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
по дисциплине «Операционные системы»

Тема: Системный вызов open()
Студент: Карпова Е. О.
Группа: ИУ7-62Б
Оценка (баллы): _____
Преподаватель: Рязанова Н. Ю.

Москва — 2023 г.

1. Системный вызов open()

Системный вызов open() открывает файл, определённый *pathname*. Если указанный файл не существует и в *flags* указан флаг O_CREAT, то open() может (необязательно) создать указанный файл с правами доступа, определёнными *mode*. Если флаг O_CREAT не указан, параметр *mode* игнорируется.

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

1.1. Возвращаемое значение

open() возвращает файловый дескриптор — небольшое неотрицательное целое число, которое является ссылкой на запись в системной таблице открытых файлов и индексом записи в таблице дескрипторов открытых файлов процесса. Этот дескриптор используется далее в системных вызовах read(), write(), lseek(), fcntl() и т.д. для ссылки на открытый файл. В случае успешного вызова будет возвращён наименьший файловый дескриптор, не связанный с открытым процессом файлом.

В случае ошибки возвращается -1 и устанавливается значение errno.

1.2. Параметры

pathname — имя файла в файловой системе. *flags* — режим открытия файла — один или несколько флагов открытия, объединенных оператором побитового ИЛИ.

Флаги:

1. O_RDONLY — открыть только для чтения;
2. O_WRONLY — открыть только для записи;
3. O_RDWR — открыть для чтения и записи.
4. O_EXEC — открыть только для выполнения (результат не определен при открытии директории);
5. O_SEARCH — открыть директорию только для поиска (результат не определен при использовании с файлами, не являющимися директорией);

6. O_APPEND — открыть в режиме добавления, перед каждой операцией записи файловый указатель будет устанавливаться в конец файла;

7. O_CLOEXEC — устанавливает флаг close-on-exec для нового файлового дескриптора (при вызове exec файл не будет оставаться открытым);

8. O_CREAT — если файл не существует, то он будет создан с правами доступа, определёнными *mode*;

9. O_DIRECTORY — вернуть ошибку, если файл не является каталогом;

10. 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);

11. O_EXCL — при использовании совместно с O_CREAT вернуть ошибку, если файл уже существует;

12. O_NOATIME — не обновлять время последнего доступа к файлу;

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

14. O_NOFOLLOW — вернуть ошибку, если часть пути является символической ссылкой;

15. O_NONBLOCK — файл открывается, по возможности, в режиме non-blocking, то есть никакие последующие операции над дескриптором файла не заставляют вызывающий процесс ждать;

16. O_RSYNC — операции записи должны выполняться на том же уровне, что и O_SYNC;

17. 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);

18. 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;

19. O_LARGEFILE — позволяет открывать файлы, размер которых не может быть представлен типом off_t (long). Для установки должен быть указан макрос _LARGEFILE64_SOU

20. O_TMPFILE — создать неименованный временный файл;

21. 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.

Если указан флаг O_CREAT, вызов open() создает новый файл с правами из *mode*:

1. S_IRWXU — права на чтение, запись, выполнение для пользователя;
2. S_IRUSR — права на чтение для пользователя;
3. S_IWUSR — права на запись для пользователя;
4. S_IXUSR — права на выполнение для пользователя;
5. S_IRWXG — права на чтение, запись, выполнение для группы;
6. S_IRGRP — права на чтение для группы;
7. S_IWGRP — права на запись для группы;
8. S_IXGRP — права на выполнение для группы;
9. S_IRWXO — права на чтение, запись, выполнение для остальных;
10. S_IROTH — права на чтение для остальных;
11. S_IWOTH — права на запись для остальных;
12. S_IXOTH — права на выполнение для остальных;
13. S_ISUID — бит set-user-ID;
14. S_ISGID — бит set-group-ID;
15. S_ISVTX — «липкий» бит.

2. Схема выполнения open()

Версия ядра: 6.3.5

2.1. open()

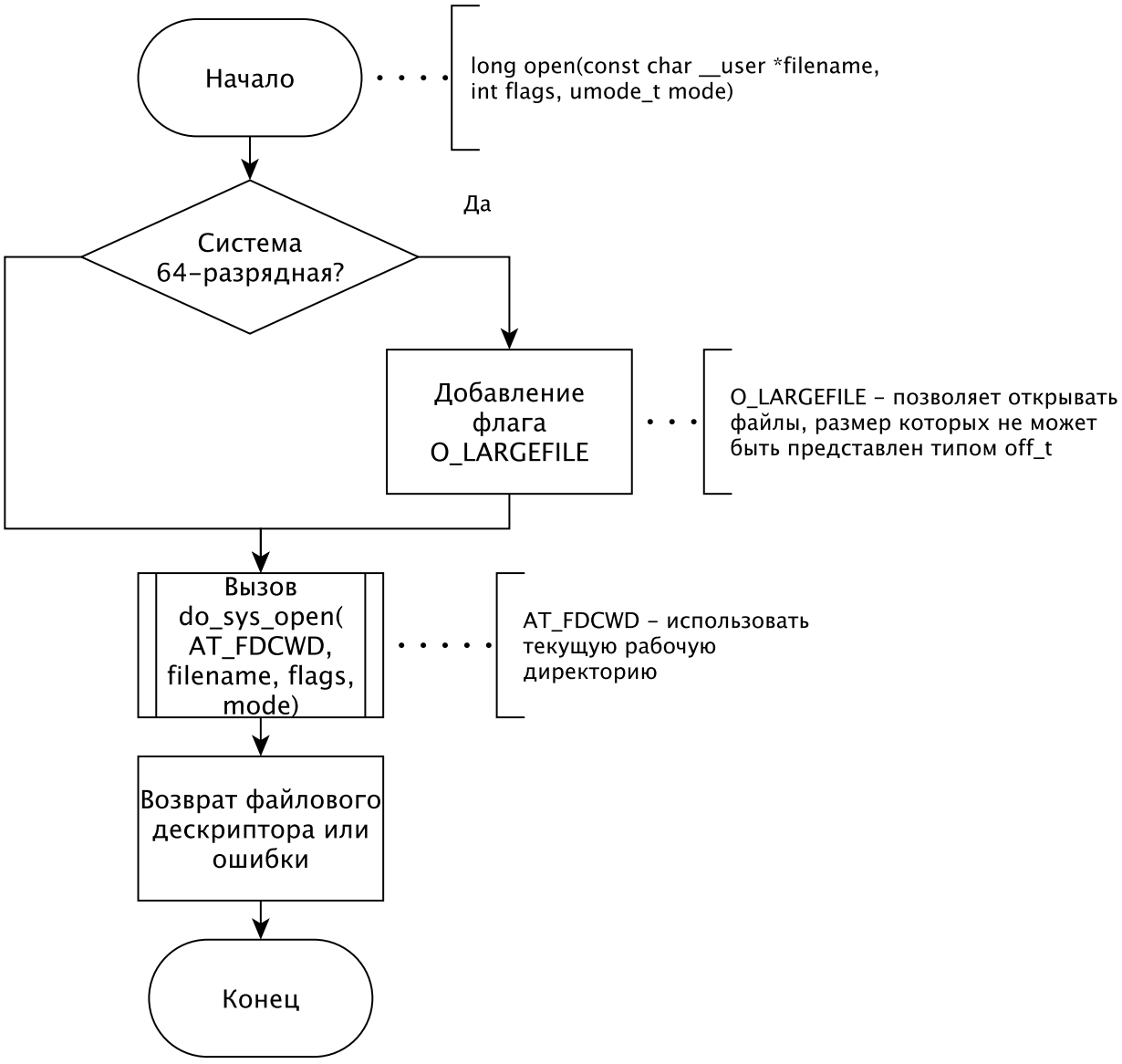

Рисунок 2.1 — open()

2.2. do_sys_open()

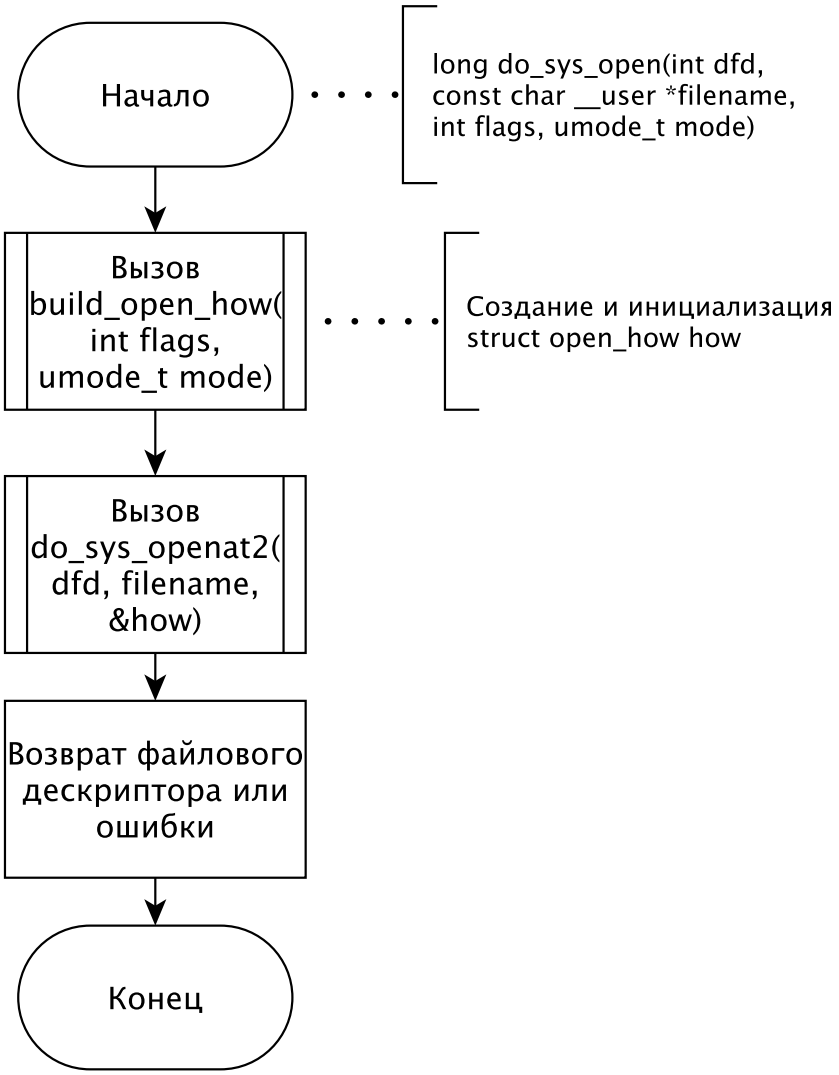

Рисунок 2.2 — do_sys_open()

2.3. build_open_how()

```
1  struct open_how {
2  __u64 flags;
3  __u64 mode;
4  __u64 resolve; };
```

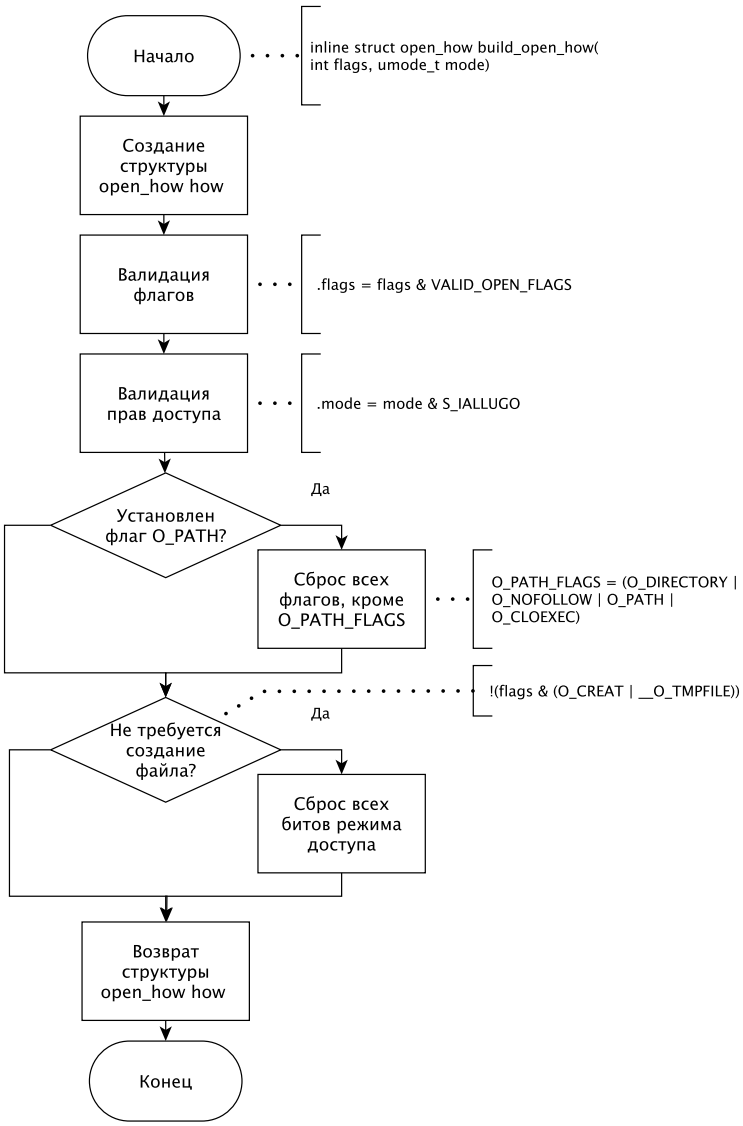

Рисунок 2.3 — build_open_how()

```
1  #define S_IRWXUGO (S_IRWXU|S_IRWXG|S_IRWXO)
2  #define S_IALLUGO (S_ISUID|S_ISGID|S_ISVTX|S_IRWXUGO)
3
4  #define VALID_OPEN_FLAGS \
5   (O_RDONLY | O_WRONLY | O_RDWR | O_CREAT | O_EXCL | O_NOCTTY | O_TRUNC |
        \
6    O_APPEND | O_NDELAY | O_NONBLOCK | __O_SYNC | O_DSYNC | \
7    FASYNC | O_DIRECT | O_LARGEFILE | O_DIRECTORY | O_NOFOLLOW | \
8    O_NOATIME | O_CLOEXEC | O_PATH | __O_TMPFILE)
```

2.4. do_sys_openat2()

```
1  struct open_flags {
2  int open_flag;
3  umode_t mode;
4  int acc_mode;
5  int intent;
6  int lookup_flags;
7 };
8
9 struct filename {
10  const char    *name;  /* pointer to actual string */
11  const __user char *uptr;  /* original userland pointer */
12  int      refcnt;
13  struct audit_names  *aname;
14  const char    iname[];
15 };
16
17 struct file {
18  union {
19    struct llist_node f_llist;
20    struct rcu_head   f_rcuhead;
21    unsigned int    f_iocb_flags;
22  };
23  struct path   f_path;
24  struct inode    *f_inode; /* cached value */
25  const struct file_operations  *f_op;
26
27  /*
28   * Protects f_ep, f_flags.
29   * Must not be taken from IRQ context.
30   */
31  spinlock_t    f_lock;
32  atomic_long_t   f_count;
33  unsigned int    f_flags;
34  fmode_t     f_mode;
35  struct mutex    f_pos_lock;
36  loff_t      f_pos;
```

```
37    struct fown_struct  f_owner;
38    const struct cred *f_cred;
39    struct file_ra_state  f_ra;
40
41    u64     f_version;
42  #ifdef CONFIG_SECURITY
43    void      *f_security;
44  #endif
45    /* needed for tty driver, and maybe others */
46    void      *private_data;
47
48  #ifdef CONFIG_EPOLL
49    /* Used by fs/eventpoll.c to link all the hooks to this file */
50    struct hlist_head *f_ep;
51  #endif /* #ifdef CONFIG_EPOLL */
52    struct address_space  *f_mapping;
53    errseq_t    f_wb_err;
54    errseq_t    f_sb_err; /* for syncfs */
55  } __randomize_layout
56    __attribute__((aligned(4)));  /* lest something weird decides that 2 is
        OK */
```

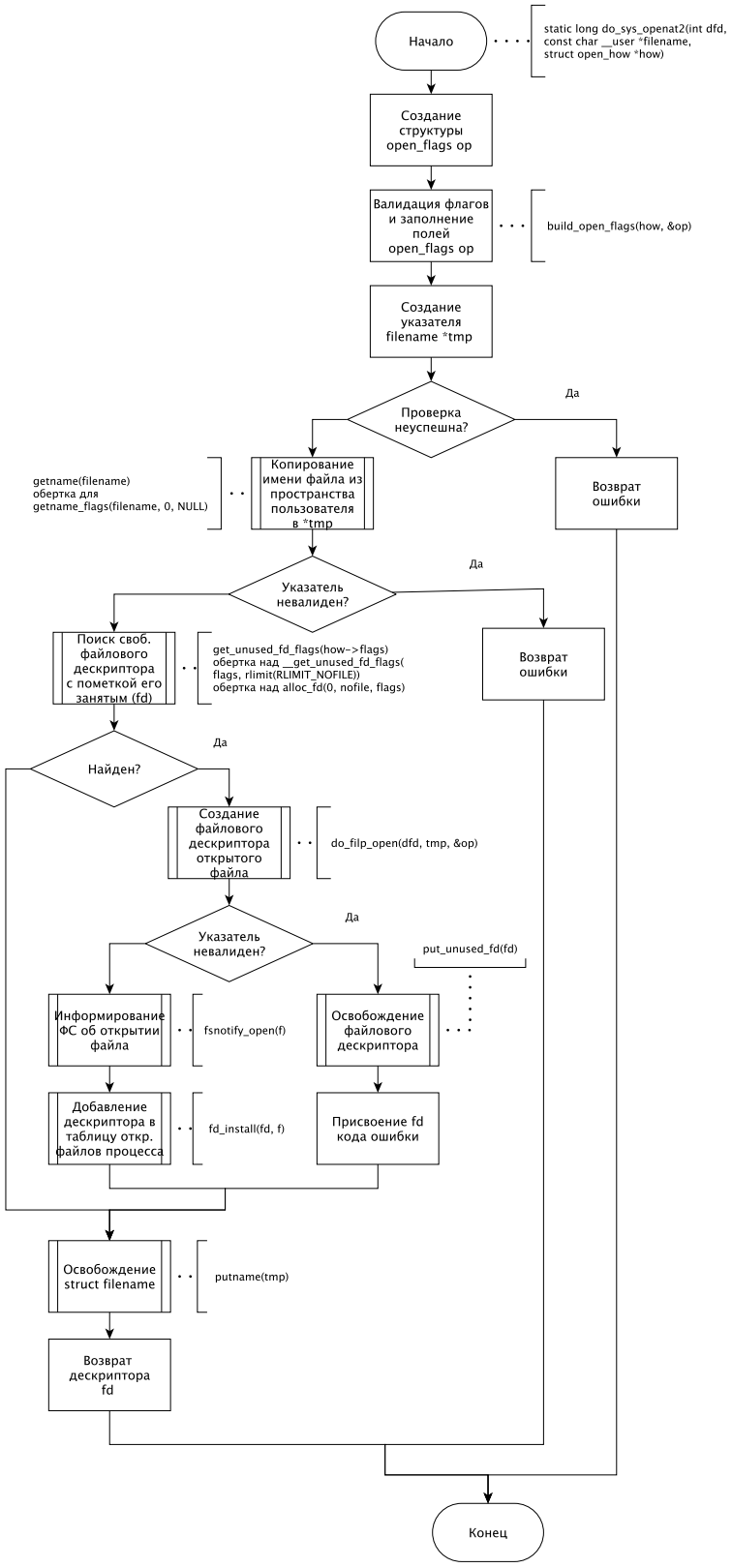

Рисунок 2.4 — do_sys_openat2()

2.5. build_open_flags()

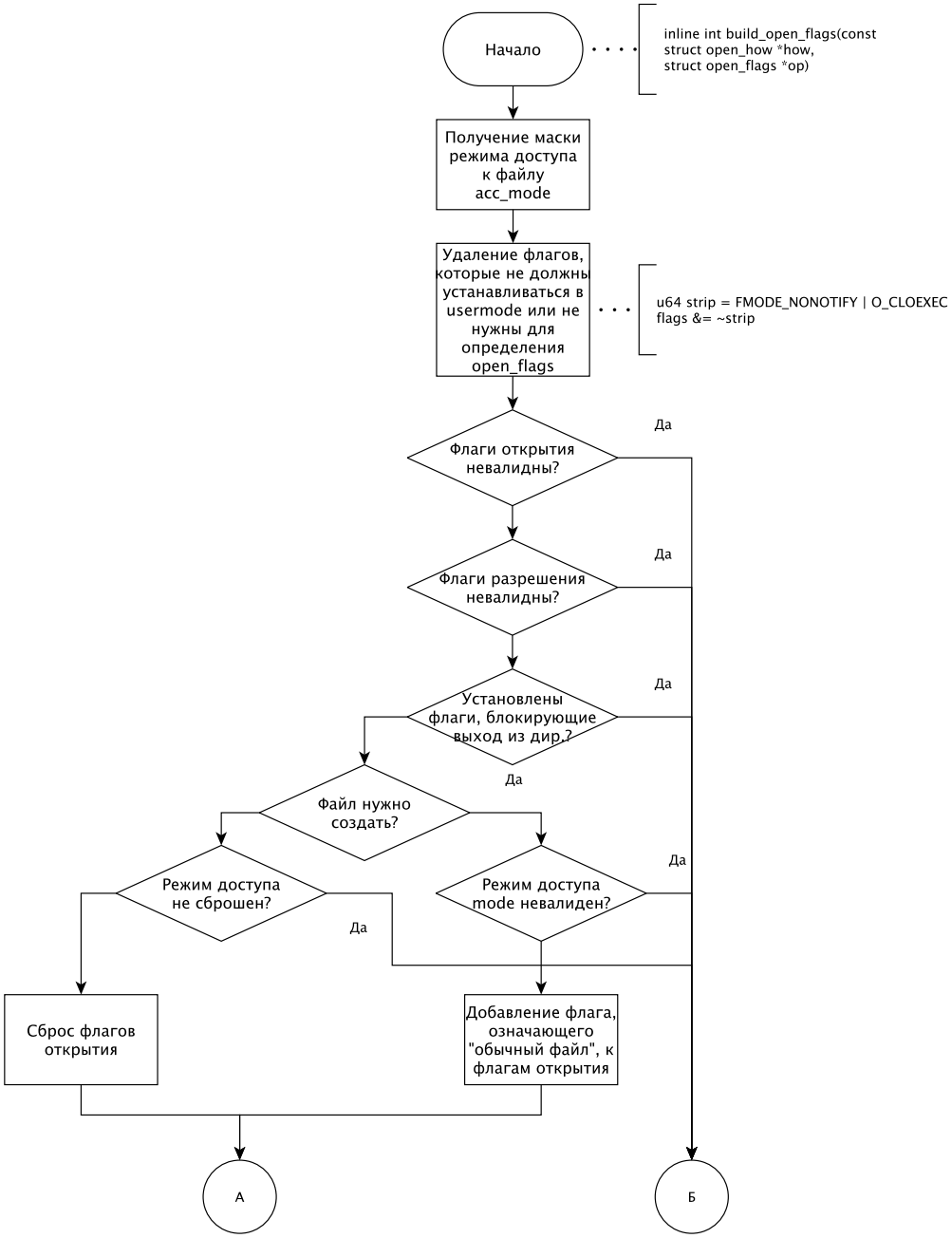

Рисунок 2.5 — build_open_flags()

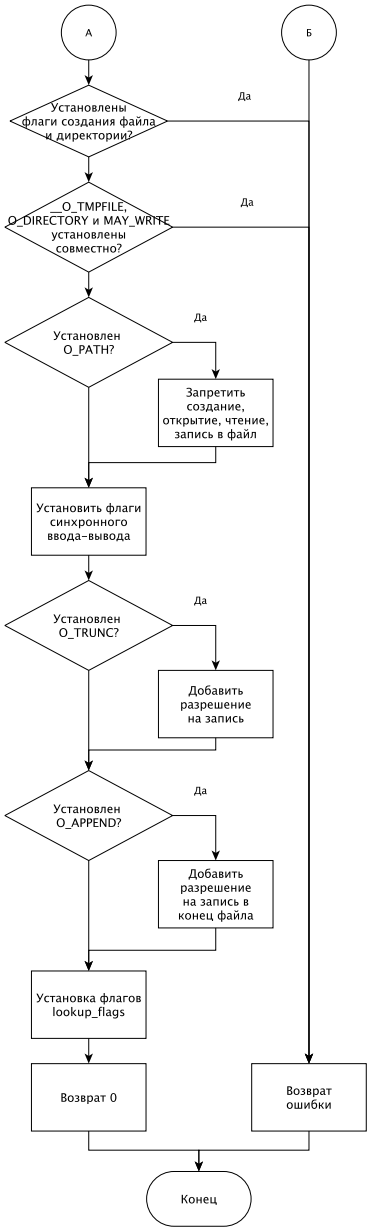

Рисунок 2.6 — build_open_flags()

2.6. getname_flags()

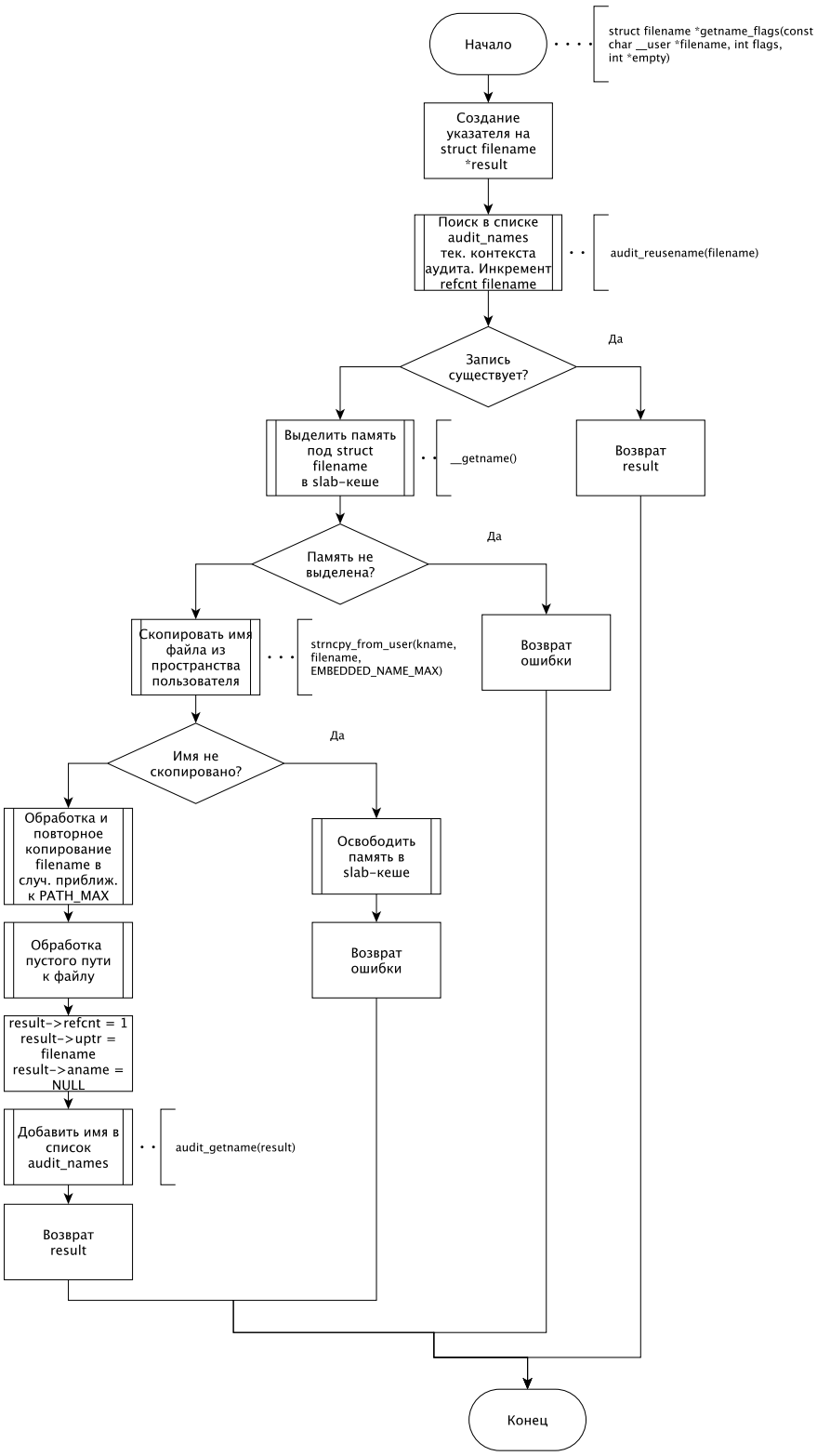

Рисунок 2.7 — getname_flags()

2.7. __alloc_fd()

```
1   /*
2  * Open file table structure
3  */
4 struct files_struct {
5   /*
6    * read mostly part
7    */
8   atomic_t count;
9   bool resize_in_progress;
10  wait_queue_head_t resize_wait;
11
12  struct fdtable __rcu *fdt;
13  struct fdtable fdtab;
14  /*
15   * written part on a separate cache line in SMP
16   */
17  spinlock_t file_lock _____cacheline_aligned_in_smp;
18  unsigned int next_fd;
19  unsigned long close_on_exec_init[1];
20  unsigned long open_fds_init[1];
21  unsigned long full_fds_bits_init[1];
22  struct file __rcu * fd_array[NR_OPEN_DEFAULT];
23 };
24
25 struct fdtable {
26  unsigned int max_fds;
27  struct file __rcu **fd;      /* current fd array */
28  unsigned long *close_on_exec;
29  unsigned long *open_fds;
30  unsigned long *full_fds_bits;
31  struct rcu_head rcu;
32 };
```

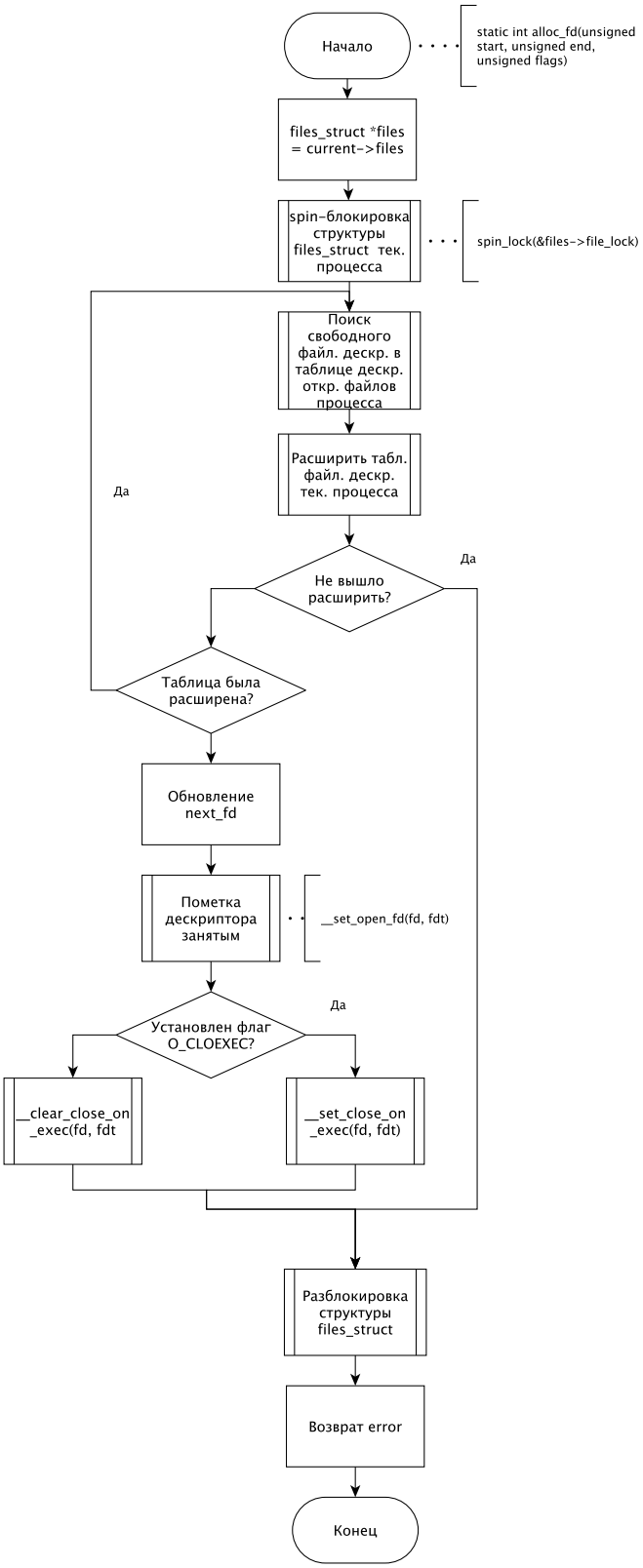

Рисунок 2.8 — __alloc_fd()

2.8. do_filp_open()

```
1   #define EMBEDDED_LEVELS 2
2  struct nameidata {
3    struct path path;
4    struct qstr last;
5    struct path root;
6    struct inode  *inode; /* path.dentry.d_inode */
7    unsigned int  flags, state;
8    unsigned  seq, next_seq, m_seq, r_seq;
9    int   last_type;
10   unsigned  depth;
11   int   total_link_count;
12   struct saved {
13     struct path link;
14     struct delayed_call done;
15     const char *name;
16     unsigned seq;
17   } *stack, internal[EMBEDDED_LEVELS];
18   struct filename *name;
19   struct nameidata *saved;
20   unsigned  root_seq;
21   int   dfd;
22   vfsuid_t  dir_vfsuid;
23   umode_t   dir_mode;
24 } __randomize_layout;
25
26 #define LOOKUP_REVAL    0x0020  /* tell ->d_revalidate() to trust no cache
       */
27 #define LOOKUP_RCU    0x0040  /* RCU pathwalk mode; semi-internal */
```

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.

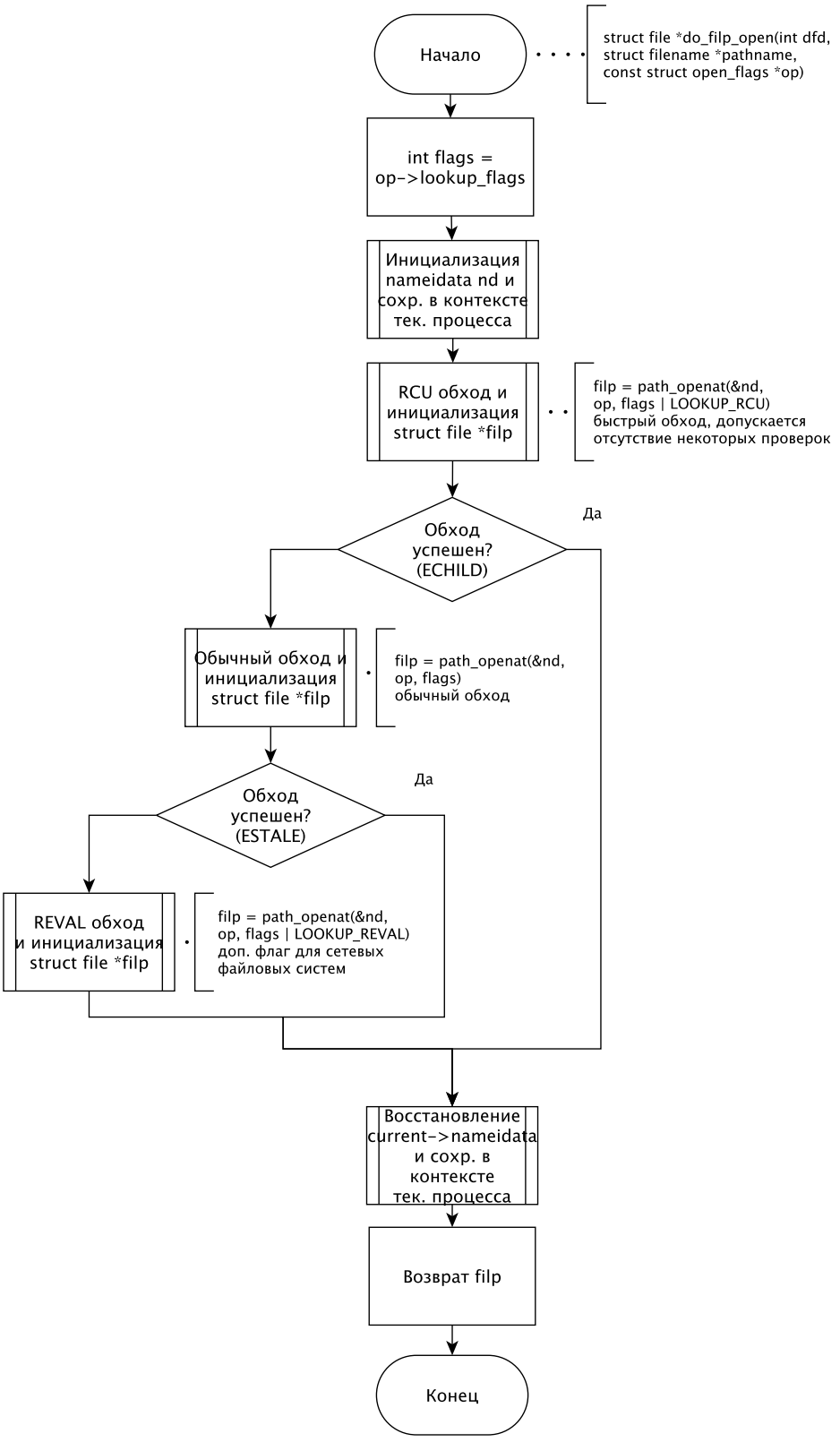

Рисунок 2.9 — do_filp_open()

2.9. set_nameidata() и restore_nameidata()

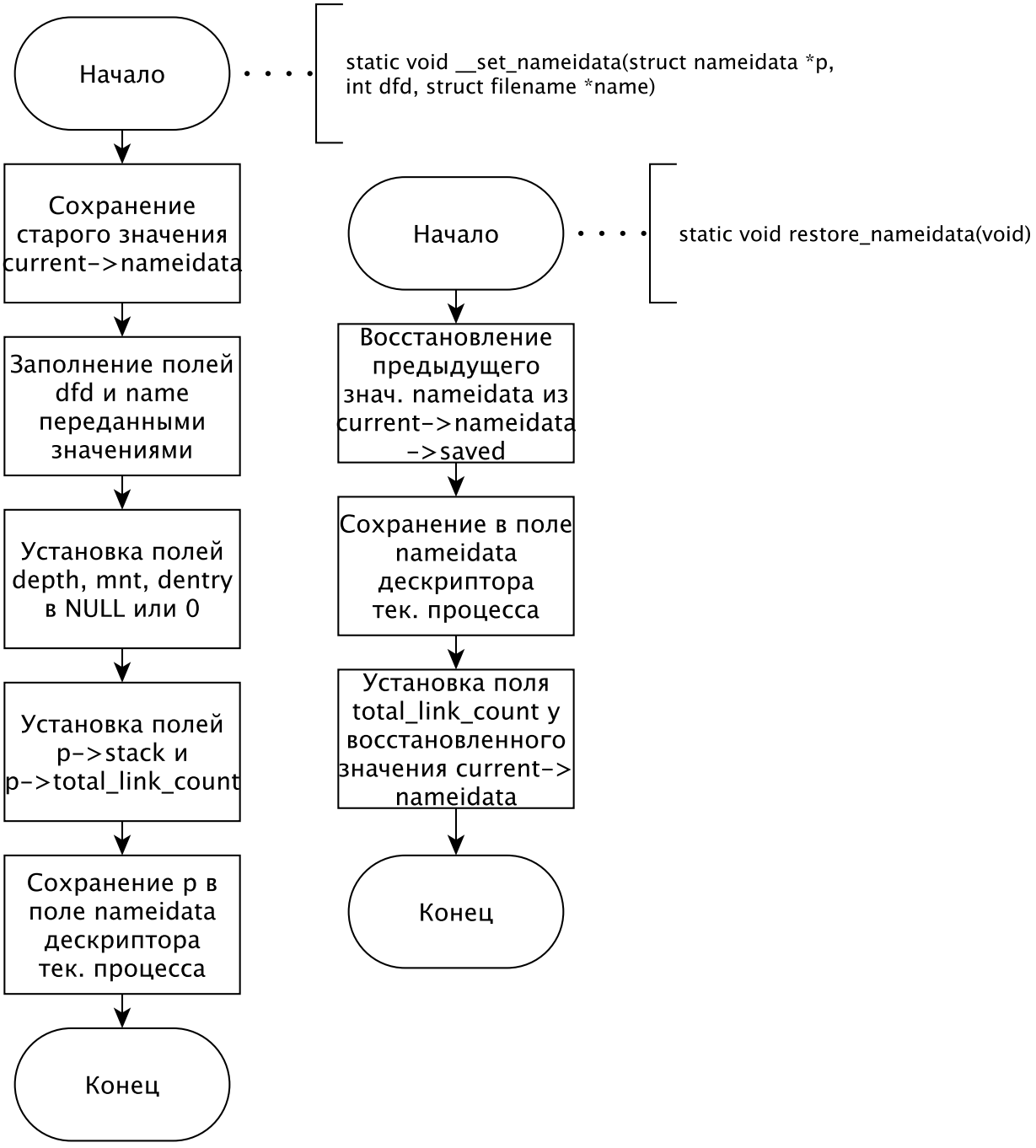

Рисунок 2.10 — set_nameidata() и restore_nameidata()

2.10. path_openat()

```
1  struct dentry {
2  /* RCU lookup touched fields */
3  unsigned int d_flags;   /* protected by d_lock */
4  seqcount_spinlock_t d_seq;  /* per dentry seqlock */
5  struct hlist_bl_node d_hash;  /* lookup hash list */
6  struct dentry *d_parent;  /* parent directory */
7  struct qstr d_name;
8  struct inode *d_inode;     /* Where the name belongs to - NULL is
9           * negative */
10 unsigned char d_iname[DNAME_INLINE_LEN];  /* small names */
11
12 /* Ref lookup also touches following */
13 struct lockref d_lockref; /* per-dentry lock and refcount */
14 const struct dentry_operations *d_op;
15 struct super_block *d_sb; /* The root of the dentry tree */
16 unsigned long d_time;   /* used by d_revalidate */
17 void *d_fsdata;     /* fs-specific data */
18
19 union {
20   struct list_head d_lru;   /* LRU list */
21   wait_queue_head_t *d_wait;  /* in-lookup ones only */
22 };
23 struct list_head d_child; /* child of parent list */
24 struct list_head d_subdirs; /* our children */
25 /*
26  * d_alias and d_rcu can share memory
27  */
28 union {
29   struct hlist_node d_alias;  /* inode alias list */
30   struct hlist_bl_node d_in_lookup_hash;  /* only for in-lookup ones */
31   struct rcu_head d_rcu;
32 } d_u;
33 } __randomize_layout;
```

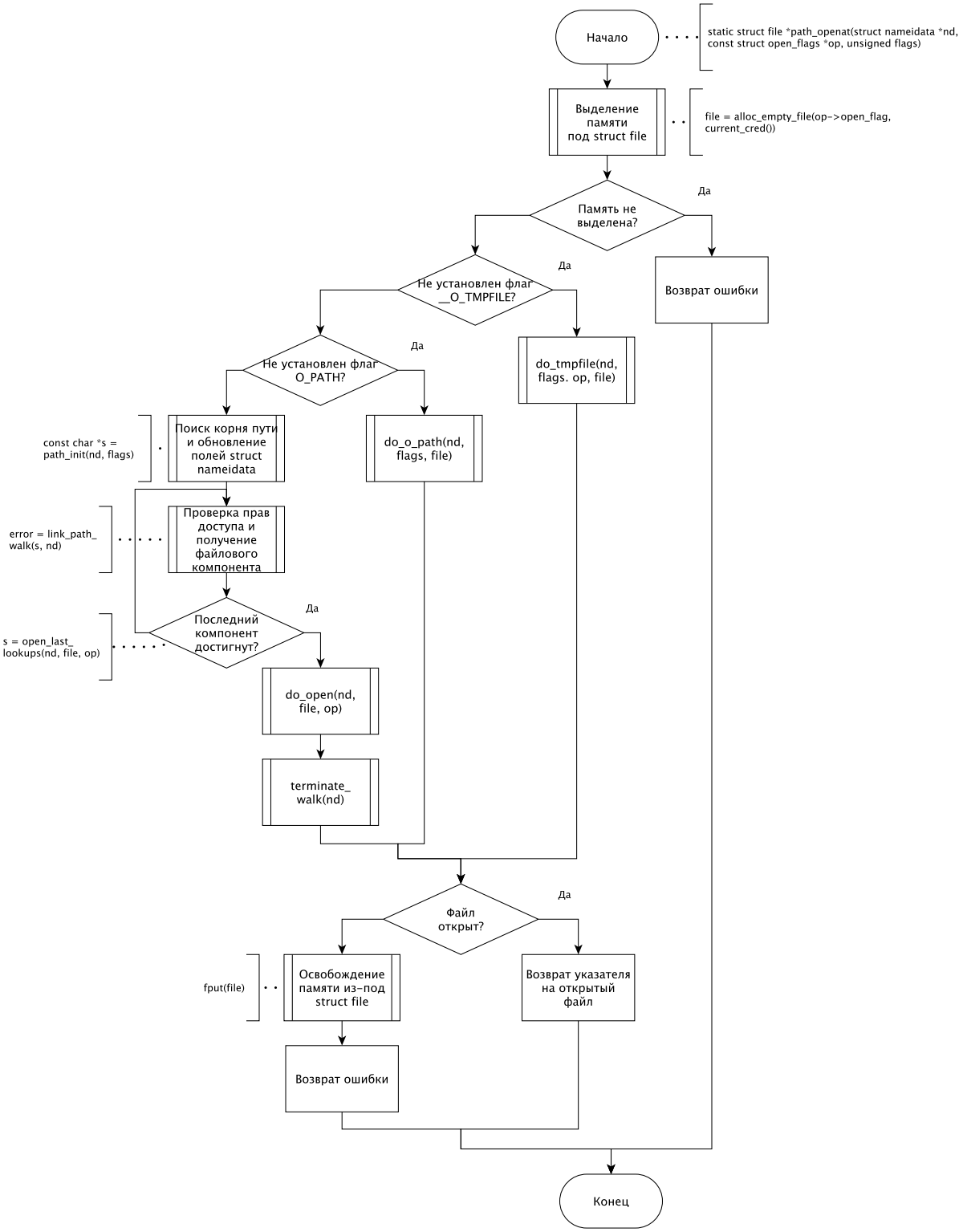

Рисунок 2.11 — path_openat()

2.11. open_last_lookups()

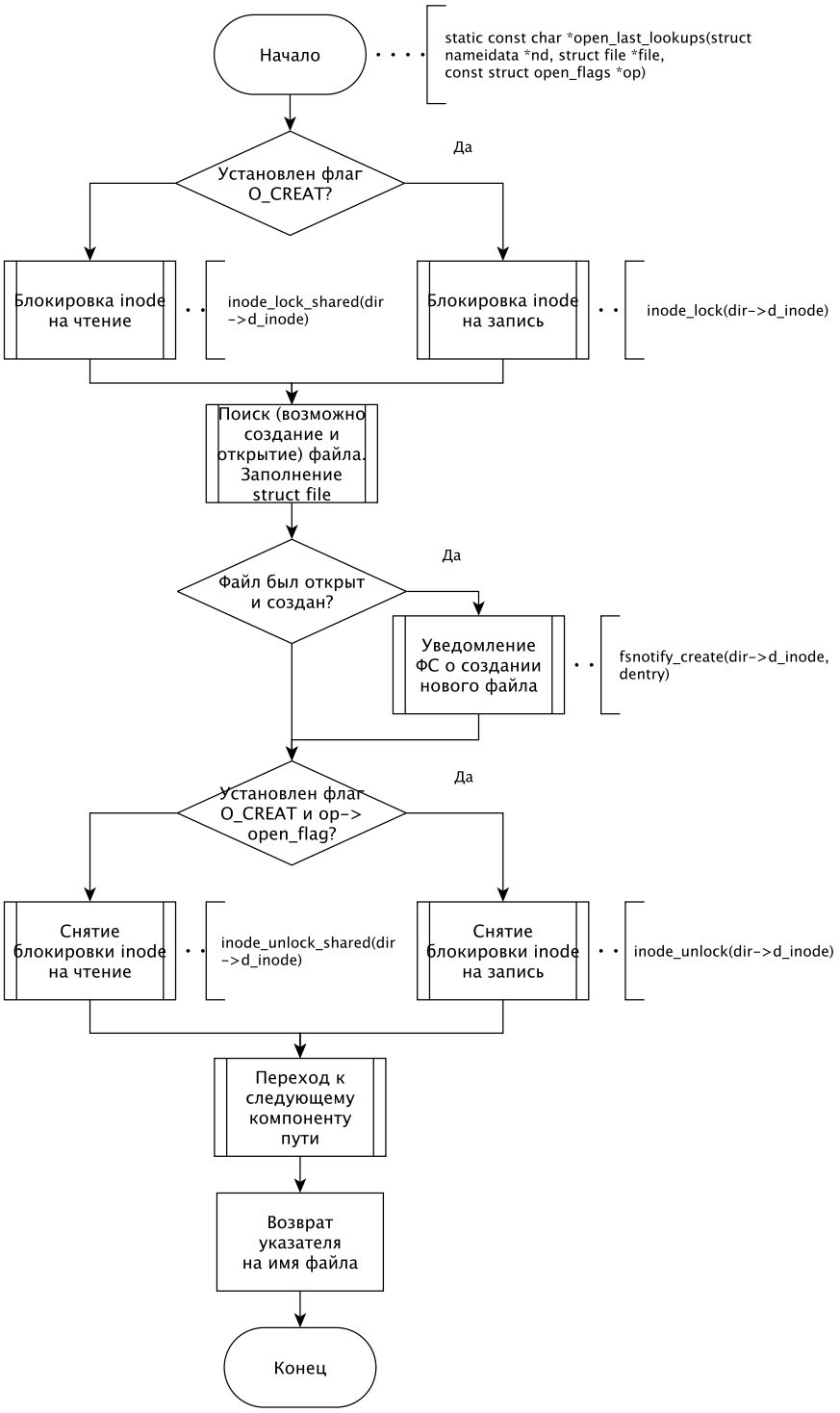

Рисунок 2.12 — open_last_lookups()

2.12. lookup_open()

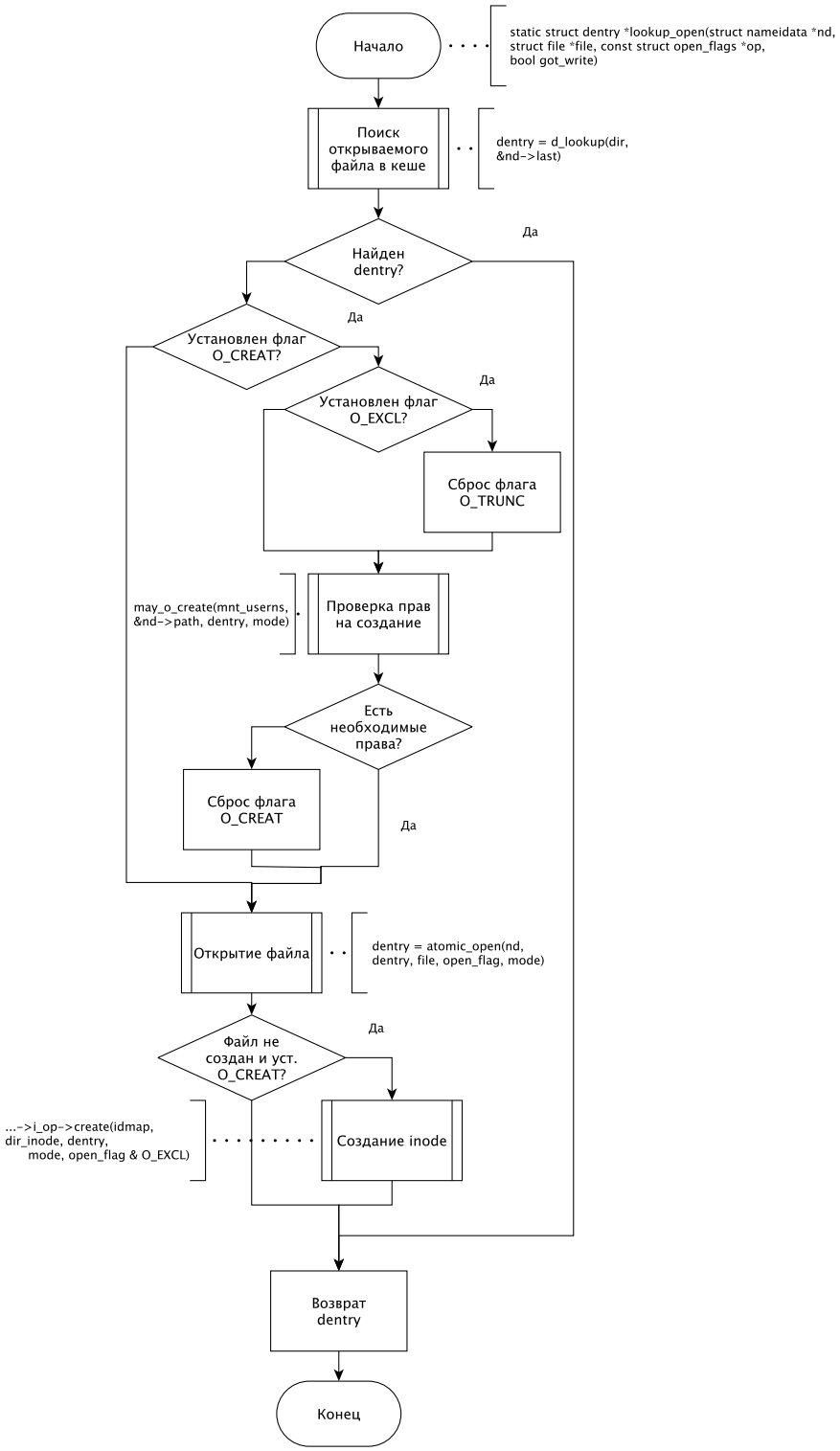

Рисунок 2.13 — lookup_open()

2.13. do_open()

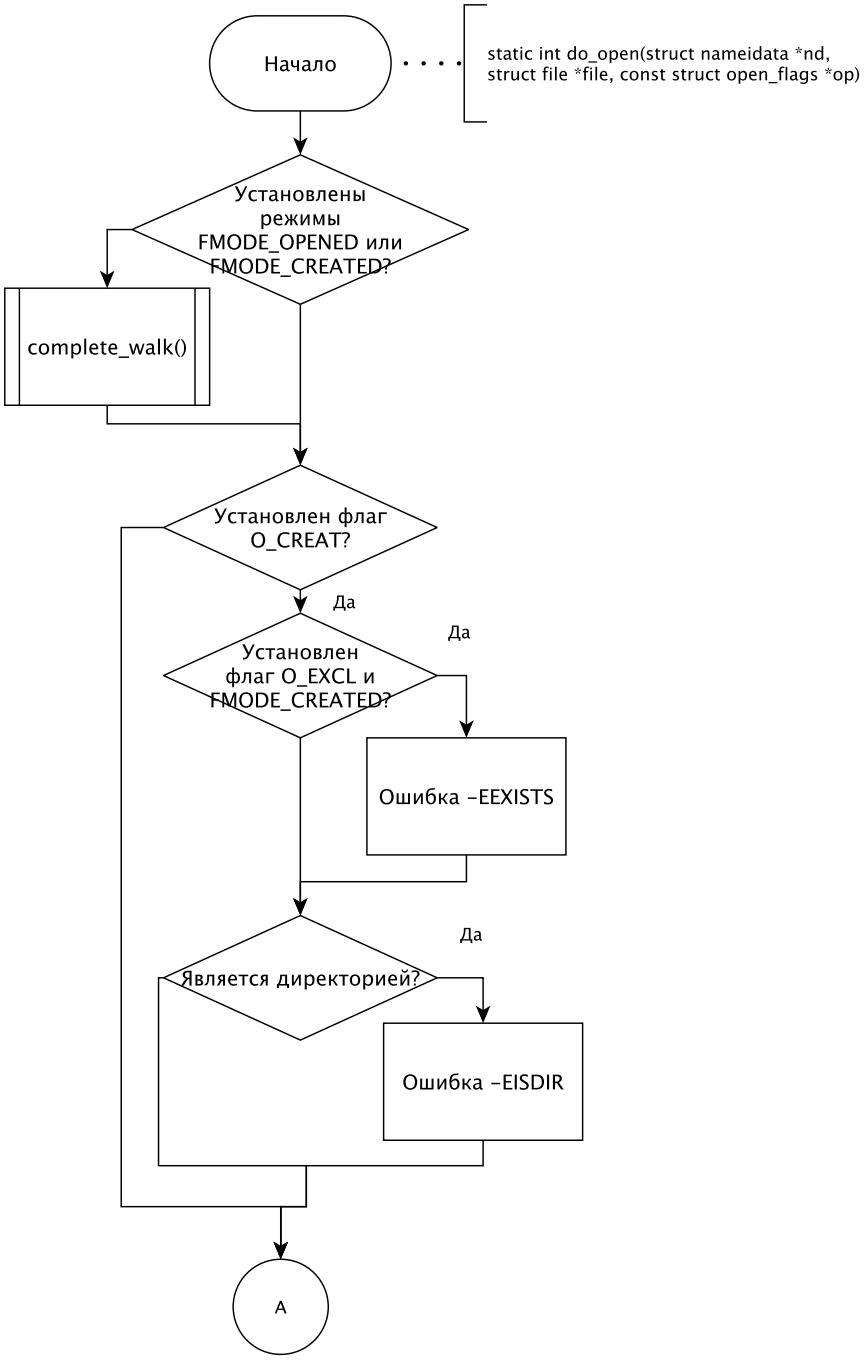

Рисунок 2.14 — do_open()

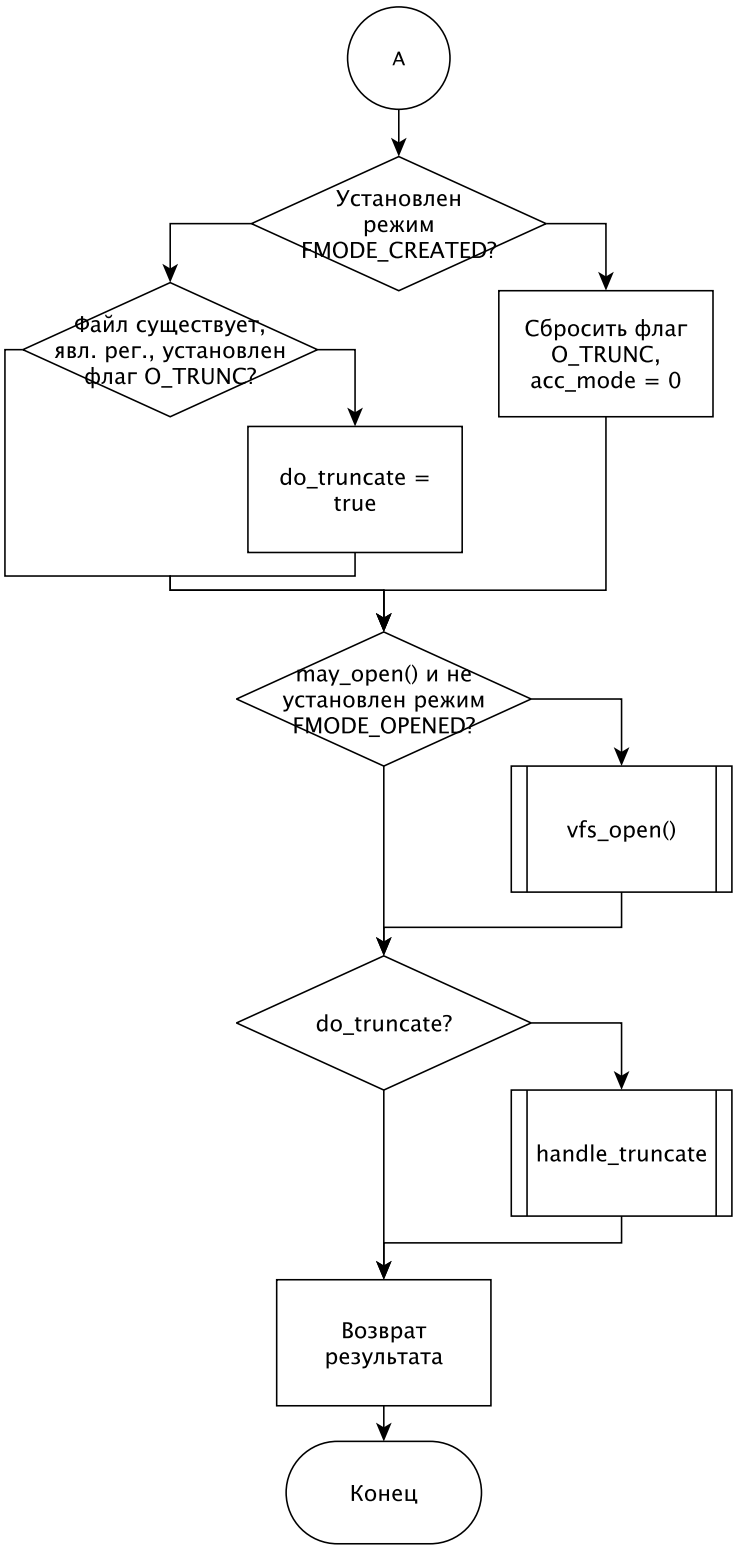

Рисунок 2.15 — do_open()