

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 дисциплине «Операционные системы»

Тема Системный вызов open()

Студент Зайцева А. А.

Группа ИУ7-62Б

Оценка (баллы) \_\_\_\_\_

Преподаватель Рязанова Н. Ю.

Москва — 2022 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_CREAT`) быть создан `open()`.

Листинг 1.1: Системный вызов `open()`

```
1 #include <sys/types.h>
2 #include <sys/stat.h>
3 #include <fcntl.h>
4
5 int open(const char *pathname, int flags);
6 int open(const char *pathname, int flags, mode_t mode);
```

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

Первый аргумент - имя файла в файловой системе: полный путь к файлу или сокращенное имя.

Второй аргумент - это режим открытия файла, представляющий собой один или несколько флагов открытия, объединенных оператором побитового ИЛИ. Список доступных флагов:

`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` — 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_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`).

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

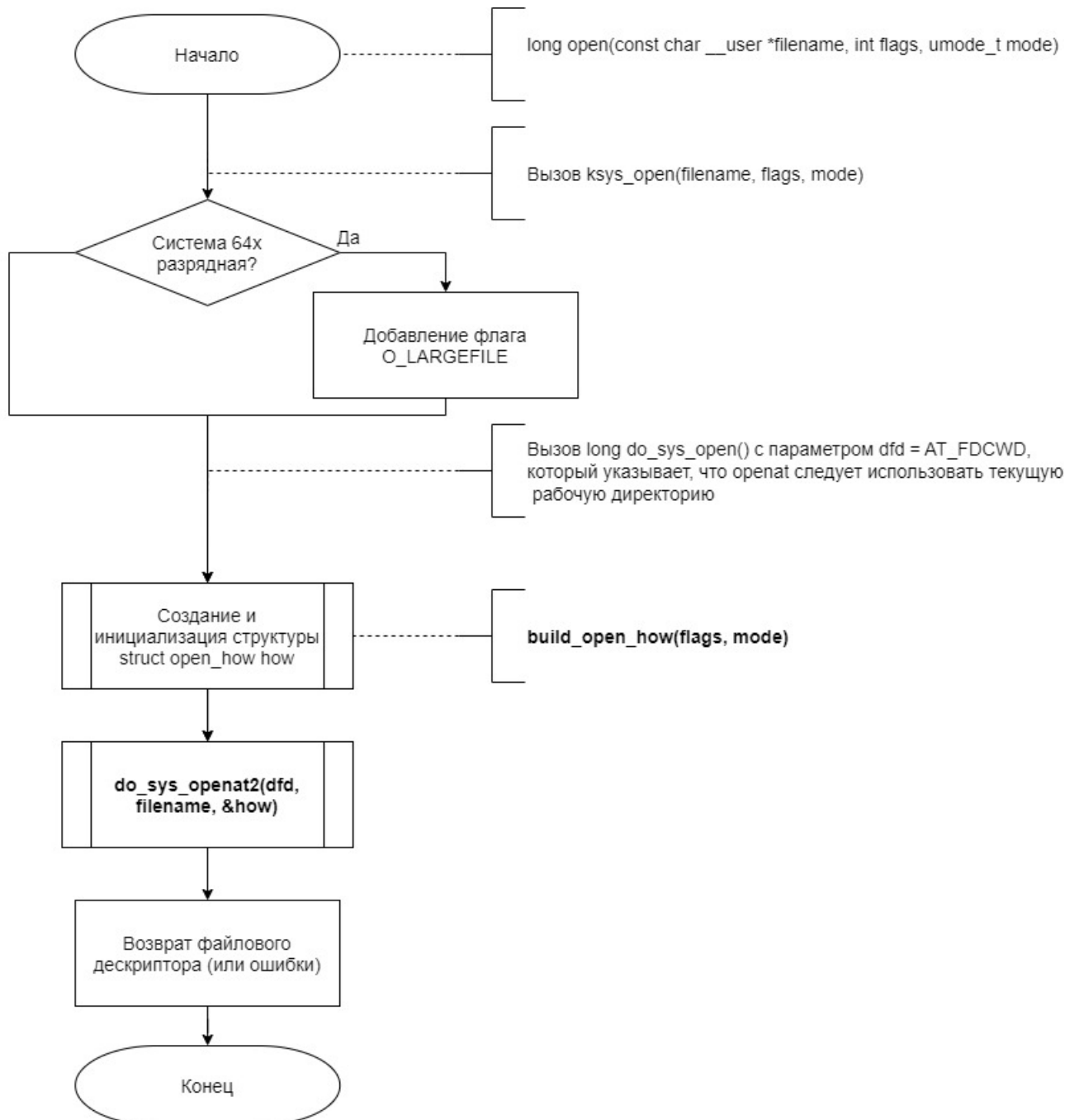
Третий аргумент используется в том случае, если `open()` создает новый файл. В этом случае файлу нужно задать права доступа (режим), с которыми он появится в файловой системе. Права доступа задаются перечислением флагов, объединенных побитовым ИЛИ. Список флагов:

`S_IRWXU 00700` пользователь (владелец файла) имеет права на чтение, запись и выполнение файла

S\_IRUSR 00400 пользователь имеет права на чтение файла  
S\_IWUSR 00200 пользователь имеет права на запись в файл  
S\_IXUSR 00100 пользователь имеет права на выполнение файла  
S\_IRWXG 00070 группа имеет права на чтение, запись и выполнение файла  
S\_IRGRP 00040 группа имеет права на чтение файла  
S\_IWGRP 00020 группа имеет права на запись в файл  
S\_IXGRP 00010 группа имеет права на выполнение файла  
S\_IRWXO 00007 все остальные имеют права на чтение, запись и выполнение файла  
S\_IROTH 00004 все остальные имеют права на чтение файла  
S\_IWOTH 00002 все остальные имеют права на запись в файл  
S\_IXOTH 00001 все остальные имеют права на выполнение файла  
S\_ISUID 0004000 бит set-user-ID  
S\_ISGID 0002000 бит set-group-ID bit  
S\_ISVTX 0001000 закрепляющий бит

## 2 Схемы алгоритмов

### 2.1 open()



## 2.2 build\_open\_how()

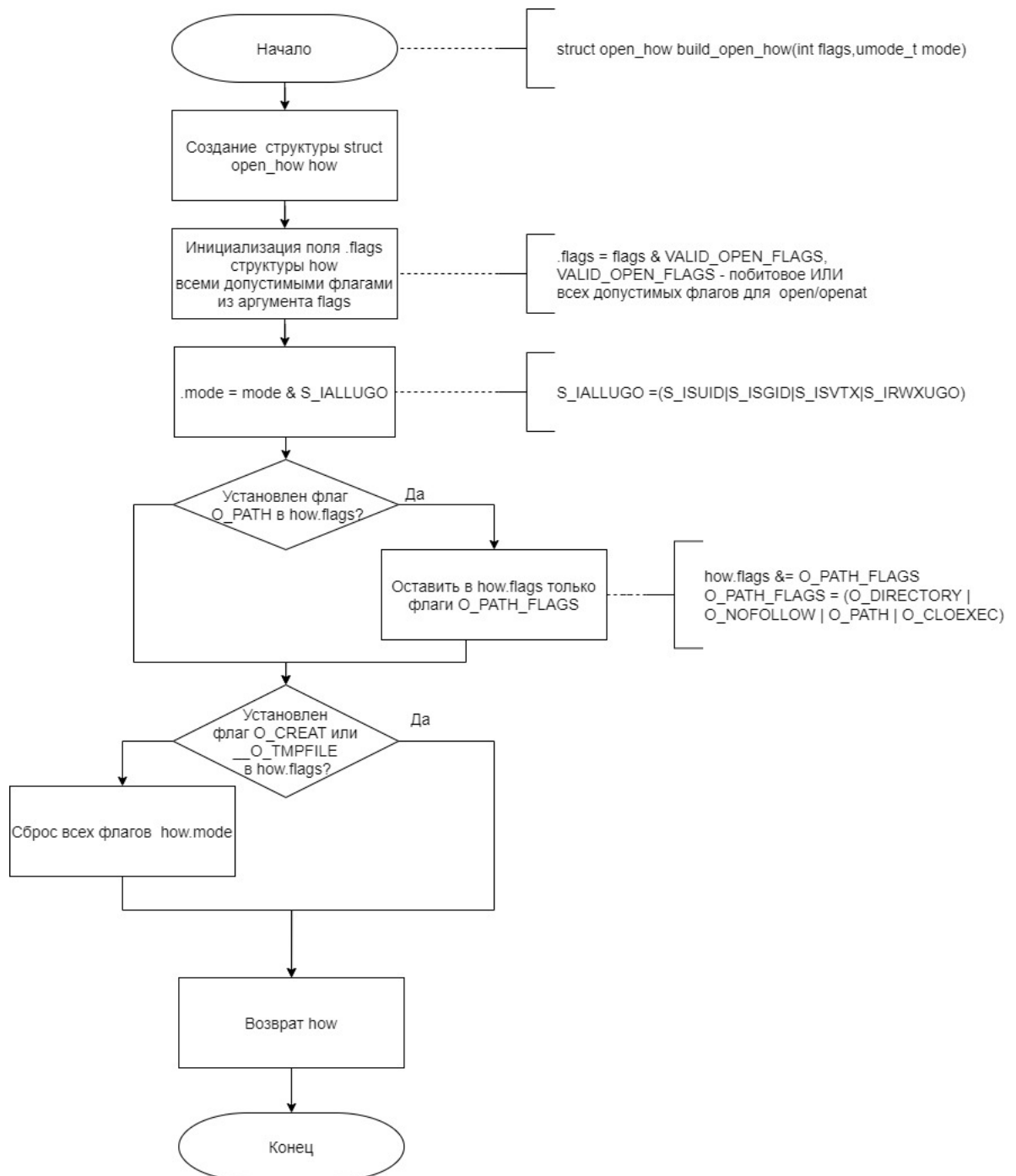
Листинг 2.1: Структура open\_how и функция build\_open\_how()

```
1  /*
2  Arguments for how openat2(2) should open the target path. If only
   @flags and @mode are non-zero, then openat2(2) operates very
   similarly to openat(2).
3  However, unlike openat(2), unknown or invalid bits in @flags
   result in -EINVAL rather than being silently ignored. @mode
   must be zero unless one of {O_CREAT, O_TMPFILE} are set.
4
5  * @flags: O_* flags.
6  * @mode: O_CREAT/O_TMPFILE file mode.
7  * @resolve: RESOLVE_* flags.
8  */
9
10 struct open_how {
11     __u64 flags;
12     __u64 mode;
13     __u64 resolve;
14 };
15
16 /* List of all valid flags for the open/openat flags argument: */
17 #define VALID_OPEN_FLAGS \
18     (O_RDONLY | O_WRONLY | O_RDWR | O_CREAT | O_EXCL | O_NOCTTY |
19      O_TRUNC | \
20      O_APPEND | O_NDELAY | O_NONBLOCK | __O_SYNC | O_DSYNC | \
21      FASYNC | O_DIRECT | O_LARGEFILE | O_DIRECTORY | O_NOFOLLOW |
22      O_NOATIME | O_CLOEXEC | O_PATH | __O_TMPFILE)
23
24 #define S_IALLUGO (S_ISUID|S_ISGID|S_ISVTX|S_IRWXUGO)
25 #define WILL_CREATE(flags) (flags & (O_CREAT | __O_TMPFILE))
26 #define O_PATH_FLAGS (O_DIRECTORY | O_NOFOLLOW | O_PATH |
27     O_CLOEXEC)
28
29 inline struct open_how build_open_how(int flags, umode_t mode)
30 {
31     struct open_how how = {
32         .flags = flags & VALID_OPEN_FLAGS,
33         .mode = mode & S_IALLUGO,
34     };
35 }
```

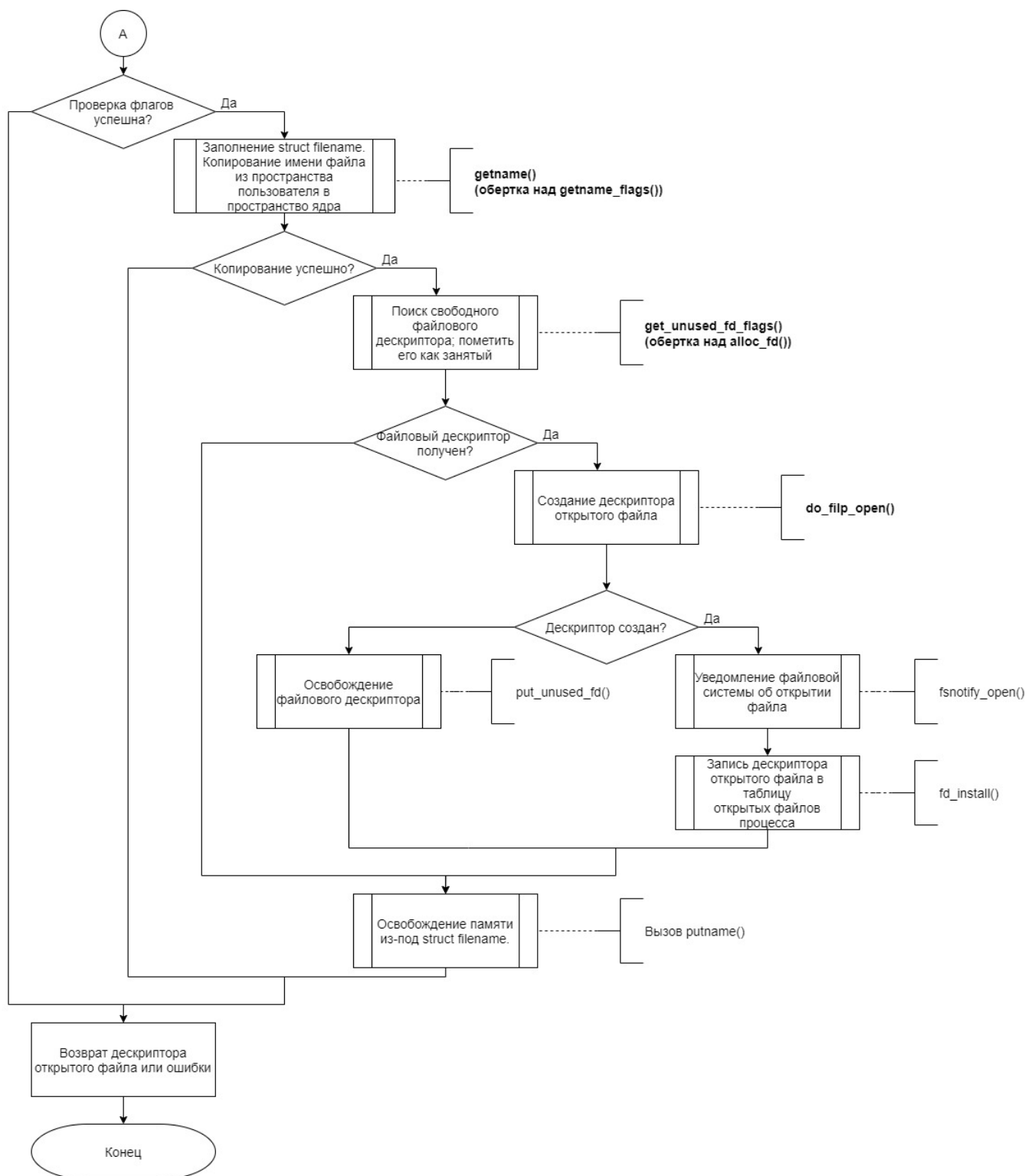
```

31  /* O_PATH beats everything else. */
32  if (how.flags & O_PATH)
33      how.flags &= O_PATH_FLAGS;
34  /* Modes should only be set for create-like flags. */
35  if (!WILL_CREATE(how.flags))
36      how.mode = 0;
37  return how;
38  }

```



## 2.3 do\_sys\_openat2()

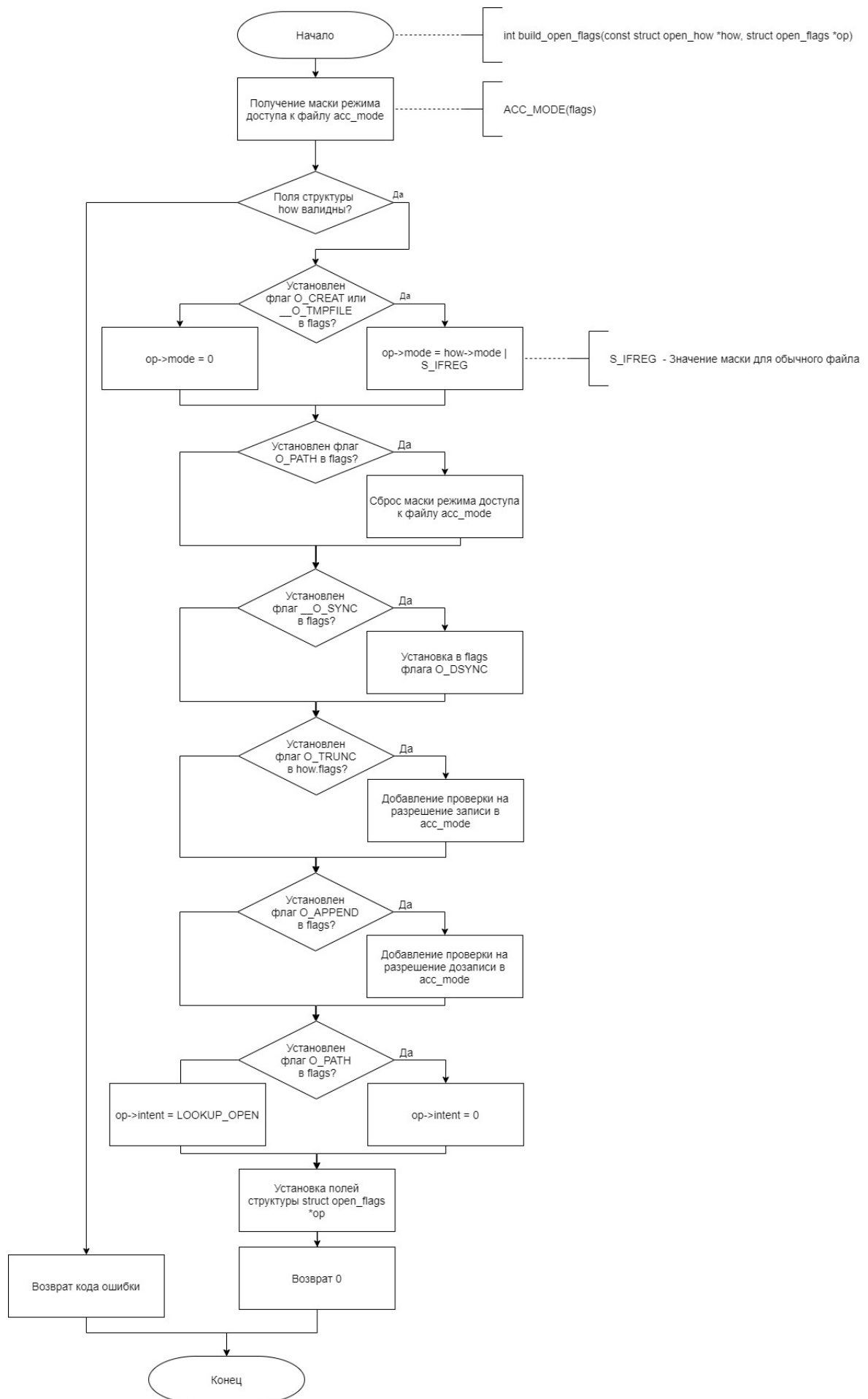


## 2.4 build\_open\_flags()



## Листинг 2.2: Структура open\_flags

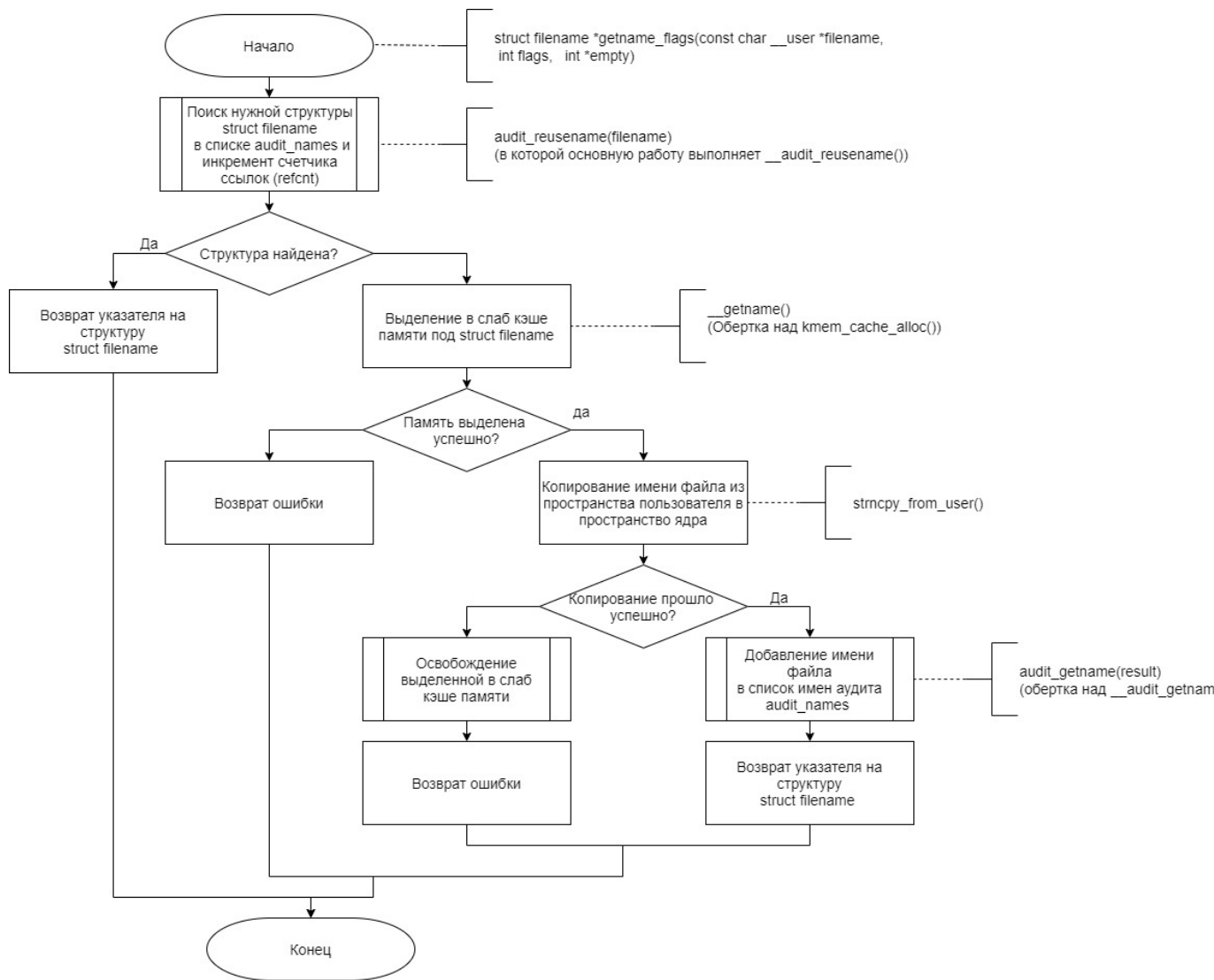
```
1
2 struct open_flags {
3     int open_flag;
4     umode_t mode;
5     int acc_mode;
6     int intent;
7     int lookup_flags;
8 };
```



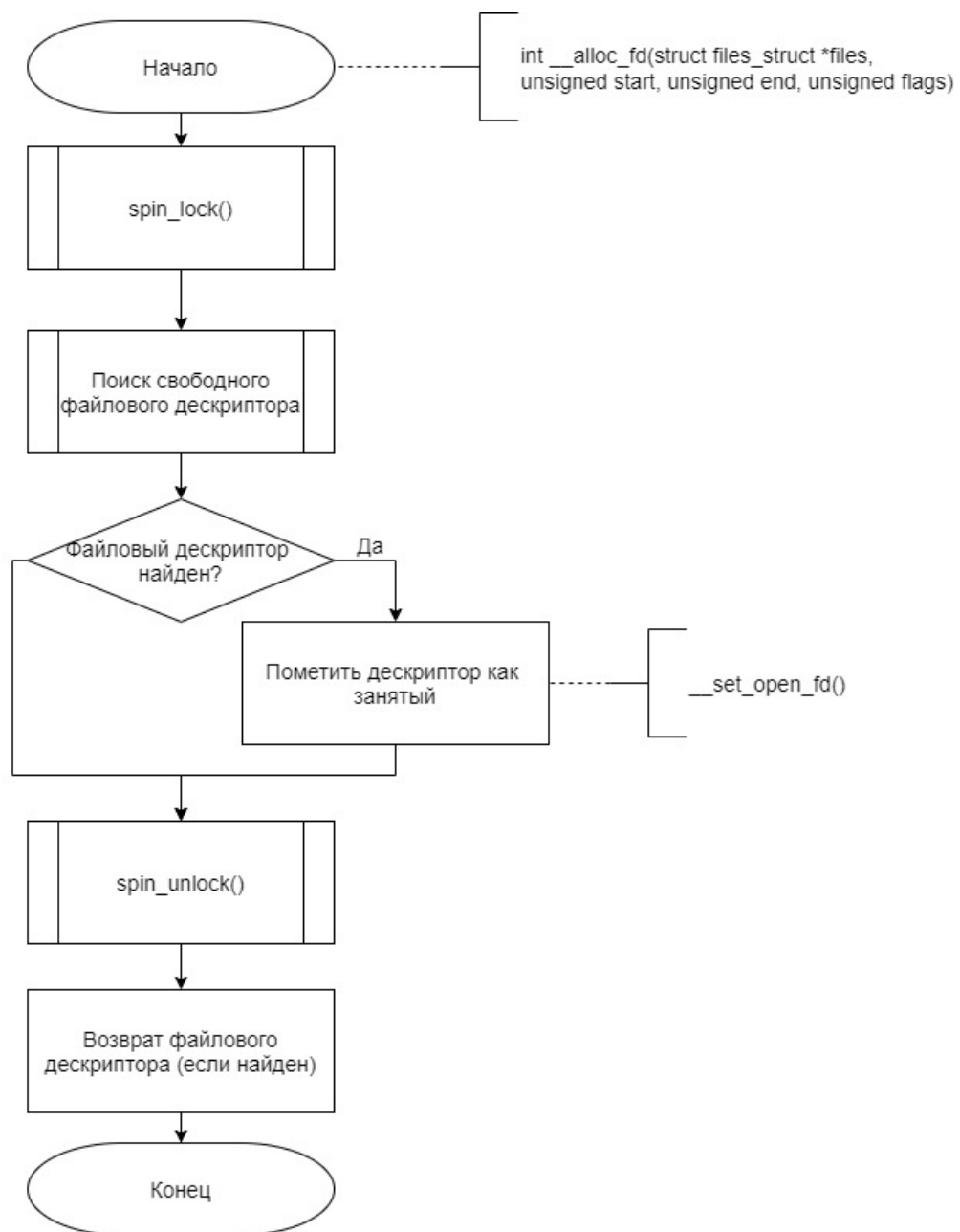
## 2.5 getname\_flags()

Листинг 2.3: Структуры audit\_names и filename

```
1
2 struct audit_names;
3
4
5 struct filename {
6     const char    *name; /* pointer to actual string */
7     const __user char *uptr; /* original userland pointer */
8     int          refcnt;
9     struct audit_names *aname;
10    const char    iname[];
11 };
12
13 struct audit_names {
14     struct list_head list; /* audit_context->names_list */
15
16     struct filename *name;
17     int          name_len; /* number of chars to log */
18     bool         hidden; /* don't log this record */
19
20     unsigned long ino;
21     dev_t         dev;
22     umode_t       mode;
23     kuid_t        uid;
24     kgid_t        gid;
25     dev_t         rdev;
26     u32           osid;
27     struct audit_cap_data fcap;
28     unsigned int   fcap_ver;
29     unsigned char  type; /* record type */
30     /*
31      * This was an allocated audit_names and not from the array of
32      * names allocated in the task audit context. * Thus this name
33      * should be freed on syscall exit.
34      */
35     bool          should_free;
36 };
```



## 2.6 \_\_alloc\_fd



## 2.7 do\_filp\_open

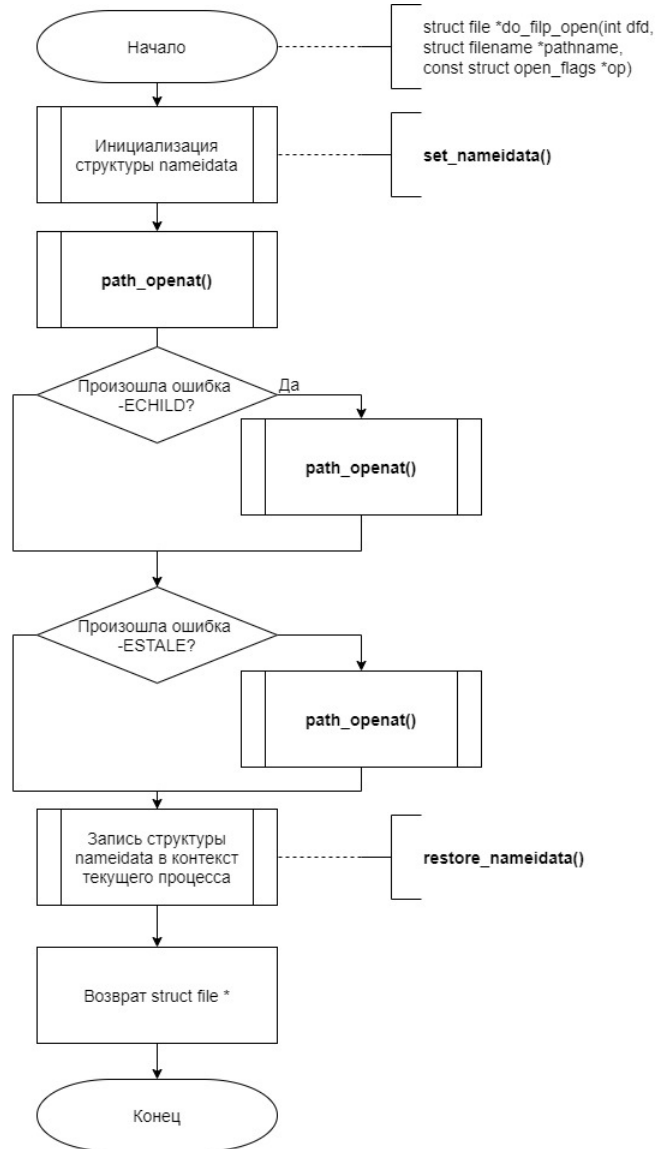
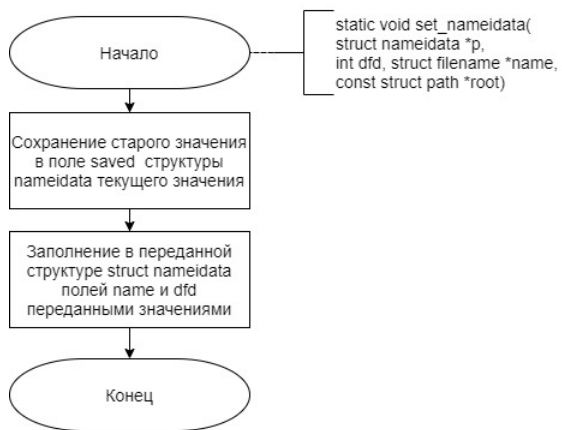
Листинг 2.4: Структура nameidata

```
1 struct nameidata {
2     struct path path;
3     struct qstr last;
4     struct path root;
```

```

5  struct inode  *inode; /* path.dentry.d_inode */
6  unsigned int  flags , state;
7  unsigned  seq , m_seq , r_seq;
8  int    last_type;
9  unsigned  depth;
10 int    total_link_count;
11 struct saved {
12     struct path link;
13     struct delayed_call done;
14     const char *name;
15     unsigned seq;
16 } *stack , internal[EMBEDDED_LEVELS];
17 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;

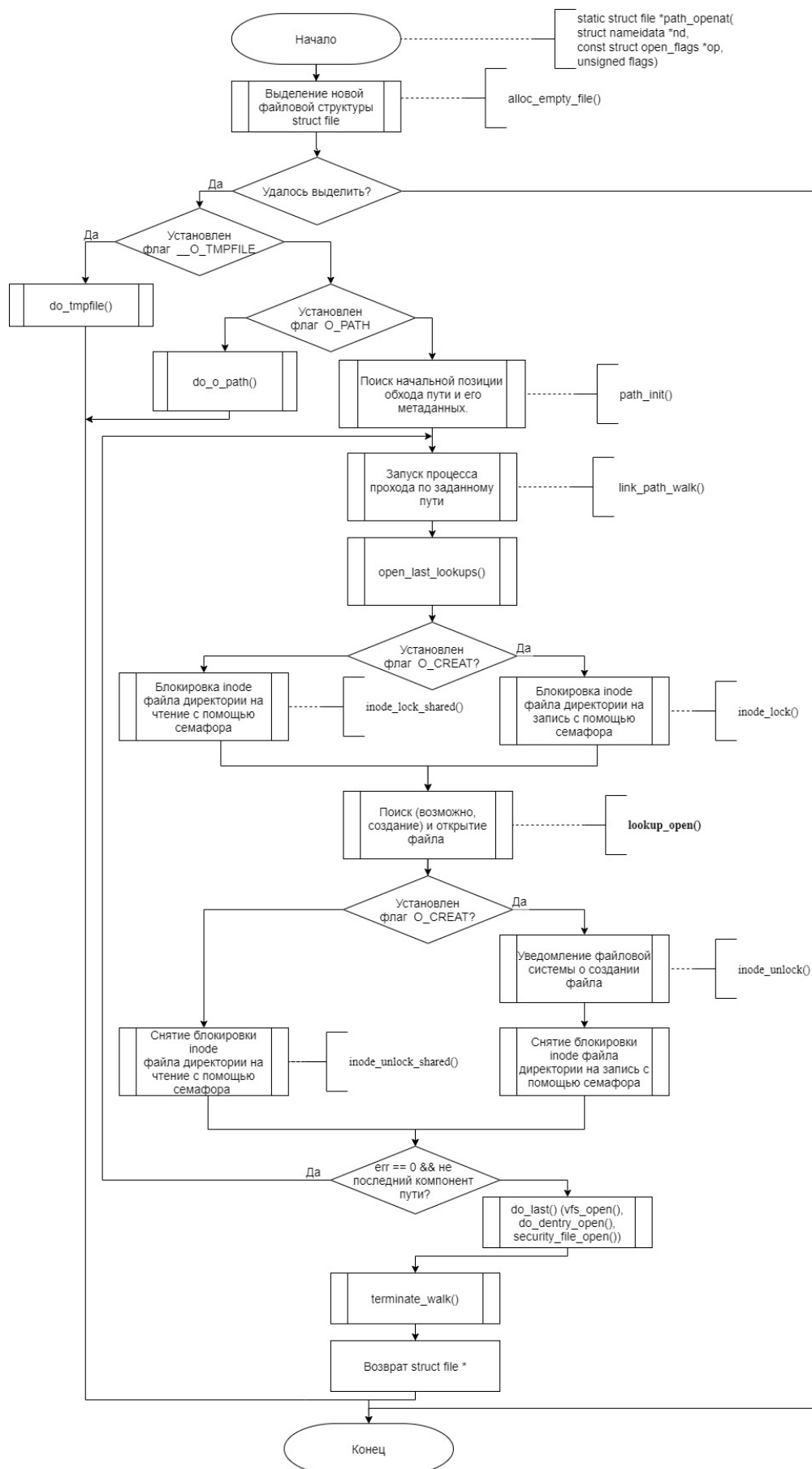
```







## 2.8 path\_openat



## 2.9 lookup\_open

