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, искусственный интеллект и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

Отчет по лабораторной работе №9

по курсу «Операционные системы»

«Системный вызов open()»

Студент группы **ИУ7-64Б** _____ **Д. С. Чепиго**

(Подпись, дата) (И.О. Фамилия)

Преподаватели _____ **Н. Ю. Рязанова**

(Подпись, дата) (И.О. Фамилия)

***2023** г.*

1 Системный вызов open()

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

```
#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, неотрицательное целое число, которое используется в последующих системных вызовах для работы с файлом.

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

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

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_CREAT — если файл не существует, то он будет создан.

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

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

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

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

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

O_TMPFILE — при наличии данного флага создаётся неименованный временный файл.

O_PATH — получить файловый дескриптор, который можно использо-

вать 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.

Третий параметр mode всегда должен быть указан при использовании *O_CREAT*; во всех остальных случаях этот параметр игнорируется.

2 Необходимые структуры

Версия ядра: 5.15.32

Листинг 1 – struct open_flags

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

Листинг 2 – struct filename и struct audit_names

```
struct audit_names;

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 3 – struct nameidata

```
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;
```

```
18      struct nameidata     *saved;
19      unsigned     root_   seq;
20      int                  dfd;
21      kuid_t               dir_uid;
22      umode_t              dir_mode;
23  } __randomize_layout;
```

Листинг 4 – struct path и struct open_how

```
1   struct path {
2       struct vfsmount *mnt;
3       struct dentry *dentry;
4   } __randomize_layout;
5
6   struct open_how {
7       __u64 flags;
8       __u64 mode;
9       __u64 resolve;
10  };
11
12  inline struct open_how build_open_how(int flags, umode_t mode)
13  {
14      struct open_how how = {
15          .flags = flags & VALID_OPEN_FLAGS,
16          .mode = mode & S_IALLUGO,};
17      /* O_PATH beats everything else. */
18      if (how.flags & O_PATH)
19          how.flags &= O_PATH_FLAGS;
20      /* Modes should only be set for create-like flags. */
21      if (!WILL_CREATE(how.flags))
22          how.mode = 0;
23      return how;
24  }
```

3 Схемы алгоритмов

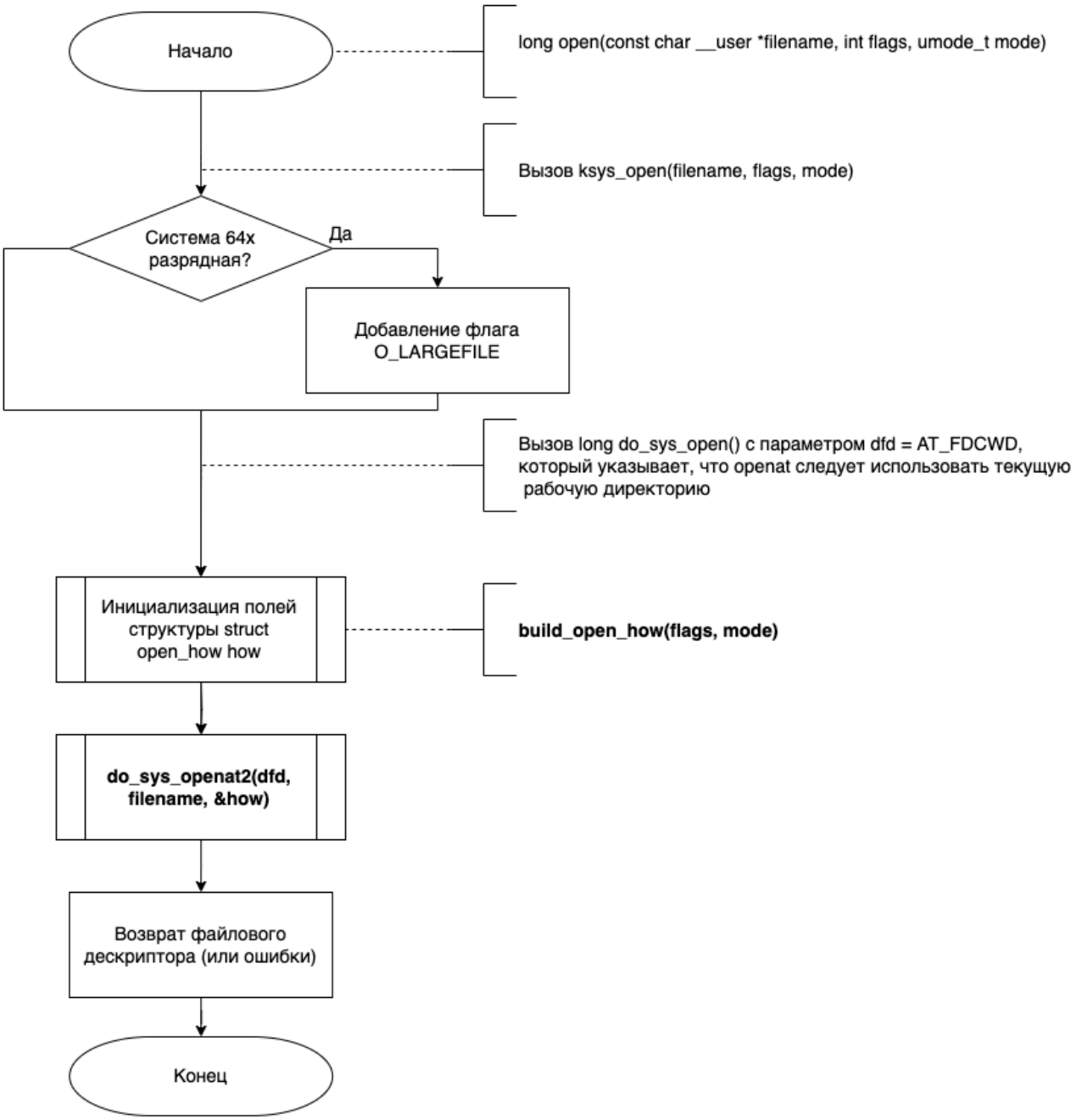

Рисунок 1 – Схема алгоритма работы системного вызова open()

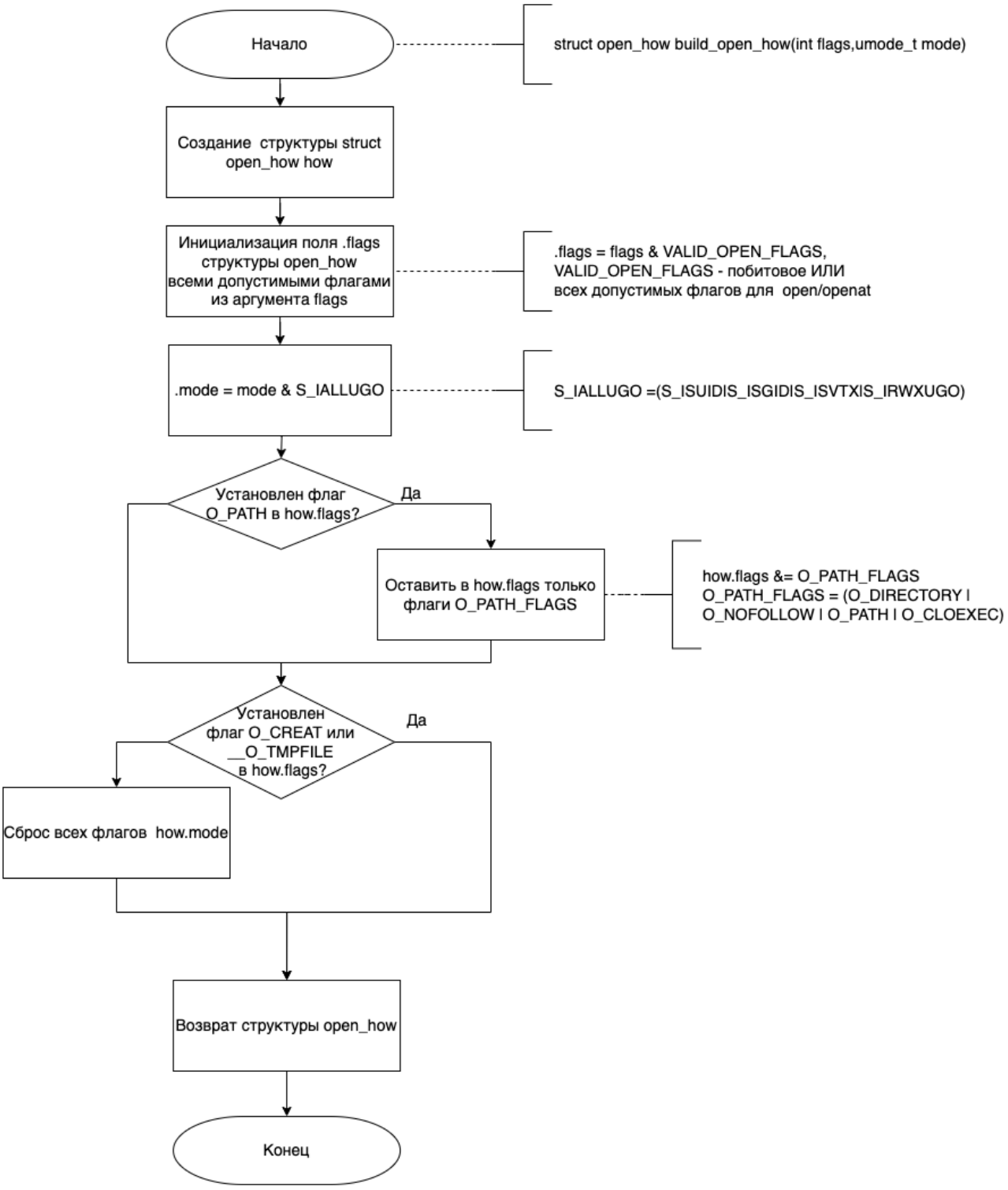

Рисунок 2 – Схема алгоритма работы функции build_open_how()

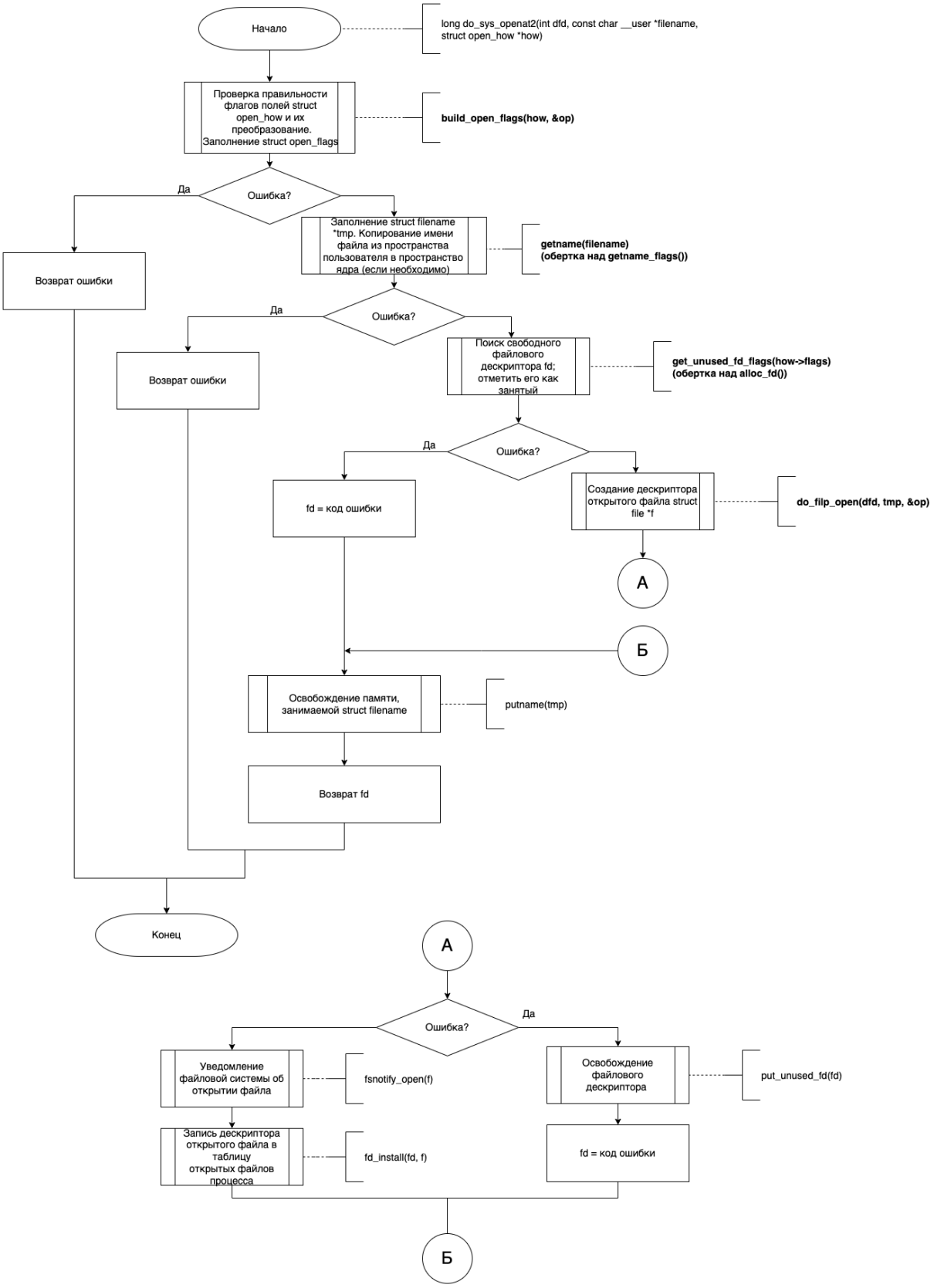

Рисунок 3 – Схема алгоритма работы функции do_sys_open_at2()

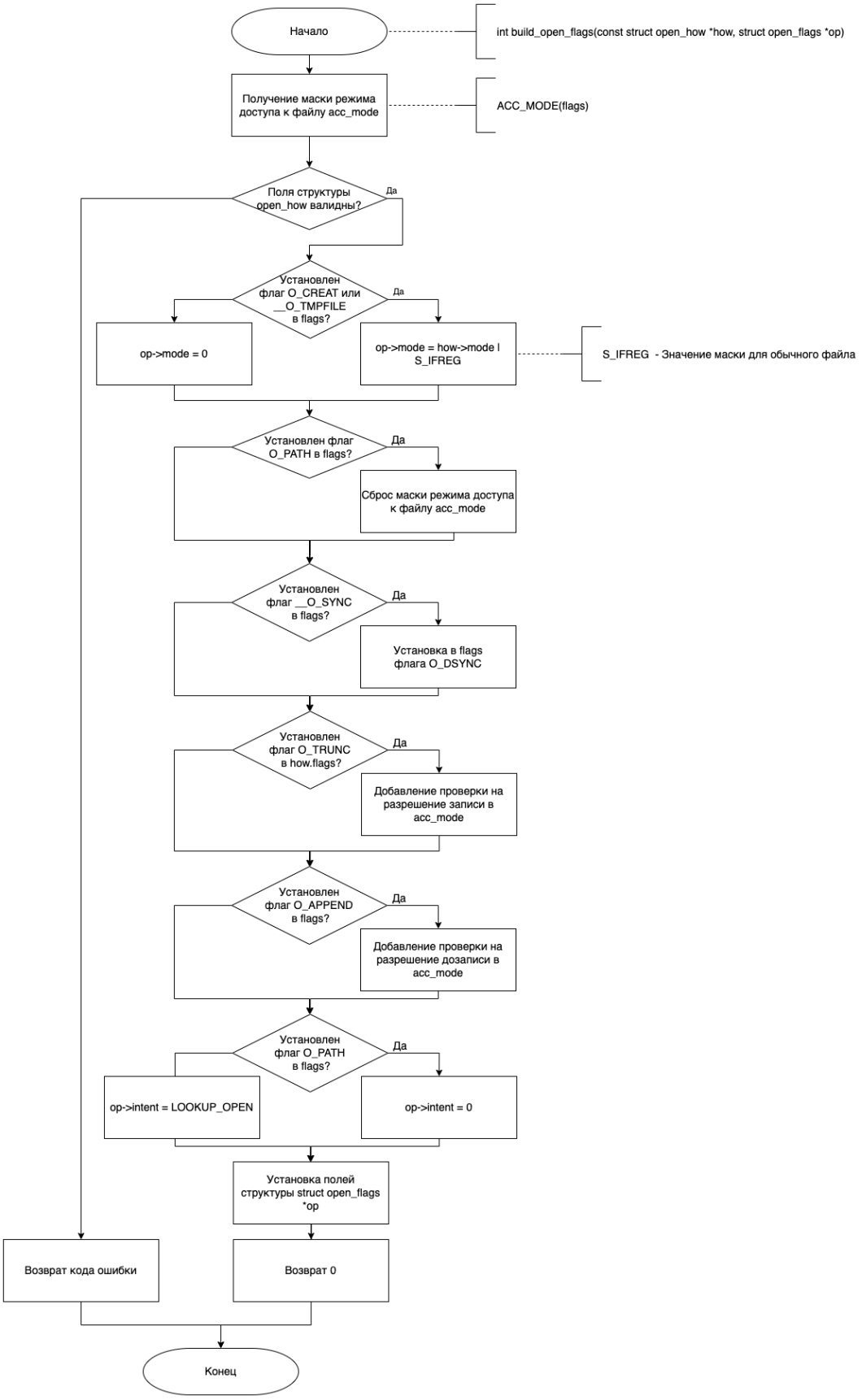

Рисунок 4 – Схема алгоритма работы функции build_open_flags()

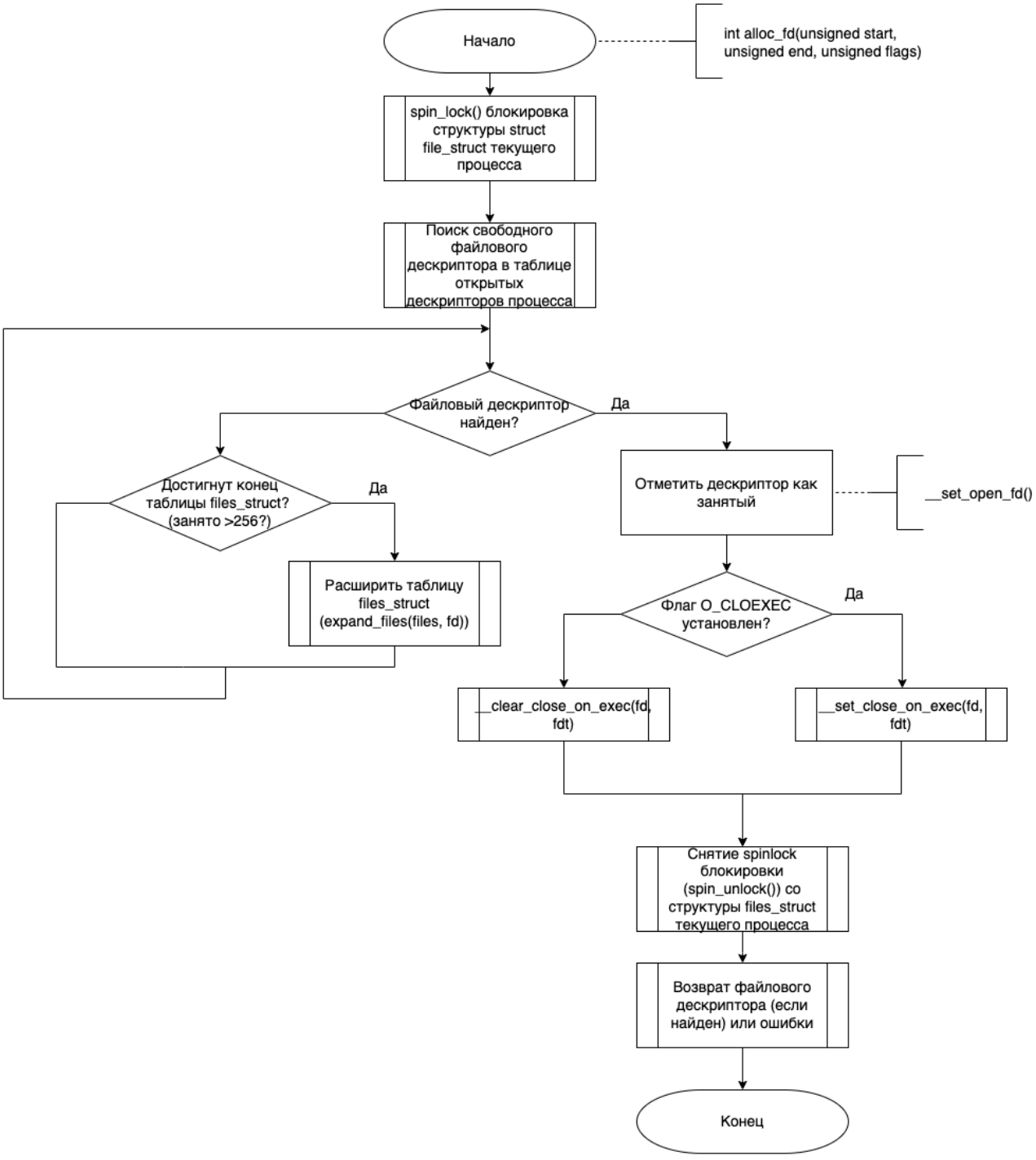

Рисунок 5 – Схема алгоритма работы функции alloc_fd()

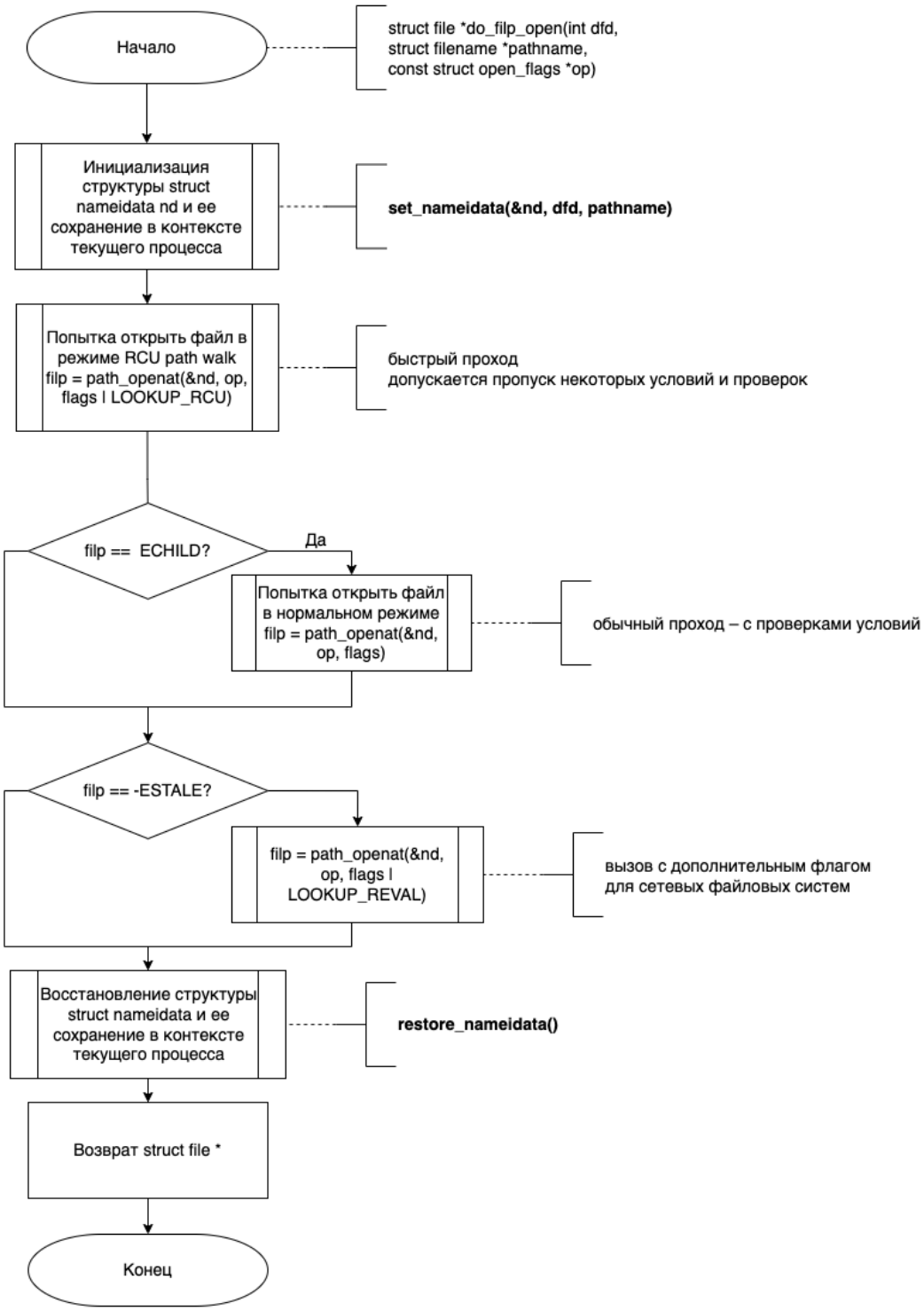

Рисунок 6 – Схема алгоритма работы функции do_filp_open()

LOOKUP_RCU – флаг используется в системе VFS для указания , что операция поиска должна выполняться с использованием RCU (Read-Copy-Update).

LOOKUP_REVAL – флаг для работы с NFS, указывает, что необходимо выполнить повторную проверку.

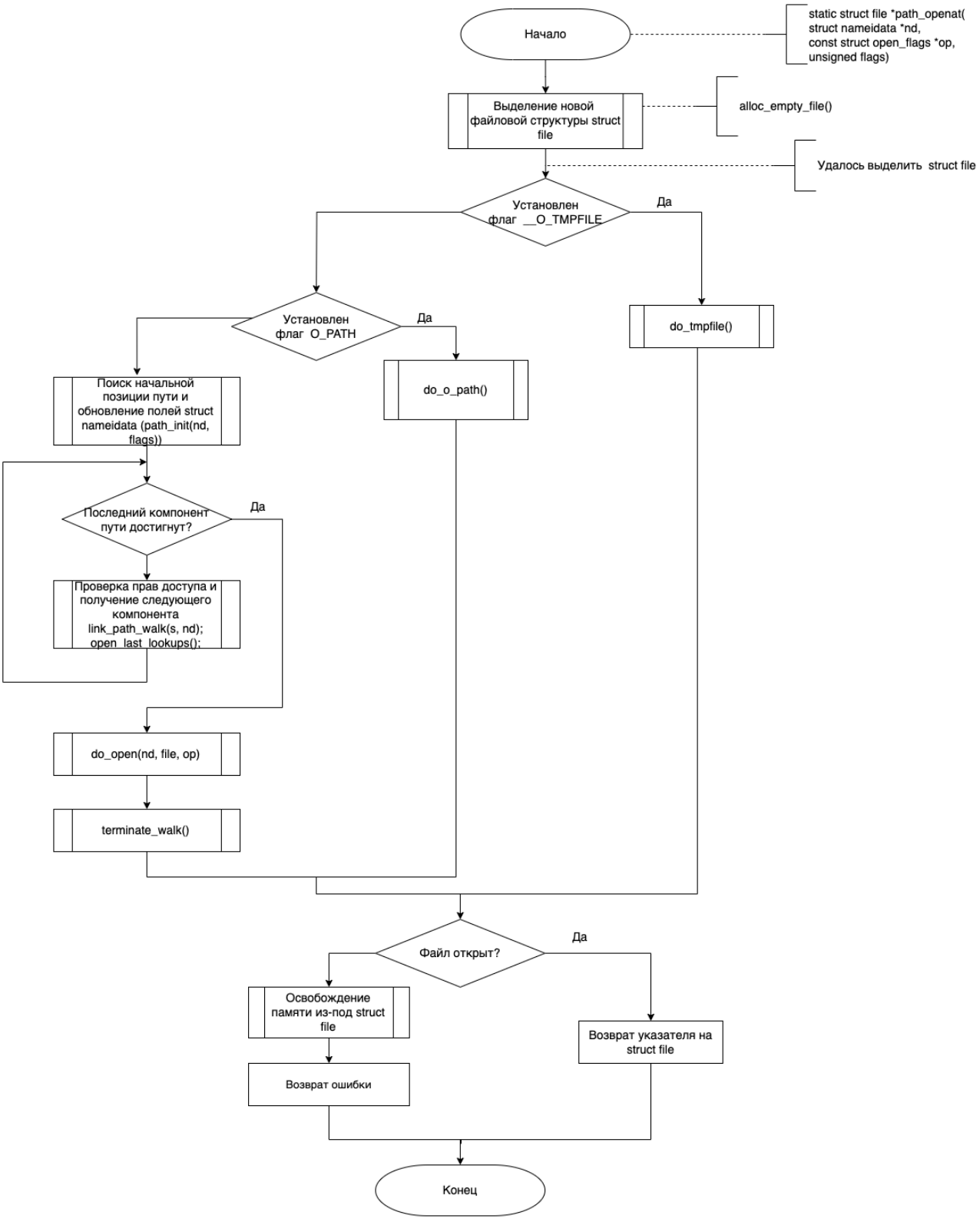

Рисунок 7 – Схема алгоритма работы функции path_openat()

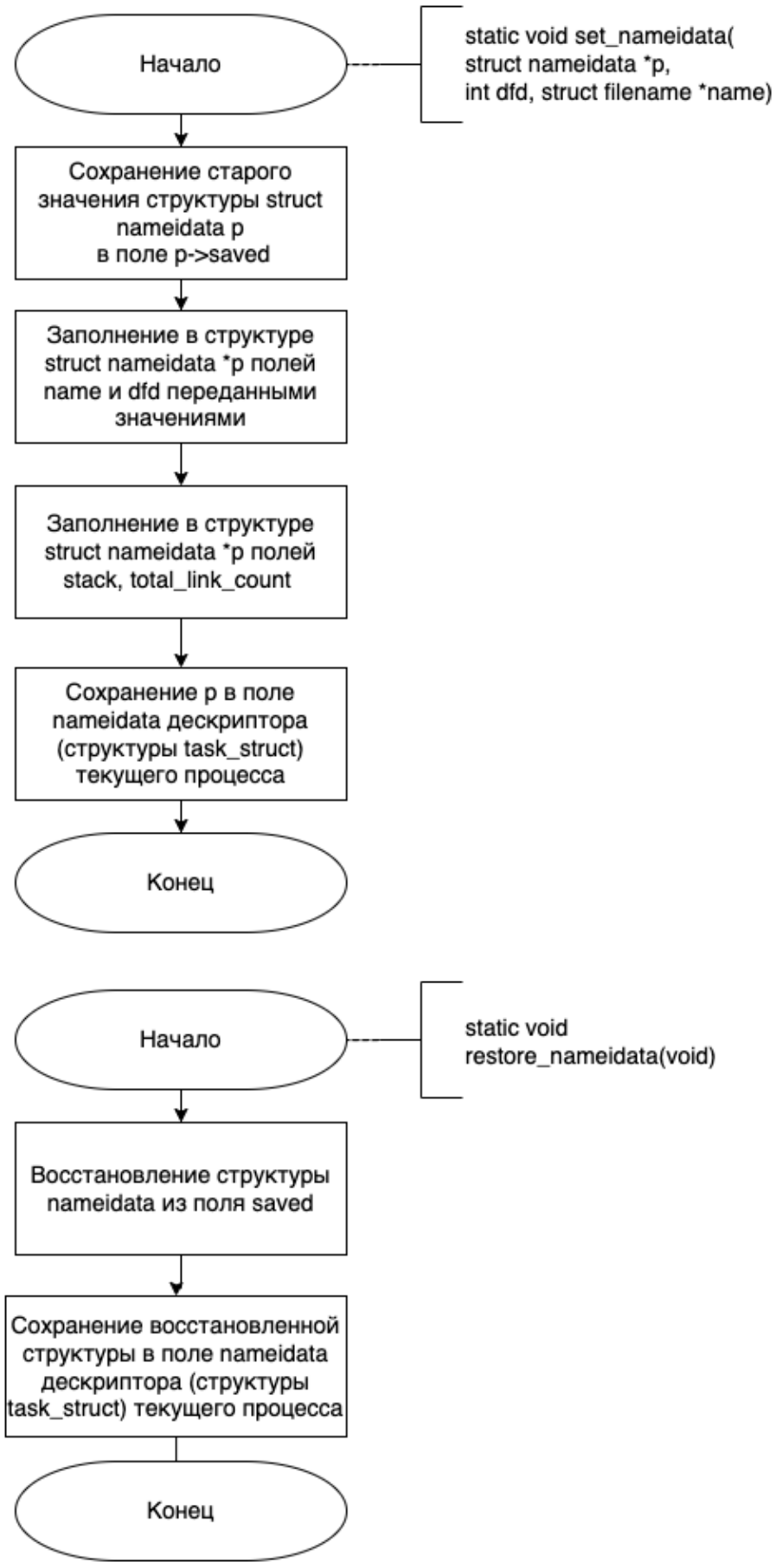

Рисунок 8 – Схема алгоритма работы функций, работающих с nameidata

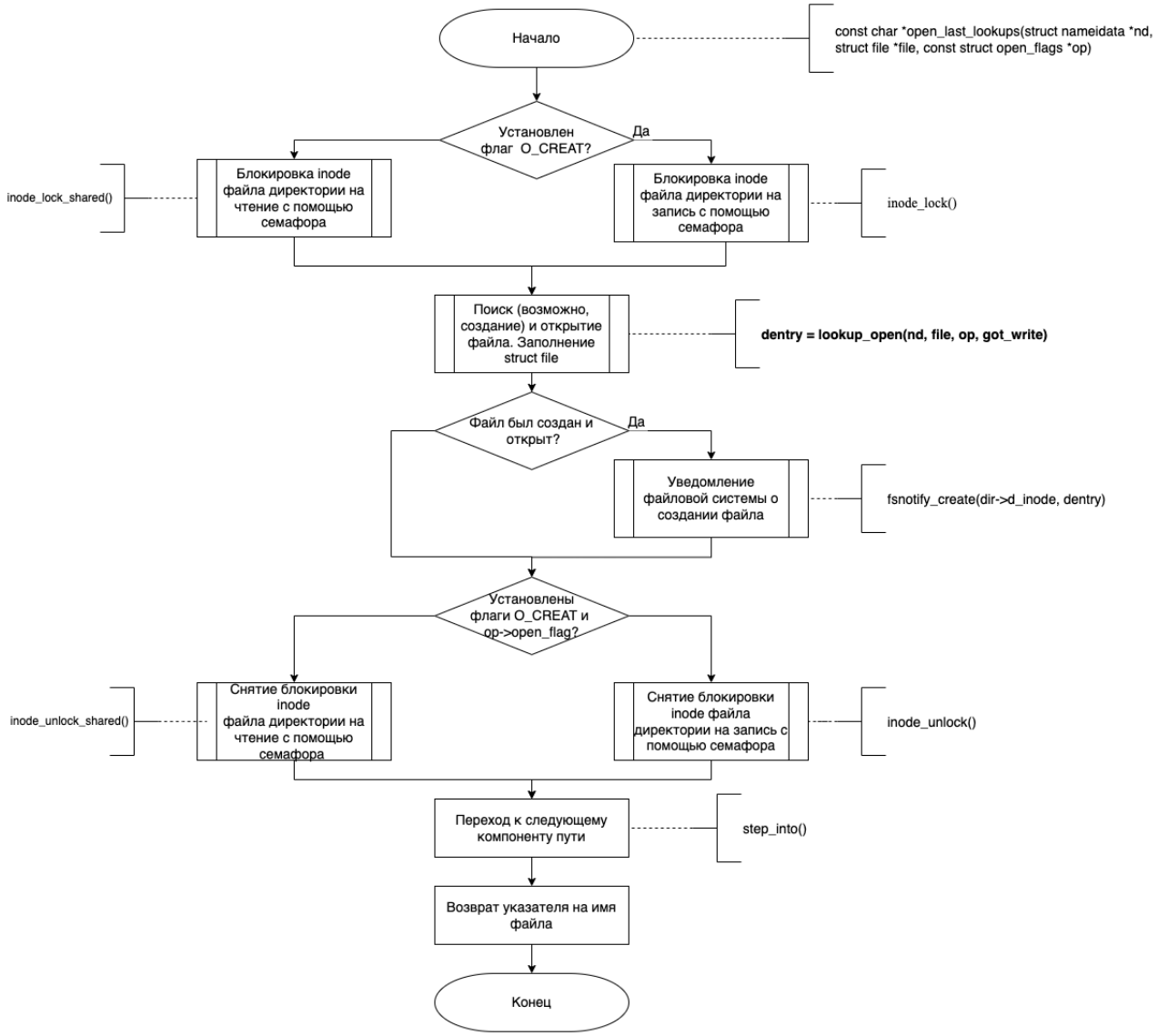

Рисунок 9 – Схема алгоритма работы функции open_last_lookups

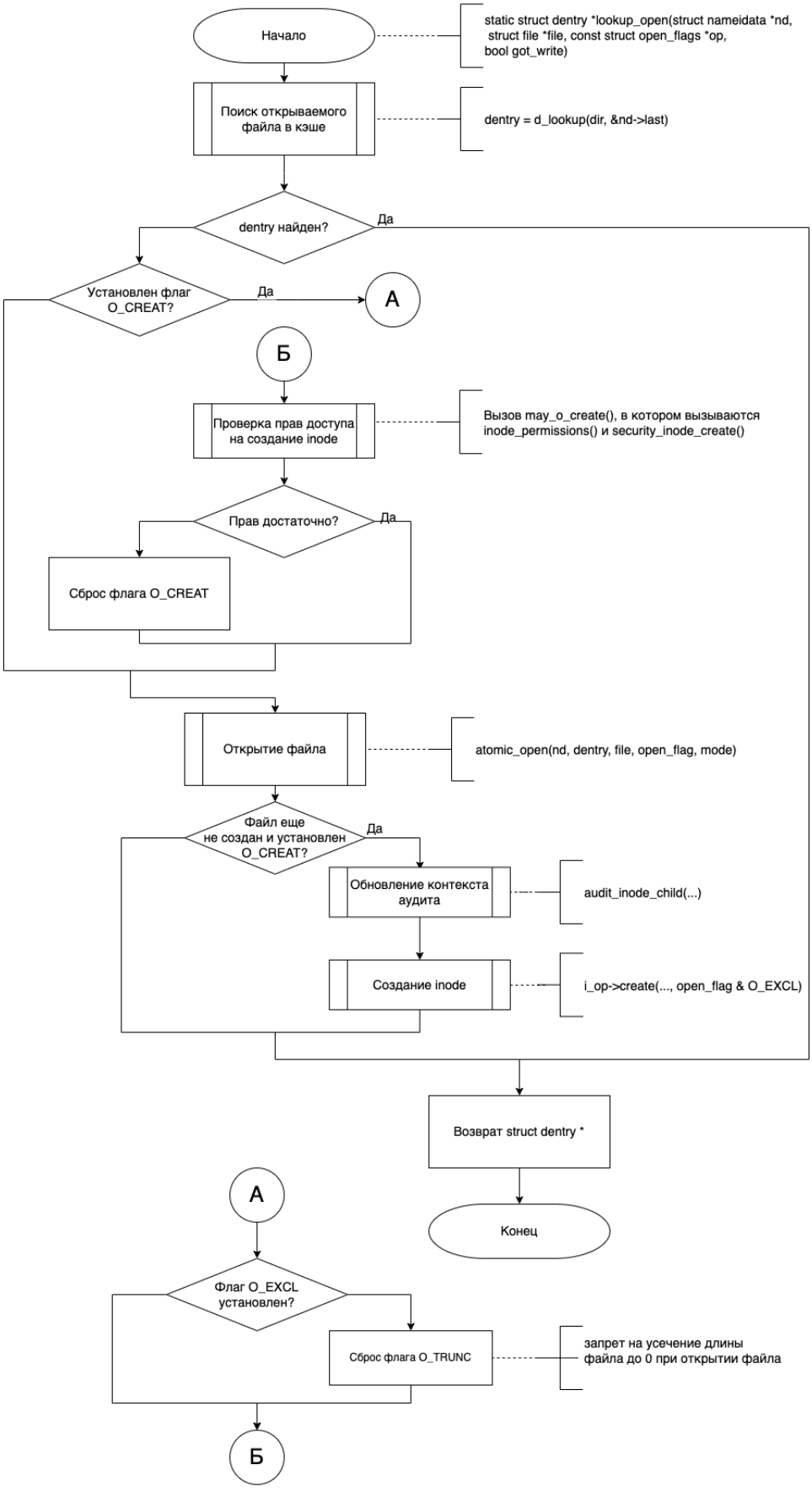

Рисунок 10 – Схема алгоритма работы функции lookup_open