Э.с. — отправная точка для всех действий, которые выполняются в системе;

Связь структур VFS.
(на основе полей структур)
Это ключ к работе системы;

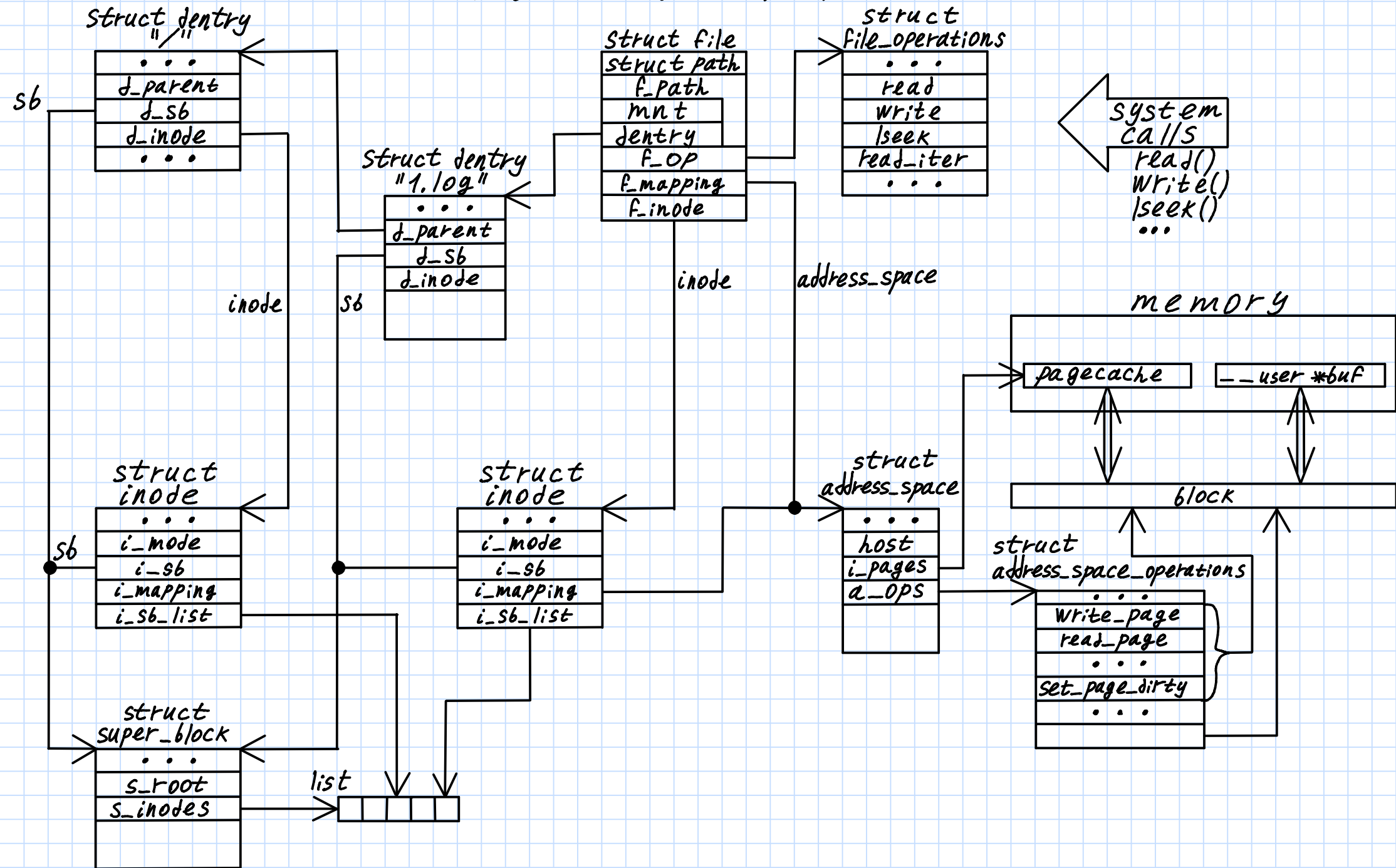Одна из отправных точек явл. сист. вызовы, связанные с файлами: read/write/lseek;

Эти сист. вызовы работают только с открытыми файлами;

Чтобы работать с файлом, его нужно открыть

Связи структур при выполнении системных вызовов
(по сути посл-ть действий, которые выполняются в ф.с.)

struct dentry "/"
. . .
d_parent
d_sb
d_inode
. . .

Struct dentry "1.log"
. . .
d_parent
d_sb
d_inode

Struct file
struct path
f_path
mnt
dentry
f_op
f_mapping
f_inode

struct file_operations
. . .
read
write
lseek
read_iter
. . .

system
calls
read()
write()
lseek()
. . .

sb

inode

sb

inode

address_space

struct inode
. . .
i_mode
i_sb
i_mapping
i_sb_list

struct inode
. . .
i_mode
i_sb
i_mapping
i_sb_list

struct address_space
. . .
host
i_pages
a_ops

sb

memory

pagecache	__user *buf

block

struct address_space_operations
. . .
write_page
read_page
. . .
set_page_dirty
. . .

struct super_block
. . .
s_root
s_inodes

list

Связь между struct file и struct file_operations:

Файл должен быть открыт. Соответственно для открытого файла должен быть создан дескриптор. В этом дескрипторе имеется указатель на struct file_operations. Это либо стандартные (установленные по умолчанию) операции на файлах для конкретной ф. с., либо зарегистрированные разработчиком (собственные ф-ции работы с файлами собственной ф. с.;

Указатель f_mapping показывает связь структур, описывающих файлы в системе, с памятью;

Также в struct inode есть поле i_mapping;

struct superblock содержит список inode (s_inodes). struct inode содержит указатель на соответствующий inode в списке (i_sb_list);

Любая ф. с. имеет корневой каталог, и именно от корневого каталога формируется путь к файлу для конкретной ф. с.

Отправная точка — сист. вызовы (read, write, lseek,...). Здесь нет open(), т.к. он открывает файл, а использование ф-ций read, write, lseek возможно только при работе с открытым файлом;

Теперь пойдём от процесса:

Отправная точка — struct task_struct;

В struct task_struct есть 2 указателя:

— на struct fs_struct (*fs);

Любой процесс относится к какой-то ф. с.

— на struct files_struct — дескриптор, описывающий файл, открытый процессом;

Любой процесс имеет собственную таблицу открытых файлов

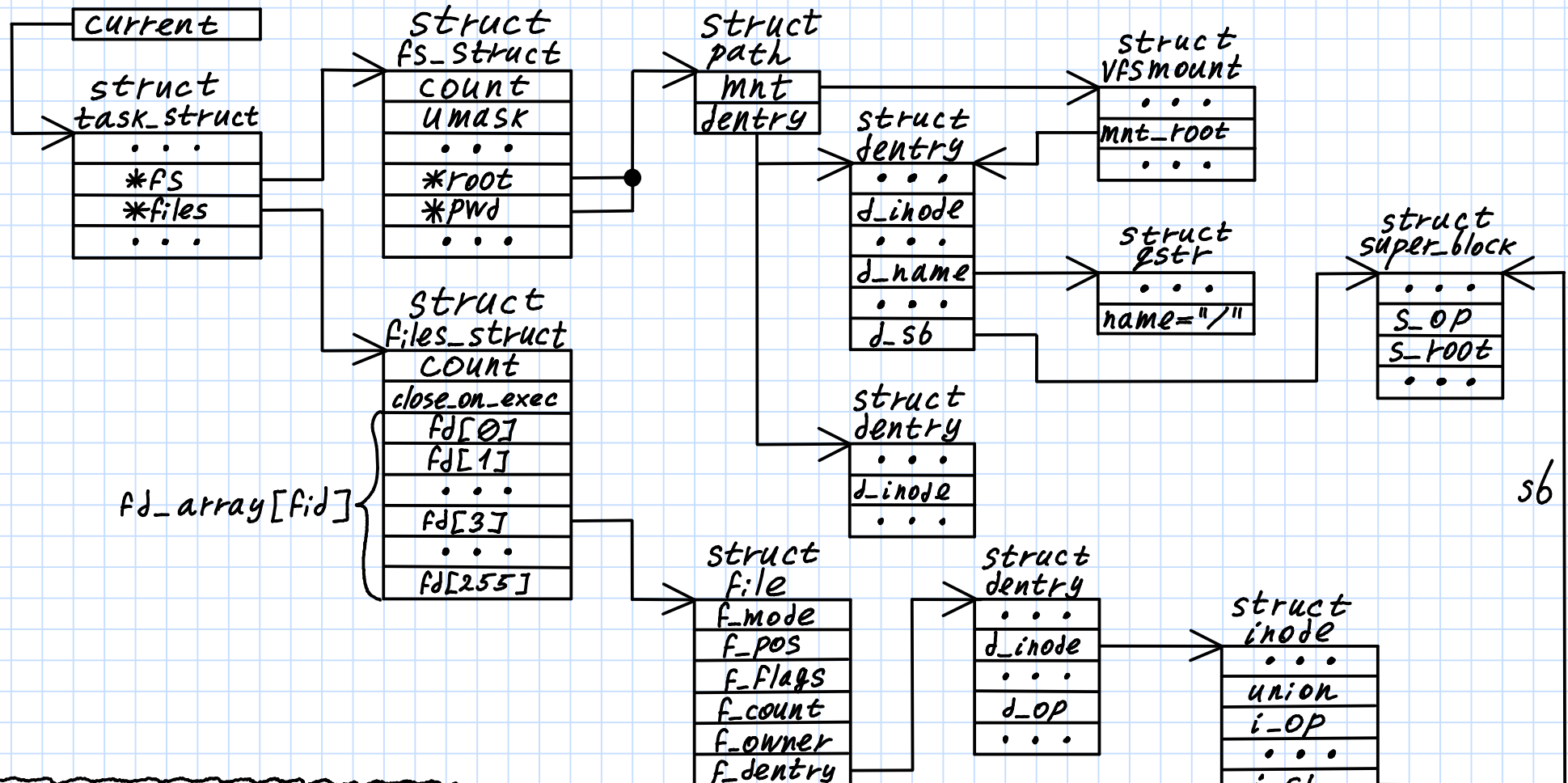Очевидно, что struct files_struct содержит массив дескрипторов открытых файлов $(0, 1, 2, 3, 4, \ldots)$;

При этом:

0 — stdin
1 — stdout
2 — stderr

$\left.\right\}$ эти файлы открываются
для процесса автоматически
(файловые дескрипторы для этих
файлов создаются автоматически)

Когда мы открываем файл, он может получить дескриптор после этих трёх (например, 3, 4, 5 и т.д.);

Всего в этой таблице м.б. 256 дескрипторов;

Всё равно, какой процесс рассматривать, возьмём current:

Связи структур (относительно процесса)

current

struct task_struct
. . .
*fs
*files
. . .

struct fs_struct
count
umask
. . .
*root
*pwd
. . .

struct files_struct
count
close_on_exec
fd[0]
fd[1]
. . .
fd[3]
. . .
fd[255]

fd_array[fid]

struct path
mnt
dentry

struct vfsmount
. . .
mnt_root
. . .

struct dentry
. . .
d_inode
. . .
d_name
. . .
d_sb

struct qstr
. . .
name="/"

struct super_block
. . .
s_op
s_root
. . .

sb

struct dentry
. . .
d_inode
. . .

struct file
f_mode
f_pos
f_flags
f_count
f_owner
f_dentry
f_op
. . .
f_mapping

struct address_space

struct dentry
. . .
d_inode
. . .
d_op
. . .

struct inode
. . .
union
i_op
. . .
i_sb
i_mapping
. . .

struct file_operations
. . .
read
write
lseek
. . .

struct super_operations
alloc_inode
destroy_inode
dirty_inode
write_inode
drop_inode
. . .

struct dentry_operations
. . .
d_hash
d_compare
. . .
d_init
d_release
. . .
d_dname

struct inode_operations
lookup
getlink
. . .
link
symlink
mkdir
rmdir
mknod
rename
. . .

struct vfs_mount заполняется, когда
ф.с. монтируется;
имя — указатель на struct qstr;

В struct superblock есть указатель
на struct super_operations (s_op) и на root
(s_root), т.к. корневой каталог (точка монти-
рования) должен быть создан, чтобы иметь
возможность смонтировать ф.с.;

N/p:
Создание корневого каталога позволяет под-
монтировать нашу ф.с. к дереву каталогов
(дереву файловых систем) Linux;

Объекты dentry создаются „на лету";
struct dentry описывает элемент пути;
Элемент пути — часть пути к файлу, ко-
торые отделяются друг от друга "/", начиная
с корневого каталога;

struct dentry_operations:
- d_hash - хеширование;
- d_compare - сравнение: когда мы проходим
по пути к файлу, мы сравниваем заданное
имя и найденное;
- d_init - выделение dentry;

- _d_release_ - освобождение dentry: освободить dentry можно, если на него нет ссылок;
- _d_dname_ - определение/создание пути к эл-ту (объекту) dentry;

Для того, чтобы создать собственную ф.с., в struct superblock есть поле filesystem_type (структура ядра);

На каждый тип ф.с. должны быть инициализированы поля struct filesystem_type;

После описания ф.с., ядро предоставляет возможность зарегистрировать/удалить ф.с.;

Структура, описывающая конкретный тип ф.с., м.б. только одна;

При этом одна и та же ф.с. (ф.с., имеющая один и тот же тип) м.б. подмонтирована много раз;

〰〰〰〰〰〰〰〰〰〰〰〰〰〰〰〰〰〰〰

Помимо таблицы открытых файлов процесса (есть у каждого процесса), в системе есть одна таблица на все открытые файлы;

Причём в этой таблице на один и тот же файл (с одним и тем же inode) м.б. создано большое кол-во дескрипторов открытых файлов, т.к. один и тот же файл м.б. открыт много раз (одним и тем

же процессом или разными процессами);

Он будет иметь много дескрипторов struct file;

Каждое открытие файла с одним и тем же inode приведёт к созданию его дескриптора (как открытого файла);

При открытии файла его дескриптор добавляется

— в таблицу открытых файлов процесса (struct files_struct): будет добавлен fd[...];

— в системную таблицу открытых файлов: будет добавлен дескриптор struct file;

Каждый дескриптор struct file имеет поле f_pos (position), и это приводит к гонкам. При работе с файлами это надо учитывать;

Один и тот же файл, открытый много раз, без соотв. способов взаимоисключения, будет фактически „атакован", и это приведёт к потере данных;

```c
struct filesystem_type
{
        const char *name;
        int fs_flags;
#define FS_REQUIRES_DEV 1
        ...
#define FS_USERNS_MOUNT 8
        ...
        struct dentry *(*mount)(struct file_system_type*,   // → раньше было get_sb(..)
                int, const char *, void *);
        void (*kill_sb)(struct super_block *);
        struct file_system_type *next;          // в системе имеется связный список типов ф.с.
        struct hlist_head fs_supers;            // для копир. ф.с. имеется список (hash list) объектов superblock
        struct lock_class_key s_lock-key;       // эти и следующие поля определяют соотв. блокировки
        struct lock_class_key s_umount;         // при работе с файловыми системами
        struct lock_class_key s_vfs_rename-key;
        ...
};
```

В ядре определено несколько фо-ций mount():

```c
extern struct dentry *mount_bdev(struct    // → block device
        file_system_type *fs_type, int flags,
        const char *dev_name, void *data,
        int (*fill_super)(struct super_block *, void *, int));
```
„отсутствие device"
```c
extern struct dentry *mount_nodev(struct
        file_system_type *fs_type, int flags, void *data,
        int (*fill_super)(struct super_block *, void*, int));
```

На ф.с., которую мы создаём, должны быть определены флаги, важные для разработчика;

Есть ещё ф-ция mount_single(), mount_ns() и mount_single();

И в mount_bdev(), и в mount_nodev() вызывается ф-ция fill_super(), которая выполняет основные действия по инициализации struct superblock;

Эти ф-ции возвращают объект dentry, и этим объектом должен быть root;

Для ф.с. необходимо создать root. Это позволит выполнить монтирование ф.с.;

Для root надо создать inode;

Пример создания собств. ф.с. :

1) инициализация полей структуры file_system_type:

```
static struct file_system_type lfs_type =
{
        .owner = THIS_MODULE,
        .name  = "myfs",
        .mount = myfs_mount,
        .kill_sb = kill_litter_super
};
```

В ф-ции myfs_mount можно вызвать mount_bdev()/mount_nodev()/mount_single();

Почитать про simple и generic ф-ции