



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 ИУ7 «Программное обеспечение ЭВМ и информационные технологии»

---

## РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

### *К КУРСОВОЙ РАБОТЕ*

### *НА ТЕМУ:*

«Загружаемый модуль ядра для защиты файла от изменения  
привилегированным пользователем»

Студент группы ИУ7-76Б

\_\_\_\_\_  
(Подпись, дата)

**П. А. Калашков**

\_\_\_\_\_  
(И.О. Фамилия)

Руководитель курсовой работы

\_\_\_\_\_  
(Подпись, дата)

**Н. Ю. Рязанова**

\_\_\_\_\_  
(И.О. Фамилия)

2023 г.

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)

---

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-7

И. В. Рудаков

«16» сентября 2023 г.

**ЗАДАНИЕ**  
**на выполнение курсовой работы**

по теме

**«Загружаемый модуль ядра для защиты файла от изменения привилегированным  
пользователем»**

Студент группы **ИУ7-76Б**

**Калашков Павел Александрович**

Направленность КР

**учебная**

Источник тематики

**НИР кафедры**

График выполнения КР: 25% к 6 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

***Техническое задание***

*Разработать загружаемый модуль ядра для защиты файла от изменения привилегированным пользователем. Обеспечить защиту файла от изменения владельца файла и группы пользователей, защиту от операций записи, переименования, перемещения и удаления.*

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на **12-20** листах формата А4.

Дата выдачи задания «16» сентября 2023 г.

**Руководитель курсовой работы**

\_\_\_\_\_  
(Подпись, дата)

**Рязанова Н. Ю.**

(Фамилия И. О.)

**Студент**

\_\_\_\_\_  
(Подпись, дата)

**Калашков П. А.**

(Фамилия И. О.)

## РЕФЕРАТ

Расчетно-пояснительная записка 37 с., 7 рис., 15 ист.

### ОПЕРАЦИОННЫЕ СИСТЕМЫ, ЗАГРУЖАЕМЫЙ МОДУЛЬ ЯДРА, ЗАЩИТА ФАЙЛА

Цель работы — разработка загружаемого модуля ядра для защиты файла от изменения привилегированным пользователем.

Разработанная программа выполняет поставленную задачу: защищает файл от воздействия привилегированного пользователя, а именно от операций записи в файл, операций переименования, перемещения и удаления, а также операций изменения доступа к файлу, владельца файла и группы пользователей файла.

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>6</b>
<b>1 Аналитический раздел</b>	<b>7</b>
1.1 Постановка задачи . . . . .	7
1.2 Анализ выбранных действий . . . . .	7
1.2.1 Использование привилегий root . . . . .	7
1.2.2 Запись в файл . . . . .	8
1.2.3 Перемещение и переименования файла . . . . .	8
1.2.4 Изменение владельцев и группы пользователей файла . . . . .	9
1.3 Анализ структур ядра . . . . .	10
1.3.1 struct inode . . . . .	10
1.3.2 struct dentry . . . . .	15
1.3.3 Регистры процессоров Intel . . . . .	16
<b>2 Конструкторский раздел</b>	<b>19</b>
2.1 Последовательность действий . . . . .	19
2.2 Разработка алгоритмов . . . . .	20
<b>3 Технологический раздел</b>	<b>25</b>
3.1 Выбор языка и среды программирования . . . . .	25
3.2 Реализация загружаемого модуля ядра . . . . .	25
<b>4 Исследовательский раздел</b>	<b>32</b>
4.1 Технические характеристики . . . . .	32
4.2 Демонстрация работы программы . . . . .	32
<b>ЗАКЛЮЧЕНИЕ</b>	<b>34</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>35</b>

## **ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

В работе используются следующие термины с соответствующими определениями:

ОС — Операционная система

ПК — Персональный компьютер

## ВВЕДЕНИЕ

Операционные системы, в основе которых лежит ядро Linux, являются одними из самых популярных ОС в мире: к апрелю 2023 года ими пользуются 42% пользователей мобильных устройств, 1.2% пользователей ПК [1], а также 40.23% разработчиков [2].

В администрировании UNIX-подобных ОС существует понятие root аккаунта, у которого есть доступ ко всем командам и файлам системы [3]. Захват root-прав или других привилегированных учетных данных повышает шанс хакеров остаться незамеченным в сети и получить доступ к важным системам и данным.

**Целью работы** является разработка загружаемого модуля ядра для защиты файла от изменения привилегированным пользователем. Для достижения поставленной цели необходимо выполнить следующие задачи:

- 1) провести анализ действий, защиту от которых нужно обеспечить;
- 2) провести анализ структур и функций, предоставляющих возможность реализовать поставленную задачу;
- 3) разработать алгоритмы и структуру загружаемого модуля ядра, обеспечивающего защиту выбранного файла от изменения привилегированным пользователем, с учётом предотвращения работы функций write, rename, remove и операций изменения владельца файла и группы пользователей файла;

# 1 Аналитический раздел

## 1.1 Постановка задачи

В соответствии с техническим заданием для курсового проекта необходимо разработать загружаемый модуль ядра для защиты файла от изменения привилегированным пользователем. Для того, чтобы разработать необходимую программу, требуется проанализировать выполнение в ОС Linux следующих действий:

- использование привилегий root аккаунта;
- запись в файл;
- перемещение и переименование файла;
- удаление файла;
- изменение владельца файла и группы пользователей файла.

Анализ данных действий, а также способов защиты от них будет производиться на основе дистрибутива Ubuntu 22.04, в основе которого лежит ядро Linux версии 5.16.0 для процессоров x86\_64.

## 1.2 Анализ выбранных действий

### 1.2.1 Использование привилегий root

root аккаунт (root-пользователь, суперпользователь) — аккаунт, у которого есть доступ ко всем командам и файлам системы. Привилегии root — возможности взаимодействия с системой, которые даёт root-аккаунт. root аккаунт обладает абсолютным контролем над системой: он имеет полный доступ ко всем файлам и командам, может изменять систему любым желаемым способом, а также разрешать или запрещать доступ к файлам другим пользователям системы (доступ на чтение, доступ на запись и доступ на исполнение файла) [3].

Существует команда `sudo`, позволяющая дать пользователю или группе пользователей временные (по умолчанию на 5 минут) привилегии root аккаунта. Чтобы использовать данную команду, необходимо написать `sudo`, после чего написать команду, которую необходимо исполнить с использованием root-прав.

Таким образом, реализуемая программа должна защищать указанный файл от воздействия команд, исполненных как без использования `sudo`, так и с использованием.

### 1.2.2 Запись в файл

В ОС Linux запись в файл, а также команды и программы, в той или иной мере изменяющие файл (команда перенаправления ввода-вывода, текстовые редакторы) используют функцию `write`, имеющую заголовок, представленный на листинге 1.

Листинг 1 – Заголовок функции `write`

```
1 ssize_t write(int fd, const void buf[.count], size_t count);
```

Функция `write` записывает `count` байт из буфера, начинающегося по указателю `buf` в открытый файл с дескриптором `fd` [4].

Таким образом, чтобы предотвратить запись в указанный файл, необходимо предотвратить открытие данного файла в режиме записи. Для этого необходимо провести анализ структур ядра, при помощи которых описывается открытый файл.

### 1.2.3 Перемещение и переименования файла

Перемещение и переименование файла в ОС Linux может быть осуществлено при помощи команды `mv`, варианты использования которой приведены в листинге 1.

Листинг 2 – Варианты использования команды `mv`

```
1 mv [опции...] исходный_файл файл_назначения  
2 mv [опции...] исходный_файл... каталог
```

Если последний аргумент является именем существующего каталога, то `mv` перемещает все остальные файлы в этот каталог. В противном случае, если задано только два файла, то имя первого файла будет изменено на имя второго.



Если последний аргумент не является каталогом и задано более чем два файла, то будет выдано сообщение об ошибке [5].

Так, `mv /a/x/y /b` переименует файл `/a/x/y` в `/b/y`, если `/b` является существующим каталогом, и в `/b`, если нет.

Для предотвращения перемещения и переименования указанного файла необходимо проанализировать структуры ядра, при помощи которых описываются элементы пути в файловой системе.

#### **1.2.4 Изменение владельцев и группы пользователей файла**

Для каждого объекта (файла, каталога, ссылки) в Linux и других Unix-подобных ОС действует система доступов [6]. Каждый объект обладает тремя типами доступов: доступ на чтение, доступ на запись (модификацию) и доступ на выполнение.

Каждый из этих доступов определён для трёх категорий пользователей:

- владелец;
- группа пользователей (т. е. набор пользователей, обладающих одинаковыми правами доступа), к которой принадлежит владелец файла;
- все остальные пользователи.

Доступы для конкретного файла могут быть представлены разными способами, один из них — текстовый, который предоставляет команда `ls` с опцией `-l`. В этом случае доступы представляются в виде 10-символьной строки, первый символ которой обозначает тип файла (например, дефис для обычного файла и *d* для каталога), а оставшиеся 9 символов состоят из трёх групп по 3 символа, представляющих доступы на чтение (символ *r* в случае наличия доступа), запись (символ *w*) и выполнение (символ *x*). В случае, если конкретного доступа нет, символом будет дефис.

Так, строка `-rwxrwx-r--`, возвращаемая для объекта командой `ls -l` будет означать, что объект является обычным файлом, его владелец имеет доступ на чтение, запись, и выполнение, группа пользователей имеет доступ на чтение и запись, а все остальные пользователи имеют доступ только на чтение.

Доступы также могут быть представлены в численном виде как три восьмеричных числа. Значения этих чисел определены следующим образом: 0 для запрета всех типов доступа, 1 для доступа только на выполнение, 2 для доступа только на запись, 3 для доступа на запись и выполнение, 4 для доступа только на чтение, 5 для доступа на чтение и выполнение, 6 для доступа на чтение и запись, 7 для доступа на чтение, запись и выполнение.

Так, число 777 будет означать, что все пользователи имеют все доступы к данному файлу.

Для изменения доступов к конкретному файлу существует команда `chmod`, которая принимает необходимые доступы в численном или текстовом виде, команда `chown`, меняющая владельца файла и команда `chgrp`, изменяющая группу пользователей файла.

Таким образом, для предотвращения изменения доступов к файлу (в том числе владельца файла и группы пользователей файла) необходимо проанализировать структуры ядра, описывающие доступы к конкретному файлу.

### **1.3 Анализ структур ядра**

Рассмотрим структуры ядра (ядро Linux версии 5.16.0), описывающие открытый файл и элемент пути в файловой системе.

#### **1.3.1 struct inode**

`inode` — структура ядра Linux, описывающая открытый файл. Она содержит в себе информацию, необходимую ядру для действий, связанных с файлом. На листингах 3–5 представлена структура `inode` с комментариями отдельных полей [7].

### Листинг 3 – Структура inode, часть 1

```
1 struct inode {
2     umode_t i_mode;                /* access permissions */
3     unsigned short i_opflags;      /* xattr support flags */
4     kuid_t i_uid;                  /* user id of owner */
5     kgid_t i_gid;                  /* group id of owner */
6     unsigned int i_flags;          /* filesystem flags */
7 #ifdef CONFIG_FS_POSIX_ACL         /* if FS supports ACL */
8     struct posix_acl * i_acl;      /* ACL mode */
9     struct posix_acl * i_default_acl; /* default ACL */
10 #endif
11     const struct inode_operations *i_op; /*inode operations */
12     struct super_block *i_sb;        /* superblock pointer */
13     struct address_space *i_mapping; /* associated mapping */
14 #ifdef CONFIG_SECURITY             /* if with security module */
15     void *i_security;              /* security module */
16 #endif
17     unsigned long i_ino;            /* inode number */
18     union {
19         const unsigned int i_nlink; /* number of hard links */
20         unsigned int __i_nlink;     /* number of hard links */
21     };
22     dev_t i_rdev;                   /* real device node */
23     loff_t i_size;                  /* file size in bytes */
24     struct timespec64 i_atime;      /* last access time */
25     struct timespec64 i_mtime;      /* last modify time */
26     struct timespec64 i_ctime;      /* last change time */
27     spinlock_t i_lock;              /* spinlock */
28     unsigned short i_bytes;         /* bytes consumed */
29     u8 i_blkbits;                   /* block size in bits */
30     u8 i_write_hint;                /* hint about block */
31     blkcnt_t i_blocks;              /* file size in blocks */
}
```

## Листинг 4 – Структура inode, часть 2

```

1  #ifdef __NEED_I_SIZE_ORDERED                /* sequence counters */
2      seqcount_t i_size_seqcount;              /* counter */
3  #endif
4      unsigned long i_state;                    /* state flags */
5      struct rw_semaphore i_rwsem;              /* semaphore */
6      unsigned long dirtied_when; /* jiffies of first dirtying */
7      unsigned long dirtied_time_when;          /* last dirtying */
8      struct hlist_node i_hash;                 /* hash list */
9      struct list_head i_io_list;               /* backing dev IO list */
10
11 #ifdef CONFIG_CGROUP_WRITEBACK /* flush dirty from page cache */
12     struct bdi_writeback *i_wb; /* the associated cgroup wb */
13     /* foreign inode detection, see wbc_detach_inode() */
14     int i_wb_frn_winner;
15     ul6 i_wb_frn_avg_time;
16     ul6 i_wb_frn_history;
17 #endif
18     struct list_head i_lru;                    /* inode LRU list */
19     struct list_head i_sb_list;                /* sb of inodes */
20     struct list_head i_wb_list;                /* backing dev wb list */
21     union {
22         struct hlist_head i_dentry; /* list of dentries */
23         struct rcu_head i_rcu; /* list of rcu updates */
24     };
25     atomic64_t i_version;                      /* version number */
26     atomic64_t i_sequence;                     /* see futex */
27     atomic_t i_count;                          /* reference counter */
28     atomic_t i_dio_count;                      /* DIO counter */
29     atomic_t i_writecount;                     /* writers counter */
30 #if defined(CONFIG_IMA) || defined(CONFIG_FILE_LOCKING) /*IMA*/
31     atomic_t i_readcount;                      /* struct files open RO */
32 #endif

```

## Листинг 5 – Структура inode, часть 3

```
1     union {
2         const struct file_operations *i_fop /*file operations*/
3         void (*free_inode)(struct inode *); /* free function */
4     };
5     struct file_lock_context *i_flctx;          /* for fcntl */
6     struct address_space i_data;                /* mapping for device */
7     struct list_head i_devices; /* list of block devices */
8     union {
9         struct pipe_inode_info *i_pipe; /* pipe information */
10        struct cdev *i_cdev;              /*char devices */
11        char *i_link;                     /* symlink */
12        unsigned i_dir_seq;               /* open-coded seqcounter */
13    };
14
15    __u32 i_generation;                    /* inode version number */
16
17    #ifdef CONFIG_FSNOTIFY
18        __u32 i_fsnotify_mask; /* all events inode cares about */
19        struct fsnotify_mark_connector __rcu *i_fsnotify_marks;
20    #endif
21
22    #ifdef CONFIG_FS_ENCRYPTION
23        struct fscrypt_info *i_crypt_info;
24    #endif
25
26    #ifdef CONFIG_FS_VERITY
27        struct fsverity_info *i_verity_info;
28    #endif
29
30    void *i_private; /* fs or device private pointer */
31 } __randomize_layout;
```

Для предотвращения операции открытия файла в режиме записи, а также самой записи необходимо изменить содержимое поля `i_for`. Структура `file_operations` содержит 4 поля, имеющих значение для поставленной задачи (листинг 6).

Листинг 6 – Структура `file_operations`

```
1 struct file_operations {
2     // ...
3     ssize_t (*read) (struct file *, char __user *,
4         size_t, loff_t *);
5     ssize_t (*write) (struct file *, const char __user *,
6         size_t, loff_t *);
7     ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
8     ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
9     int (*open) (struct inode *, struct file *);
10    // ...
11 } __randomize_layout;
```

Структура `file_operations` содержит поля `read` и `write`, в которых хранятся указатели на функции, вызывающиеся для чтения и записи данных соответственно. Также в ней содержатся поля `read_iter` и `write_iter`, содержащие указатели на функции, вызывающиеся перед `read` и `write` соответственно. Для предотвращения записи в файл необходимо переопределить значение поля `write_iter` таким образом, чтобы в случае, когда запись возможна, возвращался флаг `EACCES`, означающий запрет на запись.

Для предотвращения открытия файла в режиме записи необходимо переопределить значение поля `open` таким образом, чтобы в случае открытия файла на чтение работа осуществлялась без изменений, а в случае открытия файла на запись возвращался флаг `EACCES`.

Для предотвращения изменений доступов к файлу, владельца файла и группы пользователей файла необходимо обратиться к полю `i_or` в `inode`. В нём

есть поле `setattr`, содержащее указатель на функцию изменения таких атрибутов `inode`, как `i_uid`, `i_gid`, а также `i_mode` (листинг 7).

Листинг 7 – Структура `inode_operations`

```
1 struct inode_operations {
2     // ...
3     int (*setattr) (struct user_namespace *, struct dentry *,
4                     struct iattr *);
5     // ...
6 } ____cacheline_aligned;
```

Для корректной работы разрабатываемого модуля ядра необходимо, чтобы после выгрузки модуля значения изменённых полей возвращались обратно. Для этого необходимо предусмотреть сохранение исходных объектов полей `i_for` и `i_op`.

### 1.3.2 `struct dentry`

`dentry` — структура ядра Linux, описывающая элемент пути в файловой системе. При таких операциях, как переименование и перемещение файла, взаимодействие происходит именно с объектами `dentry`, а не с `inode`. В листингах 8–8 представлена структура `dentry` с комментариями полей [8].

Листинг 8 – Структура `dentry`, часть 1

```
1 struct dentry {
2     unsigned int d_flags; /* dentry flags */
3     seqcount_spinlock_t d_seq; /* entry seqlock */
4     struct hlist_bl_node d_hash; /* list of hash table entries */
5     struct dentry *d_parent; /* parent directory */
6     struct qstr d_name; /* dentry name */
7     struct inode *d_inode; /* associated inode */
8     unsigned char d_iname[DNAME_INLINE_LEN]; /* small names */
9     struct lockref d_lockref; /* per-dentry lock and refcount */
10    const struct dentry_operations *d_op; /* operations */
```

## Листинг 9 – Структура dentry, часть 2

```
1 struct super_block *d_sb; /* The root of the dentry tree */
2 unsigned long d_time; /* revalidate time */
3 void *d_fsdata; /* fs-specific data */
4
5 union {
6     struct list_head d_lru; /* LRU list */
7     wait_queue_head_t *d_wait; /* in-lookup ones only */
8 };
9 struct list_head d_child; /* child of parent list */
10 struct list_head d_subdirs; /* our children */
11
12 union {
13     struct hlist_node d_alias; /* inode alias list */
14     struct hlist_bl_node d_in_lookup_hash; /* in-lookup */
15     struct rcu_head d_rcu; /* RCU locking */
16 } d_u;
17 } __randomize_layout;
```

Поле `d_flags`, содержащее флаги файловой системы, позволяет определить возможность изменения прав доступа для описываемого файла. Существует флаг `FS_IMMUTABLE_FL`, запрещающий изменение полей `i_uid`, `i_gid`, а также `i_mode` у `inode`, указатель на который содержится в поле `d_inode`. Установка данного флага позволит предотвратить изменение владельца файла, группы пользователей файла, а также доступы к файлу.

После выгрузки разрабатываемого модуля ядра необходимо, чтобы значение поля `d_flags` устанавливалось в исходное значение.

### 1.3.3 Регистры процессоров Intel

Страницы памяти, в которых хранятся структуры `file_operations` и `inode_operations` помечены как `read-only`. В процессорах, построенных по ар-



хитектуре x86, существует понятие регистров контроля (англ. *Control Registers, CR*), которые содержат значения флагов, относящихся к защите памяти, многопоточности, разбиению памяти на страницы и т. д. [9]. 16-й бит регистра контроля CR0 отвечает за возможность изменения страниц, помеченных как read-only в режиме суперпользователя. Когда он установлен, суперпользователь не может изменять страницы, помеченные как read-only.

На некоторых процессорах x86 производства Intel данный бит по умолчанию установлен, поэтому прежде чем заменять значения полей `i_or` и `i_for` на новые, необходимо выключить данный флаг и установить его в исходное значение после выполнения необходимых операций.

Для улучшения безопасности регистры CR0 и CR4 в ОС Linux с версии 5.3 являются защищёнными, и функция `write_cr0`, с помощью которой можно было установить значение регистра CR0, уже не входит в набор предоставляемых разработчикам функций. Необходимо найти альтернативный способ установки значения регистра CR0 при помощи ассемблерных вставок из загружаемого модуля ядра.

## **Вывод**

В данном разделе был проведён анализ действий, которые должна предотвращать разрабатываемая программа, а также структур ядра, необходимых для реализации загружаемого модуля. Определены действия, которые необходимо предотвращать: команды записи (использование вызова `write` и текстового редактора), команд `chmod`, `chown` и `chgrp`, изменяющих доступы к файлу, а также команды `mv`, перемещающей или переименовывающей файл. Данные действия необходимо предотвращать, даже если они выполнены при помощи команды `sudo`, дающей привилегии суперпользователя.

Для предотвращения открытия и записи в файл необходимо переопределить исходные значения полей `open` и `write_iter` структуры `file_operations` в поле `i_for` у `inode`, относящегося к файлу. Для предотвращения изменения доступа к файлу, а также владельца файла и группы пользователей, необходимо пере-

определить значение поля `setattr` структуры `inode_operations` в поле `i_op` того же `inode`.

Для предотвращения перемещения и переименования файла необходимо обеспечить неизменяемость соответствующего файлу объекта `dentry` при помощи установки флага `FS_IMMUTABLE_FL` в поле `d_flags`.

Для корректной работы программы необходимо обеспечить работу с регистром контроля `CR0` при помощи ассемблерных вставок для возможности изменить необходимые поля у `inode`. При выгрузке разработанного модуля ядра значения изменённых полей должны возвращаться к исходным.

## 2 Конструкторский раздел

### 2.1 Последовательность действий

На рисунке 1 представлена последовательность действий, выполняемых при загрузке модуля, а на рисунке 2 — последовательность действий при выгрузке модуля.

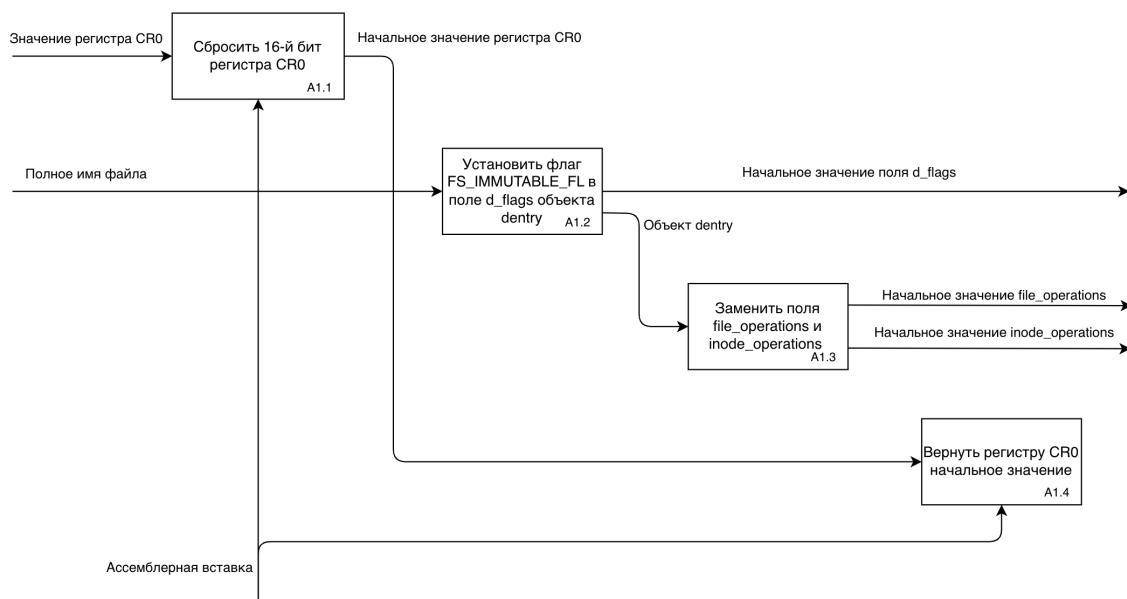


Рисунок 1 – Последовательность действий при загрузке модуля

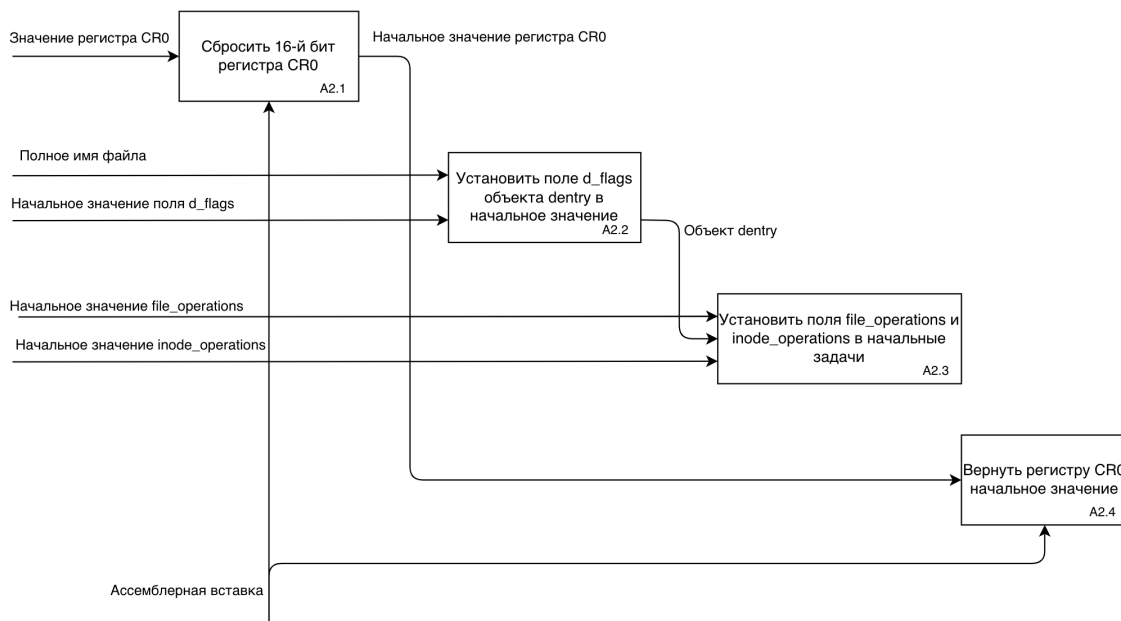


Рисунок 2 – Последовательность действий при выгрузке модуля

## 2.2 Разработка алгоритмов

На рисунке 3 представлена схема алгоритма функции загрузки модуля ядра, на рисунке 4 представлена схема алгоритма функции создания нового объекта `file_operations` и схемы алгоритмов функций `write_iter_hook` и `open_hook`, а на рисунке 5 представлена схема алгоритма функции создания нового объекта `inode_operations` и схема алгоритма функции `setattr_hook`. На рисунке 6 представлена схема алгоритма функции выгрузки загружаемого модуля.

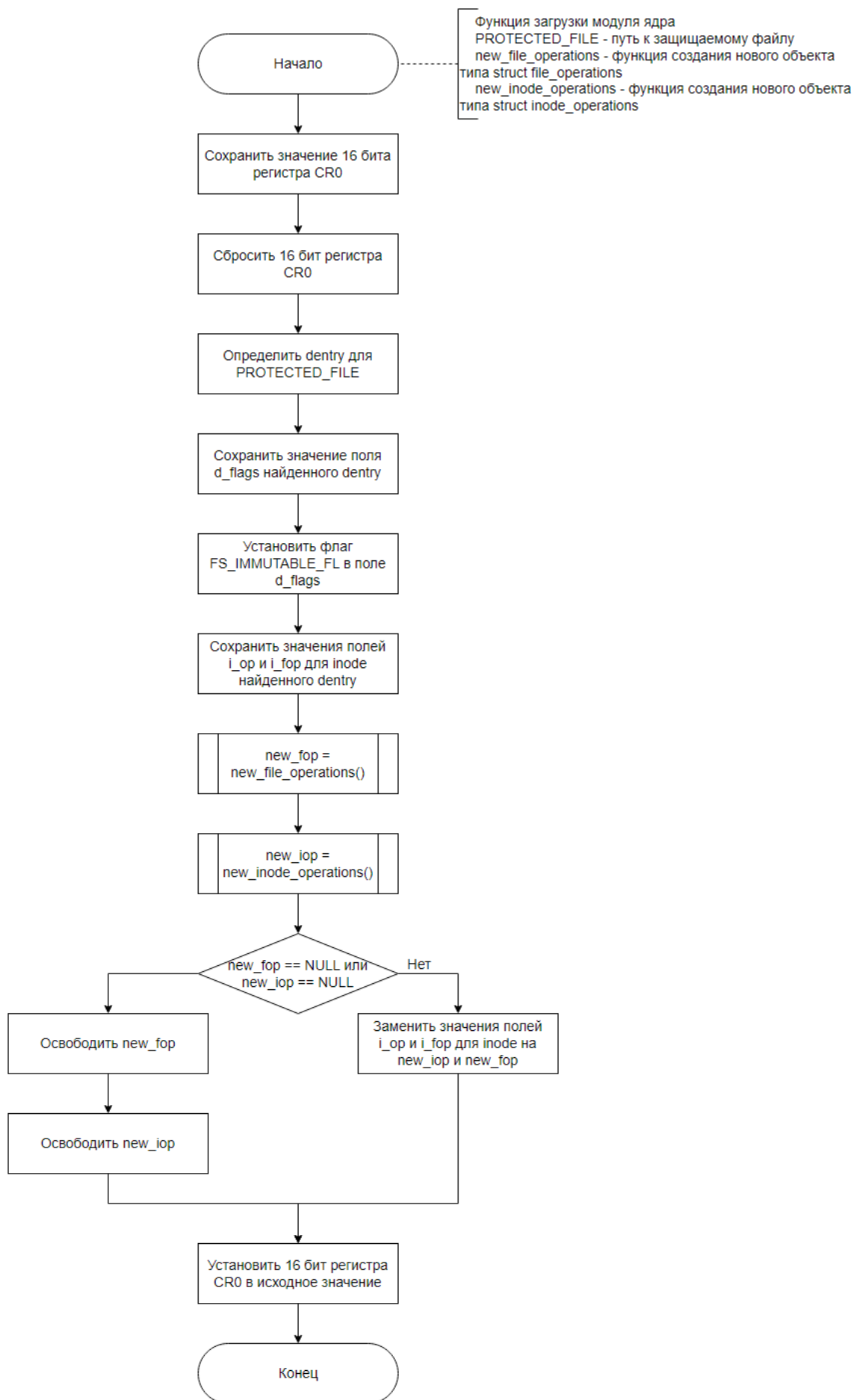


Рисунок 3 – Схема алгоритма функции загрузки модуля ядра

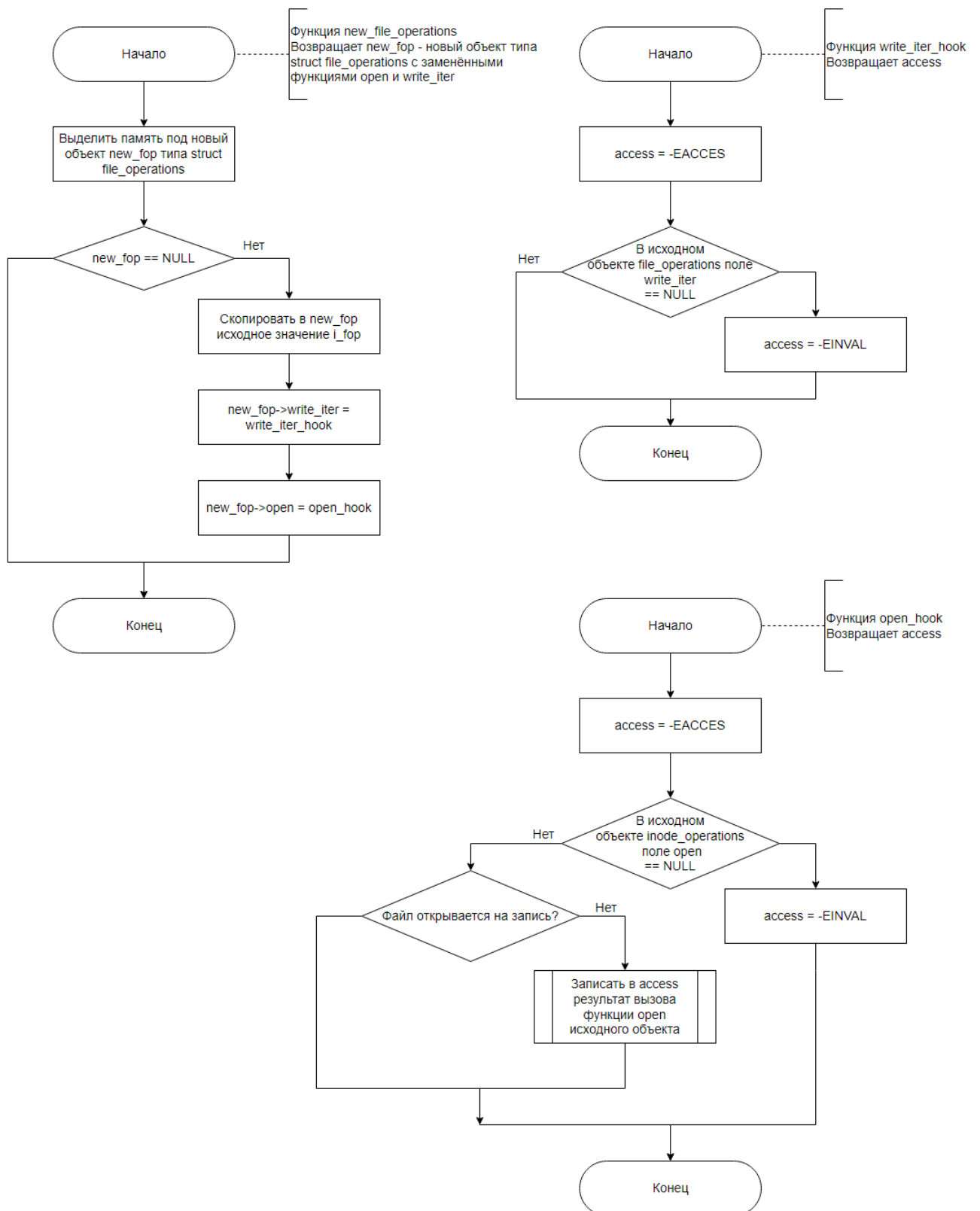


Рисунок 4 – Схемы алгоритмов создания нового объекта file\_operations и функций write\_iter\_hook и open\_hook

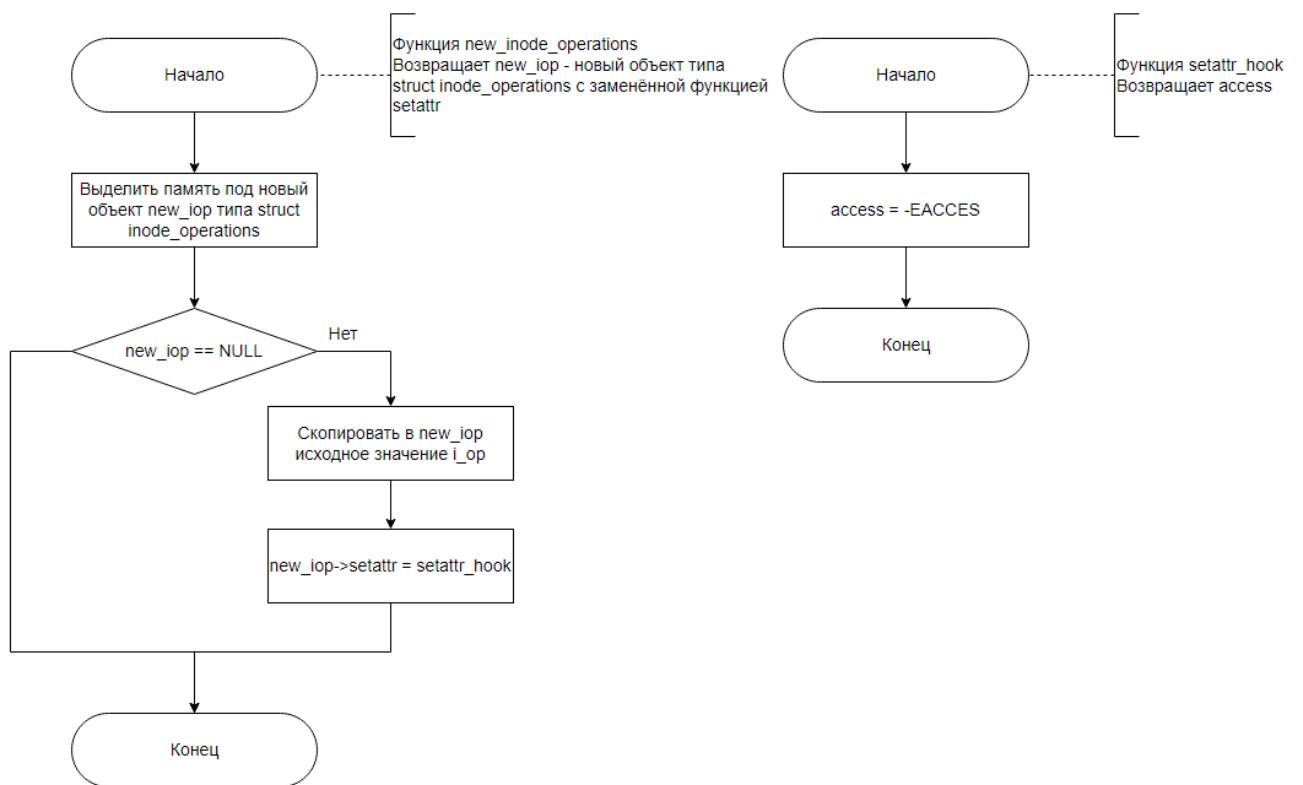


Рисунок 5 – Схемы алгоритма создания нового объекта `inode_operations` и функции `setattr_hook`

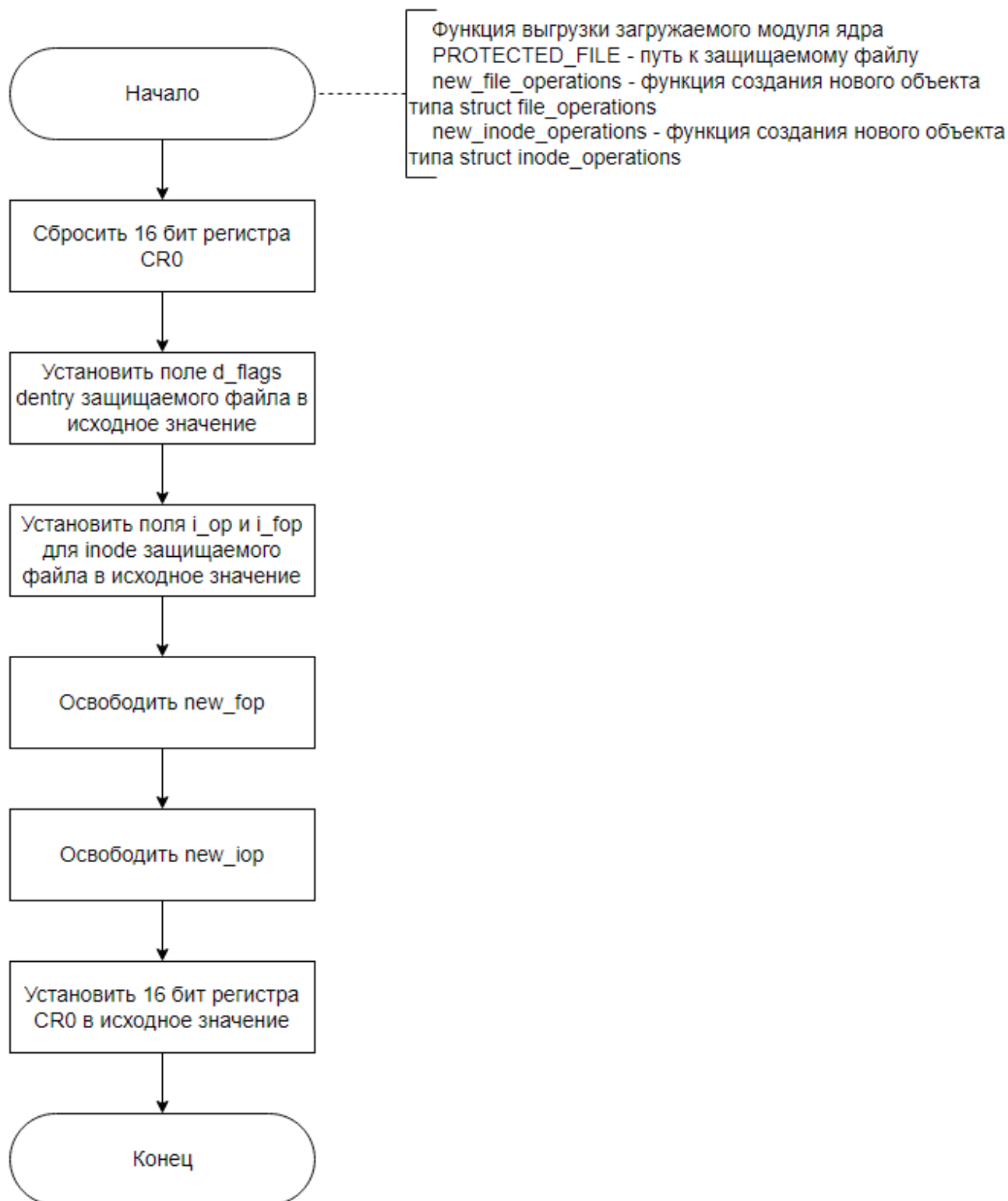


Рисунок 6 – Схема алгоритма выгрузки загружаемого модуля



### **3 Технологический раздел**

#### **3.1 Выбор языка и среды программирования**

Как основное средство реализации и разработки ПО был выбран язык программирования Си [10], поскольку в нём есть все инструменты для реализации загружаемого модуля ядра. Средой программирования послужил графический редактор Visual Studio Code [11], который известен содержанием большого количества плагинов, ускоряющих процесс разработки программы.

#### **3.2 Реализация загружаемого модуля ядра**

В расположенных ниже листингах 10 – 14 приведена реализация загружаемого модуля ядра согласно разработанным алгоритмам, а на листинге 15 — Makefile для компиляции загружаемого модуля.

## Листинг 10 – Реализация загружаемого модуля ядра, часть 1

```
1 #include <linux/module.h>
2 #include <linux/kernel.h>
3 #include <linux/syscalls.h>
4 #include <linux/unistd.h>
5 #include <asm/uaccess.h>
6 #include <linux/namei.h>
7 #include <asm/string.h>
8
9 #define PROTECTED_FILE "/home/paul/Desktop/bmstu-os-cw/\
10     src/protected"
11 MODULE_LICENSE ("GPL");
12
13 struct dentry* protected_dentry;
14 struct inode* protected_inode;
15 struct file_operations *original_fop, *new_fop;
16 struct inode_operations *original_iop, *new_iop;
17 unsigned int initial_dentry_flags;
18 unsigned long initial_cr0;
19
20 inline void write_cr0_custom(unsigned long cr0)
21 {
22     unsigned long __force_order;
23     asm volatile("mov %0,%%cr0" : "+r"(cr0),
24                 "+m"(__force_order));
25 }
26
27 void disable_write_protection(void)
28 {
29     write_cr0_custom(initial_cr0 & (~0x10000));
30 }
```

## Листинг 11 – Реализация загружаемого модуля ядра, часть 2

```
1 void enable_write_protection(void)
2 {
3     write_cr0_custom(initial_cr0);
4 }
5
6 struct inode* protect_inode(void)
7 {
8     int error;
9     struct inode *inode;
10    struct path protected_path;
11    error = kern_path(PROTECTED_FILE, LOOKUP_FOLLOW,
12                      &protected_path);
13    if (!error)
14    {
15        protected_dentry = protected_path.dentry;
16        inode = protected_path.dentry->d_inode;
17        initial_dentry_flags = protected_path.dentry->d_flags;
18        protected_path.dentry->d_flags = FS_IMMUTABLE_FL;
19        return inode;
20    }
21    printk(KERN_INFO"kern_path error %llx\n", protected_path);
22    return NULL;
23 }
24
25 ssize_t write_iter_hook(struct kiocb *X, struct iov_iter *Y)
26 {
27     ssize_t access = -EACCES;
28     if(original_fop->write_iter == NULL)
29         access = -EINVAL;
30     return access;
31 }
```

## Листинг 12 – Реализация загружаемого модуля ядра, часть 3

```
1 int open_hook(struct inode* inode, struct file* filep)
2 {
3     int access = -EACCES;
4     if(original_fop->open == NULL)
5         access = -EINVAL;
6     else if (!(filep->f_mode & FMODE_WRITE))
7         access = original_fop->open(inode, filep);
8     return access;
9 }
10
11 int setattr_hook(struct user_namespace *un, struct dentry *d,
12                 struct iattr *attr)
13 {
14     return -EACCES;
15 }
16
17 struct file_operations* new_file_operations(void)
18 {
19     struct file_operations* new_fop = kmalloc(
20         sizeof(struct file_operations), GFP_KERNEL);
21     if (new_fop)
22     {
23         memcpy(new_fop, original_fop,
24             sizeof(struct file_operations));
25         new_fop->write_iter = write_iter_hook;
26         new_fop->open = open_hook;
27     }
28     return new_fop;
29 }
```

### Листинг 13 – Реализация загружаемого модуля ядра, часть 4

```
1 struct inode_operations* new_inode_operations(void)
2 {
3     struct inode_operations* new_iop = kmalloc(
4         sizeof(struct inode_operations), GFP_KERNEL);
5     if (new_iop)
6     {
7         memcpy(new_iop, original_iop,
8             sizeof(struct inode_operations));
9         new_iop->setattr = setattr_hook;
10    }
11    return new_iop;
12 }
13
14 static int __init mod_init (void)
15 {
16     initial_cr0 = read_cr0();
17     disable_write_protection();
18     protected_inode = protect_inode();
19     if (protected_inode == NULL)
20     {
21         printk(KERN_INFO"Cannot get inode address\n\
22             Maybe protected file doesn't exist\n");
23         enable_write_protection();
24         return -1;
25     }
26     original_fop = protected_inode->i_fop;
27     original_iop = protected_inode->i_op;
28     new_fop = new_file_operations();
29     new_iop = new_inode_operations();
30     if (!new_fop || ! new_iop)
31     {
32         kfree(new_fop);
```

## Листинг 14 – Реализация загружаемого модуля ядра, часть 5

```
1         kfree(new_iop);
2     }
3     else
4     {
5         protected_inode->i_fop = new_fop;
6         protected_inode->i_op = new_iop;
7     }
8     enable_write_protection();
9     return 0;
10
11 }
12
13 static void mod_exit (void)
14 {
15     disable_write_protection();
16     protected_dentry->d_flags = initial_dentry_flags;
17     protected_inode->i_fop = original_fop;
18     protected_inode->i_op = original_iop;
19     enable_write_protection();
20     kfree(new_fop);
21     kfree(new_iop);
22 }
23
24 module_init(mod_init);
25 module_exit(mod_exit);
```

## Листинг 15 – Makefile для компиляции загружаемого модуля

```
1 obj-m = file_protection.o
2
3 all:
4     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
5
6 clean:
7     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

### Вывод

В данном разделе были рассмотрены средства разработки программного обеспечения и приведены листинги исходного кода загружаемого модуля ядра.

## **4 Исследовательский раздел**

### **4.1 Технические характеристики**

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Ubuntu 20.04.6 LTS [12];
- Память: 16 Гб с тактовой частотой 2133 МГц LPDDR3 [13];
- Процессор: Intel Core™ i7-8559U [14] с тактовой частотой 2.70 ГГц;
- Видеокарта: NVIDIA GeForce GTX 1060 [15]

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только системой тестирования (работающим приложением) и системным окружением операционной системы.

### **4.2 Демонстрация работы программы**

На рисунке 7 продемонстрирована работа загружаемого модуля ядра: осуществляется загрузка модуля, демонстрация исходного содержимого защищаемого файла и доступы к нему, производятся попытки осуществить запись в файл, переместить, переименовать, удалить файл, а также поменять доступы к нему, изменить владельца файла и группу пользователей файла (все действия были выполнены с правами суперпользователя). После этого производится повторная демонстрация содержимого файла и прав доступа к нему: содержимое и права доступа не изменились, защита работает. После выгрузки модуля ядра защита перестаёт работать и действия работают корректно.



```

• paul@paul:~/Desktop/bmstu-os-cw/src$ sudo insmod file_protection.ko
• paul@paul:~/Desktop/bmstu-os-cw/src$ cat protected
Some content
⊗ paul@paul:~/Desktop/bmstu-os-cw/src$ sudo echo I changed this file > protected
bash: protected: No such file or directory
⊗ paul@paul:~/Desktop/bmstu-os-cw/src$ sudo mv protected renamed_protected
mv: cannot move 'protected' to 'renamed_protected': No such file or directory
⊗ paul@paul:~/Desktop/bmstu-os-cw/src$ sudo mv protected ..
mv: cannot move 'protected' to '../protected': No such file or directory
⊗ paul@paul:~/Desktop/bmstu-os-cw/src$ sudo rm protected
rm: cannot remove 'protected': No such file or directory
• paul@paul:~/Desktop/bmstu-os-cw/src$ ls -l protected
-rw-rw-rw- 1 paul paul 13 дек 22 13:27 protected
⊗ paul@paul:~/Desktop/bmstu-os-cw/src$ sudo chmod 777 protected
chmod: changing permissions of 'protected': Permission denied
⊗ paul@paul:~/Desktop/bmstu-os-cw/src$ sudo chown root protected
chown: changing ownership of 'protected': Permission denied
⊗ paul@paul:~/Desktop/bmstu-os-cw/src$ sudo chgrp sudo protected
chgrp: changing group of 'protected': Permission denied
• paul@paul:~/Desktop/bmstu-os-cw/src$ ls -l protected
-rw-rw-rw- 1 paul paul 13 дек 22 13:27 protected
• paul@paul:~/Desktop/bmstu-os-cw/src$ cat protected
Some content
• paul@paul:~/Desktop/bmstu-os-cw/src$ sudo rmmod file_protection
• paul@paul:~/Desktop/bmstu-os-cw/src$ sudo echo I changed this file > protected
• paul@paul:~/Desktop/bmstu-os-cw/src$ cat protected
I changed this file
• paul@paul:~/Desktop/bmstu-os-cw/src$ sudo chmod 777 protected
• paul@paul:~/Desktop/bmstu-os-cw/src$ ls -l protected
-rwxrwxrwx 1 paul paul 20 дек 22 13:30 protected
• paul@paul:~/Desktop/bmstu-os-cw/src$ █

```

Рисунок 7 – Демонстрация работы программы

## Вывод

В данном разделе были приведены примеры работы загружаемого модуля ядра и демонстрация защиты файла от описанных в аналитическом разделе действий.

Разработанная программа выполняет поставленную задачу: защищает файл от воздействия привилегированного пользователя, а именно от операций записи в файл, операций переименования, перемещения и удаления, а также операций изменения доступа к файлу, владельца файла и группы пользователей файла.

## ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был разработанн загружаемый модуль ядра для защиты файла от воздействия привилегированного процесса, а именно от операций записи в файл, операций переименования, перемещения и удаления, а также операций изменения доступа к файлу, владельца файла и группы пользователей файла.

Были выполнены следующие задачи:

- 1) проведён анализ действий, защиту от которых нужно обеспечить;
- 2) проведён анализ структур и функций, предоставляющих возможность реализовать поставленную задачу;
- 3) разработаны алгоритмы и структура загружаемого модуля ядра, обеспечивающего защиту выбранного файла от изменения привилегированным пользователем, с учётом предотвращения работы функций write, rename, remove и операций изменения владельца файла и группы пользователей файла;

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Operating System Market Share Worldwide [Электронный ресурс]. — Режим доступа: <https://gs.statcounter.com/os-market-share#monthly-202111-202303> (дата обращения: 07.10.2023).
2. Stack Overflow Developer Survey 2022 [Электронный ресурс]. — Режим доступа: <https://survey.stackoverflow.co/2022/#operating-system> (дата обращения: 07.10.2023).
3. root Definition, The Linux Information Project [Электронный ресурс]. — Режим доступа: <https://www.linfo.org/root.html> (дата обращения: 07.10.2023).
4. System Calls Manual, write (2) [Электронный ресурс]. — Режим доступа: <https://man7.org/linux/man-pages/man2/write.2.html> (дата обращения: 21.12.2023).
5. System Calls Manual, mv (1) [Электронный ресурс]. — Режим доступа: <https://www.opennet.ru/man.shtml?topic=mv&category=1&russian=0> (дата обращения: 21.12.2023).
6. Permissions Definition, The Linux Information Project [Электронный ресурс]. — Режим доступа: <https://www.linfo.org/permissions.html> (дата обращения: 21.12.2023).
7. Bootlin, Elixir Cross Referencer, struct inode [Электронный ресурс]. — Режим доступа: <https://elixir.bootlin.com/linux/v5.16/source/include/linux/fs.h#L620> (дата обращения: 21.12.2023).
8. Bootlin, Elixir Cross Referencer, struct dentry [Электронный ресурс]. — Режим доступа: <https://elixir.bootlin.com/linux/v5.16/source/include/linux/dcache.h#L91> (дата обращения: 21.12.2023).

9. Intel's Software Developer's Manual [Электронный ресурс]. — Режим доступа: <https://software.intel.com/content/dam/develop/public/us/en/documents/325384-sdm-vol-3abcd.pdf> (дата обращения: 21.12.2023).
10. Документация по языку C. Узнайте, как использовать C и библиотеку времени выполнения C [Электронный ресурс]. — Режим доступа: <https://learn.microsoft.com/ru-ru/cpp/c-language/?view=msvc-170> (дата обращения: 22.12.2023).
11. Visual Studio Code — Code Editing. Redefined [Электронный ресурс]. — Режим доступа: <https://code.visualstudio.com> (дата обращения: 22.12.2023).
12. macOS Ventura — official site [Электронный ресурс]. — Режим доступа: <https://releases.ubuntu.com/focal/> (дата обращения: 22.12.2023).
13. LPDDR3 SDRAM Documentation [Электронный ресурс]. — Режим доступа: [https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/mobile-dram/low-power-dram/lpddr3/178b\\_8-16gb\\_2c0f\\_mobile\\_lpddr3.pdf](https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/mobile-dram/low-power-dram/lpddr3/178b_8-16gb_2c0f_mobile_lpddr3.pdf) (дата обращения: 17.05.2023).
14. Процессор Intel Core™ i7-8559U [Электронный ресурс]. — Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/137979/intel-core-i78559u-processor-8m-cache-up-to-4-50-ghz.html> (дата обращения: 17.05.2023),
15. GeForce GTX 1060 Specifications [Электронный ресурс]. — Режим досту-

па: <https://www.nvidia.com/en-gb/geforce/graphics-cards/geforce-gtx-1060/specifications/> (дата обращения: 22.12.2023),