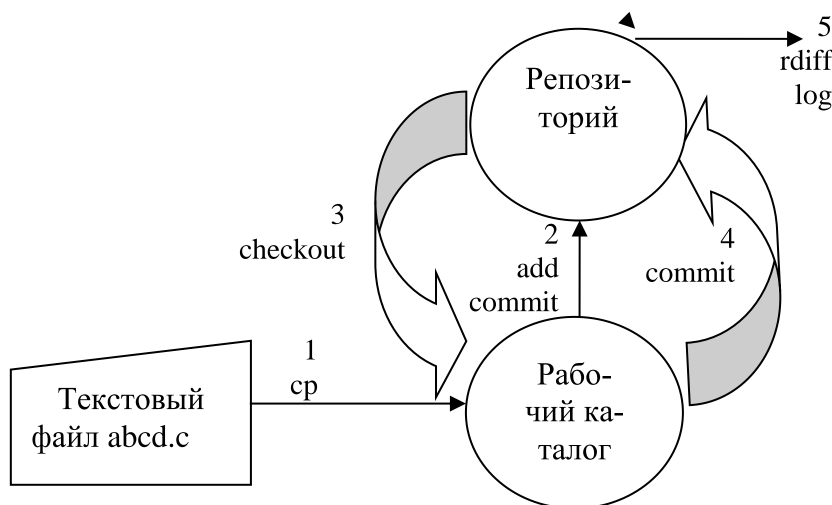


Рис. 32

2.6 Тестовый пример № 1 «Программа простой структуры»

При выполнении лабораторной работы будет использоваться программа **abcd.c**, исходный текст которой приведен ниже. Эта программа является лексическим анализатором, который читает входной текст из стандартного ввода (клавиатуры) и результаты выводит на стандартный вывод (монитор).

Для каждой строки, принятой из стандартного ввода, программа выводит слова по одному в строку. Словом является последовательность алфавитно-цифровых символов, заключенная между пробелами. Если при запуске программы в качестве аргумента задан некоторый символ (например -t), то на стандартный вывод из вводимой строки выводятся только слова, которые включают данный символ.

Приведенный ниже текст программы **abcd.c** содержит несколько синтаксических и семантических ошибок. Предполагается, что при выполнении лабораторной работы студенты должны найти и исправить эти ошибки с помощью инструментов **gcc**, **make**, **gdb**.

Тестовый входной набор данных будет представлен строкой:

this is a test the abcd program

При запуске отлаженной программы **abcd** без параметров после ввода этой строки на экране должно быть:

```
this
is
a
test
the
abcd
program
```

При запуске отлаженной программы abcd с параметром **-t** после ввода этой строки на экране должно быть:

```
this
test
the
```

Исходный текст программы abcd.c

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
/* Manifests for state machine to parse input line. */
#define WORD       0
#define IGNORE   1
/* Globals, used by both subroutines. */
char     *Words[BUFSIZ/2];                 /* Worst case, single letters. */
int      WordCount;

/* Walk through the array of works, find those with the  matching character, printing them on
* stdout. Note that  the NULL character will match all words. */
void PrintWords(wc, match)
```

```
int wc;                          /* Number of words in Words[] */
char match;                     /* Attempt to match this charakter. */
{  register int    ix;           /* Index in Words[]. */
   register char   *cp;        /* Pointer for searching. */
   for (ix=0; ix < wc; ix++) {
            cp = Words[ix];
            /* Try to match the given character. Scan the word, attempting to match,
             * or until the end of the word is found. */
            while ((*cp) && (*cp++ != match));
            if (*cp == match)  /* Found a match? Write the word on stdout. */
                  (void) printf("%s0, Words[ix]); } return; }

/* Find words in the gives buffer. The Words[] array is set
 * to point at words in the buffer, and the buffer modifeid
 * with NULL characters to delimit the words. */
int GetWords (buf)
char     buf[];                        /* The input buffer. */
{        register char    *cp;       /* Pointer for scanning. */
         int      end = strlen(buf); /* length of the buffer. */
         register int  wc = 0;        /* Number of words found. */
         int      state = IGNORE;    /* Current state. */
         /* For each character in the buffer. */
         for (cp = &buf[0]; cp < &buf[end]; cp++) {
            /* A simple state machine to process
             * the current character in the buffer. */
            switch(state) {
            case IGNORE:
                if (!isspace(*cp)) {
                    Words[wc++] = cp;  /* Just started a word? Save it. */
                    state = WORD;      /* Reset the state. */ } break;
            case WORD:
                if (isspace(*cp)) {
                    *cp = '\0';        /* Just completed aword? terminate it. */
                    state = IGNORE;    /* Reset the state. */ } break; }}
          return wc; /* Return the word count. */ }

int  main(argc, argv)  int argc; char *argv[]; { char buf[BUFSIZ], match;
/* Check command line arguments. */
if (argc < 2) match = ' ';
/* No command line argument, match all words. */
else match = *++argv[1];   /* match the char after the first - */
         /* Until no more input on stdin. */
         while(gets(buf) != (char *)NULL) {
             WordCount = GetWords(buf);      /* Paste the input buffer */
             PrintWords(WordCount, match);  /* Print the matching words */ }
         return(0);                                 /* Return success to the shell */
```

2.7 Тестовый пример № 2 «Программа сложной структуры»

В качестве тестового примера используется модифицированная программа **abcd.c** из тестового примера № 1. Для преобразования этой программы в программу сложной структуры необходимо вынести из нее функции **PrintWords** и **GetWords** в отдельные файлы, которые в основной функции **main** должны быть описаны как внешние.

Возможны два способа описания внешних функций – с помощью предложений **extern** или **include**. В первом случае внешняя функция хранится в виде файла типа .c, компилируется отдельно от основной программы и включается в исполняемый файл компоновщиком при сборке программы. Во втором случае функция хранится в файле типа .h и подключается компилятором к основной программе на этапе её препроцессорной обработки.

Ниже приведен пример программы **abcd2.c** с использованием внешней функции **PrintWords**, которая хранится в файле **printwords.c:**

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
/* Manifests for state machine to parse input line */
#define WORD      0
#define IGNORE   1
/* Globals, used by both subroutines. */
char     *Words[BUFSIZ/2];               /* Worst case, single letters. */
int      WordCount;
extern void PrintWords(int wc, char match, char *Words[]);

int GetWords (buf)
char     buf[];                          /* The input buffer. */
{        register char    *cp;        /* Pointer for scanning. */
         int      end = strlen(buf); /* length of the buffer. */
         register int  wc = 0;         /* Number of words found. */
         int      state = IGNORE;  /* Current state. */
         /* For each character in the buffer. */
         for (cp = &buf[0]; cp < &buf[end]; cp++) {
            /* A simple state machine to process
             * the current character in the buffer. */
            switch(state) {
            case IGNORE:
                if (!isspace(*cp)) {
                   Words[wc++] = cp;  /* Just started a word? Save it. */
                   state = WORD;      /* Reset the state. */ } break;
            case WORD:
                if (isspace(*cp)) {
                   *cp = '\0';        /* Just completed aword? terminate it. */
```

```
                    state = IGNORE;    /* Reset the state. */ } break; }}
            return wc; /* Return the word count. */ }

    int  main(argc, argv)  int argc; char *argv[]; { char buf[BUFSIZ], match;
    /* Check command line arguments. */
    if (argc < 2) match = '\0';
    /* No command line argument, match all words. */
    else match = *++argv[1];   /* match the char after the first - */
           /* Until no more input on stdin. */
           while(gets(buf) != (char *)NULL) {
               WordCount = GetWords(buf);  /* Paste the input buffer. */
               PrintWords(WordCount, match, Words); /* Print the matching words. */ }
           return(0);   /* Return success to the shell. */
    }
```

Файл **printwords.c**:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
   void PrintWords(wc, match, Words)
   int wc;                      /* Number of words in Words[] */
   char match;
   char *Words[BUFSIZ/2];                    /* Attempt to match this character. */
   {  register int    ix;          /* Index in Words[]. */
     register char   *cp;         /* Pointer for searching. */
     for (ix=0; ix < wc; ix++) {
             cp = Words[ix];
             /* Try to match the given character.
              * Scan the word, attempting to match,
              * or until the end of the word is found. */
             while ((*cp) && (*cp != match))
                   ++cp;
             if (*cp == match)  /* Found a match? Write the word on stdout. */
                   (void) printf("%s\n", Words[ix]);
  }
  return;
   }
```

При использовании второго способа описания внешняя функция **PrintWords** должна быть сохранена в файле **printwords.h**, для подключения которого в программе **abcd2.c** необходимо использовать директиву #include <printwords.h>. Аналогично можно вынести из главной программы функцию **GetWords**.

В таблице 26 приведены варианты заданий для выполнения тестового примера № 2.

Таблица 26

№ бригады	main()		printwords()		getwords()	
	тип	файл	тип	файл	тип	файл
1	внутренний, исходный	abcd2.c	внешний, исходный	printwords.c	внутренний, исходный	abcd2.c
2	внутренний, исходный	abcd2.c	внешний, исходный	printwords.h	внутренний, исходный	abcd2.c
3	внутренний, исходный	abcd2.c	внешний, объектный	printwords.c	внутренний, исходный	abcd2.c
4	внутренний, исходный	abcd2.c	внутренний, исходный	abcd2.c	внешний, исходный	getwords.c
5	внутренний, исходный	abcd2.c	внутренний, исходный	abcd2.c	внешний, исходный	getwords.h
6	внутренний, исходный	abcd2.c	внешний, объектный	printwords.c	внешний, объектный	getwords.o
7	внутренний, исходный	abcd2.c	внешний, исходный	printwords.c	внешний, исходный	getwords.c
8	внутренний, исходный	abcd2.c	внешний, исходный	printwords.c	внешний, исходный	getwords.h
9	внутренний, исходный	abcd2.c	внешний, исходный	printwords.h	внешний, исходный	getwords.c
10	внутренний, исходный	abcd2.c	внешний, исходный	printwords.h	внешний, исходный	getwords.h
11	внутренний, исходный	abcd2.c	внешний, объектный	printwords.o	внешний, исходный	getwords.c
12	внутренний, исходный	abcd2.c	внешний, исходный	printwords.c	внешний, объектный	getwords.o
13	внутренний, исходный	abcd2.c	внешний, объектный	printwords.o	внешний, объектный	getwords.o
14	внутренний, объектный	abcd2.o	внешний, объектный	printwords.o	внешний, объектный	getwords.o
15	внутренний, объектный	abcd2.o	внешний, исходный	printwords.c	внешний, объектный	getwords.o
16	внутренний, объектный	abcd2.o	внешний, исходный	printwords.c	внешний, исходный	getwords.c
17	внутренний, объектный	abcd2.o	внешний, объектный ный	printwords.o	внешний, исходный	getwords.c

3. Порядок выполнения работы

1. Запустите файловый менеджер **mc**

2. В домашнем каталоге создайте каталоги **cvsroot, workdir** и **examples,** в каталоге **workdir** создайте подкаталог **project**

3. Выполните поиск во внешней памяти сервера файла **testcase.c**

4. Скопируйте файл **testcase.c** в каталог **project** под именем **abcd.c** и занесите в отчет исходный текст программы, предварительно сравнив его с текстом, приведенныв в п.2.6.

5. С помощью редактора **vi** создайте в каталоге **project** make-файл согласно п. 2.3.

6. Выполните компиляцию программы **abcd.c.** с помощью make-файла, используя правило **abcd**. При необходимости исправьте синтаксические ошибки с помощью редактора **vi,** информацию по ошибкам и их устранению занесите в отчет (номер строки, значение строки до устранения и после устранения ошибки, пояснения).

7. В каталоге **cvsroot** создайте репозиторий CVS.

8. Передайте каталог **project** и файл **abcd.c** в репозиторий. При выполнении команды **commit** с помощью редактора **vi** введите комментарий, например: “Пользователь pmi7501передал файл abcd.c под управление CVS”.

9. Запустите исполняемый файл abcd и поясните результат запуска.

10. Перекомпилируйте программу с помощью правила **testabcd** make-файла.

11. С помощью отладчика **gdb** выполните поиск и устранение семантических ошибок в программе **abcd**. Каждое исправление в программе должно сопровождаться записью в репозиторий новой версии с комментарием, поясняющим на русском языке сущность исправлений (например, номер строки программы **abcd.c** и причина исправления). После устранения всех ошибок занесите в отчет результаты тестирования программы в двух вариантах запуска – без параметра и с параметром.

12. После получения корректных результатов выполнения программы **abcd** с помощью редактора **vi** в начало отлаженной программы введите комментарий: "Программа abcd отлажена с помощью отладчика gdb дд.мм.гг. бригадой группы ПМ-XX в составе: ФИО1, ФИО2..." и сохраните в репозиторий финальную версию программы.

13. Выведите список изменений файла **abcd.c**, выполненных в ходе отладки программы, занесите список в отчет.

14. Определите размер исполняемого модуля отлаженной программы. Удалите всю отладочную информацию и снова определите размер исполняемого модуля, сравните с предыдущим результатом, результат сравнения занесите в отчет.

15. Извлеките из репозитория полностью отлаженную программу **abcd** и скопируйте её в каталог **example**, заменив в нем предыдущую версию программы. Все дальнейшие действия будут выполняться в этом каталоге.

16. Выполните разбиение полностью отлаженной программы **abcd** на функции в соответствии с номером бригады из таблицы 26. Обратите внимание на тип функции (внутренняя или внешняя), тип файла (.c, .h или .o) и тип модуля (исходный или объектный). Занесите в отчет измененный текст программы.

17. Выполните сборку программы в соответствии вариантом задания, используя неявный вызов компоновщика и задав имя исполняемого файла **adcd2_1;** проверьте корректность работы программы и занесите в отчет результаты ее тестирования.

18. Выполните поиск во внешней памяти сервера каталога **obj**, скопируйте из него все файлы в каталог **example**, поясните в отчете назначение скопированных файлов.

19. Выполните сборку программы в соответствии вариантом задания, используя явный вызов компоновщика. Результатом сборки должны быть исполняемый файл **abcd2_2** и карта памяти **abcd2_map;** проверьте корректность работы программы и занесите в отчет результаты ее тестирования.

20. Из карты памяти **abcd2_map** определите размеры машинного кода модулей **abcd2.o, printwords.o** и **getwords.o,** сравните их с размерами исходного и объектного кода этих модулей (файлы типа **.c** и **.o**). Результат представьте в виде таблицы 27, все данные должны быть подтверждены скриншотами.

Таблица 27

Имя модуля (функции)	Исходный, байт	Объектный, байт	Машинный код, байт
printwords			
getwords			
abcd2			

21. Добавьте в make-файл, разработанный при выполнении п.5, два новых правила, реализующие п. 17 и 19 задания. Проверьте корректность его работы.

4. Контрольные вопросы

1. Как получить более полную информацию о программах **gcc**, **make**, **gdb** ?

2. Назовите и дайте краткую характеристику основным этапам разработки приложений в ОС Linux?

3. Назначение компилятора, основные этапы компиляции.

4. Назначение программы **make**, структура make-файла.

5. Назовите основное свойство, присущее всем программам отладки. Что необходимо сделать, чтобы его можно было использовать?

6. Основные команды отладчика **gdb**.

7. Опишите по шагам схему отладки программы **abcd.c**, которую Вы использовали при выполнении лабораторной работы.

8. Прокомментируйте реакцию компилятора на синтаксические ошибки в программе **abcd.c** при его первом запуске. Сколько синтаксических ошибок обнаружил компилятор ?

9. Дайте характеристику программе **abcd.c** и объясните сущность семантических ошибок, которые были выявлены вами при выполнении лабораторной работы.

10. Назначение и архитектура CVS.

11. Основные этапы в схеме функционирования CVS.

12. Основные команды CVS, примеры их использования.

13. Что является объектом управления CVS.

14. Понятие и структура репозитория CVS.

15. Как упростить работу с CVS для того, чтобы не набирать в каждой команде путь к репозиторию ?

16. Поясните различия между программами простой и сложной структуры.

17. Понятие внешней функции, способы их описания в языках C и C++.

18. Алгоритм работы компоновщика.

19. Дайте сравнение явного и неявного вызова компоновщика.

20. Карта памяти: назначение и структура.

Лабораторная работа № 7. Технология виртуализации

1. Цель работы

Изучить понятие виртуализации. Приобрести навыки создания виртуальных машин в среде Windows и навыки установки операционных систем Windows и Linux.

2. Методические указания

2.1 Основные понятия

Толковые словари определяют термин “виртуальный” как воображаемый, кажущийся, условный [8]. Виртуализация в вычислениях – это процесс представления набора вычислительных ресурсов или их логического объединения так, чтобы получить какие-либо преимущества перед оригинальной конфигурацией; изоляция вычислительных процессов и ресурсов друг от друга. Такие ресурсы не ограничены реализацией, физической конфигурацией или географическим положением.

В компьютерных системах технология виртуализации используется очень часто, начиная от применения систем управления виртуальной памятью и виртуальными устройствами ввода-вывода в локальных ОС до создания полноценных виртуальных машин, которые могут одновременно работать на одном физическом компьютере (хосте), причем каждая из таких машин может управляться собственной ОС.

Существует две категории виртуализации: виртуализация платформ и виртуализация ресурсов. *Виртуализации платформ* используется для создания программных систем на основе существующих аппаратно-программных комплексов (создание эмуляторов платформ, виртуализация ОС, виртуализация приложений). Продуктом этого вида виртуализации являются *виртуальные машины* – программные или программно-аппаратные системы, эмулирующие аппаратное обеспечение некоторой целевой (target) платформы и исполняющие программы для target-платформы на основной (host) платформе.

Виртуальные машины полезны для тестирования или развёртывания нескольких независимых ОС на одном физическом компьютере. Являясь независимыми от конкретного оборудования объектами, они могут распространяться в качестве предустановленных шаблонов, которые можно запустить на любой аппаратной платформе поддерживаемой архитектуры.

Виртуализация ресурсов используется:

- для объединения или агрегации ресурсов (RAID-массивы и средства управления томами, объединяющие несколько физических дисков в один логический);
- для разделения одного большого ресурса на несколько однотипных объектов, удобных для использования (зонирование ресурсов);
- для кластеризации компьютеров и распределенных вычислений (применяется при объединении множества отдельных компьютеров в метакомпьютеры – глобальные системы, совместно решающие общую задачу).

Целью данной лабораторной работы является изучение виртуализации операционных систем, поэтому далее рассмотрим некоторые виды виртуализации платформ.

Полная эмуляция основана на том, что виртуальная машина полностью виртуализует все аппаратное обеспечение при сохранении гостевой операционной системы в неизменном виде. Этот подход позволяет эмулировать различные аппаратные архитектуры. Например, можно запускать виртуальные машины с гостевыми системами для ARM-процессоров на платформах с другой архитектурой (например, на x86-процессорах) или разрабатывать программное обеспечение для новых процессоров еще до того, как они будут физически доступными. Основной недостаток данного подхода заключается в том, что эмулируемое аппаратное обеспечение существенно замедляет быстродействие гостевой системы.

Примеры продуктов для создания эмуляторов: Bochs, PearPC, Hercules Emulator.

Частичная эмуляция (нативная виртуализация) виртуализует такую часть аппаратного обеспечения, чтобы виртуальная машина могла быть запущена изолированно. Такой подход позволяет запускать гостевые ОС, разработанные только для той же архитектуры, что и у хоста. Несколько экземпляров гостевых систем могут быть запущены одновременно.

В платформах с нативной виртуализацией между гостевыми операционными системами и оборудованием вводится дополнительный уровень, называемый гипервизором или монитором виртуальных машин, который позволяет гостевой системе напрямую обращаться к ресурсам аппаратного обеспечения. Применение гиперви-

зора значительно увеличивает быстродействие платформы, приближая его к быстродействию физической платформы. Этот вид виртуализации широко используется в настоящее время.

Примеры продуктов для нативной виртуализации: Microsoft Hyper-V, VMware Workstation, VMware Server, Virtual Iron, Virtual PC, VirtualBox, Parallels Desktop.

Паравиртуализация использует вместо эмуляции аппаратного обеспечения специальный программный интерфейс (API) для взаимодействия с гостевой операционной системой. Системы для паравиртуализации также имеют свой гипервизор, а API-вызовы к гипервизору называются «hypercalls» (гипервызовы). Применение API дает возможность работать различным виртуальным машинам, не конфликтуя друг с другом. Однако такой подход требует предварительной подготовки гостевой ОС путем незначительной модификации кода ядра.

2.2 Типы гипервизоров

Гипервизор (монитор виртуальных машин) – это программа, обеспечивающая одновременное выполнение нескольких ОС на одном хост-компьютере. Он реализует разделение ресурсов хоста между различными запущенными ОС, изоляцию операционных систем друг от друга, защиту, безопасность и средства обмена данными между разными ОС. Существует два типа гипервизоров.

Гипервизор первого типа (микроядро, тонкий гипервизор или автономный гипервизор) является специфической компактной ОС, которая устанавливается прямо на «железо». Здесь основным понятием является раздел – логическая единица разграничения, в котором работают операционные системы.

Принцип работы основан на том, что после загрузки система создает изолированные друг от друга разделы, в которых запускаются гостевые ОС. Основным является корневой раздел (родительский), в котором работает хостовая ОС. Взаимодействие хостовой системы с гипервизором проводится через соответствующий API. Прямой доступ к аппаратному обеспечению хоста может проводиться только из корневого раздела через соответствующие драйверы. Разделы, в которых запускаются гостевые ОС, называются дочерними.

Способ доступа к аппаратной части из гостевых систем зависит от того, имеет ли данная ОС компоненты, обеспечивающие интеграцию в хостовую ОС. Если такие

компоненты имеются, тогда гостевая ОС может использовать виртуальные устройства, имеющие определенный функционал, предоставляемый сервером виртуализации (провайдером) хостовой ОС, например, виртуальный SCSI-контроллер. Такие устройства называются *синтетическими.* Сервер виртуализации также преобразует запросы от гостевых ОС, передаваемые с помощью клиента виртуализации по специальной шине виртуальных машин VMBus, и переадресует их драйверам физических устройств.

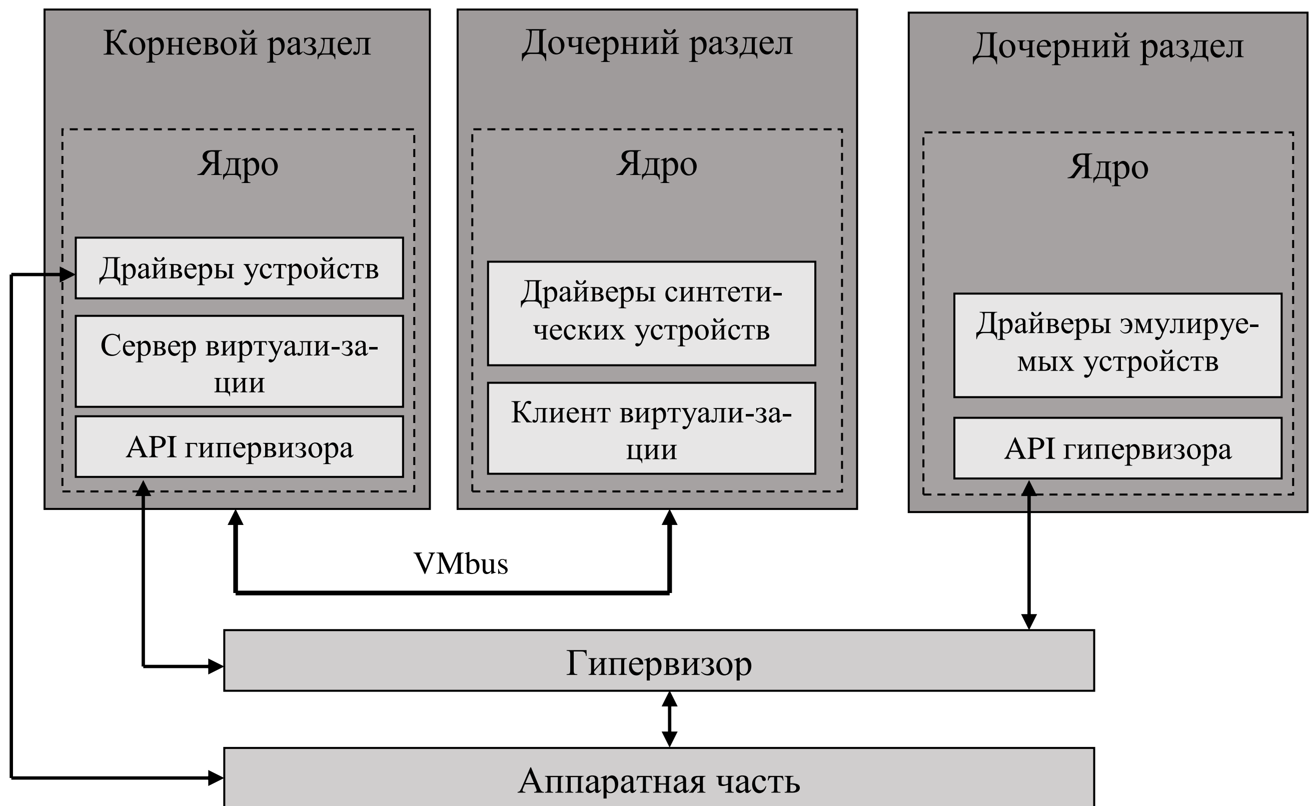


Рис.33

Если гостевая ОС не имеет интеграционных компонентов, тогда используются *эмулируемые* устройства. В этом случае VMBus не используется, а гостевая ОС через драйверы эмулируемых устройств работает с гипервизором напрямую через API. Гипервизор переадресует эти запросы драйверам устройств в корневом разделе через рабочий процесс виртуальной машины, который выполняется не в пространстве ядра, а в пространстве пользователя. Это существенно снижает производительность ВМ, поэтому использовать эмулируемые устройства не рекомендуется.

Архитектура гипервизора первого типа приведена на рис. 33. Такие гипервизоры обычно предъявляют определённый набор требований к аппаратному обеспечению. Например, является обязательным наличие поддержки виртуализации со

стороны процессора и материнской платы. Наиболее популярными гипервизорами первого типа являются Citrix Hypervisor (Xen), VMware ESXi, Microsoft Hyper-V.

Гипервизор второго типа (хостовый гипервизор) представляет собой дополнительный программный слой, расположенный поверх основной операционной системы. Обобщенная структура такого гипервизора приведена на рис. 34.

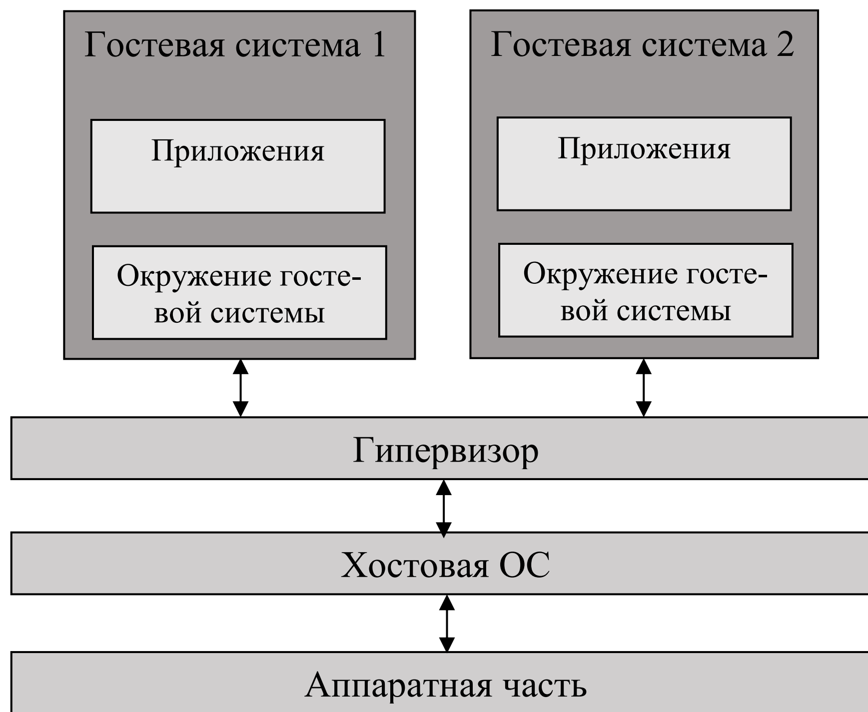


Рис. 34

Фактически гипервизор работает как один из процессов, выполняемых основной ОС. Он управляет гостевыми операционными системами, а управление физическими ресурсами берет на себя хостовая ОС. Наиболее популярные гипервизоры второго типа – Oracle VM VirtualBox, VMware Workstation, KVM.

2.3 Создание виртуальной машины в Oracle VM VirtualBox

Для создания виртуальной машины (ВМ) в Oracle VM VirtualBox необходимо выполнить следующие действия.

1. В окне Менеджера VirtualBox выбрать опцию «Создать» (рис. 35).

2. В окне «Создать виртуальную машину» необходимо нажать «Подробный режим», чтобы получить возможность настройки параметров ВМ (рис. 36). После этого необходимо ввести имя ВМ, ее папку, а также указать объем оперативной памяти и настроить использование жесткого диска (создать новый, использовать существующий или не использовать вообще). Также необходимо выбрать тип и версию ОС, для которой создается машина.

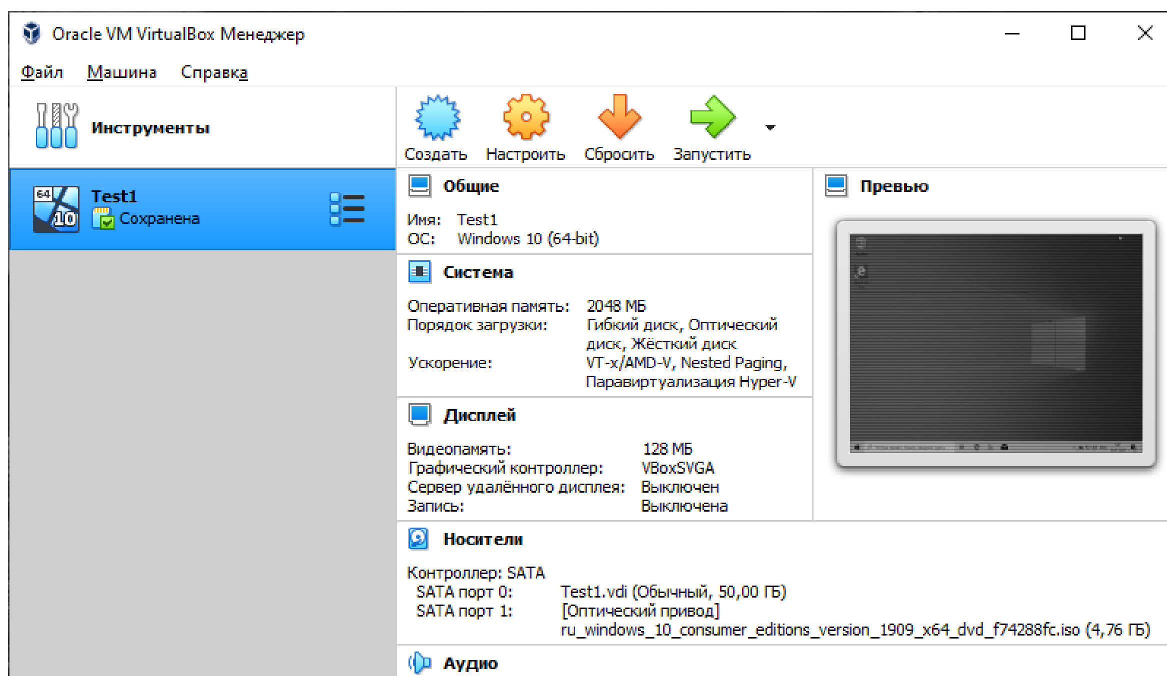


Рис. 35. Окно Менеджера VirtualBox

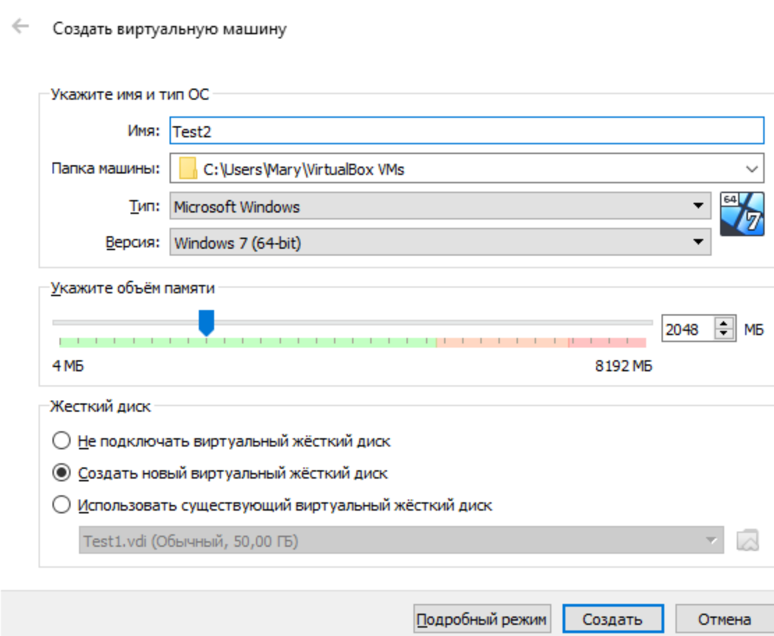


Рис. 36 Окно «Создать виртуальную машину»

3. Если была выбрана опция «Создать новый виртуальный жесткий диск», то после нажатия кнопки «Создать» появится соответствующее окно. В нем необходимо указать расположение, размер, тип и формат хранения жесткого диска (рис. 37).

Существует три основных типа жестких дисков: .vdi, .vhd и .vmdk. Их отличие состоит в том, что .vdi является внутренним типом VirtualBox, .vhd совместим с Microsoft Hyper-V, а .vmdk – с VMware. Формат хранения бывает динамический и фик-

сированный. При выборе фиксированного формата виртуальный жесткий диск будет всегда занимать весь отведенный объем, а при выборе динамического формата размер виртуального диска будет определяться фактическими размерами записанных на него файлов.

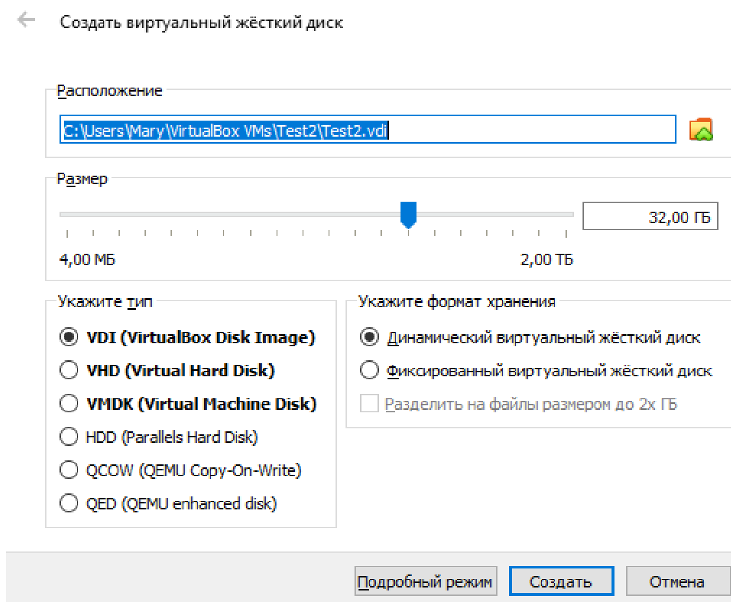


Рис. 37 Окно «Создать виртуальный жесткий диск»

При первом запуске ВМ необходимо выбрать загрузочный диск, с которого будет проводиться установка гостевой ОС. Например, если используется дистрибутив гостевой ОС в формате .iso (образ системы), то надо выбрать оптический привод (рис. 38). Далее необходимо выполнить установку и настройку гостевой ОС.

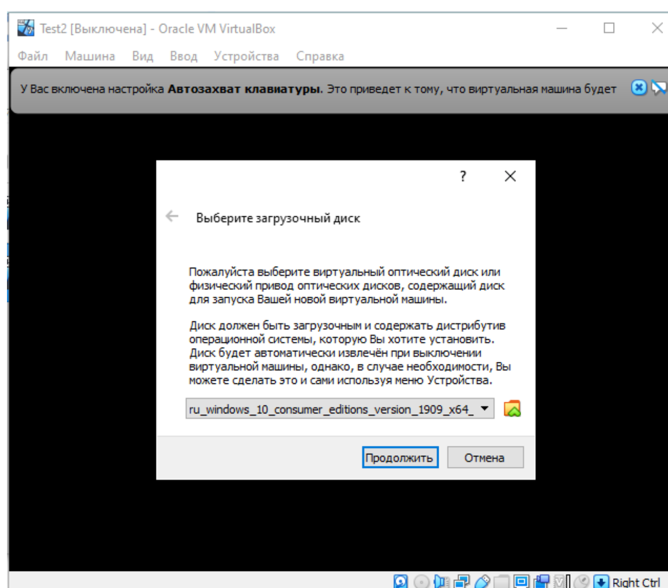


Рис. 38 Окно виртуальной машины

4. В процессе работы с виртуальной машиной часто возникает необходимость обмена данными между гостевой и основной операционными системами. Например, если Вы используете устаревшее оборудование, предназначенное для работы с

Windows XP и для которого разработчик уже не выпускает обновления драйверов, можно в качестве гостевой системы установить Windows XP, подключить это оборудование к этой системе и организовать обмен данными с Windows 10. Механизм обмена основан на использовании общих папок.

Для создания общей папки необходимо перейти в режим настройки, выбрать пункт «Общие папки» и указать имя папки основной ОС, которая будет использоваться как общая папка, путь доступа к ней, и установить режимы автоматического подключения и полного доступа (рис. 39). При необходимости можно создать несколько общих папок.

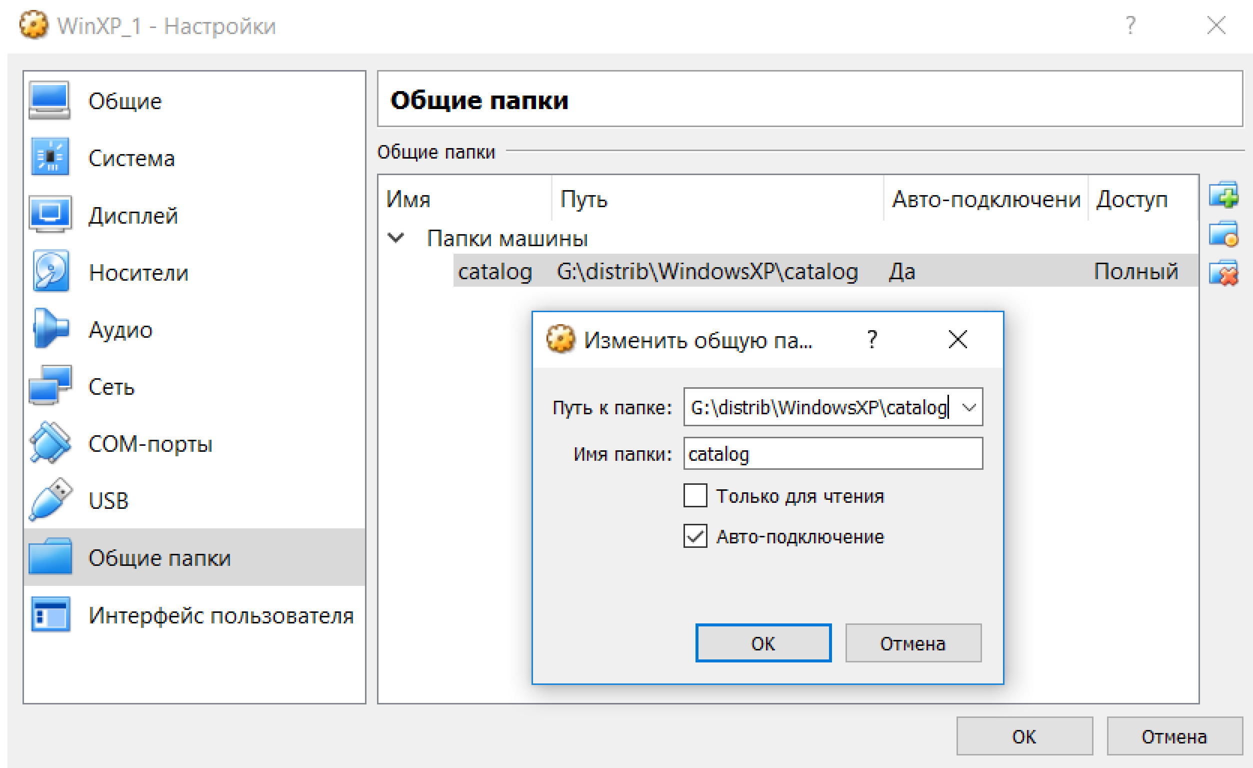


Рис. 39 Создание общей папки

3. Порядок выполнения работы

1. Скачать дистрибутив ОС Windows из личного кабинета студента НГТУ с учетом разрядности Вашего компьютера (Личный кабинет – Учебные ресурсы – Доступ к продуктам Microsoft) в соответствии с требованиями таблицы 28.

2. Скачать и установить среду виртуализации Oracle VirtualBox (сайт разработчика https://www.virtualbox.org).

3. В среде VirtualBox создать ВМ в соответствии с параметрами, заданными в таблице 28. В конфигурации ВМ рекомендуется использовать динамический жесткий диск, оптический диск и оперативную память объемом не более 2 Гб.

Таблица 28

Вариант	Параметры ВЗУ	Гостевая ОС Windows	Гостевая ОС Linux
1	Объем жесткого диска: 50 Гб Тип жесткого диска: .vdi	Windows 10	Tiny CorePlus
2	Объем жесткого диска: 50 Гб Тип жесткого диска: .vhd	Windows 8.1	LXLE
3	Объем жесткого диска: 50 Гб Тип жесткого диска: .vmdk	Windows 10	Linux Lite
4	Объем жесткого диска: 60 Гб Тип жесткого диска: .vdi	Windows 8.1	Lubuntu
5	Объем жесткого диска: 60 Гб Тип жесткого диска: .vhd	Windows 10	Linux Mint
6	Объем жесткого диска: 60 Гб Тип жесткого диска: .vmdk	Windows 8.1	Xubuntu
7	Объем жесткого диска: 55 Гб Тип жесткого диска: .vdi	Windows 10	Debian
8	Объем жесткого диска: 55 Гб Тип жесткого диска: .vhd	Windows 8.1	Bodhi Linux
9	Объем жесткого диска: 55 Гб Тип жесткого диска: .vmdk	Windows 10	AntiX
10	Объем жесткого диска: 65 Гб Тип жесткого диска: .vdi	Windows 8.1	Puppy Linux

4. Установить на ВМ гостевую ОС Windows, загрузив её образ на оптический диск.

5. Запустить ВМ, проверив её функционирование.

6. Загрузить и установить на ВМ прикладное программное обеспечение в соответствии с вариантом в таблице 29. Проверить его функционирование.

Таблица 29

Вариант	**ПО**
1	Mozilla Firefox, WinRar
2	Google Chrome, WinDjView
3	Opera, Putty
4	Mozilla Firefox, notepad++
5	Google Chrome, Adobe Acrobat Reader
6	Opera, WinDjView
7	Mozilla Firefox, Putty
8	Google Chrome, WinRar
9	Opera, notepad++

10	Mozilla Firefox, Adobe Acrobat Reader

7. Создать в хост-системе общую папку, которая будет использоваться для обмена файлами с гостевой ОС. Имя папки должно совпадать с Вашим бригадным логином.

8. Настроить конфигурацию ВМ на работу с общей папкой. Провести обмен файлами между гостевой и основной операционными системами.

9. Завершить работу ВМ.

10. Аналогичным образом создать вторую ВМ с ОС Linux в соответствии с требованиями таблицы 28. Установку прикладного ПО не выполнять.

В отчете необходимо отразить все этапы создания ВМ, установки гостевых ОС и установки прикладного ПО, подтверждая скриншотом каждый этап.

1. Контрольные вопросы

1. Что такое виртуализация, виртуальная машина, платформа виртуализации.

2. Основные виды виртуализации, их особенности.

3. Что такое гипервизор. Типы гипервизоров.

4. Процесс создания виртуальной машины. Основные параметры виртуальной машины.

5. Динамический и фиксированный жесткие диски.

6. Разница между типами жестких дисков .vdi, .vmdk и .vhd.

7. Основные этапы установки ОС Windows.

8. Как организуется обмен данными между гостевой и основной операционными системами.

Список источников

1. Кобылянский В. Г. Операционные системы, среды и оболочки: учебное пособие. – Новосибирск: Изд-во НГТУ, 2018. – 80 с.

2. Кобылянский В. Г. Операционные системы, среды и оболочки: учебное пособие. – Санкт-Петербург:Изд-во Лань, 2020. –120 с.

3. Котельников Е. Введение во внутреннее устройство Windows – [Электронный ресурс] – https://www.intuit.ru/studies/courses/10471/1078/lecture/16586

4. Описание редактора связей GNU ld – [Электронный ресурс] – https://www.opennet.ru/docs/RUS/gnu_ld/gnuld.html#toc1

5. Программирование под Linux – [Электронный ресурс] – http://www.firststeps.ru/linux/general1.html

6. Игнатов В.. Эффективное использование GNU Make – [Электронный ресурс] – http://citforum.ru/operating_systems/gnumake/gnumake_04.shtml

7. CVS – система поддержки версий текстов – [Электронный ресурс] – http://dbserv.pnpi.spb.ru/~shevel/Book/node110.html

8. Jim Blandy. Введение в CVS. Конспект первого дня двухдневного курса по CVS. Перевод на русский язык: Алексей Махоткин. – [Электронный ресурс] – http://citforum.ru/programming/digest/cvsintrorus.shtml

9. Ожегов С.И., Шведова Н.Ю. Толковый словарь русского языка. – Изд-во Азъ, 1992 https://ozhegov.slovaronline.com/

10. Самойленко А. Виртуализация: новый подход к построению IT-инфраструктуры.– https://www.ixbt.com/cm/virtualization.shtml

11. Архитектура Hyper-V.– https://habr.com/ru/post/96822/

12. Mohd Sohail. Лучшие дистрибутивы Linux для старых компьютеров. – https://habr.com/ru/post/458088/