Лекция 2

**Командная строка**

Основная среда взаимодействия с Linux - *командная строка*. Суть ее в том, что каждая строка, передаваемая пользователем системе, - это *команда*, которую та должна выполнить. Пока не нажата кнопка **Enter**, строку можно редактировать, затем она отсылается системе:[methody@localhost methody]$ cal

Сентября 2004

Вс Пн Вт Ср Чт Пт Сб

1 2 3 4

5 6 7 8 9 10 11

12 13 14 15 16 17 18

19 20 21 22 23 24 25

26 27 28 29 30

[methody@localhost methody]$ echo Hello, world!

Hello, world!

**Файлы**

Требования к хранению информации:

* возможность хранения больших объемов данных
* информация должна сохраняться после прекращения работы процесса
* несколько процессов должны иметь одновременный доступ к информации

**2.1.1 Именование файлов**

Длина имени файла зависит от ОС, может быть от 8 (MS-DOS) до 255 (Windows, LINUX) символов.

ОС могут различать прописные и строчные символы. Например, WINDOWS и windows для MS-DOS одно и тоже, но для UNIX это разные файлы.

Во многих ОС имя файла состоит из двух частей, разделенных точкой, например windows.exe. Часть после точки называют **расширением файла**. По нему система различает тип файла.

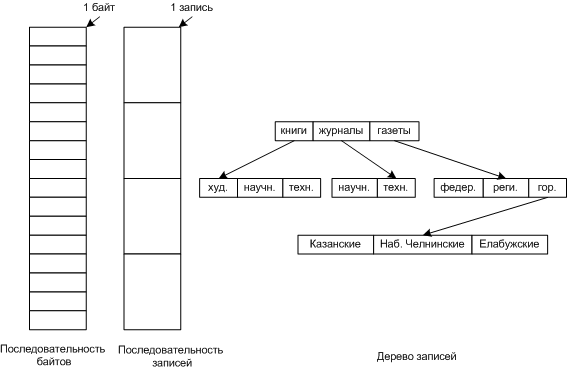
У MS-DOS расширение составляет 3 символа. По нему система различает тип файла, а также можно его исполнять или нет.

У UNIX расширение ограничено размером имени файла в 255 символов, также у UNIX может быть несколько расширений, но расширениями пользуются больше прикладные программы, а не ОС. По расширению UNIX не может определить исполняемый это файл или нет.

**2.1.2 Структура файла**

Три основные структуры файлов:

1. **Последовательность байтов** - ОС не интересуется содержимым файла, она видит только байты. Основное преимущество такой системы, это гибкость использования. Используются в Windows и UNIX.
2. **Последовательность записей** - записей фиксированной длины (например, перфокарта), считываются последовательно. Сейчас не используются.
3. **Дерево записей** - каждая запись имеет ключ, записи считываются по ключу. Основное преимущество такой системы, это скорость поиска. Пока еще используется на мэйнфреймах.



Три типа структур файла.

**2.1.3 Типы файлов**

Основные типы файлов:

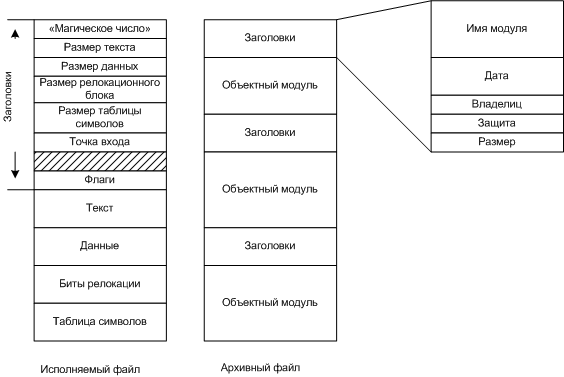
* **Регулярные** - содержат информацию пользователя. Используются в Windows и UNIX.
* **Каталоги** - системные файлы, обеспечивающие поддержку структуры файловой системы. Используются в Windows и UNIX.
* **Символьные** - для моделирования ввода-вывода. Используются только в UNIX.
* **Блочные** - для моделирования дисков. Используются только в UNIX.

Основные типы регулярных файлов:

* **ASCII файлы**- состоят из текстовых строк. Каждая строка завершается возвратом каретки (Windows), символом перевода строки (UNIX) и используются оба варианта (MS-DOS). Поэтому если открыть текстовый файл, написанный в UNIX, в Windows, то все строки сольются в одну большую строку, но под MS-DOS они не сольются (*это достаточно частая ситуация*). Основные преимущества ASCII файлов:  
  - могут отображаться на экране, и выводится на принтер без преобразований  
  - могут редактироваться почти любым редактором
* **Двоичные файлы**- остальные файлы (не ASCII). Как правило, имеют внутреннею структуру.

Основные типы двоичных файлов:

* **Исполняемые** - программы, их может обрабатывать сама операционная система, хотя они записаны в виде последовательности байт.
* **Неисполняемые** - все остальные.



Примеры исполняемого и не исполняемого файла

**«Магическое число»** - идентифицирующее файл как исполняющий.

**2.1.4 Доступ к файлам**

Основные виды доступа к файлам:

* **Последовательный** - байты читаются по порядку. Использовались, когда были магнитные ленты.
* **Произвольный** - файл можно читать с произвольной точки. Основное преимущество возникает, когда используются большие файлы (например, баз данных) и надо считать только часть данных из файла. Все современные ОС используют этот доступ.

**2.1.5 Атрибуты файла**

Основные атрибуты файла:

* Защита - кто, и каким образом может получить доступ к файлу (пользователи, группы, чтение/запись). Используются в Windows и UNIX.
* Пароль - пароль к файлу
* Создатель - кто создал файл
* Владелец - текущий владелец файла
* Флаг "только чтение" - 0 - для чтения/записи, 1 - только для чтения. Используются в Windows.
* Флаг "скрытый" - 0 - виден, 1 - невиден в перечне файлов каталога (по умолчанию). Используются в Windows.
* Флаг "системный" - 0 - нормальный, 1 - системный. Используются в Windows.
* Флаг "архивный" - готов или нет для архивации (не путать сжатием). Используются в Windows.
* Флаг "сжатый" - файл сжимается (подобие zip архивов). Используются в Windows.
* Флаг "шифрованный" - используется алгоритм шифрования. Если кто-то попытается прочесть файл, не имеющий на это прав, он не сможет его прочесть. Используются в Windows.
* Флаг ASCII/двоичный - 0 - ASCII, 1 - двоичный
* Флаг произвольного доступа - 0 - только последовательный, 1 - произвольный доступ
* Флаг "временный" - 0 - нормальный, 1 - для удаления файла по окончании работы процесса
* Флаг блокировки - блокировка доступа к файлу. Если он занят для редактирования.
* Время создания - дата и время создания. Используются UNIX.
* Время последнего доступа - дата и время последнего доступа
* Время последнего изменения - дата и время последнего изменения. Используются в Windows и UNIX.
* Текущий размер - размер файла. Используются в Windows и UNIX.

**2.1.6 Операции с файлами**

Основные системные вызовы для работы с файлами:

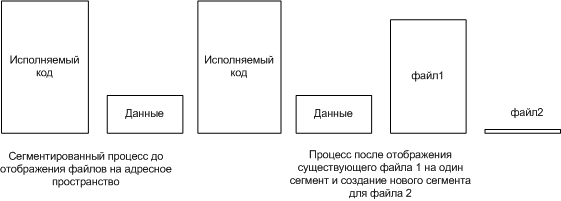
* Create - создание файла без данных.
* Delete - удаление файла.
* Open - открытие файла.
* Close - закрытие файла.
* Read - чтение из файла, с текущей позиции файла.
* Write - запись в файл, в текущею позицию файла.
* Append - добавление в конец файла.
* Seek - устанавливает файловый указатель в определенную позицию в файле.
* Get attributes - получение атрибутов файла.
* Set attributes - установить атрибутов файла.
* Rename - переименование файла.

**2.1.7 Файлы, отображаемые на адресное пространство памяти**

Иногда удобно файл отобразить в памяти (не надо использовать системные вызовы ввода-вывода для работы с файлом), и работать с памятью, а потом записать измененный файл на диск.

При использовании страничной организации памяти, файл целиком не загружается, а загружаются только необходимые страницы.

При использовании сегментной организации памяти, файл загружают в отдельный сегмент.



Пример копирования файла через отображение в памяти.

Алгоритм:

1. Создается сегмент для файла 1
2. Файл отображается в памяти
3. Создается сегмент для файла 2
4. Сегмент 1 копируется в сегмент 2
5. Сегмент 2 сохраняется на диске

Недостатки этого метода:

* Тяжело определить длину выходного файла
* Если один процесс отобразил файл в памяти и изменил его, но файл еще не сохранен, второй процесс откроет это же файл, и будет работать с устаревшим файлом.
* Файл может оказаться большим, больше сегмента или виртуального пространства.