

ОС. Семинар №12 Буферизованный и небуферизованный ввод/вывод

Системный ввод/вывод `open()` (`read, write, lseek`) - оп-чим не буфер-но I/O, а стандартная библиотека `<stdio.h>` - библиотека буфер-но I/O.

Назначение I/P- открытие файлов, т.к. только процесс может открыть файл (-61).

Системный ввод/вывод `open()`

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

```
int open (const char *filename, int flags);
```

- имя файла в файловой системе (полный путь (если не в шек.) либо сокращённое имя)
- права доступа, создание, создание (побитов. или)
- при создании нового файла - опр. права доступа к файлу (шер. гр. others)

Режимы O-CREATE - создание нового файла, если он не существует

O-EXCL - используется с O-CREATE - если файл существует, то происходит ошибка с кодом `errno = EEXIST`

O-APPEND - запись нового данных будет добавлена в конец файла (ук. номинального адреса будет указ. на конец файла).

O-TRUNC - если файл существует, удаляется его содержимое, оставив пустой ф.

O-RDONLY - только чтение

O-RDWR - чтение и запись

O-WRONLY - только запись

O_NONBLOCK / O_NDELAY - открытие файла в невложир. режиме, если возможно.

и гр.

Режимы S-* Например: S_IRWXU, S_IFDIR, S_ISUID и гр.

В основе этой работы - структуры =>

=> Файл может открыть только процесс, в дескрипторе процесса есть поле - указанием на файловую систему и открытые процессом файлы.

```
struct task_struct {
```

/* Filesystem information */

struct fs_struct *fs;

/* Open file information */

struct files_struct *files;

```
};
```

указатель на файл. систему

содержит инфу о файл. сист., к которой принадлежит процесс

и массив указателей на файл. объекты

дескрипторы файлов, открытых процессом

файл использ. прогр.

Каждый процесс имеет собственную таблицу открытых файлов

```
struct fs_struct {
    atomic_t count; - счётчик ссылок на структуру
    rwlock_t lock; - блокировка для защиты структуры
    int umask; - права доступа к файлу, исп. по умолчанию
    struct dentry *root; - дентри корневого каталога
    struct dentry *pwd; - дентри текущего каталога
    struct dentry *allroot; - дентри абсолютного корня
    struct vfsmount *rootmnt; - объект монтирования, корн. каталога
    struct vfsmount *pwdmnt; - " " " текущ. каталога
    struct vfsmount *altrootmnt; - " " " абсолют. корня
};
```

struct dentry { - структура, описывающая элемент каталога
 atomic_t d_count; - счётчик ссылок на структуру
 struct inode *d_inode; - соотв-щий файловой структуре
 ... };

atomic_t count → int users (в современных ОС) - количество процессов, которые работают с имен. структурой - объектом.

```
struct files_struct {
    atomic_t count; - счётчик ссылок на структуру
    spinlock_t file_lock; - блокировка для защиты
    int max_fds; - макс. кол-во открытых объектов
    int max_fdset; - макс. кол-во открытых дескрипторов
    struct file **fd; - массив всех открытых объектов
    fd_set *close_on_exec; - ф.дескр., которые должны закрываться при вызове exec()
    *open_fds; - ф.дескр. открыт. файлов
    close_on_exec_init;
    open_fds_init; - первоначальные наборы ф.дескрипторов
    struct file *fd_array[NR_OPEN_DEFAULT]; - массив открытых объектов
};
```



Когда создаётся процесс, для него автоматически открывается 3 ф.дескриптора - stdin, stdout, stderr (0, 1, 2) => => сreg. свобод. дескриптор - 3.

макс. кол-во

определение дескрипторов открытого файла в системе, представляющие файлы, которые открыты процессом.

struct file {

```
    struct path f_path;
    struct inode *f_inode;
    const struct file_operations *f_ops; // указывающие открытых операций
    ...
    // файлы, регистрация, метаданные;
```

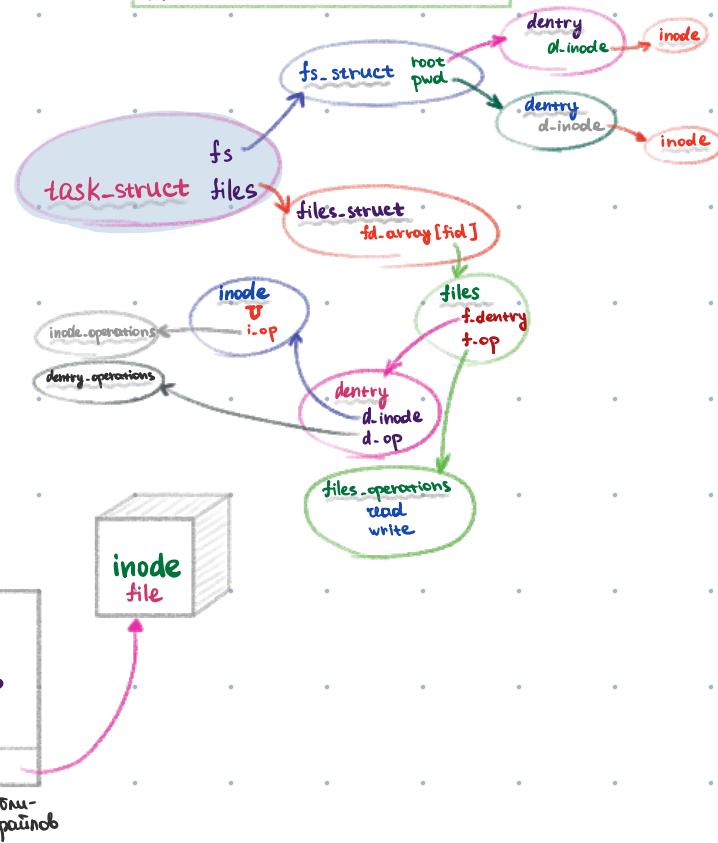
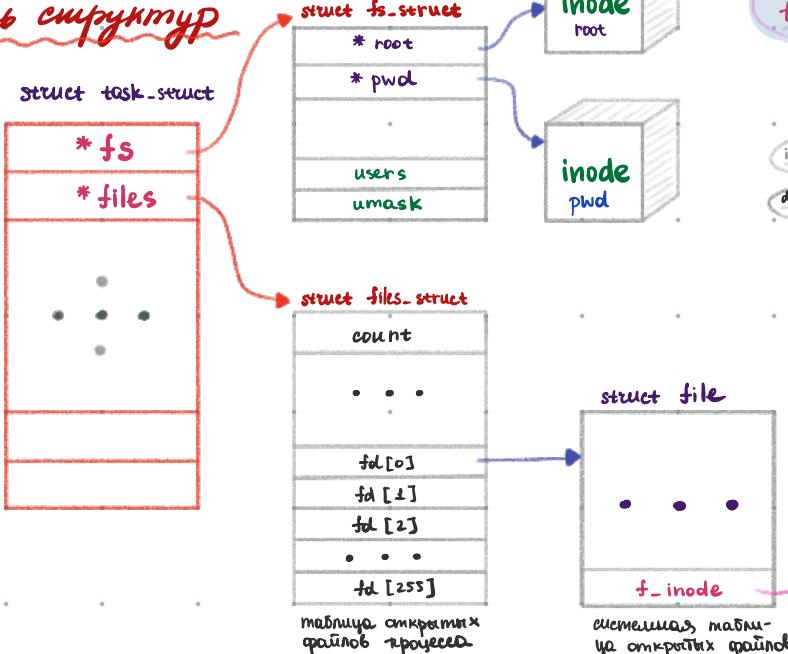
struct inode {

регистрации, открытия, манипуляции, ...
адреса и т.д.

};

содержит всю информацию, которая необходима для манипуляций с файлами и каталогами, описывает содержимое файла. Для каждого файла в системе существует представление его иноде (хотя объект inode является множеством, как и в случае с файлом, оно не доступно)

Связь структур



Изменение файлов в системе неёт
EOL - символ конца строки - '\0'

! { file - работает с логической структ. файла
 inode - работает с физ. представлением файла }