**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»

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К КУРСОВОМУ ПРОЕКТУ*

## *НА ТЕМУ:*

## *Реализация межсетевого экрана*

Студент ИУ7-72Б _____ _____ Е.В. Брянская
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ Н.Ю. Рязанова
(Подпись, дата) (И.О.Фамилия)

*2021 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 ___ИУ7____
(Индекс)
_____ __И.В. Рудаков_
(И.О.Фамилия)
« _____ » _____ 2021 г.

# З А Д А Н И Е
## на выполнение курсового проекта

по дисциплине _____Операционные системы_____

Студент группы ___ИУ7-72Б_____

_____Брянская Екатерина Вадимовна_____
(Фамилия, имя, отчество)

Тема курсового проекта _____
_____Реализация межсетевого экрана_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
__учебный_____
Источник тематики (кафедра, предприятие, НИР) ___кафедра_____

График выполнения проекта: 25% к _4_ нед., 50% к _7_ нед., 75% к 11 нед., 100% к _14_ нед.

***Задание***__Разработать межсетевой экран, осуществляющий контроль проходящего через него_ сетевого трафика, в виде загружаемого модуля, который использует для обработки пакетов___ символьное устройство. Предоставить пользователю возможность редактирования списка____ правил фильтрации и изменение видимости модуля в системе. _____

***Оформление курсового проекта:***

Расчетно-пояснительная записка на _20-30__ листах формата А4.
Расчетно-пояснительная записка должна содержать постановку задачи, введение,_____ аналитическую, конструкторскую, технологическую части, заключение, список литературы.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):_____ на защиту работы должна быть предоставлена презентация, состоящая из 15-20 слайдов._____ На слайдах должны быть отражены: постановка задачи, использованные методы и_____ алгоритмы, расчетные соотношения, структура комплекса программ, интерфейс._____

Дата выдачи задания «27» __сентября___ 2021 г.

**Руководитель курсового проекта** _____ __Н.Ю. Рязанова____
(Подпись, дата) (И.О.Фамилия)

**Студент** _____ __Е.В. Брянская_____
(Подпись, дата) (И.О.Фамилия)

**Содержание**

# ВВЕДЕНИЕ

Информация – это один из важнейших ресурсов, который представляет из себя движущую силу развития человечества. Потребность в ней – одна из основных для современного человека. Б**о**льший процент знаний приходится на приобретённые либо вследствие непосредственного получения информации, либо в результате анализа уже существующих данных.

Объём информационных ресурсов в любой области растёт огромными темпами, это связано, прежде всего, с усложнением всех сфер жизнедеятельности общества. Кроме того, непрерывно увеличиваются массивы передаваемой информации, речь идёт не только о бытовых разговорах, но и всего инфопотока в Интернете в целом.

Технический прогресс не стоит на месте, и сейчас практически каждый компьютер подключается к сети для обмена какими-либо данными. Компьютерная сеть изначально является незащищённой и уязвимой для внешних атак системой. И для того, чтобы предотвратить несанкционированный доступ к устройству, подключенному к глобальной или частной сети, необходимо использовать специальные программные средства, называемые межсетевыми экранами (также известные, как сетевые фильтры, брандмауэры, Firewall-ы).

Цель данной работы - разработать межсетевой экран, осуществляющий контроль проходящего через него сетевого трафика, в виде загружаемого модуля.

Необходимо предоставить пользователю возможность задания правил фильтрации и изменения видимости модуля в системе.

Для достижения поставленной цели необходимо решить следующие задачи:

1) изучить основные принципы работы сети и межсетевых экранов;
2) ознакомиться со способом перехвата пакетов сети;
3) проанализировать особенность misc драйвера и основные принципы ра-

боты с ним;

4) изучить методы передачи информации из пространства пользователя в пространство ядра и наоборот;
5) реализовать межсетевой экран.

# 1 Аналитическая часть

## 1.1 Постановка задачи

Необходимо разработать межсетевой экран для операционной системы Linux, задача которого – анализировать проходящий через него сетевой трафик.

Программное обеспечение должно быть реализовано в виде драйвера и позволять пользователю добавлять/удалять правила фильтрации и изменять видимость самого модуля в системе.

Правила описывают те пакеты, которые считаются «запрещёнными» по той или иной причине, и представляют из себя перечисление параметров с принимаемыми значениями, по которым будет в дальнейшем осуществляться анализ пакетов.

## 1.2 Общие принципы работы сети

**Компьютерная сеть** – совокупность компьютеров и других устройств, соединённых линиями связи и обменивающихся информацией между собой в соответствии с определёнными правилами – **протоколами**. [1]

Информация преобразуется в пакеты и передаётся от одного компьютера к другому, и для этого используются протоколы. Каждый пакет проходит несколько стадий, которые определены в модели OSI (наглядно представлена в виде таблицы 1).

Таблица 1: Модель OSI

| № | Название |
|---|---|
| 7 | Прикладной уровень |
| 6 | Уровень представления |
| 5 | Сеансовый уровень |
| 4 | Транспортный уровень |
| 3 | Сетевой уровень |
| 2 | Канальный уровень |
| 1 | Физический уровень |

Из пакета можно получить различную информацию, такую как:

- IP source – IP-адрес источника;
- IP destination – IP-адрес назначения;
- source port - порт источника;
- destination port - порт назначения (для известных сервисов порты зарезервированы и известны заранее);
- TCP/UDP - протоколы транспортного уровня;
- другое.

Для каждого приложения ведутся две системные очереди: очередь данных, поступающих к приложению из сети, и очередь данных, отправляемых этим приложением в сеть. Такие очереди называются **портами**, причём входная и выходная очереди одного приложения рассматриваются как один порт. Для идентификации портов им присваивают номера.

### 1.3 Межсетевой экран

**Межсетевой экран** – это программное средство, предназначенное для фильтрации входящего и исходящего трафика, в соответствии с некоторыми

заранее заданными критериями, правилами, тем самым, осуществляя защиту компьютера от сетевых угроз.

В общих чертах его работа изображена на Рисунке 1.1.


Рисунок 1.1 – Принцип работы межсетевого экрана

Межсетевой экран в основном работает на пакетном уровне (уровни 3 и 4 модели OSI, которая представлена в таблице 1). [2]

Пакеты могут анализироваться в соответствии со следующими критериями:

- формальная корректность пакета;
- направление (входящий или исходящий);
- тип протокола;
- порт (источника, назначения);
- и т.д.

Для того, чтобы отфильтровать пакеты по тому или иному признаку, необходимо задать соответствующие правила, которые создаются пользователем. Когда межсетевой экран перехватывает пакет, он просматривает все его поля и сравнивает их с правилами из таблицы, причём в том порядке, как они были заданы. Если совпадение было найдено, то пакет дальше никуда не передаётся.

### 1.4 Загружаемый модуль ядра

Для того, чтобы добавить новый функционал в ядро Linux, нужно либо перекомпилировать его (что небезопасно), либо воспользоваться загружаемым

модулем ядра. [3]

После загрузки модуля он становится частью операционной системы и ему доступны все структуры и функции ядра. Когда функциональность, предоставляемая модулем, больше не требуется, то он может быть выгружен.

В Linux все модули обычно хранятся в каталоге /lib/modules и имеют расширение .ko. Модули загружаются и выгружаются с помощью специальных команд, приведённых в Листинге 1.

Листинг 1: Команды для загрузки и выгрузки загружаемого модуля ядра

```
1 // загрузить модуль в ядро
2 insmod имя< модуля.ko>
3
4 // выгрузить модуль из ядра
5 rmmod имя< модуля>
```

Загружаемый модуль ядра должен иметь определённую структуру. Обязательной частью любого загружаемого модуля являются.

- **Функция загрузки (инициализации) модуля.** В заголовке используется макрос __init__. Её задачей является подготовка модуля для дальнейшего функционирования как части ядра ОС.
- **Функция выгрузки модуля.** В заголовке используется макрос __exit__. Его задачей является освобождение ресурсов, занимаемых модулем в конце его работы.
- **Макросы module_init(init_func), module_exit(exit_func)** нужны для формирования кода, который будет выполняться при загрузке/выгрузке модуля (в частности внутри этого кода содержится вызов функций __init__ и __exit__, переданных в макросы).
- **Макрос MODULE_LICENSE(char* license)** сообщает, под какой лицензией находится модуль. Обычно указывается ”GPL”.

Приведённые первые две функции являются **точками входа**. Также в модуле

может быть указано следующее.

- **MODULE_AUTHOR(”Name”), MODULE_DESCRIPTION(”LAB”)** – макросы, определяющие информацию о модуле (имя автора, краткое описание).
- **Дополнительные точки входа.** Это можно сделать, например, с помощью структуры file_operations, где можно указать функции модуля в качестве точек входа в него при различных действиях с файлом.

### 1.5 Изменение видимости модуля

Для того, чтобы посмотреть все загруженные модули можно воспользоваться командой **lsmod**, которая читает /proc/modules и отображает информацию о файле (название, размер, сколько раз и какой модуль использует его) в отформатированном списке. [4]

Модуль описывается в системе с помощью структуры **struct module** (Листинг 2).

Листинг 2: struct module

```
1   struct module
2   {
3       enum module_state state;
4
5       /* Member of list of modules */
6       struct list_head list;
7
8       /* Unique handle for this module */
9       char name[MODULE_NAME_LEN];
10      ...
11  };
```

В этой структуре содержится поле list – элемент связного списка модулей. При загрузке ядро добавляет модуль в список. При выгрузке, наоборот, исключает.

Скрытие модуля подразумевает в себе задачу удаления соответствующего элемента из этого списка. А восстановление видимости, наоборот, добавление.

Для взаимодействия со списком модулей необходимо использовать структуру **struct list_head** и функции **list_add**, **list_del**. Перед тем, как удалить модуль, следует сохранить указатель, чтобы можно было восстановить его видимость. [5]

### 1.6 Управление внешними устройствами

Одна из самых сложных задач системы – управление внешними устройствами. Их достаточно много (мыши, клавиатуры, сканеры, принтеры и т.д.), все разных типов и выполняют разные задачи.

Структура **struct file_operations** (Листинг 3) используется для определения собственных функций для работы с конкретным устройством.

Листинг 3: struct file_operations

```
1  struct file_operations {
2      struct module *owner;
3      ...
4      ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
5      ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
6      ...
7      int (*open) (struct inode *, struct file *);
8      ...
9      int (*release) (struct inode *, struct file *);
10     ...
11 }
```

Файл устройства – это специальный файл, который обеспечивает связь между файловой системой и драйверами устройств. В отличие от обычных файлов они являются только указателями на соответствующие драйверы в ядре.

Файлы устройств имеют три дополнительных атрибута, которые характеризуют устройство, соответствующее данному файлу:

1) **класс устройства**:
    - блок-ориентированные (блочные) (пример: жёсткий диск) передают данные блоками, взаимодействие только через буферную память;
    - байт-ориентированные (символьные) (пример: принтер) передают данные посимвольно, как непрерывный поток байт, для взаимодействия буфер не требуется;
2) **старший номер** устройства, обозначающий тип устройства (посмотреть их можно в /proc/devices);
3) **младший номер** устройства применяется для нумерации устройств одного типа, т. е. устройств с одинаковыми старшими номерами.

Для того, чтобы идентифицировать устройство, используются старший и младший номера. [3]

### 1.7 Драйвер в ОС Linux

**Драйвер** – особая программа, является частным случаем загружаемого модуля ядра. Есть отличительная особенность – множество точек входа (более 2), и все определены в соответствующей структуре. Основная задача драйвера – управление внешними устройствами. [3]

В Linux драйверы делятся на 3 типа:

1) **встроенные в ядро**
    - инициализируются при запуске системы;
    - позволяют автоматически находить соответствующие устройства при обращении к ним;
2) **реализованные как загружаемые модуля ядра**;
3) **код модулей поделён между ядром и специальной утилитой**
    - например, у драйвера принтера ядро отвечает за взаимодействие с параллельным портом, а формирование управляющих сигналов выполняет демон печати.

Межсетевой экран – драйвер второго типа, поскольку может быть свобод-

но загружен и выгружен пользователем.

### 1.7.1 Misc Device Driver

**Misc** (от слова ”miscellaneous”) **драйвером** называется простой символьный драйвер. Его используют в случаях, когда возможно использовать упрощение: не задавать самостоятельно старший номер устройства (major), поскольку он определён заранее и равен 10. Но разработчик должен задать младший номер в пределах от 1 до 255. [6]

Такой подход позволяет сэкономить оперативную память, особенно, если необходимо создать несколько символьных драйверов для нескольких простых устройств.

На misc драйверах могут быть определены операции open, read, write, close и т.д. И так же создаётся файл /dev/{misc_file}.

По аналогии с символьными драйверами, которые описываются структурой struct cdev, misc драйверы описываются с помощью структуры **struct miscdevice**, поля которой приведены в Листинге 4.

Листинг 4: struct miscdevice

```
1 struct miscdevice {
2     int minor;
3     const char *name;
4     struct file_operations *fops;
5     umode_t i_mode;
6     struct miscdevice *next, *prev;
7 };
```

Поле minor может принимать значение MISC_DYNAMIC_MINOR, означающее, что он будет назначен автоматически.

Для работы с таким драйвером используются функции, представленные в Листинге 5.

Листинг 5: Функции для регистрации и удаления misc драйвера

```
1    // регистрация драйвера
2    int misc_register(struct miscdevice *misc);
3
4    // удаление драйвера
5    int misc_deregister(struct miscdevice *misc);
```

## 1.8 netfilter

netfilter – это механизм фильтрации сетевых пакетов, который позволяет отслеживать их перемещение и при необходимости можно перехватить, блокировать. [7]

### 1.8.1 Точки перехвата

Путь сетевого пакета в ядре изображен на Рисунке 1.2.

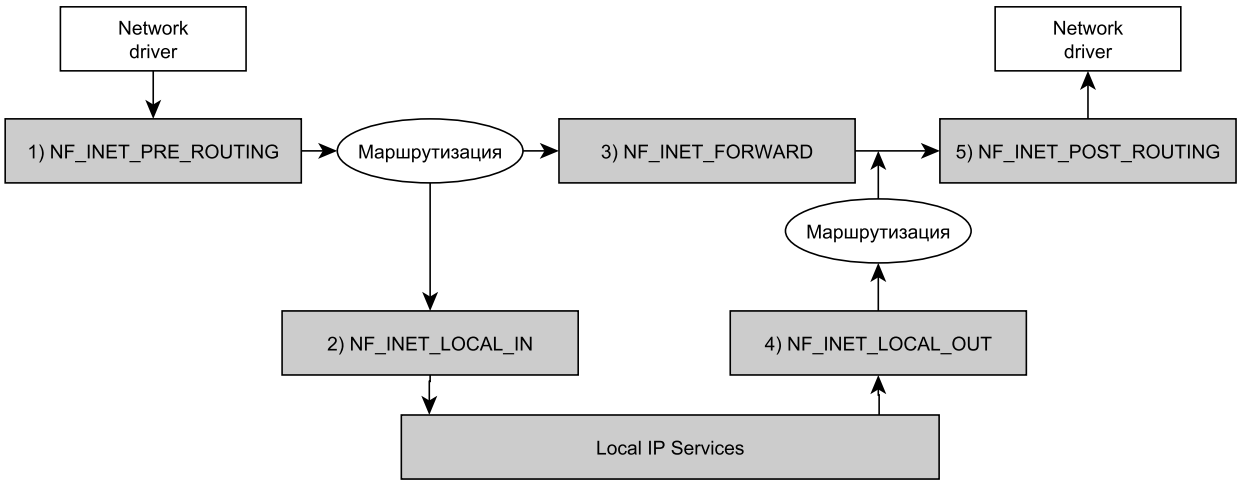


Рисунок 1.2 – Путь сетевого пакета

Предоставляется 5 точек, на которых могут быть определены функции перехвата, которые называются **хук-функциями**:

1) NF_INET_PRE_ROUTING – для всех входных пакетов;
2) NF_INET_LOCAL_IN – используется, чтобы перехватить пакеты, предназначенные для локального процесса;

3) NF_INET_FORWARD – используется для пакетов, предназначенных для другого интерфейса;
4) NF_INET_LOCAL_OUT – для пакетов, которые создают локальные процессы;
5) NF_INET_POST_ROUTING – для пакетов, которые уже настроены для дальнейшего прохождения по сети к своему адресату и готовы покинуть текущий сетевой стек.

### 1.8.2 Хук-функции

Для того, чтобы использовать хук-функцию, необходимо сначала заполнить структуру **struct nf_hook_ops**. Структура с основными полями приведена в Листинге 6.

Листинг 6: struct nf_hook_ops

```
1 struct nf_hook_ops {
2     nf_hookfn           *hook;
3     ...
4     u_int8_t            pf;
5     unsigned int        hooknum;
6     int                 priority;
7 };
```

В структуре находятся следующие поля:

- **hook** – функция, которая будет вызвана для обработки пакета, принимается решение отбросить или принять пакет;
- **pf** – семейство протоколов;
- **hooknum** – точка перехвата;
- **priority** – приоритет.

Регистрация и удаление хуков осуществляется посредством вызова функций, которые представлены в Листинге 7.

Листинг 7: Функции для регистрации и удаления хук-функций

```
1  // регистрация
2  int nf_register_net_hook(struct net *net, const struct nf_hook_ops *ops);
3
4  // удаление
5  void nf_unregister_net_hook(struct net *net, const struct nf_hook_ops *
   ops);
```

## 1.9 Передача данных из адресного пространства пользователя в адресное пространство ядра и наоборот

Поскольку правила фильтрации пакетов задаются пользователем, то необходимо передавать данные из адресного пространства пользователя в адресное пространство ядра.

И так как межсетевой экран сохраняет их у себя для дальнейшей работы, действующие правила можно посмотреть, вызвав соответствующую команду, то есть, необходимо также обеспечивать передачу данных и в обратную сторону.

Для осуществления таких задач используются специальные функции (Листинг 8).

Листинг 8: Специальные функции

```
1  // копирование буфера из пользовательского пространства в пространство
   ядра
2  unsigned long copy_from_user(void * to, const void __user * from,
   unsigned long n);
3
4  // копирует данные из ядра в пространство пользователя
5  unsigned long copy_to_user(void __user * to, const void * from, unsigned
   long n);
```

## 1.10 Выводы из аналитического раздела

В этом разделе была формализована поставленная задача, рассмотрены основные её этапы, также подробно изучены принципы работы межсетевого

экрана.

Для достижения поставленной цели было принято решение использовать простой misc драйвер, поскольку он ориентирован на выполнение небольших задач и имеет упрощённую схему создания.

# 2 Конструкторская часть

## 2.1 Требования к программе

Межсетевой экран должен быть реализован в виде загружаемого модуля, и предоставлять следующие возможности:

- добавление нового правила;
- удаление правила;
- просмотр всех правил;
- сокрытие модуля в системе;
- обнаружение модуля в системе.

Для непосредственного взаимодействия с пользователем необходимо разработать отдельную программу, представляющую из себя интерфейс для работы с загружаемым модулем. Программа выполняет не только функцию связующего звена между пользователем и межсетевым экраном, но и проверку входных данных: корректность вводимых команд и правил.

## 2.2 Инициализация модуля

На Рисунке 2.1 представлена подробная схема алгоритма инициализации модуля.

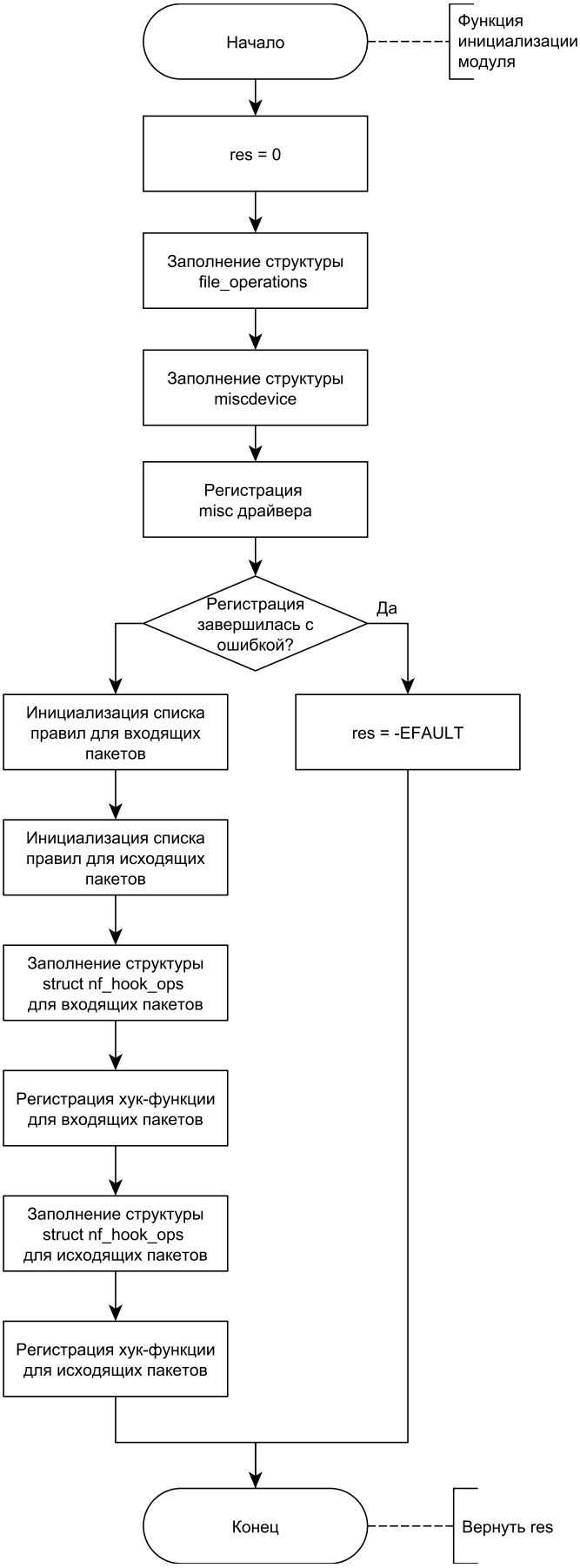


Рисунок 2.1 – Схема инициализации модуля

## 2.3 Завершение работы модуля

При выгрузке необходимо освободить все ресурсы, которые были зарезервированы в процессе работы модуля.

Детально работа этой функции изложена на Рисунке 2.2.


Рисунок 2.2 – Схема выгрузки модуля

## 2.4 Основные функции, определяемые в struct file_operations

При инициализации модуля одним из первых действий является заполнение структуры **struct file_operations**, в которой определяются функции для работы с файлами. В рамках поставленной задачи необходимо указать свои функции read (для того, чтобы получить список всех правил), write (используется для обработки нового правила).

На Рисунках 2.3 – 2.4 приведены схемы для функций чтения и записи.

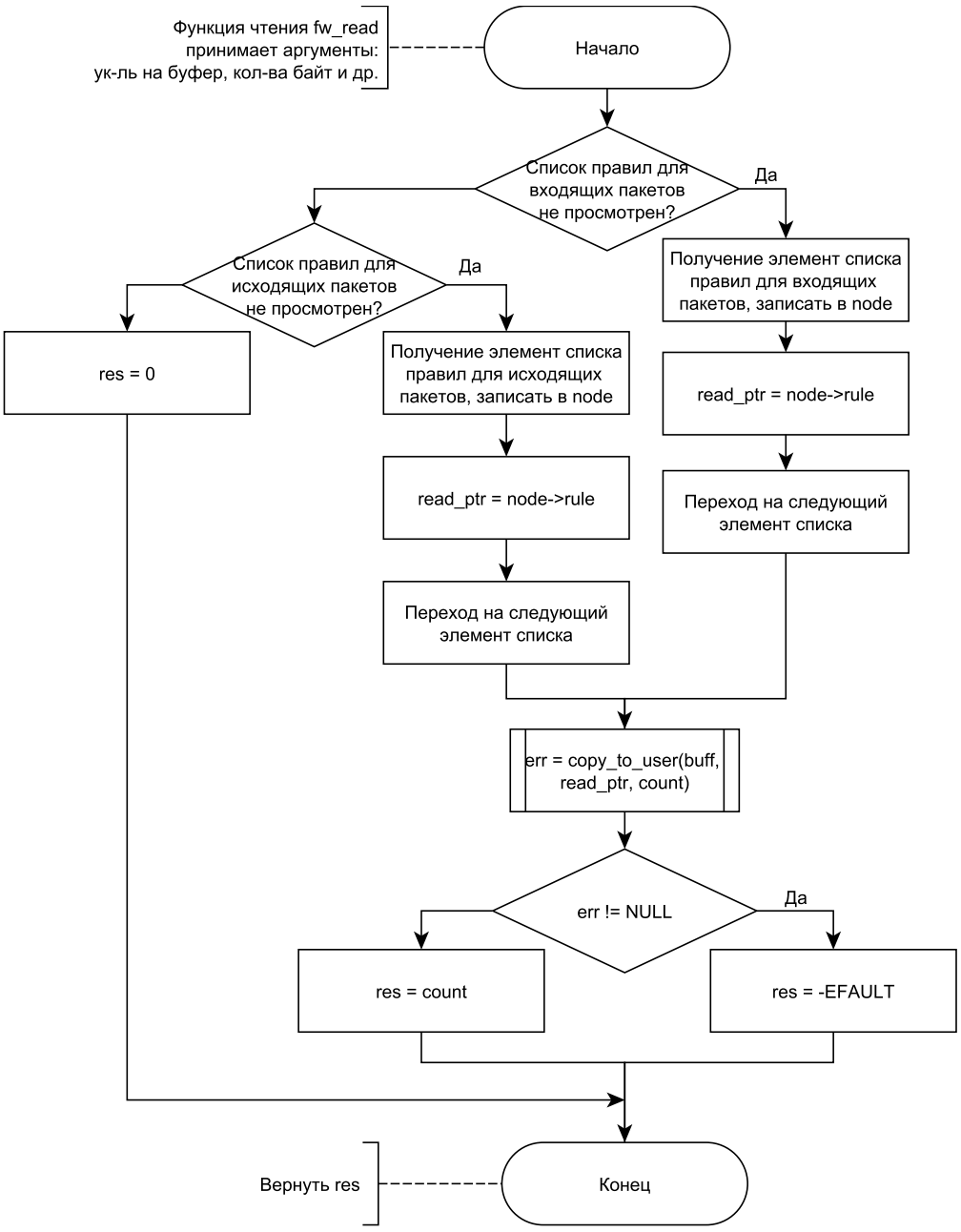


Рисунок 2.3 – Схема работы функции чтения

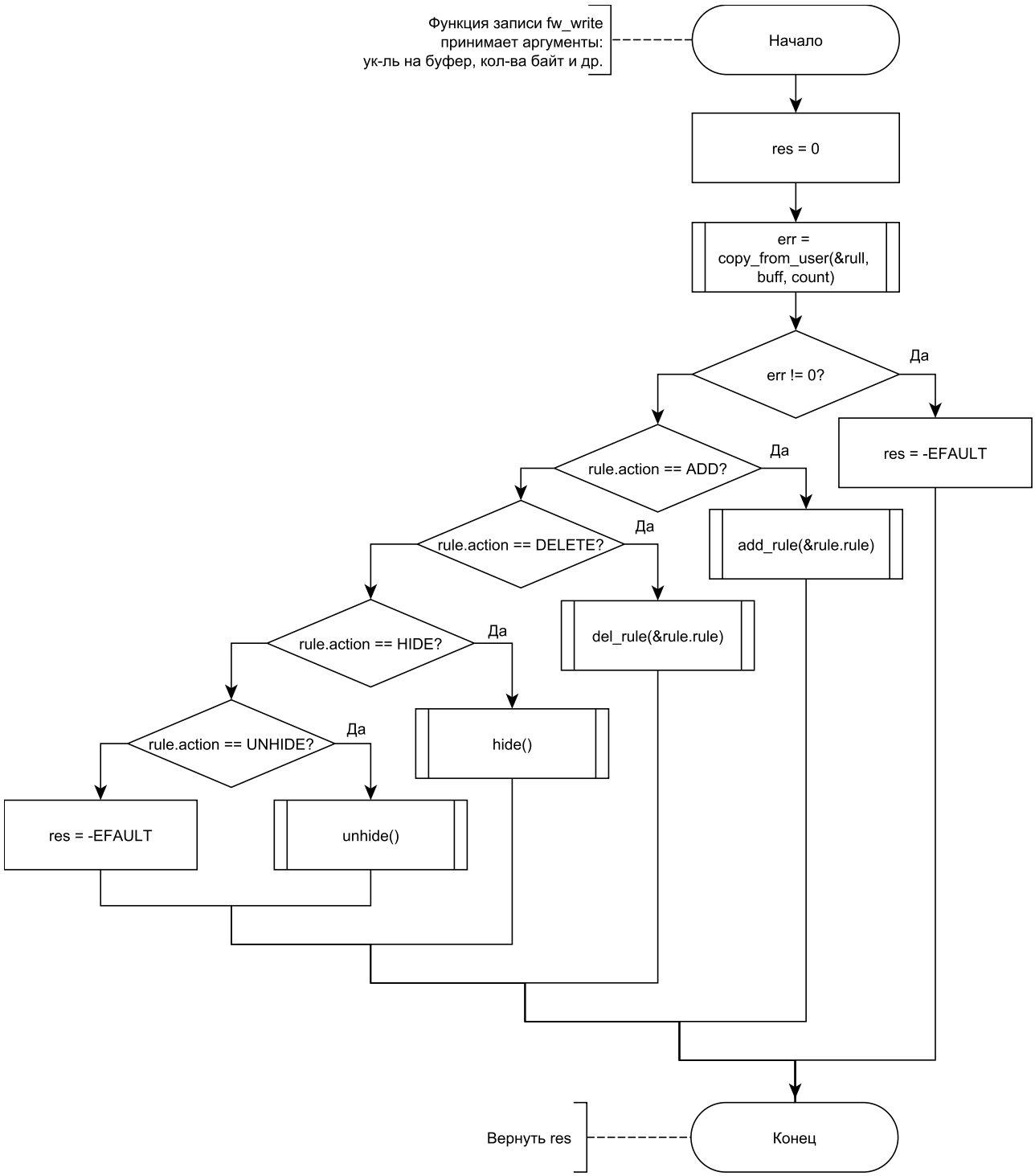


Рисунок 2.4 – Схема работы функции записи

## 2.5 Функции добавления и удаления правил

Пользователю предоставляется возможность добавить новое правило или удалить уже существующее. На Рисунках 2.5 и 2.6 показан общий подход к реализации каждой из них.

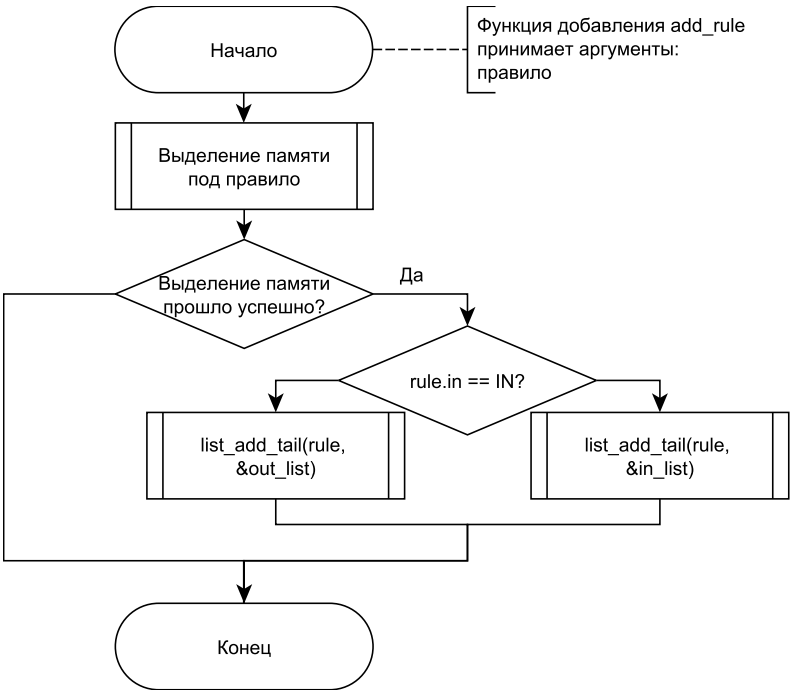


Рисунок 2.5 – Схема работы функции добавления нового правила

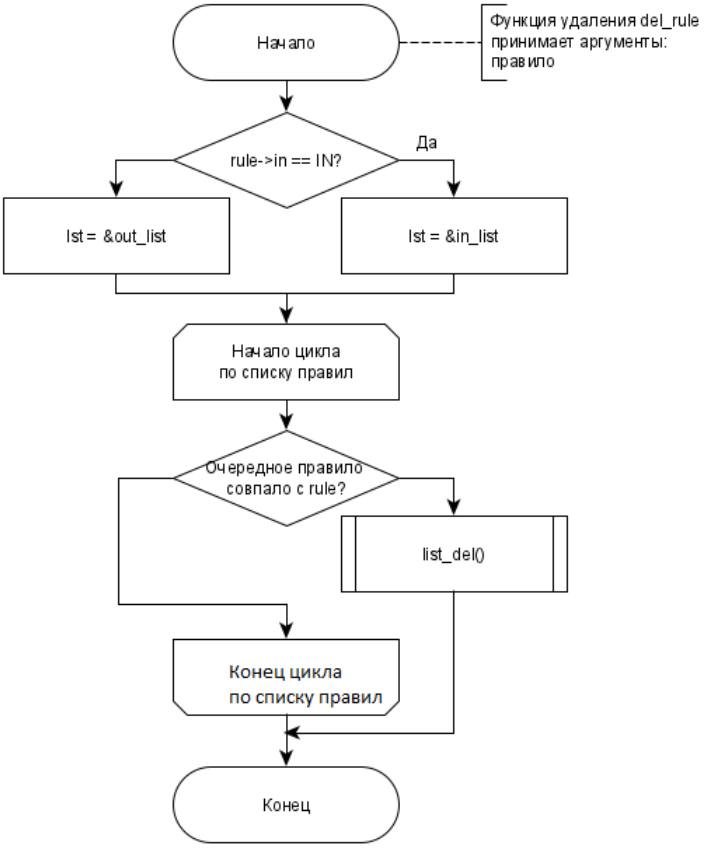


Рисунок 2.6 – Схема работы функции удаления правила

## 2.6 Функция фильтрации пакетов

В процессе инициализации модуля также происходит регистрация хук-функций, заданных в структуре **struct nf_hook_ops** и необходимых для работы межсетевого экрана.

В рамках поставленной задачи регистрируются две функции: для обработки входящих и исходящих пакетов. Поскольку главное их отличие – направление анализируемых единиц, то рекомендуется реализовать одну функцию, в которую подаётся соответствующий список правил. Детали работы этой функции представлены на Рисунках 2.7 – 2.8.

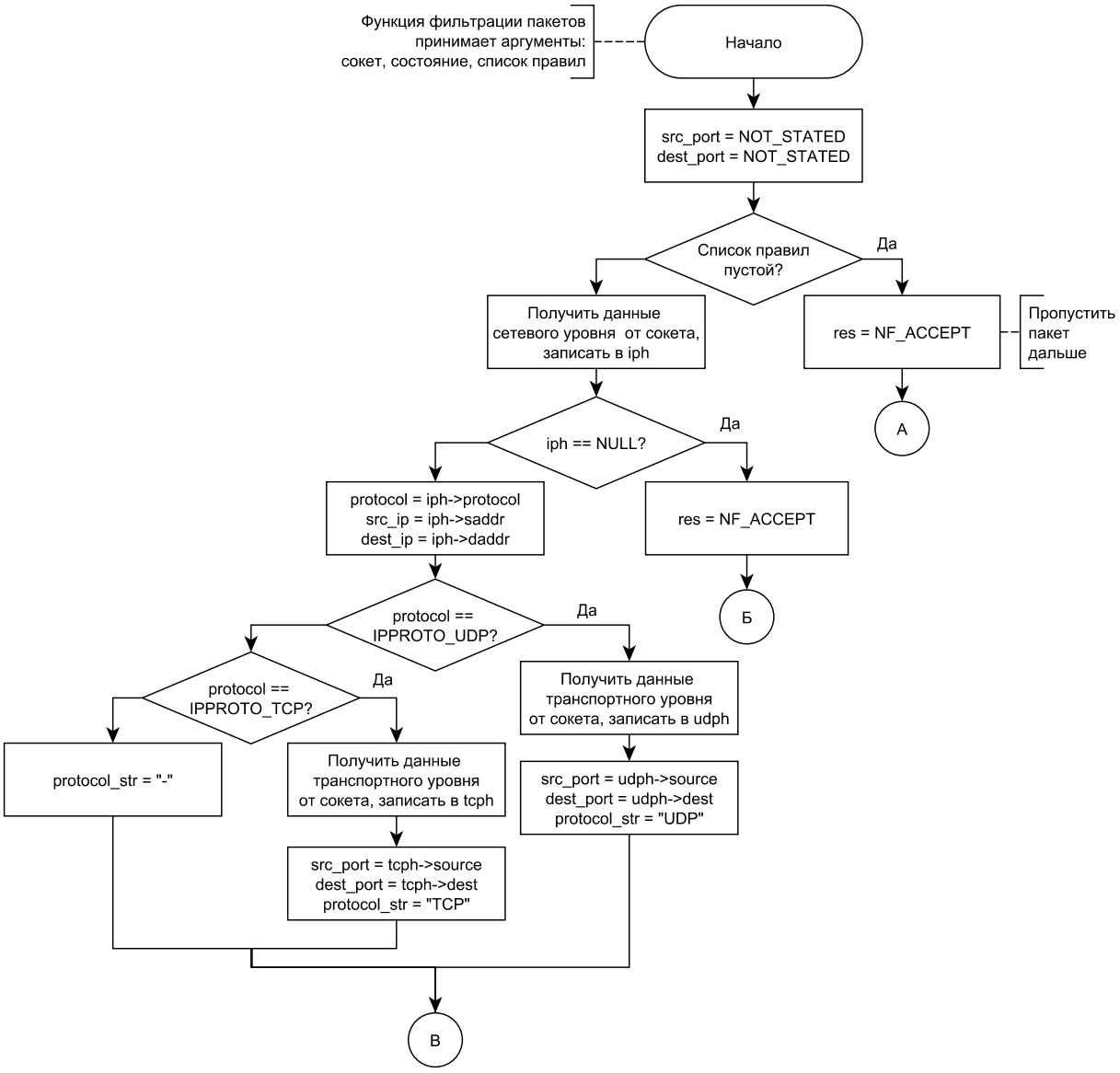


Рисунок 2.7 – Схема работы функции фильтрации пакетов

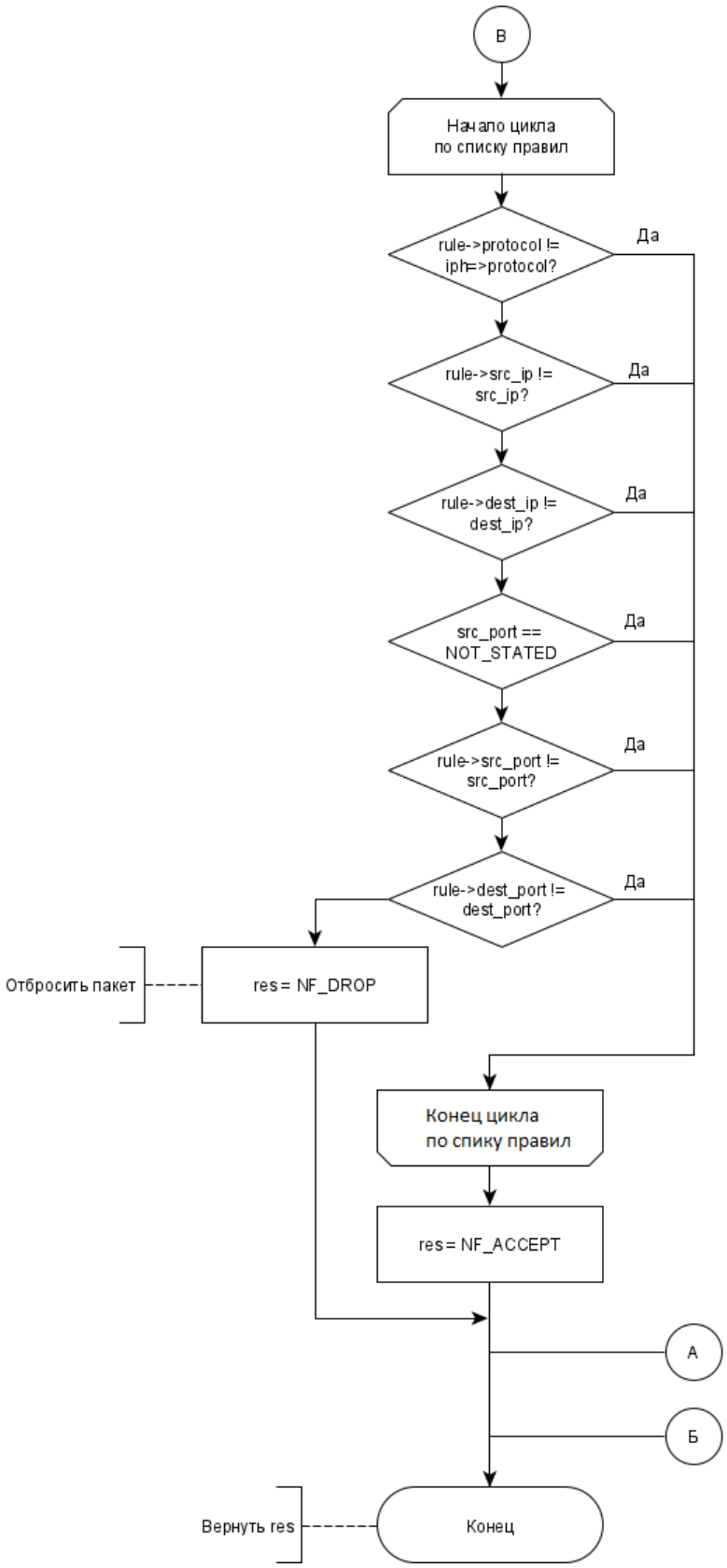


Рисунок 2.8 – Схема работы функции фильтрации пакетов (продолжение)

## 2.7 Выводы

В разделе рассмотрены требования к программе, основные сведения о модуле, а также предоставлены схемы, описывающие ключевые моменты в его работе.

# 3 Технологическая часть

## 3.1 Выбор языка программирования

В качестве языка программирования был выбран C. [8] Для сборки модуля использовалась утилита make.

Была выбрана среда разработки Visual Studio Code [9], так как она бесплатная, кроссплатформенная, а также позволяет использовать все возможности консоли, не переключаясь между окнами.

## 3.2 Структура модуля

Реализованный модуль включает в себя следующие функции:

- **fw_init()** – функция инициализации модуля;
- **fw_exit()**– функция выгрузки модуля;
- **hide()** – функция изменения видимости модуля (скрытие);
- **unhide()** – функция изменения видимости модуля (обнаружение);
- **fw_read(struct file *filp, char __user *buff, size_t count, loff_t *f_pos)** – функция чтения, описываемая в структуре struct file_operations;
- **fw_write(struct file *filp, const char __user *buff, size_t count, loff_t *f_pos)** – функция записи, описываемая в структуре struct file_operations;
- **add_rule(struct fw_rule *rule)** – добавление нового правила;
- **del_rule(struct fw_rule *rule)** – удаление правила;
- **fw_in_filter(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)** – «обёртка» функции фильтрации для входящих пакетов;
- **fw_out_filter(void *priv, struct sk_buff *skb, const struct nf_hook_state *state)** – «обёртка» функции фильтрации для исходящих пакетов;
- **filter(void *priv, struct sk_buff *skb, const struct nf_hook_state *state, struct list_head *list_rule)** – основная функция фильтрации пакетов;
- **str_rule(struct fw_rule *rule)** – функция преобразования правила фильтрации в удобный для восприятия человеком вид;

- **str_packet(uint32_t src_ip, uint16_t src_port,uint32_t dest_ip, uint16_t dest_port, char *protocol_str)** – функция преобразования информации о перехваченном пакете в удобный для восприятия человеком вид.

Были также определены структуры:

- struct file_operations;
- struct miscdevice;
- struct nf_hook_ops.

В Приложении А представлены листинги кода программы.

### 3.3 Makefile

### 3.4 Демонстрация работы модуля

### 3.5 Вывод

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

## Список литературы


1. Рогозин Н.О., Курс лекций по дисциплине «Компьютерные сети» [Текст]
2. Программные межсетевые экраны [Электронный ресурс]. – Режим доступа: https://www.securitylab.ru/analytics/240197.php (дата обращения 01.10.2021).
3. Рязанова Н.Ю., Курс лекций по дисциплине «Операционные системы» [Текст]
4. lsmod [Электронный ресурс]. – Режим доступа: https://www.opennet.ru/man.shtml?topic=lsmod&category=8&russian=2
5. Простая маскировка модуля ядра Linux с применением DKOM [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/205274/
6. Misc Device Driver – Linux Device Driver Tutorial Part 32 [Электронный ресурс]. – Режим доступа: https://embetronicx.com/tutorials/linux/device-drivers/misc-device-driver/#Misc_Device_Driver (дата обращения 05.10.2021).
7. NETFILTER [Электронный ресурс]. – Режим доступа: http://samag.ru/archive/article/169 (дата обращения 07.10.2021).
8. Документвция по языку C [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/cpp/c-language/?view=msvc-170
9. Документация по Visual Studio [Электронный ресурс]. – Режим доступа: https://code.visualstudio.com/docs

# ПРИЛОЖЕНИЕ А

Листинг 9: Загружаемый модуль

```
#define IP_POS(ip, pos) (ip >> ((8 * (3 - pos))) & 0xFF)
#define SAME_ADDR(ip1, ip2) ((ip1 ^ ip2) == 0)


MODULE_LICENSE("GPL");
MODULE_AUTHOR("Bryanskaya Ekaterina <bryanskayakatya@yandex.ru>");

static char *buffer;
static int flag_hidden = 0;

struct list_head in_list;
struct list_head out_list;

struct list_head *module_prev;

struct rule_item {
    struct fw_rule rule;
    struct list_head list;
};

void hide(void)
{
    if (flag_hidden)
        return;

    module_prev = THIS_MODULE->list.prev;
    list_del(&THIS_MODULE->list);
    flag_hidden = 1;

    printk(">>> FIREWALL: module was hidden");
}

void unhide(void)
{
    if (!flag_hidden)
        return;
```

```
37
38      list_add(&THIS_MODULE->list, module_prev);
39      flag_hidden = 0;
40
41      printk(">>> FIREWALL: module was exposed");
42  }
43
44  int fw_open(struct inode *inode, struct file *file)
45  {
46      printk(KERN_INFO ">>> FIREWALL: associated char device was opened");
47      return 0;
48  }
49
50  int fw_release(struct inode *inode, struct file *file)
51  {
52      printk(KERN_INFO ">>> FIREWALL: associated char device was closed");
53      return 0;
54  }
55
56
57  char* str_rule(struct fw_rule *rule)
58  {
59      int count_bytes = 0;
60
61      char *res = kmalloc(BUF_LEN, GFP_KERNEL);
62      if (!res)
63      {
64          printk(KERN_INFO "FIREWALL: error in formating rule");
65          return NULL;
66      }
67
68      if (rule->in == IN)
69          count_bytes += snprintf(res, 10, "IN \t ");
70      else if (rule->in == OUT)
71          count_bytes += snprintf(res, 10, "OUT \t ");
72      else
73          printk(KERN_INFO "%d", rule->in);
74
75      if (rule->src_ip != NOT_STATED)
76          count_bytes = snprintf(res, 30, "src_ip: %u.%u.%u.%u \t ",
```

```
77              IP_POS(rule->src_ip, 3),
78              IP_POS(rule->src_ip, 2),
79              IP_POS(rule->src_ip, 1),
80              IP_POS(rule->src_ip, 0));
81
82      if (rule->src_port != NOT_STATED)
83      count_bytes += snprintf(res + count_bytes, 20, "src_port: %u \t ", ntohs(
        rule->src_port));
84
85      if (rule->dest_ip != NOT_STATED)
86          count_bytes += snprintf(res + count_bytes, 30, "dest_ip: %u.%u.%u.%u
        \t ",
87              IP_POS(rule->dest_ip, 3),
88              IP_POS(rule->dest_ip, 2),
89              IP_POS(rule->dest_ip, 1),
90              IP_POS(rule->dest_ip, 0));
91
92      if (rule->dest_port != NOT_STATED)
93          count_bytes += snprintf(res + count_bytes, 20, "dest_port: %u \t ",
        ntohs(rule->dest_port));
94
95      if (rule->protocol != NOT_STATED)
96      {
97          if (rule->protocol == IPPROTO_TCP)
98          snprintf(res + count_bytes, 20, "protocol: TCP");
99          else if (rule->protocol == IPPROTO_UDP)
100         snprintf(res + count_bytes, 20, "protocol: UDP");
101     }
102
103     return res;
104 }
105
106 char* str_packet(uint32_t src_ip, uint16_t src_port,uint32_t dest_ip,
        uint16_t dest_port, char *protocol_str)
107 {
108     int count_bytes = 0;
109
110     char *res = kmalloc(BUF_LEN, GFP_KERNEL);
111     if (!res)
112     {
```

```
113         printk(KERN_INFO "FIREWALL: error in formating rule");
114         return NULL;
115     }
116 
117     if (src_ip != NOT_STATED)
118         count_bytes = snprintf(res, 30, "src_ip: %u.%u.%u.%u \t ",
119             IP_POS(src_ip, 3),
120             IP_POS(src_ip, 2),
121             IP_POS(src_ip, 1),
122             IP_POS(src_ip, 0));
123 
124     if (src_port != NOT_STATED)
125     count_bytes += snprintf(res + count_bytes, 20, "src_port: %u \t ", ntohs(
    src_port));
126     else
127     count_bytes += snprintf(res + count_bytes, 20, "src_port: - \t ");
128 
129     if (dest_ip != NOT_STATED)
130     count_bytes += snprintf(res + count_bytes, 30, "dest_ip: %u.%u.%u.%u \t "
    ,
131         IP_POS(dest_ip, 3),
132         IP_POS(dest_ip, 2),
133         IP_POS(dest_ip, 1),
134         IP_POS(dest_ip, 0));
135 
136     if (dest_port != NOT_STATED)
137         count_bytes += snprintf(res + count_bytes, 20, "dest_port: %u \t ",
    ntohs(dest_port));
138     else
139         count_bytes += snprintf(res + count_bytes, 20, "dest_port: - \t ");
140 
141     snprintf(res + count_bytes, 20, "protocol: %s", protocol_str);
142 
143     return res;
144 }
145 
146 static void add_rule(struct fw_rule *rule)
147 {
148     struct rule_item *node;
149 
```

```
150     node = (struct rule_item *)kmalloc(sizeof(struct rule_item), GFP_KERNEL);
151     if (node == NULL)
152     {
153         printk(KERN_INFO ">>> FIREWALL: addition a new rule was failed");
154         return;
155     }
156 
157     node->rule = *rule;
158 
159     if (node->rule.in == IN)
160     list_add_tail(&node->list, &in_list);
161     else
162     list_add_tail(&node->list, &out_list);
163 
164     printk(KERN_INFO ">>> FIREWALL: new rule was added. Rule: %s", str_rule
      (&(node->rule)));
165 }
166 
167 static void del_rule(struct fw_rule *rule)
168 {
169     struct list_head *lst, *temp;
170     struct rule_item *node;
171 
172     if (rule->in == IN)
173     lst = &in_list;
174     else
175     lst = &out_list;
176 
177     for (temp = lst; temp->next != lst; temp = temp->next)
178     {
179         node = list_entry(temp->next, struct rule_item, list);
180 
181         if (node->rule.in == rule->in &&
182         node->rule.src_ip == rule->src_ip &&
183         node->rule.src_port == rule->src_port &&
184         node->rule.dest_ip == rule->dest_ip &&
185         node->rule.dest_port == rule->dest_port &&
186         node->rule.protocol == rule->protocol)
187         {
188             list_del(temp->next);
```

```
189             kfree(node);
190
191             printk(KERN_INFO ">>> FIREWALL: rule was removed. Rule: %s",
    str_rule(rule));
192
193             return;
194         }
195     }
196
197     printk(KERN_INFO ">>> FIREWALL: rule was not found. Rule: %s", str_rule(
    rule));
198 }
199
200 ssize_t fw_read(struct file *filp, char __user *buff, size_t count, loff_t *
    f_pos)
201 {
202     static struct list_head *in_lst = &in_list;
203     static struct list_head *out_lst = &out_list;
204     struct rule_item *node;
205     char *read_ptr;
206
207     if (in_lst->next != &in_list)
208     {
209         node = list_entry(in_lst->next, struct rule_item, list);
210         read_ptr = (char *)&node->rule;
211         in_lst = in_lst->next;
212     }
213     else if (out_lst->next != &out_list)
214     {
215         node = list_entry(out_lst->next, struct rule_item, list);
216         read_ptr = (char *)&node->rule;
217         out_lst = out_lst->next;
218     }
219     else
220     {
221         in_lst = &in_list;
222         out_lst = &out_list;
223
224         return 0;
225     }
```

```
226
227     if (copy_to_user(buff, read_ptr, count))
228     {
229         printk(KERN_INFO ">>> FIREWALL: copy_to_user error");
230         return -EFAULT;
231     }
232
233     return count;
234 }
235
236 ssize_t fw_write(struct file *filp, const char __user *buff, size_t count,
        loff_t *f_pos)
237 {
238     struct fw_comm rule_full;
239
240     if (count < sizeof(struct fw_comm))
241     {
242         printk(KERN_INFO ">>> FIREWALL: incorrect rule");
243         return -EFAULT;
244     }
245
246     if (copy_from_user(&rule_full, buff, count))
247     {
248         printk(KERN_INFO ">>> FIREWALL: copy_from_user error");
249         return -EFAULT;
250     }
251
252     switch (rule_full.action)
253     {
254         case ADD:
255             add_rule(&rule_full.rule);
256             break;
257         case DELETE:
258             del_rule(&rule_full.rule);
259             break;
260         case HIDE:
261             hide();
262             break;
263         case UNHIDE:
264             unhide();
```

```
265             break;
266         default:
267             printk(KERN_INFO ">>> FIREWALL: unknown command");
268             break;
269     }
270 
271     return 0;
272 }
273 
274 static unsigned int filter(void *priv, struct sk_buff *skb, const struct
      nf_hook_state *state,
275 struct list_head *list_rule)
276 {
277     struct iphdr *iph;  /* An IPv4 packet header */
278     struct tcphdr *tcph;
279     struct udphdr *udph;
280 
281     unsigned char protocol;
282     char *protocol_str;
283     uint32_t src_ip, dest_ip;
284     uint16_t src_port = NOT_STATED, dest_port = NOT_STATED;
285 
286     struct list_head *lst;
287     struct rule_item *node;
288     struct fw_rule *rule;
289 
290     if (!skb || list_rule->next == list_rule)
291     return NF_ACCEPT;
292 
293     iph = (struct iphdr *)skb_network_header(skb);
294     if (iph == NULL)
295     return NF_ACCEPT;
296 
297     protocol = iph->protocol;
298     src_ip = iph->saddr;
299     dest_ip = iph->daddr;
300 
301     if (protocol == IPPROTO_UDP)
302     {
303         udph = (struct udphdr *)(skb_transport_header(skb));
```

```
304         src_port = udph->source;
305         dest_port = udph->dest;
306         protocol_str = "UDP";
307     }
308     else if (protocol == IPPROTO_TCP)
309     {
310         tcph = (struct tcphdr *)(skb_transport_header(skb));
311         src_port = tcph->source;
312         dest_port = tcph->dest;
313         protocol_str = "TCP";
314     }
315     else
316     protocol_str = "-";
317 
318     lst = list_rule;
319     list_for_each_entry(node, lst, list)
320     {
321         rule = &node->rule;
322 
323         if (rule->protocol != NOT_STATED && rule->protocol != iph->protocol)
324         continue;
325 
326         if (rule->src_ip != NOT_STATED && !SAME_ADDR(rule->src_ip, src_ip))
327         continue;
328 
329         if (rule->dest_ip != NOT_STATED && !SAME_ADDR(rule->dest_ip, dest_ip)
    )
330         continue;
331 
332         if (src_port == NOT_STATED)
333         continue;
334 
335         if (rule->src_port != NOT_STATED && rule->src_port != src_port)
336         continue;
337 
338         if (rule->dest_port != NOT_STATED && rule->dest_port != dest_port)
339         continue;
340 
341         printk(KERN_INFO ">>> FIREWALL: packet was dropped. Details: %s",
342         str_packet(src_ip, src_port, dest_ip, dest_port, protocol_str));
```

```
343
344             return NF_DROP; /* discarded the packet */
345         }
346
347         return NF_ACCEPT;   /* the packet passes, continue iterations */
348 }
349
350 static unsigned int fw_in_filter(void *priv, struct sk_buff *skb, const
        struct nf_hook_state *state)
351 {
352         return filter(priv, skb, state, &in_list);
353 }
354
355 static unsigned int fw_out_filter(void *priv, struct sk_buff *skb, const
        struct nf_hook_state *state)
356 {
357         return filter(priv, skb, state, &out_list);
358 }
359
360 static struct file_operations fw_fops = {
361         .owner = THIS_MODULE,
362         .read = fw_read,
363         .write = fw_write,
364         .open = fw_open,
365         .release = fw_release,
366 };
367
368 struct miscdevice dev = {
369         .minor = MISC_DYNAMIC_MINOR,
370         .name = DEVICE_FNAME,
371         .fops = &fw_fops,
372         .mode = S_IRWXU | S_IWGRP | S_IWOTH | S_IROTH,
373 };
374
375 static struct nf_hook_ops fw_in_hook_ops =
376 {
377         .hook = fw_in_filter,
378         .pf = PF_INET,
379         .hooknum = NF_INET_PRE_ROUTING,
380         .priority = NF_IP_PRI_FIRST
```

```
381 };
382
383 static struct nf_hook_ops fw_out_hook_ops =
384 {
385     .hook = fw_out_filter,
386     .pf = PF_INET,
387     .hooknum = NF_INET_LOCAL_OUT,
388     .priority = NF_IP_PRI_FIRST
389 };
390
391 static int __init fw_init(void)
392 {
393     int res = 0;
394
395     res = misc_register(&dev);
396     if (res)
397     {
398         printk(KERN_INFO ">>> FIREWALL: registration was failed");
399         return res;
400     }
401
402     buffer = (char *)kmalloc(sizeof(struct fw_comm*), GFP_KERNEL);
403     if (buffer == NULL)
404     {
405         printk(KERN_INFO ">>> FIREWALL: kmalloc error");
406         return -EFAULT;
407     }
408
409     INIT_LIST_HEAD(&in_list);
410     INIT_LIST_HEAD(&out_list);
411
412     nf_register_net_hook(&init_net, &fw_in_hook_ops);
413     nf_register_net_hook(&init_net, &fw_out_hook_ops);
414
415     printk(KERN_INFO ">>> FIREWALL: module was loaded. Major number of char
      device %s is 10, minor is %d",
416     DEVICE_FNAME, dev.minor);
417
418     return res;
419 }
```

```
420
421 static void __exit fw_exit(void)
422 {
423     struct rule_item *node;
424     struct rule_item *node_temp;
425
426     kfree(buffer);
427
428     list_for_each_entry_safe(node, node_temp, &in_list, list)
429     {
430         printk(KERN_INFO ">>> FIREWALL: rule was removed. Rule: %s", str_rule
      (&(node->rule)));
431         list_del(&node->list);
432         kfree(node);
433     }
434
435     list_for_each_entry_safe(node, node_temp, &out_list, list)
436     {
437         printk(KERN_INFO ">>> FIREWALL: rule was removed. Rule: %s", str_rule
      (&(node->rule)));
438         list_del(&node->list);
439         kfree(node);
440     }
441
442     misc_deregister(&dev);
443
444     nf_unregister_net_hook(&init_net, &fw_in_hook_ops);
445     nf_unregister_net_hook(&init_net, &fw_out_hook_ops);
446
447     printk(KERN_INFO ">>> FIREWALL: module was unloaded!\n");
448 }
449
450
451 module_init(fw_init);
452 module_exit(fw_exit);
```