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()

```
#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() - 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

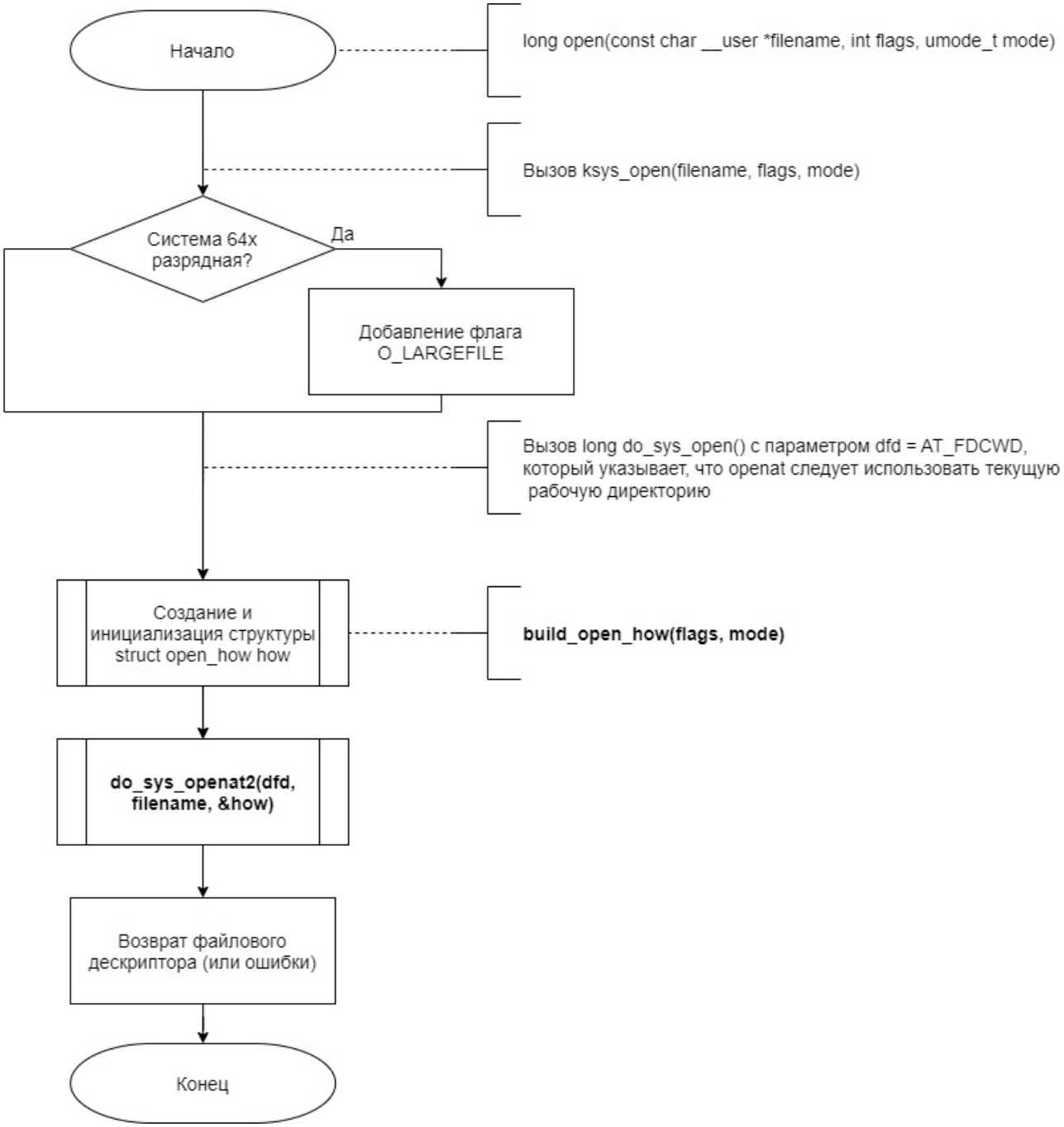
S_ISUID 0004000 бит set-user-ID

S_ISGID 0002000 бит set-group-ID bit

S_ISVTX 0001000 закрепляющий бит

2 Схемы алгоритмов

2.1 open()

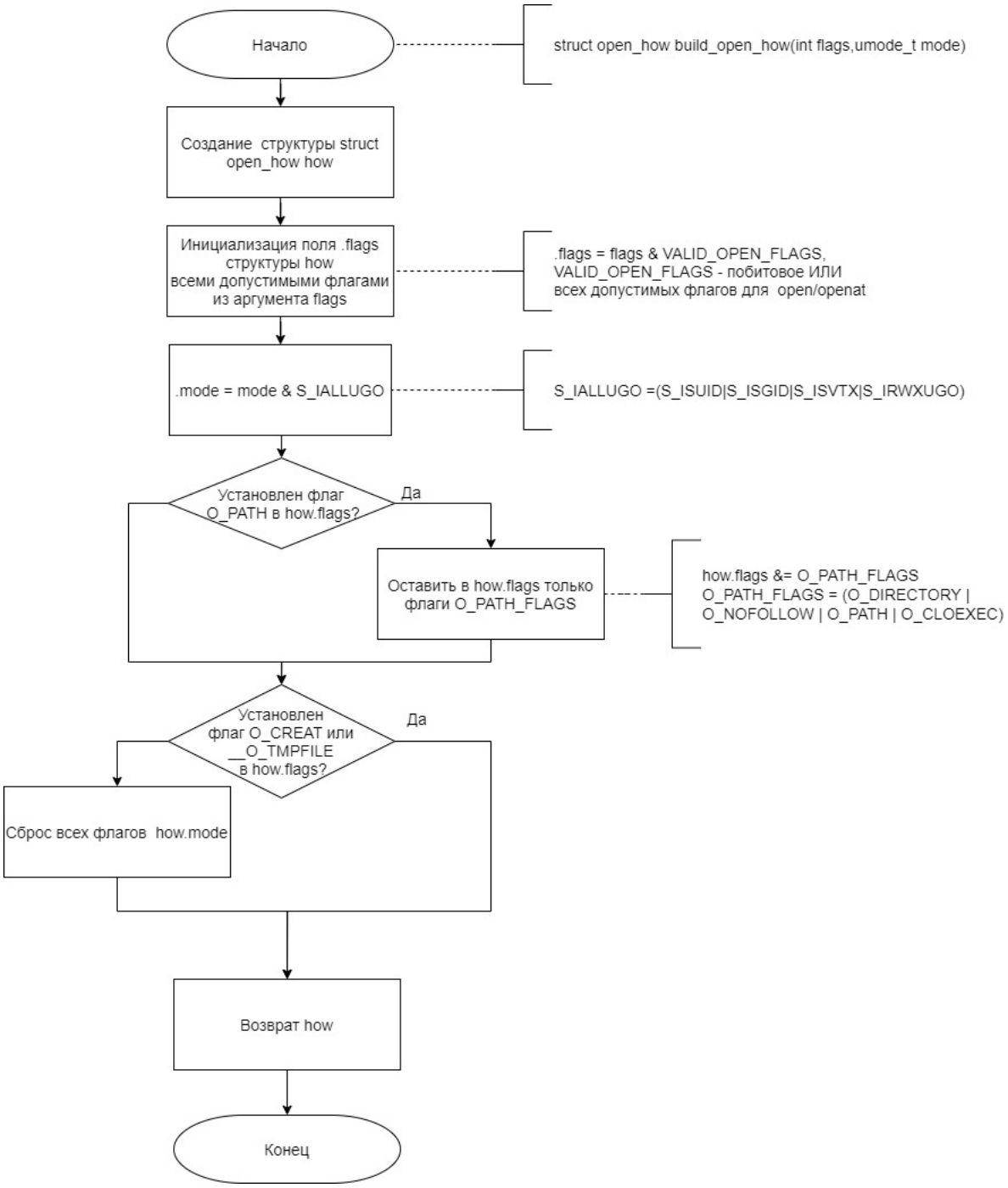


2.2 build_open_how()

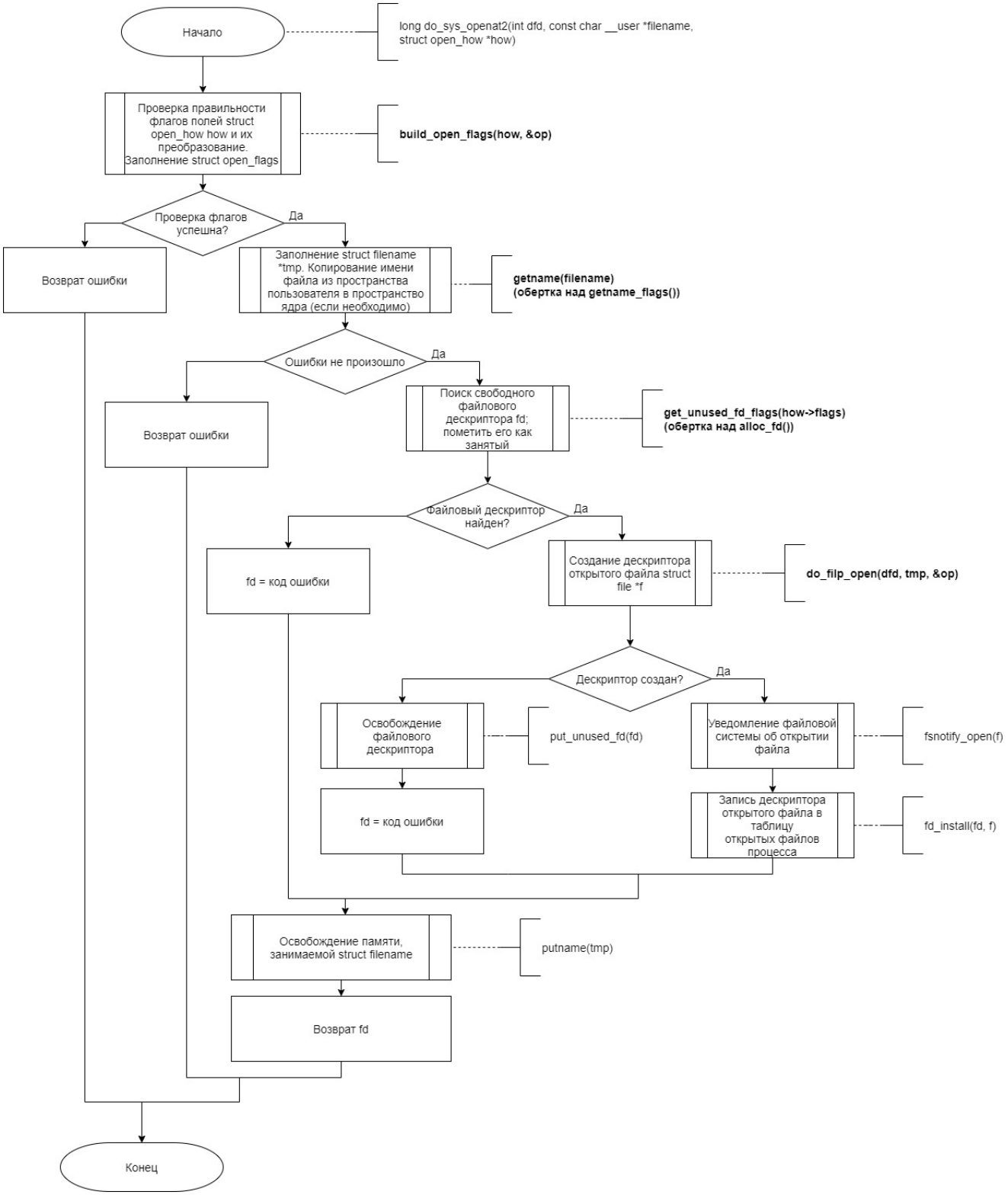
Листинг 2.1: Структура open_how и функция build_open_how()

```
/*
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).
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.

 * @flags: O_* flags.
 * @mode: O_CREAT/O_TMPFILE file mode.
 * @resolve: RESOLVE_* flags.
 */

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

/* List of all valid flags for the open/openat flags argument: */
#define VALID_OPEN_FLAGS \
  (O_RDONLY | O_WRONLY | O_RDWR | O_CREAT | O_EXCL | O_NOCTTY |
     O_TRUNC | \
   O_APPEND | O_NDELAY | O_NONBLOCK | __O_SYNC | O_DSYNC | \
   FASYNC | O_DIRECT | O_LARGEFILE | O_DIRECTORY | O_NOFOLLOW |
      O_NOATIME | O_CLOEXEC | O_PATH | __O_TMPFILE)

#define S_IALLUGO (S_ISUID|S_ISGID|S_ISVTX|S_IRWXUGO)
#define WILL_CREATE(flags)  (flags & (O_CREAT | __O_TMPFILE))
#define O_PATH_FLAGS    (O_DIRECTORY | O_NOFOLLOW | O_PATH |
   O_CLOEXEC)

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,};
```

```
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  }
```



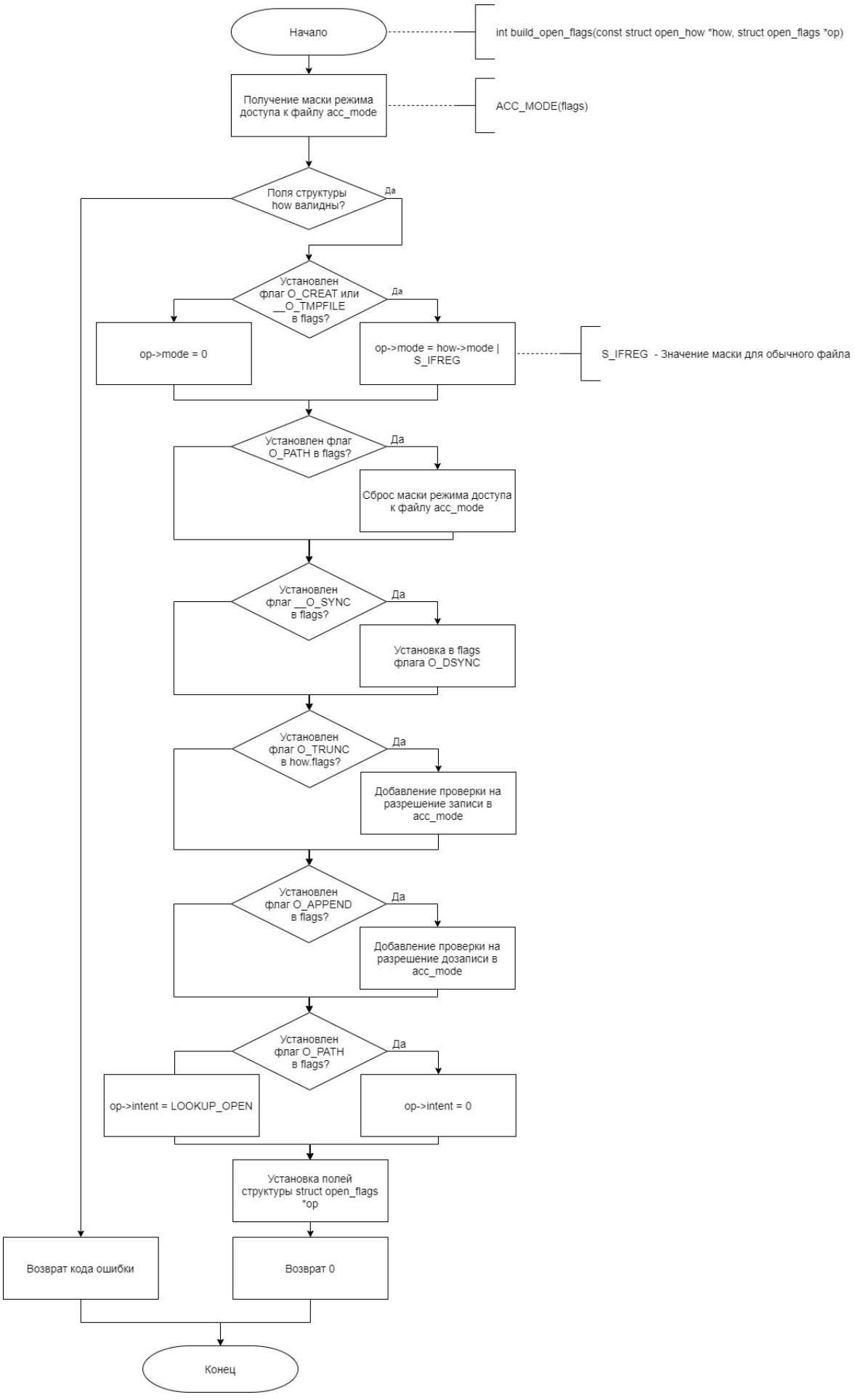
2.3 do_sys_openat2()



2.4 build_open_flags()

Листинг 2.2: Структура open_flags

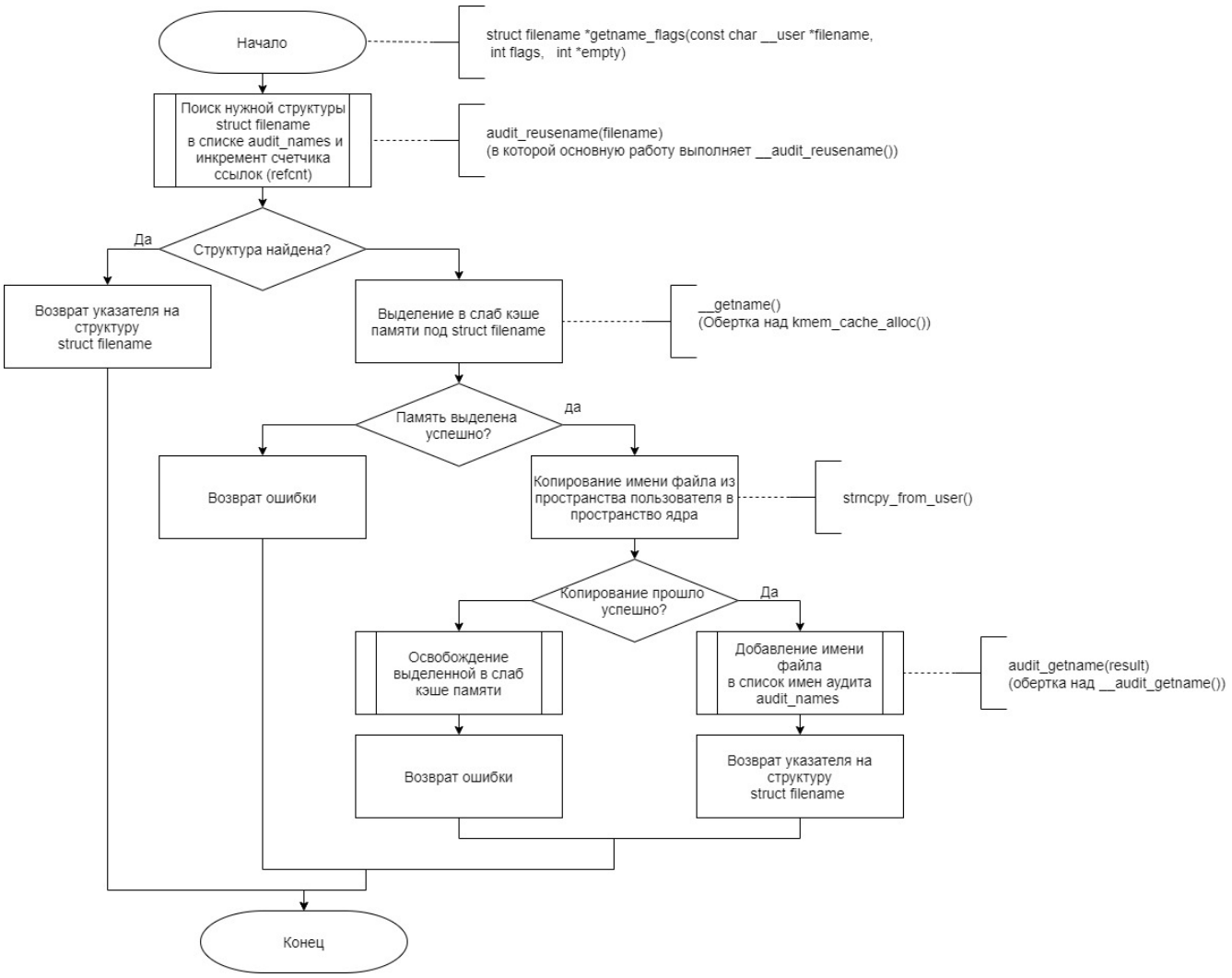
```
1
2  struct open_flags {
3    int open_flag;
4    umode_t mode;
5    int acc_mode;
6    int intent;
7    int lookup_flags;
8  };
```


Начало
int build_open_flags(const struct open_how *how, struct open_flags *op)
Получение маски режима доступа к файлу acc_mode
ACC_MODE(flags)
Поля структуры how валидны?
Да
Установлен флаг O_CREAT или __O_TMPFILE в flags?
Да
op->mode = 0
op->mode = how->mode | S_IFREG
S_IFREG - Значение маски для обычного файла
Установлен флаг O_PATH в flags?
Да
Сброс маски режима доступа к файлу acc_mode
Установлен флаг __O_SYNC в flags?
Да
Установка в flags флага O_DSYNC
Установлен флаг O_TRUNC в how.flags?
Да
Добавление проверки на разрешение записи в acc_mode
Установлен флаг O_APPEND в flags?
Да
Добавление проверки на разрешение дозаписи в acc_mode
Установлен флаг O_PATH в flags?
Да
op->intent = LOOKUP_OPEN
op->intent = 0
Установка полей структуры struct open_flags *op
Возврат кода ошибки
Возврат 0
Конец

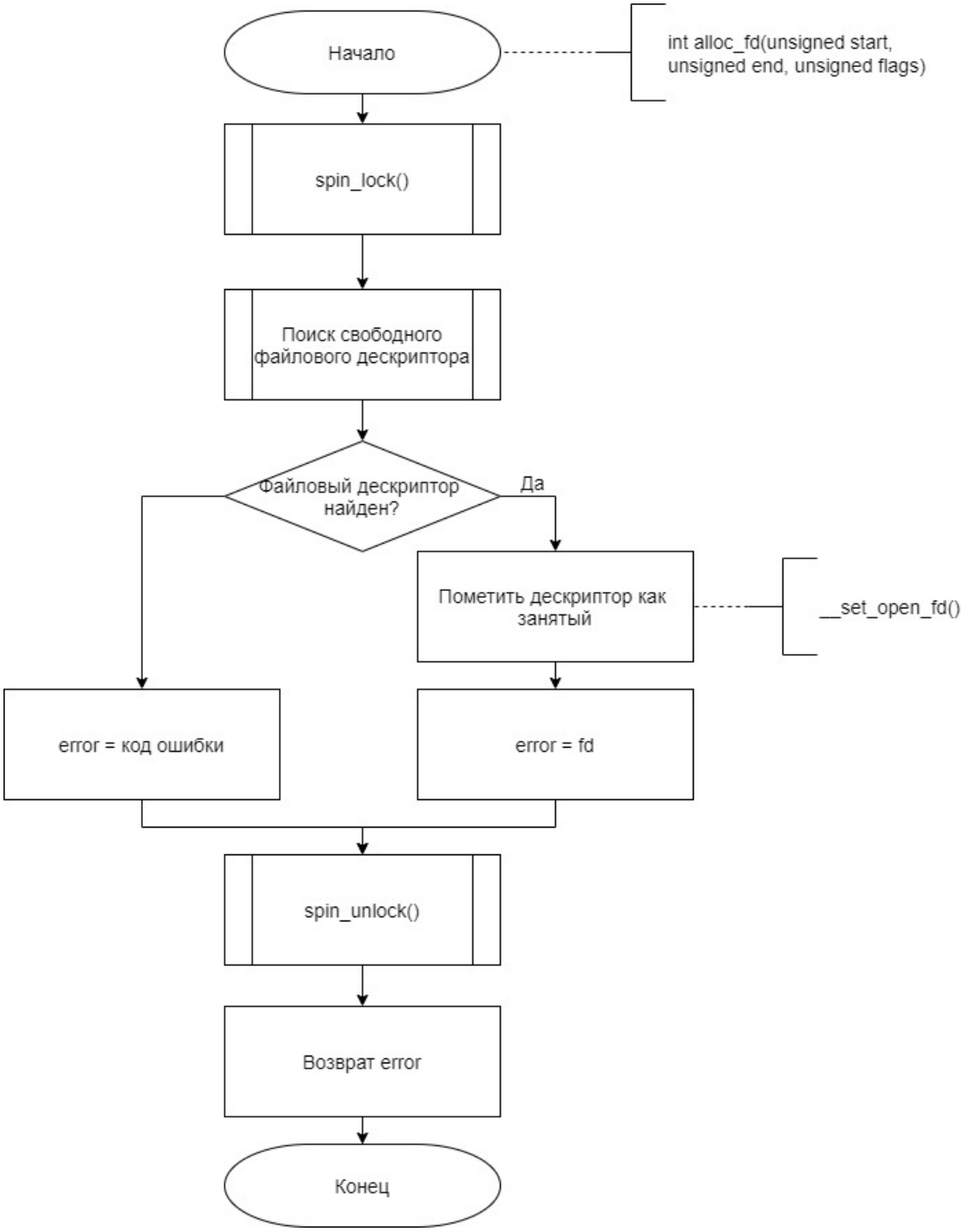
2.5 getname_flags()

Листинг 2.3: Структуры audit_names и filename

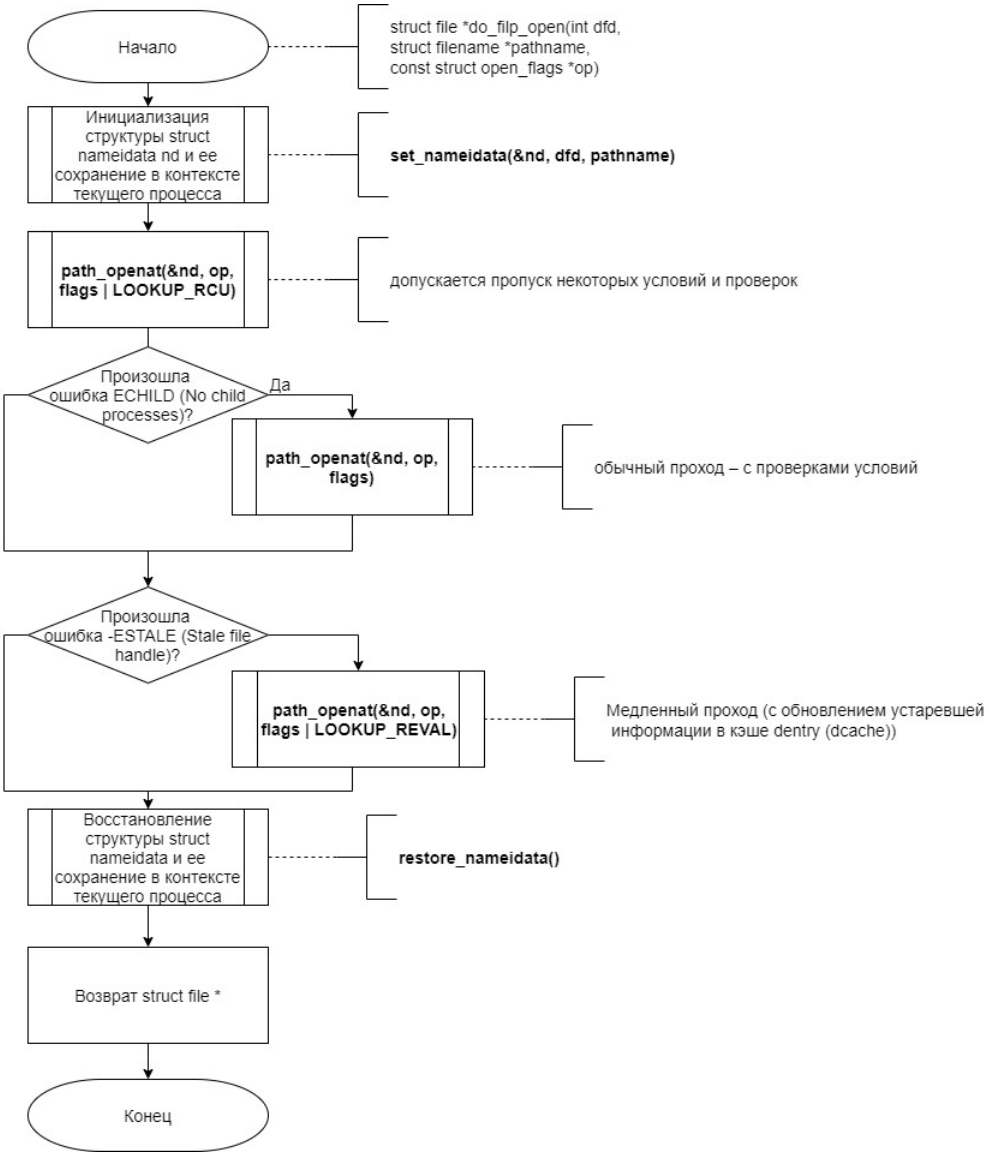
```
1  struct audit_names;
2
3  struct filename {
4    const char     *name;  /* pointer to actual string */
5    const __user char *uptr;  /* original userland pointer */
6    int      refcnt;
7    struct audit_names  *aname;
8    const char     iname[];
9  };
10
11 struct audit_names {
12   struct list_head  list; /* audit_context->names_list */
13
14   struct filename   *name;
15   int      name_len; /* number of chars to log */
16   bool       hidden; /* don't log this record */
17
18   unsigned long   ino;
19   dev_t      dev;
20   umode_t      mode;
21   kuid_t       uid;
22   kgid_t       gid;
23   dev_t      rdev;
24   u32      osid;
25   struct audit_cap_data fcap;
26   unsigned int    fcap_ver;
27   unsigned char   type;   /* record type */
28   /*
29   * 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.
30   */
31   bool       should_free;
32 };
```


Начало
struct filename *getname_flags(const char __user *filename,
int flags, int *empty)
Поиск нужной структуры
struct filename
в списке audit_names и
инкремент счетчика
ссылок (refcnt)
audit_reusename(filename)
(в которой основную работу выполняет __audit_reusename())
Структура найдена?
Да
Возврат указателя на
структуру
struct filename
Выделение в слаб кэше
памяти под struct filename
__getname()
(Обертка над kmem_cache_alloc())
Память выделена
успешно?
да
Возврат ошибки
Копирование имени файла из
пространства пользователя в
пространство ядра
strncpy_from_user()
Копирование прошло
успешно?
Да
Освобождение
выделенной в слаб
кэше памяти
Добавление имени
файла
в список имен аудита
audit_names
audit_getname(result)
(обертка над __audit_getname())
Возврат ошибки
Возврат указателя на
структуру
struct filename
Конец

2.6 alloc_fd


Начало
int alloc_fd(unsigned start, unsigned end, unsigned flags)
spin_lock()
Поиск свободного файлового дескриптора
Файловый дескриптор найден?
Да
Пометить дескриптор как занятый
__set_open_fd()
error = код ошибки
error = fd
spin_unlock()
Возврат error
Конец

2.7 do_filp_open

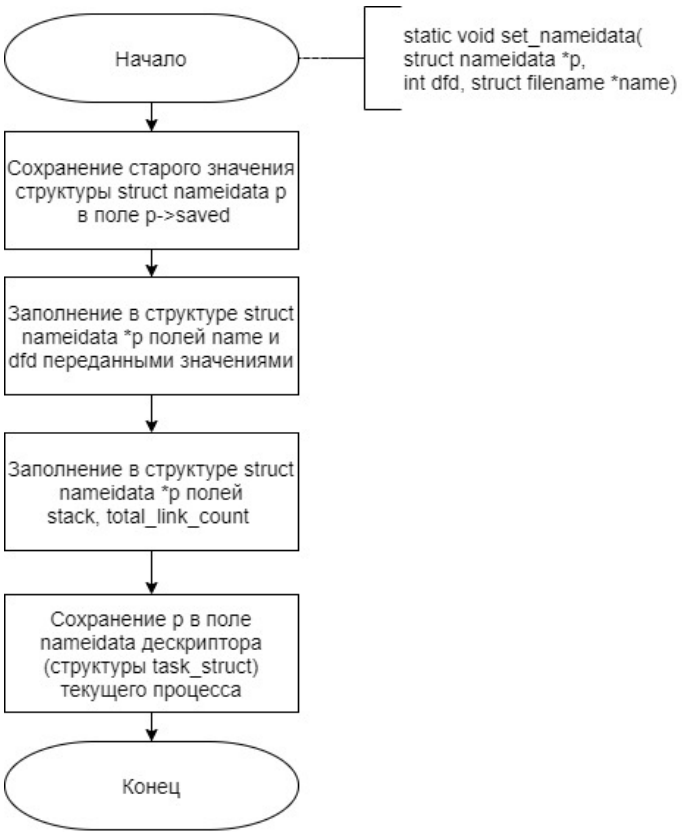


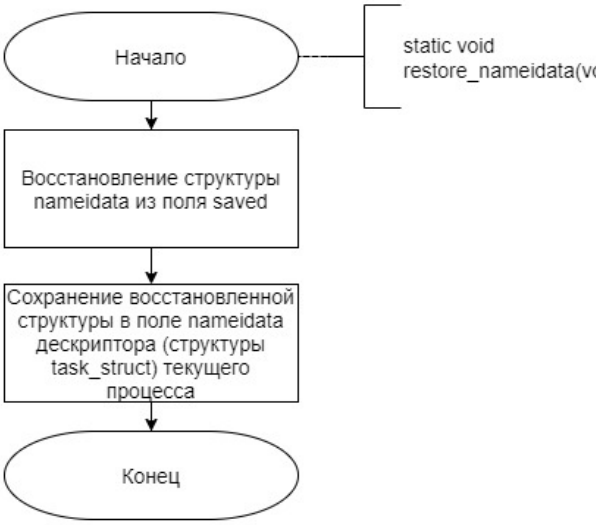
2.8 set_nameidata() и resotre_nameidata()

Листинг 2.4: Структура nameidata

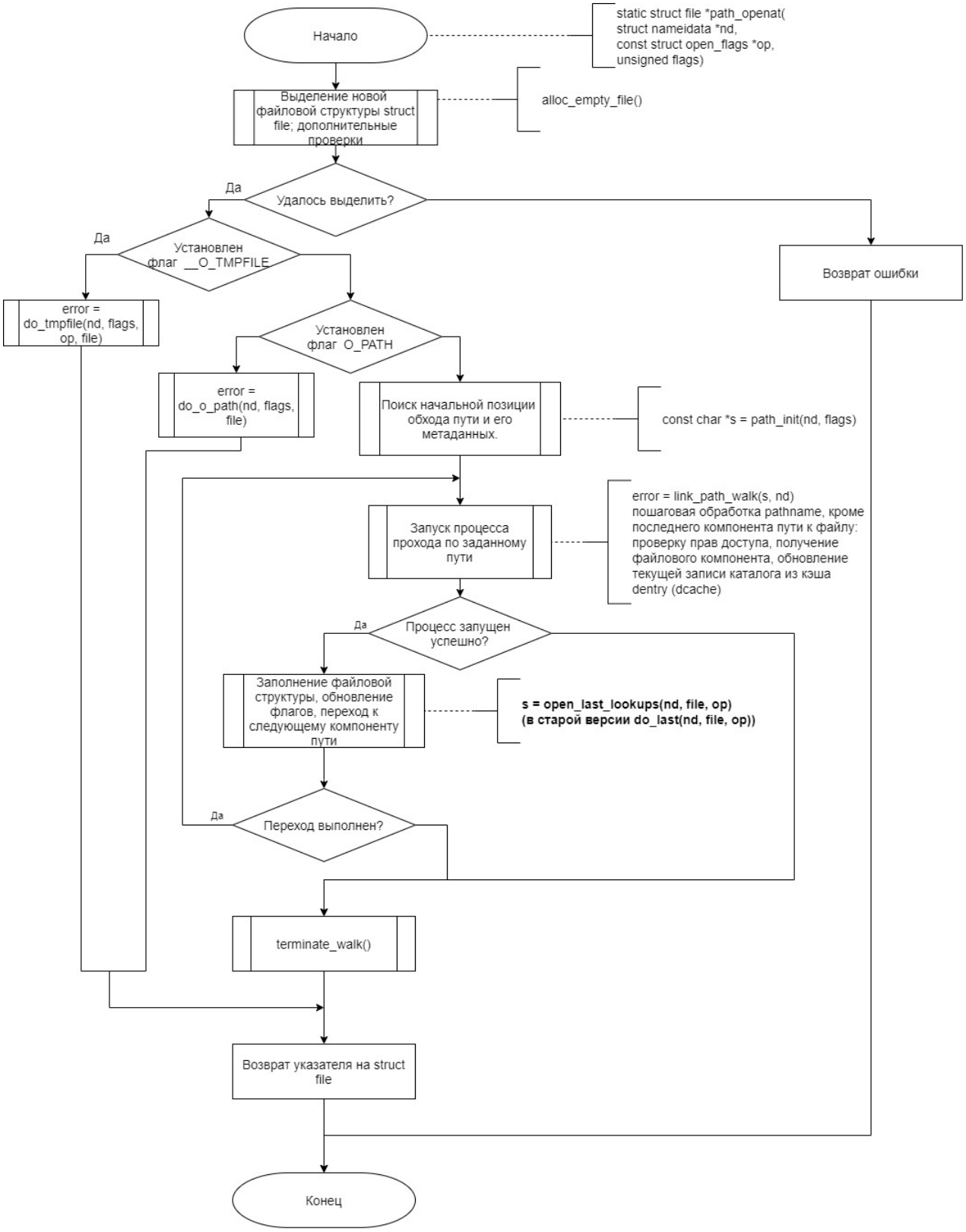
```
struct nameidata {
  struct path path;
  struct qstr last;
  struct path root;
  struct inode  *inode; /* path.dentry.d_inode */
  unsigned int  flags, state;
  unsigned  seq, m_seq, r_seq;
```

```
    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;
    struct nameidata *saved;
    unsigned  root_seq;
    int     dfd;
    kuid_t      dir_uid;
    umode_t     dir_mode;
} __randomize_layout;
```

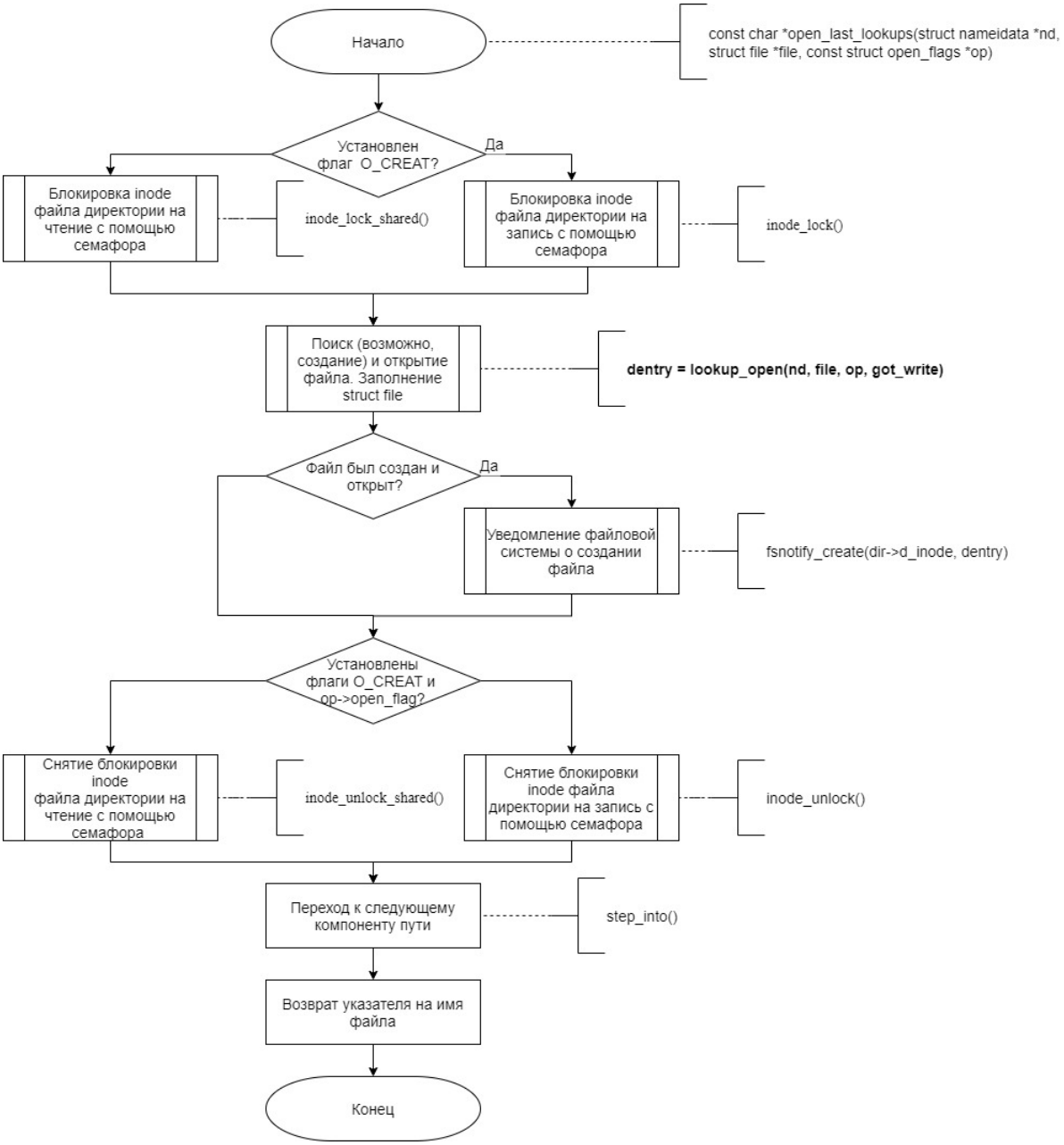




2.9 path_openat



2.10 open_last_lookups



2.11 lookup_open