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»

Лабораторная работа
по дисциплине "Операционные системы"

Тема Системный вызов open()

Студент Светличная А.А.

Группа ИУ7-63Б

Преподаватель Рязанова Н.Ю.

Москва — 2023 г.

1 Системный вызов open()

Системный вызов open() открывает файл, указанный в pathname. Если указанный файл не существует, он может быть создан open(), если указан флаг O_CREAT (необязательно).

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

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

Возвращаемое значение open() — дескриптор файла, небольшое неотрицательное целое число, которое используется в последующих системных вызовах для ссылки на открытый файл.

Параметр pathname — имя файла в файловой системе: полный путь к файлу или сокращенное имя.

Параметр mode всегда должен быть указан при использовании O_CREAT, во всех остальных случаях этот параметр игнорируется.

Параметр flags — флаги, которые собираются с помощью побитовой операции ИЛИ из таких значений, как:

- 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_TMPFILE — при наличии данного флага создаётся неименованный временный файл
- O_APPEND — файл открывается в режиме добавления, перед каждой операцией записи файловый указатель будет устанавливаться в конец файла
- O_PATH — получить файловый дескриптор, который можно использовать для двух целей: для указания положения в дереве файловой системы и для выполнения операций, работающих исключительно на уровне файловых дескрипторов. Если O_PATH указан, то биты флагов, отличные от

O_CLOEXEC, O_DIRECTORY и O_NOFOLLOW, игнорируются (когда этот флаг установлен, будет возвращен дескриптор struct file, при этом сам файл не открывается)

- O_CLOEXEC — устанавливает 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_DIRECTORY — если файл не является каталогом, то open вернёт ошибку
- O_NOFOLLOW — если файл является символической ссылкой, то open вернёт ошибку
- O_EXEC — открыть только для выполнения (результат не определен при открытии директории)
- O_RDONLY — открыть только на чтение
- O_RDWR — открыть на чтение и запись
- O_SEARCH — открыть директорию только для поиска (результат не определен при использовании с файлами, не являющимися директорией)
- O_WRONLY — открыть только на запись
- O_DSYNC — файл открывается в режиме синхронного ввода-вывода (все операции записи для соответствующего дескриптора файла блокируют вызывающий процесс до тех пор, пока данные не будут физически записаны)
- O_NOCTTY — если файл указывает на терминальное устройство, то оно не станет терминалом управления процесса, даже при его отсутствии
- 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_SYNC — файл открывается в режиме синхронного ввода-вывода (все операции записи для соответствующего дескриптора файла блокируют вызывающий процесс до тех пор, пока данные не будут физически записаны)

- 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_LARGEFILE — позволяет открывать файлы, размер которых не может быть представлен типом off_t (long). Для установки должен быть указан макрос _LARGEFILE64_SOURCE

2 Используемые структуры

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

Листинг 2.1 – Структура open_flags

```
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[];
};

struct audit_names {
    struct list_head    list  /* audit_context->names_list */
    struct filename     *name
    int                 name_len /* number of chars to log */
    bool                hidden  /* don't log this record */
    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
}
```

Листинг 2.2 – Структуры filename и audit_names

```
struct nameidata {
    struct path path;
    struct qstr last;
    struct path root;
```

```
    struct inode    *inode; /* path.dentry.d_inode */
    unsigned int    flags, state;
    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;
```

Листинг 2.3 – Структура nameidata

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

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

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))
        how.mode = 0;
    return how;
}
```

Листинг 2.4 – Структуры path и open_how

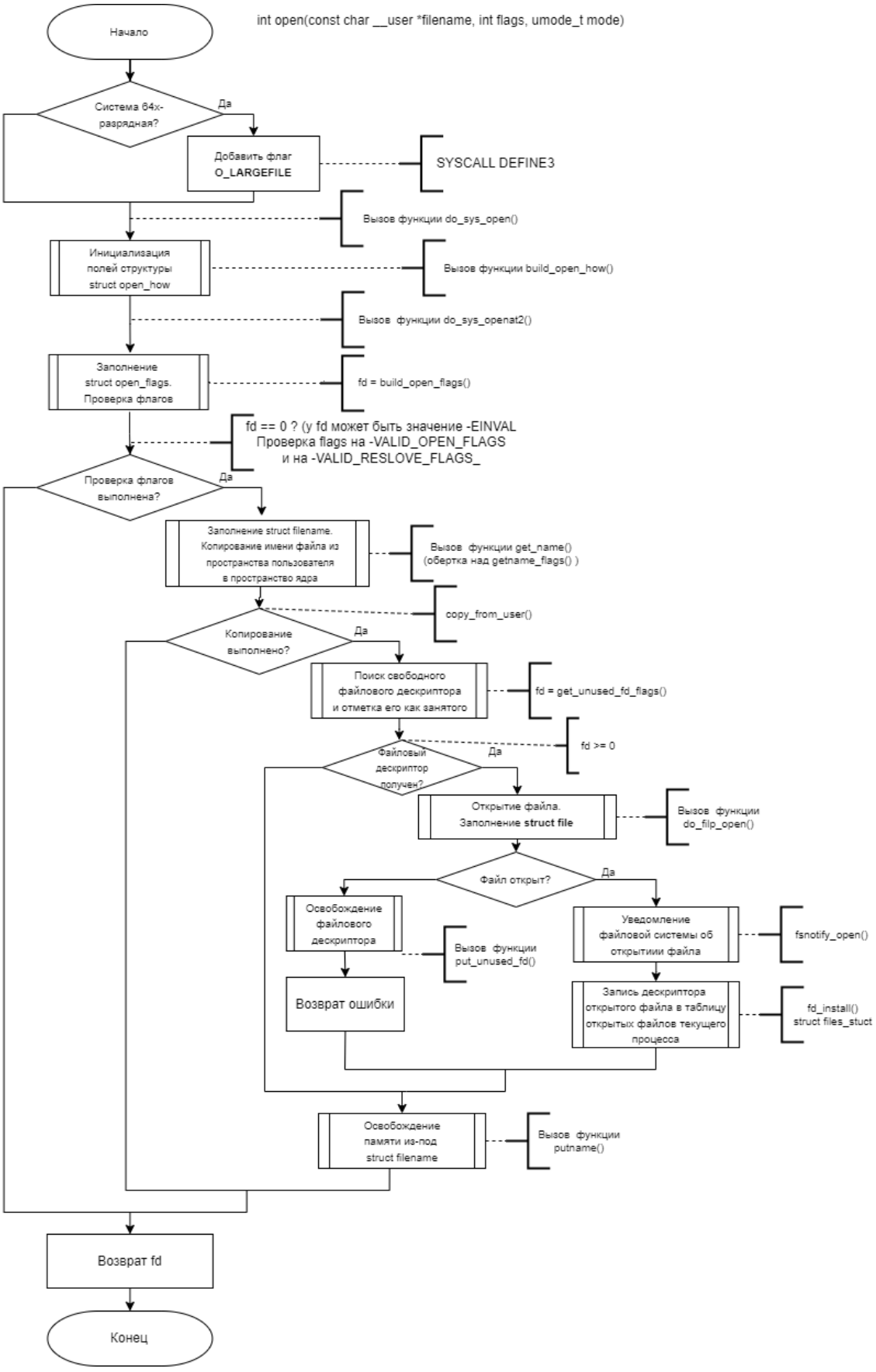

Рисунок 2.1 – Схема алгоритма работы системного вызова open()

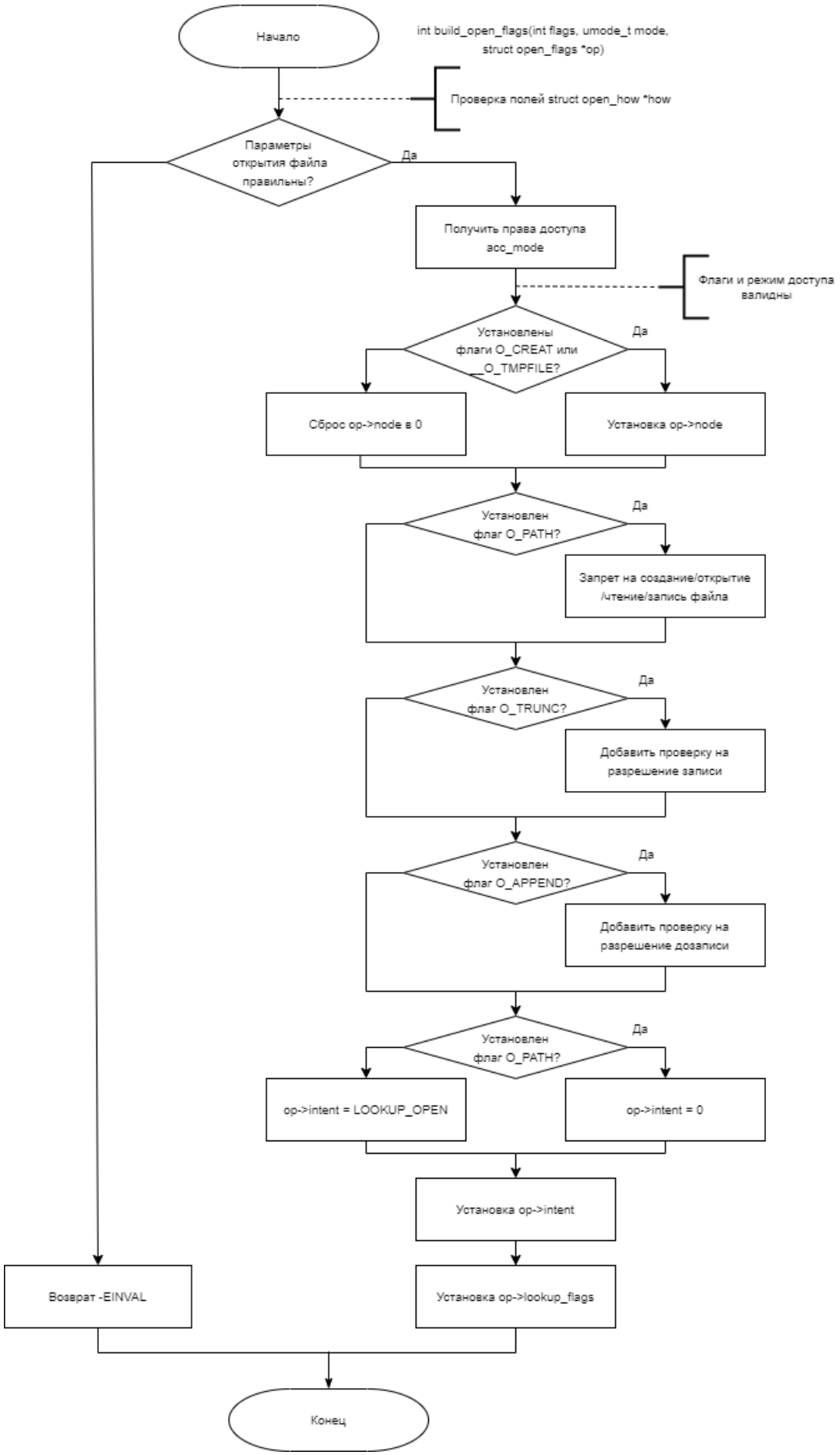

Рисунок 2.2 – Схема алгоритма работы функции build_open_flags()

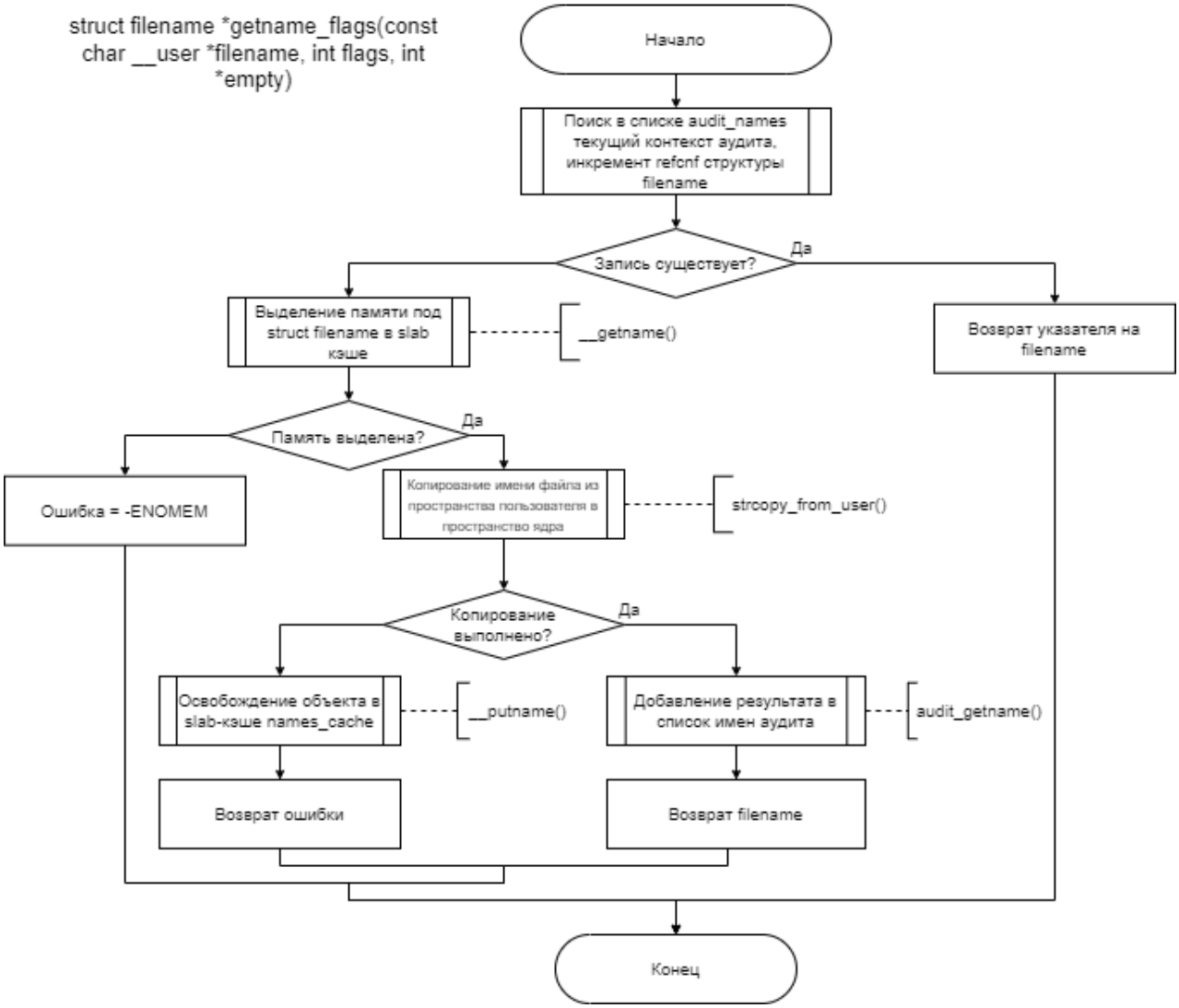

Рисунок 2.3 – Схема алгоритма работы функции getname_flags()

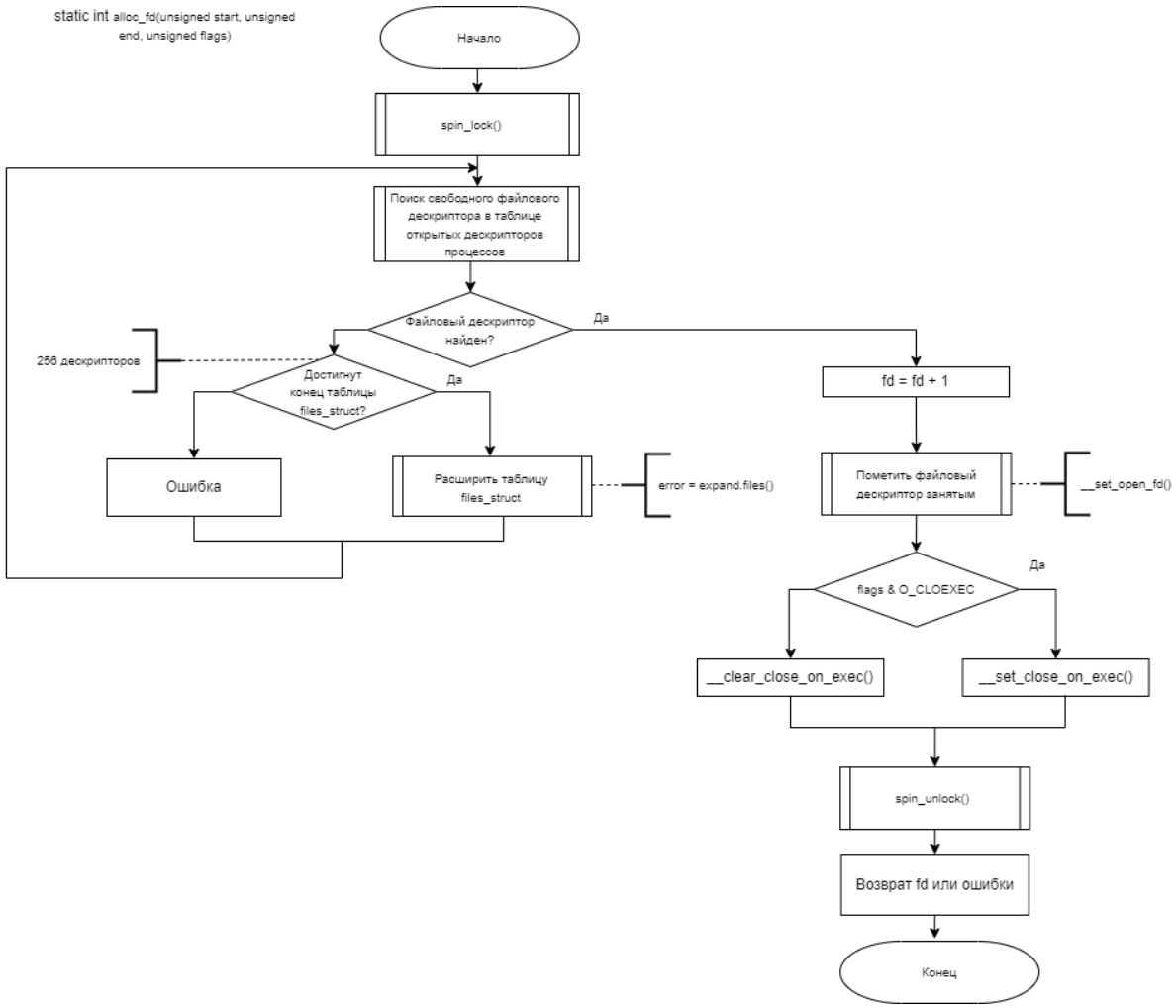

Рисунок 2.4 – Схема алгоритма работы функции alloc_fd()

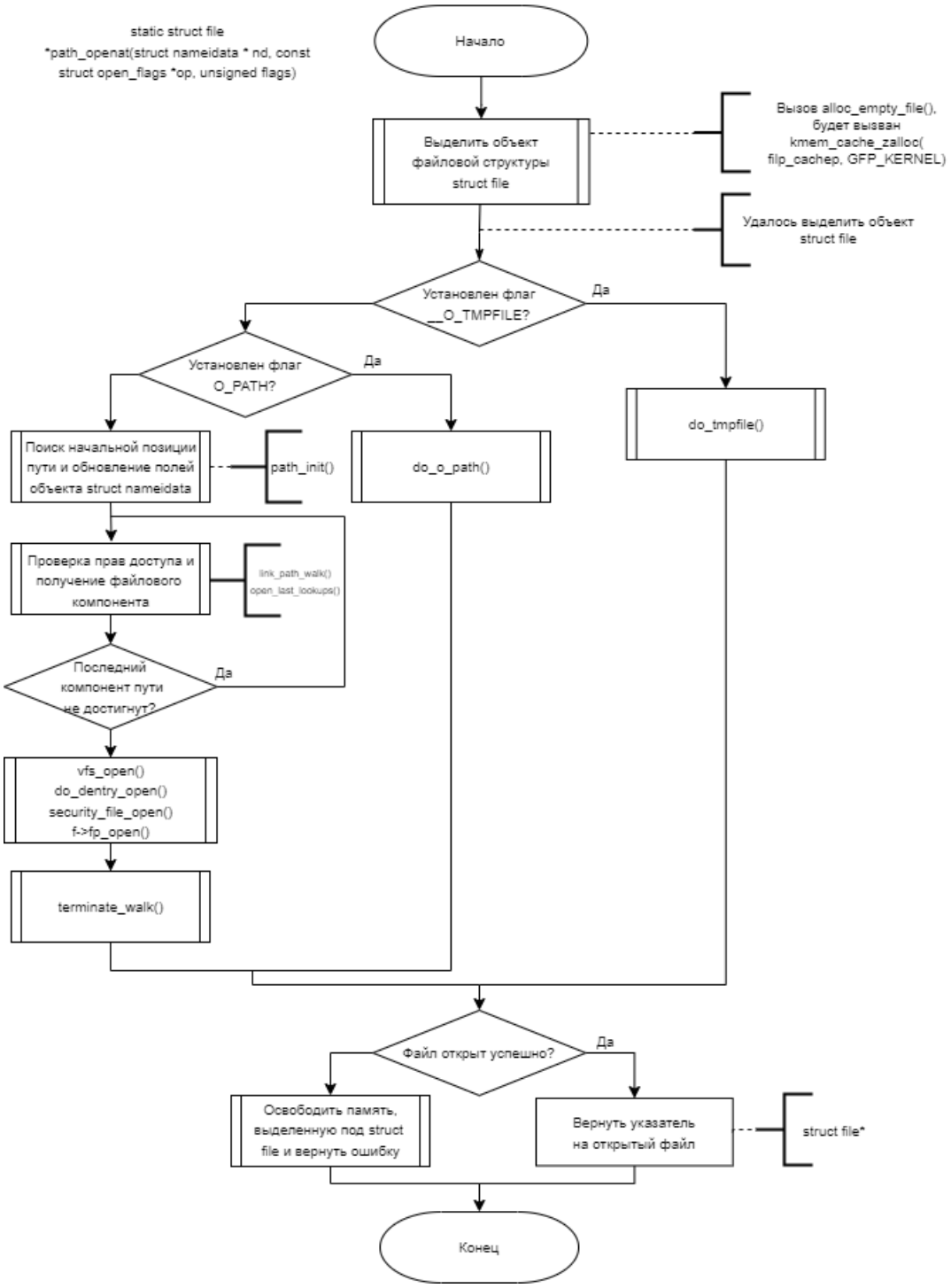

Рисунок 2.5 – Схема алгоритма функции path_openat()

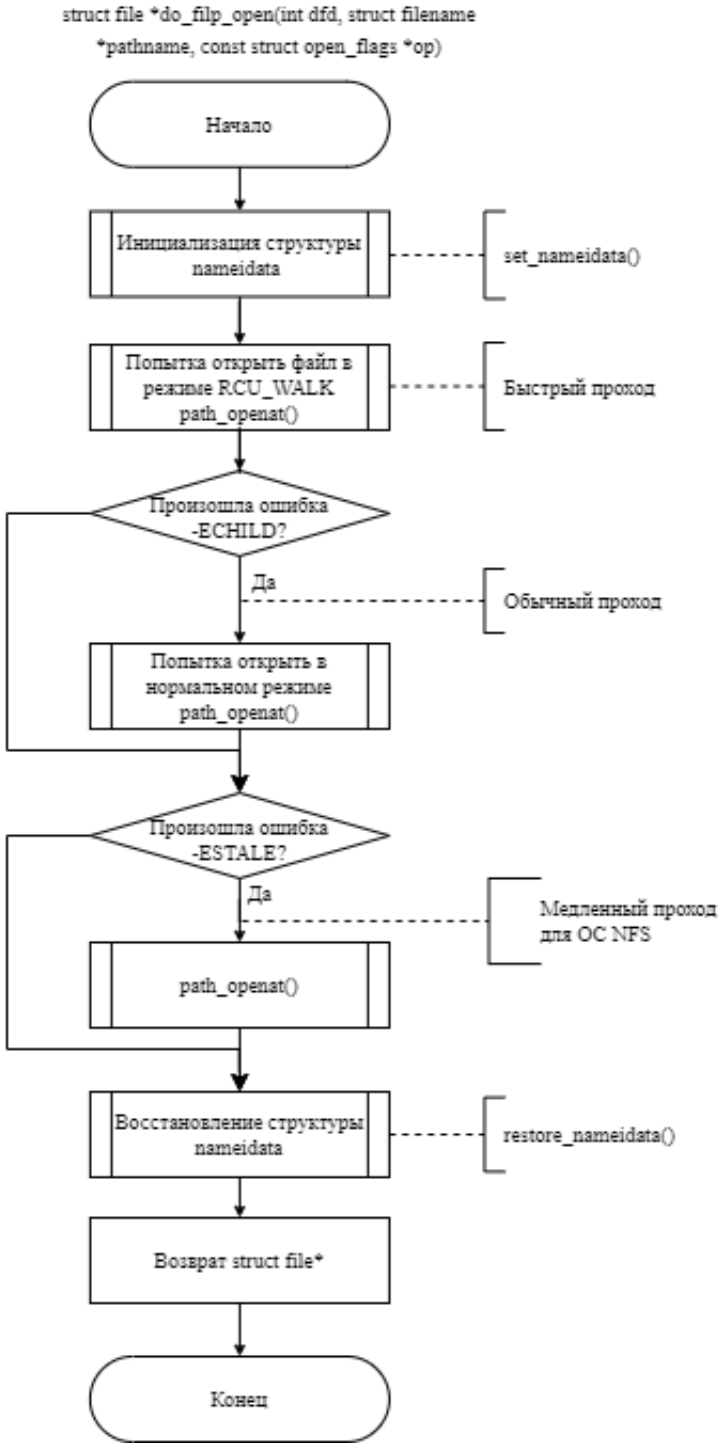

Рисунок 2.6 – Схема алгоритмов функций, работающих с nameidata (do filp open)

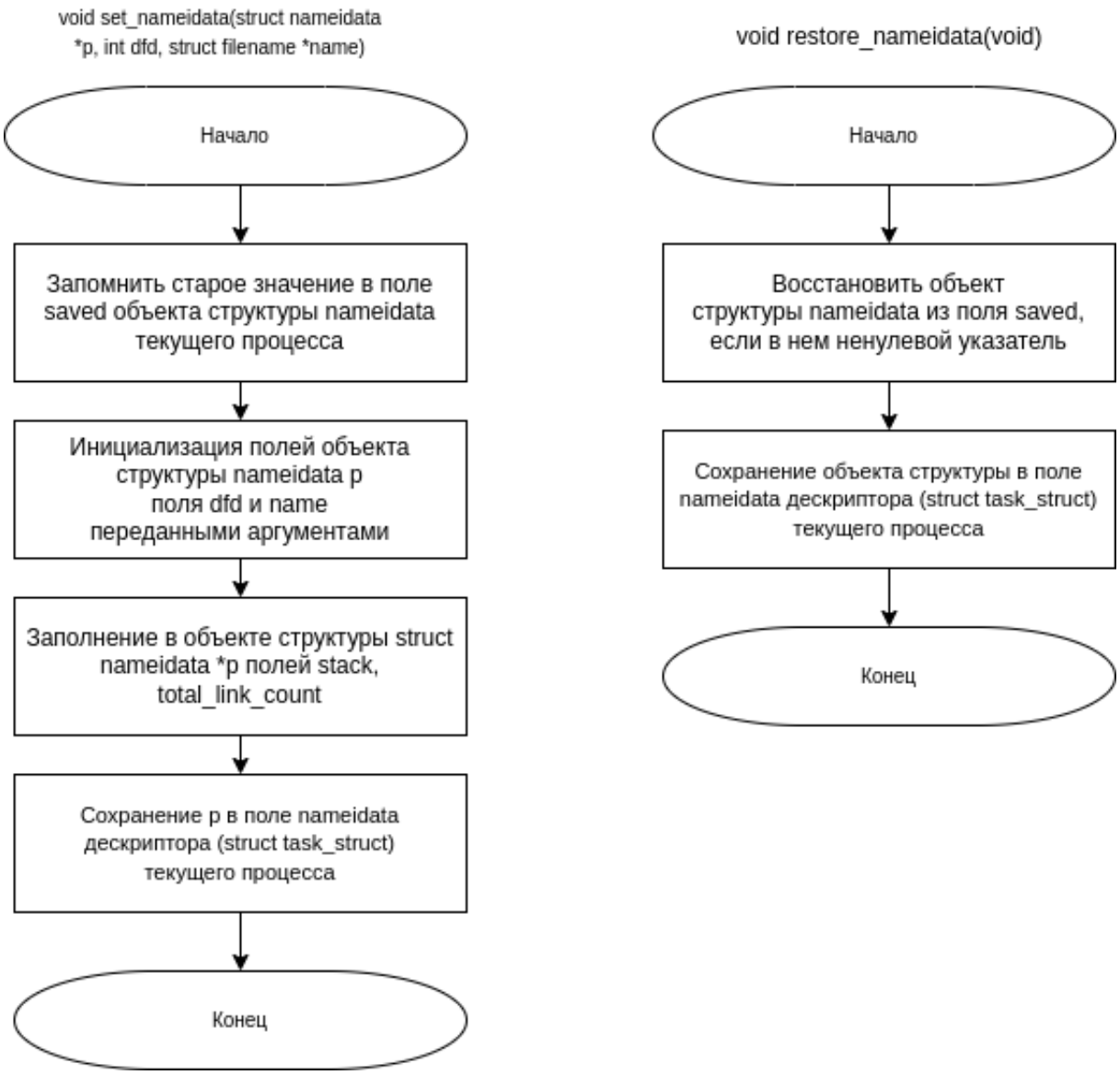

Рисунок 2.7 – Схема алгоритмов функций, работающих с nameidata

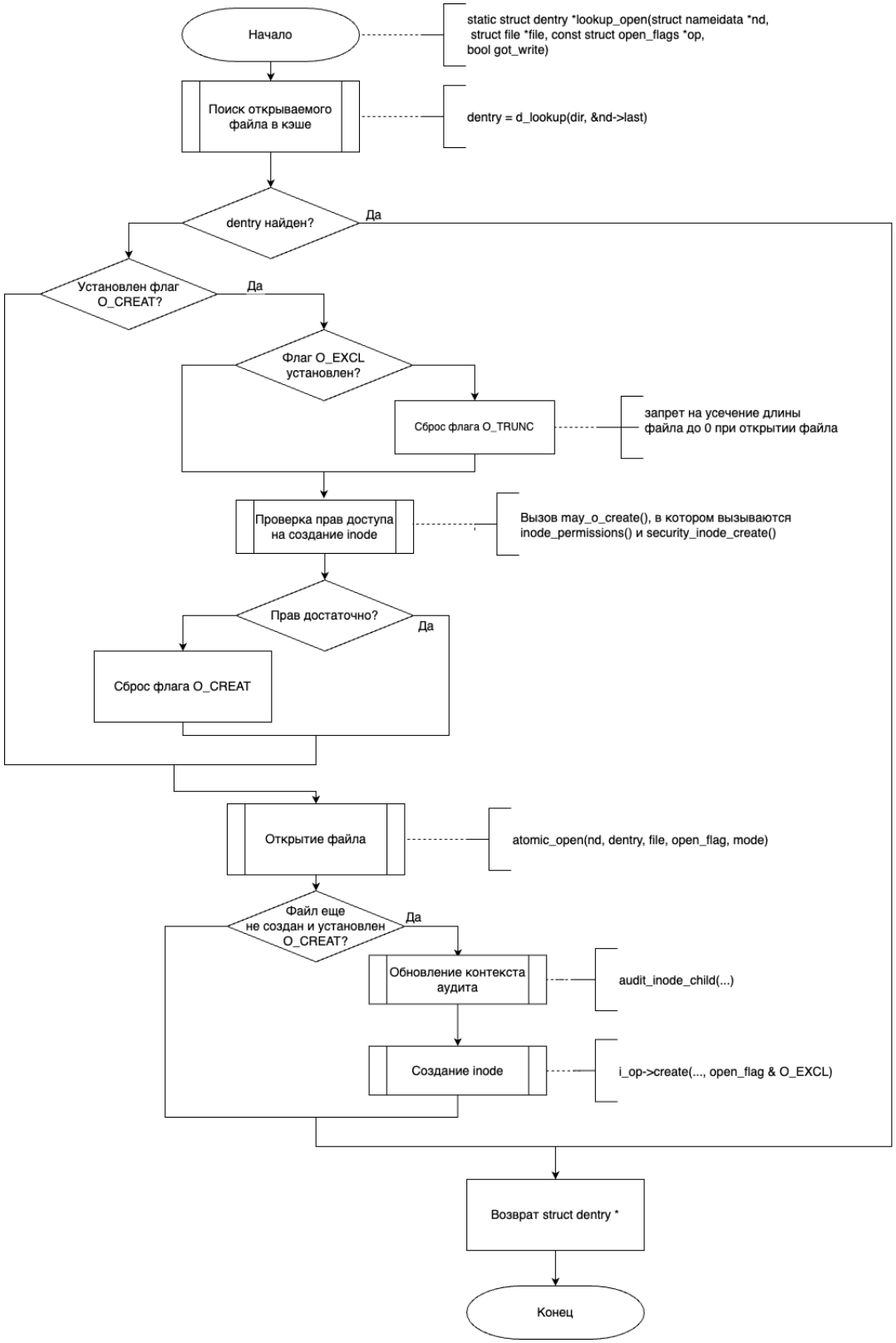

Рисунок 2.8 – Схема алгоритма функции open_last_lookups()

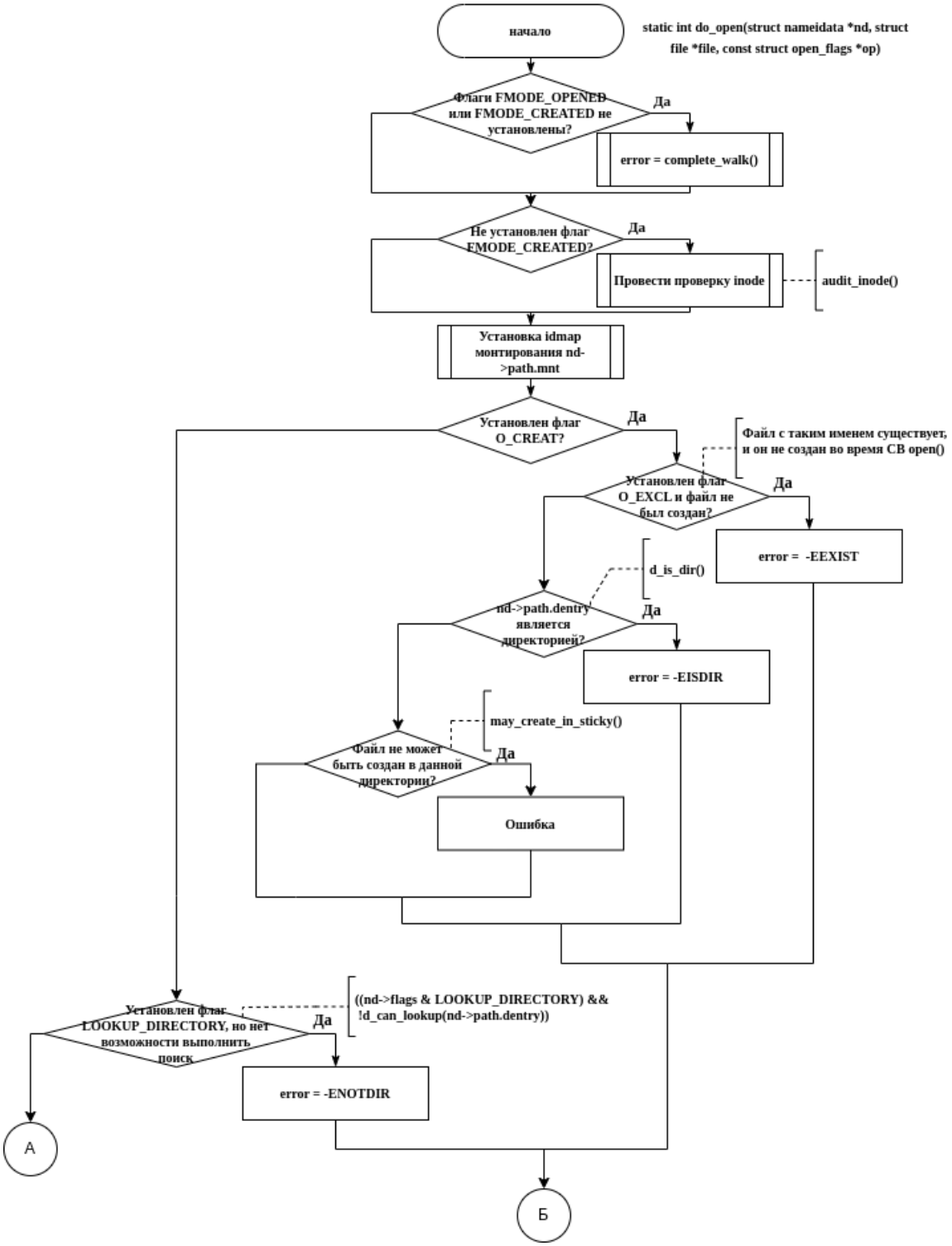

Рисунок 2.9 – Схема алгоритма функции do_open() начало

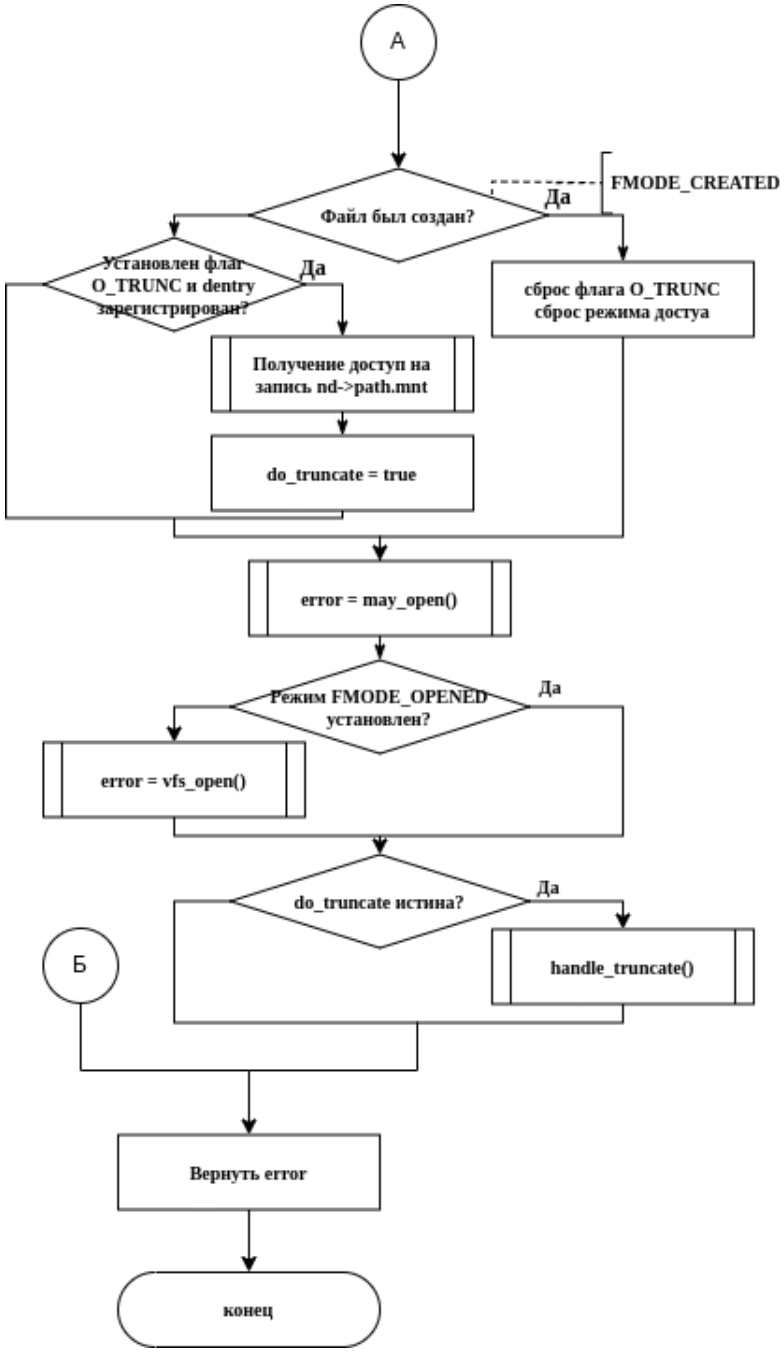

Рисунок 2.10 – Схема алгоритма функции do_open() продолжение

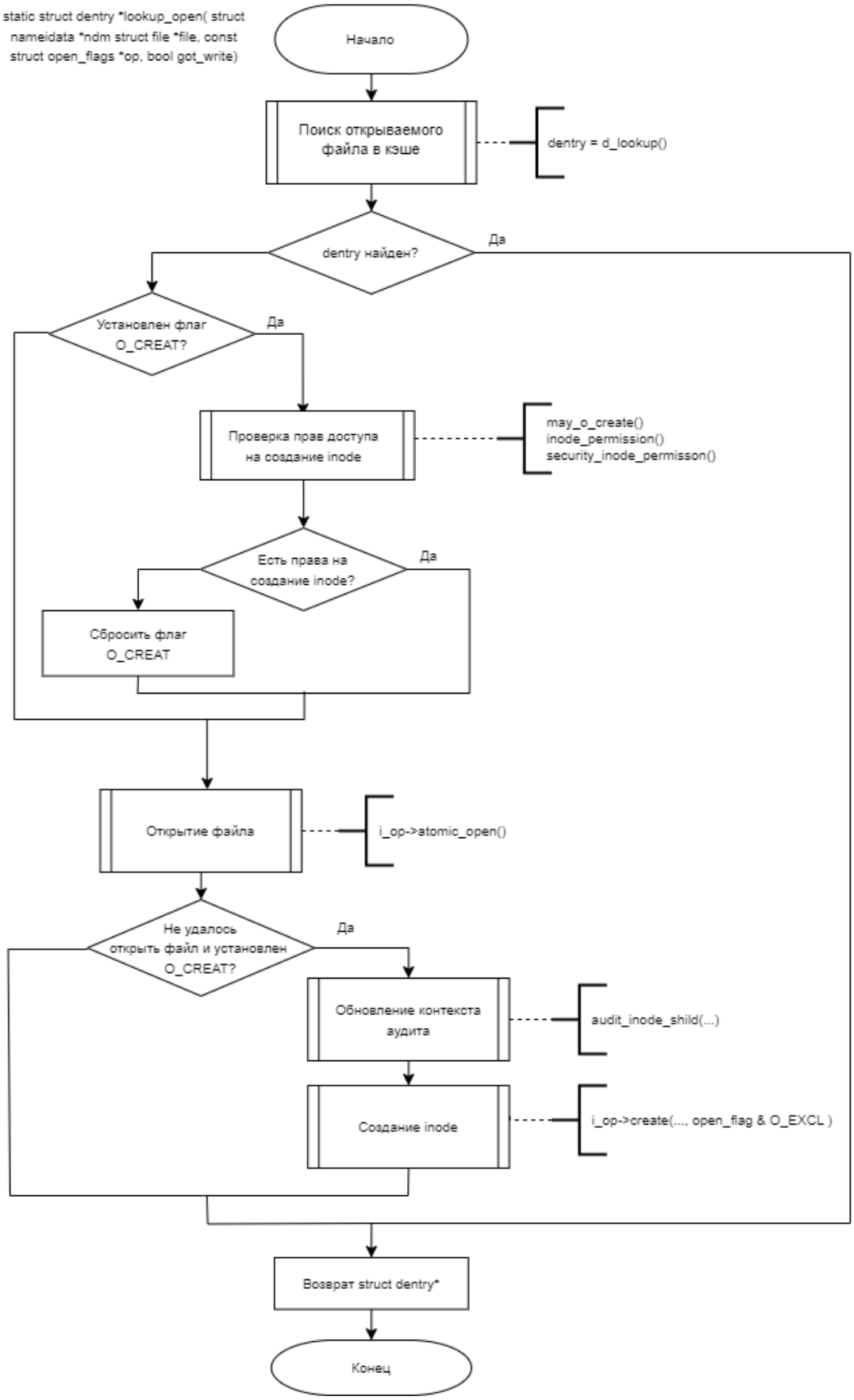

Рисунок 2.11 – Схема алгоритма функции open_lookup()

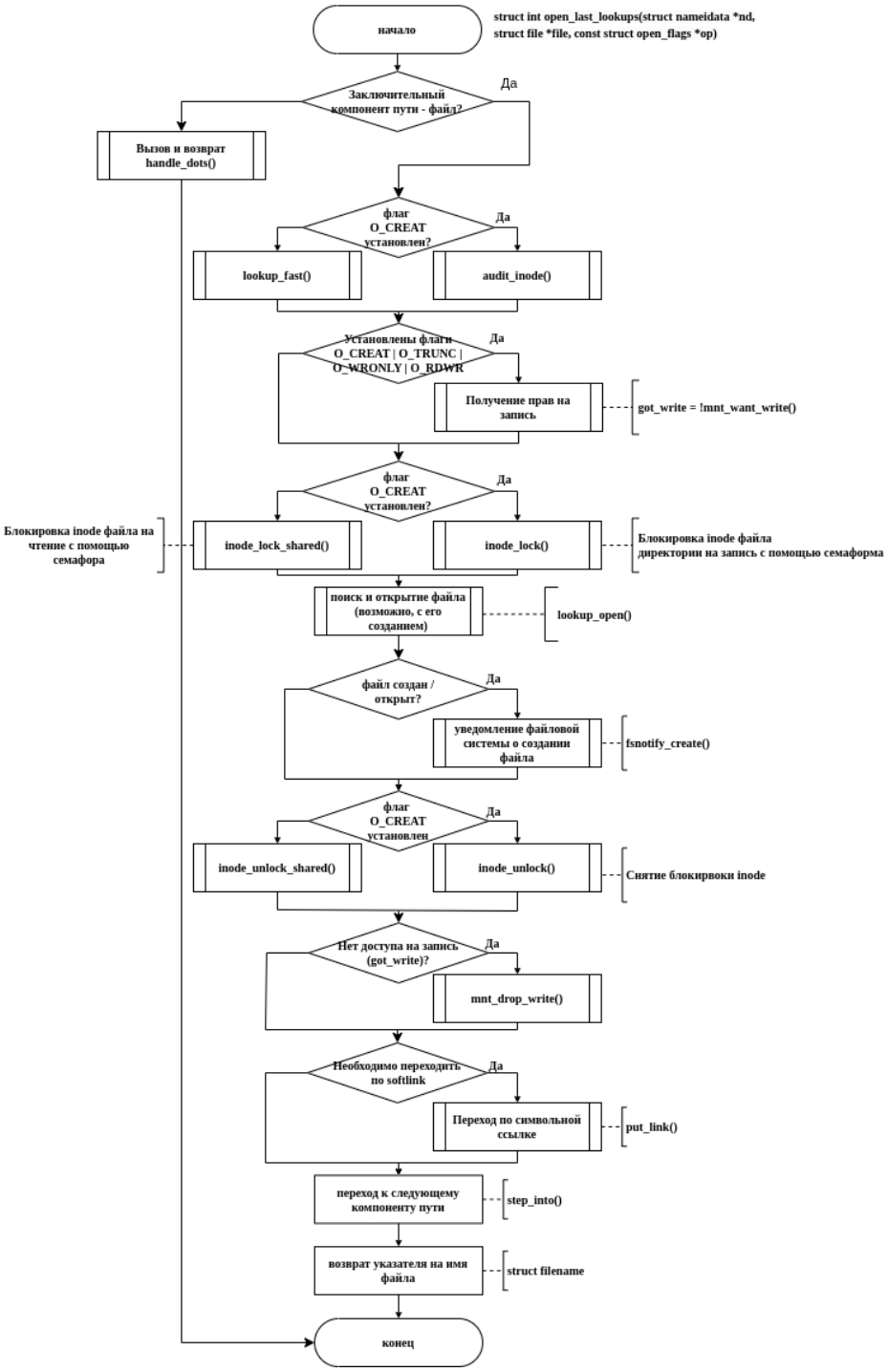

Рисунок 2.12 – Схема алгоритма функции last_lookup()