МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Озерский технологический институт — филиал НИЯУ МИФИ

Кафедра прикладной математики

Вл. Пономарев

ПРАКТИКУМ

по операционным системам

Учебно-методическое пособие
по дисциплине «Операционные системы»

Часть 1. Работа в командной строке

2016 г.

УДК 681.3.06
П 56

Вл. Пономарев. Практикум по операционным системам. Учебно-методическое пособие по дисциплине «Операционные системы». Часть 1. Работа в командной строке. Озерск: ОТИ НИЯУ МИФИ, 2016. — 64 с.

В пособии подробно излагается, как выполнять практические работы по дисциплине «Операционные системы». Работы первой части включают в себя работу в командной строке MS-DOS, формирование пакетных файлов, изучение структур файловой системы FAT.

В качестве основного материала при выполнении практических работ пособие предназначено для студентов, обучающихся по направлению подготовки 09.03.01 «Информатика и вычислительная техника» и специальности 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения».

Рецензенты:
1.
2.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

Содержание

0. Общие цели занятий

В ходе практических работ предлагается изучить основы управления операционной системой MS-DOS при помощи командного языка. В этой части работ рассматриваются следующие темы:

1) командный язык MS-DOS;

2) пакетные файлы MS-DOS;

3) файловая система FAT.

На выполнение и защиту работ этой части предположительно отводится 12 академических часов.

Предполагается, что перед выполнением работы студент самостоятельно изучает описание работы и изучаемые команды. Это позволит сэкономить время, отведенное на выполнение работы.

Каждая выполненная работа должна быть защищена.

Для защиты студент готовит отчет, в которые заносятся команды и результаты их выполнения по мере необходимости.

В начале отчета должны быть записаны:

- дата,
- группа,
- фамилия, имя, отчество,
- код работы,
- название работы.

1. OS-101. Работа в командной строке MS-DOS

Цели:
- изучение командного языка на примере MS-DOS.

Задачи:
- изучение спецификации;
- изучение основных команд;
- формирование и вывод текстовых файлов.

1.1. Требования

Для выполнения работы требуется чистый внешний носитель, например, накопитель типа флэш-памяти.

В описании этот накопитель будет обозначен как "X:".

1.2. MS-DOS

MS-DOS (*Microsoft Disk Operating System*) — однозадачная ОС для настольного компьютера (1981 г.).

Эта система состоит из *ядра* и *вспомогательных файлов*. В ядро входят два двоичных файла и программа командного процессора:

IO.SYS — система ввода-вывода;

MSDOS.SYS — файловый процессор;

COMMAND.COM — командный процессор.

Система ввода-вывода предоставляет интерфейс к внешним устройствам компьютера. Файловый процессор предоставляет интерфейс для выполнения основных функций ОС. Командный процессор выполняет команды оператора, введенные с клавиатуры.

Вспомогательные файлы MS-DOS можно поделить на *специальные* и *утилиты*. К специальным файлам относятся файл автозапуска autoexec.bat и файл конфигурации config.sys. Утилиты предназначены для управления и обслуживания компьютера, например

tree — выводит дерево каталогов и файлов,

format — форматирует дисковое устройство,

mode — настраивает внешнее устройство.

Общее количество таких утилит варьируется в разных версиях, и составляет несколько десятков.

1.3. Командная строка

Основное средство управления MS-DOS — командная строка.

Командная строка — это одна или несколько фраз, разделенных пробелами, которые вводятся оператором в ответ на приглашение ОС. На рисунке 1 показано, как выглядит стандартное приглашение ОС, и приведен пример выполненной команды. Приглашение называется PROMPT.

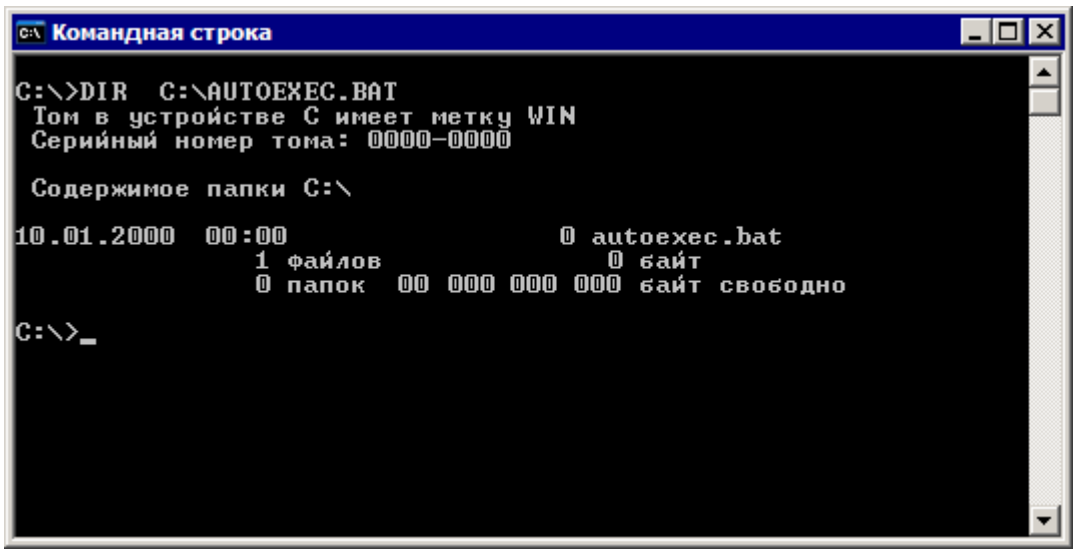


Рисунок 1 - Приглашение командной строки

Как видим, приглашение выглядит в виде строки "C:\>". Небольшой прямоугольник после него — мигающий курсор. В начале рисунка в командной строке введена команда "DIR C:\AUTOEXEC.BAT", а следом за ней показан результат ее выполнения.

Структура команды в общем случае имеет вид:

COMMAND [SPEC] [SPEC] ... [/OPTION] [/OPTION] ...

Здесь:

COMMAND — *команда* (ключевое слово или спецификация),

SPEC — *спецификация*,

OPTION — *модификатор* команды.

Квадратные скобки в описании команды обозначает необязательность того или иного ее элемента. Таким образом, команда может состоять из следующих фраз:

- ключевого слова,
- ключевого слова и спецификации,
- ключевого слова и модификатора,
- ключевого слова и нескольких спецификаций и нескольких ключей,
- других вариантов сочетаний спецификаций и модификаторов.

Конкретный состав команды зависит от команды и цели.

Например, в команде

DIR C:\AUTOEXEC.BAT

DIR — ключевое слово, C:\AUTOEXEC.BAT — спецификация. Модификаторов в этой команде нет.

При вводе команды регистр букв *не имеет значения*.

Поэтому команды "DIR", "DiR", "dir" на самом деле являются одной и той же командой "DIR". Система автоматически переводит все вводимые фразы в требуемый ей регистр.

Спецификации в команде указывают на те объекты, с которыми производится действие. Обычно это файлы и (или) каталоги.

Модификаторы видоизменяют действие команды, и позволяют получить требуемый вывод команды на дисплее.

Модификатору всегда предшествует знак "/" (*прямой слеш*).

Модификаторы часто называют *ключами*, *опциями*, или *переключателями*. Мы будем использовать слово *ключ*.

Пример команды с ключом:

```
DIR /W
```

Здесь ключ /W предписывает вывести содержимое текущего каталога в *широком* (*кратком*) формате. Название ключа происходит от соответствующего английского слова WIDE.

1.4. Файловая структура

Информация в компьютере хранится на *устройствах* в виде *файлов*.

Основной вид устройств — *дисковые* устройства.

Каждому устройству ОС присваивает *имя*. *Имя устройства состоит из латинской буквы и двоеточия*, например, X:.

При этом имена "A:" и "B:" зарезервированы для обозначения дискет.

Таким образом, для других устройств остается 24 имени. Большего количества устройств не может быть (нечем обозначать).

Каждому файлу в системе присваивается имя, состоящее из собственно имени и расширения, между которыми ставится точка.

Например, утилита для форматирования имеет имя format.com.

Здесь format — собственно имя, com — расширение (имени).

Расширение (*extension*) помогает сортировать файлы по назначению, так как оно призвано указывать на тип файла, поэтому иногда расширение называют *типом* файла.

Файлы лучше хранить так, чтобы их можно было легко найти. Если все файлы записать в одно и то же место на устройстве, то при большом количестве файлов получится большая свалка. Если в ранних ОС так делалось, то только потому, что на устройство можно было записать лишь несколько файлов (устройства имели небольшой объем).

В современных системах файлы приписывают к так называемым каталогам (которые сейчас часто называют папками). Группы файлов одного назначения приписываются к одному каталогу, а это дает возможность легко их найти, открыв этот каталог.

Выглядит это так, как будто файлы кладут в каталог, как вещи кладут в ящик, а документы в папки. На этом аналогия заканчивается.

Каталоги устройства образуют *иерархию* в виде *дерева*.

Это означает, что есть некоторый *начальный* каталог, который называется *корневым*.

Все другие каталоги находятся в корневом каталоге, или в каталогах корневого каталога и так далее.

Корневой каталог создается во время форматирования устройства и не может быть удален. Для обозначения корневого каталога иногда используется его имя, "\" (обратный слеш).

Всем другим каталогам присваивается имя, формат которого в точности совпадает с форматом имени файла (собственно имя и расширение).

1.5. Спецификация

Спецификация, — это *запись точного местоположения объекта* (либо файла, либо каталога). ОС использует спецификацию для выполнения операций с файлами или каталогами. Оператор использует спецификацию для указания конкретных файлов или каталогов.

Спецификация состоит из перечня имен. Одно имя от другого отделяется знаком "\" (обратный слеш). Абстрактный пример спецификации:

C:\ALPHA\BETA\GAMMA

Спецификация состоит из *трех* частей:

- *имя устройства* C: (*буква и двоеточие*),
- *путь к объекту* \ALPHA\BETA\ и
- *имя объекта* GAMMA.

Путь к объекту — это имена тех каталогов, в которые надо последовательно «зайти», чтобы найти объект. В этом примере сначала следует «зайти» в каталог ALPHA, находящийся в корневом каталоге \, затем в каталог BETA. В каталоге BETA находится искомый объект GAMMA.

Является ли объект GAMMA файлом или каталогом, *по данной спецификации определить нельзя*.

Любая часть спецификации может отсутствовать, но не все сразу.

Вместо термина *спецификация* иногда используют термин *путь*.

1.6. Имя файла или каталога

Как было сказано, каждый файл или каталог имеет имя.

В MS-DOS для формирования имени есть определенные правила.

Имя формируется в формате 8+3. Это означает, что для собственно имени предусмотрено максимум 8 символов, а для расширения 3 символа.

Всего получается 11 символов, так как точка между именем и расширением операционной системой в файловых структурах не записывается.

Собственно имя является обязательным, а расширение обязательным не является и может отсутствовать. Это справедливо как для файлов, так и для каталогов. Записи имен "MYCAT" и "MYCAT." идентичны.

Для записи имен в MS-DOS очень жесткие правила. Для формирования имени или расширения можно использовать только буквы латинского алфавита от A до Z, цифры от 0 до 9, а также следующие 16 знаков:

Знак	*Название*
-	тире
_	знак подчеркивания
`	левый апостроф (просто апостроф)
'	одинарная кавычка (правый апостроф)
~	тильда
!	восклицательный знак
@	коммерческое эт (от англ. *at*)
#	знак номера (номер)
$	знак доллара (доллар)
%	знак процента (процент)
^	каре
&	амперсанд
()	круглые скобки
{}	фигурные скобки

Найдите все знаки на клавиатуре и выучите названия. Называть знак @ собакой, знак _ нижним подчеркиванием, знак # решеткой или диезом, а знак ^ крышкой допустимо для домохозяйки или диктора телевидения, но не для специалиста в области информационных технологий.

Нельзя использовать следующие имена, так как они зарезервированы операционной системой для обозначения системных устройств.

Имя	*Обозначаемый объект*
CON	консоль (клавиатура и дисплей)
AUX	коммуникационный порт
NUL	абстрактное устройство
PRN	принтер
CLOCK$	системные часы
LPT1…LPT3	три параллельных порта
COM1…COM4	четыре последовательных порта

Здесь всего 12 имен. Их также следует выучить.

Примечание. В современной операционной системе *Windows* действуют другие, более свободные правила для формирования имен. Максимальная длина имени при этом может составлять 255 символов, а максимальная длина спецификации 260 символов. Имена могут включать почти любые знаки, в том числе пробелы. Недопустимыми в именах являются символы:

```
:   \   |   /
```

При этом, если в командной строке нужно указать спецификацию, которая содержит пробелы, то спецификация заключается в двойные кавычки, например, "C:\Program Files". Количество пробелов при этом имеет значение, поскольку каждый пробел является частью имени.

1.7. Правила умолчания

Для работы с объектом операционной системе нужна полная спецификация объекта, включая устройство, полный путь и объект. Иначе система не сможет ничего найти. Однако для оператора крайне неудобно каждый раз вводить полную спецификацию. Поэтому система использует два правила умолчания, которые нужно знать и использовать.

1. Если в спецификации не указано устройство, система подставляет текущее устройство.

2. Если в спецификации не указан путь к объекту, система подставляет путь к текущему каталогу.

Операционная система всегда знает текущий каталог для каждого устройства. Это значительно облегчает формирование команд.

Текущие путь и каталог — те, с которыми в настоящий момент работает (или работал, если устройство не текущее) оператор. Так, когда оператор открывает каталог с помощью проводника, то этот каталог становится текущим каталогом, а устройство, на котором он расположен — текущим диском. То же самое происходит, когда оператор использует программу файловый менеджера, например, FAR Manager.

Если, к примеру, мы хотим вывести содержимое каталога C:\OS, то следует ввести команду DIR C:\OS . Если же в этот момент мы находимся в каталоге C:\OS, то достаточно ввести команду DIR .

Рассмотрим более сложный пример. Пусть мы работали на диске X: в каталоге \OS. Затем мы перешли на диск C:. Чтобы вывести содержимое каталога X:\OS, достаточно ввести команду DIR X: .

1.8. Специальные имена

Система и оператор используют также *специальные имена* для обозначения некоторых важных каталогов.

Имя	*Обозначаемый объект*
\\	начало сетевого пути
\	корневой каталог устройства
.	текущий каталог
..	родительский каталог

1.9. Абсолютная и относительная спецификации

Есть два способа записи спецификации.

Условимся, что *путь к объекту* обозначает перечень имен каталогов перед именем объекта. Например, если есть спецификация X:\A\B\C\D, то путь к объекту \A\B\C\.

Абсолютная спецификация указывает путь от *корневого* каталога.

Относительная спецификация указывает путь от *текущего* каталога.

Отличить их легко. Абсолютный путь к объекту *всегда* начинается со знака обратный слеш "\" (имя корневого каталога). Относительный путь к объекту *никогда* не начинается со знака "\".

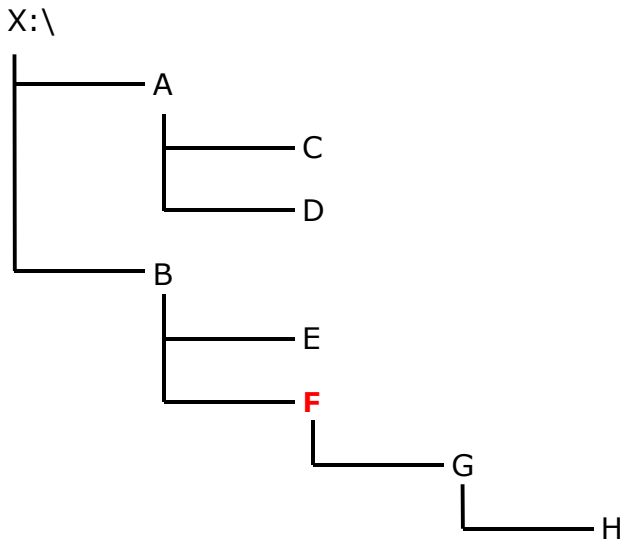
Пример абсолютной спецификации:

C:\WINDOWS\SYSTEM32\FORMAT.COM

Пример относительной спецификации:

C:..\WINDOWS\SYSTEM32\FORMAT.COM

Изучите формирование спецификаций, приведенных на рисунке 2.



Абсолютная	Относительная
X:\	..\..
X:\A	..\..\A
X:\A\C	..\..\A\C
X:\A\D	..\..\A\D
X:\B	..
X:\B\E	..\E
X:\B\F	.
X:\B\F\G	G
X:\B\F\G\H	G\H

Рисунок 2 - Относительная спецификация

Текущий каталог F (выделен красным цветом). Имя родительского каталога (две точки) означает, что нужно сделать шаг наверх по иерархии.

1.10. Исполняемые файлы

Единственная цель использования компьютера — выполнение программ. При этом некоторые программы предназначены для решения насущных задач пользователя, а другие — для управления им. К первым относятся такие программы, как редакторы, игры или системы проектирования, ко вторым — утилиты ОС и разные полезные программы.

В MS-DOS существует только три обязательных расширения:

.COM — команда (небольшая программа),

.EXE — программа (*executable*),

.BAT — пакет команд (*batch*).

Только эти файлы являются исполняемыми файлами, теми, которые можно «запустить». Только эти файлы могут заставить компьютер что-то сделать, чтобы получить какой-то требуемый эффект для пользователя.

Все остальные расширения не являются обязательными. Например, расширение .TXT, предназначенное для обозначения текстовых файлов, не обязательно приписывать к имени текстового файла, это не повлияет на работоспособность системы. Более того, файлы, по сути являющиеся текстовыми, могут иметь и другие расширения, например, файлы XML.

1.11. Внешние и внутренние команды

Все команды делятся на *внутренние* команды и *внешние* команды.

Внутренние команды — это наиболее часто используемые и составляющие основу командного языка команды. Они состоят из одного ключевого слова, а исполняет их непосредственно командный процессор. Количество внутренних команд ограничено. Примерами внутренних команд являются DIR, CD, MD, RD, COPY, REN, DEL, TYPE, которые следует изучить.

Внешней командой называется любой *исполняемый файл*, который доступен системе в данный момент. В этом случае вместо ключевого слова первой фразой команды может быть:

- собственно имя исполняемого файла,
- полное имя (собственно имя и расширение),
- спецификация исполняемого файла.

Например, если утилита MS-DOS, предназначенная для форматирования дискового устройства, и имеющая полное имя format.com, находится в каталоге C:\WINDOWS\SYSTEM32\, то для ее выполнения с дисковым устройством X: могут быть использованы следующие команды:

```
FORMAT X:
FORMAT.COM X:
C:\WINDOWS\SYSTEM32\FORMAT X:
C:\WINDOWS\SYSTEM32\FORMAT.COM X:
```

Когда и какую команду следует использовать, в этом случае зависит от текущей ситуации. Наибольший успех гарантируется при использовании команды C:\WINDOWS\SYSTEM32\FORMAT X:, потому что здесь явно указано местоположение исполняемого файла, и не указан его тип.

1.12. Пути по умолчанию

Для облегчения работы в командной строке в системе задаются пути, в которых следует искать исполняемые файлы тогда, когда оператор не указывает путь к внешней команде.

Операционная система в этом случае ищет исполняемый файл сначала в текущем каталоге, затем последовательно в каталогах, указанных в переменной окружения PATH.

Если при этом не указано расширение файла, то система ищет сначала файл с расширением .com, затем .exe, затем .bat.

Все указанные выше команды будут выполнены, если в переменной окружения PATH задан каталог для поиска "C:\WINDOWS\SYSTEM32".

1.13. Метка тома

Каждое устройство может быть поименовано при помощи так называемой метки тома. Под томом при этом понимается устройство. Метка тома — это строковое имя устройства длиной до 11 символов, которое может содержать пробелы.

1.14. Прерывание выполнения команды

Для остановки команды, которая выполняется долго, с клавиатуры нужно ввести одно из сочетаний (^C читается "Контро́л Цэ"):

Ctrl+C (на дисплее выглядит как ^C)
Ctrl+Break (на дисплее выглядит как ^C)

1.15. Начало выполнения работы

Для выполнения работы на диске C: должен быть рабочий каталог OS.

Этот каталог нужно скопировать из предоставляемого преподавателем пакета заданий на диск C:.

Работа выполняется в командной строке, использование проводника *Windows* или программы файлового менеджера не разрешается, кроме как по прямому указанию преподавателя.

Для работы следует подключить съемное устройство (флешку).

1.16. Командная строка

Открываем окно командной строки (рисунок 1).

При необходимости окно нужно настроить на правильное отображение. Рекомендуемый шрифт *Lucida Console* размером 14, ширина окна 80 символов, высота в зависимости от монитора 25-50, размер буфера 300.

Чтобы уточнить настройки, щелкните правой кнопкой на заголовок окна командной строки, выберите пункт Свойства.

Вводим с клавиатуры команду смены текущего диска. Эта команда имеет вид, совпадающий с именем диска. Чтобы, например, перейти на диск X:, нужно ввести с клавиатуры X:. Заметим, что ввод каждой команды должен завершаться нажатием клавиши Enter.

Сейчас нужно перейти на диск C:, поэтому вводим команду C: .

Далее нужно изменить текущий каталог на \OS.

Для смены текущего каталога используется команда CD (от английских слов *Change Directory*). Команда требует спецификацию каталога, поэтому вводим CD\OS . Убедиться в правильности действий можно, посмотрев на приглашение командной строки. Оно всегда показывает текущий диск и каталог, поэтому сейчас оно должно иметь вид "C:\OS>".

1.17. Команда DIR

Далее все вводимые команды записываем в отчет.

Команда DIR (от английского слова *Directory*) — одна из самых часто используемых команд. Она выводит на указанное устройство (по умолчанию на консоль, то есть на дисплей) содержание заданного в команде каталога.

Каталог — это список записей, в котором одни записи относятся к файлам, а другие записи относятся к каталогам, которые находятся в данном каталоге. Кроме записей, команда DIR выводит дополнительную информацию, например, метку тома, свободный объем и др.

Изучите возможности команды, введя команду HELP DIR.

Команда HELP (*помощь*) выводит справку.

Текущий каталог C:\OS. Введите команду DIR.

Изучите листинг команды (листинг — результат вывода на консоль).

Найдите в листинге:

- метку тома,
- запись, относящуюся к текущему каталогу (с именем точка),
- запись, относящуюся к родительскому каталогу (две точки),
- записи, относящиеся к файлам,
- записи, относящиеся к каталогам,
- общий объем файлов,
- свободный объем устройства.

Введите команду DIR\ (посмотреть корневой каталог).

Запишите в отчет дату создания каталога WINDOWS, общее количество записей в корневом каталоге.

Введите команду DIR. (DIR точка). Запишите в отчет команду и спецификацию выведенного каталога.

Введите команду DIR.. (DIR две точки). Запишите в отчет команду и спецификацию выведенного каталога.

Командой DIR можно вывести не только содержание всего каталога, но и одну запись.

Введите команду DIR \WINDOWS\SYSTEM32\FORMAT.COM. Запишите в отчет команду и размер файла.

Изучите действие следующих ключей команды DIR\WINDOWS:

/A
/OD
/OE
/ON
/OS
/S
/W

Для каждого ключа нужно знать обоснование его происхождения (то есть соответствующее английское слово или слова).

1.18. Команда CD

С помощью этой команды изменяется текущий каталог. Команда требует спецификацию каталога. Полное имя команды CHDIR.

Изучите возможности команды, введя команду HELP CD.

Перейдите в каталог C:\WINDOWS, используя относительную спецификацию, команду запишите. Если не удалось попасть в нужный каталог, вернуться в каталог C:\OS можно с помощью команды CD\OS.

Далее перейдите в каталог C:\WINDOWS\SYSTEM32, используя относительную спецификацию, команду запишите.

Далее перейдите в корневой каталог, команду запишите.

Далее перейдите в каталог C:\OS, команду запишите.

1.19. Команды MD и RD

Команда MD создает каталог. Полное название MKDIR (от английских слов *Make Directory*). Требует *спецификацию*.

Изучите возможности команды, введя команду HELP MD.

Команда RD удаляет *пустой* каталог. Полное название RMDIR (от английских слов *Remove Directory*). Требует *спецификацию*.

Изучите возможности команды, введя команду HELP RD.

Сейчас вам на своем съемном носителе нужно создать структуру каталогов, показанную на рисунке 3.

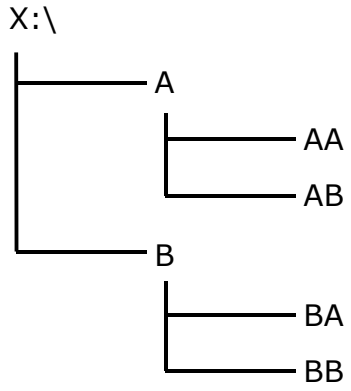


Рисунок 3 - Учебная структура каталогов

Есть два способа создать эту структуру.

Первый способ — создать каталоги, используя абсолютные спецификации, оставаясь при этом в текущем каталоге.

Второй способ кажется более длинным. Несмотря на это, этот способ в некотором смысле более простой.

- создать каталог A,
- сделать его текущим,
- создать каталог AA,
- создать каталог AB,

- перейти в корневой каталог
- создать каталог B,
- сделать его текущим,
- создать каталог BA,
- создать каталог BB,
- перейти в корневой каталог.

Заметим, что на рисунке 3 устройство обозначено как X: . На самом деле ваше съемное устройство получит другую букву, например D: .

Вам нужно сначала создать структуру каталогов первым способом, затем удалить ее при помощи RD, затем создать вторым способом. Текущим каталогом при выполнении этой части работы является X:\ . Команды должны быть записаны в отчет для первого и второго способов.

Важно: после выполнения каждой команды следует убедиться в том, что она выполнена надлежащим образом. Средством контроля текущей структуры каталогов является внешняя команда TREE (дерево). Поэтому после каждой команды, изменяющей структуру каталогов, *обязательно* должна быть введена команда:

TREE X:

Не забываем, что X: — условное обозначение вашего устройства.

1.20. Обозначение группы файлов

Часто требуется указать группу файлов. В этом случае используются так называемые символы-заменители (*wildcards*). Называются они так потому, что заменяют собой часть имени объекта.

Есть два символа-заменителя.

Символ "*" заменяет собой любую последовательность символов имени объекта. Ниже приведено несколько характерных примеров.

Пример	*Группа объектов*
A*	Все объекты, имя которых начинается с A
A*.exe	Все .exe файлы, собственно имя которых начинается с A
A.*	Все объекты, собственно имя которых равно A
A	Все объекты, имя которых содержит A
A.exe	Все .exe файлы, собственно имя которых содержит A
*.exe	Все .exe файлы
.	Все файлы

Символ "?" заменяет собой единственный символ в имени объекта.

Например, запись A??? выбирает группу объектов, имя которых начинается с A, при этом длина полного имени равна 4 символам.

Запись *.? выбирает группу объектов, расширение которых состоит из одного символа, а собственно имя не имеет значения.

1.21. Команда COPY

С помощью этой команды копируется один или несколько файлов.

Команда требует две спецификации — спецификацию файла источника и спецификацию файла приемника.

Файл *источника* — это тот файл, который нужно скопировать (или группа файлов).

Файл *приемника* — это спецификация *файла*, в который будет скопирован файл источника, или спецификация *каталога*, в который будет скопирован файл источника или группа исходных файлов.

Изучите возможности команды, введя команду HELP COPY.

Следующие примеры команды COPY следует выполнить и записать в отчет без контролирующих команд TREE. Структура каталогов приведена на рисунке 3. Текущий каталог B.

Пример 1.

Скопировать файл DA.TXT из каталога C:\OS в текущий.

```
COPY  C:\OS\DA.TXT
```

В этом случае используется одна спецификация, потому что спецификация текущего каталога может быть опущена.

Контроль выполнения команды с помощью команды TREE X:\ /F:

```
X:\
├───A
│   ├───AA
│   └───AB
└───B
        DA.TXT
    ├───BA
    └───BB
```

Пример 2.

Скопировать файл DA.TXT из каталога C:\OS в текущий каталог и присвоить ему другое имя D1.TXT.

```
COPY  C:\OS\DA.TXT  D1.TXT
```

Здесь вторая спецификация может быть сокращена до имени, потому что копируем в текущий каталог, путь можно опустить.

Контроль выполнения команды с помощью команды TREE X:\ /F:

```
X:\
├───A
│   ├───AA
│   └───AB
└───B
        DA.TXT
        D1.TXT
    ├───BA
    └───BB
```

Пример 3.

Дать файлу DA.TXT в текущем каталоге другое имя D2.TXT.

```
COPY  DA.TXT  D2.TXT
```

Здесь обе спецификации сокращены до имени, потому что оба файла находятся в одном каталоге.

Контроль выполнения команды с помощью команды TREE X:\ /F:

```
X:\
├───A
│   ├───AA
│   └───AB
└───B
        DA.TXT
        D1.TXT
        D2.TXT
    ├───BA
    └───BB
```

Пример 4.

Скопировать все .TXT файлы из C:\OS в каталог X:\B\BA.

```
COPY  C:\OS\*.TXT  BA
```

Используется относительная спецификация каталога назначения. Поскольку каталог назначения находится в текущем каталоге, его спецификация опускается до имени.

Контроль выполнения команды с помощью команды TREE X:\ /F:

```
X:\
├───A
│   ├───AA
│   └───AB
└───B
        DA.TXT
        D1.TXT
        D2.TXT
    ├───BA
    │       DA.TXT
    │       DB.TXT
    └───BB
```

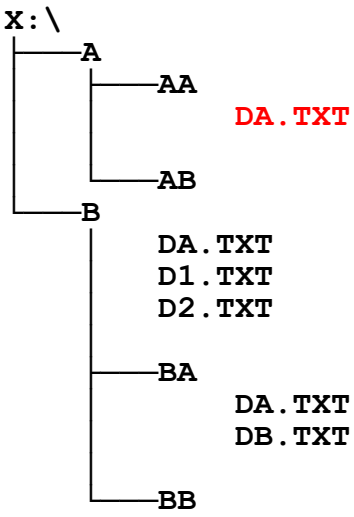
Пример 5.

Скопировать файл DA.TXT из текущего каталога в каталог AA.

```
COPY  DA.TXT  \A\AA
```

Здесь используется тот факт, что оба каталога находятся на одном устройстве, поэтому можно опустить имя диска, а для файла источника можно опустить и путь, поскольку копируем из текущего каталога.

Контроль выполнения команды с помощью команды TREE X:\ /F:

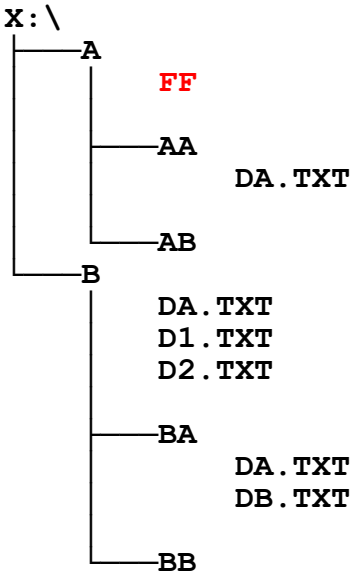


Применение команды такого вида может привести к неожиданным последствиям, если указать несуществующий каталог.

```
COPY  DA.TXT  \A\FF
```

Здесь ошибочно указан несуществующий каталог \A\FF. В результате система решит, что указана спецификация файла назначения, и скопирует файл в каталог \A с именем FF.

Контроль выполнения команды с помощью команды TREE X:\ /F:



Чтобы подобной ошибки избежать, нужно либо *явно указывать* имя файла приемника, либо *явно указывать*, что спецификация является спецификацией каталога.

```
COPY  DA.TXT  \A\FF\
```

Эта команда выполнена не будет, поскольку указана спецификация каталога приемника, который не существует. Слеш в конце спецификации указывает на то, что задан каталог, а не файл.

Далее *самостоятельно* выполните следующие действия. После каждого действия применяйте команду для контроля. Текущий каталог всегда остается прежним: X:\B. Будет оцениваться экономность команды, то есть ее сокращение за счет правил умолчания.

Действие 1.

Файл FF из каталога X:\A скопируйте в каталог BB.

Действие 2.

Все файлы из каталога BA скопируйте в корневой каталог X: .

Действие 3.

Файл X:\A\FF скопируйте в текущий каталог с новым именем D3.TXT.

Действие 4.

Все файлы из каталога BB скопируйте в каталог AB.

Действие 5.

Файл D3.TXT из текущего каталога скопируйте в каталог AB с новым именем D4.TXT.

1.22. Команда REN

При помощи команды REN (*Rename*) можно переименовать файл.

Изучите возможности команды, введя команду HELP REN.

Переименуйте файл D4.TXT в D4.ABC. Команду запишите.

1.23. Команда DEL

При помощи команды DEL (*Delete*) удаляются файлы. Команда требует спецификацию удаляемого файла или группы файлов. При удалении группы файлов команда иногда требует подтверждения Y (*Yes*).

Изучите возможности команды, введя команду HELP DEL.

Пример удаления файла FF из каталога A:

```
DEL  \A\FF
```

Самостоятельно удалите все другие файлы на своем устройстве. Все команды запишите в отчет. Проконтролировать удаление всех файлов можно с помощью команды TREE X:\ /F. Результат выполнения этой команды должен показать, что остались только каталоги.

Удаление всех файлов каталога выполняется, если указана спецификация каталога, а не файла (или группы файлов). Например, все файлы в текущем каталоге удаляются командой DEL. (DEL точка).

1.24. Текстовые файлы

Текстовые файлы имеют большое значение в операционных системах, поскольку с их помощью записываются различные сведения, необходимые для работы системы. Противоположностью текстовых файлов являются двоичные, или бинарные файлы.

Примерами текстовых файлов являются autoexec.bat или win.ini.

Текстовые файлы создаются и изменяются текстовыми редакторами, поэтому в любой операционной системе такой редактор должен быть.

Например, в MS-DOS есть возможность непосредственной записи текстового файла с консоли, а также внешняя команда EDLIN. В *Windows* для редактирования текстовых файлов применяется, как известно, приложение Блокнот (*Notepad*).

Текстовый файл содержит *текстовые символы*. К ним относятся все символы кодовой страницы, начиная с кода 32 (за исключением нескольких, не имеющих отображения), а также несколько символов, код которых меньше 32 (символы с кодом меньше 32 называются *управляющими*, и они предназначены для управления терминалами).

Следующие управляющие символы допустимы в текстовых файлах.

Код	*Обозначение*			
7	BEL	Звонок	*Bell*	^G
8	BS	Забой	*Backspace*	^H
9	TAB	Табуляция		^I
10	LF	Перевод строки	*Line Feed*	^J
11	VT	Вертикальная табуляция		^K
12	FF	Перевод формата	*Form Feed*	^L
13	CR	Возврат каретки	*Carriage Return*	^M
26		Конец текста		^Z

Заметим, что символ с кодом 26 при управлении терминалом имеет другое назначение, но в текстовом файле он завершает текст.

Текстовые файлы состоят из строк. При выводе текстового файла на консоль (дисплей) нужно показать текст в виде строк, поэтому нужен механизм, при помощи которого текст делится на строки. Для этого в текст добавляются управляющие символы.

В разных операционных системах конец строки обозначается разным способом. В MS-DOS в конце строки записывается два управляющих символа: символ возврата каретки CR, затем символ перевода строки LF. Названия этих символов произошли от действий, которые выполняет пишущая машинка во время перехода на следующую строчку.

Эти два символа вставляются в текст автоматически каждый раз, когда оператора вводит (нажимает) клавишу Return (Enter).

При выводе на консоль (дисплей) эти символы не отображаются, вместо этого они перемещают курсор, который показывает то место, в которое будет выводиться очередной отображаемый символ.

Символ BS (звонок) заставляет терминал (компьютер) выдать звуковой сигнал. Символ BS (забой) заставляет курсор переместиться на один символ назад для того, чтобы напечатать еще один символ поверх уже выведенного (например, диакритический знак), или тот же символ для его выделения.

Символ TAB (табуляция) заставляет курсор переместиться к ближайшей справа позиции табуляции и выводить текст с этой позиции. Если позиции табуляции не заданы, то по умолчанию курсор перемещается в позицию (столбец), кратный восьми.

Символ FF (перевод формата) переводит курсор к началу новой страницы (новой формы).

Символ вертикальной табуляции должен перемещать курсор по вертикали вниз, но это обычно нигде и никем не реализуется (сложно).

Использование не текстовых символов в текстовых файлах может привести к неожиданным последствиям. Но, как правило, текстовые редакторы не разрешают вставлять недопустимые символы, хотя другими средствами сделать это можно.

Текстовые файлы отображаются при помощи кодовой страницы (*code page*). В кодовой странице каждому коду назначен символ. Наиболее известной является кодовая страница ASCII или ANSI. Первая половина этих страниц содержит символы латиницы и хорошо подходит для отображения текстов на английском языке.

Первая половина описывает коды 0-127. В этой части находятся:

- символ пробела, имеющий код 32 (0x20),
- символ цифры ноль, имеющий код 48 (0x30),
- символ прописной латинской буквы A, код 65 (0x41),
- символ строчной латинской буквы a, код 97 (0x61),
- другие буквы, цифры и знаки.

От кода 48 легко вычислить код любой цифры. От кода 65 легко вычислить код любой латинской буквы. Прописные и строчные буквы имеют коды, отличающиеся ровно на 32 (на код пробела).

Полностью кодовая страница приведена, например, в ТПМ.

Вторая половина кодовой страницы описывает коды для отображения символов не латинских алфавитов, например, символы кириллицы.

Сейчас мы запишем текстовый файл непосредственно с консоли.

Смысл действия — копировать с консоли в файл.

Сначала вводим команду

```
COPY CON X:\1.TXT
```

Далее начинаем вводить текст:

- вводим сочетание Ctrl+G, которое на экране отобразится как ^G,
- вводим две буквы латинского алфавита AB,
- вводим сочетание Ctrl+I, которое на экране отобразится как перемещение курсора, поскольку это символ табуляции,
- вводим две буквы латинского алфавита CD и Enter,
- вводим 123 и Enter,
- вводим сочетание Ctrl+Z, которое на экране отобразится как ^Z,
- вводим Enter.

Поскольку мы ввели сочетание Ctrl+Z, которое соответствует концу текстового файла, сразу после ввода Enter файл записывается и команда завершается. Заметим, что символ ^Z можно ввести при помощи F6.

Чтобы убедиться в создании текстового файла, выведем его на консоль (дисплей). Для этой цели в MS-DOS предусмотрена команда TYPE:

```
TYPE  X:\1.TXT
```

В результате должны увидеть примерно следующее:

```
AB        CD
123
```

В зависимости от операционной системы и конкретного компьютера мы можем также услышать сигнал динамика, соответствующий символу звонка, который мы ввели в текст первым.

Сейчас нужно открыть программу FAR Manager.

Далее при помощи сочетания Ctrl+F1 сделать так, чтобы скрыть левую панель. Если при этом не видна правая панель, ее можно открыть с помощью сочетания Ctrl+F2.

Далее введем команды

```
X:
CD\
```

В правой панели появится содержимое корневого каталога X:\ .

Перемещая указатель положения (цветная полоска) при помощи клавиш со стрелками, установим его на файл 1.TXT.

Нажмите клавишу F3 (смотреть файл).

Если текст выглядит как текст, нажмите клавишу F4 (изменить способ отображения).

В результате мы должны увидеть так называемый *дамп* (*dump*) — содержимое файла в виде шестнадцатеричных байтов.

Изучите дамп и найдите в нем символы звонка, табуляции, возврата каретки и перевода строки.

Чтобы закрыть просмотр, нажмите клавишу Escape.

Чтобы закрыть FAR Manager, нажмите клавишу F10, а если появится запрос подтверждения, нажмите клавишу Enter.

1.25. Перенаправление ввода и вывода

Перенаправление — это изменение источника или приемника информационного потока, обрабатываемого операционной системой при помощи команд.

Знак > указывает на перенаправление *вывода*, то есть на замену приемника информационного потока.

Два знака >> указывают на то, что информационный поток добавляется к уже имеющейся информации.

Перенаправление *ввода* обозначается знаком < и заменяет источник информационного потока для данной программы.

В качестве примера покажем, как перенаправить вывод команды не на консоль, а в файл, что иногда очень полезно. Команда DIR выводит содержимое каталога на стандартное устройство вывода (консоль). Заменив приемник информации, вывод можно получить в виде файла.

Следующая команда перенаправляет вывод в файл:

```
DIR C:\OS > X:\1.TXT
```

Посмотрим, что получилось, при помощи команды TYPE:

```
TYPE X:\1.TXT
```

Следующая команда добавит информацию в файл:

```
DIR C:\OS\NC >> X:\1.TXT
```

Опять смотрим, что получилось:

```
TYPE X:\1.TXT
```

Самостоятельно:
- перенаправьте вывод команды TYPE в файл 2.TXT.
- перенаправьте вывод команды HELP в файл HELP.TXT.

Команды запишите в отчет.

1.26. Контрольные вопросы

1. Состав MS-DOS.
2. Состав командной строки.
3. Что такое спецификация.
4. Из чего состоит спецификация.
5. Имя диска.
6. Имя файла или каталога (8+3, разрешенные символы).
7. Запрещенные имена.
8. Правила умолчания.
9. Специальные имена.
10. Абсолютная, относительная, полная и неполная спецификации.
11. Это спецификация файла или каталога — X:\FILE.TXT ?
12. Символы замены.
13. Исполняемые файлы.
14. Внешние и внутренние команды.
15. Как выполняются внешние команды.
16. Текстовые файлы. Кодовые страницы.
17. Управляющие символы текстовых файлов.
18. Перенаправление.

2. OS-102. Макропроцессор и пакетные файлы

Цели:
- изучение пакетных файлов MS-DOS;
- закрепление знаний и умений работы в командной строке.

Задачи:
- изучение специальных команд макропроцессора;
- изучение формальных и фактических параметров пакета;
- формирование простейших алгоритмов.

2.1. Требования

Для выполнения работы требуется съемное устройство (флешка).

2.2. Пакетные файлы

Пакетным, или *командным* файлом называется обычный текстовый файл с расширением .BAT, состоящий строк команд, меток, пустых строк, а также из строк, являющихся комментариями. В современных операционных системах пакеты могут также иметь расширение .CMD.

Поскольку пакетный файл является исполняемым файлом, его можно выполнить как обычную внешнюю команду. Выполнением пакета занимается часть командного процессора под названием *макропроцессор*.

При этом команды, записанные в пакетном файле, выполняются одна за другой, а наличие в составе командного языка MS-DOS условных команд и команд для организации циклов позволяет формировать простейшие алгоритмы.

В принципе, команды пакета можно вводить с клавиатуры вручную. Однако это требует присутствия человека возле компьютера, что не всегда возможно и удобно. Пакеты расширяют набор команд операционной системы, они как бы добавляют новые сложные команды.

Создание пакета значительно проще процесса написания программы и не требует привлечения дополнительных средств, — достаточно иметь редактор текстовых файлов, являющийся неотъемлемой частью любой операционной среды. Синтаксис и семантика команд макропроцессора достаточно просты и не требуют от пользователя значительных усилий для изучения, как, например, для изучения языка программирования. Поэтому пакеты — это гибкое, удобное средство для управления компьютером, создания определенной рабочей среды, организации работ.

2.3. Команды пакета

Как уже отмечалось, пакетный файл содержит команды, а также метки, строки комментариев, и пустые строки.

Строка, начинающаяся с REM — строка комментария. Пример:

```
REM Какой-то комментарий
```

Строка, начинающаяся с двоеточия, предполагает наличие метки. Для этого сразу после двоеточия должен следовать идентификатор. Идентификатор — это просто какое-то слово. Все метки в одном пакетном файле должны быть, естественно, разными, с учетом того, что регистр букв значения не имеет. В MS-DOS двоеточие должно находиться в начале строки, а длина идентификатора не должна превышать 8 символов (на самом деле идентификатор может иметь любую длину, но только первые 8 символов имеют значение, то есть распознаются). Пример:

```
:END_OF_BATCH
```

В чистой MS-DOS это метка END_OF_B.

Метки используются для указания строк, на которые можно перейти с помощью команды GOTO METKA (иди на метку).

Команды, которые можно использовать в пакетном файле — это внутренние команды MS-DOS, внешние команды, а также несколько специальных команд для пакетных файлов. GOTO как раз и есть одна из специальных команд.

2.4. Рабочее место

Будем использовать файловый менеджер FAR Manager для выполнения данной работы. Найдем и откроем эту программу.

Нужно настроить FAR так, чтобы панель слева была закрыта, а панель справа была открыта. Тогда в левой части мы будем наблюдать результат выполнения пакетного файла, а в правой части будет находиться пакетный файл, который с помощью FAR легко создавать и редактировать. Получится очень удобно работать, и другого столь удачного варианта нет. FAR — программа, написанная профессионалом для профессионалов.

Мышь для работы нам не требуется, поскольку FAR отлично управляется клавиатурой, поэтому мышь переверните и накройте чем-нибудь.

Чтобы закрыть или открыть левую панель FAR, следует ввести сочетание клавиш Ctrl+F1, соответственно, Ctrl+F2 управляет правой панелью. Заодно запомните и другие полезные сочетания для управления окнами:

Ctrl+U — поменять панели местами.

Ctrl+O — скрыть или показать панели. Если вы пытаетесь показать панели, но ничего не выходит, скорее всего, обе панели выключены при помощи сочетаний Ctrl+F1 и Ctrl+F2.

Alt+F1 меняет устройство в левой панели.

Alt+F2 меняет устройство в правой панели.

В один момент времени активна только одна панель. Если обе панели открыты, перейти из одной панели в другую можно с помощью Tab. Если одна панель открыта, то она и есть активная. Выбранный в активной панели каталог является текущим.

Чтобы открыть каталог, нужно переместить указатель (цветную полоску) клавишами управления курсором на соответствующую строчку и нажать клавишу Enter. Чтобы выйти из каталога, нужно установить указатель на строку из двух точек и нажать клавишу Enter.

Чтобы выполнить команду MS-DOS, ее нужно просто ввести и нажать Enter, как и обычно в MS-DOS. На самом деле, находясь в FAR, вы находитесь в командной строке.

Чтобы выполнить команду, которую уже когда-то выполняли, нужно выбрать ее, вводя сочетание Ctrl+E (движение назад по списку) или Ctrl+X (движение вперед по списку). Другим способом является сочетание Alt+F8, которое открывает панель со списком команд (история команд), которые запоминаются в огромном количестве (почти все).

Команду можно редактировать непосредственно в командной строке, в командной строке можно выделять, вырезать, копировать и вставлять.

2.5. Создание пакетного файла

Прежде нужно создать каталог для работы.

Место для каталога — диск C:, название каталога — BAT.

Сейчас правая панель открыта и активна.

Выбираем диск C:.

Либо с клавиатуры команда C:, либо Alt+F2, C.

Переходим в корневой каталог сочетанием Ctrl+\.

Создать каталог в FAR легко.

Нажмите клавишу F7, СРАЗУ наберите BAT и нажмите Enter. Если нажмете Enter два раза, то не только создадите каталог, но и сразу откроете его (войдете в него).

Текстовый файл создает сочетание Shift+F4.

Сразу после этого наберите название первого пакетного файла A.BAT и нажмите Enter ОДИН РАЗ.

После этого вы окажетесь в редакторе. Введите слово REM.

Выйти из редактора — Escape. Если появится подтверждение сохранения текста, нажмите Enter чтобы сохранить текст. Если не хотите сохранять, клавишами управления курсором выберите «Нет» и нажмите Enter.

Сейчас выйдите из редактора и убедитесь, что файл создан. Вы должны видеть его в правой панели. Выберите его (установите на него указатель — цветную полоску).

Если файл создан, редактор открывает F4, а также Shif+F4. Первый вариант проще, поэтому F4.

2.6. Команда ECHO

Для вывода информации из пакетного файла на стандартное устройство (консоль) используется команда ECHO.

Давайте сразу попробуем, как это работает.

Редактор открыт, пишем строчку пакетного файла:

```
ECHO HELLO ПРИВЕТ
```

Важное замечание: в конце каждой строки нажимайте Enter, чтобы в файле формировались управляющие символы CR LF. Недопустимо просто написать эту строчку и не нажать Enter. С другой стороны, следите за тем, чтобы в конце файла не было лишних пустых строк. Чтобы перейти в конец файла, введите сочетание Ctrl+End, End — одна из клавиш управления курсором. Соответственно, Ctrl+Home — перейти в начало файла.

Закройте редактор Escape, Enter.

Установите указатель на файл A.BAT.

Чтобы запустить исполняемый файл, нужно нажать Enter.

Нажмите Enter, и пакет исполнится.

При этом на экране вы должны увидеть HELLO ПРИВЕТ.

Если нет, то есть если вместо слова ПРИВЕТ вы видите какие-то непонятные символы (крокозябры), это означает, что вы не угадали с кодировкой, то есть с кодовой страницей.

Откроем редактор для пакета A.BAT. Для тех, кто еще не запомнил, как это сделать, напоминаю: указатель на файл пакета, F4.

Обратим внимание на строку вверху редактора (рисунок 4).

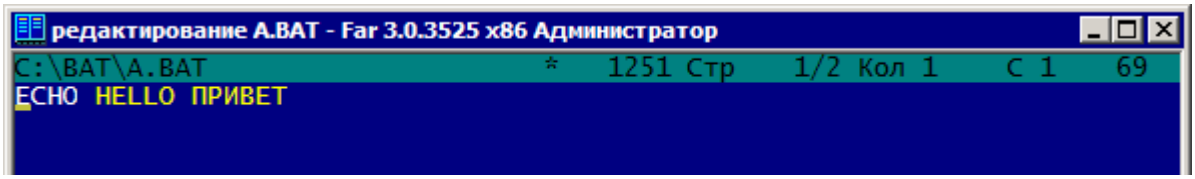


Рисунок 4 - Строка состояния FAR

В ней * показывает, что файл не сохранен (был изменен).

1251 указывает на текущую кодовую страницу.

Стр 1/2 показывает, что курсор в строке 1, в файле 2 строки.

Кол 1 показывает, что курсор в первой колонке.

С1 показывает, что это первый символ в строке.

69 показывает десятичный код символа над курсором.

Нам нужно изменить кодовую страницу.

Вводим F8 несколько раз и смотрим, как меняется кодовая страница.

Можно выбрать любую другую страницу, если ввести Shift+F8.

Если сейчас кодировка 1251, и она не совпадает с командной строкой, значит нужно выбрать кодировку 866 (или наоборот). При этом нужно переписать русское слово, потому что теперь в файле появятся крокозябры.

Есть и другой путь смены кодировки без переписывания.

Вернуть кодировку, при которой русское слово отображается.

Выделить весь текст Ctrl+A.

Удалить весь текст Shift+Delete (аналог Ctrl+X).

Сменить кодировку F8.

Вставить текст Shift+Insert (аналог Ctrl+V).

Выйти и сохранить Escape, Enter.

Снова запускаем пакет и убеждаемся, что в консоли слово русское.

Продолжаем изучение команды ECHO.

Чтобы вывести пустую строку из пакетного файла на консоль, нужно сразу после команды ECHO поставить точку.

Редактируем пакет следующим образом:

```
ECHO HELLO ПРИВЕТ
ECHO.
ECHO .
ECHO END
```

Сохраняем и закрываем редактор.

Сейчас нам нужно сделать перенаправление в файл. Для этого нужно написать команду A.BAT > 1.TXT. Делается это так.

Указатель FAR установлен на пакетном файле.

Вводим сочетание Ctrl+Enter. В результате название файла переносится в командную строку, но запуска не происходит.

Дописываем остаток команды (> 1.TXT) и Enter.

В панели FAR появится файл 1.TXT. Откроем его F4. Если не собираетесь редактировать, открыть можно F3, и далее, возможно, F4, чтобы изменить дамп на текст.

Вероятно, понадобится опять выбрать кодировку.

Результат примерно такой:

```
C:\BAT>ECHO HELLO ПРИВЕТ
HELLO ПРИВЕТ

C:\BAT>ECHO.


C:\BAT>ECHO .
.

C:\BAT>ECHO END
END
```

Ничего не понятно, я бы сказал.

Это все из-за *эха* команд.

2.7. Управление эхо-выводом

Эхо-вывод — это повторение каждой команды в консоли, так, как будто бы их вводили с клавиатуры. Он полезен во время отладки пакетного файла, потому что показывает, какие команды выполнялись.

Для нормальной же работы пакетного файла эхо-вывод обычно отключают. Есть два способа отключить эхо-вывод.

Первый способ — отключить эхо для одной команды. Для этого перед командой нужно поставить знак @ (*эт*). Пробуем:

```
@ECHO HELLO ПРИВЕТ
@ECHO.
@ECHO .
@ECHO END
```

Чтобы повторно запустить команду с перенаправлением, Alt+F8.

Если теперь посмотреть файл 1.TXT, увидим:

```
HELLO ПРИВЕТ

.
END
```

Понятно, что ставить перед каждой командой @ не очень удобно.

Команда ECHO не только выводит текст, но и переключает режим.

ECHO OFF *выключает* эхо-вывод.

ECHO ON *включает* эхо-вывод.

ECHO безо всего также *включает* эхо-вывод.

Редактируем A.BAT, добавляем команду в начало, убираем @:

```
ECHO OFF
ECHO HELLO ПРИВЕТ
ECHO.
ECHO .
ECHO END
```

Теперь тоже не все хорошо. Сама команда ECHO OFF выводится, поэтому нужно подавить ее эхо-вывод:

```
@ECHO OFF
ECHO HELLO ПРИВЕТ
ECHO.
ECHO .
ECHO END
```

Снова пробуем вывести в файл, и убеждаемся, что вывод не содержит никаких команд.

2.8. Параметры пакетного файла

Пакетный файл является, в сущности, программой, реализующей некоторый, достаточно простой алгоритм. Мощь использованию пакетов дает возможность задавать параметры, схожие с параметрами функций.

Как мы знаем из курса программирования, параметры бывают фактические и формальные. Формальные параметры — это те переменные, которые используются в программе. Фактические параметры указываются во время вызова функции, и они передаются формальным параметрам.

В пакетных файлах *фактические* параметры задаются в *командной* строке после названия пакетного файла. Фактические параметры принимаются внутри пакетного файла в специальные, заранее имеющиеся переменные, имеющие вид %N, где N — одна цифра. То есть внутри пакетного файла есть 10 переменных %0, %1, %2, %3, %4, %5, %6, %7, %8, %9.

Поскольку пакетный файл — это все же не программа, объявлять какие-либо переменные в нем нельзя. Поэтому необходимые для выполнения переменные формируются с привлечением знака %.

Редактируем файл A.BAT:

```
@ECHO OFF
ECHO "%0"
ECHO "%1"
ECHO "%2"
ECHO "%3"
ECHO "%4"
ECHO "%5"
ECHO "%6"
ECHO "%7"
ECHO "%8"
ECHO "%9"
```

Пробуем запустить пакет, и убеждаемся, что все переменные пусты.

Теперь пакет запустим с параметрами (параметров десять, первая буква A — название пакетного файла A.BAT, *не надо* использовать Ctrl+Enter):

```
A 1 2 3 4 5 6 7 8 9 10
```

На дисплей выводится 9 параметров, от 1 до 9, параметр 10 не выводится. Вместо параметра %0 выводится спецификация самого пакетного файла C:\BAT\A.BAT (в дополнительных кавычках).

Должен автоматически появиться вопрос: как получить параметр 10?

2.9. Команда SHIFT

Эта команда без параметров сдвигает формальные параметры относительно фактических вправо. При этом прежние значения формальных параметров, естественно, теряются, заменяются новыми.

Для нашего примера, до команды SHIFT соотношение между фактическими и формальными параметрами такое:

```
 A  1  2  3  4  5  6  7  8  9 10
%0 %1 %2 %3 %4 %5 %6 %7 %8 %9
```

После применения SHIFT соотношение будет:

```
 A  1  2  3  4  5  6  7  8  9 10
   %0 %1 %2 %3 %4 %5 %6 %7 %8 %9
```

И, таким образом, параметр 10 станет доступен.

Заметим, что в чистой MS-DOS команда SHIFT не имеет параметров.

В современных операционных системах параметр команды SHIFT указывает, на какое количество позиций выполнить сдвиг.

Редактируем пакет A.BAT (дописываем две команды):

```
@ECHO OFF
ECHO "%0"
ECHO "%1"
ECHO "%2"
ECHO "%3"
ECHO "%4"
ECHO "%5"
ECHO "%6"
ECHO "%7"
ECHO "%8"
ECHO "%9"
SHIFT
ECHO "%9"
```

Пробуем запустить с параметрами как прежде, при помощи Alt+F8, и убеждаемся, что теперь все параметры выводятся.

2.10. Разбор спецификации

Пакетные файлы дают возможность извлекать информацию из переменных, значениями которых являются спецификации. (Эта возможность не предусмотрена в чистой MS-DOS.)

Для изучения этих возможностей введите команду HELP CALL.

Как мы только что выяснили, переменная %0 содержит спецификацию запущенного пакетного файла. Будем использовать ее для извлечения информации с помощью операций подстановки ~буква.

Для начала извлечем информацию об устройстве.

Редактируем пакет A.BAT (~d — извлечь имя устройства, от *device*):

```
@ECHO OFF
ECHO %~d0
```

Обратим внимание, что здесь информация извлекается из переменной %0, однако извлечь информацию можно из любой переменной, заменив 0 на соответствующий номер от 1 до 9.

Запускаем пакет и убеждаемся, что выводится C: .

Заметим, что значение переменной не изменяется.

В этом легко убедиться, добавив еще одну строчку в пакет:

```
@ECHO OFF
ECHO %~d0
ECHO %0
```

Далее вам предлагается поэкспериментировать самостоятельно, используя другие похожие конструкции, приведенные в справке команды CALL. Все исследованные конструкции должны быть записаны в отчет.

2.11. Переменная ERRORLEVEL

ERRORLEVEL — это специальная переменная пакетного файла, которой присваивается *код завершения* последней команды.

Прежде нужно понять, что такое код завершения.

Для этого нам потребуется создать проект на языке C++, в котором этот код можно сформировать.

Открываем *Visual Studio* и создаем проект с названием MAKEERR. Тип проекта — *консольное* приложение. Следите за тем, что проект располагался на диске C: в корневом каталоге. Требуется создать пустой проект.

После создания проекта добавим в него новый модуль, для чего можно ввести сочетание Ctrl+Shift+A. В диалоге *Add New Item* выбираем тип добавляемого модуля .cpp, вводим название модуля main.cpp.

Появится новый модуль, создадим в нем основную функцию:

```
// main.cpp
// тестирование кода завершения
#include <stdlib.h>
int main(int argc, char * args[]) {
    if (argc < 2) return 255;
    int errorlevel = atoi(args[1]);
    return errorlevel;
}
```

Код достаточно простой.

Сначала проверяется количество аргументов программы. Если оно меньше двух, при запуске команды параметры не были заданы, и в этом случае мы возвращаем значение 255.

В противном случае пытаемся преобразовать первый введенный параметр в целое число, результат преобразования возвращаем.

Компилируем программу, обязательно запускаем при помощи Ctrl+F5, чтобы убедиться, что она работает.

Переходим в программу FAR. Рабочим каталогом является C:\OS. Если в правой панели сейчас не этот каталог, нужно зайти в него.

Открываем левую панель FAR при помощи Alt+F1, выбираем устройство C: (для этого достаточно просто нажать клавишу с буквой C). Если левая панель неактивна, переходим в нее при помощи клавиши Tab. Открываем папку MAKERR, в ней папку Debug, устанавливаем указатель на файл MAKERR.EXE, нажимаем F5, Enter. В результате файл нашей программы скопируется в каталог C:\OS, что нам и нужно.

На самом деле все программы возвращают код завершения. Этот код передается операционной системе для того, чтобы сообщить, как успешно или неуспешно программа завершила свою работу. При этом принято считать возвращаемое значение 0 как признак успешного завершения, а другие значения как признак неуспешного завершения. Кроме того, для обозначения определенных ситуаций, связанных с файлами, каталогами, правами доступа и других, принято возвращать вполне определенные коды.

Нам нужно найти файл winerror.h (или winerr.h), в котором описаны все коды ошибок *Windows*.

Для поиска используем FAR.

Закрываем левую панель при помощи Ctrl+F1.

Выходим в корневой каталог при помощи Ctrl+\.

Заходим в папку Program Files.

Вводим сочетание Alt+F7.

Появится диалог для поиска, в котором в самое первое поле нужно ввести название искомого файла winer*.h, и нажать Enter.

Когда файл будет найден, введем F4. Открываем файл для редактирования, однако не надо его изменять.

Найдите описание ошибки с номером 2:

```
// MessageId: ERROR_FILE_NOT_FOUND
//
// MessageText:
//
//  The system cannot find the file specified.
//
#define ERROR_FILE_NOT_FOUND             2L
```

Код 2 возвращается в случае, если программа не может найти заданный файл.

Ошибка с номером 3:

```
// MessageId: ERROR_PATH_NOT_FOUND
//
// MessageText:
//
//  The system cannot find the path specified.
//
#define ERROR_PATH_NOT_FOUND             3L
```

Код 3 возвращается в случае, если путь, заданный в спецификации, не существует.

Ошибка с номером 5:

```
// MessageId: ERROR_ACCESS_DENIED
//
// MessageText:
//
//  Access is denied.
//
#define ERROR_ACCESS_DENIED              5L
```

Код 5 возвращается в случае, если не хватает прав доступа.

Закроем файл Escape.

Программа на языке C возвращает код завершения в виде возвращаемого значения основной функции main(), как вы теперь понимаете. Многие программы возвращают код завершения, в том числе и утилиты MS-DOS, однако не все они возвращают коды, соответствующие файлу winerror.h. Вот в чистой MS-DOS утилиты возвращают правильные коды.

Вернемся в рабочий каталог C:\OS.

Создадим новый пакетный файл EL.BAT. Напоминаю, что для создания нового текстового файла вводится Shift+F4.

В этом пакете мы вызываем свою программу и смотрим, чему равен код ее завершения:

```
@ECHO OFF
MAKEERR
ECHO %ERRORLEVEL%
```

Запускаем пакет и убеждаемся, что выводится 255, что и ожидалось.

Снова открываем пакет для редактирования F4, и добавляем к команде MAKEERR параметр 0:

```
@ECHO OFF
MAKEERR 0
ECHO %ERRORLEVEL%
```

Запускаем пакет и убеждаемся, что выводится 0.

Можно использовать параметр пакетного файла.

Снова открываем пакет и изменяем следующим образом:

```
@ECHO OFF
MAKEERR %1
ECHO %ERRORLEVEL%
```

Теперь команду нужно ввести с клавиатуры:

```
EL 2
```

Убеждаемся, что выводится 2.

Попробуем проверить, какой код завершения возвращают внутренние команды и утилиты MS-DOS. Редактируем пакет EL.BAT:

```
@ECHO OFF
REM MAKEERR %1
DIR %1
ECHO %ERRORLEVEL%
```

Для проверки вводим команду

```
EL C:
```

Затем вводим команду, задавая несуществующий объект:

```
EL QWE
```

Редактируем EL.BAT. Пробуем другую команду:

```
@ECHO OFF
REM MAKEERR %1
MD %1
ECHO %ERRORLEVEL%
```

Для проверки вводим команду

```
EL \OS
```

Заметим, что для использования ERRORLEVEL нужно всегда уточнять, какие коды завершения возвращает команда.

Самостоятельно проверьте, какой код возвращает команда

```
FORMAT O:
```

Здесь O: — имя устройства, которое не должно существовать.

2.12. Команда IF

Команда IF позволяет проверить *три условия*:

- равенство строк;
- наличие файла или каталога;
- значение переменной ERRORLEVEL.

Использование оператора NOT инвертирует эти условия.

Сначала посмотрим, как проверяется переменная ERRORLEVEL.

Редактируем пакет EL.BAT:

```
@ECHO OFF
MAKEERR %1
IF ERRORLEVEL 100 GOTO 100
ECHO %ERRORLEVEL%
GOTO END
:100
ECHO GOTO
:END
```

Здесь IF проверяет, что ERRORLEVEL равно 100, и в случае истинности условия выполняется переход на метку 100, в которой выводится слово GOTO, сигнализирующее о выполнении условия.

Если условие не выполняется, выводится код завершения.

Запускаем пакет без параметров:

```
EL
```

Запускаем пакет с параметром 100, затем с параметром 99:

```
EL 100
EL 99
```

Таким образом, условие проверки IF ERRORLEVEL истинно, если код завершения больше или равен 100.

Редактируем пакет EL.BAT таким образом, чтобы проверялось условие истинности двух строк:

```
@ECHO OFF
MAKEERR %1
IF "%ERRORLEVEL%" == "100" GOTO 100
ECHO %ERRORLEVEL%
GOTO END
:100
ECHO GOTO
:END
```

Здесь переменная ERRORLEVEL проверяется как переменная, а не как аргумент команды IF, поэтому ее нужно заключить в проценты.

Запускаем пакет с параметром 100, затем с параметром 99:

```
EL 100
EL 99
```

Алгоритм действия пакета изменился. Теперь пакет проверяет точное соответствие ERRORLEVEL заданному значению.

Будем теперь выяснять, как работает команда IF, проверяя наличие файла или каталога. Для работы нам понадобятся:

- существующий файл — C:\OS\DA.TXT,
- несуществующий файл — C:\OS\DA.TX,
- существующий каталог — C:\OS\,
- несуществующий каталог — C:\OS9\.

В наличии или отсутствии этих объектов следует убедиться.

Создадим новый пакетный файл EXIST.BAT.

Если объект существует, пакет печатает на консоль YES, иначе NO:

```
@ECHO OFF
IF EXIST C:\OS\DA.TXT GOTO 1
ECHO NO
GOTO END
:1
ECHO YES
:END
```

Запускаем пакет, убеждаемся, что выводится YES.

Сохраним этот пакет с именем EXIST-1.BAT. В FAR для этого нужно ввести сочетание Shift+F5, изменить название и нажать Enter.

В современных операционных системах возможен и более структурированный синтаксис оператора IF. Попробуем изменить пакет EXIST.BAT:

```
@ECHO OFF
IF EXIST C:\OS\DA.TXT (
    ECHO YES
) else (
    ECHO NO
)
```

Запускаем и убеждаемся в том, что выводится YES, или в том, что в пакете обнаружена синтаксическая ошибка. В последнем случае нужно вернуться к предыдущей версии пакета.

Для того, чтобы проверить несуществующий файл, нужно удалить последнюю букву спецификации файла. Затем запускаем пакет и убеждаемся, что выводится NO.

Пробуем таким же образом проверить существующий каталог, для чего редактируем пакет EXIST.BAT, если структурированный оператор IF работает, или пакет EXIST-1.BAT, если не работает.

Если пакет со структурированным оператором IF:

```
@ECHO OFF
IF EXIST C:\OS\NUL (
    ECHO YES
) else (
    ECHO NO
)
```

Убеждаемся, что выводится YES. Редактируем пакет, добавляя цифру к имени каталога, и убеждаемся, что выводится NO.

Заметим, что в современных операционных системах указание имени несуществующего файла NUL не требуется, в этом вам предлагается убедиться самостоятельно. Однако рекомендуется в случае проверки каталога завершать его спецификацию знаком обратного слеша, чтобы избежать неоднозначности.

2.13. Переменные окружения

Переменные окружения — это строки вида «Имя=значение», составляющие в совокупности *блок окружения*, который придается исполняемой программе. Программы могут извлекать значения переменных из блока окружения для получения необходимой информации.

Например, командная строка MS-DOS использует переменную окружения PATH для того, чтобы извлечь из нее пути для поиска исполняемых файлов. Другая переменная, COMSPEC, указывает путь к командному процессору COMMAN.COM.

Переменные окружения создаются, например, во время установки программ, их настройки, во время работы в командной строке.

Для создания переменной окружения используется команда SET.

Введем команду:

```
SET A=ABC
```

В результате в блок окружения командного процессора запишется переменная окружения A со значением ABC. Фактически в блок окружения добавится строка "A=ABC". Неважно, что записанная информация не имеет никакого смысла в данном случае. Это просто пример.

Посмотреть переменную окружения можно также с помощью команды SET. Введем команду:

```
SET A
```

В чистой MS-DOS при этом будет выведена строка "A=ABC". В современной операционной системе будут выведены все переменные окружения, имя которых начинается с "A".

Удаляет переменную окружения также команда SET.

Введем команду для удаления переменной A:

```
SET A=
```

Убедиться в том, что переменная A удалена, можно командой SET A.

В пакетных файлах переменные окружения можно устанавливать, использовать и удалять. При использовании переменной окружения ее заключают в проценты, например, %PATH%. В современных операционных системах с переменными окружения можно выполнять многочисленные операции.

Создадим новый пакетный файл ENV.BAT. Название пакета берем от слова *Environment*, которым обозначают окружение.

Пакет создает некоторую переменную окружения, выводит ее значение на консоль, и удаляет ее:

```
@ECHO OFF
SET A=ABC
ECHO %A%
SET A=
```

Никакого особого смысла здесь нет. Заметим только, что конструкция %A% извлекает значение ABC, считывая его из блока окружения.

В этом легко убедиться, если разделить действия.

Изменим пакет следующим образом:

```
@ECHO OFF
ECHO "%A%"
```

Здесь переменная окружения выводится в кавычках для того, чтобы видеть пустое значение, и чтобы при пустом значении не был включен режим эхо-вывода, так как в этом случае будет выполнена команда ECHO без каких-либо параметров.

Запустим пакет и убедимся, что выводятся только кавычки.

Введем команду в командной строке:

```
SET A=1
```

Снова запустим пакет. Выводится "1".

Изменим пакет таким образом, что если переменная окружения равна единице, то выводится YES, иначе выводится NO:

```
@ECHO OFF
IF "%A%"=="1" (
    ECHO YES
) ELSE (
    ECHO NO
)
```

Запускаем пакет, убеждаемся, что выводится YES.

Можно убедиться также в том, что переменная окружения существует, если сравнить ее не с каким-то значением, а с пустой строкой.

В заключении нужно убедиться в том, что использование в пакетных файлах переменных, создаваемых самим пакетом, не влияет на исходный блок окружения.

Редактируем пакет ENV.BAT следующим образом:

```
@ECHO OFF
SET ABC=ABC
ECHO "%ABC%"
```

Запускаем пакет, убеждаемся, что ABC выводится, что означает, что переменная окружения была создана.

Вводим команду SET ABC, чтобы проверить существование переменной ABC, и убеждаемся, что не существует.

Таким образом, пакетный файл работает с *копией* блока окружения и этот блок можно использовать для хранения значений в переменных окружения. Однако при этом нужно помнить, что блок окружения имеет ограниченный объем, и записать в него много информации вряд ли удастся.

2.14. Команда FOR

Команда FOR позволяет организовать цикл выполнения какой-либо команды для набора файлов. В современных операционных системах эта команда значительно модифицирована. Здесь мы рассмотрим только основы ее применения. В простейшем случае команда имеет вид:

```
FOR %%переменная IN (набор) DO команда [параметры]
```

Здесь переменная, обозначенная двумя знаками процента, последовательно принимает значения из набора, например, спецификаций, и выполняет для текущего значения переменной заданную команду с заданными параметрами, которые являются необязательными.

Используем эту команду для создания серии каталогов командой MD.

Создаем новый пакетный файл FORMD.BAT:

```
@ECHO OFF
FOR %%A IN (1 2 3) DO MD C:\OS\%%A
```

Запускаем пакет и видим, как в каталоге появились подкаталоги.

Получим копию пакета с именем FORRD.BAT, используя сочетание Shift+F5. Установим указатель на этот новый пакет FORRD.BAT, и отредактируем его для того, чтобы удалить созданные каталоги. В отчет запишем текст пакета FORRD.BAT.

Заметим, что команду FOR можно применять и непосредственно в командной строке, однако вместо %%A нужно писать %A.

Попробуйте в командной строке ввести:

```
FOR %A IN (1 2) DO MD C:\OS\%A
```

Каталоги появятся. Затем введите команду:

```
FOR %A IN (1 2) DO RD C:\OS\%A
```

Каталоги исчезнут.

2.15. Использование перенаправления

Пакетные файлы часто исполняют длинные последовательности команд, при том сами команды на консоль не выводятся, однако вывод этих команд выводиться на консоль будет.

Поэтому в пакетных файлах можно и нужно использовать перенаправление вывода или ввода. Чаще используется перенаправление вывода для *подавления* стандартного вывода команд.

Создадим новый пакетный файл CO.CMD. Пусть пакет выполняет две команды: сначала копирует какой-нибудь файл в какой-нибудь каталог, затем удаляет этот файл. Действие абсолютно бессмысленное, но позволит продемонстрировать использование перенаправления.

Начальный текст пакета CO.CMD:

```
@ECHO OFF
COPY DA.TXT NC\
DEL NC\DA.TXT
```

Перед выполнением пакета нужно убедиться, что в текущем каталоге существуют файл DA.TXT и каталог NC.

Выполняем пакет. При выполнении на консоль выводится ничего не значащее сообщение:

```
Скопировано файлов:          1.
```

Это сообщение как раз и нужно подавить. Для этого нужно направить вывод команды COPY в абстрактное устройство.

Редактируем пакет CO.CMD:

```
@ECHO OFF
COPY DA.TXT NC\ > NUL
DEL NC\DA.TXT
```

Запускаем пакет, убеждаемся, что вывод подавлен.

Изменим пакет CO.CMD. Пусть он оперирует группой файлов:

```
@ECHO OFF
COPY *.TXT A\ > NUL
DEL A\.
```

Здесь намеренно указан несуществующий каталог назначения, чтобы показать одну особенность выводов команд. Запускаем пакет и видим:

```
Системе не удается найти указанный путь.
Системе не удается найти указанный путь.
Не удается найти указанный файл.
```

Несмотря на то, что вывод подавлен, в консоли появляются сообщения. Здесь нужно понимать, что обычный вывод команд направляется в так называемое стандартное устройство вывода, а сообщения об ошибках всегда направляются в *устройство вывода ошибок*. По умолчанию оба этих устройства совпадают с консолью.

Перенаправление вывода воздействует на стандартное устройство вывода, а устройство вывода ошибок по-прежнему остается консолью.

Для работы пакета нужно создать каталог A в текущем каталоге.

Вводим F7, вводим название каталога A, нажимаем Enter.

Запускаем пакет CO.CMD.

В результате того, что задана операция удаления, требующая подтверждения, на консоль выводится запрос:

```
C:\OS\A\*, Продолжить [Y(да)/N(нет)]?
```

В ответ на этот запрос требуется ввести Y и Enter.

Ответные действия оператора можно ввести из файла с помощью перенаправления ввода. Для этого нужно создать новый текстовый файл, назовем его ANS (без расширения), который должен содержать ровно те символы, которые требовалось ввести для подтверждения, то есть Y и Enter.

После создания этого файла убедитесь, что он его размер составляет в точности 3 байта!

Далее нужно отредактировать пакет CO.CMD:

```
@ECHO OFF
COPY *.TXT A\ > NUL
DEL A\. < ANS
```

Запускаем пакет, видим, что пакет выполняется без запроса, однако на консоль выводится и запрос, и ответ на него:

```
C:\OS\A\*, Продолжить [Y(да)/N(нет)]? Y
```

Остается только подавить этот вывод.

Снова редактируем пакет CO.CMD:

```
@ECHO OFF
COPY *.TXT A\ > NUL
DEL A\. < ANS > NUL
```

Выполнение этого пакета ничего на консоль не выводит.

В заключение покажем использование еще одной команды.

Редактируем пакет CO.CMD:

```
@ECHO OFF
COPY *.TXT A\ > NUL
ECHO Going to delete files...
PAUSE
DEL A\. < ANS > NUL
```

Команда PAUSE приостанавливает выполнение файла. Это дает возможность прервать выполнение с помощью Ctrl+C или Ctrl+Break.

Заметим, что данный пример является чисто демонстрационным. На самом деле для подавления запроса подтверждения операции удаления нужно использовать соответствующий ключ команды DEL.

Удалите каталог C:\OS\A. В FAR для этого введите F8.

2.16. Самостоятельно

Разработайте пакетный файл MS.CMD, который выполняет следующие действия:

1. Создает неопределенное количество каталогов на вашем съемном устройстве. Названия каталогов задаются в командной строке.

2. В каждый из этих каталогов копируется указанный в командной строке файл.

Командная строка должна иметь следующий формат:

```
MS  ФАЙЛ  КАТАЛОГ1  КАТАЛОГ2 . . .
```

Пример командной строки для проверки пакета:

```
MS  C:\OS\DA.TXT  C:\OS\1\  C:\OS\2\  C:\OS\3\
```

Поскольку количество каталогов заранее неизвестно, вам потребуется использовать команду SHIFT для сдвига параметров.

Для запоминания спецификации копируемого файла ФАЙЛ следует использовать переменную окружения. Для проверки окончания списка каталогов используется команда IF, которая сравнивает строку, содержащую формальный параметр, с пустой строкой.

Проверка существования файла обязательна. Проверка успешности создания каталога или копирования файла при помощи внутренней переменной ERRORLEVEL также обязательна.

Вывод команд должен быть подавлен.

В случае возникновения ошибки пакет должен сообщать об этом.

Запишите в отчет текст пакета и нарисуйте блок-схему алгоритма.

2.17. Контрольные вопросы

1. Назначение пакетных файлов.
2. Расширения файла пакетного файла.
3. Команды, используемые в пакетных файлах.
4. Команда "@".
5. Команда "ECHO".
6. Фактические и формальные параметры пакетного файла.
7. Команда "SHIFT".
8. Команда "IF".
9. Команда "FOR".
10. Команда "SET".
11. Команда "PAUSE".
12. Команда "REM".
13. Переменная "ERRORLEVEL".
14. Подавление вывода, перенаправление ввода.

3. OS-103. Файловая система FAT

Цели:
- изучение структур файловой системы FAT.

Задачи:
- физическая и логическая структура жесткого диска;
- логическая структура логического диска;
- каталоги и каталожные записи.

Работа состоит из трех частей, защищаемых отдельно.

3.1. Требования

Для выполнения работы требуется:
- чистое съемное устройство (флешка);
- права администратора при работе в среде *Windows*;
- опорный документ ReVoL-FAT-2003.pdf, далее документ FAT;
- программа просмотра секторов диска, например, ReVoL DiskView.

Теоретические сведения этой работы изложены в опорном документе.

3.2. Структура жесткого диска

Жесткий диск (*Hard Disk*) мы рассматриваем с двух сторон:
- как физическое устройство,
- как логическая структура.

3.2.1. Физическая структура жесткого диска

Как физическое устройство жесткий диск состоит из поверхностей дисковых пластин, дорожек (цилиндров), секторов. К физической структуре будем относить также адресацию секторов диска.

Эта часть структуры изучается по опорному документу FAT.

При защите этой части работы вы должны знать определения таких понятий, как поверхность, головка, дорожка, цилиндр, сектор, физический адрес, CHS, логический адрес, LBA, перевод физического адреса в логический адрес.

Есть два способа адресации.

Физический адрес в виде номера цилиндра, номера поверхности и номера сектора обозначается CHS, от английских слов *Cylinder*, *Head*, *Sector*. Это устаревшая форма адресации. Она ограничивает общий объем информации максимальными значениями составляющих адрес частей. Для записи номера цилиндра отводится 10 бит, для записи номера головки — 8 бит, а для записи номера сектора — 6 бит.

Нумерация секторов или других физических или логических компонентов физического или логического диска всегда упирается в количество бит, отводимое для записи номера (адреса). В файловых системах, особенно устаревших, это ведет к ограничению значений номеров (адресов).

Если для записи номера (адреса, индекса) отводится N бит, максимальное количество значений номера (адреса, индекса) составит 2^N.

Таким образом, максимальное количество цилиндров составляет значение 1024, максимальное количество головок (дисковых поверхностей) равно 256, максимальное количество секторов равно 63.

Перемножая максимальные значения, получаем:

1024×256×63 = 16515072 (секторов).

Переводя секторы в байты, получаем:

16515072/512 = 8455716864 (байт).

Переводя байты в мегабайты, получим:

8455716864/1024/1024 = 8064 (мегабайт).

Поэтому в настоящее время используется адресация в формате LBA (*Logical Block Addressing*), в которой секторы физического диска нумеруются последовательными числами от нуля по определенному алгоритму.

Адресация в формате LBA на самом деле является логической (что следует из названия). Однако мы рассматриваем ее в разделе физической структуры диска, так как она непосредственно связана с физическим устройством.

Адресовать секторы жесткого диска можно только после операции его разметки. Разметка выполняется во время так называемого *низкоуровневого форматирования*. При этом на диске отмечаются начала секторов с тем, чтобы аппаратно-программная часть устройства «жесткий диск» могла находить секторы, и синхронизировать фактическую скорость вращения со скоростью считывания или записи потока информационных бит.

Когда операционная система выдает команду найти сектор диска, аппаратно-программная часть устройства устанавливает считывающую головку в необходимое положение и считывает разметку, в которой записан адрес сектора. Если требуемый и фактический считанный адреса совпадают, выполняется операция чтения или записи.

Вторая сторона структуры жесткого диска — это распределение пространства диска для *организации* хранения информации.

Если бы для хранения информации использовались только сектора, управлять информационными потоками на диске было бы затруднительно.

Возникает множество вопросов.

Как узнать, какие секторы свободны от информации, а какие заняты?

Куда лучше записывать, в какие секторы?

Как узнать, что на диске есть операционная система?

Как загрузить операционную систему, как запустить ее?

Как на одном физическом диске разместить несколько независимых дисков для упорядочивания информации?

На все эти вопросы дает ответ организация на диске так называемой *файловой системы*. В этой системе используются такие понятия, как *логический диск*, *файл* и *каталог*, и устанавливается соответствие между этими

понятиями и секторами диска, адресуемыми абсолютным или относительным логическим адресом.

3.2.2. Логическая структура жесткого диска

В этой части работы изучается логическая структура физического жесткого диска. Эта структура дает возможность:

- поделить пространство диска на логические части (диски);
- определить активное логическое устройство (диск);
- загрузить загрузчик операционной системы активного устройства.

Самым важным сектором физического диска является сектор с порядковым номером 0. Этот сектор не нужно искать (все составляющие адреса равны нулю) и предполагается, что этот сектор всегда существует. Во время старта компьютера этот сектор автоматически считывается в память загрузчиком постоянной памяти (ROM) и управление передается на нулевой относительный адрес сектора.

По нулевому относительному адресу находится начало программы, которая считывает из этого же сектора информацию о логических частях диска, определяет активную часть, загружает ее нулевой сектор и передает управление на нулевой относительный адрес считанного сектора.

Таким образом, в нулевом физическом секторе находится загрузчик и сведения о логических частях. Нулевой сектор имеет специальное название MBR (*Master Boot Record*, главная загрузочная запись).

Части, на которые делится физический диск, называют *разделами*, в оригинале *partition*. Сведения о разделах составляют так называемую таблицу разделов (*Partition Table*).

Таблица разделов MBR содержит 4 дескриптора (описателя).

Дескриптор раздела содержит сведения о первом физическом секторе раздела и его размере в секторах. В одном из дескрипторов устанавливается признак активности раздела, который означает, что этот раздел содержит операционную систему, которую следует загрузить.

Более подробные сведения приведены в опорном документе FAT.

После тщательного изучения логической структуры физического диска нужно проверить и уточнить знания.

Для этой цели напишем простую программу для исследования диска рабочего компьютера.

3.2.3. Чтение секторов

Подключите свое съемное устройство к компьютеру.

Откройте *Панель управления*, выберите

- *Администрирование*,
- *Управление компьютером*,
- *Управление дисками*.

Замените букву вашего съемного устройства буквой U.

Найдите и запишите в отчет:

- номер первого жесткого диска компьютера, например 0,
- букву системного логического диска компьютера, например C,
- номер вашего съемного диска, например 1,

На рисунке 5 показан примерный вид консоли управления дисками.

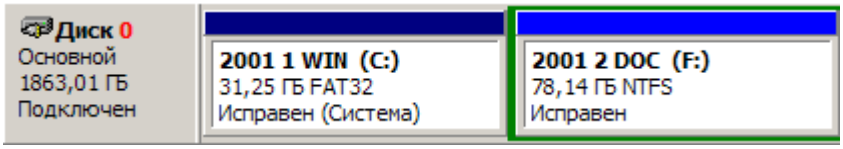


Рисунок 5 — Консоль управления дисками

Номер физического диска 0 выделен на рисунке красным цветом. На рисунке это жесткий диск, а не съемное устройство.

Введите команду форматирования вашего съемного устройства:

```
FORMAT U: /FS:FAT32 /V:MYDISK
```

Выполните форматирование.

При помощи *Microsoft Visual Studio* создайте консольное приложение *Win32*, язык программирования C++. Название проекта sread, размещение проекта — диск C:, корневой каталог.

Далее нужно установить параметры проекта.

Открываем свойства проекта и устанавливаем:

- режим Use Multi-Byte Character Set в разделе Configuration Properties, General (рисунок 6);

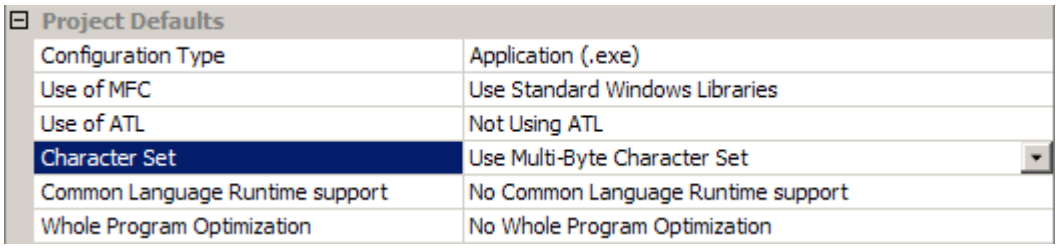


Рисунок 6 — Установка Character Set

Открываем основной модуль sread.cpp и редактируем его, так, чтобы он выглядел следующим образом:

```
// sread.cpp

#include "stdafx.h"
// основная функция
void main() {
}
```

Убедитесь, что программа компилируется и запускается.

Откройте модуль stdafx.h и добавьте в него три строки:

```
// stdafx.h
#pragma once
#define _CRT_SECURE_NO_WARNINGS 1
#include "targetver.h"
#include <stdio.h>
#include <tchar.h>
#include <windows.h>
#include <winioctl.h>
```

Убедитесь, что проект компилируется без ошибок.

Объявим три константы в модуле sread.cpp:

```
#include "stdafx.h"
// устройство
#define DISK "\\\\.\\PhysicalDrive0"
// номер сектора
#define SECTOR 0
// каталог сохранения дампа
#define PATH "c:\\dumps\\s.bin"
```

В константе DISK цифра 0 должна соответствовать номеру системного диска, который вы определили ранее.

Создайте каталог для дампов секторов C:\dumps\.

Для создания можно использовать программу FAR, клавишу F7.

Диск открывает системная функция CreateFile(). Эта функция возвращает дескриптор файла. Параметрами функции являются:

- путь к файлу, константа DISK,
- требуемый доступ, константа GENERIC_READ,
- режим разделения доступа, складывается из двух констант:
FILE_SHARE_READ | FILE_SHARE_WRITE,
- атрибуты безопасности, равно NULL,
- режим создания, константа OPEN_EXISTING,
- флаги и атрибуты, равны нулю,
- шаблон файла, равно NULL.

Описываем вызов функции в main().

После вызова нужно убедиться, что диск подключен:

```
// основная функция
void main() {
    // открываем диск
    HANDLE hDisk = CreateFile(DISK, GENERIC_READ,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL, OPEN_EXISTING, 0, NULL);
    // проверяем результат
    if (INVALID_HANDLE_VALUE == hDisk) {
        printf("Disk Open Error %d.\n\n", GetLastError());
        return;
    }
}
```

Запускаем проект и убеждаемся, что диск открывается.

Далее нам нужно считать сектор размером 512 байт в буфер.

Объявляем константу и буфер в начале модуля:

```
#define SECTOR 0
// размер сектора
#define SECTOR_SIZE 512
// буфер сектора
__declspec(align(64)) unsigned char buffer[SECTOR_SIZE] = {0};
```

В функции main() описываем чтение сектора.

Код пишем в продолжение существующего кода.

Сначала нам нужны две переменные.

Первая переменная — структура OVERLAPPED, с ее помощью задается адрес сектора в байтах. Чтобы его получить, нужно умножить номер сектора на 512, или, что одно и то же, сдвинуть его влево 9 раз:

```
void main() {
    . . . (написанный ранее код)
    OVERLAPPED ov = {0};
    ov.Offset = (SECTOR << 9) & 0xFFFFFFFF;
```

Вторая переменная — количество прочитанных байтов:

```
    OVERLAPPED ov = {0};
    ov.Offset = (SECTOR << 9) & 0xFFFFFFFF;
    DWORD dwRead = 0;
```

Сектор считывается в буфер buffer.

Чтение выполняет функция ReadFile():

```
    // читаем сектор
    if (!ReadFile(hDisk, buffer, SECTOR_SIZE, &dwRead, &ov)) {
        printf("Disk Read Error %d.\n\n", GetLastError());
        // закрываем диск
        CloseHandle(hDisk);
        return;
    }
}
```

Запускаем проект и убеждаемся, что чтение происходит. Если нет, то нужно смотреть, какая ошибка произошла. Например, ошибка 87 говорит о том, что адрес вычислен неверно, он должен быть кратен числу 512.

Если сектор прочитан, нужно записать его в файл, для чего сначала файл нужно открыть. Открываем его с помощью CreateFile():

```
    // открываем файл для записи сектора
    HANDLE hFile = CreateFile(PATH, GENERIC_WRITE, 0,
        NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (INVALID_HANDLE_VALUE == hFile) {
        printf("File Open Error %d.\n\n", GetLastError());
        // закрываем диск
        CloseHandle(hDisk);
        return;
    }
```

Запускаем проект и убеждаемся, что файл открывается.

Если нет, ищем причину, анализируя номер ошибки.

Остается только записать буфер в файл при помощи WriteFile():

```
DWORD dwWritten = 0;
// записываем бинарный сектор
if (!WriteFile(hFile, buffer, dwRead, &dwWritten, NULL)) {
    printf("File Write Error %d.\n\n", GetLastError());
} else {
    printf("Success.\n\n");
}
// закрываем открытые объекты
CloseHandle(hFile);
CloseHandle(hDisk);
```

Если все правильно, то после выполнения этой программы в каталоге C:\dumps\ должен появиться файл s.bin.

И это цель проекта.

Теперь при помощи этого проекта мы можем смотреть содержимое секторов как физических, так и логических дисков.

3.2.4. Таблица разделов MBR

Открываем файловый менеджер FAR. Открыта только правая панель. Панели включаются или выключаются Ctrl+F1, Ctrl+F2.

Переходим в каталог C:\dumps\.

Находим двоичный файл s.bin и переименовываем его в MBR0.bin, здесь 0 — номер физического диска, указанный в программе. Для переименования вводим сочетание Shift+F6 (F6, если открыта одна панель).

Открываем двоичный файл MBR0.bin для чтения в режиме дампа. Для открытия вводим F3, режим выбирает F4.

Убеждаемся, что сектор является загрузочным.

У загрузочного сектора последние два байта равны 55AAh.

Открываем опорный документ FAT.

Находим описание MBR и таблицы разделов.

Находим в конце дампа начало таблицы разделов по адресу 1BEh.

Записываем в отчет «Диск 0». Здесь 0 — номер физического диска, указанный в программе.

Определяем количество дескрипторов в таблице разделов. Один дескриптор занимает ровно 16 байт. Одна строка дампа это ровно 16 байт.

Записываем в отчет «Разделов 2». Здесь 2 — то количество, которое определили вы. Записываем в отчет «Раздел 1».

Определяем, является ли раздел активным, для чего находим первый байт дескриптора. Если он равен 80h, раздел активный. Записываем в отчет «Активный» или «Не активный».

Находим байт дескриптора, соответствующий полю «Тип раздела».

Записываем в отчет «Тип 0Fh». Значение здесь приведено для примера, у вас получится какое-то, возможно другое. Смотрим в опорном документе, какой это раздел. Если в опорном документе не описывается тип раздела, который вы определили, найдите его в сети интернет.

Находим байт дескриптора, соответствующий началу поля «Первый сектор раздела». Поле занимает 4 байта. Записываем в отчет шестнадцатеричные значения четырех байтов, начиная с четвертого байта, то есть задом наперед. Записываем в отчет «Начало 00 00 00 3Fh».

Здесь значения приведены для примера, у вас могут быть другие.

Переводим полученное число в десятичное, значение записываем за шестнадцатеричным значением.

Находим байт дескриптора, соответствующий началу поля «Размер раздела в секторах». Поле занимает 4 байта. Точно также определяем шестнадцатеричное и десятичное значение, записываем в отчет «Размер …».

Переходим ко второму разделу и выполняем все действия для него.

Если есть еще разделы, описываем их тоже.

Возвращаемся в проект sread. Находим строку:

```
// устройство
#define DISK "\\\\.\\PhysicalDrive0"
```

Записываем вместо выделенного нуля номер вашего съемного устройства N, запускаем наш проект, получаем файл s.bin. Переименовываем файл s.bin в MBRN.bin.

Записываем в отчет «Диск N». Далее определяем характеристики из таблицы разделов, как описано выше, записываем в отчет.

Проверяем правильность вычислений, запустив программу DiskView.

Программа DiskView есть у преподавателя на съемном устройстве.

Если все значения совпадают, продолжаем дальше, иначе вычисляем значения заново.

3.2.5. Загрузчик раздела BR или SMBR

Возвращаемся в проект sread.

Находим строку:

```
// номер сектора
#define SECTOR 0
```

Заменяем выделенный здесь ноль на номер первого сектора первого раздела жесткого диска XXX. Запускаем проект, получаем файл s.bin.

Первый сектор диска всегда загрузочный, о чем свидетельствует метка 55AA в конце сектора. Если это не так, прочитан не тот сектор.

Это либо загрузчик операционной системы BR, либо таблица разделов SMBR. Определить, что это, можно по разным признакам.

Например, если это загрузчик операционной системы BR, то первые три байта сектора — команда JMP (прыжок, переход), с помощью которой обходится (перепрыгивается) BPB и другие сведения о диске.

Код первых трех байтов команды JMP может быть следующим.

Если первый (нулевой) байт команды JMP равен E9h, используется длинный переход, следующие два байта — адрес перехода.

Если первый байт команды JMP равен EBh, используется короткий переход, следующий байт — относительный адрес перехода, третий байт команды при этом равен 90h (команда NOP, нет операции).

Если первый байт сектора не равен ни EBh ни E9h, то это не BR.

Другим признаком загрузчика BR является наличие в дампе строк сообщений загрузчика. Эти строки легко читаются, могут быть как на английском, так и на локальном (русском) языке.

Определив назначение сектора, записываем в отчет «Раздел 1 - BR», или «Раздел 1 - SMBR». Файл s.bin переименовываем в 01BR.bin, если это BR или в 01SMBR.bin, если это SMBR.

Здесь 0 — номер жесткого диска, 1 — номер раздела.

Эти же операции проделываем для других разделов жесткого диска, а затем для разделов вашего съемного диска. На вашем съемном диске может оказаться ноль, один и больше разделов.

На этом первая часть работы завершается, требуется защита.

3.3. Структура логического диска

Изучите опорный документ FAT.

Начинайте изучение со структуры логического диска, с понятий системной области диска и области файлов.

В системной области диска находятся:

- нулевой сектор, загрузчик операционной системы BR,
- резервные секторы,
- две таблицы размещения файлов FAT,
- корневой каталог фиксированного размера, если файловая система FAT12 или FAT16. FAT12 используется для дискет. В FAT32 корневой каталог находится в области файлов, а не в системной области.

Нужно знать, что такое BPB, какие сведения он содержит, в частности:

- размер сектора,
- размер кластера,
- количество загрузочных секторов,
- размер корневого каталога,
- размер одной таблицы FAT,
- количество секторов на диске.

Эти сведения позволяют вычислить размер системной области C2, равный номеру первого сектора второго кластера (начало области файлов) по формуле:

C2 = секторов загрузчика + размер FAT×2 + размер корневого каталога

На самом деле размер FAT нужно умножать на количество FAT, считанное из BPB, однако на практике оно всегда равно двум.

Далее изучите понятие «*кластер*», и устройство таблицы FAT, ее связь с кластерами. Нужно понять, зачем нужны кластеры.

Научитесь рассчитывать размер кластера, исходя из размера диска.

Если размер ячейки FAT в битах равен M, то максимальное количество кластеров N равно $2^M - 11$. Заметим, что для FAT32 размер M равен 28.

Если размер диска равен S, то начальный, ориентировочный размер кластера C равен S/N. Полученное значение C округляется до ближайшего большего числа, кратного степени двойки, с учетом допустимых размеров кластеров.

Для закрепления этих знаний снова используем проект sread.

3.3.1. Блок параметров BIOS

Найдем в отчете букву системного логического диска жесткого диска, это, скорее всего, окажется буква C.

Переходим в проект sread.

Находим строку

```
// устройство
#define DISK "\\\\.\\PhysicalDrive0"
```

Для чтения логического диска строка должна иметь другой вид.

Если имя логического диска C:, запишем следующее имя устройства:

```
// устройство
//#define DISK "\\\\.\\PhysicalDrive0"
#define DISK "\\\\.\\C:"
// номер сектора
#define SECTOR 0
```

Заметим, что номер сектора должен быть равен *нулю*.

Запускаем проект, получаем файл s.bin.

Переименовываем этот файл в CBR.bin. Здесь буква C должна совпадать с той, которая указана в константе DISK. Велика вероятность того, что этот файл в точности совпадет с файлом 01BR.bin (тот же сектор).

Файл CBR.bin открываем для чтения в режиме дампа F3.

Убеждаемся, что это загрузочный сектор по метке 55AAh.

Файловая система диска может быть FAT32, NTFS или другая. Нас интересует только FAT. Поэтому сейчас нужно определить, какая файловая система на данном логическом диске.

Конечно, можно посмотреть в проводнике. Но мы должны научиться определять файловую систему, изучая на дамп сектора. Это легко.

BPB в секторе есть в любом случае. Он располагается в начальной части сектора, начиная с байта 0Bh (11), и занимает всего несколько десятков байт (для разных систем разное количество, но не более 80).

Метка файловой системы должна быть хорошо видна в этой части.

Если это NTFS, то метка NTFS видна в первой строке дампа.

Если это FATn, то метка FATn видна не далее шестой строки.

Остальная часть сектора — загрузчик операционной системы.

Запишите в отчет «Диск C: - NTFS» или «Диск C: - FAT32».

Вероятность того, что это не FAT, близка к 100%.

Возвращаемся в проект sread.

Находим строку

```
#define DISK "\\\\.\\C:"
```

Заменяем букву диска буквой вашего съемного устройства:

```
#define DISK "\\\\.\\U:"
```

Запускаем проект, получаем файл s.bin.

Переименовываем файл s.bin в UBR.bin.

Файл UBR.bin открываем для чтения в режиме дампа F3.

Убеждаемся, что это загрузочный сектор по метке 55AAh.

Убеждаемся, что это файловая система FAT32.

Диск только что был отформатирован с указанием этой системы.

Далее нужно определить:

- размер сектора,
- размер кластера,
- количество секторов загрузчика,
- число копий FAT,
- размер корневого каталога,
- количество секторов на диске,
- размер FAT в секторах.

Для этого нужно для каждого параметра найти номер первого байта параметра и количество байт, которое параметр занимает (все эти сведения приведены в опорном документе FAT).

В отчет для каждого параметра записываем его наименование, шестнадцатеричные значения байтов и эквивалентные десятичные значения.

Параметры «Количество секторов на диске» и «Размер FAT» встречаются в описании BPB два раза. Это связано с тем, что размер дисков увеличивался по мере их усовершенствования, и разрядности для представления параметров перестало хватать. Поэтому были введены новые поля, а старые оставлены для совместимости. В практическом плане это означает, что если вы определили нулевое значение параметра, его значение нужно искать в другом поле (далее в блоке).

Заметим также, что нулевой размер корневого каталога для FAT32 является правильным значением. Оно используется для расчета номера первого сектора второго кластера C2 в любом случае.

После определения всех параметров нужно рассчитать:

- номер первого сектора первой FAT,
- номер первого сектора второй FAT,
- номер первого сектора второго кластера (начало области файлов).

Вычисленные значения записываем в отчет, указывая их названия:

«Первый сектор FAT-1 - F1»

«Первый сектор FAT-2 - F2»

«Первый сектор кластера 2 - C2»

Здесь F1, F2, C2 обозначают вычисленные значения.

Все значения, как параметры BPB, так и вычисленные номеров секторов, проверяем с помощью программы DiskView. Программа находится на съемном устройстве преподавателя.

Если значения правильные, то продолжаем, иначе уточняем значения.

3.3.2. Таблица FAT, корневой каталог

Возвращаемся в проект sread.

Заменяем номер сектора значением F1.

Запускаем проект, файл s.bin переименовываем в FAT1.bin.

Заменяем номер сектора значением F2.

Запускаем проект, файл s.bin переименовываем в FAT2.bin.

Убеждаемся, что оба файла равны, FAT-1 = FAT-2.

Заменяем номер сектора значением C2.

Запускаем проект, файл s.bin переименовываем в ROOT.bin.

Открываем файл ROOT.bin для чтения в режиме дампа.

Убеждаемся, что сектор начинается с MYDISK.

Если нет, скорее всего, вам нужно форматировать диск заново. Обратитесь в этом случае к преподавателю за инструкциями.

MYDISK — метка тома, указанная в команде форматирования:

```
FORMAT L: /FS:FAT32 /V:MYDISK
```

Открываем файл FAT1.bin для чтения в режиме дампа.

Первая ячейка FAT содержит байт-описатель, который равен для вашего диска, скорее всего, значению F8, дополненным слева двоичными единицами до 28 разрядов.

Вторая ячейка должна содержать значение, используемое для обозначения последнего кластера.

Третья ячейка, имеющая номер 2, должна содержать признак последнего кластера файла корневого каталога. Корневой каталог, таким образом, имеет после форматирования размер 1 кластер.

Корневой каталог не пустой, как мы только что убедились.

В нем есть одна *запись*, это метка тома MYDISK.

Размер ячейки FAT32 равен в точности 32 бита, то есть 4 байта, хотя для адресации используется всего 28 бит. Поэтому у всех значений ячеек FAT, кроме второй, первая шестнадцатеричная цифра равна нулю.

Запишите в отчет шестнадцатеричные значения:

- байта-описателя,
- второй ячейки FAT,
- третьей ячейки FAT.

Далее мы будем выполнять следующие действия.

1. Создадим каталог U:\firstcat\.
2. Создадим файл U:\firstcat\файл-содержит-текст-123.txt.
3. Создадим каталог U:\firstcat\second\.
4. Удалим файл U:\firstcat\файл-содержит-текст-123.txt.
5. Создадим файл U:\firstcat\1.txt.
6. Установим атрибуты Hidden и Read Only каталога U:\firstcat\second\.

После каждого действия запишем секторы, которые изменились, при помощи проекта sread. Получится множество дампов. Каждому из них дадим осмысленное название. Эти дампы используем для изучения структур FAT, каталожных записей, и для защиты.

1. Создаем каталог U:\firstcat\.

В FAR перейдем на диск U:, — Alt+F2, U.
Для создания каталога введите F7.
Введите название каталога firstcat, нажмите Enter.
После создания каталога на диске произошли следующие изменения:
- каталог записан в кластер 3, первый свободный;
- в FAT-1 и FAT-2 в ячейку 3 (четвертую) записан признак конца файла;
- в корневой каталог добавлены записи.

В FAR перейдем на диск C:, — Alt+F2, C.
Проект sread.
Вычисляем номер первого сектора третьего кластера C3:
C3 = C2 + размер кластера.
Вычисление записываем в отчет.
Укажем номер сектора C3.
Запустим проект. Переименуем файл s.bin в firstcat.bin.

Открываем файл firstcat.bin для чтения, убеждаемся, что это файл каталога следующим образом:
- в начале первой строки дампа стоит знак «точка»,
- в начале третьей строки дампа стоит два знака «точка».

Если нет, то считан не тот сектор, надо разбираться.
Укажем номер сектора F1.
Запустим проект, Переименуем файл s.bin в FAT-firstcat.bin.
Укажем номер сектора C2.
Запустим проект. Переименуем файл s.bin в ROOT-firstcat.bin.

2. Создаем файл U:\firstcat\файл-содержит-текст-123.txt.

В FAR перейдем на диск U:, — Alt+F2, U.
Для создания файла нужно зайти в каталог firstcat.
Убедитесь, что вы находитесь в каталоге U:\firstcat.
Создаем текстовый файл при помощи Shift+F4.
Название файла: Файл-содержит-текст-123.txt

Длина собственно имени 23, длина полного имени 27.
Когда откроется редактор, вводим текст 123.
Для сохранения файла нажимаем Escape и Enter.

После создания файла на диске произошли следующие изменения:
- в файле каталога firstcat появились новые записи;
- в первом свободном кластере 4 записан текст файла;
- в FAT-1 и FAT-2 в ячейке 4 (пятой) записан признак конца файла.

В FAR перейдем на диск C:, — Alt+F2, C.
Проект sread.
Укажем сектор C3, сектор каталога firstcat.
Запускаем проект. Переименуем файл s.bin в firstcat-file.bin.
Вычисляем номер первого сектора четвертого кластера C4:
C4 = C2 + размер кластера×2.
Вычисление записываем в отчет.
Укажем сектор C4.
Запускаем проект. Переименуем файл s.bin в C4-file.bin.
Укажем сектор F1.
Запускаем проект. Переименуем файл s.bin в F1-file.bin.

3. Создаем каталог U:\firstcat\second\.

В FAR перейдем на диск U:, — Alt+F2, U.
Убедимся, что находимся в каталоге firstcat.
Для создания каталога нажимаем F7, вводим second, Enter.

После создания каталога на диске произошли следующие изменения:
- в первом свободном кластере 5 записан файл каталога;
- в файле каталога firstcat появились новые записи;
- в FAT-1 и FAT-2 в ячейке 5 (шестой) записан признак конца файла.

В FAR перейдем на диск C:, — Alt+F2, C.
Проект sread.
Вычисляем номер первого сектора пятого кластера C5:
C5 = C2 + размер кластера×3.
Вычисление записываем в отчет.
Укажем сектор C5.
Запускаем проект. Переименуем файл s.bin в second.bin.
Укажем сектор C3.
Запускаем проект. Переименуем файл s.bin в firstcat-second.bin.
Укажем сектор F1.
Запускаем проект. Переименуем файл s.bin в FAT-second.bin.

4. Удаляем файл U:\firstcat\файл-содержит-текст-123.txt.

В FAR перейдем на диск U:, — Alt+F2, U.
Убедимся, что каталог U:\firstcat\.
Установим указатель на файл файл-содержит-текст-123.txt.
Для удаления нажмем F8 и Enter.

После удаления на диске произошли следующие изменения:
- в файле каталога firstcat изменились записи;
- в FAT-1 и FAT-2 ячейка 4 (пятая) помечена как пустая.

В FAR перейдем на диск C:, — Alt+F2, C.
Проект sread.
Укажем сектор C3.
Запускаем проект. Переименуем файл s.bin в firstcat-delete.bin.
Укажем сектор F1.
Запускаем проект. Переименуем файл s.bin в FAT-delete.bin.
Укажем сектор C4.
Запускаем проект. Переименуем файл s.bin в C4-delete.bin.
Сравниваем файлы C4-file.bin и C4-delete.bin, убеждаемся, что они одинаковы, с самим файлом ничего не произошло.

5. Создаем файл U:\firstcat\1.txt.

В FAR перейдем на диск U:, — Alt+F2, U.
Убедимся, что каталог U:\firstcat\.
Создаем текстовый файл 1.txt. Вводим Shift+F4, вводим 1.txt, нажимаем Enter, появится редактор, вводим текст 1.txt. Сохраняем файл, нажимая Escape и Enter.

После создания файла на диске произошли следующие изменения:
- в файле каталога firstcat появились записи;
- в FAT-1 и FAT-2 в ячейке 4 (пятой) появился признак конца файла;
- в первом свободном кластере 4 записан текст файла.

В FAR перейдем на диск C:, — Alt+F2, C.
Проект sread.
Укажем сектор C3.
Запускаем проект. Переименуем файл s.bin в firstcat-file-1.bin.
Укажем сектор F1.
Запускаем проект. Переименуем файл s.bin в FAT-file-1.bin.
Укажем сектор C4.
Запускаем проект. Переименуем файл s.bin в C4-file-1.bin.

6. Устанавливаем атрибуты Hidden и Read Only каталога second.

В FAR перейдем на диск U:, — Alt+F2, U.
Перейдем в каталог U:\firstcat\.
Установим указатель на запись second, введем Ctrl+A.
Установим флажки:
«Только для чтения» (Read Only) и
«Скрытый» (Hidden),
нажмем Enter.
При этом изменится каталожная запись в каталоге firstcat.

В FAR перейдем на диск C:, — Alt+F2, C.
Проект sread.
Укажем сектор C3.
Запускаем проект. Переименуем файл s.bin в second-attr.bin.

Таким образом, у нас есть дампы необходимых секторов в разные моменты времени, и их можно использовать для исследования.

Проект sread нам больше не понадобится, его можно закрыть, скопировать на свой носитель информации. Дампы нужны для защиты.

На этом эта часть работы завершена.

При защите нужно уметь объяснять действия файловой системы при создании файлов и каталогов, показывать это на примере дампов.

3.4. Каталожные записи

Изучите раздел «Каталожные записи» опорного документа FAT.
Для закрепления знаний используем дампы, подготовленные ранее.

Любая каталожная запись имеет размер 32 байта, то есть ровно две строчки дампа. Один сектор содержит максимум 16 записей.

Если первый байт записи равен 0h, записей больше нет, это *конец каталога*. Если первый байт записи равен E5h или 05h, запись *удалена*.

Можно различать четыре вида каталожных записей:
- запись метки тома;
- запись MS-DOS (оригинальная);
- запись MS-DOS Win32 (короткое имя, будем обозначать ее DOS-32);
- запись Windows (длинное имя, будем обозначать ее WIN-32).

Начнем с изучения каталожных записей корневого каталога.
Файл ROOT.bin. Единственная запись описывает метку тома:

```
4D 59 44 49 53 4B 20 20  20 20 20 08 00 00 00 00  MYDISK      .....
00 00 00 00 00 00 AF 6B  3D 31 00 00 00 00 00 00  ......¯k =1......
```

Первые 11 байт этой записи отводятся под буквы метки тома. Длина метки MYDISK равна 6. Остальные 5 байт заполнены пробелами (код 20h).

Чтобы отличить эту запись от записей файлов и каталогов, в байте атрибутов бит V (*Volume*) установлен.

Байт атрибутов находится в 12 байте записи, подчеркнут. Запишите в отчет значение этого байта в шестнадцатеричной и в двоичной форме: «Метка тома - атрибуты 08h, 00001000b».

Поле «первый кластер» этой записи равно нулю, в записи подчеркнуто. Это также отличает запись метки тома.

Кроме того, видно, что это оригинальная запись MS-DOS. Это следует из того, что зарезервированные поля имеют нулевое значение.

Далее рассмотрим файл ROOT-firstcat.bin. Он соответствует созданию каталога \firstcat. При этом появилась следующая запись (у вас поля, в которых записаны дата или время, будут другими, ниже они затенены):

```
46 49 52 53 54 43 41 54  20 20 20 10 08 AA F4 89  FIRSTCAT    ..Єф‰
39 49 39 49 00 00 AF 6B  3D 31 03 00 00 00 00 00  9I9I..x‰ 9I......
```

Запишите в отчет «Запись каталога firstcat».

Запишите в отчет «DOS-32».

Это запись DOS-32, так как зарезервированные поля не пусты. На то, что это каталог, указывает значение 10h байта атрибутов — в нем бит D (*Directory*) установлен. Этот байт также подчеркнут, его значение 10h.

Запишите в отчет значение байта атрибутов в шестнадцатеричной и в двоичной форме: «Атрибуты …».

Так как это запись DOS-32, номер первого кластера каталога записан в двух местах, подчеркнуто. Младшая часть в двух байтах имеет значение 03 00h, старшая 00 00h. Разворачивая байты «задом наперед» и складывая обе части, получим номер кластера каталога, — 00 00 00 03h, то есть 3.

Запишите в отчет «Первый кластер 3».

Последние четыре байта записи равны нулю. В этом месте записывается размер. Размер каталога всегда равен нулю.

Следует также обратить внимание на первые 11 байт, полное имя.

Видно, что символы переведены в верхний регистр.

Далее рассмотрим пустой каталог firstcat, файл firstcat.bin.

```
2E 20 20 20 20 20 20 20  20 20 20 10 00 BB 5C 6E  .           ..Єф‰
3D 31 3D 31 00 00 5D 6E  3D 31 03 00 00 00 00 00  9I9I..x‰ 9I......
2E 2E 20 20 20 20 20 20  20 20 20 10 00 BB 5C 6E  ..          ..Єф‰
3D 31 3D 31 00 00 5D 6E  3D 31 00 00 00 00 00 00  9I9I..x‰ 9I......
```

Пустой каталог на самом деле не пустой, если только он не корневой. В нем содержатся две специальных записи, запись текущего каталога, имеющего имя ".", и запись родительского каталога, имеющего имя "..".

Если родительский каталог — корневой, то в записи родительского каталога поле первого кластера равно нулю.

Если родительский каталог не корневой, поле первого кластера в записи действительно указывает на первый кластер.

Рассмотрим пустой каталог second, файл second.bin.

```
2E 20 20 20 20 20 20 20  20 20 20 10 00 5A 59 6F  .           ..ZYo
3D 31 3D 31 00 00 5A 6F  3D 31 05 00 00 00 00 00  =1=1..Zo =1......
2E 2E 20 20 20 20 20 20  20 20 20 10 00 5A 59 6F  ..          ..ZYo
3D 31 3D 31 00 00 5A 6F  3D 31 03 00 00 00 00 00  =1=1..Zo =1......
```

Он отличается от предыдущего тем, что в записи родительского каталога указан номер первого кластера 3.

Далее интересно рассмотреть файл firstcat-file.bin, соответствующий созданию файла с длинным именем. Скорее всего, потребуется выбрать кодировку 866, чтобы увидеть имя русскими буквами в записи DOS-32.

```
43 74 00 00 00 FF FF FF  FF FF FF 0F 00 68 FF FF  Ct...       ..h
FF FF FF FF FF FF FF FF  FF FF 00 00 FF FF FF FF              ..

02 2D 00 42 04 35 04 3A  04 41 04 0F 00 68 42 04  .-.B.5.: .A...hB.
2D 00 31 00 32 00 33 00  2E 00 00 00 74 00 78 00  -.1.2.3. ....t.x.

01 44 04 30 04 39 04 3B  04 2D 00 0F 00 68 41 04  .D.0.9.; .-...hA.
3E 04 34 04 35 04 40 04  36 04 00 00 38 04 42 04  >.4.5.@. 6...8.B.

94 80 89 8B 2D 91 7E 31  54 58 54 20 00 55 A9 6E  ФАЙЛ-С~1 TXT .Uйn
3D 31 3D 31 00 00 AA 6E  3D 31 04 00 03 00 00 00  =1=1..кn =1......
```

Поскольку полная длина имени составляет 27 символов, создаются дополнительные записи с длинным именем. Заметим, что запись длинного имени создается и тогда, когда полная длина имени менее 11 символов, но не все символы соответствуют ограничениям MS-DOS. При этом формируется короткое имя, которое соответствует требованиям MS-DOS, а для записи действительного имени формируется одна запись длинного имени.

Записи длинного имени располагаются перед записью DOS-32 в обратном порядке. Они пронумерованы, номер содержится в первом байте записи, эти номера подчеркнуты. Последняя запись (первая по порядку расположения в каталоге) имеет номер 43h, из которого нужно вычесть 40h, признак последней записи.

Записи длинного имени отличаются байтом атрибутов, равным 0Fh.

Запишите в отчет «Запись длинного имени», далее количество записей WIN-32. Для каждой записи WIN-32 укажите записанные в ней буквы полного имени.

Далее нужно посмотреть файл firstcat-delete.bin, соответствующий удалению файла из каталога firstcat. Здесь можно увидеть, что после удаления файла в каталожных записях первые байты заменены на E5h. Это дает возможность восстановить файл сразу после удаления.

Рассмотрим файл firstcat-file-1.bin, соответствующий созданию еще одного текстового файла 1.txt.

```
31 20 20 20 20 20 20 20  54 58 54 20 10 9C ED 70  1       TXT .œíp
3D 31 3D 31 00 00 EE 70  3D 31 04 00 05 00 00 00  =1=1..îp =1......
```

Здесь интересно отметить, как записывается имя, если его длина небольшая. Видно, что расширение записывается всегда в последних байтах.

Наконец, рассмотрим файл second-attr.bin.

```
53 45 43 4F 4E 44 20 20  20 20 20 13 08 5A 59 6F  SECOND       ..ZYo
3D 31 3D 31 00 00 5A 6F  3D 31 05 00 00 00 00 00  =1=1..Zo =1......
```

Запишите в отчет: «Атрибуты каталога second» и далее значение байта атрибутов в шестнадцатеричной и двоичной форме.

Кроме, того, для этой записи вычислим дату и время создания.

Запишите в отчет:

«Дата создания: 29.09.2004»

«Время создания: 13:58:50,9».

Значения даты и времени нужно вычислить.

Дата вычисляется так.

Записываем значения байтов со смещением 17 и 16:

31 3D

Переводим цифры в двоичную форму:

0011 0001 0011 1101

Делим полученную последовательность на части 7—4—5 байт:

0011000—1001—11101

Каждую часть переводим в десятичное число:

24 9 29

К первой части прибавляем 1980:

2004 9 29

Получили дату 29.09.2004.

Время вычисляется так.

Записываем значения байтов со смещением 15 и 14:

6F 59

Переводим цифры в двоичную форму:

0110 1111 0101 1001

Делим полученную последовательность на части 5—6—5 байт:

01101—111010—11001

Каждую часть переводим в десятичное число:

13 58 25

Переводя двухсекунды в секунды (умножая на два), получим

13:58:50

Переводим байт со смещением 13 в десятичное число, получаем:

5Ah = 90

Это означает, что к секундам нужно прибавить 0,9.

Окончательно получим время создания 13:58:50,9.

Чтобы проверить, установим указатель FAR на запись каталога second и введем Ctrl+A.

Третья часть и работа в целом завершена.

Требуется защита изученного материала.

Литература

1. Нортон П. Персональный компьютер фирмы IBM и операционная система MS-DOS: Пер. с англ. - М.: Радио и связь. 1992 г. - 416 с.: ил.

2. Данкан Р. Профессиональная работа в MS-DOS: Пер. с англ. — М.: Мир, 1993. — 509 с., ил.

3. Сетевые операционные системы / В.Г. Олифер, Н.А. Олифер. - СПб.: Питер, 2002. - 544 с., ил.

4. Таненбаум Э. Современные операционные системы. 2-е изд. - СПб.: Питер, 2002. - 1040 с., ил.

5. Гордеев А.В. Операционные системы: Учебник для вузов/ А.В. Гордеев. - 2-е изд. - СПб: Изд. дом «Питер», 2004. - 416 с.

6. Рихтер Дж. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows / Пер. с англ. - 4-е изд. - СПб.: Питер; Издательско-торговый дом «Русская Редакция», 2001. - 753 с.: ил.