

# Лекция 20

## Файловая система

# Файловая система

- Формат хранения данных на носителе (ф.с. FAT32, NTFS, EXT4)
  - Каждая ф.с. имеет свои особенности: максимальный размер, ограничения на размеры файлов, длину имени и т. п.
- Компонент ядра ОС, отвечающий за доступ к файлам
  - API для работы с файлами и файловой системой
  - Драйвера конкретных файловых систем

# Требования к файловым системам

- Постоянство (сохранение данных после окончания процесса и после остановки ОС)
- Поддержка данных огромного размера (одного файла и суммарного)
- Эффективность (скорость поиска файла и обращения к файлу)
- Поддержка разделения прав доступа и квотирования
- Устойчивость к программному сбою
- Устойчивость к аппаратному сбою

# Задачи ядра ОС

- Предоставление стандартного интерфейса
  - POSIX API для работы с файлами и файловой системой: user space ↔ kernel space
  - VFS (virtual file system): kernel ↔ FS driver
- Управление ресурсами
  - Разграничение прав доступа к объектам ФС
  - Квотирование ресурсов ФС
  - Арбитраж параллельного доступа к ФС (блокировки, атомарность)

# Основные абстракции

- Файл — именованный набор данных (для Unix-систем — регулярный файл)
- Или файл — запись в каталоге. В Unix-системах:
  - Регулярные файлы
  - Файлы-каталоги
  - Файлы-устройства (блочные и символьные)
  - Символические ссылки
  - Именованные каналы (FIFO)
  - Сокеты

# Регулярный файл

- Регулярный файл содержит данные, находящиеся на устройстве (в блоках данных)
- Представляет собой поток байт
- Структурирование, поддержание корректности – задача программ пользовательского режима
- В некоторых случаях ядро требует файлы определенной структуры (например, загрузка ELF-файла на выполнение)

# Идентификация файлов

- В загруженной и работающей системе – пути к файлам
- В файловой системе на уровне ядра ОС – номер индексного дескриптора
- Открытый файл на уровне процесса – файловый дескриптор

# Файловый дескриптор

- Файловый дескриптор — идентификатор открытого файла в процессе
- Каждый процесс имеет свой набор файловых дескрипторов, независимый от других процессов
- Неотрицательное целое число
- Обычно при старте процесса:
  - 0 — stdin
  - 1 — stdout
  - 2 — stderr
- При выделении нового ф. д. всегда выбирается свободный с минимальным номером
- ф. д. - индекс в таблицу открытых файлов процесса



# Открытие файла

```
#include <sys/types.h>
```

```
#include <sys/stat.h>
```

```
#include <fcntl.h>
```

```
int open(const char *path, int flags, int mode);
```

- Возвращает файловый дескриптор при успехе и -1 при ошибке
- При ошибке код ошибки в errno (<errno.h>)
- path — путь к открываемому файлу

# Флаги открытия

- Основные режимы открытия:
  - O\_RDONLY — только чтение
  - O\_WRONLY — только запись
  - O\_RDWR — чтение-запись
- Флаги управления файловым дескриптором
  - O\_CLOEXEC — файловый дескриптор закрывается автоматически при exec
- Модификаторы режима записи
  - O\_APPEND — режим добавления в конец файла
  - O\_TRUNC — очистить файл

# Флаги открытия

- Модификаторы создания файла
  - O\_CREAT — создание файла
  - O\_EXCL — создание файла только в случае, если он еще не существует
- Типичные комбинации флагов
  - O\_RDONLY
  - O\_WRONLY | O\_CREAT | O\_TRUNC
  - O\_WRONLY | O\_CREAT | O\_APPEND

# Режим создания файла

- При указании флага `O_CREAT` используется параметр `mode` — права доступа на создаваемый файл
- Права доступа накладываются на параметр `umask`: `mode & ~umask`
- Например, `mode == 0666`, `umask == 0022`, права на создаваемый файл `0644`
- `Mode == 0700`, `umask == 0007`, права `0700`

# umask

- Атрибут процесса
- Указывает, какие биты прав доступа должны быть сброшены в задаваемых правах (9 основных бит)
- Могут быть получены/изменены с помощью системного вызова

`int umask(int newmask);`

- Возвращается старое значение `umask`

# Заккрытие файла

```
int close(int fd);
```

- При успехе возвращается 0, при неудаче - -1.
- Причины неудачи:
  - EBADF — неправильный файловый дескриптор
  - EINTR — операция была прервана
  - EIO — ошибка записи
- В любом случае, ничего разумного при ошибке сделать нельзя!

# Синхронизация с диском

```
int fsync(int fd);
```

- Для избежания потерь данных сохранение данных на диск не должно выполняться при закрытии
- В случае ошибки EIO вызова fsync ф. д. fd не закрыт и есть возможность ситуацию исправить

# Позиционирование в файле

- Если открытый файл является файлом произвольного доступа, текущую позицию в файле (`f_pos`) можно произвольно изменять

`off_t lseek(int fd, off_t offset, int whence);`

- Whence:
  - `SEEK_SET` — относительно начала файла
  - `SEEK_END` — относительно конца файла
  - `SEEK_CUR` — относительно текущей позиции
- Возвращается новое положение в файле относительно начала или -1 в случае ошибки



# Позиционирование в файле

- Для каналов, сокетов, символьных устройств, псевдотерминалов позиционирование невозможно!
- Позиционирование на позицию до начала файла невозможно (EINVAL)
- В файле, открытом O\_RDONLY, позиционирование после текущего конца невозможно (EINVAL)
- В файле, открытом O\_WRONLY или O\_RDWR, позиционирование после текущего конца файла допускается. Последующая операция записи заполняет пропуск между старым концом файла и текущей позицией нулевыми байтами

# 32-битные системы

- `off_t` — знаковый, 32-битный, то есть `lseek` не может работать с файлами  $> 2G$
- Чтобы работать с большими файлами:
  - `-D_FILE_OFFSET_BITS=64` в командной строке `gcc`
  - Все смещения будут 64-битными знаковыми

# Установка размера файла

```
int truncate(const char *path, off_t length);
```

```
int ftruncate(int fd, off_t length);
```

Эти системные вызовы позволяют задать новый размер файла (length). Он может быть как больше (файл дополняется нулевыми байтами), так и меньше текущего.

# Чтение

`ssize_t read(int fd, void *buf, size_t count);`

- `size_t` — **беззнаковый** целый тип размера, достаточного для хранения размера любого объекта в C/C++ (обычно на Unix это `unsigned long`)
- `ssize_t` — **знаковый** тип такого же размера, как и `size_t`
- Если `count > SSIZE_MAX`, поведение `read` не определено
- На 32-битных системах `count < 2G`

# Чтение

`ssize_t read(int fd, void *buf, size_t count);`

- Возвращает -1 при ошибке (EIO, EAGAIN, ...) - см. man 2 read
- Если `count == 0`, то выполняется проверка на ошибки и возвращается либо 0, либо -1

# Чтение

```
ssize_t read(int fd, void *buf, size_t count);
```

- Обычный случай: `count > 0`, успешное завершение (возвр. значение  $\geq 0$ )
- 0 — признак конца файла (то есть данных больше нет и не будет)
- Иначе не более чем `count` байт считано в буфер `buf` и количество байт возвращено

# Чтение

```
ssize_t read(int fd, void *buf, size_t count);
```

- Если готовых к чтению данных нет, read переведет процесс в состояние ожидания до появления данных
  - Но в режиме O\_NONBLOCK read вернет EAGAIN немедленно!
- Если есть хоть один байт готовых к чтению данных read возвращает их немедленно
- Никогда не ждет полного заполнения буфера до размера count

# Чтение

- Как правило, при работе с регулярными файлами на обычных файловых системах, если нет данных доступных немедленно, процесс переводится в состояние «uninterruptible sleep» (D-state) до получения данных
- Как правило, при этом возвращается столько данных, сколько запрошено
- **НО ПОЛАГАТЬСЯ НА ЭТО НЕЛЬЗЯ!**



# Запись

`ssize_t write(int fd, const void *buf, size_t count);`

- Возвращает -1 при ошибке
- Если `count > SSIZE_MAX`, поведение `read` не определено
- Если `count == 0` и файл регулярный, выполняется проверка на ошибки и возвращается либо 0, либо -1
- Если `count > 0`, возвращается количество записанных байт

# Запись

- Как правило, при работе с регулярными файлами на обычных файловых системах `write` записывает все за один раз и возвращает `count`
- **НО ПОЛАГАТЬСЯ НА ЭТО НЕЛЬЗЯ!**

# Разграничение доступа

- Разграничение доступа (access control) – одна из основных задач ядер операционных систем
- Процесс (активная сущность) – субъект, запрашивает разрешения на выполнения операций
- Ресурс (пассивная сущность)
- Разграничение доступа дает ответ на вопрос “может или нет”
- Как может – другие механизмы

# Разграничение доступа

- Идентификация
- Аутентификация
- Авторизация

# Идентификация

- Субъекты систем разграничения доступа должны быть идентифицированы
- На основе идентификатора субъекта принимается решение о предоставлении доступа
- У каждого процесса есть идентификатор (PID), но он плох для работы с персистентными объектами (файловой системой)

# Идентификация в POSIX

- Основной субъект - “пользователь” (user)
- Дополнительный субъект - “группа” (group): пользователь имеет одну “основную группу” и может находиться в нескольких “вторичных группах” (в Linux до 32 групп)
- Идентификатор пользователя (uid) – неотрицательное число
- Идентификатор группы (gid) – неотрицательное число

# root

- Пользователь с `uid == 0` – специальный
- Обычно он называется “root” (это не фиксировано)
- Если процесс запущен пользователем root, для него не действуют ограничения access control, такой процесс может все

# Атрибуты пользователя

- Файл с атрибутами пользователей (кроме паролей)  
- /etc/passwd – но может быть по-другому - PAM
- Файл с парольной информацией - /etc/shadow
- Идентификатор пользователя (uid)
- Идентификатор основной группы (gid)
- Имя пользователя (user name, login)
- Комментарий (например, ФИО человека)
- “Домашний” каталог (home directory)
- Командная оболочка (login shell)



# Атрибуты группы

- Идентификатор группы (gid)
- Имя группы (group name)
- Имена пользователей, у которых эта группа является вторичной

# Атрибуты процесса

- Реальный идентификатор пользователя (real user id) – (упрощенно) – кто запустил данный процесс, берется от процесса-родителя
- Эффективный и. п. (effective user id - euid) – с какими правами работает данный процесс
- Обычно `real user id == effective user id`
- Реальный идентификатор группы (real group id - gid)
- Эффективный идентификатор группы (effective group id – egid)
- Процесс может свободно переключаться между своими основной и вторичными группами

# Аутентификация

- Аутентификация – проверка подлинности
- Например, поступает команда – от имени какого пользователя она должна быть выполнена?
- Аутентификация – назначение идентификатора пользователя/группы поступающим командам
- Механизмы аутентификации:
  - По паролю
  - По ключу (public/secret key)
  - Биометрия

# Аутентификация в Unix

- В результате успешной аутентификации создается процесс корневого интерпретатора командной строки (`login shell`) с идентификатором аутент. пользователя/группы
- `Login shell` выполняет скрипты инициализации (например, `.login`)
- Все процессы для пользователя порождаются (прямо или косвенно) от `login shell`, и поэтому получают идентификатор пользователя/группы
- `Login shell` может сам переходить в интерактивный режим, может запускать другие средства взаимодействия с пользователем

# Авторизация

- Проверка прав доступа идентифицированного пользователя к заданным ресурсам
- Например, проверка возможности выполнения операции с файловой системой

# Атрибуты файла

- Владелец – идентификатор пользователя
- Группа – идентификатор группы  
(у файла только одна группа!)
- Права доступа
- Дополнительные списки прав доступа  
(Access Control List - ACL)

# Избирательное управление доступом (discretionary access control)

- Модель владелец-группа-прочие
- Если uid процесса и uid файла совпадают, берется множество прав доступа владельца (user)
- Если один из gid процесса совпадает с gid файла, берется множество прав доступа группы (group)
- Иначе берется множество прав доступа прочих (other)

# Множество прав доступа

- r,w,x — интерпретация зависит от того, является ли файл каталогом или нет
- Права для файлов:
  - "r" — право на чтение из файла (вызов системного вызова read или lseek)
  - "w" — право на запись в файл (вызов write)
  - "x" — право на выполнение файла (вызов exec\*)

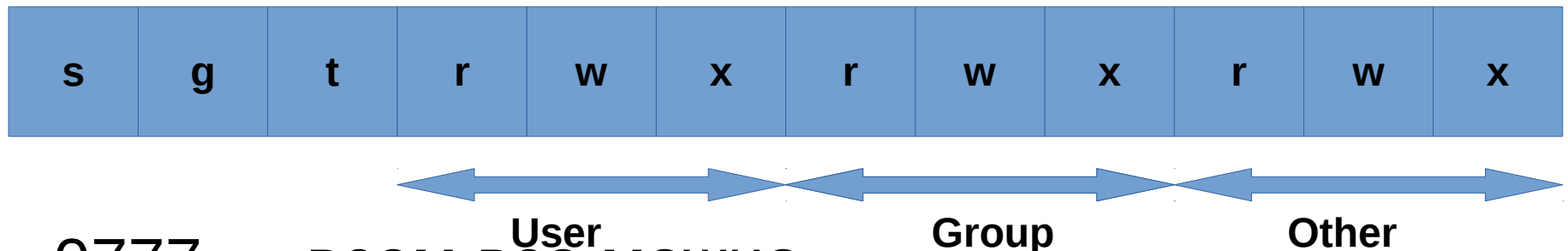


# Права доступа

- Права для каталогов
  - "r" — право читать список файлов в каталоге (вызовы opendir/readdir/...)
  - "w" — право модифицировать список файлов в каталоге (создавать, удалять, переименовывать)
  - "x" — право на поиск заданного имени в каталоге
- Права "--x" — пользователь не может посмотреть какие файлы есть в каталоге, но если он знает имя файла в нем, с этим файлом может работать

# Права доступа

- Полные права — 12 бит (9 основных + 3 доп.)



- 0777 — всем все можно
- 0664 — чтение/запись для владельца и группы, только чтение для остальных
- 0700 — все права только для владельца

# Дополнительные биты

| Бит       | Для файлов  | Для каталогов  |
|-----------|---|--|
| S (04000) | При выполнении процесс, запущенный из данного файла, может изменить свой uid на uid файла | Не используется  |
| G (02000) | При выполнении процесс, запущенный из данного файла, может изменить свой gid на gid файла | При создании новых файлов и каталогов группа наследуется из родительского каталога, а не из процесса |
| T (01000) | Не используется   | Только владелец может удалить созданный им файл  |