# Архитектура вычислительных систем.

Лекция 6.

## Файловый ввод-вывод

- Москва, ноябрь 2012

Ловецкий К.П.

# Общие понятия файловых систем

- Под файлом в терминологии, связанной с компьютерами, обычно понимается некий набор хранимых на внешнем запоминающем устройстве данных, имеющий имя.
- Подсистема операционной системы, отвечающая за хранение информации на внешних запоминающих устройствах в виде именованных файлов, называется файловой системой.
- Термин файловая система иногда используется в совершенно ином значении, а именно, для обозначения структур данных, создаваемых на внешнем запоминающем устройстве с целью организации хранения на этом устройстве данных в виде именованных файлов. Обычно из контекста можно однозначно определить, какая из двух «файловых систем» имеется в виду, так что путаницы такая перегрузка термина не создает.

# Общие понятия файловых систем

- **Файловая система**
- *Файловая система* - это часть операционной системы, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске, и обеспечить совместное использование файлов несколькими пользователями и процессами.
- **В широком смысле понятие "файловая система" включает:**
- совокупность всех файлов на диске,
- наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске,
- комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

# Общие понятия файловых систем

- **Фа́йловая систе́ма** (*file system*) — регламент, определяющий способ организации, хранения и именования данных на носителях информации. Она определяет формат физического хранения информации, которую принято группировать в виде файлов. Конкретная файловая система определяет **размер имени файла**, **максимальный** возможный **размер файла**, **набор атрибутов** файла. Некоторые файловые системы предоставляют сервисные возможности, например, разграничение доступа или шифрование файлов.
- Файловая система связывает носитель информации, с одной стороны, и API (Application Programming Interface) для доступа к файлам — с другой. Когда прикладная программа обращается к файлу, она не имеет никакого представления о том, каким образом расположена информация в конкретном файле, также, как и на каком физическом типе носителя он записан. Всё, что знает программа — это имя файла, его размер и атрибуты. Эти данные она получает от драйвера файловой системы. Именно файловая система устанавливает, где и как будет записан файл на физическом носителе (например, жёстком диске).

# Общие понятия файловых систем

- С точки зрения операционной системы, весь диск представляет из себя набор кластеров размером от 512 байт и выше (*В современных - 2010 год - жёстких дисках, размер кластера стал 4096 байт*). Драйверы файловой системы организуют кластеры в файлы и каталоги (реально являющиеся файлами, содержащими список файлов в этом каталоге). Эти же драйверы отслеживают, какие из кластеров в настоящее время используются, какие свободны, какие помечены как неисправные.

- Однако файловая система не обязательно напрямую связана с физическим носителем информации. Существуют виртуальные файловые системы, а также сетевые файловые системы, которые являются лишь способом доступа к файлам, находящимся на удалённом компьютере.

# Имена файлов

**Ранние файловые системы** позволяли давать файлам **имена** только **ограниченной длины**, причем эти имена находились в одном общем пространстве имен; естественно, от имен требовалась **уникальность**. Ясно, что такой подход может работать лишь до тех пор, пока файлов на одном устройстве сравнительно немного. С ростом объемов дисковых накопителей потребовалась более гибкая схема именования.

Чтобы создать такую гибкость, ввели понятие **директории**, или **каталога**. Каталог представляет собой особый тип файла, хранящий имена файлов (некоторые из которых, возможно, сами являются каталогами). При создании на диске файловой системы создается один каталог, называемый корневым каталогом. При необходимости можно создать дополнительные каталоги, а их имена записать в корневой. Такие каталоги иногда называют каталогами первого уровня (вложенности). В них, в свою очередь, тоже можно создавать каталоги (второго уровня), и т.д.

# Краткое и полное имя файлов

При использовании каталогов можно говорить отдельно о **кратком** или **локальном имени файла** (то имя, которое файл имеет в «своем» каталоге) и о **полном** имени файла или полном пути к файлу (строка символов, включающая последовательно имена каталогов первого, второго и т.д. уровней, а затем имя самого файла в последнем из каталогов).
Имена каталогов и файла обычно разделяются специальным символом; в системах MS-DOS и Windows это обратная косая черта («\»), в системе Unix − прямая косая черта («/»). Так, если мы создадим каталог первого уровня work, в нем создадим каталог второго уровня project, в котором разместим файл program.c, то полный путь к этому файлу в ОС Unix будет выглядеть так: /work/project/program.c.

# Обращение к файлам

- Обычно в каждом каталоге существует файл со специальным именем, обозначающий **каталог более высокого уровня**, содержащий данный каталог (так называемый **родительский каталог**). В ОС Unix, а позднее и в MS-DOS и Windows, в качестве такого имени используется «..» (две точки).
- При работе с файлами некоторый каталог на диске тем или иным способом объявляется **текущим**. К файлам, находящимся в текущем каталоге, можно обращаться без указания пути, используя только краткое имя.
- Для обращения к файлам, находящимся **вне текущего каталога**, можно использовать как полный путь, так и относительный путь, состоящий из имен каталогов, первый из которых находится в текущем, второй − в первом, и т.д. При необходимости можно использовать специальное имя, ссылающееся на родительский каталог.
- Операционные системы отличают полные пути от относительных по наличию символа-разделителя (прямая или обратная косая черта, в зависимости от ОС) перед первым именем в пути.
  *Если этот символ в начале пути присутствует, то мы имеем дело с полным путем, если нет − то с относительным.*

# Иерархия каталогов

- Практически всегда файлы на дисках объединяются в *каталоги*.
- В простейшем случае все файлы на данном диске хранятся в одном каталоге. Такая *одноуровневая* схема использовалась в CP/M и первых версиях MS-DOS. *Иерархическая файловая система* со вложенными друг в друга каталогами впервые появилась в UNIX.

Wiki.txt
Tornado.jpg
Notepad.exe

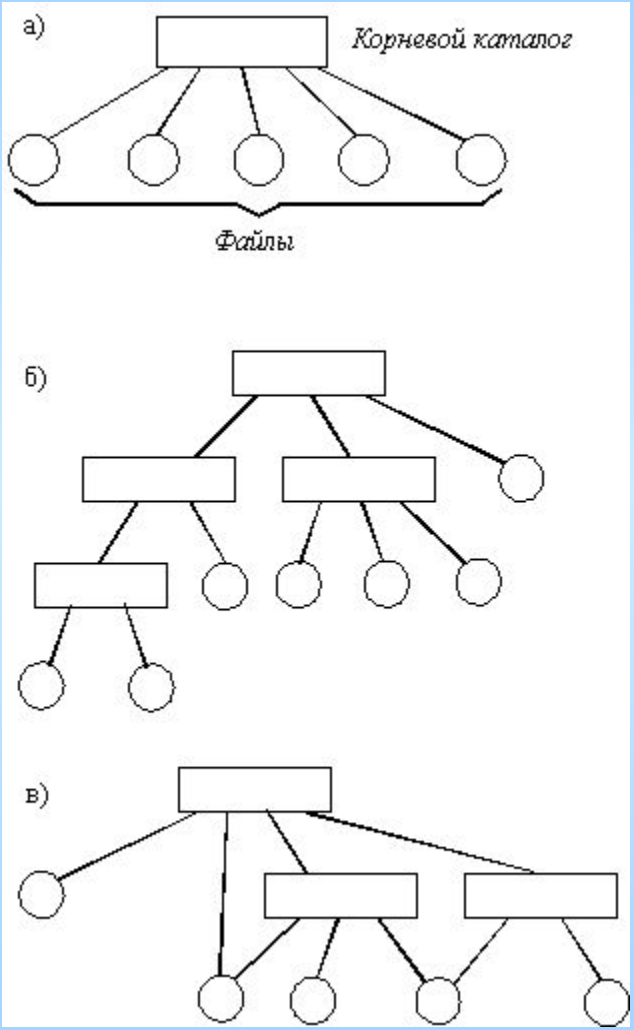
(Одноуровневая файловая система)

C:
\Program files
\CDEx
\CDEx.exe
\CDEx.hlp
\mppenc.exe
\Мои документы
\Wiki.txt
\Tornado.jpg
D: \Music \ABBA
\1974 Waterloo
\1976 Arrival
\Money, Money, Money.ogg
\1977 The Album

(Иерархическая файловая система Windows/DOS)

# Иерархия каталогов

- Иерархия каталогов может быть деревом или сетью. Каталоги образуют дерево, если файлу разрешено входить только в один каталог, и сеть - если файл может входить сразу в несколько каталогов. В MS-DOS каталоги образуют древовидную структуру, а в UNIX'е - сетевую. Как и любой другой файл, каталог имеет символьное имя и однозначно идентифицируется составным именем, содержащим цепочку символьных имен всех каталогов, через которые проходит путь от корня до данного каталога.




*Логическая организация файловой системы*
*а - одноуровневая; б - иерархическая (дерево); в - иерархическая (сеть)*

# Задачи файловой системы

- Основные функции любой файловой системы нацелены на решение следующих задач:
  - именование файлов;
  - программный интерфейс работы с файлами для приложений;
  - отображения логической модели файловой системы на физическую организацию хранилища данных;
  - устойчивость файловой системы к сбоям питания, ошибкам аппаратных и программных средств;
  - содержание параметров файла, необходимых для правильного его взаимодействия с другими объектами системы (ядро, приложения и пр.)
- В многопользовательских системах появляется еще одна задача: защита файлов одного пользователя от несанкционированного доступа другого пользователя.

# Файловая система ОС Unix

*На самом деле, в DOS/Windows системах также, как и в UNIX-подобных существует один корневой каталог со вложенными директориями, имеющими названия "c:", "d:" и т.д. В эти каталоги монтируются разделы жёсткого диска. Т.е., c:\ - это всего лишь ссылка на file:///c:/. Однако, в отличие от UNIX-подобных файловых систем, в Windows запись в корневой каталог запрещена, как и просмотр его содержимого.*

В **UNIX** существует только один корневой каталог, а все остальные файлы и каталоги вложены в него. Чтобы получить доступ к файлам и каталогам на каком-нибудь диске, необходимо *примонтировать* этот диск командой mount. Например, чтобы открыть файлы на CD, нужно, говоря простым языком, сказать операционной системе: «возьми файловую систему на этом компакт-диске и покажи её в каталоге /mnt/cdrom». Все файлы и каталоги, находящиеся на CD, появятся в этом каталоге /mnt/cdrom, который называется *точкой монтирования* (англ. *mount point*). В большинстве UNIX-подобных систем съёмные диски (дискеты и CD), флеш-накопители и другие внешние устройства хранения данных монтируют в каталог /mnt, /mount или /media. Unix и UNIX-подобные операционные системы также позволяет автоматически монтировать диски при загрузке операционной системы.

# Монтирование

- В отличие от многих других операционных систем, в ОС Unix файловая система представляет собой единое дерево каталогов. В имя файла ни в каком виде не входит имя устройства, на котором этот файл находится (то есть ничего аналогичного привычным для пользователей Windows обозначениям A:, C: и т.п. в ОС Unix нет).
- В случае, если в системе имеется несколько дисков, один из них объявляется корневым, а остальные монтируются в тот или иной каталог, называемый точкой монтирования (англ. ***mount point***), при этом для указания полных путей к файлам на этом диске необходимо к полному имени файла в рамках диска добавить спереди полный путь точки монтирования. К примеру, если у нас есть дискета, на ней создан каталог ***work***, в нем − файл ***prog.c***, а сама дискета смонтирована под каталог ***/mnt/floppy***, полный путь к нашему файлу будет выглядеть так: ***/mnt/floppy/work/prog.c***.

# Имена файлов и индексные дескрипторы

- В ОС Unix каталоги хранят только имя файла и некоторый номер, позволяющий идентифицировать соответствующий файл. Вся прочая информация о файле, включая его размер, расположение на диске, даты создания, модификации и последнего обращения, данные о владельце файла и о правах доступа к нему связываются не с именем файла (как это делается во многих других операционных системах), а с вышеупомянутым номером.
- Хранимая на внешнем запоминающем устройстве (диске) структура данных, содержащая всю информацию о файле, исключая его имя, называется индексным дескриптором (***index node***, или ***i-node***). Индексные дескрипторы имеют номера, уникальные в рамках файловой системы данного диска. Именно номер файлового дескриптора и хранится в каталоге вместе с именем файла.

# Имена файлов и индексные дескрипторы

- Отметим, что имя файла в ОС Unix может быть достаточно длинным (обычно ограничение составляет 255 символов) и содержать, вообще говоря, любые символы кроме нулевого и символа-разделителя. Так, имя файла из пятнадцати точек является с точки зрения Unix вполне допустимым.
- Тем не менее, настоятельно не рекомендуется использование в именах файлов таких символов, как ***пробел, звездочка, восклицательный и вопросительный знаки***, хотя это и возможно. Также рекомендуется воздержаться от использования в именах файлов ***спецсимволов*** (такие как перевод строки, табуляция, звонок, backspace и пр.) и символов с кодом, превышающим 127 (таких, как ***русские буквы***).
- Наконец, имя файла крайне не рекомендуется начинать с символа «-» (минус).
- ***Несоблюдение этих рекомендаций приводит к возникновению проблем в работе. Проблемы такого рода всегда могут быть преодолены, однако преодолимость трудностей не является поводом для их создания.***

# Жесткие ссылки

- В ОС Unix допускается, чтобы два или более имен файлов, расположенных как в разных каталогах, так и в одном, ссылались на один и тот же номер индексного дескриптора.
- Ясно, что создается файл под одним определенным именем. Дополнительные имена файл может получить позже с помощью системного вызова **int link**(const char *oldpath, const char *newpath); где **oldpath** − существующее имя файла, **newpath** − новое имя. Такие имена называются жесткими ссылками (англ. **hardlinks**). Отличить жесткую ссылку от оригинального имени файла невозможно: эти имена совершенно равноправны.
- Ясно, что жесткая ссылка может быть установлена только в рамках одного диска; действительно, **нумерация** индексных **дескрипторов у каждого диска своя**, так что сослаться на индексный дескриптор другого диска не представляется возможным.

# Жесткие ссылки

- В индексном дескрипторе содержится, кроме всего прочего, счетчик количества ссылок на данный дескриптор. При создании файла этот счетчик устанавливается в единицу, при создании новой жесткой ссылки − увеличивается на единицу.
- Функция, предназначенная для удаления файла, называется **unlink()**, что иногда вызывает удивление у программистов, плохо знакомых с ОС Unix.
- В ОС Unix эта функция является системным вызовом, удаляющим ссылку на файл, что и объясняет причины такого названия. При выполнении вызова ***unlink()*** имя удаляется из каталога, а счетчик ссылок в соответствующем индексном дескрипторе уменьшается. Сам файл удаляется только в случае, если удаленная ссылка была последней (счетчик обратился в нуль), и при этом файл не был ни одним из процессов открыт на запись или чтение. Если счетчик обратился в нуль, но файл кем-то открыт, удален он будет только после закрытия.

# Типы файлов. Символические ссылки

- Каталоги в файловой системе ОС Unix являются не более чем файлами специального типа. Вообще говоря, информацию, содержащуюся в каталоге (имена и номера индексных дескрипторов), необходимо где-то хранить. Поэтому все, чем на низком уровне отличается каталог от обычного файла − это значение признака типа в индексном дескрипторе. В остальном хранение на диске каталога организовано точно так же, как и хранение обычного файла.
- Кроме обычных файлов и каталогов, операционные системы обычно поддерживают и другие специальные типы файлов. Так, в файловых системах семейства ***FAT*** (MSDOS, Windows и некоторые другие ОС) примером такого специального типа файла может быть метка тома (**volume label**).
- В ОС Unix поддерживается сравнительно большое количество разновидностей файлов специального типа: файлы байт-ориентированных и блок-ориентированных устройств, имена сокетов, именованные каналы (**FIFO**) и, наконец, символические ссылки.

# Символические ссылки

- Символическая ссылка (англ. **symbolic link** ) представляет собой файл специального типа, содержащий имя другого файла. Операция открытия символической ссылки на чтение или запись приводит на самом деле к открытию файла, на который она ссылается, а не ее самой.
- В отличие от жесткой ссылки, символическая ссылка легко отличима от основного имени файла. Символическая ссылка имеет свой собственный номер индексного дескриптора и имеет свой тип. Создание и удаление символической ссылки никак не затрагивает ни файл, на который она ссылается, ни его индексный дескриптор. Более того, файл, на который указывает ссылка, может вообще не существовать в момент ее создания, или может быть удален позднее.
- Символические ссылки создаются вызовом ***int symlink***(const char *oldpath, const char *newpath); очень похожим на уже рассматривавшийся вызов **link()**. Удаление символической ссылки происходит уже рассмотренным вызовом **unlink()**.

# Права доступа к файлам

- Права доступа к файлу (англ. **access permissions**) определяют, кто из пользователей (точнее, процессов) какие операции может с данным файлом произвести.
- Права хранятся в индексном дескрипторе в виде 12-битного слова. Младшие 9 бит этого слова объединены в три группы по три бита; каждая группа задает права доступа для владельца файла, для группы владельцев и для всех остальных пользователей. Три бита в каждой группе отвечают за право чтения файла, право записи в файл и право исполнения файла.
- Чтобы узнать права доступа к тому или иному файлу, можно воспользоваться командой **ls -l**, например:
- $ ls -l /bin/cat
- -rwxr-xr-x 1 root root 14232 Feb 4 2003 /bin/cat

# Права доступа к файлам

- Чтобы узнать права доступа к тому или иному файлу, можно воспользоваться командой **ls -l**, например:
- $ ls -l /bin/cat
- -rwxr-xr-x 1 root root 14232 Feb 4 2003 /bin/cat
- Расположенная в начале строки группа символов -**rwx*r-x*r-x** показывает тип файла (первый символ − «минус» означает, что мы имеем дело с обыкновенным файлом, буква d означала бы каталог и т.п.) и права доступа, соответственно, для владельца (в данном случае rwx, т.е. чтение, запись и исполнение), группы и всех остальных (в данном случае r-x, т.е. права на запись отсутствуют).
- Таким образом, файл /bin/cat доступен любому пользователю на чтение и исполнение, но модифицировать его может только пользователь root (т.е. администратор).

# Системные вызовы для работы с файлами

***Открытие файла***

- Чтобы начать работу с файлом, его необходимо открыть, то есть объявить операционной системе, что наша программа собирается работать с данным файлом. Это делается системным вызовом

**int open(const char *name, int mode);**

**int open(const char *name, int mode, int perms);**

- Параметр **name** задает имя файла, который мы хотим открыть. Это может быть короткое имя файла, находящегося в текущем каталоге или же путь к файлу (как полный, так и относительный).
- Параметр **mode** задает режим, в котором мы намерены работать с файлом.

# Системные вызовы для работы с файлами

- Основными режимами являются **O_RDONLY** (только чтение), **O_WRONLY** (только запись) и **O_RDWR** (чтение и запись). Одну их этих трех констант указать необходимо.
- Кроме перечисленных основных констант, существуют еще и модифицирующие константы, которые при необходимости можно добавить к основным, используя операцию побитового «или». К этим константам относятся:
- • **O_APPEND** − открыть файл на добавление в конец: каждая операция
- записи будет осуществлять запись в конец файла;
- • **O_CREAT** − если файла не существует, разрешить операционной системе его создание;
- • **O_TRUNC** − если файл существует, перед началом работы сбросить его старое содержимое, в результате чего длина файла станет нулевой; запись начнется с начала файла;
- • **O_EXCL** − (используется только в сочетании с O_CREAT) потребовать от операционной системы создания нового файла; если файл уже существует, не открывать его, выдать ошибку;
- Существуют и некоторые другие модификаторы.
- Третий параметр (**perms**) следует задавать только в случае, если значение второго параметра предполагает возможность создания файла (то есть если используется **O_CREAT**). Этот параметр задает права доступа для создаваемого файла.

# Системные вызовы для работы с файлами

Чтение из файла (или, говоря более широко, из любого потока ввода)

производится вызовом **int read(int fd, void *buf, int len);**

Параметр **fd** задает файловый дескриптор; **buf** указывает на буфер, в который следует поместить прочитанные данные; **len** сообщает вызову размер буфера, чтобы избежать его переполнения.

Вызов **read**() пытается прочитать из заданного потока **len** байт данных.

Если в указанном потоке отсутствуют данные, готовые к прочтению, вызов блокирует вызвавший процесс до тех пор, пока данные не появятся, и только после их прочтения вернет управление. Если данные присутствуют, но их менее чем **len** байт, вызов сохранит их в **buf** и вернет управление.

Вызов возвращает −1 в случае ошибки. В случае успешного чтения возвращается положительное число, означающее количество прочитанных байт. Естественно, это число не может быть больше **len**.

Особым случаем является возвращаемое значение 0. Если **read**() вернул ноль, это означает, что в заданном потоке ввода наступила ситуация конца файла. В частности, для обычного файла это означает, что мы дочитали до его конца и больше там читать нечего.

# Системные вызовы для работы с файлами

- Для записи в файл (или другой поток вывода) можно пользоваться вызовом **int write(int fd, const void *buf, int len)**;
- Параметр **fd** задает файловый дескриптор; **buf** указывает на буфер, содержащий данные, которые необходимо записать в файл (или другой поток вывода); **len** задает количество этих данных.
- Вызов возвращает −1 в случае ошибки. В случае успешного чтения возвращается положительное число, означающее количество записанных байт. Естественно, это число не может быть больше **len**. В большинстве случаев число записанных байт в точности равняется значению **len**, однако полагаться на это опасно. ***В корректно написанной программе значение, возвращаемое вызовом write(), обязательно должно проверяться.***

# Системные вызовы для работы с файлами

- Закрытие файла
- После окончания работы с файлом его следует закрыть. Это особенно важно, поскольку дескрипторы представляют собой ограниченный ресурс (попросту говоря, их общее количество в системе не может превышать некоторого числа, как и количество дескрипторов, открытых одним процессом).
- Закрытие файла производится вызовом **int close(int fd)**; где **fd** − дескриптор, подлежащий закрытию. Вызов возвращает 0 в случае успеха, −1 в случае ошибки.
- Отметим, что при завершении процесса все его дескрипторы закрываются автоматически.

# Системные вызовы для работы с файлами

***Позиционирование***

- При выполнении операций чтения и записи доступ автоматически осуществляется к последовательным порциям данных в файле. Допустим, мы открыли файл на чтение. Если теперь начать вызывать **read()** с параметром **len**, равным **100**, то первый вызов прочитает из файла байты с нулевого по 99й, второй − с 100го по 199й, третий − байты с 200го по 299й и т.д.
- Этот порядок можно при необходимости нарушить, изменив значение текущей позиции, связанной с файловым дескриптором, вызвав **int lseek(int fd, int offset, int whence);**
- Параметр **fd** задает номер файлового дескриптора. Параметр **offset** указывает, на сколько байт следует сместиться, и параметр **whence** определяет, от какого места эти байты следует отсчитывать. При значении **whence**, равном константе SEEK_SET, отсчет пойдет от начала файла; при значении SEEK_CUR − от текущей позиции, и при значении SEEK_END − от конца файла. Вызов процедуры возвращает новое значение текущей позиции, считая от начала.

# Файлы устройств и классификация устройств

- **Обобщенное понятие файла**
- Одним из замечательных свойств ОС Unix является обобщенная концепция файла как универсальной абстракции. Практически любое внешне устройство представляется на пользовательском уровне как файл специального типа. Это касается, в частности, и жестких дисков, и всевозможных последовательных и параллельных портов, и виртуальных терминалов, и т.п.
- Так, часто возникает задача создания образа компакт-диска (CD). Такое может потребоваться, например, при создании копии диска. В некоторых других операционных системах для проведения такой операции необходимо специальное программное обеспечение. Что касается ОС Unix, то тут обычно достаточно вставить диск в привод и дать команду
- cat /dev/cdrom > image.iso
- Точно так же, чтобы, скажем, отправить файл на печать, достаточно команды
- cat myfile.ps > /dev/lp0
- Конечно, обычно так не делают, полагаясь на подсистему печати, однако нам в данном случае важнее сам факт такой возможности.

# Два типа устройств

- Устройства, для которых имеется представление в виде файла, делятся на два типа: символьные (или потоковые) и блочные. Для обозначения тех же понятий могут использоваться термины «байт-ориентированные» и «блок-ориентированные» устройства.
- Основными операциями над байт-ориентированными устройствами являются запись и чтение одного (очередного) символа (байта). В противоположность этому, блок-ориентированные устройства воспринимаются как хранилище данных, разделенных на блоки фиксированного размера; основными операциями, соответственно, являются чтение и запись заданного блока.
- Примерами байт-ориентированных устройств являются терминал (клавиатура и устройство отображения), принтер, манипулятор «мышь», звуковая карта.

# Типы устройств

- Существуют также чисто виртуальные символьные устройства, не имеющие физического воплощения.
- Так, устройство **/dev/null** позволяет производить в него запись любой информации, которая попросту игнорируется; попытка читать из этого устройства сразу вызывает ситуацию «конец файла».
- Устройство **/dev/zero** позволяет прочитать любое количество байт, причем все прочитанные байты будут равны нулю.
- Устройство **/dev/random** выдает читающему процессу последовательность псевдослучайных чисел, и т.п.

# Типы устройств

- В виде блок-ориентированных устройств обычно представляются диски и другие подобные им устройства. Действительно, дисковые устройства обычно позволяют чтение или запись только целыми секторами, что обусловлено особенностями физической реализации таких устройств.
- Наряду с директориями и символическими ссылками, блок-ориентированные и байт-ориентированные устройства представляют собой еще два специальных типа файлов.

# Th-th-th-that's all folks!