

Дополнительно о sequence file.
В stop к.м.б. вызвана ф-ция
seq-printf(), но м.б. вызвана printf();

Сист. вызов open()

Читать мануал, см. флаги;

Особенности вызова open:

Необходимо четко понимать, что мы хотим;

Если в fopen() указать w (write), то вся информация в пред. файле будет потеряна;

флаг O_EXCL; если установлен O_CREAT, то, если существует путь к файлу (указан существующий путь \Rightarrow указано имя существующего файла), будет возвращена ошибка;

Библиотечная ф-ция fopen() этого не делает;

2 варианта open():

1) Если ф-ция open() предназначена для работы с существующим файлом, то эта ф-ция вызывается с 2 параметрами:

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

2) Если пользователь желает создать файл и использовать флаг O_CREAT или O_TMPFILE, то он должен указать 3-й пар-р — mode;

Если эти флаги не указаны, то 3-й пар-р игнорируется;

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

Все флаги из методички важны для анализа посл-ти действий, выполняемых ядром при открытии файла;

Так, можно открыть существующий файл, а можно открыть новый файл;
= создать файл

Создать файл прежде всего создать inode

Любое действие в системе связано с инициализацией одной или нескольких структур;

Работа с памятью прежде всего связана с созданием таблиц страниц;

В коде `foren()` из `studio.h` есть вызов `open()`;

Сист. вызов `open()` приведёт к переходу в ядро;

Любой сист. вызов приведёт к переходу системы в ядро;

В ядре есть `syscall table`;

В системе есть 6 макросов —

— `system call macro`;

У всех 1-й пар-р — имя сист. вызова;

С `open()` работает так:

`SYSCALL_DEFINE3(open,`

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

интерфейсный
макрос

`filename` - имя файла, которое передается из пр-ва пользователя в пр-во ядра;

Это нельзя сделать напрямую;

Вместо этого будет вызвана ф-ция

`strncpy_from_user()` для передачи имени файла в ядро (это делается посл-тельно в рез-те ряда вызовов ф-ций);

Ф-ции ядра специализированы, но не стандартизованы (в отличие от сист. вызовов, которые стандартизованы POSIX);

Поэтому ф-ции и структуры ядра переносываются;

Код ядра очень сложный, т.к. в нём много ветвлений, обусловленных значениями флагов;

Для того, чтобы определить, существует ли файл, нужно пройти по цепочке `dentry` (задействуется `struct dentry`);

Флаги:

Все связи структур, рассм. для ф.с., здесь задействованы;

Открытый файл - файл, который открывает процесс;

Для такого файла создаётся дескриптор файла в таблице открытых файлов процесса (`struct Files_struct`);

Но этого мало;

Необходимо создать дескриптор открытого файла в сист. таблице открытых файлов (`struct file`);

Любой файл принадлежит конкретному ф.с. (`struct fs_struct`);

`struct file` имеет указатель на `struct inode`;

Режим (`mode`, права доступа):

Если мы создаём новый файл, то мы должны указать права доступа к этому файлу;

```
SYSCALL_DEFINE3(open,  
                    const char __user *filename,  
                    int flags,  
                    mode_t mode)  
{  
    if (force_o_largefile())  
        flags |= O_LARGEFILE;  
    return do_sys_open(AT_FDCWD, filename,  
                       flags, mode);  
}
```


В макросе выполняется проверка того, какая у нас система: если 64-разр., то в ней есть большие файлы (large file), и флаг O_LARGEFILE добавляется к флагам, которые были установлены;

Основная задача макроса - вызов ф-ции ядра do_sys_open();

goto в ядре сейчас используется для сокращения кол-ва одинаковых сообщений об ошибках;

Чтобы нарисовать схему алгоритма, можно выделить "начальные" действия, взять их за основу и далее представлять алгоритм отдельными основными ф-циями;

Должен быть анализ флагов O_TMPFILE, O_CREAT, O_PATH, O_APPEND, O_EXCL (рядом с O_CREAT);

```

long do_sys_open(int dfd, const char__user *filename,
                 int flags, umode_t mode)
{
    struct open_flags op;
    int fd = build_open_flags(flags, mode, &op);
    struct filename *tmp;
    if (fd) return fd;
    tmp = getname(filename);
    if (IS_ERR(tmp)) return PTR_ERR(tmp);
    fd = get_unused_fd_flags(flags); // обёртка оп-цум
    if (fd >= 0) {
        struct file *f = do_filp_open(dfd, tmp, &op);
        if (IS_ERR(f)) {
            put_unused_fd(f);
            fd = PTR_ERR(*);
        } else {
            fnotify_open(f);
        }
    }
    putname(tmp);
    return (fd);
}

```

Основную работу по открытию файла и связанные с этим действия выполняет оп-цум do_filp_open();

Задача оп-цум build_open_flags — инициализация полей struct open_flags на основе флагов, указанных пользователем;

В этой ф-ции анализируются все файлы;

Алгоритм этой ф-ции надо привести в отчёте

Можно предположить, что ф-ция `get_unused_fd_flags()` должна найти неиспользуемый файловый дескриптор в таблице дескрипторов открытых файлов для того, чтобы выделить его, и `open()` мог его вернуть;

При этом ф-ция `_alloc_fd()` использует spin-лок'и, т.к. эти действия могут выполняться несколькими процессами/потоками;
(надо отразить на схеме алгоритма)

Ф-ция `getname()` вызывает `get_name_flags()`, которая копирует имя файла из пр-ва пользователя в пр-во ядра;

При этом исп-ся ф-ция `str_copy_from_user()`;
(надо отразить в отчёте);

Про „`if (fd >= 0)`“:

Для любого процесса файловые дескрипторы 0, 1 и 2 (`stdin`, `stdout`, `stderr`) занимаютсЯ автоматически, но для этих дескрипторов необходимо проделывать все действия так же, как при вызове `open()` в приложении;

В отчёте надо включить алгоритм ф-ции `fd_file_open()`;

```
struct file *do_filp_open(int dfd,
    struct filename *pathname,
    const struct open_flags *op)
```

Эта структура
инициализируется
в раз-ме работы
с-тии, которые
были вызваны
ранее

```
{
    struct nameidata nd; // внутренняя/сущестная  
                           структура (присутствует)  
    int flags = op->lookup_flags;  
    struct file *filp;  
    set_nameidata(&nd, dfd, pathname);  
    filp = path_openat(&nd, op, READ-COPY-UPDATE  
                       flags | LOOKUP_RCU);  
    if (unlikely(...))  
        filp = path_openat(&nd, op, flags);  
    if (unlikely(...))  
        filp = path_openat(&nd, op, READ-COPY-UPDATE  
                           flags | LOOKUP_REVAL);  
    restore_nameidata();  
    return filp;  
}
```

Ф-ция set_nameidata() инициализирует новую
struct nameidata;

Ф-ция path_openat() возвращает инициализированный
дескриптор открытого файла (struct file)

1-й вызов:

„Быстрый проход“: инициализируются некоторые
проверки;

Это проход по всем файлам и определение соотв.
анализа;

2-й вызов:

„Обычный“ (если быстрый проход вернул ошибку);

3-й вызов:

Для файлов NFS (network filesystem);

В NFS не работает флаг O_APPEND;

O_APPEND позволяет дописывать данные в конец файла без потери данных в нём;

Ф.с., предназначенная для работы с сетью не обеспечивает безопасной работы с файлом, даже если установлен флаг O_APPEND; Всё равно возникают гонки;

Гонки при разделении файлов: один и тот же файл м.б. открыт разными процессами;

Если установлен флаг O_CREAT и указано несуществующее имя файла (система это контролирует), то должен быть создан inode;

Это почти конец цепочки вызовов ф-ций ядра, и там необходимо обратить внимание на флаги O_EXCL и семафоры read/write;

Flashback: читатели - писатели:

Если информация только читается, то возможно разделение (share), т.е. совместное чтение;

Если информация изменяется, то возможен только монопольный доступ;

(отразить это в схемах алгоритмов соотв. ф-ций)

```
static struct File *path_opend(
    struct nameidata *nd,
    const struct open_Flags *op,
    unsigned flags)
{
```

```
    struct File *file;
    int error;
    file = alloc_empty_file(op->open_Flags,
                           current_cred());
```

```
    //err
```

```
    if (unlikely(file->f_flags & _O_TMPFILE))
        error = do_tmpfile(...);
```

```
    else if (unlikely(file->f_flags & _O_PATH))
        do_opath(...);
```

```
    else
```

```
{
```

```
    const char *s = path_init(nd, flags);
    while (! (error = link_path_walk(s, nd) &&
        error = do_last(nd, file, op)) > 0)
    {
```

```
        nd->flags &= ~(LOOKUP_OPEN |
                       LOOKUP_CREATE |
                       LOOKUP_EXCL);
```

```
        s = trailing_symlink(nd);
```

```
    } //else
```

```
    terminate_walk(nd);
```

```
}
```

```
... ? ...
```

```
}
```

Флаг O_TMPFILE предлагает создание временного файла;
Если он установлен, будет вызвана ф-ция `do_tmpfile()`;

Ф-ция `do_last()` вызывает `audit_inode()`;

Обратить внимание на ф-цию `vfs_open`
и на ф-ции проверки прав доступа к `inode`:

- `security_inode_permission()`;
- `security_inode_create()`;