Лекция 21
Файловая система

Модель файловой системы

- Файловая система размещается на блок-ориентированном устройстве
- Блок-ориентированные устройства:
  - Предоставляют произвольный доступ (seekable)
  - Обмен блоками фиксированного размера
  - Постоянное хранение (повторное чтение одного и того же блока дает один и тот же результат)
- Ядро кеширует блоки устройства в «буферном кеше»

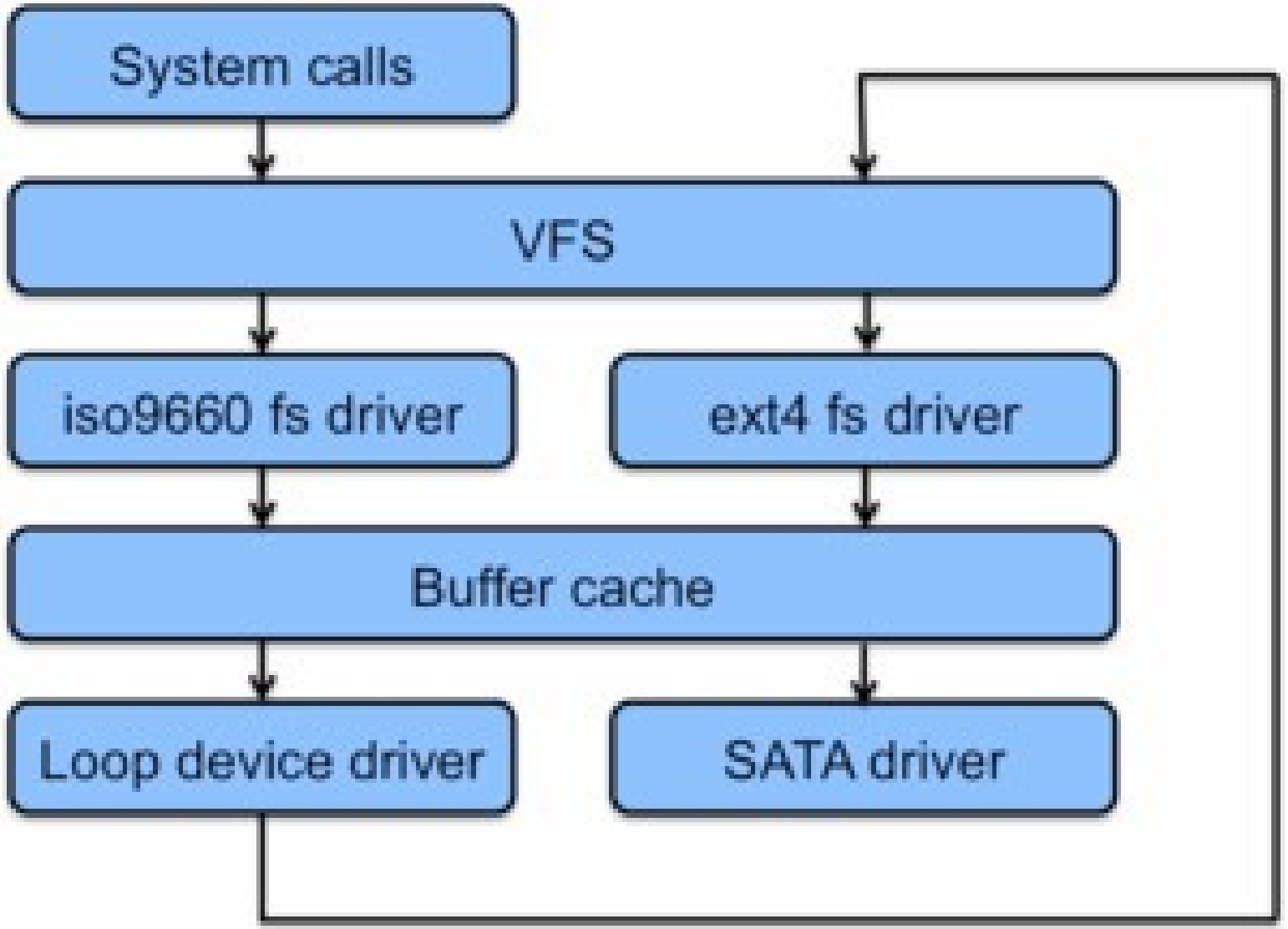
Номер устройства

- Все устройства (блок- и символ-ориентированные) идентифицируются **номером устройства** (st_dev)
- Номер устройства фиксирован в одном сеансе работы, но может меняться после перезагрузки
- Традиционно номер устройства делился на major (24 бита) и minor (8 бит)
  - Major — идентификация типа устройства (напр. SATA диск)
  - Minor — идентификация конкретного устройства

Номер устройства

- Одно блочное устройство — одна файловая система
- Каждая файловая система идентифицируется номером своего блочного устройства
- /dev/loop позволяет отобразить блочное устройство на файл в файловой системе

Блочные устройства



Модель файловой системы

- Все файлы (регулярные, каталоги, файлы устройств) идентифицируются номером индексного дескриптора (st_ino)
- Номер inode уникален для каждой файловой системы
- Индексный дескриптор хранит основную метаинформацию о файле
- Пара st_dev:st_ino уникально идентифицирует любой файл в системе

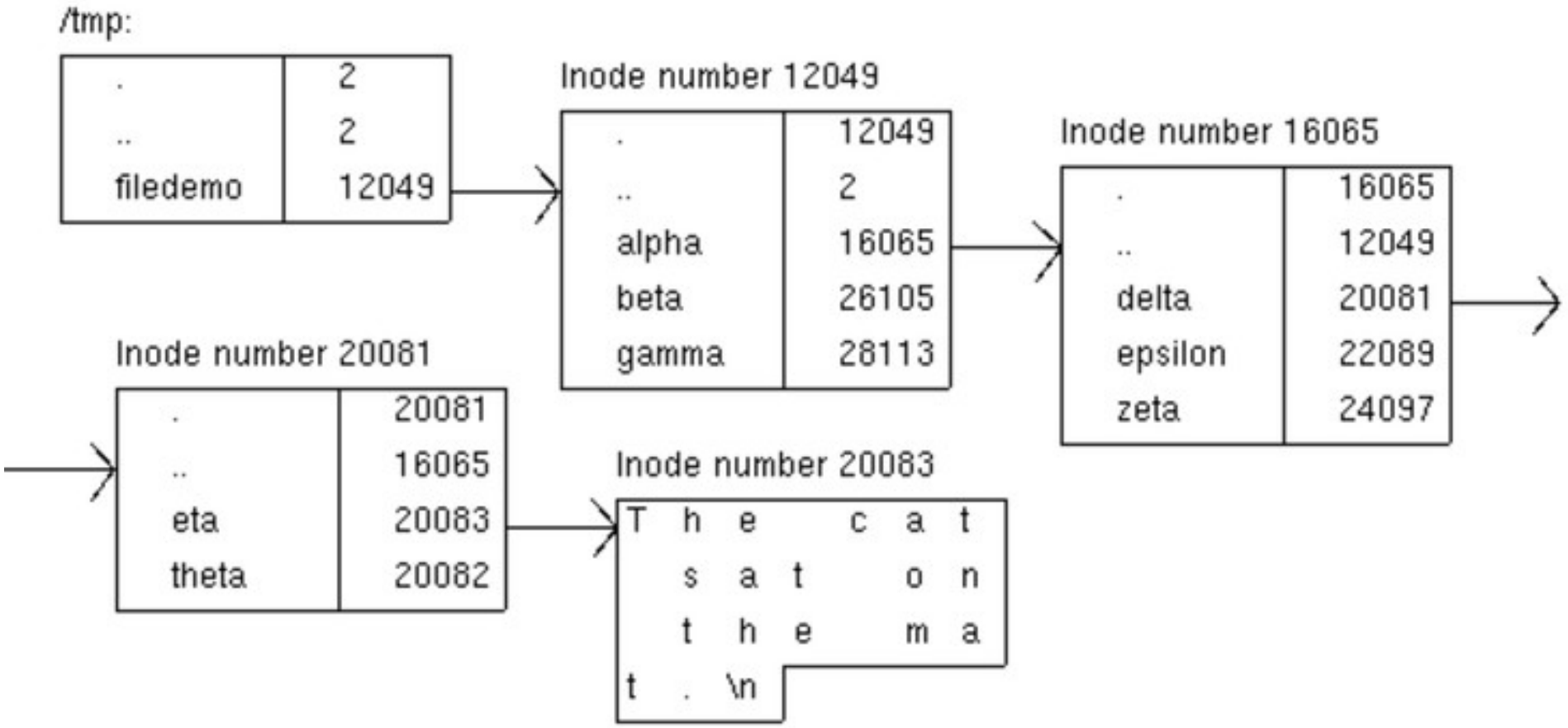
Модель файловой системы

- Массив индексных дескрипторов
  - Имеет фиксированный размер, не может быть изменен
  - Размер задается при создании файловой системы
- Блоки данных
  - Блоки регулярных файлов
  - Каталоги — тоже файлы, идентифицируемые по inode, имеют блоки данных
- Номер inode корневого каталога

Каталог

- Каталог — массив записей (dirent — directory entry)
- Каждая запись содержит:
  - Имя файла
  - Номер индексного дескриптора

Структура файловой системы



Struct stat — модель inode

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device ID (if special file) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```

st_mode

- Хранит и права доступа (12 бит), и тип файла
- Тип файла проверяется макросом:
  - S_ISREG(m) регулярный
  - S_ISDIR(m) каталог
  - S_ISCHR(m) символ-ориент. устройство
  - S_ISBLK(m) блок-ориентир. устройство
  - S_ISFIFO(m) именованный канал
  - S_ISLNK(m) символическая ссылка
  - S_ISSOCK(m) локальный сокет (PF_UNIX)

Спец. файлы

- Файлы устройств, именованный канал, сокет — только индексный дескриптор, нет блоков данных
- st_rdev — номер устройства

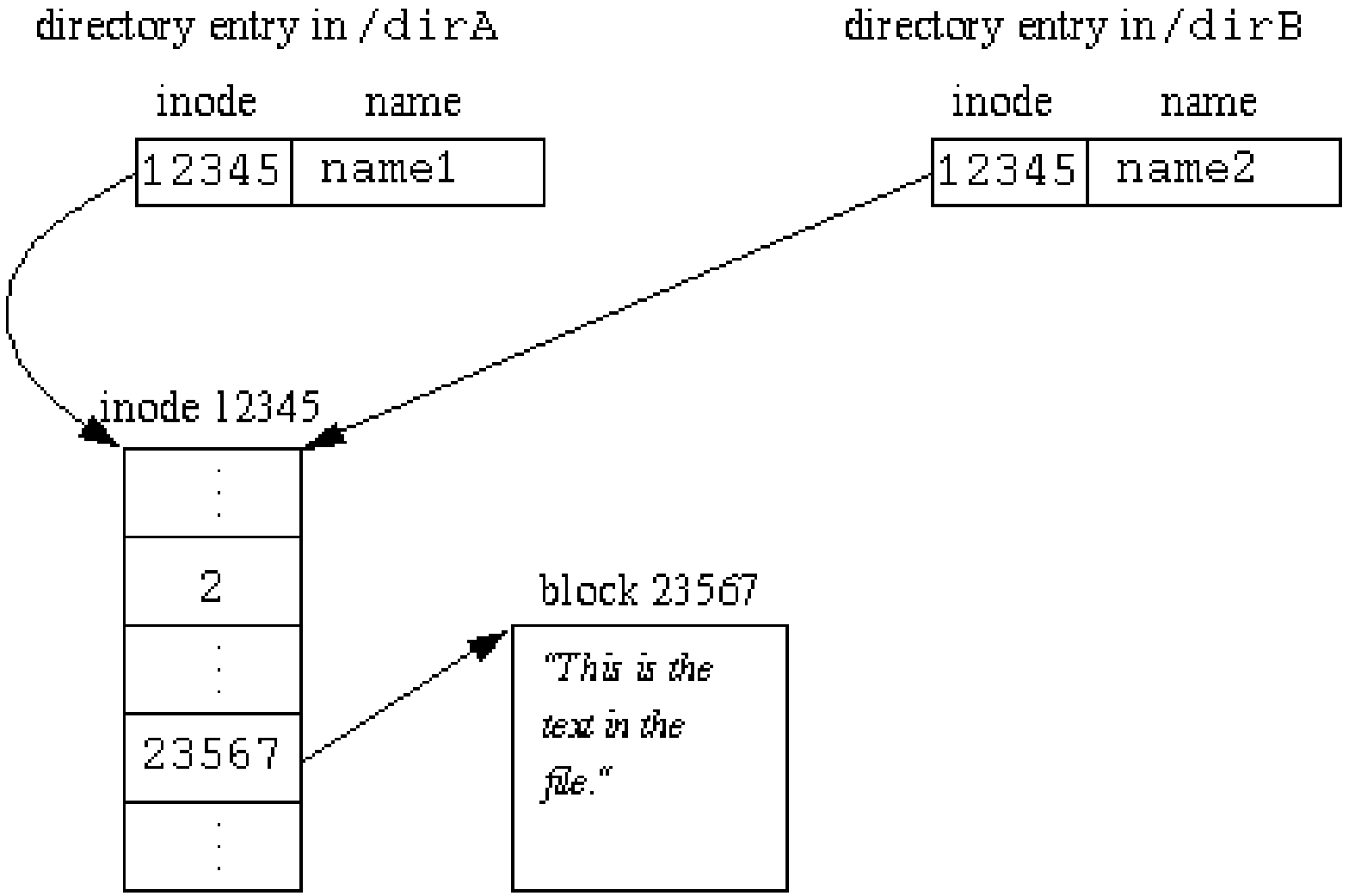
Символическая ссылка (symlink)

- Специальный файл, содержимое которого — путь в файловой системе (относительный, абсолютный, часть пути)
- При работе с файлами ядро прослеживает символические ссылки (глубина рекурсии ограничена)
- Символические ссылки могут указывать «в никуда»
- При удалении удаляется символическая ссылка, а не файл, на который она указывает
- Специальные системные вызовы позволяют работать с файлом-симв. связью, а не с файлом-целью
- Могут пересекать границы файловой системы

Жесткие ссылки (hardlinks)

- Несколько записей в каталогах содержат один и тот же номер inode
- Альтернативные имена файла
- Не могут создаваться для каталогов
- Не могут пересекать границы файловой системы

Жесткие ссылки



st_nlink

- Счетчик ссылок на индексный дескриптор из записей в каталогах
- Создание жесткой связи увеличивает счетчик ссылок
- Удаление — уменьшает счетчик ссылок
- Когда счетчик ссылок обнуляется, inode и блоки данных освобождаются
- Открытие файла (open) увеличивает счетчик ссылок
- Файл реально уничтожается когда закрывается последний файловый дескритор

Время в Unix

- Тип для хранения времени: time_t
- Тип знаковый, допускаются отрицательные значения
- Время хранится в UTC
- Число секунд, прошедших с 1970-01-01 00:00:00 UTC
- Leap seconds не учитываются (т.е. в сутках всегда 86400 секунд)

Недостатки

- Если sizeof(time_t) == 4, то проблема переполнения time_t (проблема 2038 года)
- Точность в 1с часто недостаточна
- Необходимость специальной обработки leap seconds

Работа со временем

```
#include <time.h>
time_t time(time_t *tptr);
// получить время в секундах
struct tm *localtime(time_t *tptr);
time_t mktime(struct tm *ptm);
```

Struct tm

```
struct tm {
    int tm_sec;         /* seconds */
    int tm_min;         /* minutes */
    int tm_hour;        /* hours */
    int tm_mday;        /* day of the month */
    int tm_mon;         /* month */
    int tm_year;        /* year */
    int tm_wday;        /* day of the week */
    int tm_yday;        /* day in the year */
    int tm_isdst;       /* daylight saving time */
};
```

mktime

- Работает в локальной временной зоне
- Для корректной обработки летнего времени при вызове mktime поле tm_isdst должно быть -1
- Возвращает -1, если время не представимо в time_t
- Нормализует значения полей tm_year, tm_mon, tm_mday, tm_hour, tm_min, tm_sec
- Заполняет tm_wday, tm_yday

Системные вызовы

```
#include <unistd.h>

int link(const char *oldpath, const char *newpath);
int symlink(const char *oldpath, const char *newpath);
int unlink(const char *pathname);
int rename(const char *oldpath, const char *newpath);

int mkdir(const char *pathname, mode_t mode);
int rmdir(const char *pathname);
```

Получение информации о файле

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

int stat(const char *path, struct stat *buf);
int fstat(int fd, struct stat *buf);
int lstat(const char *path, struct stat *buf);
```

Просмотр каталогов

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

DIR *opendir(const char *name);
int closedir(DIR *dirp);
struct dirent *readdir(DIR *dirp);
```

Просмотр каталога

```
vector<string> names;
DIR *d = opendir(dir);
struct dirent *dd;
while ((dd = readdir(d))) {
        string dn(dd->d_name);
        if (dn == "." || dn == "..") continue;
        string path = string(argv[1]) + "/" + dn;
        struct stat st;
        if (lstat(path.c_str(), &st) < 0)
                continue;
        if (S_ISREG(st.st_mode))
                names.emplace_back(dn);
}
closedir(d);
```

Сложности

- Файловая система может изменяться одновременно несколькими процессами
- Недопустимо блокировать каталоги файловой системы на длительное время
- Формат хранения данных в каталоге оптимизирован для большого количества файлов (ext4 – B-tree)

Telldir/seekdir

- Позволяют получить текущую “позицию” в каталоге и вернуться к ранее полученной “позиции” в каталоге
- “Позиция” - на самом деле какой-то хеш от имени
- В стандарте POSIX есть оговорка: только для “the same directory stream”, то есть закрывать каталог в промежутке между telldir и seekdir нельзя
- В современном Linux этой оговорки нет в man, но есть в документации на glibc
- Следует придерживаться консервативного правила: **seekdir после closedir и opendir не работает**

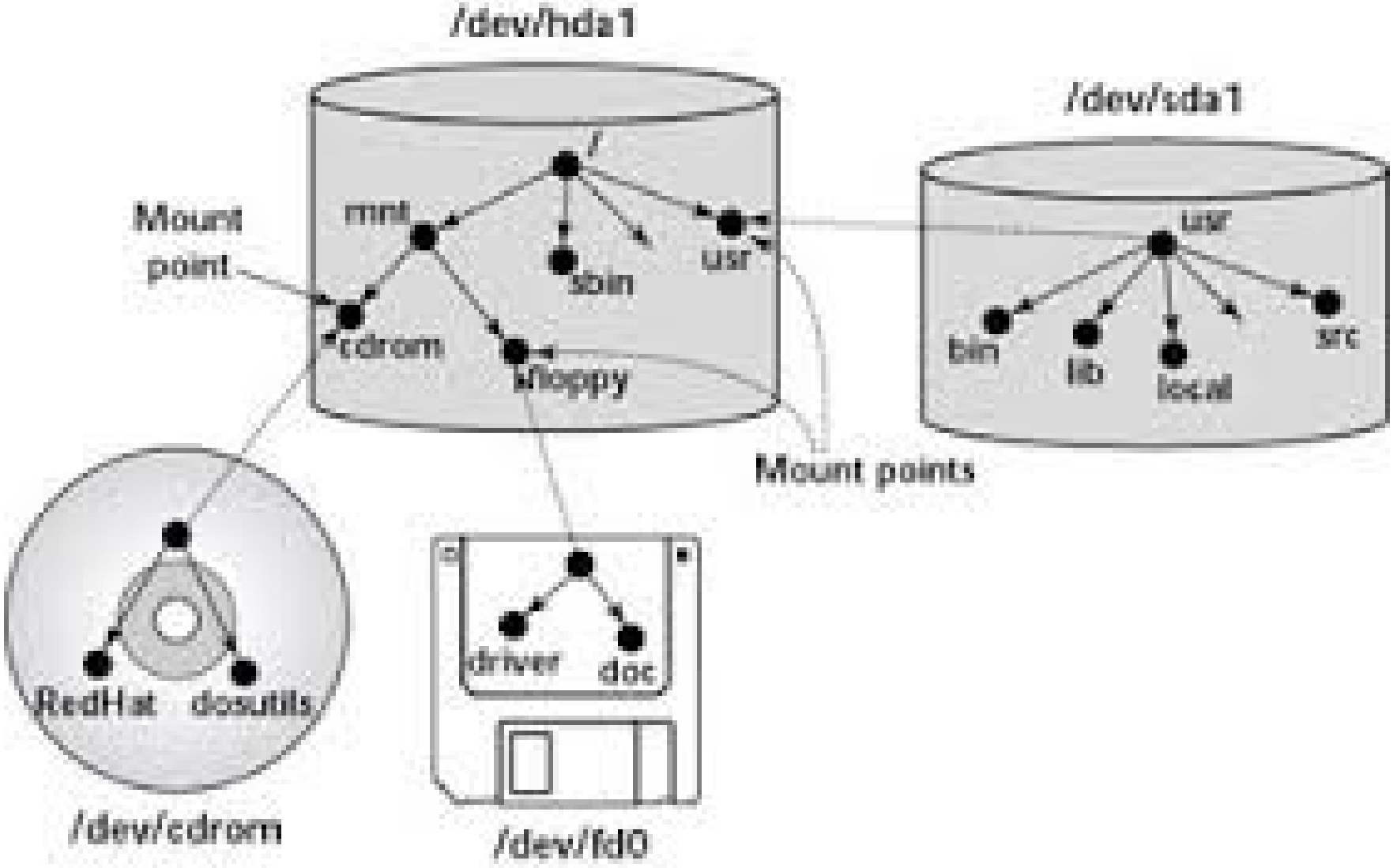
Требования к readdir

- Если файл существовал и не был удален за время сканирования каталога, информация о нем должна быть получена readdir
- Информация о любом файле должна быть получена не более одного раза

readdir

- Readdir возвращает указатель на структуру с информацией о текущей записи
- Readdir может использовать одну и ту же область память каждый раз
- Нельзя предполагать, что вызов readdir не испортит память по указателю, полученному предыдущим вызовом readdir
- Каталоги могут меняться одновременно с чтением, при использовании readdir + (l)stat нельзя предполагать, что файл еще будет существовать к моменту (l)stat

Монтирование



Пример монтирования

$ mount

```
/dev/md0 on / type ext4 (rw,relatime)
/dev/md1 on /home type ext4 (rw,noatime,nodiratime)
```

Номера st_dev:st_ino

```
2304:2 /

2305:2 /home


2305:2 /home/.

2304:2 /home/..
```

Файловые системы /proc и /sys

- Ядро предоставляет доступ к информации о состоянии системы с помощью этих псевдофайловых систем
- Структуры данных этих файловых систем существуют только в памяти ядра
- /proc более «историческая»
- /sys более «структурированная»

/proc

- Информация о процессах /proc/${PID}:
  - fd — открытые файловые дескрипторы
  - exe — путь к исполняемому файлу
  - cwd — текущий рабочий каталог
  - maps — карта памяти
- Информация о системе
  - interrupts — прерывания
  - modules — загруженные модули ядра

Модификация параметров

- /proc/sys/kernel/core_pattern — шаблон для имени core-файла

```
cat /proc/sys/kernel/core_pattern
```

```
echo 'mycore' > /proc/sys/kernel/core_pattern
```

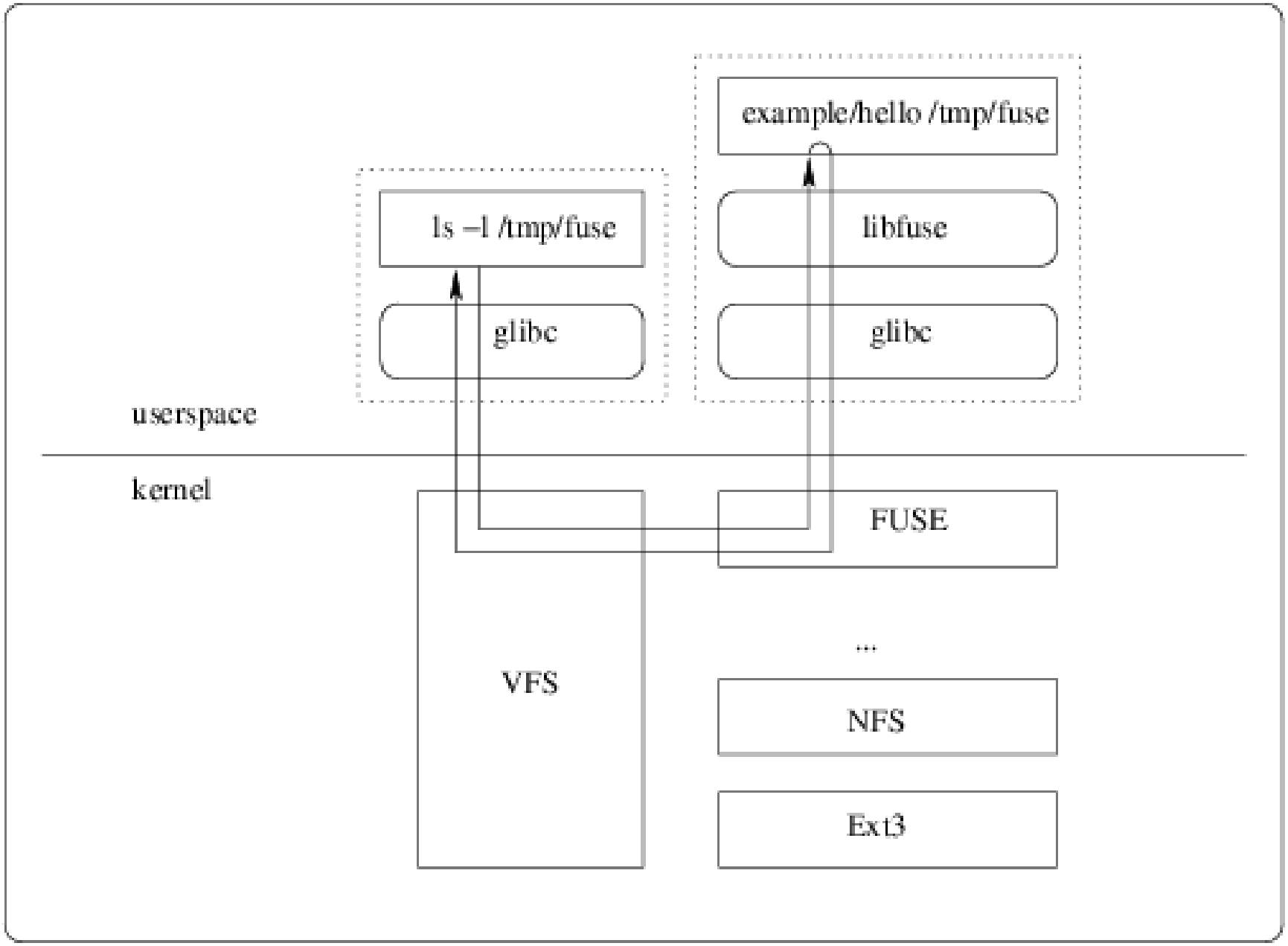
FUSE (Filesystem in userspace)

- Позволяет реализовать драйвер файловой системы в пользовательском процессе
  - Надежность: ошибка в драйвере не приводит к краху ядра
  - Гибкость: можно менять без перезагрузки, права обычного пользователя
  - Безопасность: смонтировавший пользователь может ограничить доступ
- Применения: fuse-sshfs, ntfs

Архитектура FUSE

- Файловая система fuse — модуль ядра Linux
  - Запросы к VFS собираются в пакеты данных и пересылаются пользовательскому процессу через /dev/fuse
  - Ответы отправляются обратно в VFS
- Библиотека libfuse для драйверов ФС
- Привилегированная утилита монтирования fusermount

FUSE


example/hello /tmp/fuse
ls –l /tmp/fuse
libfuse
glibc
glibc
userspace
kernel
FUSE
VFS
...
NFS
Ext3