# Архитектура ЭВМ. Операционные системы

Власов Евгений

# Файл

сущность, позволяющая получить доступ к какому- либо ресурсу вычислительной системы.
В большинстве случаев файл определяет именованную область на устройстве хранения данных. В unix-like операционных системах файлом является большинство ресурсов вычислительной системы.

В UNIX и POSIX- системах существуют следующие типы файлов []

- Обычный файл
- Каталог
- FIFO-файл
- Символическая ссылка
- Байт-ориентированный файл устройства
- Блок-ориентированный файл устройства.

***Обычный файл*** может быть тестовым и двоичным. В unix системах эти типы файлов не различаются и оба могут быть «исполняемыми» при условии, что на них установлено разрешение на выполнение и они могут читаться и записываться пользователем, имеющим соответствующие права доступа.

**FIFO-файл** является специальным файлом, который предназначен для организации обмена данными между процессами.

**Символическая ссылка** содержит путевое имя, которое обозначает другой файл в файловой системе.

***Каталог*** содержит файлы и каталоги. Концепция каталога позволяет организовать файлы в некоторую иерархическую структуру. В Unix ситемах базовым является каталог ‘/’.

***Блок-ориентированные файл устройства*** служит для представления физического устройства, которое передает данные блоками. Примерами таких устройств являются жесткие диски.

***Байт-ориентированный файл устройства*** служит представления физического устройства, которое передает данные побайтно, например, модем.

# Файловый дескриптор

Информация о файлах, используемых процессом, входит в состав его системного контекста и хранится в его блоке управления – PCB. В операционной системе UNIX можно упрощенно полагать, что информация о файлах, с которыми процесс осуществляет операции потокового обмена, наряду с информацией о потоковых линиях связи, соединяющих процесс с другими процессами и устройствами ввода-вывода, хранится в некотором массиве, получившем название таблицы открытых файлов или таблицы файловых дескрипторов. Индекс элемента этого массива, соответствующий определенному потоку ввода-вывода, получил название файлового дескриптора для этого потока.

*Дескриптор файла* – это индекс открытого файла в таблице дескрипторов файлов.

Файловый дескриптор представляет собой небольшое целое неотрицательное число, которое для текущего процесса в данный момент времени однозначно определяет некоторый действующий канал ввода-вывода. Некоторые файловые дескрипторы на этапе старта любой программы ассоциируются со стандартными потоками ввода-вывода. Файловый дескриптор 0 соответствует стандартному потоку ввода, файловый дескриптор 1 – стандартному потоку вывода, файловый дескриптор 2 – стандартному потоку для вывода ошибок. В нормальном интерактивном режиме работы стандартный поток ввода связывает процесс с клавиатурой, а стандартные потоки вывода и вывода ошибок – с текущим терминалом.

# API работы с файлами

| Название системного вызова | Описание |
|---|---|
| ***open()*** | Открывает(создает) файл для доступа к данным |
| ***read ()*** | Считывает данные из открытого файла |
| ***write ()*** | Записывает данные в открытый̆ файл |
| ***lseek ()*** | Устанавливает позицию для чтения/записи в открытом файле |
| ***close ()*** | Закрывает открытый̆ файл |
| ***stat, fstat()*** | Запрашивает атрибуты файла |

```
#include <sys/types.h>
#include <fcntl.h>
int open(const char *path_name, int access_mode,
mode_t permission);
```

Системный вызов `open()` устанавливает соединение между процессом и файлом. Этот системный вызов позволяет создавать файлы. В случае успешного выполнения функция `open()` возвращает дескриптор файла. Все другие системные вызовы для работы с файлами используют файловый дескриптор, полученный после выполнения `open()`.

Параметр `access_mode` может быть составлен как битовая маска из следующих макросов

- O_RDONLY – открыть файл для чтения;
- O_WRONLY – открыть файл для записи;
- O_RDWR – открыть файл чтения и записи.

Параметр `permission` – необходим только в том случае, если в access_mode установлен флаг O_CREAT. Он задает права доступа к файлу для его владельца, членов группы и все остальных пользователей.

Каждое из этих значений может быть скомбинировано посредством операции «побитовое или ( | )» с одним или несколькими флагами:

- O_CREAT – создать файл, если файла с таким именем не существует;
- O_EXCL – применяется совместно с флагом O_CREAT. При совместном их использовании и существовании файла с указанным именем, открытие файла не производится и констатируется ошибочная ситуация;
- O_NDELAY – запрещает перевод процесса в состояние ожидание при выполнении операции открытия и любых последующих операциях над этим файлом;
- O_APPEND – при открытии файла и перед выполнением каждой операции записи (если она, конечно, разрешена) указатель текущей позиции в файле устанавливается на конец файла;
- O_TRUNC – если файл существует, уменьшить его размер до 0, с сохранением существующих атрибутов файла, кроме, быть может, времен последнего доступа к файлу и его последней модификации.

Кроме того, в некоторых версиях операционной системы UNIX могут применяться дополнительные значения флагов:

- O_SYNC – любая операция записи в файл будет блокироваться (т. е. процесс будет переведен в состояние ожидание) до тех пор, пока записанная информация не будет физически помещена на соответствующий нижележащий уровень hardware;
- O_NOCTTY – если имя файла относится к терминальному устройству, оно не становится управляющим терминалом процесса, даже если до этого процесс не имел управляющего терминала.

Параметр permission задается как сумма следующих восьмеричных значений:

- 0400 – разрешено чтение для пользователя, создавшего файл;
- 0200 – разрешена запись для пользователя, создавшего файл;
- 0100 – разрешено исполнение для пользователя, создавшего файл;
- 0040 – разрешено чтение для группы пользователя, создавшего файл;
- 0020 – разрешена запись для группы пользователя, создавшего файл;
- 0010 – разрешено исполнение для группы пользователя, создавшего

файл;

- 0004 – разрешено чтение для всех остальных пользователей;
- 0002 – разрешена запись для всех остальных пользователей;
- 0001 – разрешено исполнение для всех остальных

пользователей.

При создании файла реально устанавливаемые права доступа получаются из стандартной комбинации параметра mode и маски создания файлов текущего процесса umask, а именно – они равны mode & ~umask.

В случае, когда мы **требуем**, чтобы файл на диске отсутствовал и был создан в момент открытия, флаг для набора операций должен использоваться в комбинации с флагами O_CREAT и O_EXCL.

# *Системные вызовы read() и write()*

```
#include <sys/types.h>
#include <unistd.h>
size_t read(int fd, void *addr, size_t nbytes);
size_t write(int fd, void *addr,size_t nbytes);
```

Системные вызовы read() и write() предназначены для осуществления потоковых операций ввода (чтения) и вывода (записи) с файлами.

Параметр fd является файловым дескриптором, полученным с помощью системного вызова `open`().

Параметр addr представляет собой адрес области памяти, начиная с которого будет браться информация для передачи или размещаться принятая информация.

Параметр nbytes для системного вызова `write` определяет количество байт, которое должно быть передано, начиная с адреса памяти addr. Параметр nbytes для системного вызова read определяет количество байт, которое мы хотим получить из канала связи и разместить в памяти, начиная с адреса addr.

# Дополнительные системные вызовы

| Системный вызов | Описание |
|---|---|
| `mode_t  umask(mode_t newMask)` | установить те права доступа, которые необходимо автоматически маскировать (исключать) для всех файлов, создаваемых процессом |
| `off_t lseek(int fd, off_t pos, int whence);` | произвольно установить указатель позиции в открытом файле.<br>Важно! Системный вызов lseek() нельзя применять к FIFO, байт-ориентированным файлам устройств и символическим ссылкам. |
| `int close(int fd);` | Корректное завершение работы с файлом. |
| `int stat(const char *path_name, struct stat *statv);`<br>`int fstat(int fd, struct stat *statv);` | Возвращают атрибуты заданного файла. fstat принимает файловый дескриптор, полученный после вызова open, а stat символическое имя файла. |

# Блокирующий режим

По умолчанию все файловые дескрипторы в Unix-системах создаются в “блокирующем” режиме. Это означает, что системные вызовы для ввода-вывода, такие как read, write или connect, могут заблокировать выполнение программы вплоть до готовности результата операции. Например, чтение данных из потока stdin в консольной программе. Как вызывается `read` для `stdin`, выполнение программы блокируется, пока данные не будут введены пользователем с клавиатуры и затем прочитаны системой. То же самое происходит при вызове функций стандартной библиотеки, таких как fread, getchar, std::getline, поскольку все они в конечном счёте используют системный вызов read. Если говорить конкретнее, ядро переводит процесс в состояние «Ожидание», пока данные не станут доступны в псевдо-файле stdin. То же самое происходит и для любых других файловых дескрипторов.

Блокировки - это проблема для всех программ, требующих конкурентного выполнения, поскольку заблокированные потоки процесса засыпают и не получают процессорное время. Существует несколько различных, но взаимодополняющих способа устранить блокировки:

- неблокирующий режим ввода-вывода
- мультиплексирование с помощью системного API, такого как select либо epoll
- Асинхронный ввод/вывод

Эти решения часто применяются совместно, но предоставляют разные стратегии решения проблемы.