**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н. Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н. Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

# Отчет по лабораторной работе №6 по курсу "Операционные системы"

**Тема** Системный вызов open()

**Студент** Цветков И.А.

**Группа** ИУ7-63Б

**Оценка (баллы)**

**Преподаватель** Рязанова Н. Ю.

Москва — 2022 г.

# Системный вызов open()

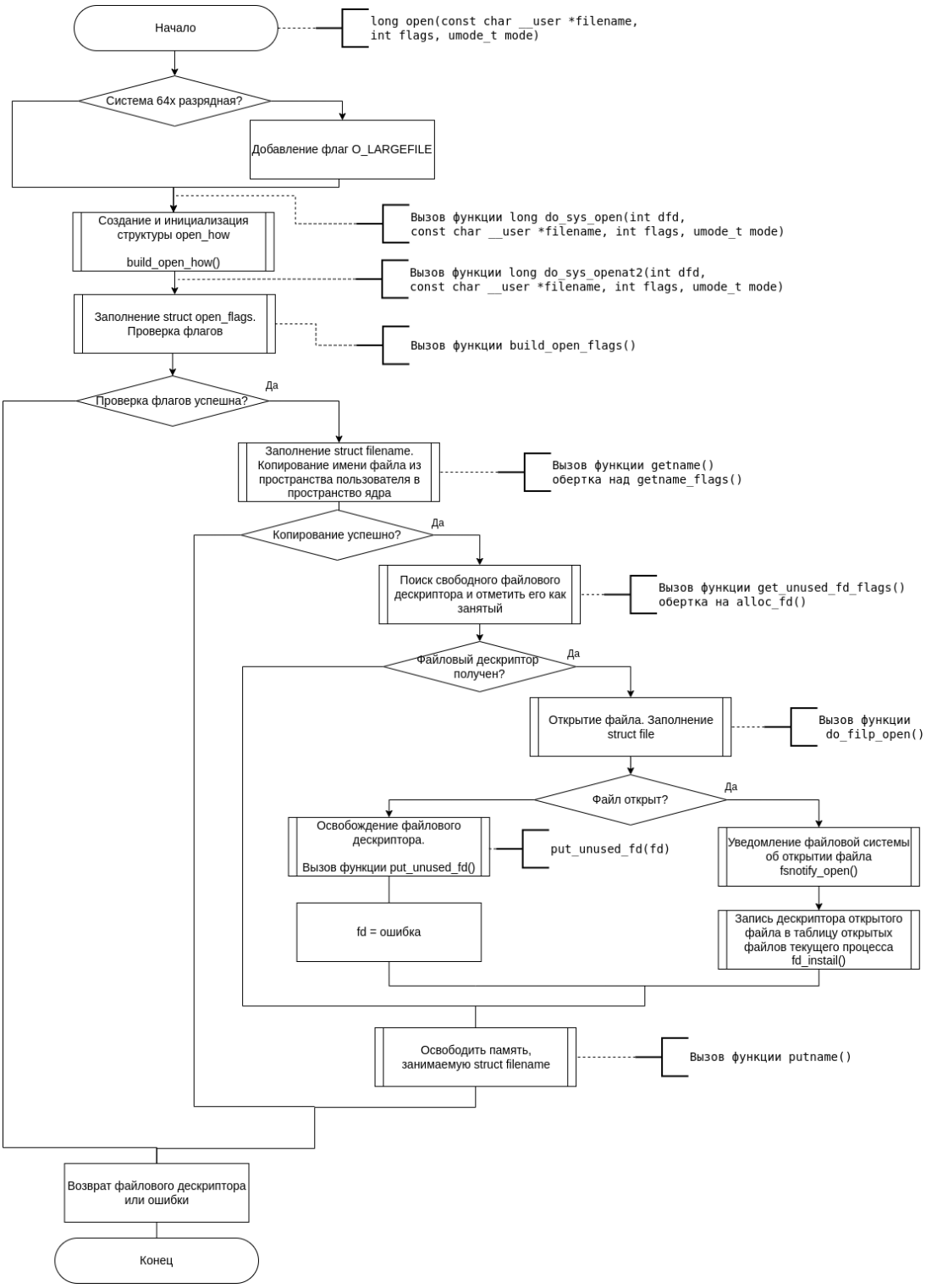


Рисунок 1 – Схема алгоритма системного вызова open()

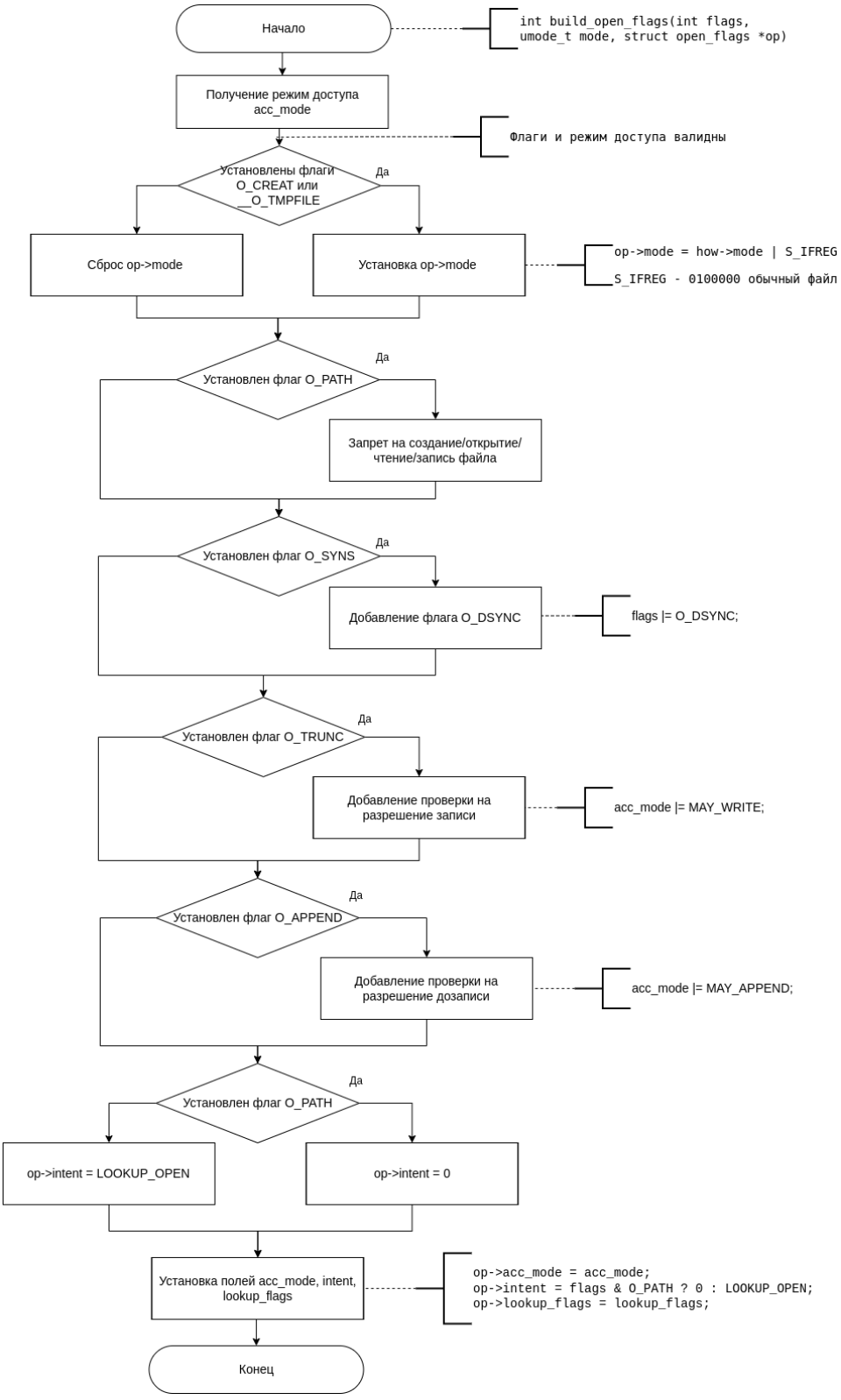


Рисунок 2 – Схема алгоритма build_open_flags()

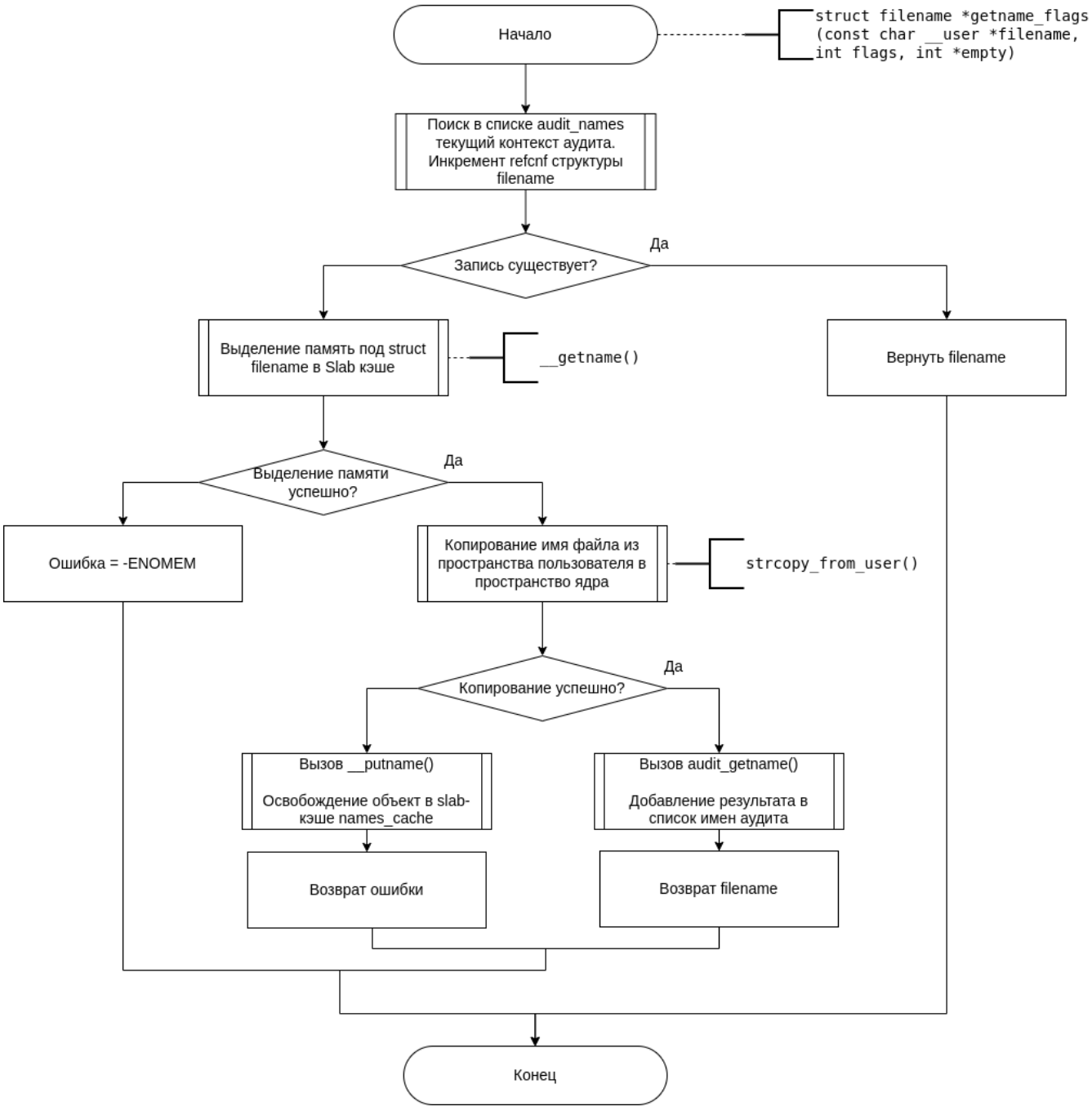


Рисунок 3 – Схема алгоритма getname_flags()

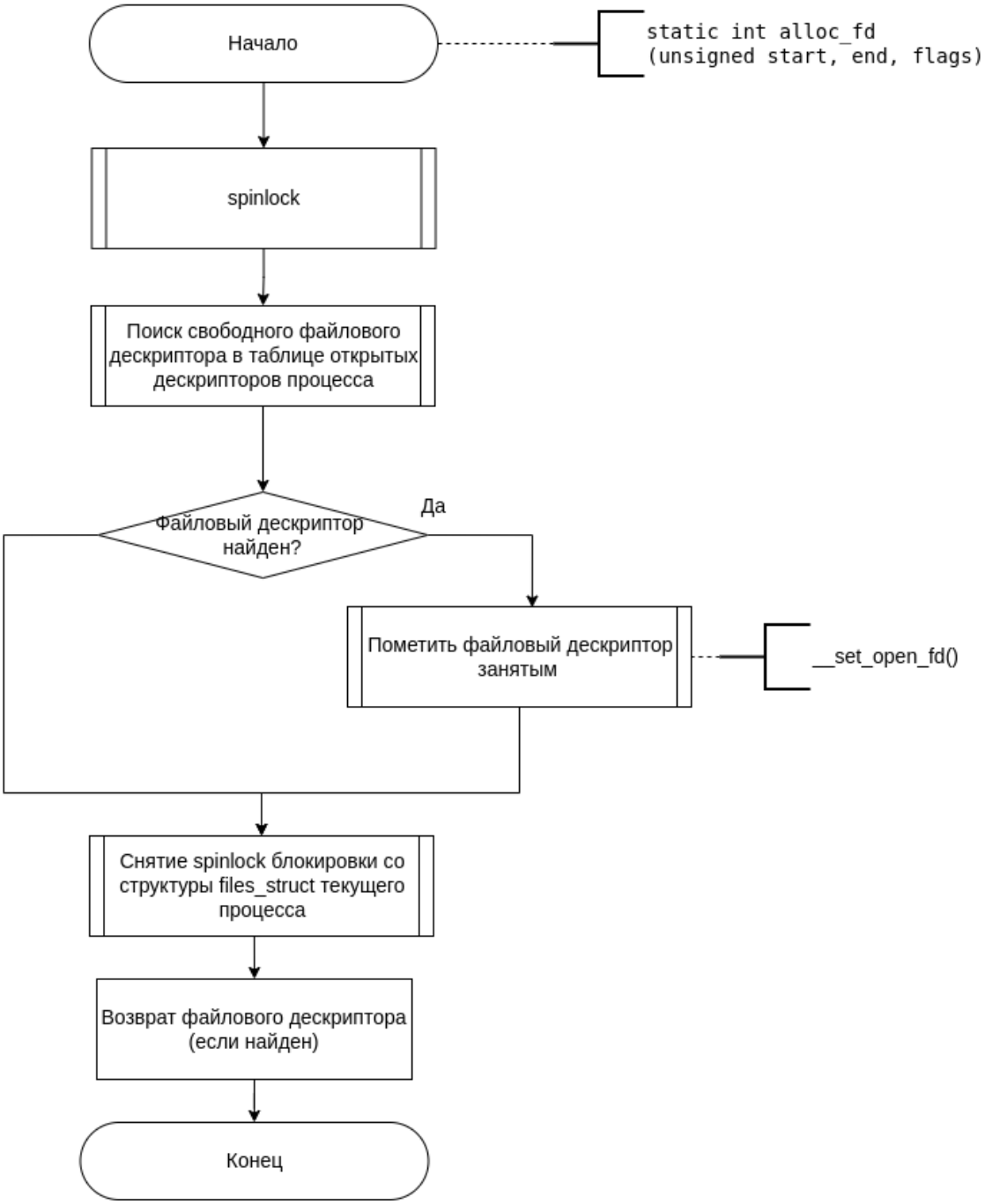


Рисунок 4 – Схема алгоритма alloc_fd()

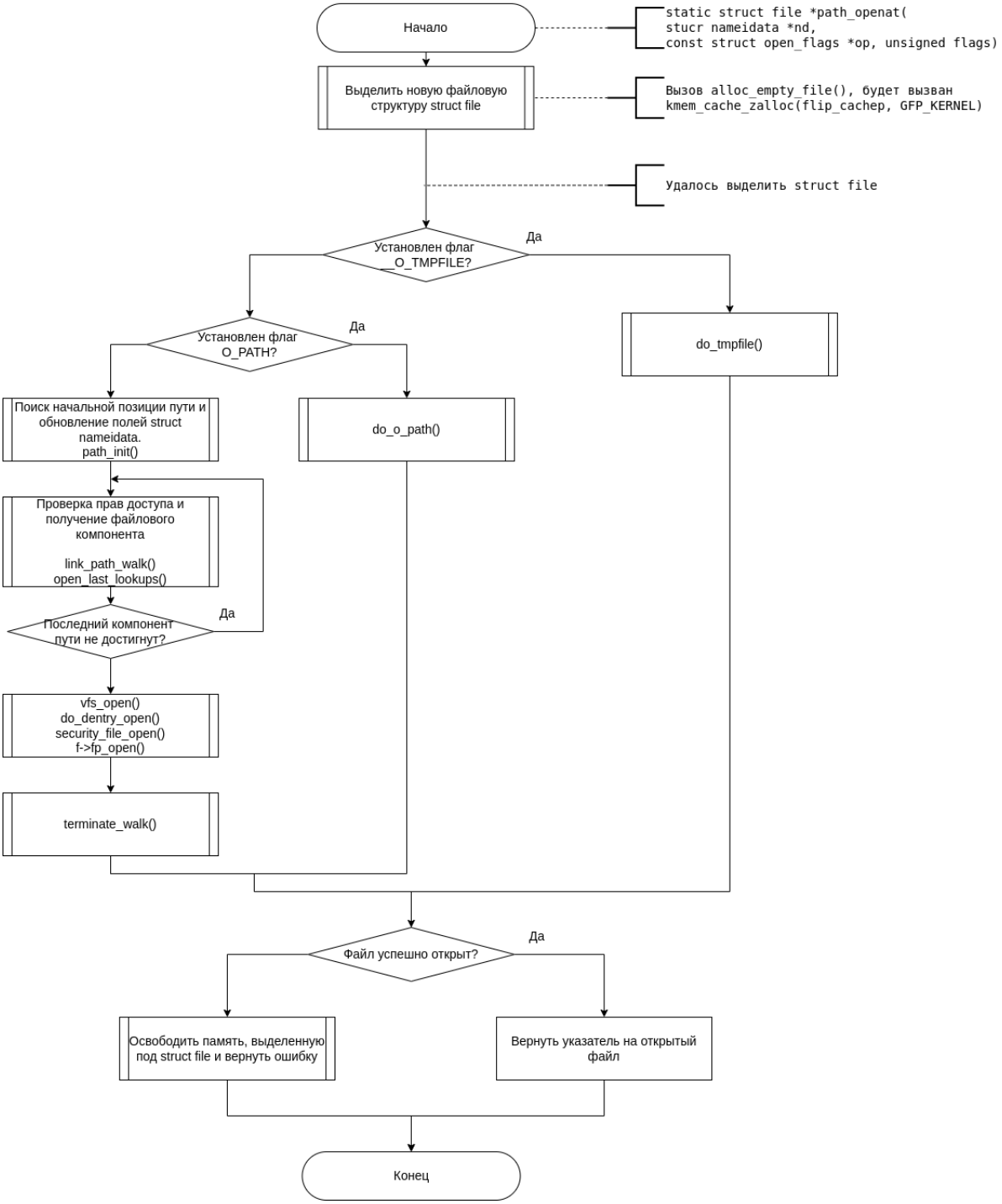


Рисунок 5 – Схема алгоритма path_openat()

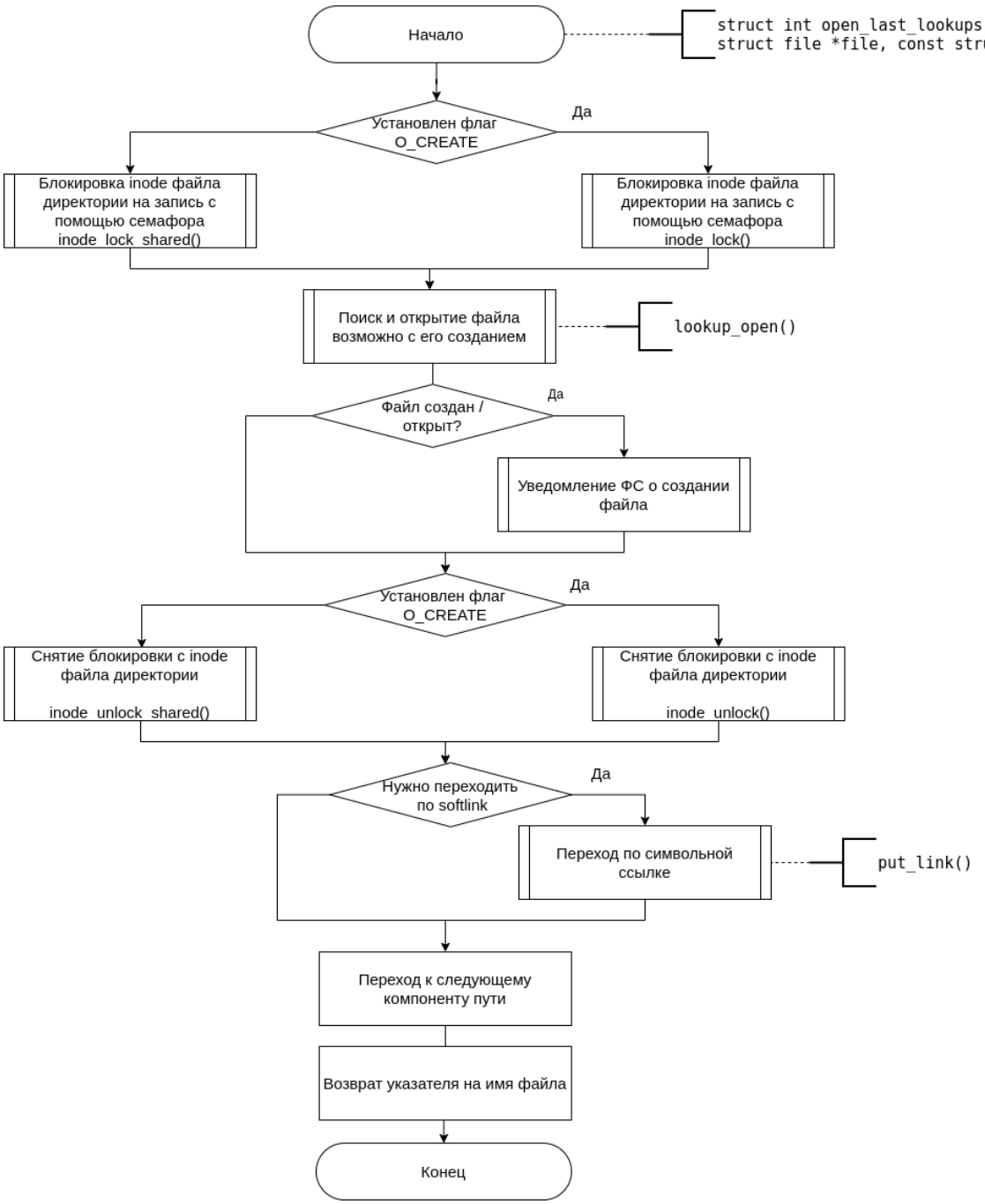


Рисунок 6 – Схема алгоритма open_last_lookups()

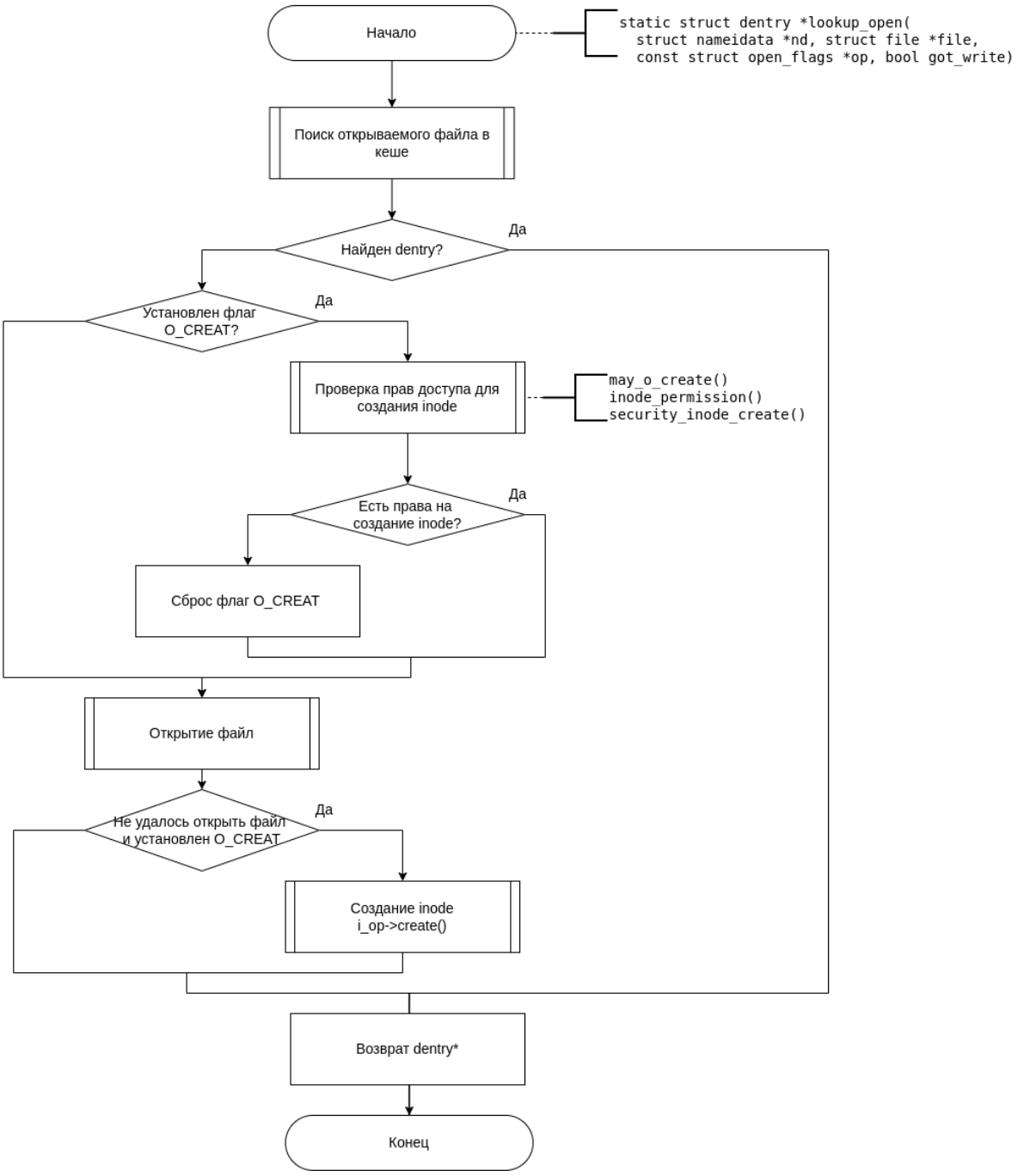


Рисунок 7 – Схема алгоритма lookup_open()

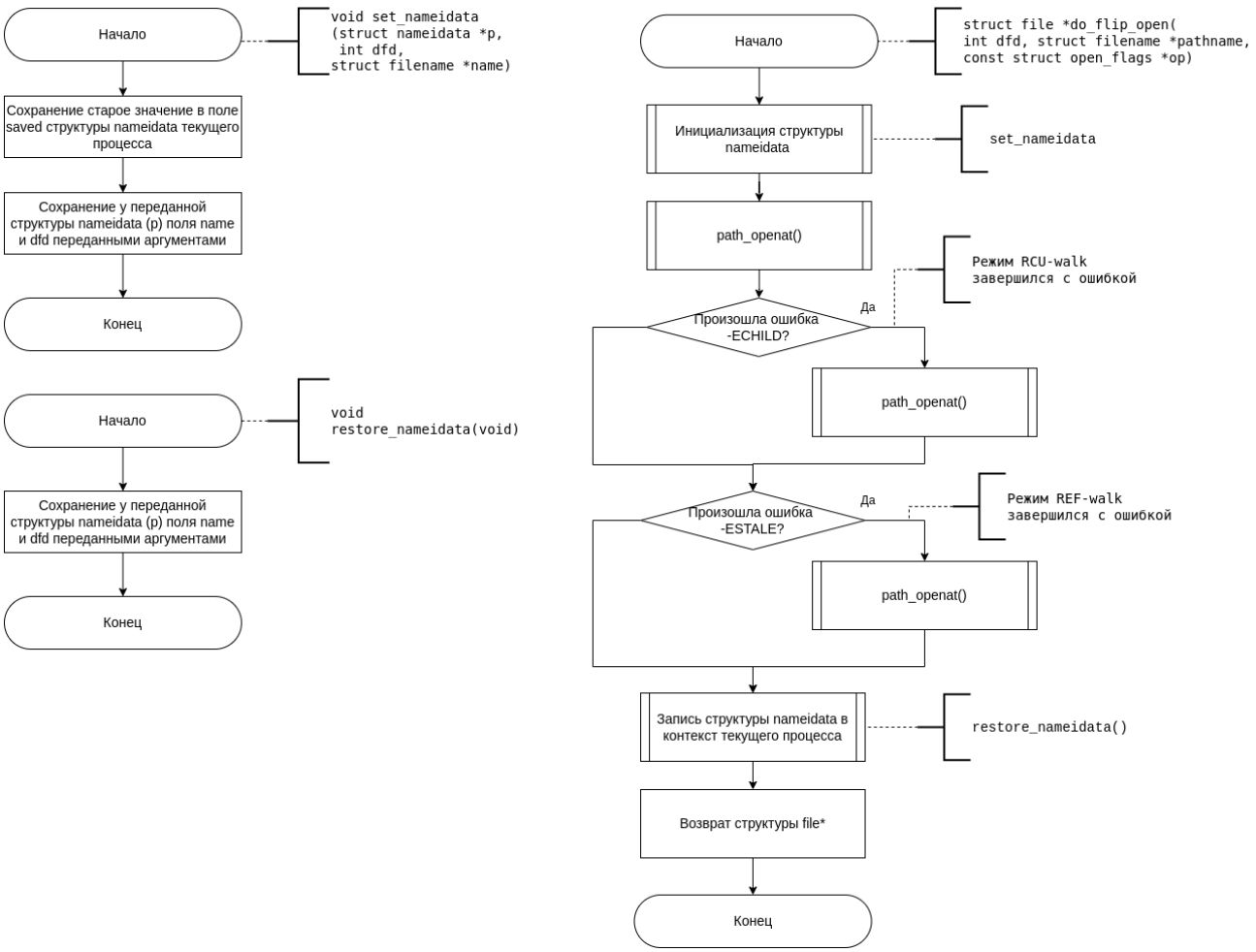


Рисунок 8 – Схема алгоритма nameidata() и do_flip_open()

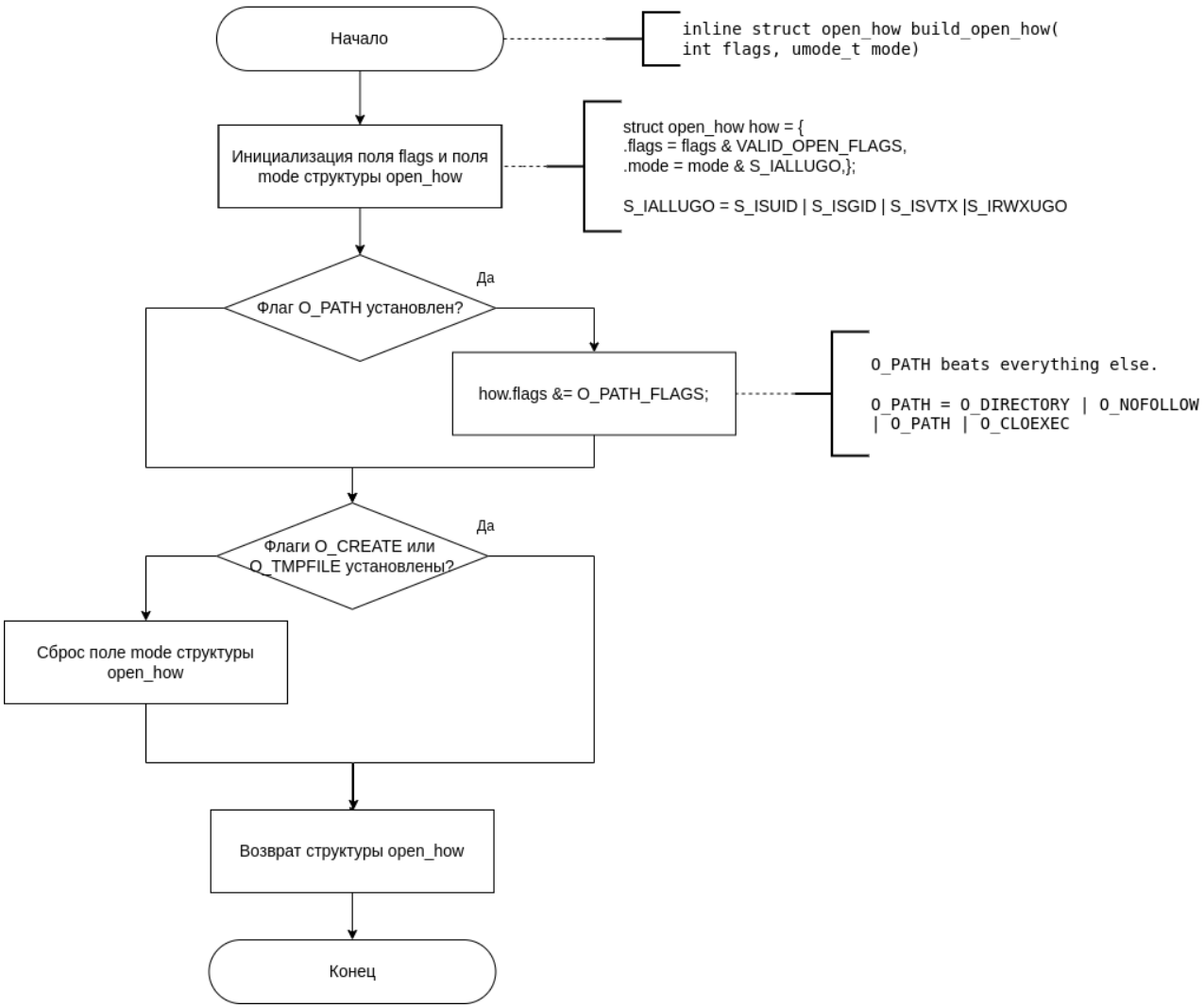


Рисунок 9 – Схема алгоритма build_open_how()

# Дополнение

## Необходимые структуры

```
struct filename {
    const char *name;
    const __user char *uptr;
    int refcnt;
    struct audit_names *aname;
    const char iname[];
};
```

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

```
struct audit_names {
    struct list_head    list;

    struct filename     *name;
    int          name_len;
    bool             hidden;

    unsigned long        ino;
    dev_t            dev;
    umode_t          mode;
    kuid_t           uid;
    kgid_t           gid;
    dev_t            rdev;
    u32          osid;
    struct audit_cap_data   fcap;
    unsigned int         fcap_ver;
    unsigned char        type;
    /* record type */
    /*
        * This was an allocated audit_names and not from the array of
        * names allocated in the task audit context.  Thus this name
        * should be freed on syscall exit.
        */
    bool             should_free;
};
```

```c
#define EMBEDDED_LEVELS 2
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;
```

```c
struct path {
    struct vfsmount *mnt;
    struct dentry *dentry;
} __randomize_layout;
```

```c
struct open_how {
    __u64 flags; // @flags: O_* flags .
    __u64 mode; //@mode : O_CREAT/O_TMPFILE file mode .
    __u64 resolve; //@ resolve : RESOLVE_* flags
};
```

```c
inline struct open_how build_open_how(int flags, umode_t mode)
{
    struct open_how how = {
        .flags = flags & VALID_OPEN_FLAGS,
        .mode = mode & S_IALLUGO,
    };

    /* O_PATH beats everything else. */
    if (how.flags & O_PATH)
        how.flags &= O_PATH_FLAGS;
    /* Modes should only be set for create-like flags. */
    if (!WILL_CREATE(how.flags))
```

```
13          how.mode = 0;
14      return how;
15  }
```

## Флаги системного вызова `open()`

`O_CREAT` — если файл не существует, то он будет создан.

`O_EXCL` — если используется совместно с `O_CREAT`, то при наличии уже созданного файла вызов завершится ошибкой.

`O_APPEND` — файл открывается в режиме добавления, перед каждой операцией записи файловый указатель будет устанавливаться в конец файла.

`O_NOCTTY` — если файл указывает на терминальное устройство, то оно не станет терминалом управления процесса, даже при его отсутствии.

`O_TRUNC` — если файл уже существует, он является обычным файлом и заданный режим позволяет записывать в этот файл, то его длина будет урезана до нуля.

`O_NONBLOCK` — файл открывается, по возможности, в режиме non-blocking, то есть никакие последующие операции над дескриптором файла не заставляют в дальнейшем вызывающий процесс ждать.

`O_RSYNC` — операции записи должны выполняться на том же уровне, что и `O_SYNC`.

`O_DIRECTORY` — если файл не является каталогом, то open вернёт ошибку.

`O_NOFOLLOW` — если файл является символической ссылкой, то open вернёт ошибку.

`O_LARGEFILE` — позволяет открывать файлы, размер которых не может быть представлен типом off_t (long).

`O_TMPFILE` — при наличии данного флага создаётся неименованный временный файл.

`O_CLOEXEC` — включает флаг `close-on-exec` для нового файлового дескриптора, указание этого флага позволяет программе избегать дополнительных операций fcntl `F_SETFD` для установки флага `FD_CLOEXEC`.

`O_NOATIME` — запрет на обновление времени последнего доступа к файлу при его чтении