

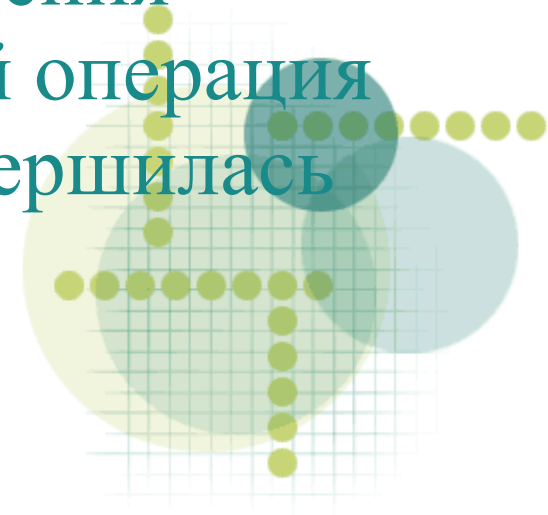
Работа с файлами в многопроцессной системе

- Файлы и файловая система — ресурс, общий для всех процессов в системе
- Несколько процессов могут выполнять операции над копиями одного и того же файлового дескриптора или над одним и тем же файлом



Атомарность

- Атомарность (неделимость) операции — свойство операции по отношению к наблюдателям (процессам).
- Операция атомарна, если никакой наблюдатель не может наблюдать промежуточные результаты выполнения операции, то есть для наблюдателей операция либо еще не началась, либо уже завершилась



Чтение/запись

- Если размер считываемых/записываемых данных не превосходит PIPE_BUF, операция чтения/записи атомарна:
- Размер PIPE_BUF системно-зависим (Linux - 4096)

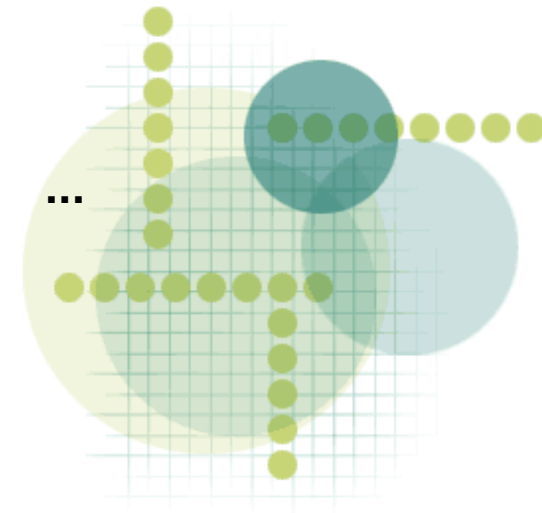
P1:
`write(fd, "str1\n", 5);`
`write(fd, "str2\n", 5);`

P2:
`write(fd, "str3\n", 5);`
`write(fd, "str4\n", 5);`

Возможные результаты операции: (всего 6)

`str1`
`str2`
`str3`
`str4`

`str1`
`str3`
`str2`
`str4`



Создание файла

- Для атомарного создания файла используется флаг `O_EXCL`

```
open("f", O_WRONLY|O_CREAT|O_EXCL,0666);
```

- Только один процесс сможет создать файл, остальные процессы получают ошибку `EEXIST`
- Файл должен быть удален после использования
- Не работает (не гарантируется атомарность) на сетевых файловых системах



Виды блокировок

- Блокировка на чтение (read lock)
 - Произвольное количество процессов могут одновременно выполнять чтение
 - Попытка заблокировать файл на запись приводит к приостановке работы процесса до снятия блокировки
- Блокировка на запись (write lock)
 - Только один процесс может выполнять запись
 - Попытка заблокировать файл на запись или на чтение приводит к приостановке работы процесса до снятия блокировки



Блокировки файлов

- Добровольные (advisory)
 - Процессы добровольно запрашивают требуемую блокировку перед выполнением операции
 - Процесс может игнорировать блокировки и просто выполнить операцию
- Принудительные (mandatory)
 - Ядро ОС проверяет совместимость запрошенной операции чтения/записи с установленными блокировками



Системный вызов `fcntl` (добровольная блокировка)

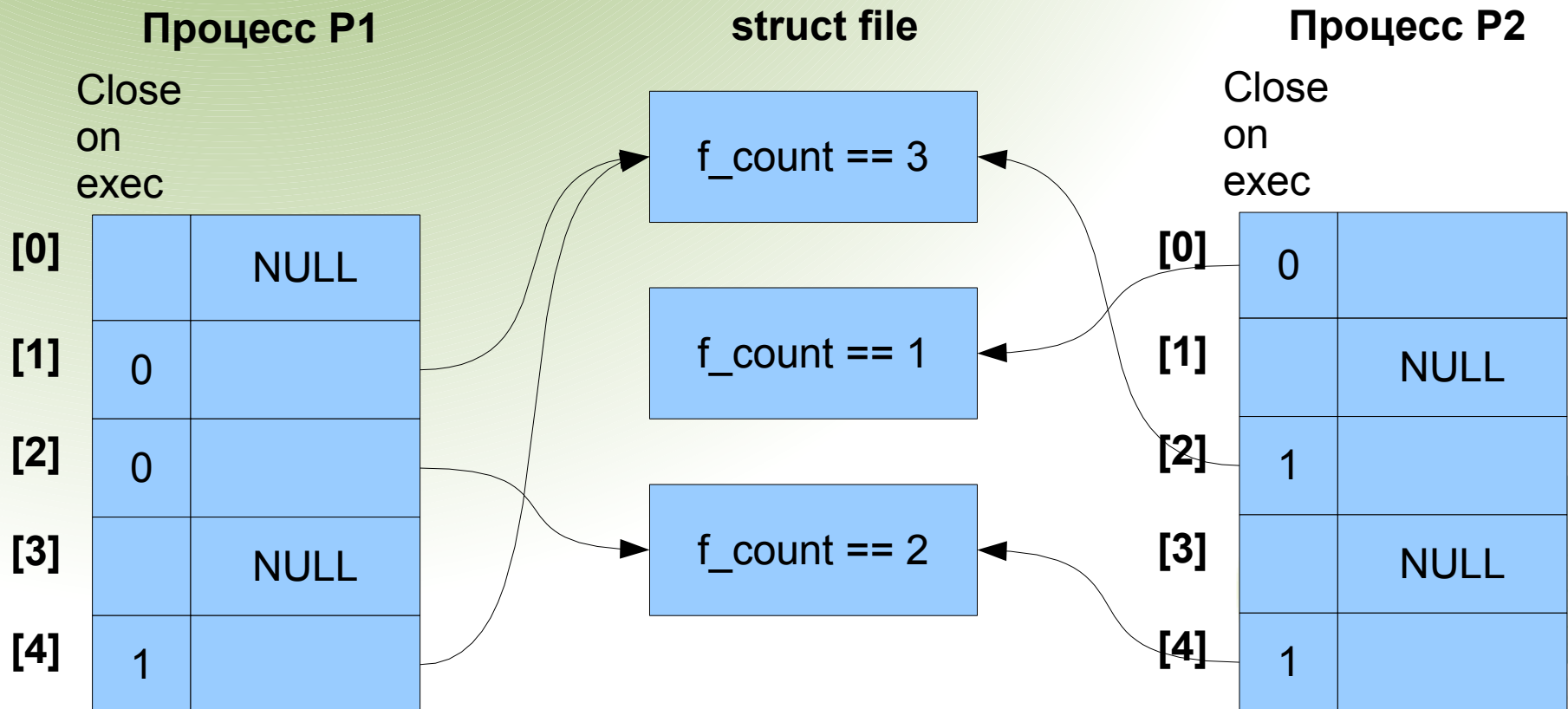
```
int fcntl(int fd, int cmd, struct flock *lock);
struct flock {
    ...
    short l_type; /* F_RDLCK, F_WRLCK, F_UNLCK */
    short l_whence; /* SEEK_SET, SEEK_CUR, SEEK_END */
    off_t l_start; /* Starting offset for lock */
    off_t l_len; /* Number of bytes to lock */
    pid_t l_pid; /* PID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

- Команды: `F_SETLK` — попробовать взять блокировку
- `F_SETLKW` — взять блокировку (возможно с приостановкой процесса)
- `F_GETLK` — получить информацию о текущей блокировке



Реализация работы с файлами в ядре ОС

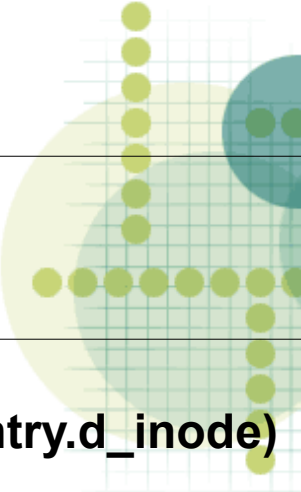
- Массив файловых дескрипторов:



`dup`, `dup2`, `fork` — создает копию файлового дескриптора, увеличивая счетчик `f_count` для `struct file`

struct file

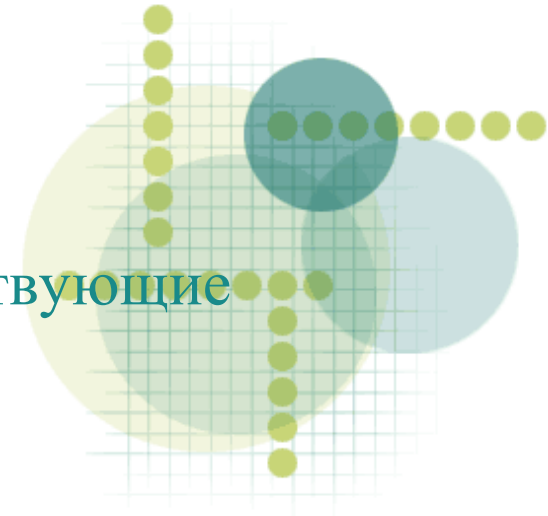
f_count	Счетчик ссылок из массива файловых дескрипторов на структуру
f_op	Массив операций над данным файлом (массив указателей на функции)
f_flags	Флаги открытия файла
f_mode	Режимы открытия файла
f_pos	Текущая позиция в файле
f_path	Ссылка на inode: (f_path.dentry.d_inode)



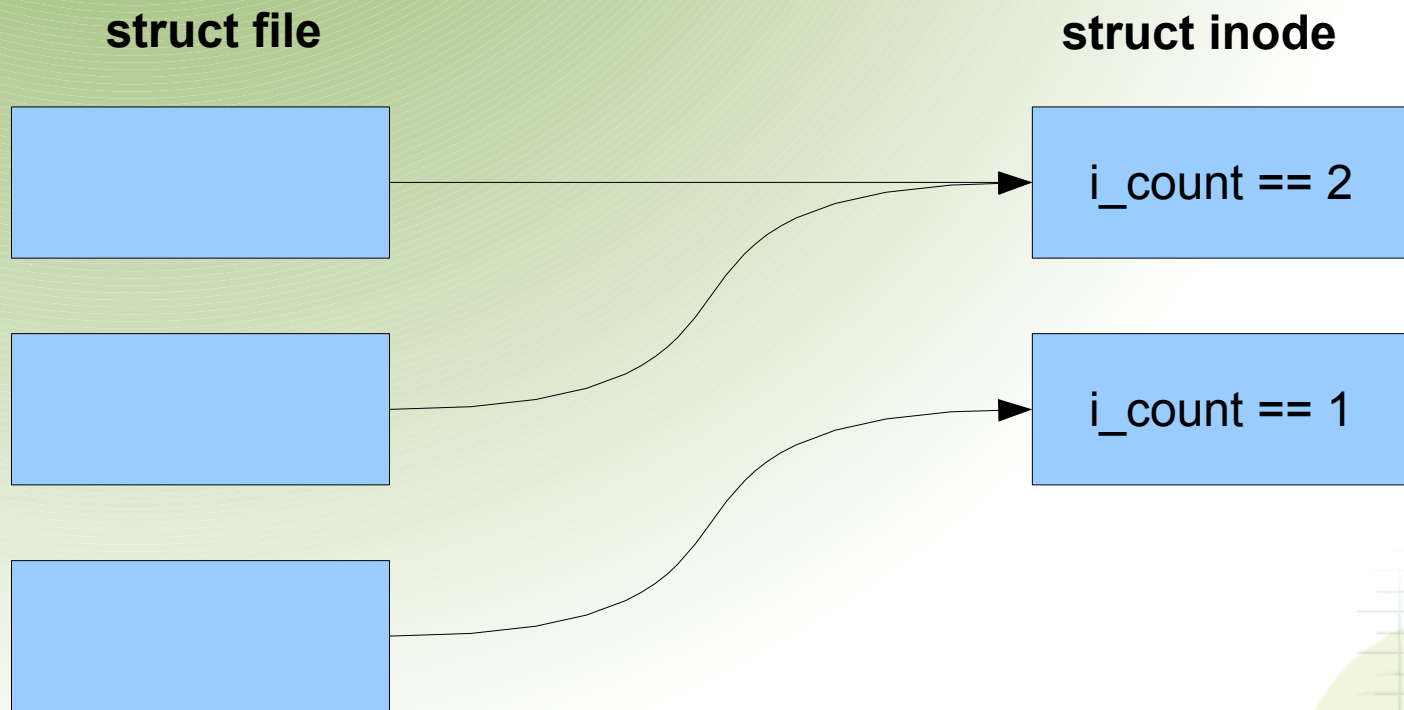
file_operations

```
struct file_operations {  
    // ...  
    loff_t (*llseek) (struct file *, loff_t, int);  
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);  
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);  
    int (*readdir) (struct file *, void *, filldir_t);  
    unsigned int (*poll) (struct file *, struct poll_table_struct *);  
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned  
long);  
    // ...  
};
```

- Указатели на функции, выполняющие соответствующие операции над открытым файлом



Отображение файлов в и.д.



Операция открытия файла `open` создает новый объект `struct file` и увеличивает `i_count` у соответствующего объекта `struct inode`

Индексный дескриптор в памяти (struct inode)

- Содержит все поля индексного дескриптора на диске, кроме того дополнительно:
- `i_ino` — номер индексного дескриптора
- `i_dev` — номер устройства
- `i_count` — счетчик ссылок из `struct file`
- `i_op` — операции над индексным дескриптором
- Блокировки
- Отображения в память



Уничтожение ресурсов

- Ресурс уничтожается, когда счетчик ссылок становится равным 0

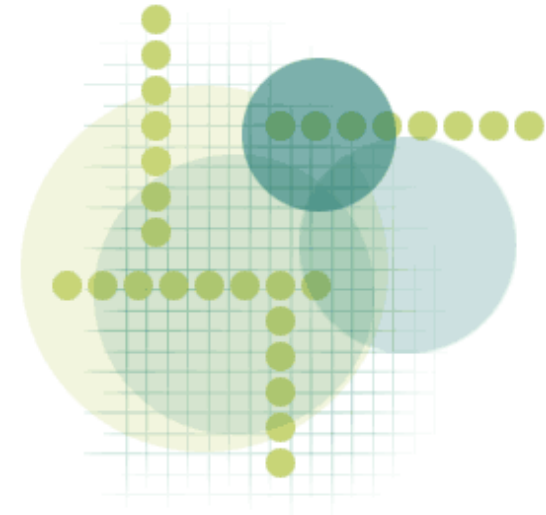
```
fd = open("file", O_RDWR|O_CREAT,0666);  
unlink("file");
```

- Открытый файл существует, пока не будет закрыта последняя копия fd, только после этого файл на диске будет уничтожен



Работа с ФС (системные вызовы)

- Получение информации о файле: `stat`, `fstat`, `lstat`
- Удаление файла: `unlink`, `rmdir`
- Создание записей: `mkdir`, `link`, `symlink`
- Сканирование каталога: `opendir`, `readdir`, `closedir`

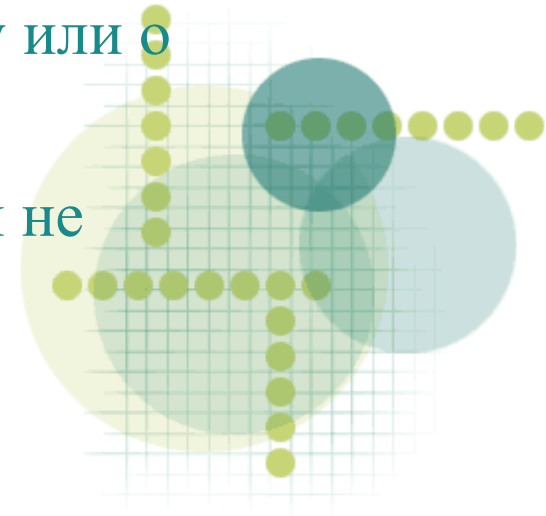


Получение информации о файле

```
#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);
```

- Функции заполняют структуру `buf` информацией о файле по пути или по открытому файловому дескриптору или о символической ссылке
- 0 — при успешном завершении, -1 — если файл не существует



struct stat

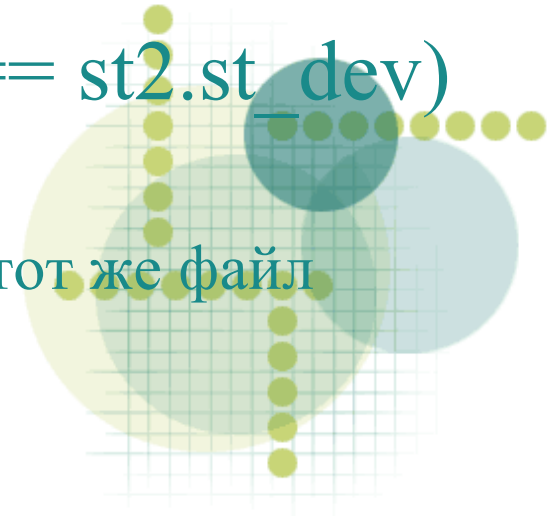
```
struct stat {  
    dev_t    st_dev;    /* ID of device containing file */  
    ino_t    st_ino;    /* inode number */  
    mode_t   st_mode;   /* protection */  
    nlink_t  st_nlink;  /* number of hard links */  
    uid_t    st_uid;    /* user ID of owner */  
    gid_t    st_gid;    /* group 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 filesystem I/O */  
    blkcnt_t st_blocks; /* number of 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 */  
};
```



Проверка идентичности файла

- Пара `st_dev:st_ino` уникально идентифицирует каждый файл в системе, а путь к файлу не является уникальным идентификатором.

```
struct stat st1, st2;  
st1 = stat(path1, &st1);  
st2 = stat(path2, &st2);  
if (st1.st_ino == st2.st_ino && st1.st_dev == st2.st_dev)  
{  
    // файл, задаваемые путями path1 и path2 — один и тот же файл  
}
```



Проверка типа файла

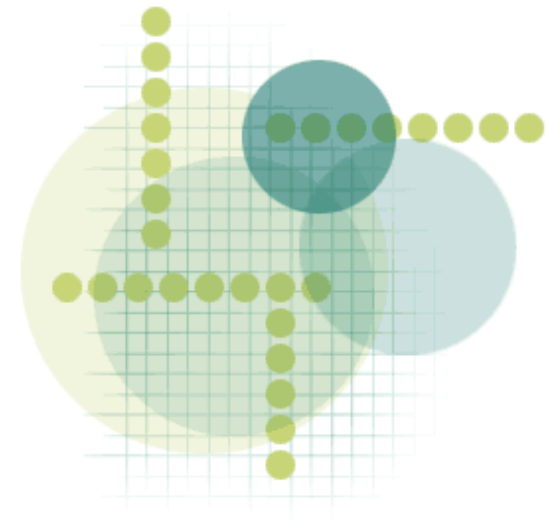
```
struct stat stb;  
stat(path, &stb);  
if (S_ISREG(stb.st_mode)) {  
    // регулярный файл  
} else if (S_ISDIR(stb.st_mode)) {  
    // каталог  
} ...
```

- S_ISCHR, S_ISBLK, S_ISFIFO, S_ISLNK, S_ISSOCK



Просмотр каталогов

- `opendir` открывает указанный каталог для просмотра файлов
- `readdir` считывает очередную запись, поле `d_name` содержит компоненту имени файла в каталоге



Пример

```
#include <dirent.h>
#include <stdio.h>
#include <limits.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
int main(int argc, char *argv[])
{
    struct dirent *dd; char buf[PATH_MAX]; struct stat st;
    DIR *d = opendir(argv[1]);
    while ((dd = readdir(d))) {
        if (!strcmp(dd->d_name, ".")
            || !strcmp(dd->d_name, "..")) continue;
        snprintf(buf, sizeof(buf), "%s/%s", argv[1],
                 dd->d_name);
        if (stat(buf, &st) < 0) continue;
        printf("%s: %ld\n", buf, st.st_size);
    }
    closedir(d); return 0;
}
```



Работа с ФС

- `int unlink(const char *path);`
- `int link(const char *oldp, const char *newp);`
- `int symlink(const char *oldp, const char *newp);`
- `int rename(const char *oldp, const char *newp);`
- `int mkdir(const char *path, mode_t mode);`
- `int rmdir(const char *path);`



Монтирование ФС

- Монтирование — подключение ФС к уже смонтированному ФС в ядре.
- Существующий каталог, к которому подключается ФС — точка монтирования.
- Старое содержимое этого каталога становится недоступным.

```
mount /dev/sda1 /var -t ext3
```



Специальные файлы устройств

- Специальные файлы устройств соответствуют внешним устройствам:
 - `/dev/sda` — диск 1 (обычно нжмд)
 - `/dev/sda1` — первый раздел диска 1
 - `/dev/ttyS0` — последовательный порт
 - `/dev/null` — пустое устройство
 - `/dev/zero` — чтение с устройства возвращает нулевые байты
 - `/dev/cdrom` — устройство CD-ROM



Типы устройств

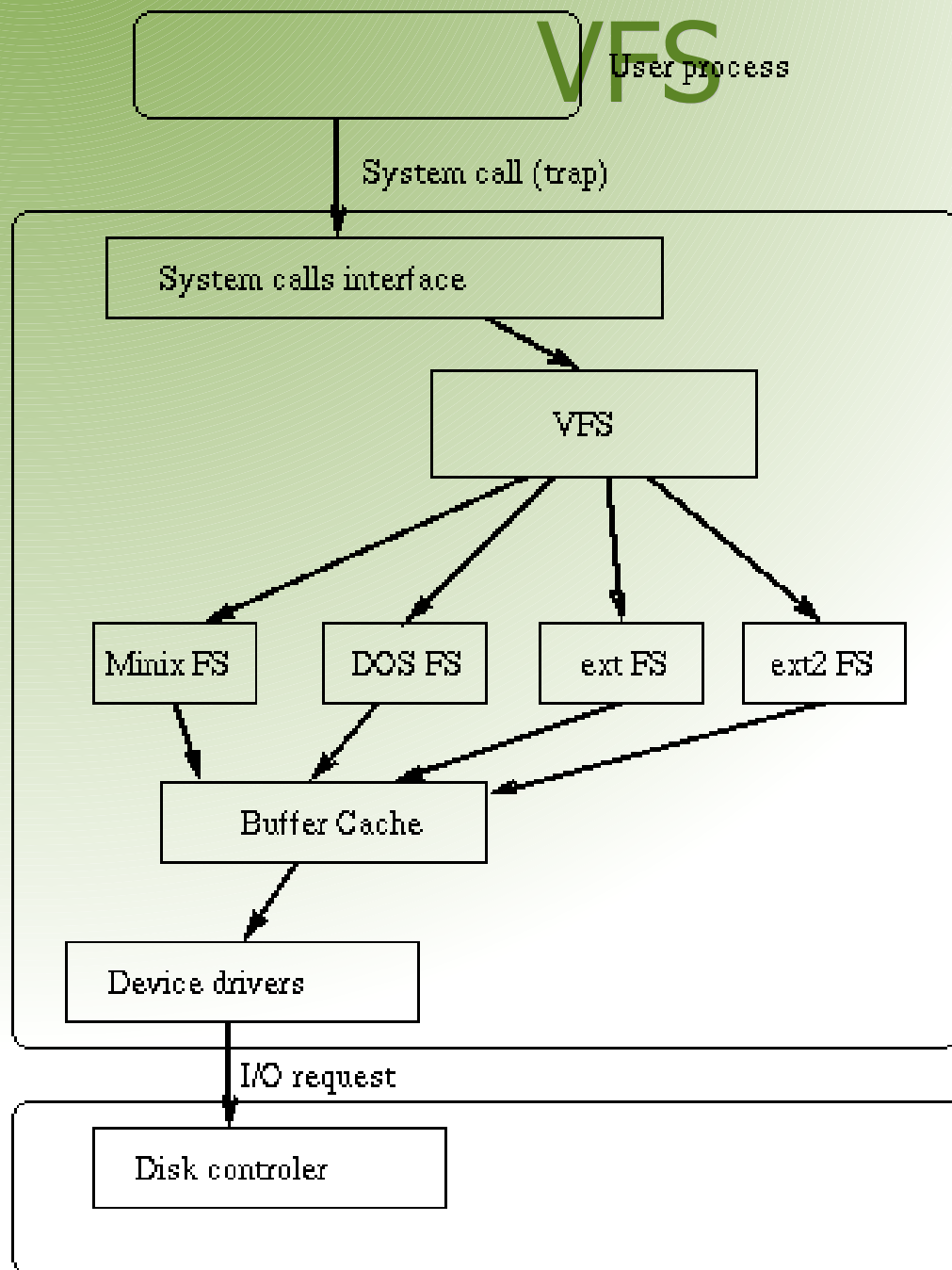
- Блочное устройство — block device
 - Допускает позиционирование на устройстве
 - Блоки кешируются ядром ОС (буферный кеш)
 - На блочных устройствах возможно размещение ФС
- Символьное (последовательное) устройство — character device



Файлы устройств

- Информация о специальном файле устройств хранится только в индексном дескрипторе и записях каталога — отсутствуют блоки данных.
- Идентификация устройств (традиционная)
 - Тип файла устройства (b/c)
 - Номер устройства (major number)
 - Номер подустройства (minor number)





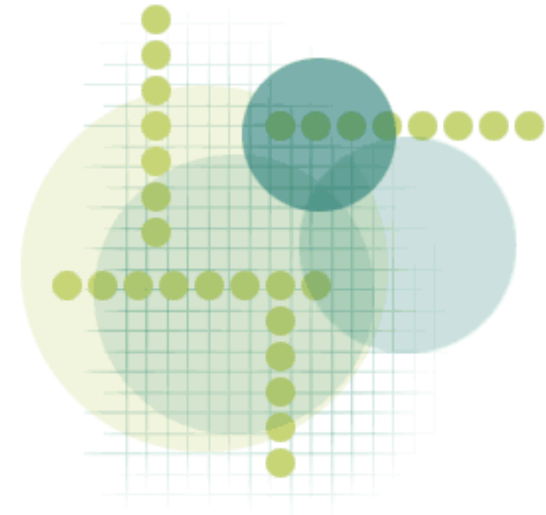
Linux Kernel

Hardware



VFS (virtual file system)

- Общий интерфейс для всех конкретных типов файловых систем.
- Впервые реализован в SunOS 2.0 (1985)
- Присутствует во всех современных ОС



Интерфейс суперблока

```
struct super_operations {  
    void (*read_inode) (struct inode *);  
    int (*notify_change) (struct inode *, struct iattr *);  
    void (*write_inode) (struct inode *);  
    void (*put_inode) (struct inode *);  
    void (*put_super) (struct super_block *);  
    void (*write_super) (struct super_block *);  
    void (*statfs) (struct super_block *, struct statfs *, int);  
};
```

