

## Лекция 35

# Внутреннее устройство файловой системы

# Файловый дескриптор

- Fd – универсальный способ доступа к разного типа ресурсам:
- Операции: read, write, close, dup, epoll, fcntl, ioctl
- Fd – целое число
- Индексы лучше указателей: fd – индекс в таблицу в ядре

# Таблица файловых дескрипторов

- Хранится для каждого процесса
- Находится в `include/linux/fdtable.h`

```
struct fdtable
{
    unsigned int max_fds;
    struct file **fd;           // file pointer
    unsigned long *close_on_exec; //CLOEXEC bitset
    unsigned long *open_fds;     // opened bitset
};
```

# struct file

- Состояние открытого файла
- Находится в исходном коде ядра Linux в `include/linux/fs.h`

```
struct file
{
    atomic_long_t    f_count; // счетчик ссылок
    unsigned int     f_flags; // флаги open
    fmode_t          f_mode;  // внутр. флаги
    loff_t           f_pos;    // текущее смещение
    // много всего еще
};
```

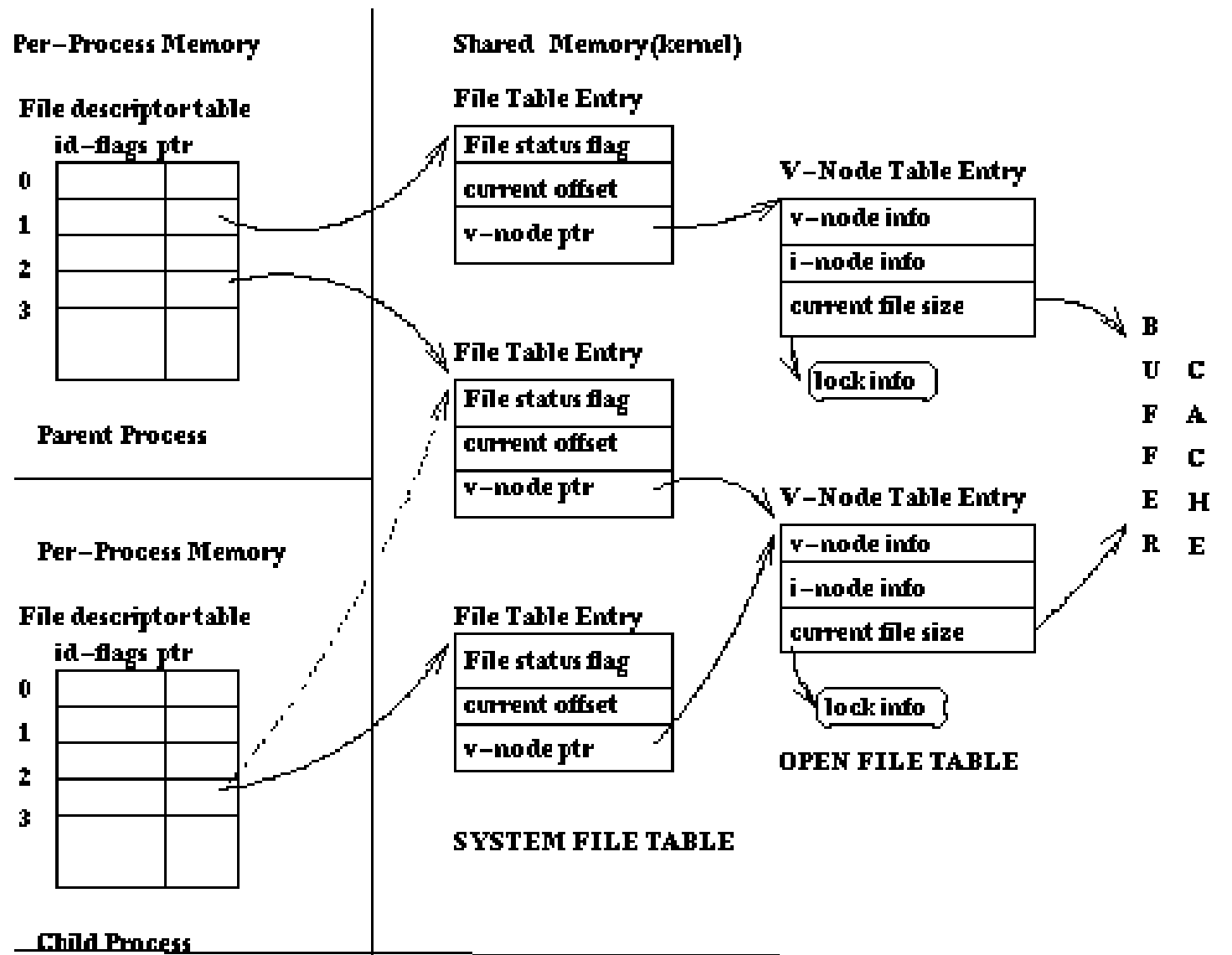
# Подсчет ссылок

- При выполнении `open` (создание первого ф. д.): `f_count = 1`
- При копировании ф. д. (`dup*` или `fork`):  
`++f_count`
- При закрытии ф. д. (`close`)  
`if (--f_count == 0) {`  
    `// реально закрыть файл:`  
    `// сохранить несохраненные данные,`  
    `// освободить ресурсы ядра`  
`}`
- Подсчет ссылок — эффективный способ управления ресурсами в случае ациклических графов

# Разделение открытого файла

- Все ф. д. - копии разделяют (имеют общую) следующую информацию:
  - Режим открытия файла
  - Текущую позицию в файле
- Открытый файл закрывается, когда закрывается последний ф. д.-копия
- Каждый ф. д. - копия имеет свое значение флага O\_CLOEXEC

# Структуры ядра



# Одновременная работа с файлами

- В Unix если одновременно несколько процессов работают с копиями одного и того же файлового дескриптора или с одним и тем же файлом, и операции чтения, и операции записи разрешены без ограничений
- Процессы должны сами согласовать свое поведение, чтобы избежать порчи данных
- Варианты: флаг O\_APPEND, рекомендательные блокировки, обязательные блокировки



# Атомарность с файлами

- POSIX не гарантирует атомарности чтения/записи при работе с файлами
- Реально Linux записывает/считывает данные небольшого (зависит от типа ФС, около 1KiB) размера атомарно, то есть при записи данные двух процессов не перемешаются
- **НО! Запись/чтение данных и изменение значения текущей позиции в совокупности могут быть не атомарны! В современных ядрах – атомарны.**

# Чтение/запись с файлами

- Процесс 1  
`write(fd, "123\n", 4);`
- Процесс 2  
`write(fd, "456\n", 4);`
- Два возможных результата:  
123  
456
- Или  
456  
123

# Блокировки файлов (file locking)

- Advisory (рекомендательная) — для процессов, которые добровольно соглашаются соблюдать блокировки
  - Процесс может игнорировать блокировки других процессов
- Mandatory (обязательная) — для любых процессов
  - Не везде поддерживаются
  - Требуют специального монтирования файловой системы

# Типы блокировки

- Read (shared) — блокировка на чтение. Несколько процессов могут заблокировать ресурс на чтение, при условии, что нет блокировок на запись
- Write (exclusive) — единственный процесс блокирует на запись, нет блокировок на чтение
- Если требуемый тип блокировки не может быть немедленно удовлетворен, процесс переводится в состояние ожидания или блокировка завершается ошибкой

# СИСТЕМНЫЙ ВЫЗОВ fcntl

```
struct flock {  
    short l_type;      /* F_RDLCK, F_WRLCK, F_UNLCK */  
    short l_whence;    /* SEEK_SET, SEEK_CUR, SEEK_END */  
    off_t l_start;     /* Starting offset for lock */  
    off_t l_len;       /* Number of bytes to lock */  
    pid_t l_pid;       /* PID of process blocking our lock  
(F_GETLK only) */  
};
```

```
struct flock flk;
```

```
int res = fcntl(fd, OPER, &flk);
```

```
// OPER – один из F_SETLK, F_SETLKW, F_GETLK
```

# Операции fcntl

- F\_SETLK
  - F\_RDLCK — заблокировать на чтение
  - F\_WRLCK — заблокировать на запись
  - F\_UNLCK — разблокировать
  - Если операция невозможна, возвращается ошибка EACCESS или EAGAIN
- F\_SETLKW
  - Те же операции блокировки
  - Если операция невозможна, процесс блокируется
- F\_GETLK
  - Проверить возможность блокировки
  - Если блокировка невозможна, получить информацию о процессе

# Особенности fcntl

- Per-process, т. е. каждый процесс может иметь только один тип блокировки на каждый конкретный байт файла
- Advisory, т. е. системные вызовы read и write не проверяют наличие и тип блокировки
- При закрытии любого файлового дескриптора, связанного с файлом, в этом процессе блокировки процесса сбрасываются

# Mandatory locking

- В Linux необходимо выполнить следующее:
  - Разрешить обязательные блокировки при монтировании  
`mount DEVICE PATH -o mand`
  - Файл не должен иметь разрешение исполнения на группу (бит 010 прав)
  - Файл должен иметь `sgid` бит (02000)
- Тогда `read write` будут проверять блокировку файлов и переводить процесс в состояние ожидания в случае конфликта



# Квотирование

- Можно квотировать (ограничить число):
  - Индексных дескрипторов (т. е. файлов)
  - Блоков данных (т. е. суммарный размер файлов)
- Типы квоты:
  - Hard — ограничение не может быть превышено
  - Soft — ограничение может быть превышено на ограниченное время (grace period)
- Квоты могут применяться к пользователю и группе

# Управление квотированием

- Квотирование включается при монтировании файловой системы  
`mount DEV PATH -o usrquota,grpquota`
- Базы данных квотирования создаются и управляются с помощью  
`quotacheck OPTIONS`
- Квота для пользователя (группы) редактируется с помощью  
`edquota`

# Типы файловых систем

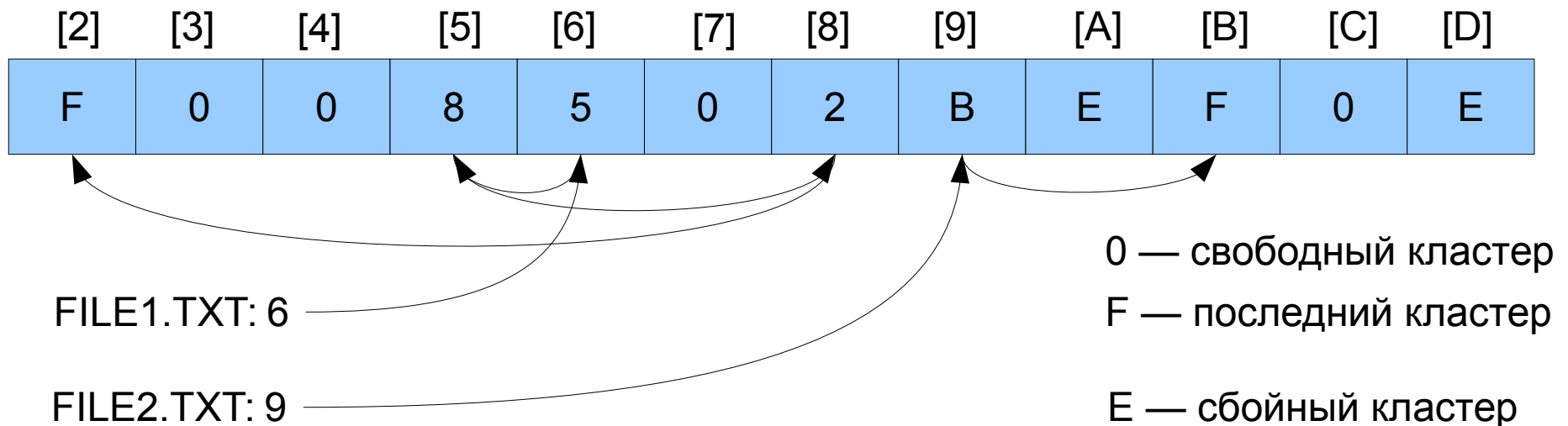
- Поддержка иерархии каталогов (иерархические/не иерархические)
- Поддержка журналирования
- Поддержка особенностей хранения данных (dvd, flash)

# Файловая система RT-11

- Одноуровневая иерархия файлов
- Файлы хранятся в непрерывных областях области данных диска
- Имена файлов — 6 + 3 заглавные латинские буквы и цифры (кодируется в 6 байтах)
- Записи о файлах в каталоге диска располагаются в порядке размещения файлов в области данных диска, специальные записи для «дыр»

# FAT

- Иерархическая файловая система
- Имена файлов: 8 + 3 (символы занимают один байт)
- Таблица размещения файлов:



# FAT

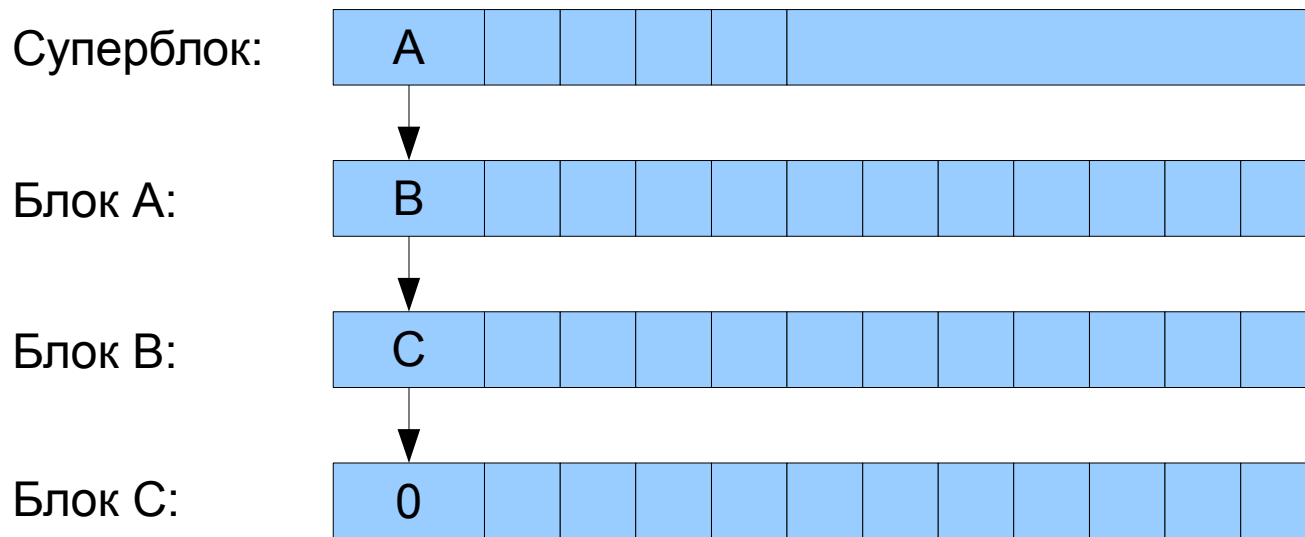
- Метаинформация о файле хранится в записях каталога
- FAT16: Максимальный размер кластера — 32 KiB (при размере блока 512 байт — 64 блока на кластер)
- Для надежности на диске хранится две копии FAT
- Для эффективной работы в памяти приходится держать FAT целиком
- Фрагментация файлов

# UNIX System V FS (s5fs)

Загрузчик	Суперблок	Область инд. дескр.	Область данных
-----------	-----------	---------------------	----------------

- Суперблок хранит информацию о файловой системе:
  - Размер файловой системы в блоках
  - Размер области индексных дескрипторов (inode) в блоках
  - Число свободных блоков и инд. дескр.
  - Номер первого свободного инд. дескр.
  - Список свободных блоков данных (частично)
- Загружается в память при монтировании

# Список свободных блоков



- При удалении блока он добавляется в начало списка
- При выделении блока он берется из начала списка



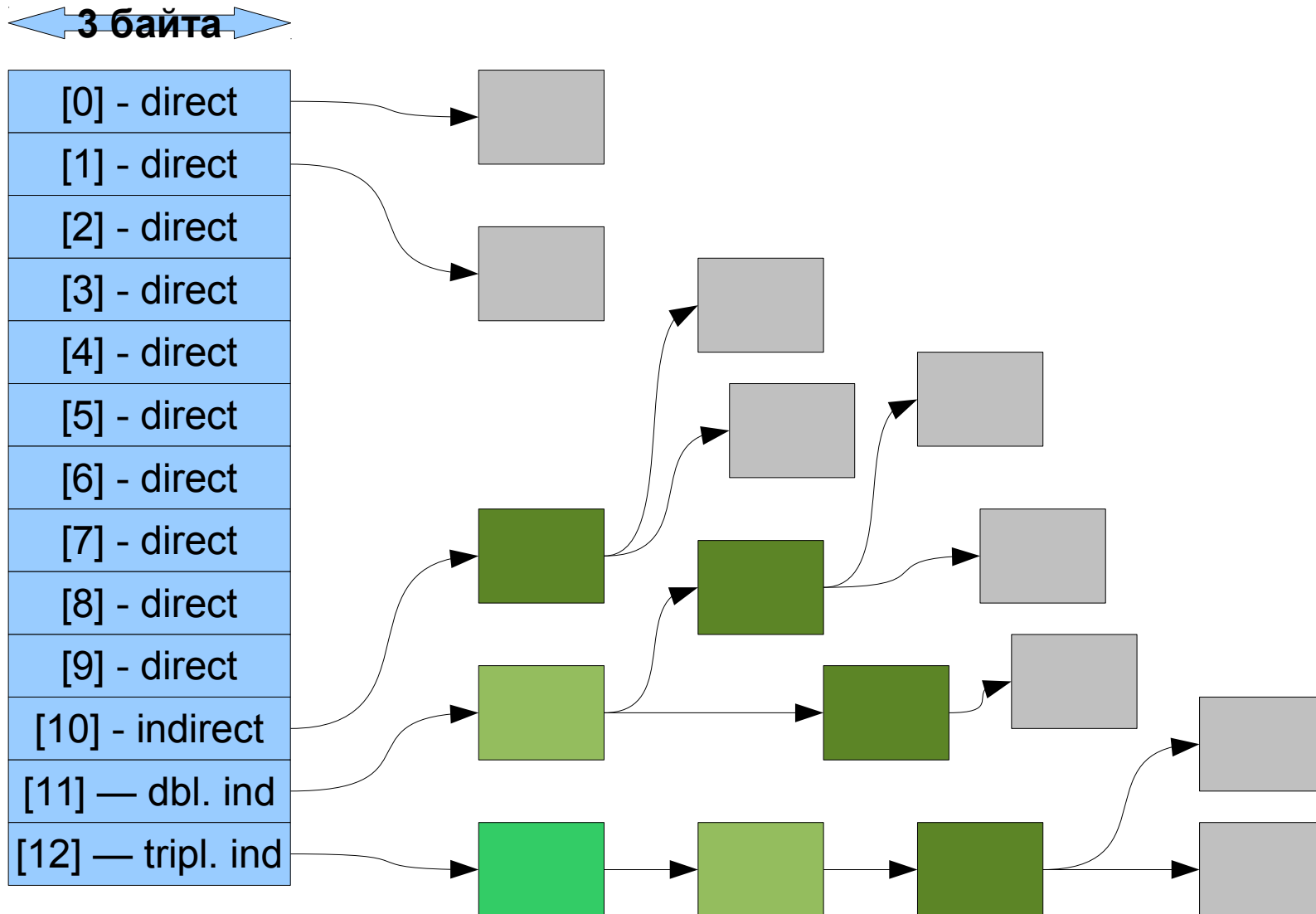
# Индексный дескриптор (inode)

Поле	Размер	Описание
di_mode	2	Права доступа и тип файла
di_nlinks	2	Число ссылок на этот и. д.
di_uid	2	Идентификатор пользователя
di_gid	2	Идентификатор группы
di_size	4	Размер файла
di_addr	39	Массив адресов блоков данных
di_gen	1	Поколение
di_atime	4	Время посл. доступа к файлу
di_mtime	4	Время модификации файла
di_ctime	4	Время создания файла

**Размер — 64 байта**

**Индексный дескриптор в памяти содержит дополнительные поля!**

# Массив адресов блоков



# Размеры файлов

- Размер блока — 512 байт
- Один номер блока – 3 байта
- 10 непоср. номеров блоков — 5 KiB
- Номер косв. Блока –  $512/3 = 170$  номеров блоков
  - Итого:  $10 + 170 = 180$  блоков
- Номер двойного косв. блока -  $170^2$  блоков
  - Итого:  $10 + 170 + 170^2 = \sim 14 \text{ MiB}$
- Номер тройного косв. блока -  $170^3$  блоков
  - Итого:  $10 + 170 + 170^2 + 170^3 = \sim 2.5 \text{ GiB}$
- Максимальный размер файла в системных вызовах – `INT_MAX` = 2GiB

# Структура каталога

- Каталог — файл, содержащий список файлов и каталогов
- Каждая запись в каталоге — 16 байт
  - Имя файла — 14 байтов
  - Номер индексного дескриптора — 2 байта

# Недостатки

- Суперблок может быть поврежден
- Размер блока недостаточный (низкая скорость передачи)
- Блоки файлов и каталогов разбросаны по диску
- Индексные дескрипторы находятся далеко от блоков данных

# Файловая система Ext2 (Linux)

Загрузочный Сектор	Группа блоков 1	Группа блоков 2	Группа блоков 3	Группа блоков N
--------------------	-----------------	-----------------	-----------------	-----------------

Группа блоков:

Суперблок	Дескриптор ФС	Карта своб. блоков	Карта своб. и.д.	Массив и.д.	Блоки данных
-----------	---------------	--------------------	------------------	-------------	--------------

- Размер блока данных: 1024, 2048, 4096 байт
- Номера индексных дескрипторов и блоков — 32 битные беззнаковые
- Размер индексного дескриптора — 128 байт
- Запись в каталоге имеет переменный размер (до 256 с)

# Ext2 (1993)

- В inode хранится 12 прямых ссылок на блоки, indirect, double indirect, triple indirect (каждая — 32 бита)
- Если длина symlink < 60 байт, то он хранится в inode
- Свободные блоки хранятся в битовом множестве

# Журналирование

- Обеспечение целостности файловой системы в случае краха ОС или сбоя питания
- Журнал — специальная область на диске
- Каждая операция, модифицирующая данные, выполняется в три стадии:
  - В журнал записывается операция (с флагом невыполненной)
  - Выполняется операция
  - Операция в журнале помечается как выполненная



# Ext3 (2001)

- Совместима снизу вверх с ext2
- Обеспечивает журналирование
- Индексирование больших каталогов (htree)
- Максимальный размер файла увеличен до  $2 \text{ TiB} = 2 * 1024 \text{ GiB}$

# Ext4 (2008)

- Макс. размер файловой системы до  $2^{60}$
- Макс. размер файла — 16TiB
- Extends вместо отображения блоков
  - До 4 на каждый inode, 128 MiB каждый
  - Оптимизация размещения на диске больших файлов
- Отметки времени с точностью до наносекунд
- 34 бита на секундную часть времени