МГТУ им. Баумана

Лабораторная работа №6

По курсу: "Операционные системы"

# Системный вызов open()

Работу выполнил: студент группы ИУ7-63Б
Наместник Анастасия

Преподаватели: Рязанова Н. Ю.

*Москва, 2021*

Версия ядра Linux: v5.8

Рассматриваемые структуры:

**struct filename**

```
struct filename {
  const char    *name;  /* pointer to actual string */
  const __user char *uptr;  /* original userland pointer */
  int      refcnt;
  struct audit_names  *aname;
  const char    iname[];
};
```

Описание структуры находится в источнике:

https://elixir.bootlin.com/linux/v5.8/source/include/linux/fs.h#L2537

**struct open_flags**

```
struct open_flags {
  int open_flag;
  umode_t mode;
  int acc_mode;
  int intent;
  int lookup_flags;
};
```

Описание структуры находится в источнике:

https://elixir.bootlin.com/linux/v5.8/source/fs/internal.h#L114

**struct open_how**

```
struct open_how {
  __u64 flags;
  __u64 mode;
  __u64 resolve;
};
```

Описание структуры находится в источнике:

https://elixir.bootlin.com/linux/v5.8/source/include/uapi/linux/openat2.h#L19

**struct file**

```
struct file {
  union {
    struct llist_node fu_llist;
    struct rcu_head   fu_rcuhead;
  } f_u;
```

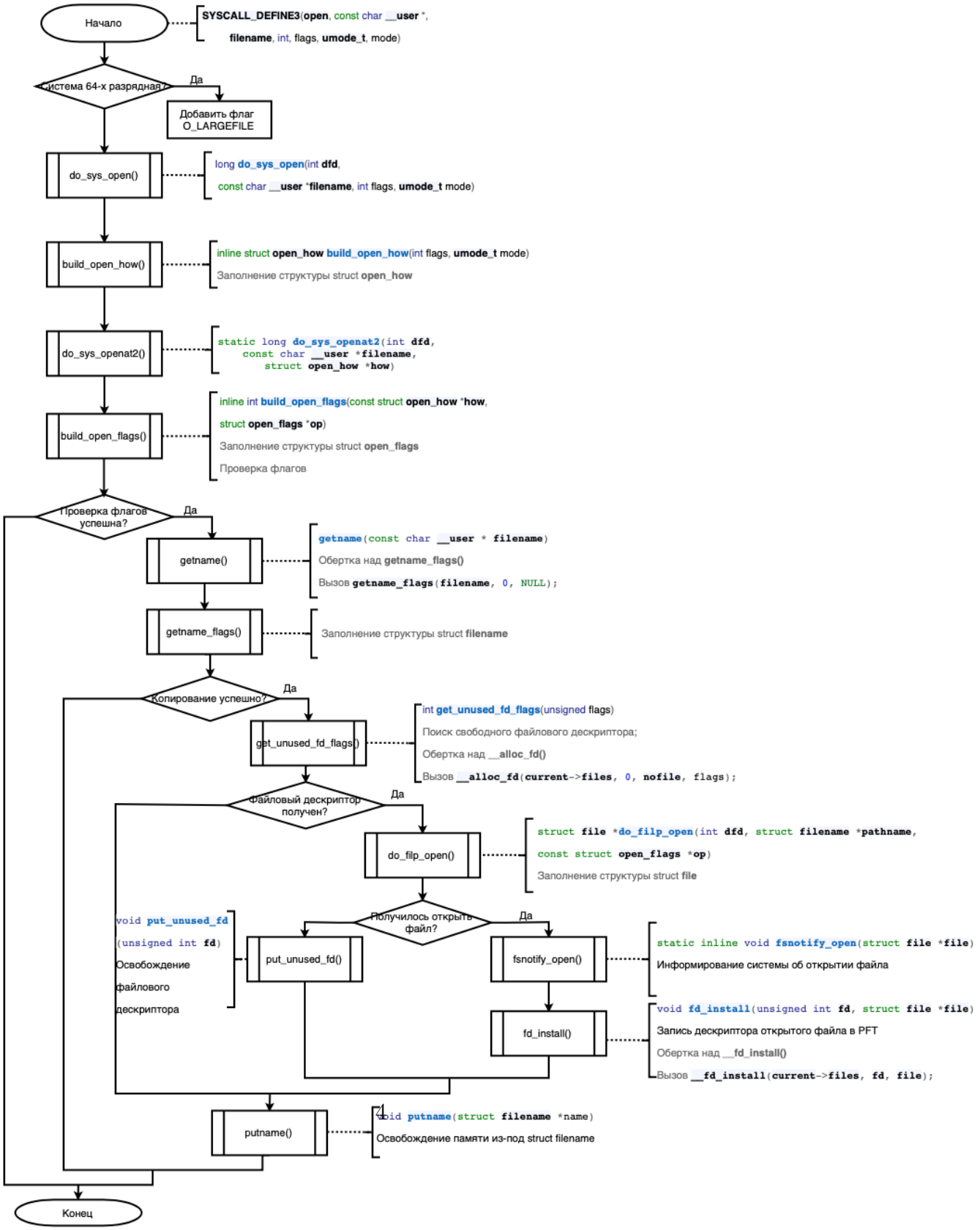
```
6     struct path   f_path;
7     struct inode    *f_inode; /* cached value */
8     const struct file_operations  *f_op;
9
10    /*
11     * Protects f_ep_links, f_flags.
12     * Must not be taken from IRQ context.
13     */
14    spinlock_t    f_lock;
15    enum rw_hint    f_write_hint;
16    atomic_long_t   f_count;
17    unsigned int    f_flags;
18    fmode_t     f_mode;
19    struct mutex    f_pos_lock;
20    loff_t      f_pos;
21    struct fown_struct  f_owner;
22    const struct cred *f_cred;
23    struct file_ra_state  f_ra;
24
25    u64     f_version;
26 #ifdef CONFIG_SECURITY
27    void      *f_security;
28 #endif
29    /* needed for tty driver, and maybe others */
30    void      *private_data;
31
32 #ifdef CONFIG_EPOLL
33    /* Used by fs/eventpoll.c to link all the hooks to this
          file */
34    struct list_head  f_ep_links;
35    struct list_head  f_tfile_llink;
36 #endif /* #ifdef CONFIG_EPOLL */
37    struct address_space  *f_mapping;
38    errseq_t    f_wb_err;
39    errseq_t    f_sb_err; /* for syncfs */
40 } __randomize_layout
41    __attribute__((aligned(4)));  /* lest something weird
          decides that 2 is OK */
```

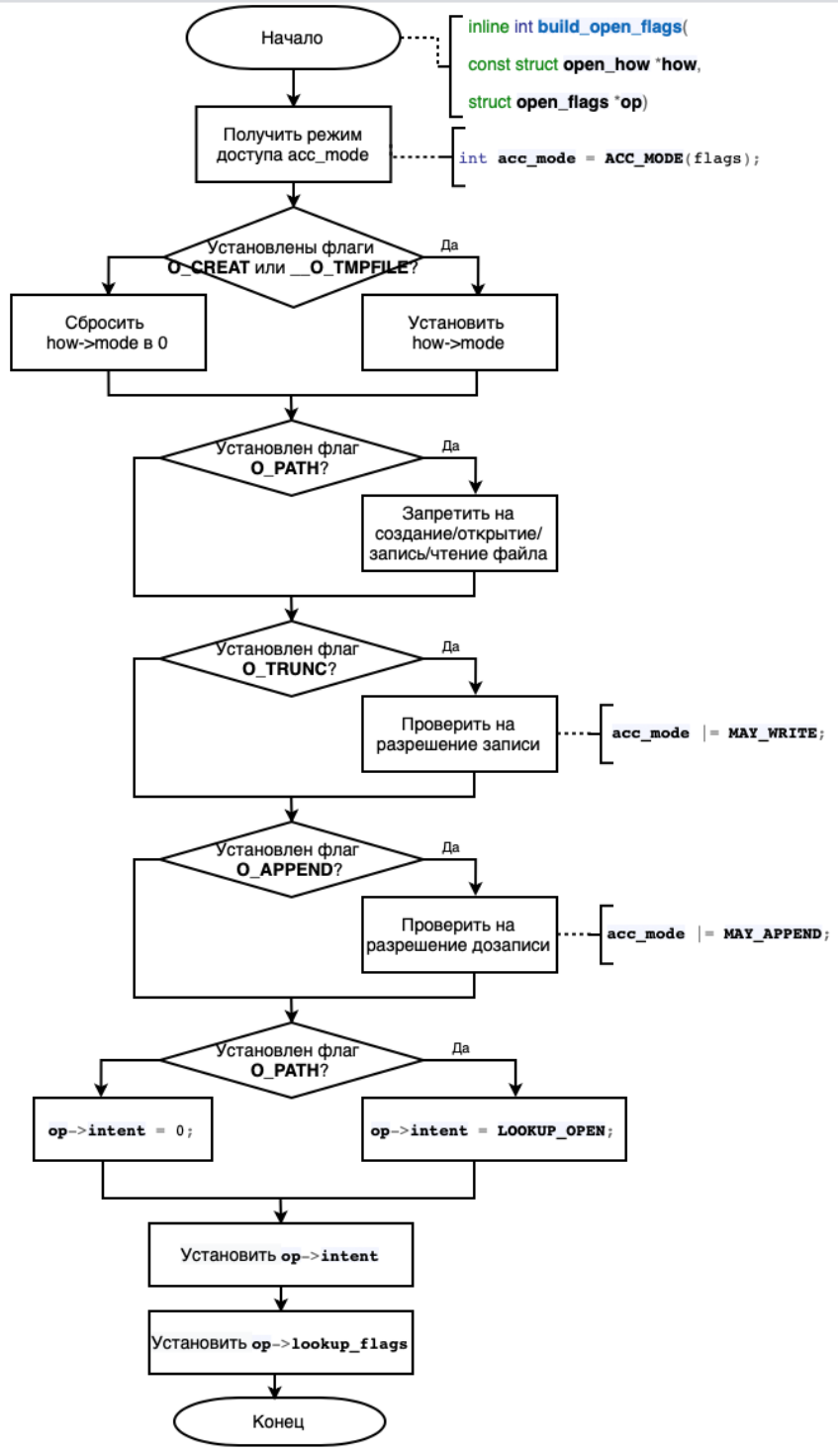
Описание структуры находится в источнике:

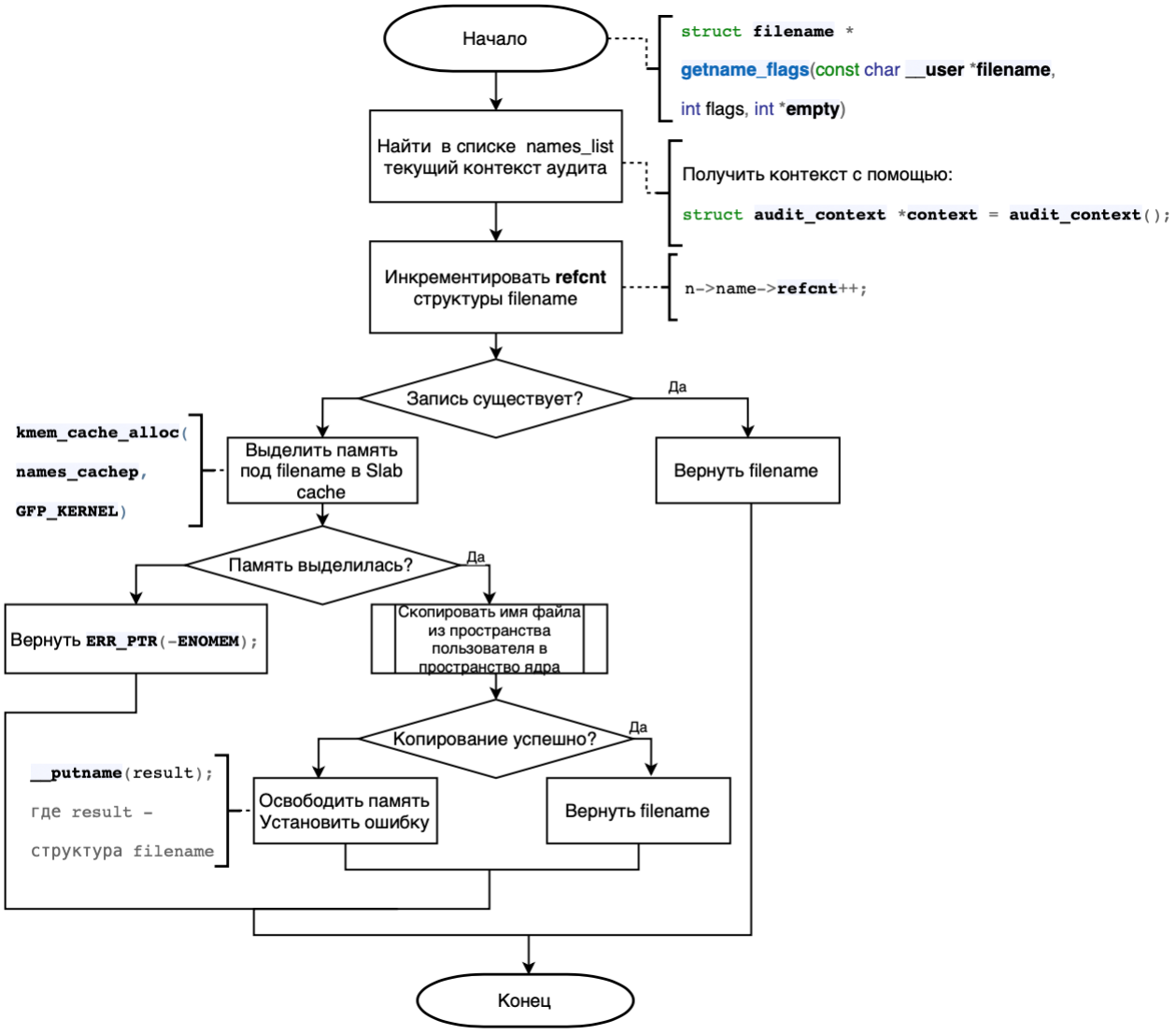
https://elixir.bootlin.com/linux/v5.8/source/include/linux/fs.h#L944
**struct nameidata**

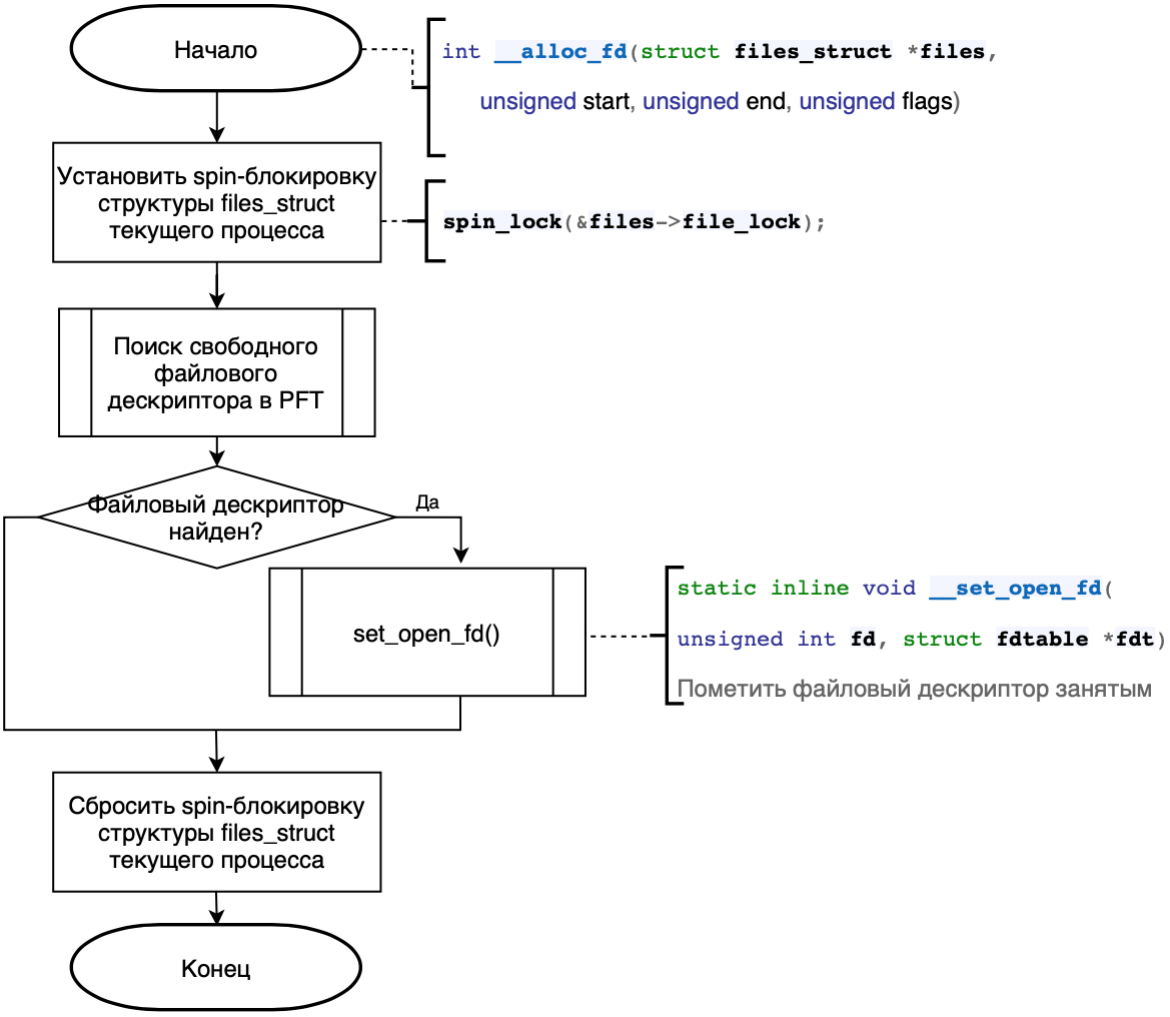
```
struct nameidata {
  struct path path;
  struct qstr last;
  struct path root;
  struct inode  *inode; /* path.dentry.d_inode */
  unsigned int  flags;
  unsigned  seq, m_seq, r_seq;
  int   last_type;
  unsigned  depth;
  int   total_link_count;
  struct saved {
    struct path link;
    struct delayed_call done;
    const char *name;
    unsigned seq;
  } *stack, internal[EMBEDDED_LEVELS];
  struct filename *name;
  struct nameidata *saved;
  unsigned  root_seq;
  int   dfd;
  kuid_t    dir_uid;
  umode_t   dir_mode;
} __randomize_layout;
```

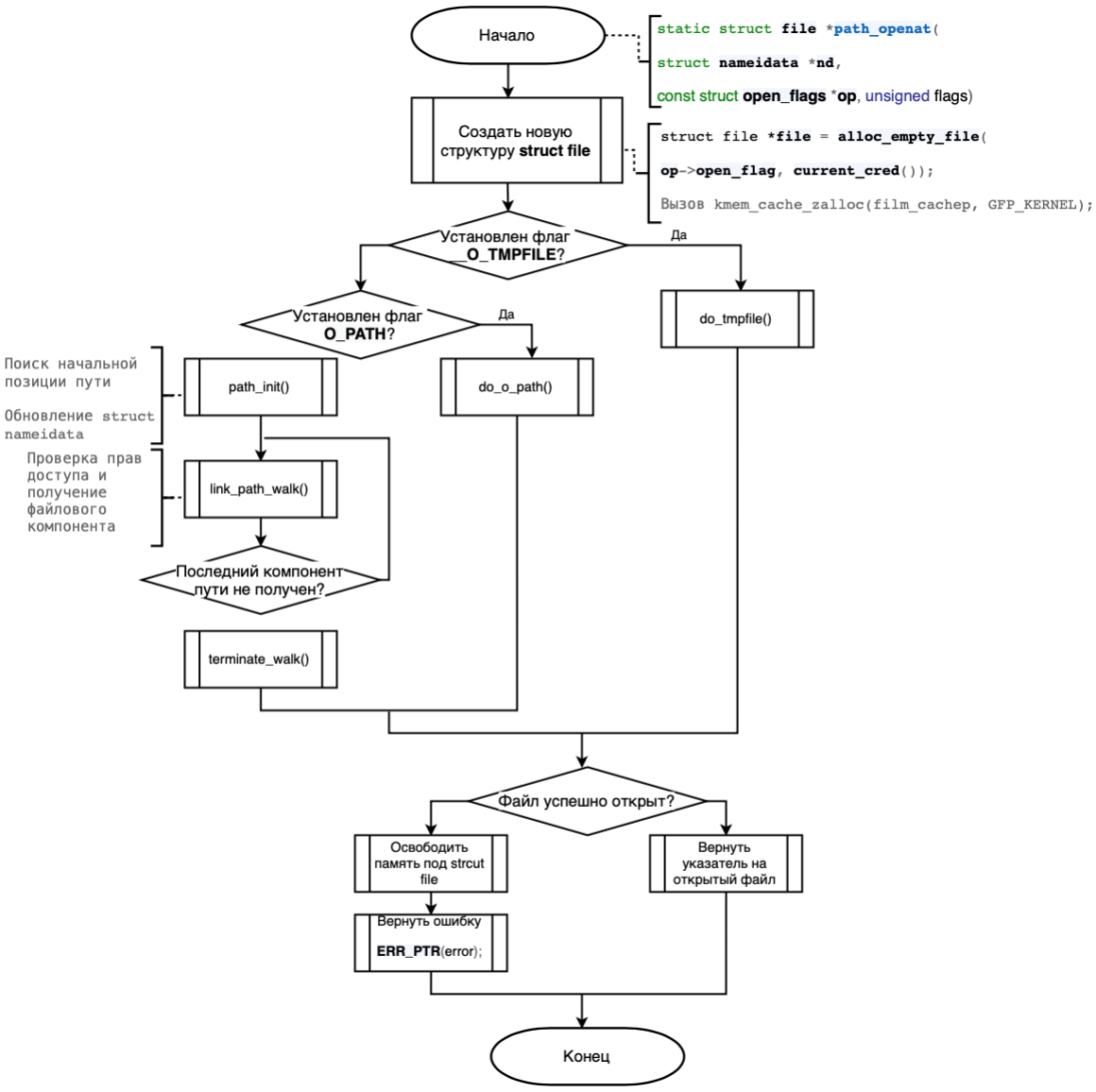
Описание структуры находится в источнике:
https://elixir.bootlin.com/linux/v5.8/source/fs/namei.c#L502

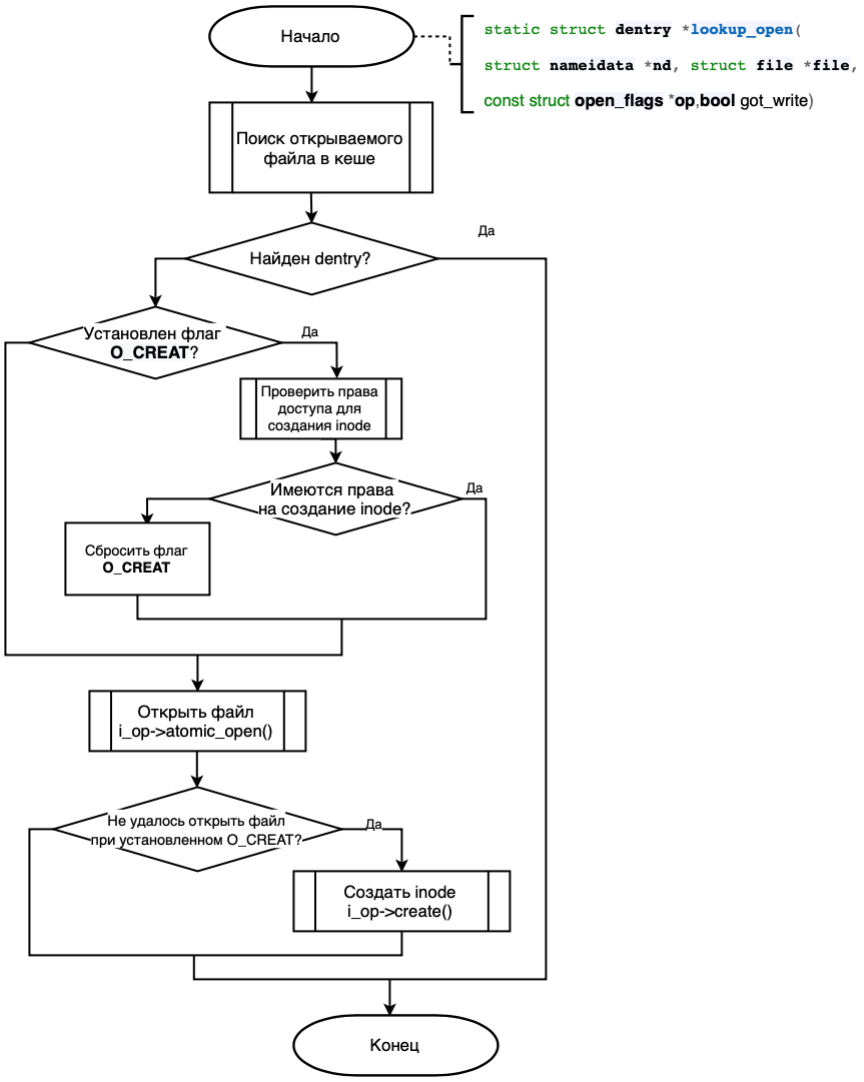

Начало
SYSCALL_DEFINE3(open, const char __user *, filename, int, flags, umode_t, mode)
Система 64-х разрядная?
Да
Добавить флаг O_LARGEFILE
do_sys_open()
long do_sys_open(int dfd, const char __user *filename, int flags, umode_t mode)
build_open_how()
inline struct open_how build_open_how(int flags, umode_t mode)
Заполнение структуры struct open_how
do_sys_openat2()
static long do_sys_openat2(int dfd, const char __user *filename, struct open_how *how)
build_open_flags()
inline int build_open_flags(const struct open_how *how, struct open_flags *op)
Заполнение структуры struct open_flags
Проверка флагов
Проверка флагов успешна?
Да
getname()
getname(const char __user * filename)
Обертка над getname_flags()
Вызов getname_flags(filename, 0, NULL);
getname_flags()
Заполнение структуры struct filename
Копирование успешно?
Да
get_unused_fd_flags()
int get_unused_fd_flags(unsigned flags)
Поиск свободного файлового дескриптора;
Обертка над __alloc_fd()
Вызов __alloc_fd(current->files, 0, nofile, flags);
Файловый дескриптор получен?
Да
do_filp_open()
struct file *do_filp_open(int dfd, struct filename *pathname, const struct open_flags *op)
Заполнение структуры struct file
Получилось открыть файл?
Да
void put_unused_fd (unsigned int fd)
Освобождение файлового дескриптора
put_unused_fd()
fsnotify_open()
static inline void fsnotify_open(struct file *file)
Информирование системы об открытии файла
fd_install()
void fd_install(unsigned int fd, struct file *file)
Запись дескриптора открытого файла в PFT
Обертка над __fd_install()
Вызов __fd_install(current->files, fd, file);
putname()
void putname(struct filename *name)
Освобождение памяти из-под struct filename
Конец


Начало
inline int build_open_flags(
const struct open_how *how,
struct open_flags *op)
Получить режим доступа acc_mode
int acc_mode = ACC_MODE(flags);
Установлены флаги O_CREAT или __O_TMPFILE?
Да
Сбросить how->mode в 0
Установить how->mode
Установлен флаг O_PATH?
Да
Запретить на создание/открытие/ запись/чтение файла
Установлен флаг O_TRUNC?
Да
Проверить на разрешение записи
acc_mode |= MAY_WRITE;
Установлен флаг O_APPEND?
Да
Проверить на разрешение дозаписи
acc_mode |= MAY_APPEND;
Установлен флаг O_PATH?
Да
op->intent = 0;
op->intent = LOOKUP_OPEN;
Установить op->intent
Установить op->lookup_flags
Конец


Начало
struct filename *
getname_flags(const char __user *filename,
int flags, int *empty)
Найти в списке names_list текущий контекст аудита
Получить контекст с помощью:
struct audit_context *context = audit_context();
Инкрементировать refcnt структуры filename
n->name->refcnt++;
Запись существует?
Да
kmem_cache_alloc(
names_cachep,
GFP_KERNEL)
Выделить память под filename в Slab cache
Вернуть filename
Память выделилась?
Да
Вернуть ERR_PTR(-ENOMEM);
Скопировать имя файла из пространства пользователя в пространство ядра
Копирование успешно?
Да
__putname(result);
где result - структура filename
Освободить память
Установить ошибку
Вернуть filename
Конец


Начало
int __alloc_fd(struct files_struct *files,
unsigned start, unsigned end, unsigned flags)
Установить spin-блокировку структуры files_struct текущего процесса
spin_lock(&files->file_lock);
Поиск свободного файлового дескриптора в PFT
Файловый дескриптор найден?
Да
set_open_fd()
static inline void __set_open_fd(
unsigned int fd, struct fdtable *fdt)
Пометить файловый дескриптор занятым
Сбросить spin-блокировку структуры files_struct текущего процесса
Конец


Начало
static struct file *path_openat(
struct nameidata *nd,
const struct open_flags *op, unsigned flags)
Создать новую структуру struct file
struct file *file = alloc_empty_file(
op->open_flag, current_cred());
Вызов kmem_cache_zalloc(film_cachep, GFP_KERNEL);
Установлен флаг O_TMPFILE?
Да
do_tmpfile()
Установлен флаг O_PATH?
Да
do_o_path()
Поиск начальной позиции пути
path_init()
Обновление struct nameidata
Проверка прав доступа и получение файлового компонента
link_path_walk()
Последний компонент пути не получен?
terminate_walk()
Файл успешно открыт?
Освободить память под strcut file
Вернуть ошибку
ERR_PTR(error);
Вернуть указатель на открытый файл
Конец




## 0.1 Флаги open()

**O_CREAT** (если файл не существует, то он будет создан. Владелец (идентификатор пользователя) файла устанавливается в значение эффективного идентификатора пользователя процесса. Группа (идентификатор группы) устанавливается либо в значение эффективного идентификатора группы процесса, либо в значение идентификатора группы ро-

дительского каталога (зависит от типа файловой системы, параметров подсоединения (mount) и режима родительского каталога, см. например, параметры подсоединения bsdgroups и sysvgroups файловой системы ext2.

**O_EXCL** (Если он используется совместно с O_CREAT, то при наличии уже созданного файла вызов open завершится с ошибкой. В этом состоянии, при существующей символьной ссылке не обращается внимание, на что она указывает.)

**O_NOCTTY** (если pathname указывает на терминальное устройство, то оно не станет терминалом управления процесса, даже если процесс такового не имеет);

**O_TRUNC** (если файл уже существует, он является обычным файлом и режим позволяет записывать в этот файл (т.е. установлено O_RDWR или O_WRONLY), то его длина будет урезана до нуля. Если файл является каналом FIFO или терминальным устройством, то этот флаг игнорируется. Иначе действие флага O_TRUNC не определено.

**O_APPEND** (Файл открывается в режиме добавления. Перед каждой операцией write файловый указатель будет устанавливаться в конце файла, как если бы использовался lseek); O_APPEND (может привести к повреждению файлов в системе NFS, если несколько процессов одновременно добавляют данные в один файл. Это происходит из-за того, что NFS не поддерживает добавление в файл данных, поэтому ядро на машине-клиенте должно эмулировать эту поддержку);

**O_NONBLOCK или O_NDELAY** (если возможно, то файл открывается в режиме non-blocking. Ни open, ни другие последующие операции над возвращаемым описателем файла не заставляют вызывающий процесс ждать. Для работы с каналами FIFO. Этот режим не оказывает никакого действия на не-FIFO файлы.);

**O_SYNC** (Файл открывается в режиме синхронного ввода-вывода. Все вызовы write для соответствующего описателя файла блокируют вызывающий процесс до тех пор, пока данные не будут физически записаны.)

**O_NOFOLLOW** (если pathname - это символьная ссылка, то open содержит код ошибки. Это расширение FreeBSD, которое было добавлено в Linux версии 2.1.126. Все прочие символьные ссылки в имени будут обработаны как обычно. Заголовочные файлы из glibc версии 2.0.100 (и более поздних) содержат определение этого флага; ядра версий, более ранних, чем 2.1.126, игнорируют этот флаг)

**O_DIRECTORY** (Если pathname не является каталогом, то open укажет на ошибку. Этот флаг используется только в Linux и был добавлен к ядру 2.1.126, чтобы избежать проблем с "отказом от обслуживания если opendir(2) был вызван для канала FIFO или ленточного устройства. Этот флаг не следует использовать вне реализации opendir );

**O_LARGEFILE** (На 32-битных системах, поддерживающих файловые системы (Large), этот флаг позволяет открывать файлы, длина которых больше 31-ого бита.