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»

Отчёт по лабораторной работе №6 по курсу «Операционные системы»

Тема Системный вызов open()

Студент Козлова И.В.

Группа ИУ7-62Б

Оценка (баллы) _____

Преподаватель Рязанова Н.Ю.

Москва, 2022 г

Необходимые структуры

Версия ядра: 5.13.0

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

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;
};
```

```
#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;

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

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

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;
}
```

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

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

O_SYNC – файл открывается в режиме синхронного ввода-вывода, то есть все операции записи для соответствующего дескриптора файла блокируют вызывающий процесс до тех пор, пока данные не будут физически записаны

O_NOFOLLOW – если файл является символической ссылкой, то open вернёт ошибку

O_DIRECTORY – если файл не является каталогом, то open вернёт ошибку

O_LARGEFILE – позволяет открывать файлы, размер которых не может быть представлен типом off_t (long)**O_DSYNC –** операции записи в файл будут завершены в соответствии с требованиями целостности данных синхронизированного завершения ввода-вывода

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

O_TMPFILE – при наличии данного флага создаётся неименованный временный обычный 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

2 вариант схемы.

Схема алгоритма

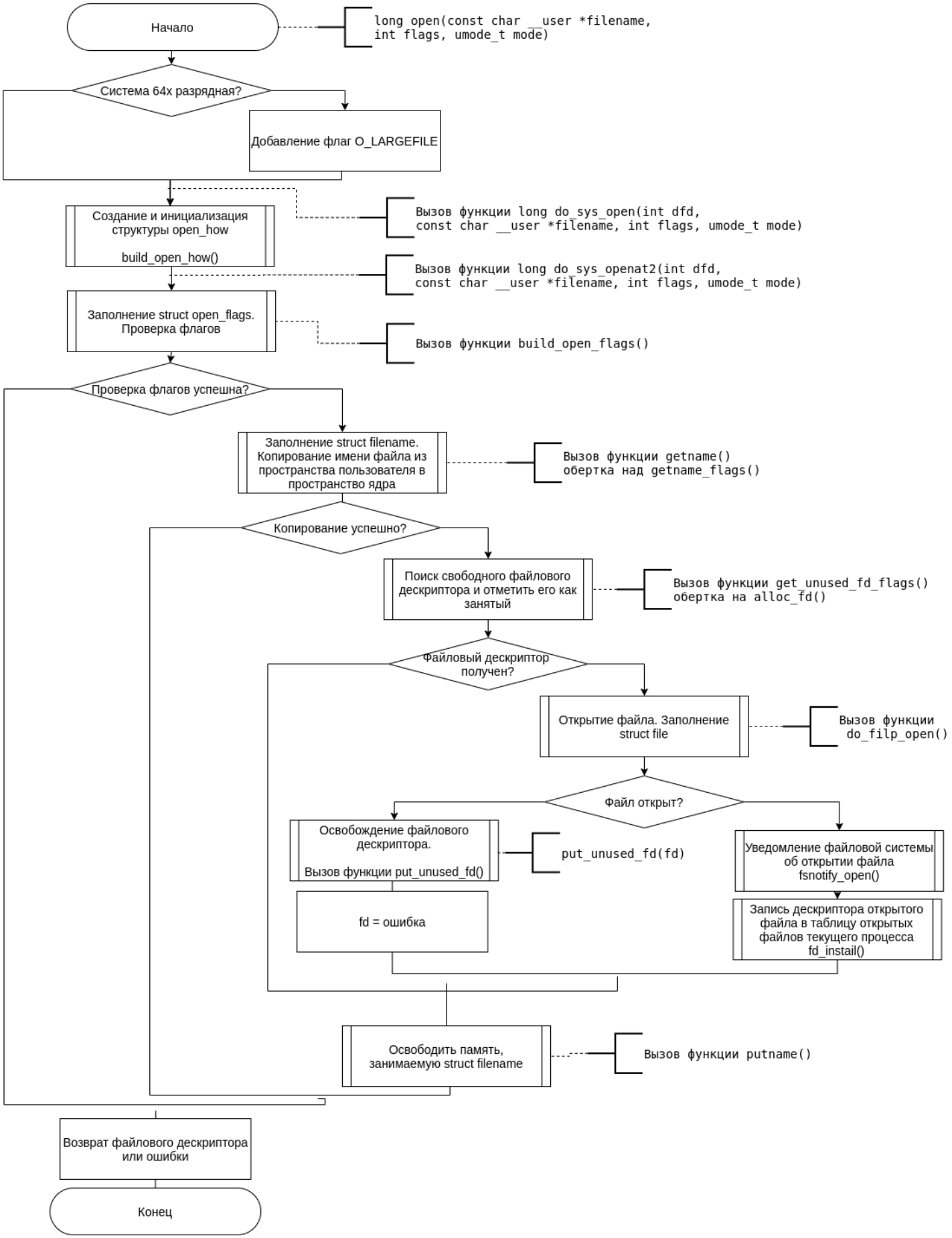


Схема работы алгоритма функции build_open_flags

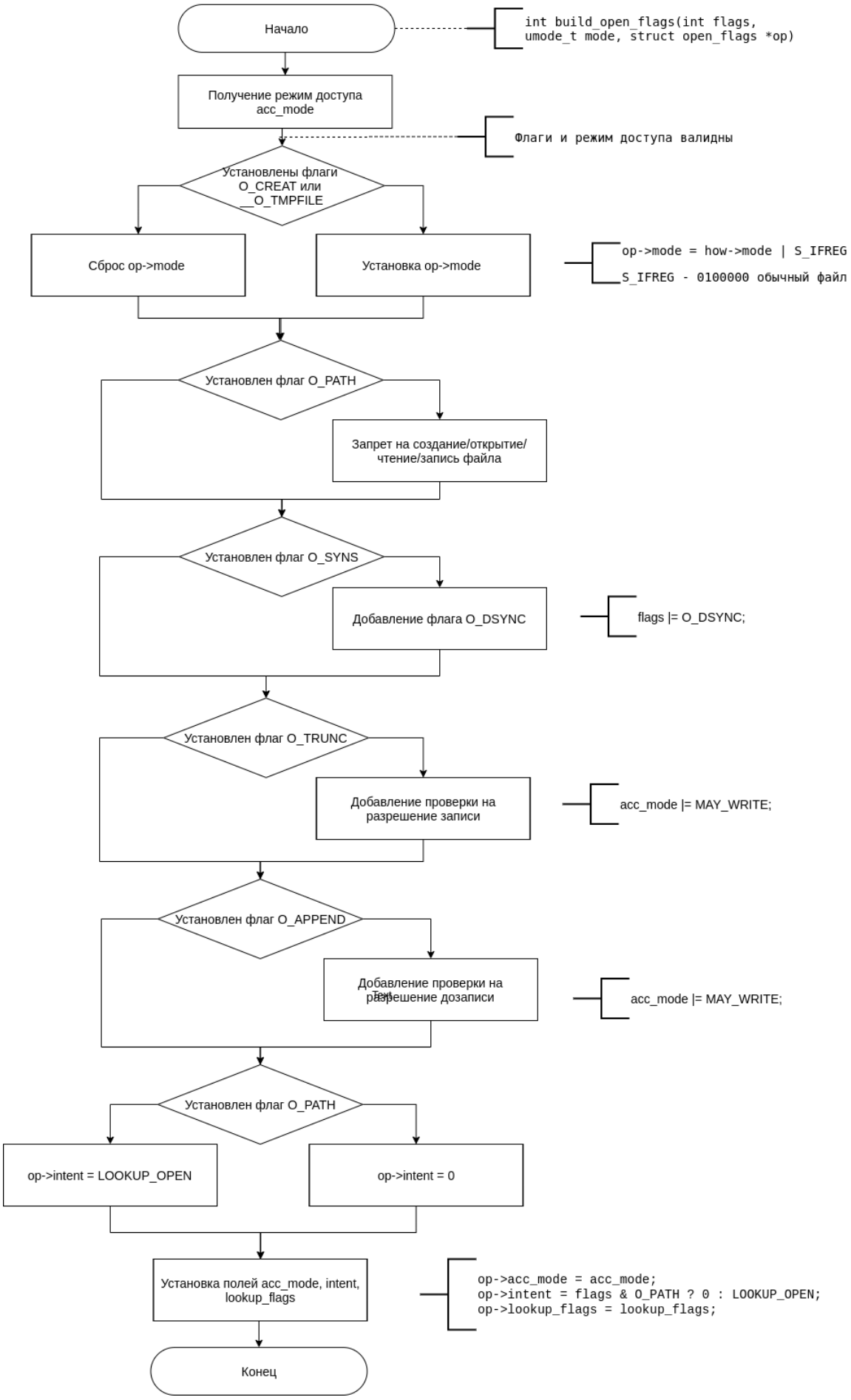


Схема работы алгоритма функции getname_flags

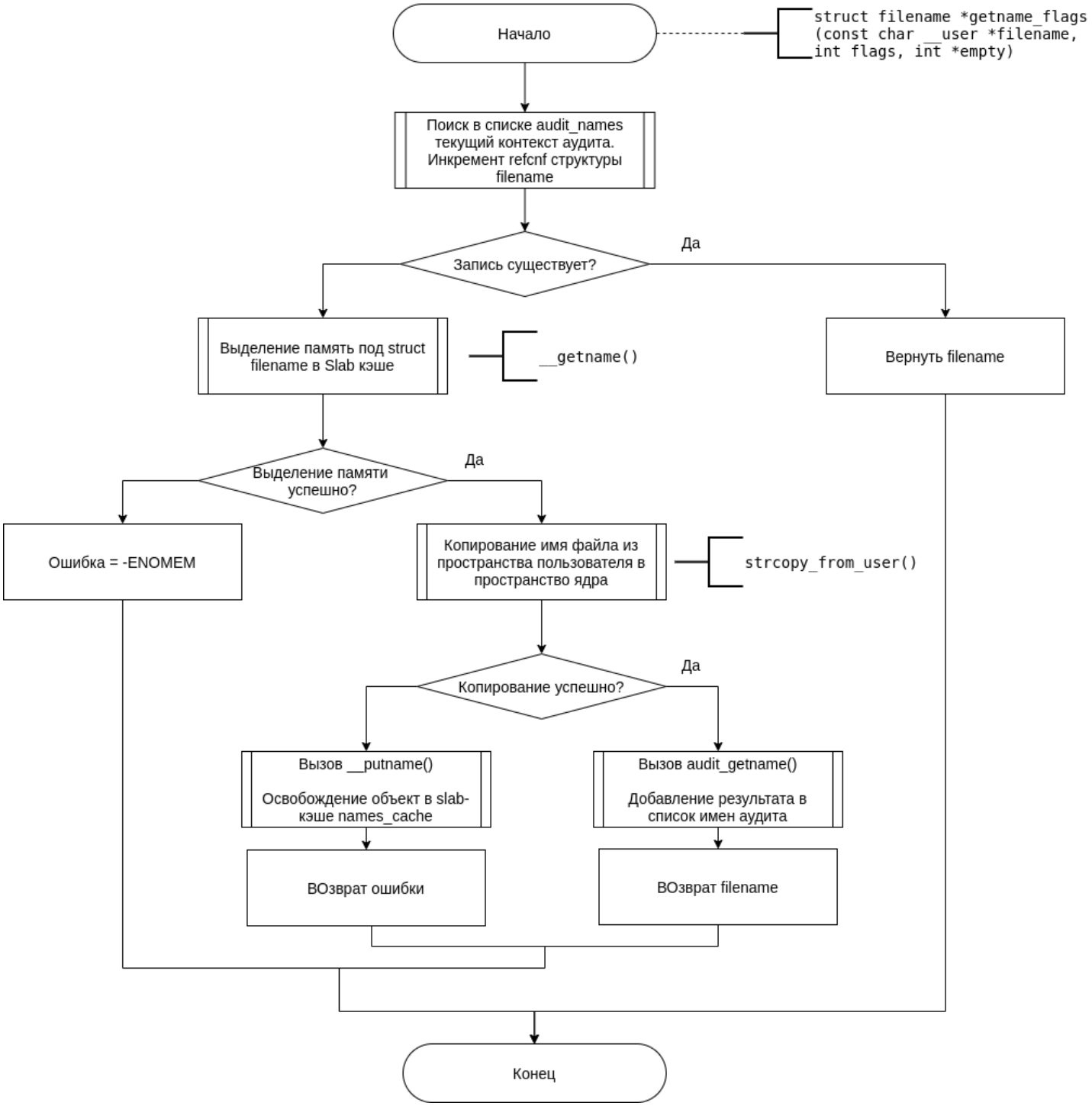


Схема работы алгоритма функции alloc_fd

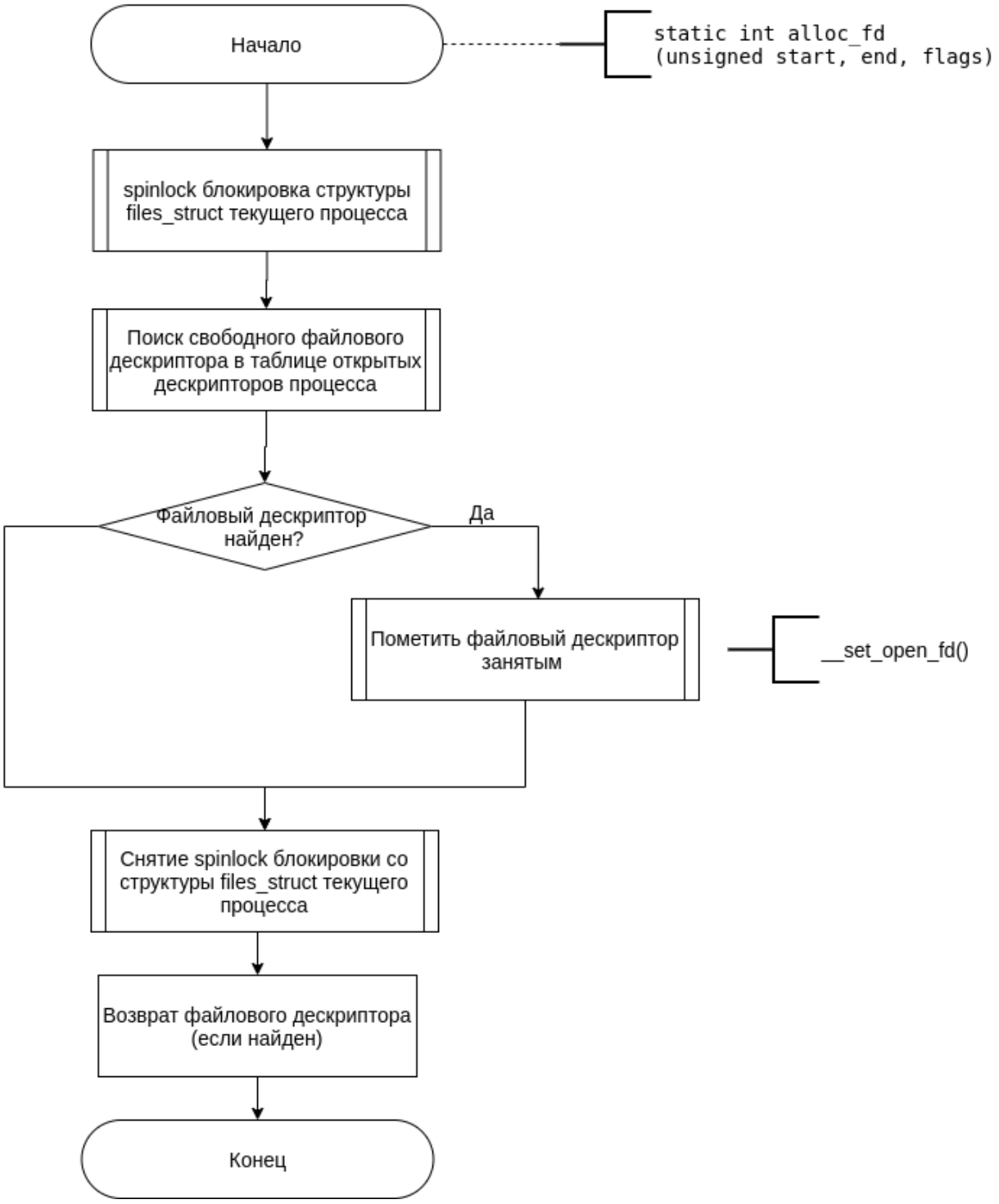


Схема работы алгоритма функции path_openat

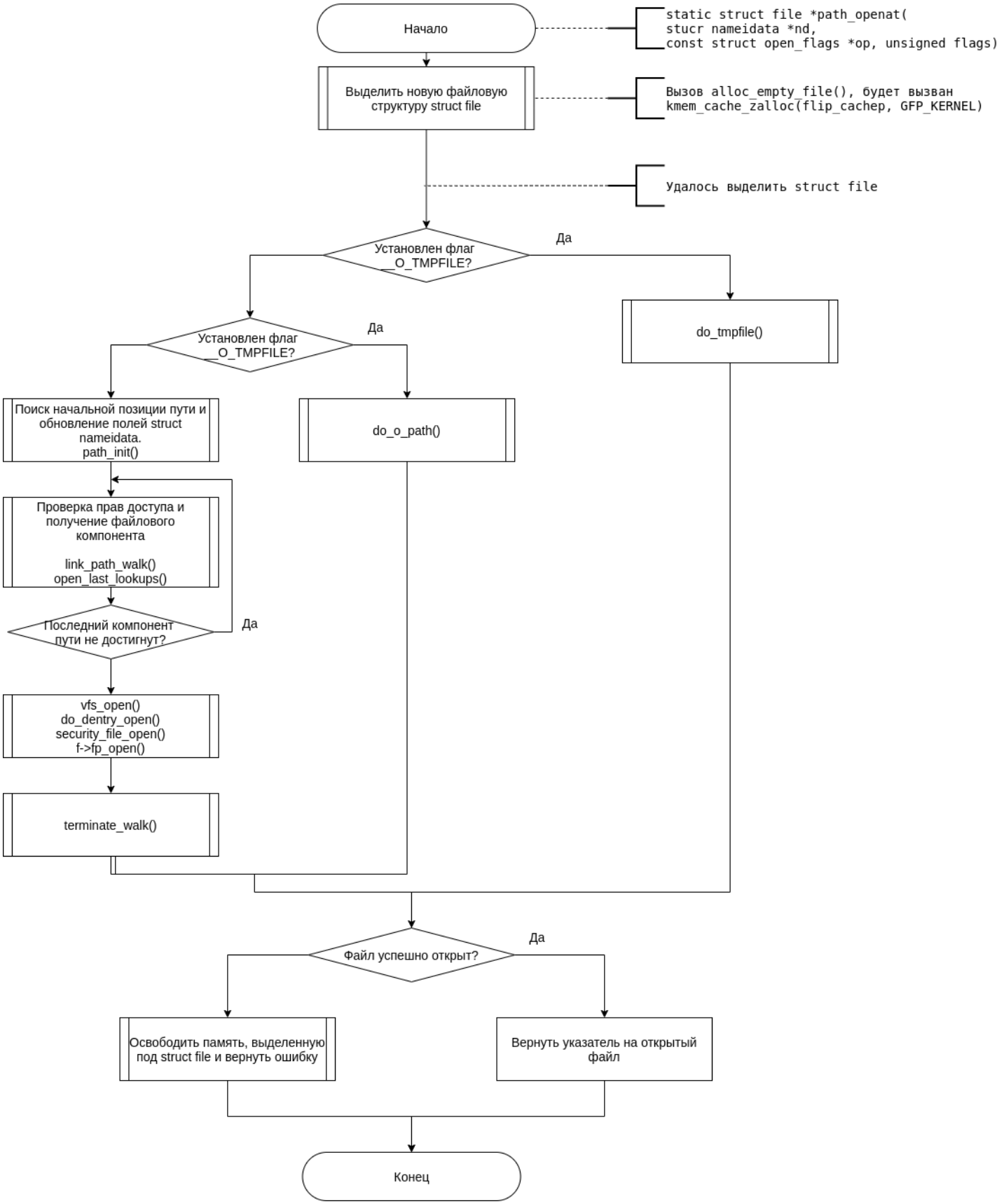


Схема работы алгоритма функции open_last_lookups

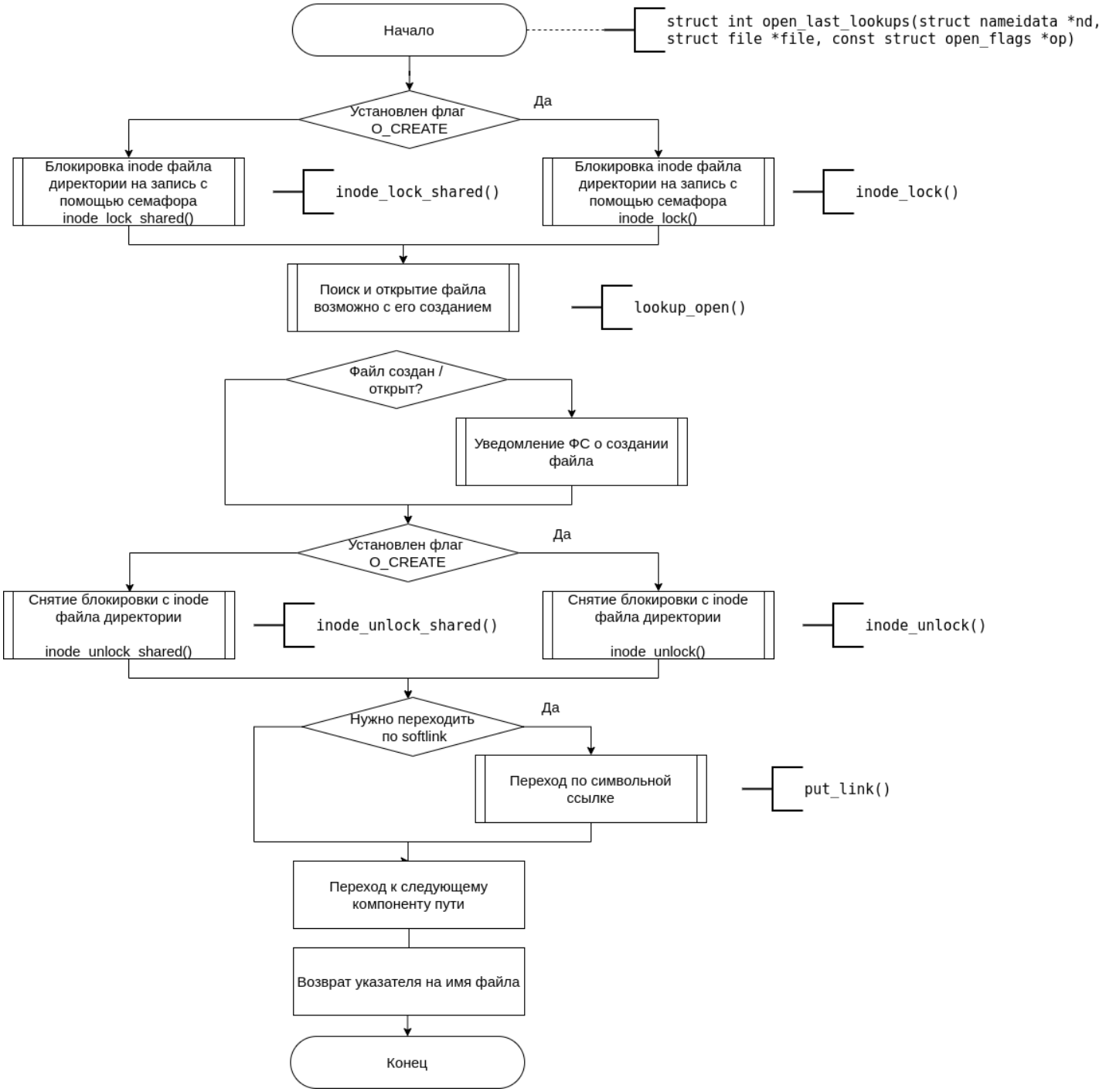


Схема работы алгоритма функции lookup_open

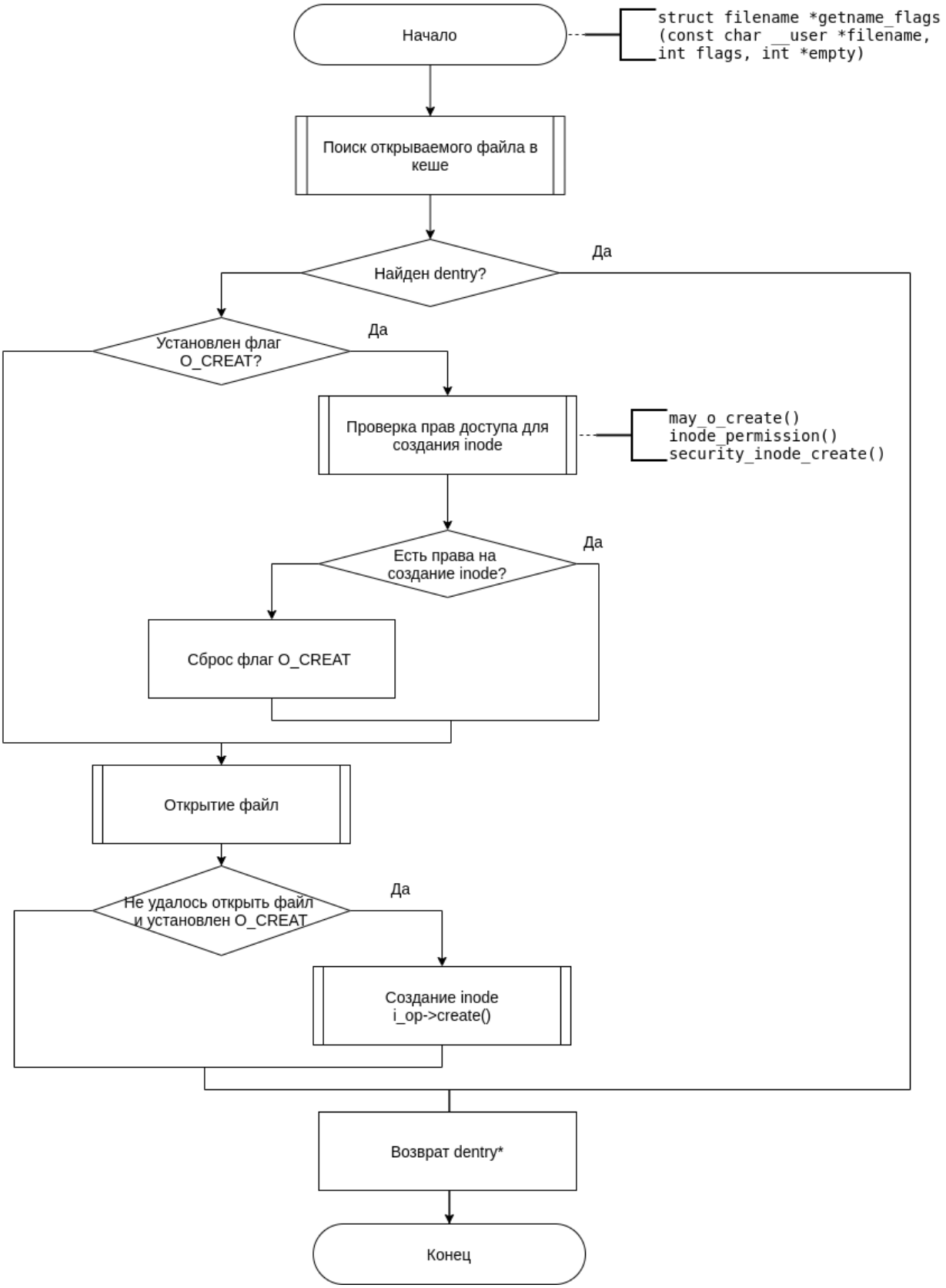
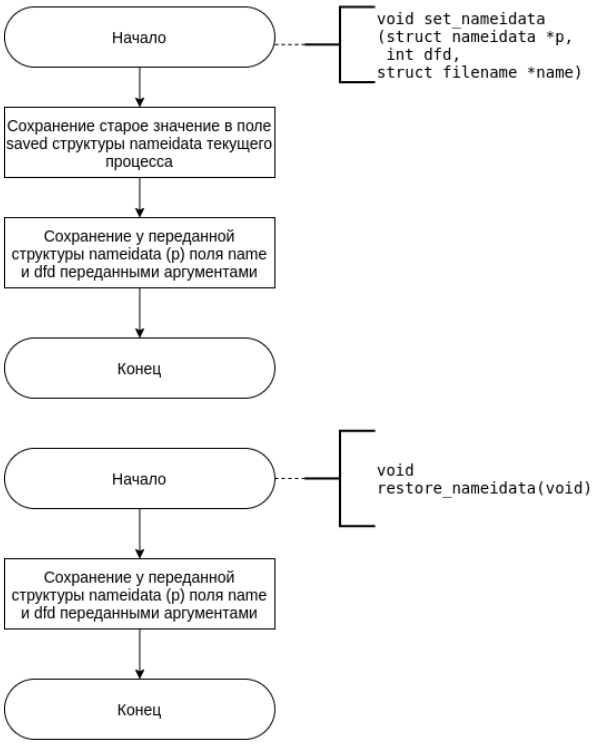


Схема работы алгоритма функции set_nameidata



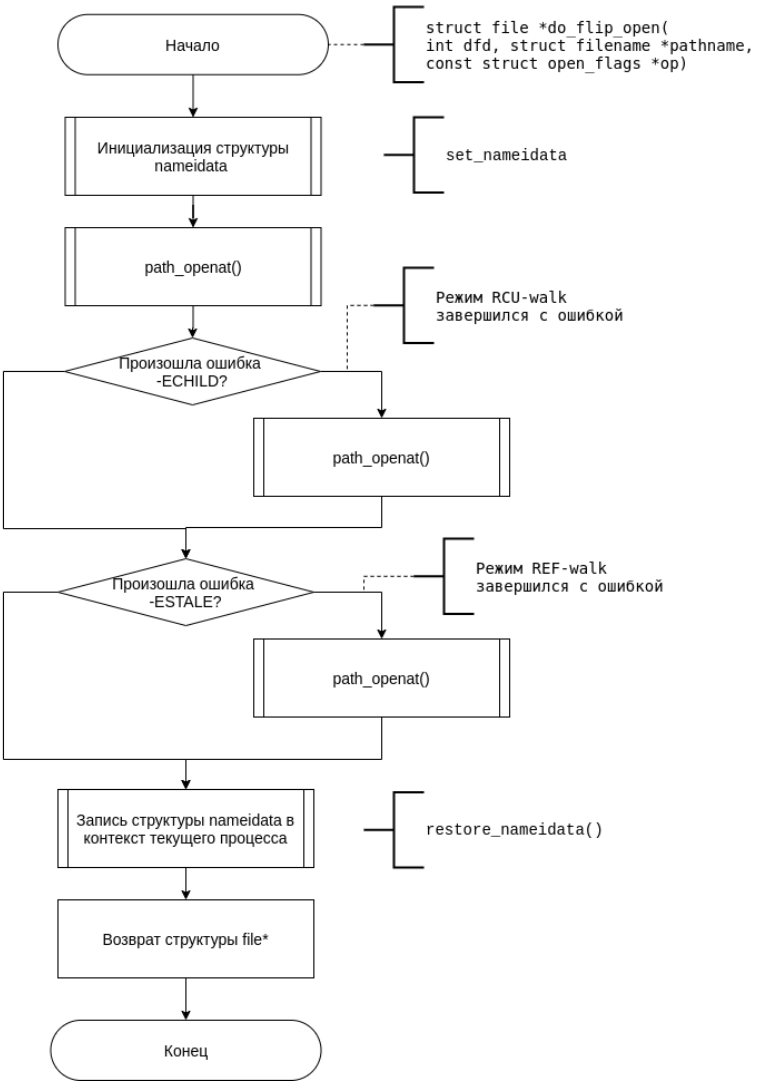


Схема работы алгоритма функции biuld_open_how()