



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
(Группа)

(Подпись, дата) Е.В. Брянская
(И.О.Фамилия)

Руководитель курсового проекта

(Подпись, дата) Н.Ю. Рязанова
(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	<u>Н.Ю. Рязанова</u> (Подпись, дата)	(И.О.Фамилия)
Студент	<u>Е.В. Брянская</u> (Подпись, дата)	(И.О.Фамилия)

Содержание

ВВЕДЕНИЕ	5
1 Аналитическая часть	6
1.1 Цель и задачи	6
1.2 Общие принципы работы сети	6
1.3 Межсетевой экран	7
1.4 Загружаемый модуль ядра	8
1.5 Изменение видимости модуля	10
1.6 Управление внешними устройствами	11
1.7 Драйвер в ОС Linux	12
1.7.1 Misc Device Driver	12
1.8 netfilter	13
1.8.1 Точки перехвата	14
1.8.2 Хук-функции	15
1.9 Выводы из аналитического раздела	16
2 Конструкторская часть	17
2.1 Требования к программе	17
2.2 Инициализация модуля	17
2.3 Завершение работы модуля	19
2.4 Основные функции, определяемые в struct file_operations	20
2.5 Функции добавления и удаления правил	21
2.6 Функция фильтрации пакетов	23
2.7 Выводы	24
3 Технологическая часть	25
3.1 Выбор языка программирования	25
3.2 Структура загружаемого модуля	25

3.3	Структура проекта	26
3.4	Сборка и запуск модуля	26
3.5	Демонстрация работы модуля	27
3.5.1	Команды и формат задания правил	27
3.5.2	Видимость модуля	28
3.5.3	Фильтрация по протоколу	29
3.5.4	Фильтрация по IP-адресу	31
3.6	Вывод	32
ЗАКЛЮЧЕНИЕ		33
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		34
ПРИЛОЖЕНИЕ А		36

ВВЕДЕНИЕ

Информация – это один из важнейших ресурсов, который представляет из себя движущую силу развития человечества. Потребность в ней – одна из основных для современного человека. Большой 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).

1 Аналитическая часть

1.1 Цель и задачи

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

Необходимо предоставить пользователю возможность задания правил фильтрации пакетов 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 работы с ним;
- 4) реализовать межсетевой экран.

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.

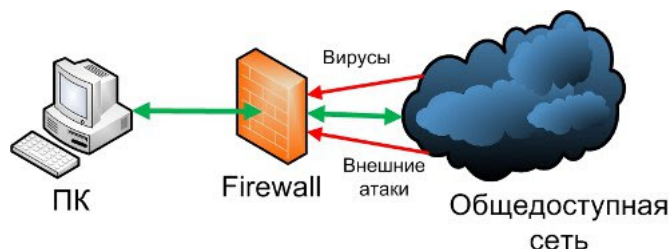


Рисунок 1.1 – Принцип работы межсетевого экрана

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      struct list_head list; /* Member of list of modules */
5      char name[MODULE_NAME_LEN]; /* Unique handle for this module */
6      ...
7  };
```

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 струк-

типу **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**
- 2) **реализованные как загружаемые модуля ядра;**
- 3) **код модулей поделён между ядром и специальной утилитой**

Межсетевой экран – драйвер второго типа, поскольку может быть свободно 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, 7]

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. [8]

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. [9]

1.8.1 Точки перехвата

Путь сетевого пакета в ядре изображен на Рисунке 1.2.

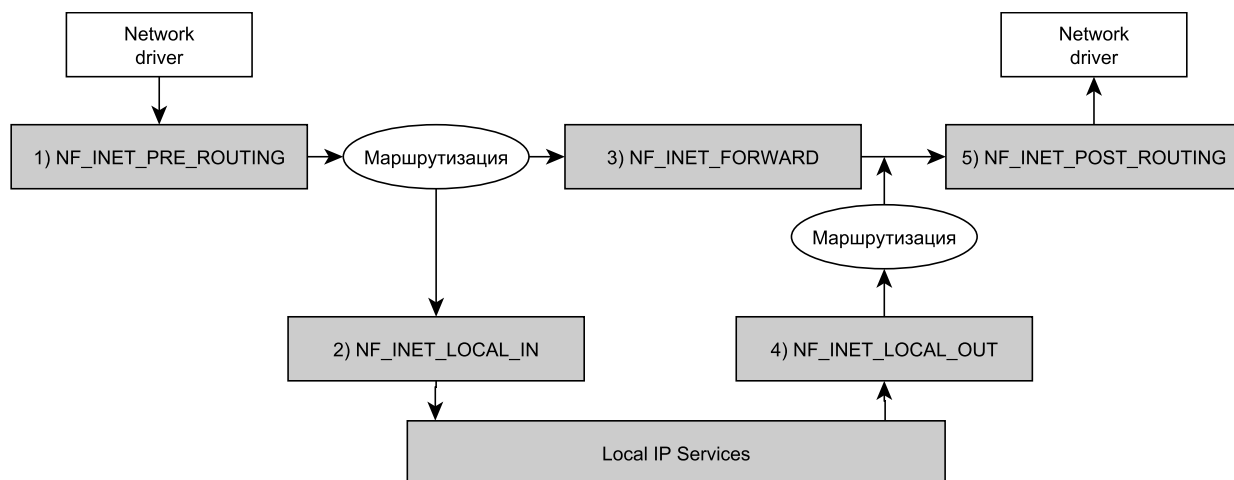


Рисунок 1.2 – Путь сетевого пакета

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 Выводы из аналитического раздела

В этом разделе были сформулированы цель и необходимые для её достижения задачи, рассмотрены основные 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.

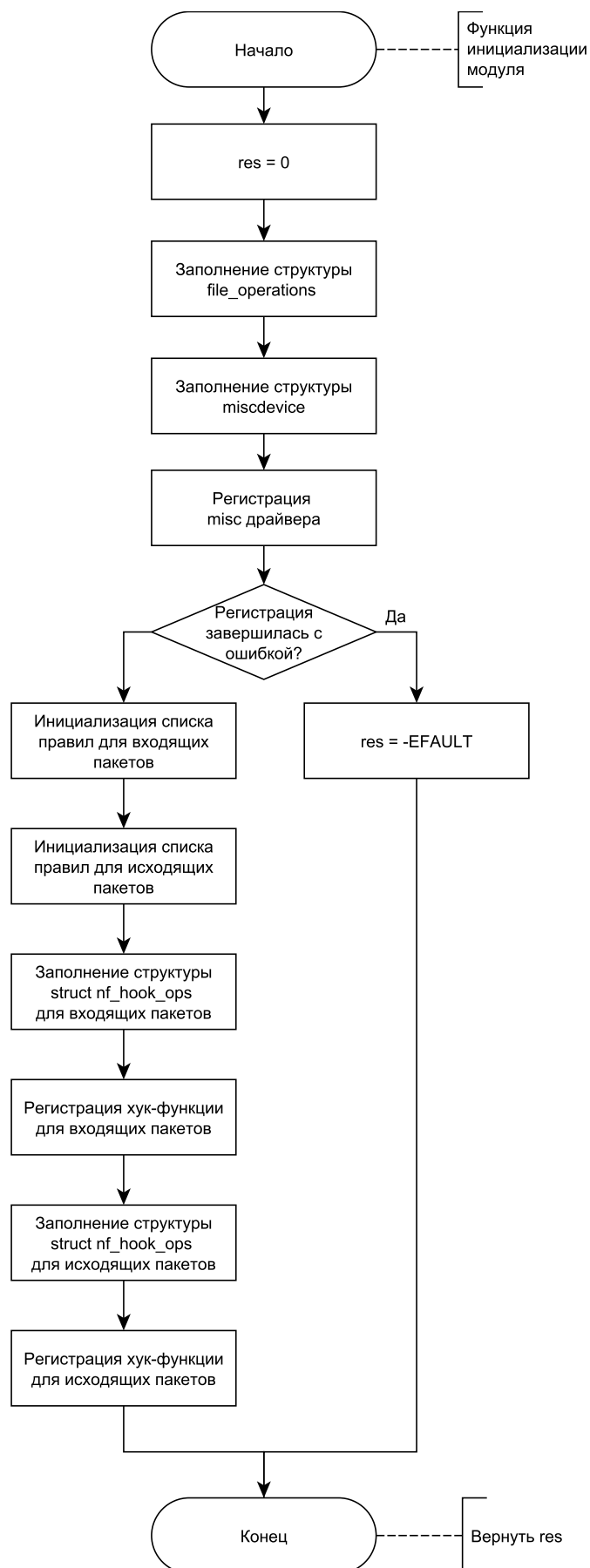


Рисунок 2.1 – Схема инициализации модуля

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.

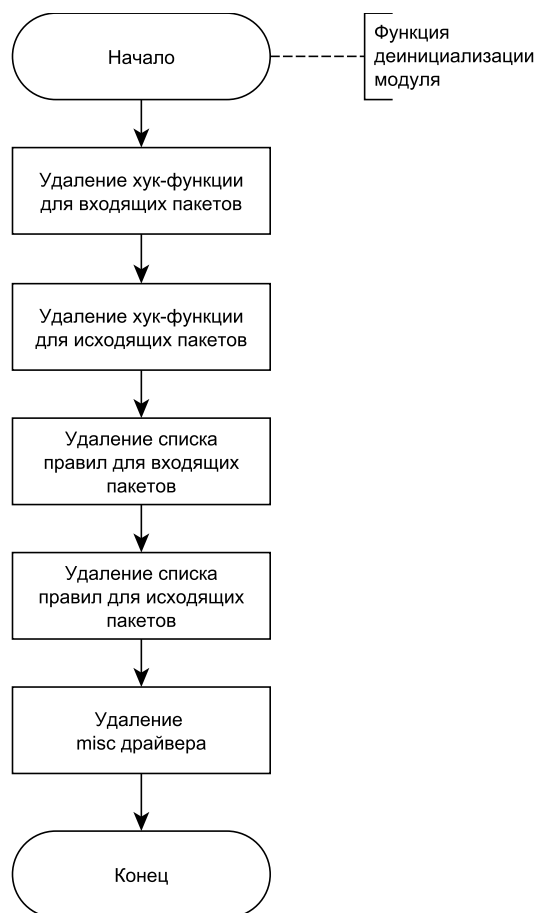


Рисунок 2.2 – Схема выгрузки модуля

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.

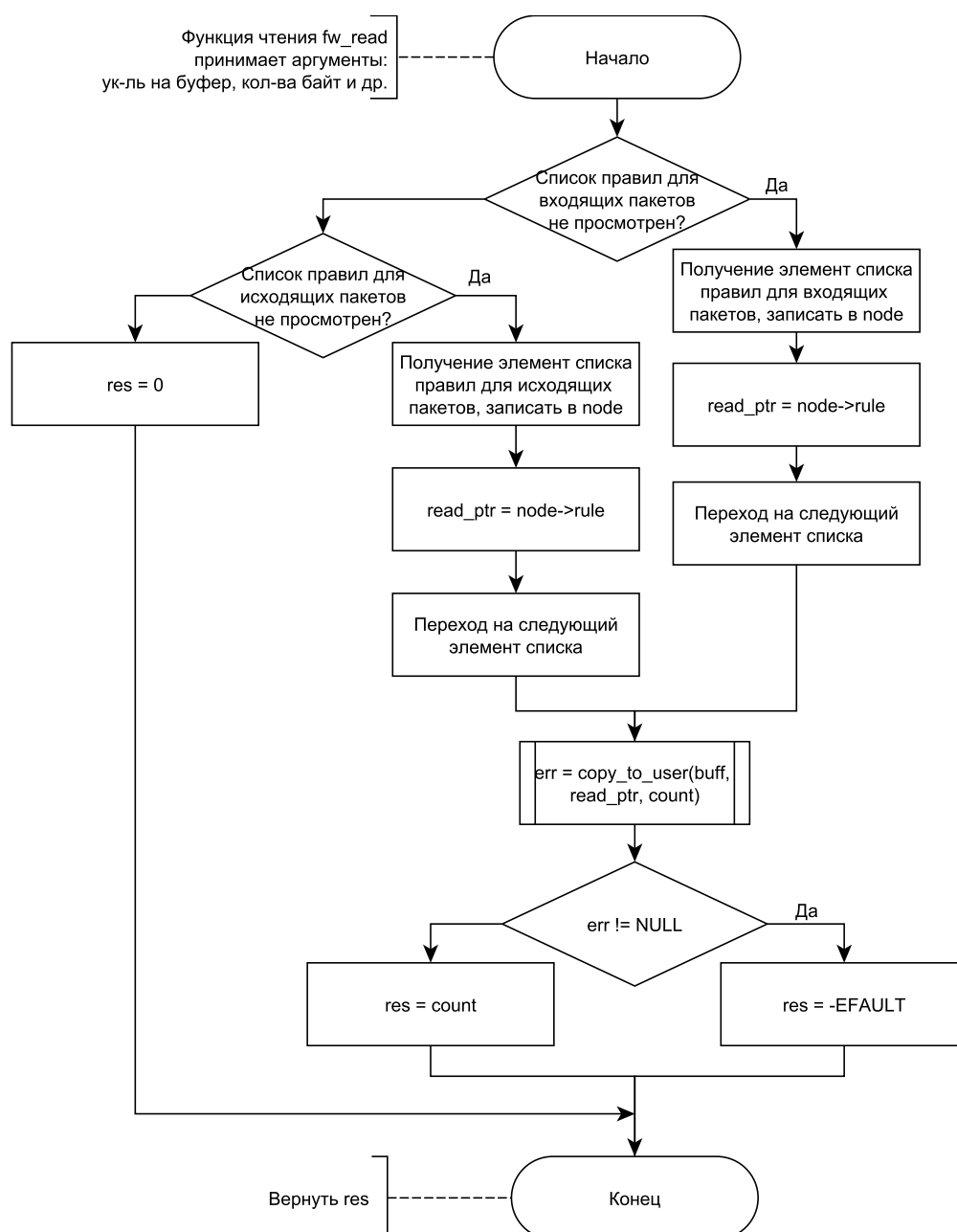


Рисунок 2.3 – Схема работы функции чтения

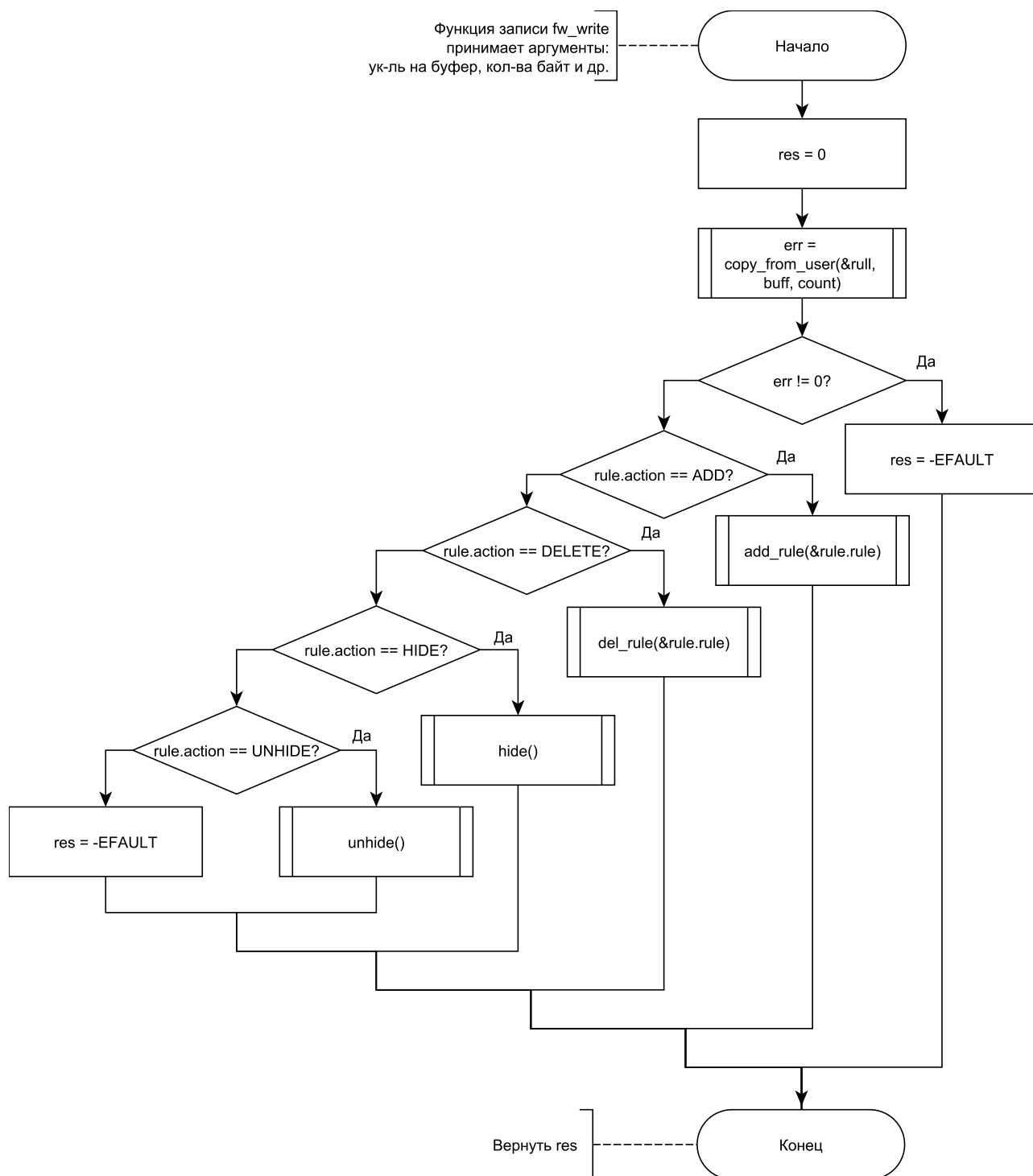


Рисунок 2.4 – Схема работы функции записи

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.

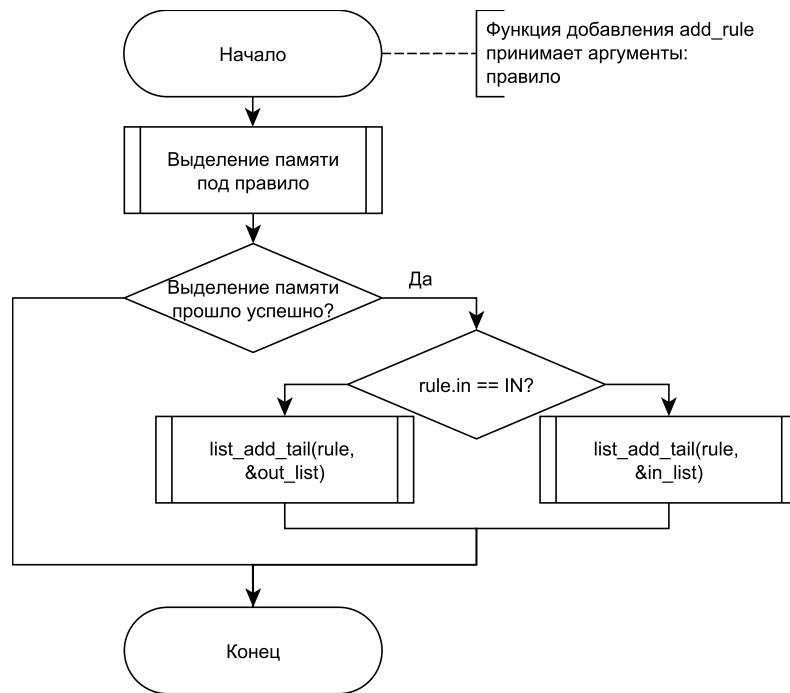


Рисунок 2.5 – Схема работы функции добавления нового правила

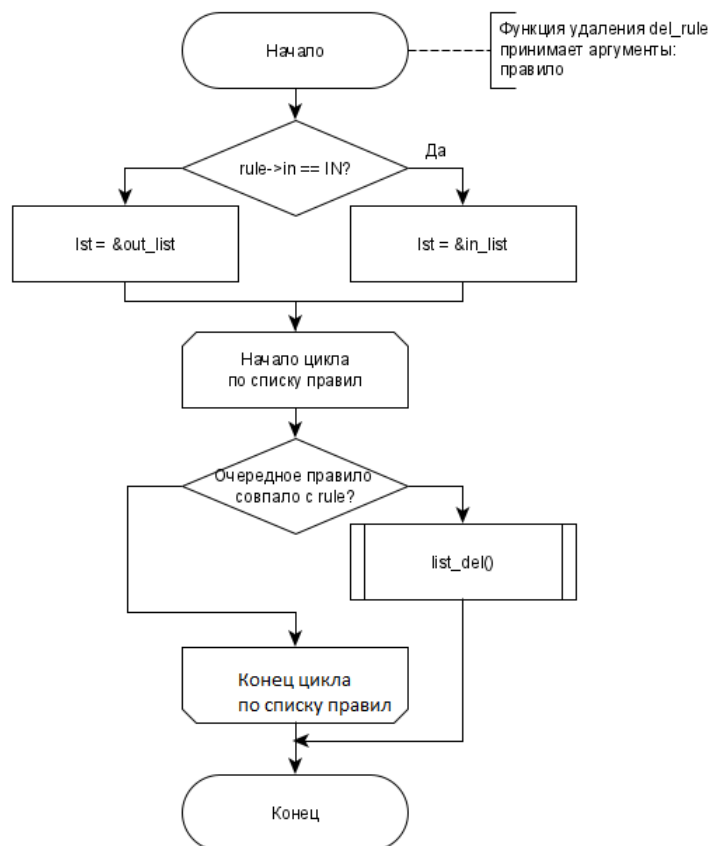


Рисунок 2.6 – Схема работы функции удаления правила

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.

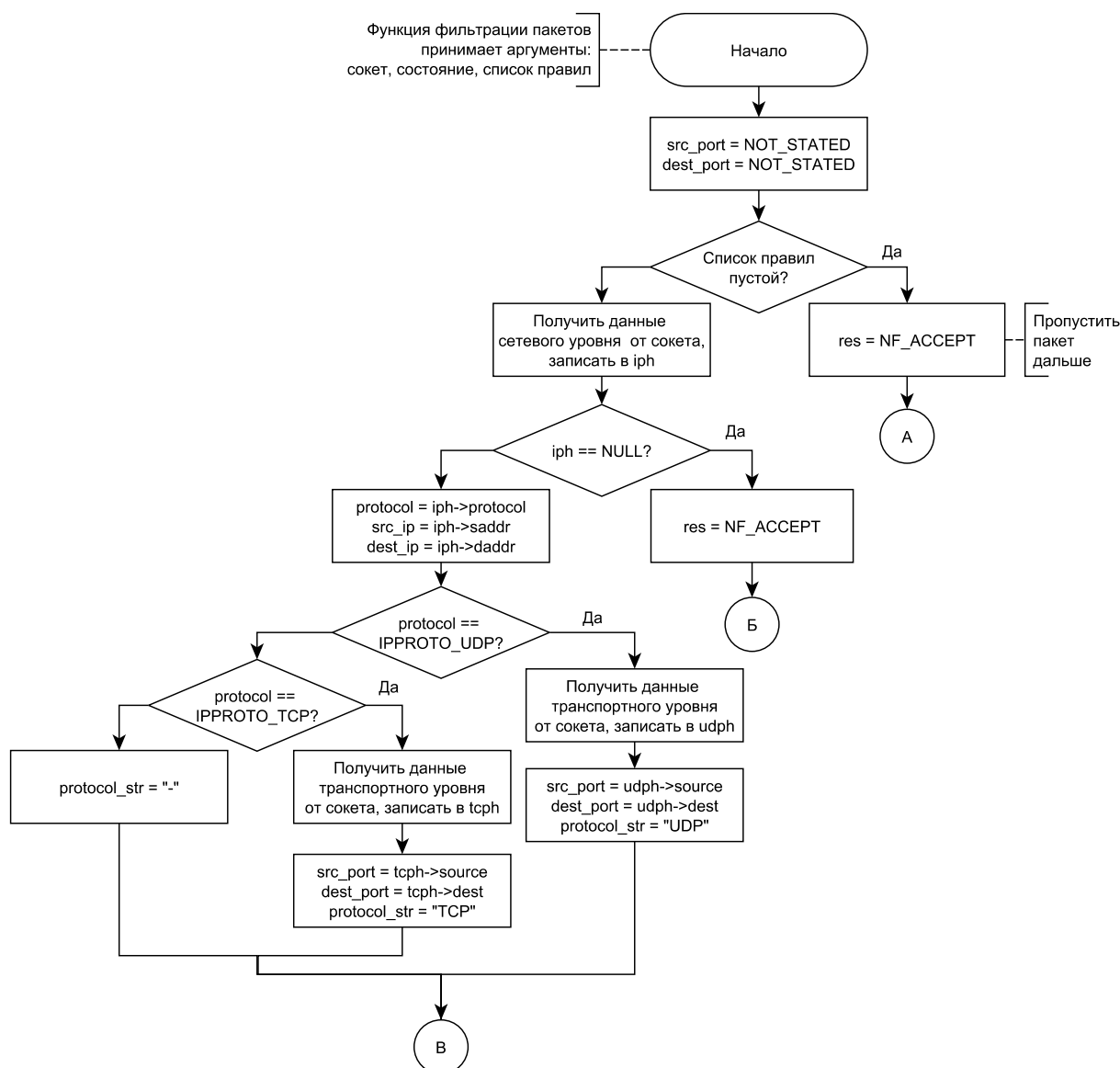


Рисунок 2.7 – Схема работы функции фильтрации пакетов

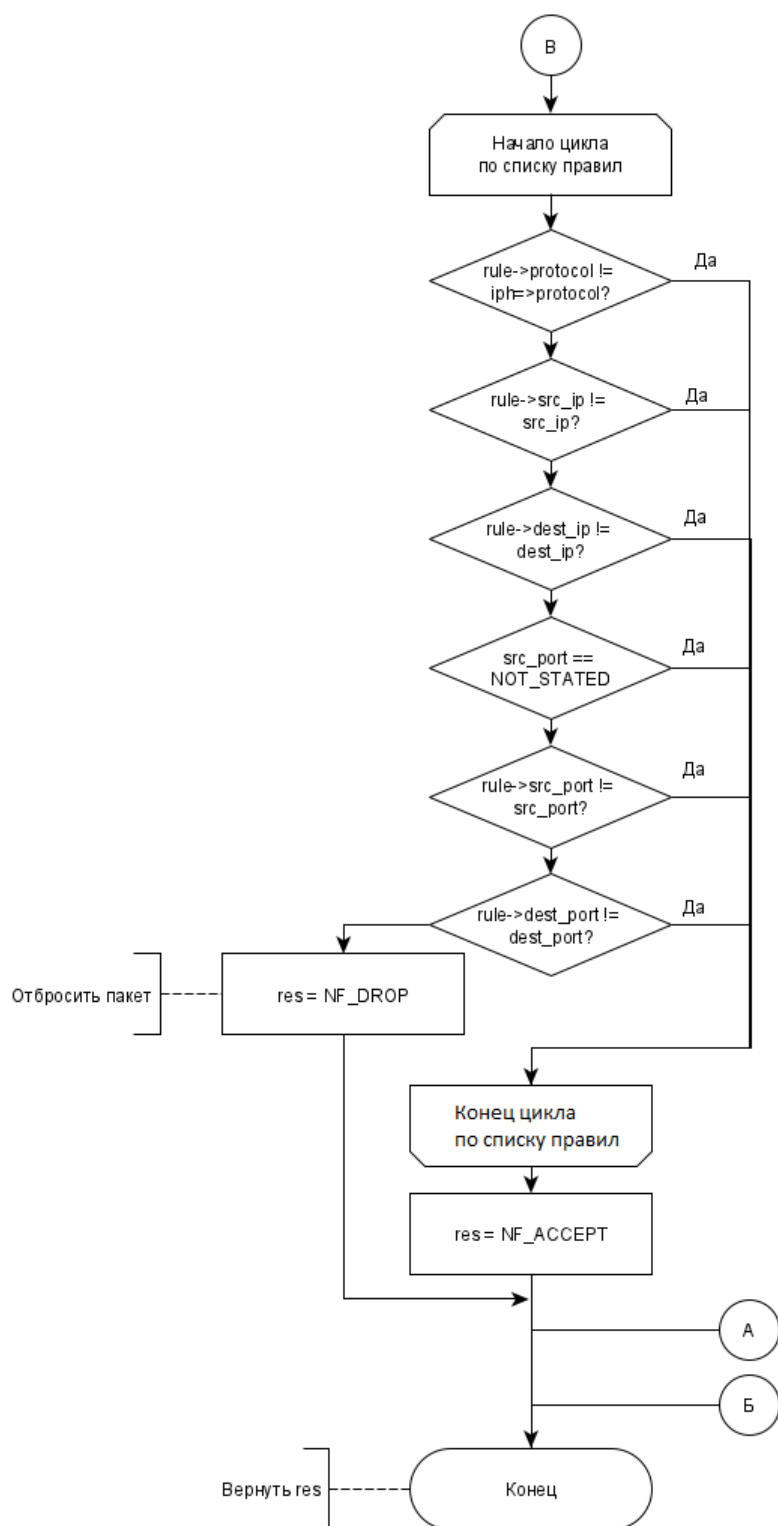


Рисунок 2.8 – Схема работы функции фильтрации пакетов (продолжение)

2.7 Выводы

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

3 Технологическая часть

3.1 Выбор языка программирования

В качестве языка программирования был выбран C. [10] Для сборки модуля использовалась утилита make.

Была выбрана среда разработки Visual Studio Code [11], так как она бесплатная, кроссплатформенная, а также позволяет использовать все возможности консоли, не переключаясь между окнами.

3.2 Структура загружаемого 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.

3.3 Структура проекта

Проект состоит из нескольких частей:

- **fw_module.c** – загружаемый модуль;
- **fw.h** – основные структуры;
- **fw.c** – непосредственное взаимодействие с пользователем, включающее обработку вводимых им данных;
- **errors.h** – коды обрабатываемых ошибок.

В Приложении А представлены листинги каждой из частей проекта.

3.4 Сборка и запуск модуля

Сборка модуля осуществляется командой make. На Листинге 8 приведено содержимое Makefile.

Листинг 8: Makefile

```

1 obj-m += fw_module.o
2
3 all: fw.o fw_module.o
4
5 fw.o: fw.c fw.h
6     gcc -o fw.o fw.c
7
8 fw_module.o: fw_module.c
9     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
10
11 clean:
```

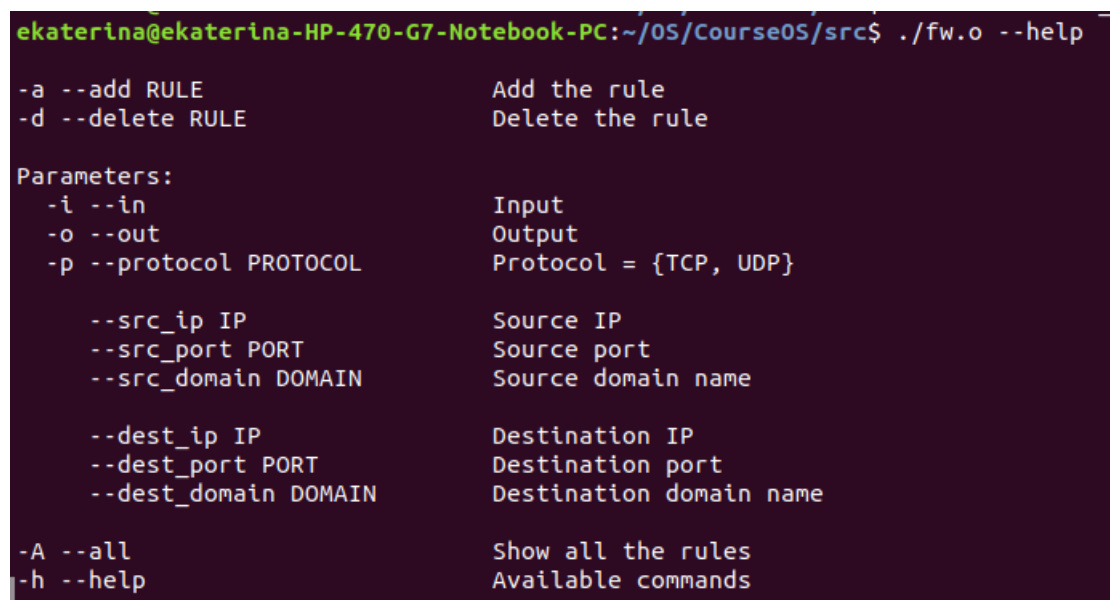
```
12     rm -rf fw *.o
13     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Для того, чтобы загрузить модуль, нужно воспользоваться командой **sudo insmod fw_module.ko**, для того, чтобы выгрузить – **sudo rmmod fw_module**.

3.5 Демонстрация работы модуля

3.5.1 Команды и формат задания правил

Для того, чтобы посмотреть все команды и формат задаваемых правил, необходимо вызвать **help**. На Рисунке 3.1 представлен результат.



```
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/CourseOS/src$ ./fw.o --help
-a --add RULE           Add the rule
-d --delete RULE        Delete the rule

Parameters:
-i --in                 Input
-o --out                 Output
-p --protocol PROTOCOL  Protocol = {TCP, UDP}

    --src_ip IP          Source IP
    --src_port PORT      Source port
    --src_domain DOMAIN  Source domain name

    --dest_ip IP          Destination IP
    --dest_port PORT      Destination port
    --dest_domain DOMAIN  Destination domain name

-A --all                 Show all the rules
-h --help                Available commands
```

Рисунок 3.1 – Вызов команды help

Для того, чтобы корректно задать правило фильтрации, необходимо соблюдать следующий формат.

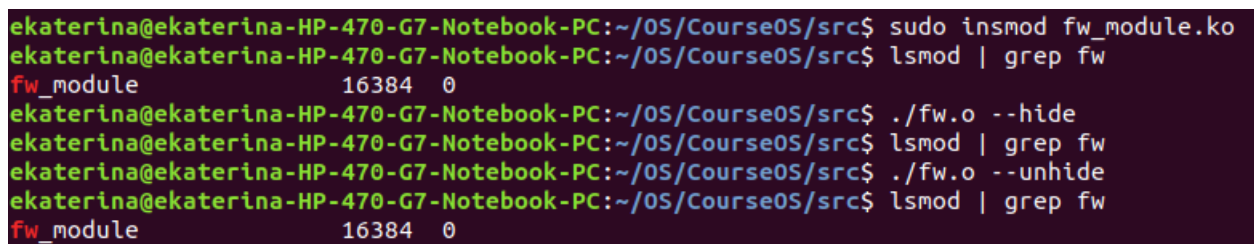
- 1) Обязательным является указание действий:
 - добавление (add);
 - удаление (del).
- 2) Также в обязательном порядке следует конкретизировать направление пакетов:

- входящие (in);
 - исходящие (out).
- 3) Далее указываются основные признаки фильтрации (один или несколько):
- протокол (TCP/UDP);
 - IP-адрес (источника/назначения);
 - порт (источника/назначения);
 - доменное имя (источника/назначения).

3.5.2 Видимость модуля

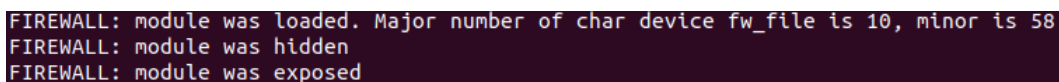
Для скрытия модуля следует вызвать команду **hide**, а для обратного действия **unhide**. На Рисунках 3.2 – 3.3 демонстрируется следующее:

- 1) модуль загружен и виден в системе;
- 2) вызвана команда **hide**;
- 3) модуль не отображается при вызове команды **lsmod**;
- 4) вызвана команда **unhide**;
- 5) модуль обнаруживается при вызове команды **lsmod**.



```
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/Course05/src$ sudo insmod fw_module.ko
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/Course05/src$ lsmod | grep fw
fw_module                16384  0
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/Course05/src$ ./fw.o --hide
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/Course05/src$ lsmod | grep fw
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/Course05/src$ ./fw.o --unhide
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~/OS/Course05/src$ lsmod | grep fw
fw_module                16384  0
```

Рисунок 3.2 – Порядок действий



```
FIREWALL: module was loaded. Major number of char device fw_file is 10, minor is 58
FIREWALL: module was hidden
FIREWALL: module was exposed
```

Рисунок 3.3 – Логи

3.5.3 Фильтрация по протоколу

Для наглядности было добавлено правило, блокирующее все входящие пакеты, для передачи которых используется протокол TCP (Рисунок 3.4).

FIREWALL: new rule was added. Rule: IN protocol: TCP

Рисунок 3.4 – Результат добавления правила

До того, как правило было добавлено, наблюдалась следующая динамика (Рисунок 3.5): происходил активный обмен пакетами, ошибки в процессе были, но не значительные.

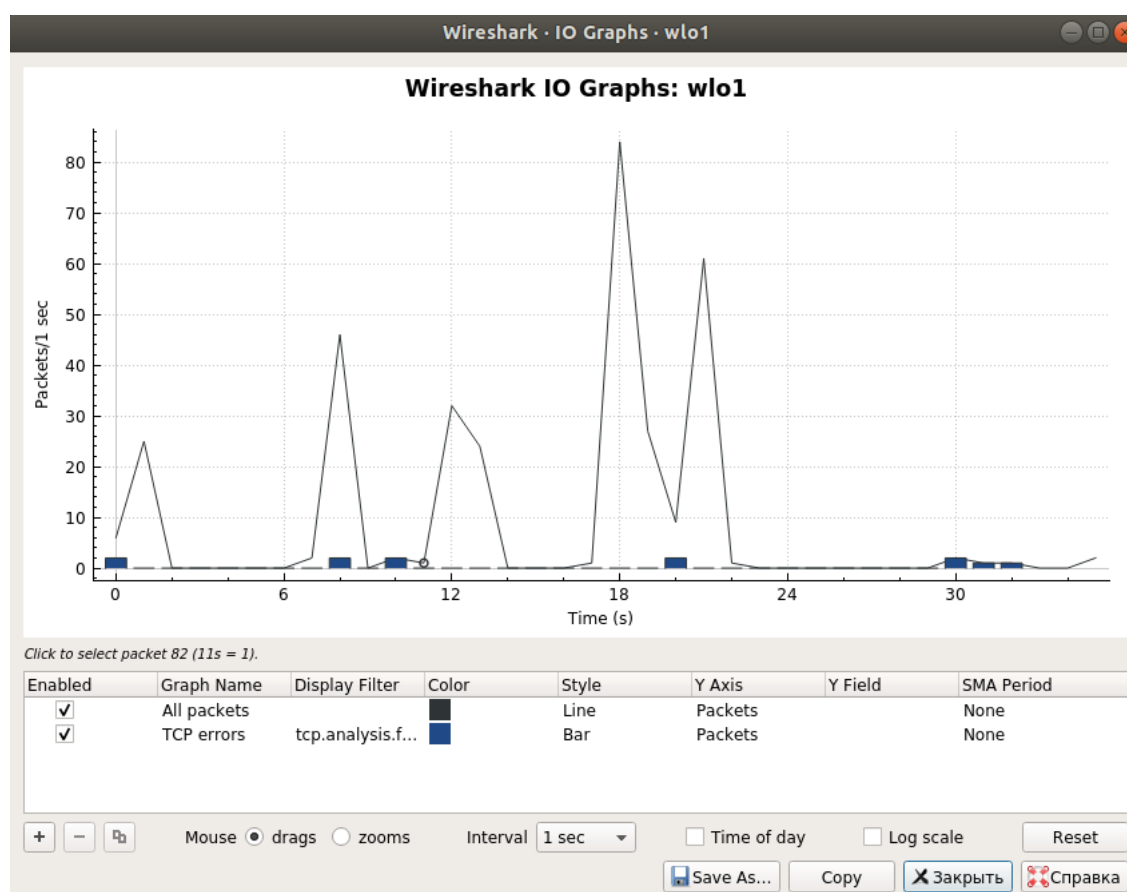


Рисунок 3.5 – До добавления правила фильтрации

Когда же правило было зарегистрировано, ситуация изменилась (Рисунок 3.6). Примерно с 50 секунды наблюдается увеличение непринятых пакетов, более детальную информацию о них можно получить из log-файла (Рисунок 3.7).

Легко заметить, что общее у всех непринятых пакетов – поле протокола.

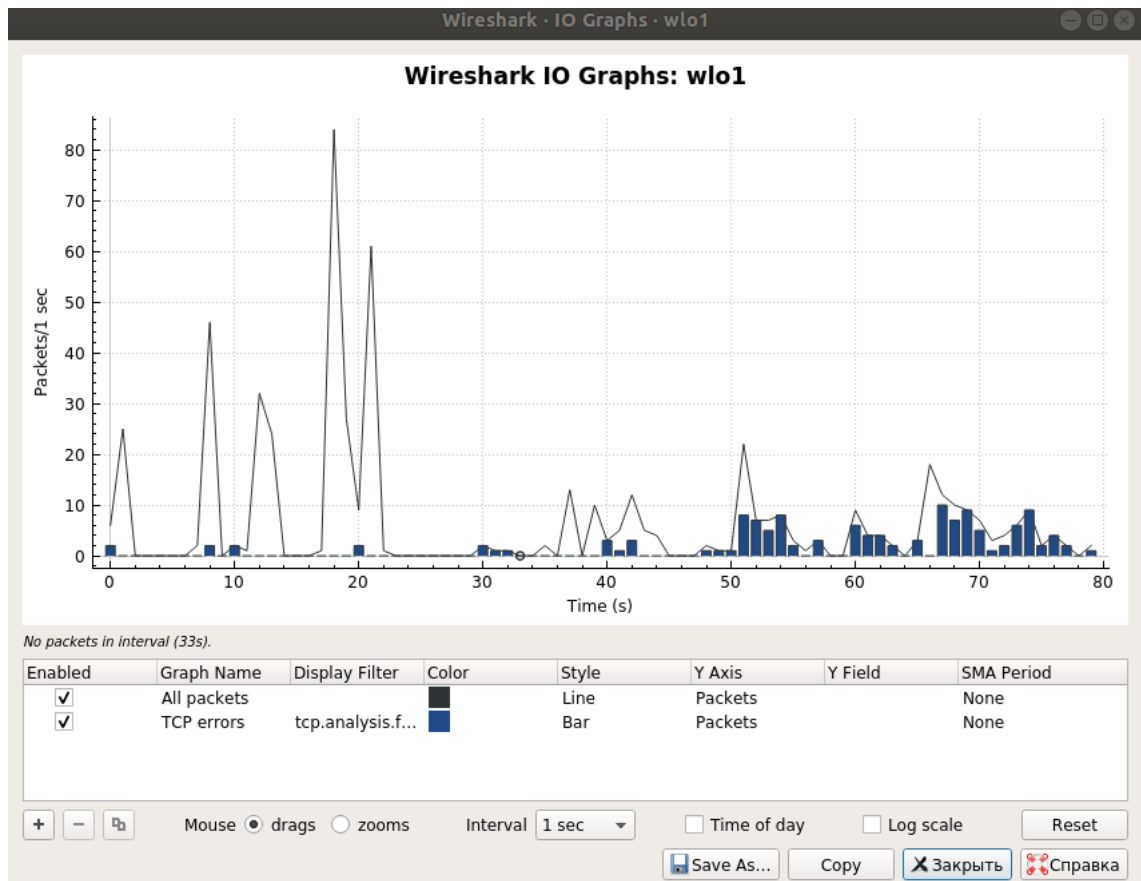


Рисунок 3.6 – После добавления правила фильтрации

```
Details: src_ip: 140.82.112.26      src_port: 443  dest_ip: 192.168.67.4  dest_port: 49048  protocol: TCP
Details: src_ip: 140.82.121.3     src_port: 443  dest_ip: 192.168.67.4  dest_port: 49548  protocol: TCP
Details: src_ip: 140.82.121.3     src_port: 443  dest_ip: 192.168.67.4  dest_port: 49548  protocol: TCP
Details: src_ip: 93.184.220.29    src_port: 80   dest_ip: 192.168.67.4  dest_port: 34002  protocol: TCP
```

Рисунок 3.7 – Подробная информация о заблокированных пакетах

При удалении правила, пакеты больше не блокируются, и это видно на Рисунке 3.8.

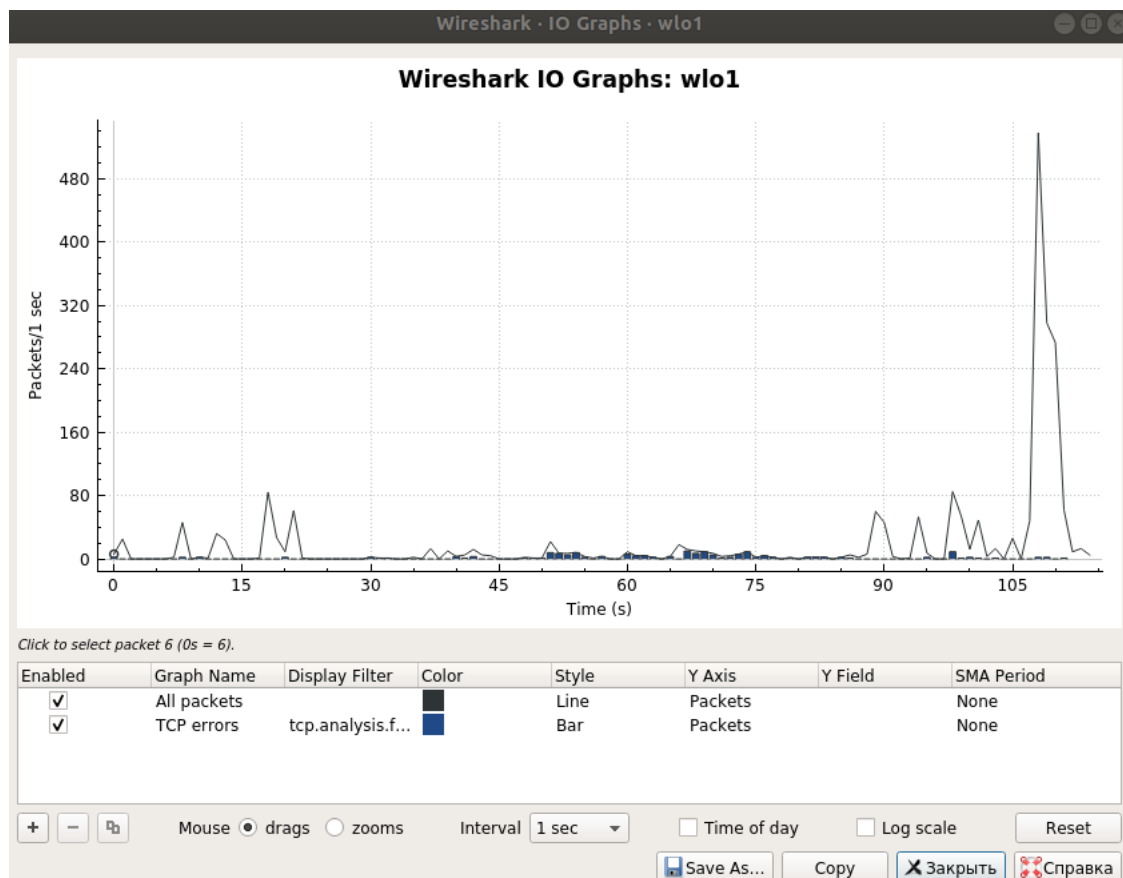


Рисунок 3.8 – После удаления правила фильтрации

С 80 секунды наблюдается резкое уменьшение ошибок и увеличение успешно обработанных пакетов.

3.5.4 Фильтрация по IP-адресу

Модуль также предоставляет возможность отбора пакетов по IP-адресу получателя (в случае исходящих пакетов) и отправителя (входящие).

Было добавлено следующее правило (Рисунок 3.9) и через некоторое время удалено (Рисунок 3.10).

```
./fw.o --add --out --dest_ip 192.168.67.132
```

Рисунок 3.9 – Добавление правила фильтрации по IP-адресу

```
./fw.o --del --out --dest_ip 192.168.67.132
```

Рисунок 3.10 – Удаление правила фильтрации по IP-адресу

Устройство, имеющее IP-адрес 192.168.67.132, и устройство, на котором работает межсетевой экран находятся в одной подсети. Поэтому можно с помощью команды **ping** проверить соединение.

Рисунок 3.11 – результат. Первые 5 и последние 6 пакетов успешно были отправлены в силу того, что правило ещё не было задано или уже удалено на момент их отправки.

```
ekaterina@ekaterina-HP-470-G7-Notebook-PC:~$ ping 192.168.67.132
PING 192.168.67.132 (192.168.67.132) 56(84) bytes of data.
64 bytes from 192.168.67.132: icmp_seq=1 ttl=64 time=6.14 ms
64 bytes from 192.168.67.132: icmp_seq=2 ttl=64 time=4.47 ms
64 bytes from 192.168.67.132: icmp_seq=3 ttl=64 time=4.13 ms
64 bytes from 192.168.67.132: icmp_seq=4 ttl=64 time=4.15 ms
64 bytes from 192.168.67.132: icmp_seq=5 ttl=64 time=4.19 ms
ping: sendmsg: Операция не позволена
ping: sendmsg: Операция не позволена
ping: sendmsg: Операция не позволена
ping: sendmsg: Операция не позволена
ping: sendmsg: Операция не позволена
ping: sendmsg: Операция не позволена
64 bytes from 192.168.67.132: icmp_seq=12 ttl=64 time=6.33 ms
64 bytes from 192.168.67.132: icmp_seq=13 ttl=64 time=6.52 ms
64 bytes from 192.168.67.132: icmp_seq=14 ttl=64 time=4.48 ms
64 bytes from 192.168.67.132: icmp_seq=15 ttl=64 time=3.92 ms
64 bytes from 192.168.67.132: icmp_seq=16 ttl=64 time=4.52 ms
64 bytes from 192.168.67.132: icmp_seq=17 ttl=64 time=5.10 ms
^C
--- 192.168.67.132 ping statistics ---
17 packets transmitted, 11 received, 35% packet loss, time 16149ms
rtt min/avg/max/mdev = 3.923/4.908/6.524/0.928 ms
```

Рисунок 3.11 – Результат

3.6 Вывод

В этом разделе был выбран язык программирования и среда разработки, описана общая структура модуля и продемонстрированы некоторые его возможности.

ЗАКЛЮЧЕНИЕ

В ходе работы были выполнены поставленная цель и необходимые для этого задачи:

- были изучены основные принципы работы сети и межсетевых экранов;
- а также способы перехвата пакетов сети (основные параметры задаются в структуре `struct nf_hook_ops`) и описания правил;
- определена целесообразность с `misc` драйвера:
 - не нужно указывать старший номер устройства – он задан заранее;
 - младший номер следует выделять динамически;
 - все необходимые структуры создаются автоматически при регистрации, в отличие от `char` драйвера, при работе с которым нужно вызывать функции `cdev_init`, `cdev_add`, `class_create`;
- были изучены основные принципы работы с подобным драйвером;
- разработано соответствующее программное обеспечение.

Таким образом, был реализован загружаемый модуль ядра, выполняющий роль межсетевого экрана, осуществляющего контроль проходящего через него сетевого трафика в соответствии с правилами, которые задаёт пользователь.

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. lsmmod [Электронный ресурс]. – Режим доступа: <https://www.opennet.ru/man.shtml?topic=lsmmod&category=8&russian=2>
5. Простая маскировка модуля ядра Linux с применением DKOM [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/205274/>
6. Alessandro Rubini, Jonathan Corbet Linux Device Drivers. – 2nd Edition изд. O'Reilly Media, 2001. – 562 с.
7. 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).
8. Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman Linux Device Drivers. – Third Edition изд. – Gravenstein Highway North, Sebastopo: O'Reilly Media, 2005. – 630 с.
9. NETFILTER [Электронный ресурс]. – Режим доступа: <http://samag.ru/archive/article/169> (дата обращения 07.10.2021).
10. Документация по языку C [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/cpp/c-language/?view=msvc-170> (дата обращения 01.10.2021).

11. Документация по Visual Studio [Электронный ресурс]. – Режим доступа: <https://code.visualstudio.com/docs> (дата обращения 01.10.2021).

ПРИЛОЖЕНИЕ А

Листинг 9: fw_module.c

```
1 #include "fw.h"
2
3 #define IP_POS(ip, pos) (ip >> ((8 * (3 - pos))) & 0xFF)
4 #define SAME_ADDR(ip1, ip2) ((ip1 ^ ip2) == 0)
5
6 MODULE_LICENSE("GPL");
7 MODULE_AUTHOR("Bryanskaya Ekaterina <bryanskayakatyayandex.ru>");
8
9 static char *buffer;
10 static int flag_hidden = 0;
11
12 struct list_head in_list;
13 struct list_head out_list;
14
15 struct list_head *module_prev;
16
17 struct rule_item {
18     struct fw_rule rule;
19     struct list_head list;
20 };
21
22 void hide(void)
23 {
24     if (flag_hidden)
25         return;
26
27     module_prev = THIS_MODULE->list.prev;
28     list_del(&THIS_MODULE->list);
29     flag_hidden = 1;
30
31     printk(">>> FIREWALL: module was hidden");
32 }
33
34 void unhide(void)
35 {
36     if (!flag_hidden)
```

```

37         return;
38
39     list_add(&THIS_MODULE->list, module_prev);
40     flag_hidden = 0;
41
42     printk(">>> FIREWALL: module was exposed");
43 }
44
45 int fw_open(struct inode *inode, struct file *file)
46 {
47     printk(KERN_INFO ">>> FIREWALL: associated char device was opened");
48     return 0;
49 }
50
51 int fw_release(struct inode *inode, struct file *file)
52 {
53     printk(KERN_INFO ">>> FIREWALL: associated char device was closed");
54     return 0;
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 ", IP_POS(
rule->src_ip, 3), IP_POS(rule->src_ip, 2), IP_POS(rule->src_ip, 1), IP_POS
(rule->src_ip, 0));
77
78     if (rule->src_port != NOT_STATED)
79         count_bytes += snprintf(res + count_bytes, 20, "src_port: %u \t ",
ntohs(rule->src_port));
80
81     if (rule->dest_ip != NOT_STATED)
82         count_bytes += snprintf(res + count_bytes, 30, "dest_ip: %u.%u.%u.%u
\t ", IP_POS(rule->dest_ip, 3), IP_POS(rule->dest_ip, 2), IP_POS(rule->
dest_ip, 1), IP_POS(rule->dest_ip, 0));
83
84     if (rule->dest_port != NOT_STATED)
85         count_bytes += snprintf(res + count_bytes, 20, "dest_port: %u \t ",
ntohs(rule->dest_port));
86
87     if (rule->protocol != NOT_STATED)
88     {
89         if (rule->protocol == IPPROTO_TCP)
90             snprintf(res + count_bytes, 20, "protocol: TCP");
91         else if (rule->protocol == IPPROTO_UDP)
92             snprintf(res + count_bytes, 20, "protocol: UDP");
93     }
94
95     return res;
96 }
97
98 char* str_packet(uint32_t src_ip, uint16_t src_port, uint32_t dest_ip,
uint16_t dest_port, char *protocol_str)
99 {
100     int count_bytes = 0;
101
102     char *res = kmalloc(BUF_LEN, GFP_KERNEL);
103     if (!res)
104     {
105         printk(KERN_INFO "FIREWALL: error in formating rule");
106         return NULL;
107     }
108

```

```

109     if (src_ip != NOT_STATED)
110         count_bytes = snprintf(res, 30, "src_ip: %u.%u.%u.%u \t ", IP_POS(
src_ip, 3), IP_POS(src_ip, 2), IP_POS(src_ip, 1), IP_POS(src_ip, 0));
111
112     if (src_port != NOT_STATED)
113         count_bytes += snprintf(res + count_bytes, 20, "src_port: %u \t ",
ntohs(src_port));
114     else
115         count_bytes += snprintf(res + count_bytes, 20, "src_port: - \t ");
116
117     if (dest_ip != NOT_STATED)
118         count_bytes += snprintf(res + count_bytes, 30, "dest_ip: %u.%u.%u.%u
\t ", IP_POS(dest_ip, 3), IP_POS(dest_ip, 2), IP_POS(dest_ip, 1), IP_POS(
dest_ip, 0));
119
120     if (dest_port != NOT_STATED)
121         count_bytes += snprintf(res + count_bytes, 20, "dest_port: %u \t ",
ntohs(dest_port));
122     else
123         count_bytes += snprintf(res + count_bytes, 20, "dest_port: - \t ");
124
125     snprintf(res + count_bytes, 20, "protocol: %s", protocol_str);
126
127     return res;
128 }
129
130 static void add_rule(struct fw_rule *rule)
131 {
132     struct rule_item *node;
133
134     node = (struct rule_item *)kmalloc(sizeof(struct rule_item), GFP_KERNEL);
135     if (node == NULL)
136     {
137         printk(KERN_INFO ">>> FIREWALL: addition a new rule was failed");
138         return;
139     }
140
141     node->rule = *rule;
142
143     if (node->rule.in == IN)

```

```

144     list_add_tail(&node->list, &in_list);
145 else
146     list_add_tail(&node->list, &out_list);
147
148     printk(KERN_INFO ">>> FIREWALL: new rule was added. Rule: %s", str_rule
149             (&(node->rule)));
150 }
151
152 static void del_rule(struct fw_rule *rule)
153 {
154     struct list_head *lst, *temp;
155     struct rule_item *node;
156
157     if (rule->in == IN)
158         lst = &in_list;
159     else
160         lst = &out_list;
161
162     for (temp = lst; temp->next != lst; temp = temp->next)
163     {
164         node = list_entry(temp->next, struct rule_item, list);
165
166         if (node->rule.in == rule->in && node->rule.src_ip == rule->src_ip &&
167             node->rule.src_port == rule->src_port && node->rule.dest_ip ==
168             rule->dest_ip &&
169             node->rule.dest_port == rule->dest_port && node->rule.protocol ==
170             rule->protocol)
171         {
172             list_del(temp->next);
173             kfree(node);
174
175             printk(KERN_INFO ">>> FIREWALL: rule was removed. Rule: %s",
176                     str_rule(rule));
177             return;
178         }
179     }
180
181     printk(KERN_INFO ">>> FIREWALL: rule was not found. Rule: %s", str_rule(
182             rule));
183 }

```



```

179
180 ssize_t fw_read(struct file *filp, char __user *buff, size_t count, loff_t *
    f_pos)
181 {
182     static struct list_head *in_lst = &in_list;
183     static struct list_head *out_lst = &out_list;
184     struct rule_item *node;
185     char *read_ptr;
186
187     if (in_lst->next != &in_list)
188     {
189         node = list_entry(in_lst->next, struct rule_item, list);
190         read_ptr = (char *)&node->rule;
191         in_lst = in_lst->next;
192     }
193     else if (out_lst->next != &out_list)
194     {
195         node = list_entry(out_lst->next, struct rule_item, list);
196         read_ptr = (char *)&node->rule;
197         out_lst = out_lst->next;
198     }
199     else
200     {
201         in_lst = &in_list;
202         out_lst = &out_list;
203
204         return 0;
205     }
206
207     if (copy_to_user(buff, read_ptr, count))
208     {
209         printk(KERN_INFO ">>> FIREWALL: copy_to_user error");
210         return -EFAULT;
211     }
212
213     return count;
214 }
215
216 ssize_t fw_write(struct file *filp, const char __user *buff, size_t count,
    loff_t *f_pos)

```

```

217 {
218     struct fw_comm rule_full;
219
220     if (count < sizeof(struct fw_comm))
221     {
222         printk(KERN_INFO ">>> FIREWALL: incorrect rule");
223         return -EFAULT;
224     }
225
226     if (copy_from_user(&rule_full, buff, count))
227     {
228         printk(KERN_INFO ">>> FIREWALL: copy_from_user error");
229         return -EFAULT;
230     }
231
232     switch (rule_full.action)
233     {
234         case ADD:
235             add_rule(&rule_full.rule);
236             break;
237         case DELETE:
238             del_rule(&rule_full.rule);
239             break;
240         case HIDE:
241             hide();
242             break;
243         case UNHIDE:
244             unhide();
245             break;
246         default:
247             printk(KERN_INFO ">>> FIREWALL: unknown command");
248             break;
249     }
250
251     return 0;
252 }
253
254 static unsigned int filter(void *priv, struct sk_buff *skb, const struct
    nf_hook_state *state,
255 struct list_head *list_rule)

```

```

256 {
257     struct iphdr *iph; /* An IPv4 packet header */
258     struct tcphdr *tcph;
259     struct udphdr *udph;
260
261     unsigned char protocol;
262     char *protocol_str;
263     uint32_t src_ip, dest_ip;
264     uint16_t src_port = NOT_STATED, dest_port = NOT_STATED;
265
266     struct list_head *lst;
267     struct rule_item *node;
268     struct fw_rule *rule;
269
270     if (!skb || list_rule->next == list_rule)
271         return NF_ACCEPT;
272
273     iph = (struct iphdr *)skb_network_header(skb);
274     if (iph == NULL)
275         return NF_ACCEPT;
276
277     protocol = iph->protocol;
278     src_ip = iph->saddr;
279     dest_ip = iph->daddr;
280
281     if (protocol == IPPROTO_UDP)
282     {
283         udph = (struct udphdr *) (skb_transport_header(skb));
284         src_port = udph->source;
285         dest_port = udph->dest;
286         protocol_str = "UDP";
287     }
288     else if (protocol == IPPROTO_TCP)
289     {
290         tcph = (struct tcphdr *) (skb_transport_header(skb));
291         src_port = tcph->source;
292         dest_port = tcph->dest;
293         protocol_str = "TCP";
294     }
295     else

```

```

296     protocol_str = "-";
297
298     lst = list_rule;
299     list_for_each_entry(node, lst, list)
300     {
301         rule = &node->rule;
302
303         if (rule->protocol != NOT_STATED && rule->protocol != iph->protocol)
304             continue;
305         if (rule->src_ip != NOT_STATED && !SAME_ADDR(rule->src_ip, src_ip))
306             continue;
307         if (rule->dest_ip != NOT_STATED && !SAME_ADDR(rule->dest_ip, dest_ip)
308         )
309             continue;
310         if (src_port == NOT_STATED)
311             continue;
312         if (rule->src_port != NOT_STATED && rule->src_port != src_port)
313             continue;
314         if (rule->dest_port != NOT_STATED && rule->dest_port != dest_port)
315             continue;
316
317         printk(KERN_INFO ">>> FIREWALL: packet was dropped. Details: %s",
318             str_packet(src_ip, src_port, dest_ip, dest_port, protocol_str));
319
320         return NF_DROP; /* discarded the packet */
321     }
322
323     return NF_ACCEPT; /* the packet passes, continue iterations */
324 }
325
326 static unsigned int fw_in_filter(void *priv, struct sk_buff *skb, const
327     struct nf_hook_state *state)
328 {
329     return filter(priv, skb, state, &in_list);
330 }
331
332 static unsigned int fw_out_filter(void *priv, struct sk_buff *skb, const
333     struct nf_hook_state *state)
334 {
335     return filter(priv, skb, state, &out_list);

```

```

333 }
334
335 static struct file_operations fw_fops = {
336     .owner = THIS_MODULE,
337     .read = fw_read,
338     .write = fw_write,
339     .open = fw_open,
340     .release = fw_release,
341 };
342
343 struct miscdevice dev = {
344     .minor = MISC_DYNAMIC_MINOR,
345     .name = DEVICE_FNAME,
346     .fops = &fw_fops,
347     .mode = S_IRWXU | S_IWGRP | S_IWOTH | S_IROTH,
348 };
349
350 static struct nf_hook_ops fw_in_hook_ops =
351 {
352     .hook = fw_in_filter,
353     .pf = PF_INET,
354     .hooknum = NF_INET_PRE_ROUTING,
355     .priority = NF_IP_PRI_FIRST
356 };
357
358 static struct nf_hook_ops fw_out_hook_ops =
359 {
360     .hook = fw_out_filter,
361     .pf = PF_INET,
362     .hooknum = NF_INET_LOCAL_OUT,
363     .priority = NF_IP_PRI_FIRST
364 };
365
366 static int __init fw_init(void)
367 {
368     int res = 0;
369
370     res = misc_register(&dev);
371     if (res)
372     {

```

```

373     printk(KERN_INFO ">>> FIREWALL: registration was failed");
374     return res;
375 }
376
377 buffer = (char *)kmallocc(sizeof(struct fw_comm*), GFP_KERNEL);
378 if (buffer == NULL)
379 {
380     printk(KERN_INFO ">>> FIREWALL: kmalloc error");
381     return -EFAULT;
382 }
383
384 INIT_LIST_HEAD(&in_list);
385 INIT_LIST_HEAD(&out_list);
386
387 nf_register_net_hook(&init_net, &fw_in_hook_ops);
388 nf_register_net_hook(&init_net, &fw_out_hook_ops);
389
390 printk(KERN_INFO ">>> FIREWALL: module was loaded. Major number of char
device %s is 10, minor is %d", DEVICE_FNAME, dev.minor);
391
392 return res;
393 }
394
395 static void __exit fw_exit(void)
396 {
397     struct rule_item *node;
398     struct rule_item *node_temp;
399
400     kfree(buffer);
401
402     list_for_each_entry_safe(node, node_temp, &in_list, list)
403     {
404         printk(KERN_INFO ">>> FIREWALL: rule was removed. Rule: %s", str_rule
(&(node->rule)));
405         list_del(&node->list);
406         kfree(node);
407     }
408
409     list_for_each_entry_safe(node, node_temp, &out_list, list)
410     {

```

```
411     printk(KERN_INFO ">>> FIREWALL: rule was removed. Rule: %s", str_rule
    (&(node->rule)));
412     list_del(&node->list);
413     kfree(node);
414 }
415
416 misc_deregister(&dev);
417
418 nf_unregister_net_hook(&init_net, &fw_in_hook_ops);
419 nf_unregister_net_hook(&init_net, &fw_out_hook_ops);
420
421 printk(KERN_INFO ">>> FIREWALL: module was unloaded!\n");
422 }
423
424 module_init(fw_init);
425 module_exit(fw_exit);
```

Листинг 10: fw.h

```
1 #define IN          1
2 #define OUT         2
3 #define NOT_STATED  10 /* Because there is no such tcp/udp port + define for
   protocols */
4
5 #define DEVICE_FNAME      "fw_file"
6 #define DEVICE_CLASS     "fw_class"
7
8 struct fw_rule
9 {
10     u_int32_t in;
11
12     u_int32_t src_ip;
13     u_int16_t src_port;
14
15     u_int32_t dest_ip;
16     u_int32_t dest_port;
17
18     u_int8_t protocol;
19 };
20
21 enum fw_action
22 {
23     ADD = 1,
24     DELETE = 2,
25     SHOW = 3,
26     HIDE = 4,
27     UNHIDE = 5,
28     NONE = 0
29 };
30
31 struct fw_comm
32 {
33     enum fw_action action;
34     struct fw_rule rule;
35 };
```


Листинг 11: fw.c

```

1 #include "errors.h"
2 #include "fw.h"
3
4 #define TCP_PROTOCOL    "TCP"
5 #define UDP_PROTOCOL    "UDP"
6
7 #define SRC              1
8 #define DEST             2
9
10 char ip_arr[64][INET_ADDRSTRLEN];
11 int domain_flag = NOT_STATED;
12
13 void show_info()
14 {
15     printf("\n"
16         "-a --add RULE \t\t\t Add the rule\n"
17         "-d --delete RULE \t\t\t Delete the rule\n"
18         "\n"
19         "Parameters:\n"
20         "  -i --in \t\t\t Input\n"
21         "  -o --out \t\t\t Output\n"
22         "  -p --protocol PROTOCOL \t Protocol = {TCP, UDP}\n"
23         "\n"
24         "    --src_ip IP \t\t\t Source IP\n"
25         "    --src_port PORT \t\t\t Source port\n"
26         "    --src_domain DOMAIN \t Source domain name\n"
27         "\n"
28         "    --dest_ip IP \t\t\t Destination IP\n"
29         "    --dest_port PORT \t\t\t Destination port\n"
30         "    --dest_domain DOMAIN \t Destination domain name\n"
31         "\n"
32         "-A --all \t\t\t Show all the rules\n"
33         "-h --help \t\t\t Available commands\n");
34 }
35
36 void print_head()
37 {
38     printf("IN/OUT \t source address \t source port \t destination address \t\n"
39         "destination port \t protocol\n");

```

```

39
40     for (int i = 0; i < 110; i++)
41         printf("-");
42     printf("\n");
43 }
44
45 int show_rules()
46 {
47     int fd;
48     char *buf;
49     struct fw_rule *rule;
50     struct in_addr addr;
51
52     fd = open("/dev/fw_file", O_RDONLY);
53     if (fd < 0)
54         return DEVICE_NOT_AVAILABLE;
55
56     buf = (char *)malloc(sizeof(struct fw_rule));
57     if (buf == NULL)
58         return MEMORY_ERROR;
59
60     print_head();
61
62     while (read(fd, buf, sizeof(struct fw_comm)) > 0)
63     {
64         rule = (struct fw_rule *)buf;
65
66         printf("%-8s ", rule->in == IN ? "IN" : "OUT");
67
68         if (rule->src_ip != NOT_STATED)
69         {
70             addr.s_addr = rule->src_ip;
71             printf("%-23s ", inet_ntoa(addr));
72         }
73         else
74             printf("%-23s ", "---");
75
76         if (rule->src_port != NOT_STATED)
77             printf("%-15d ", ntohs(rule->src_port));
78         else

```

```

79         printf("%-15s ", "---");
80
81     if (rule->dest_ip != NOT_STATED)
82     {
83         addr.s_addr = rule->dest_ip;
84         printf("%-23s ", inet_ntoa(addr));
85     }
86     else
87         printf("%-23s ", "---");
88
89     if (rule->dest_port != NOT_STATED)
90         printf("%-23d ", ntohs(rule->dest_port));
91     else
92         printf("%-23s ", "---");
93
94     if (rule->protocol != NOT_STATED)
95     {
96         if (rule->protocol == IPPROTO_TCP)
97             printf("%-5s ", "TCP");
98         else if (rule->protocol == IPPROTO_UDP)
99             printf("%-5s ", "UDP");
100    }
101    else
102        printf("%-8s ", "---");
103    printf("\n");
104 }
105
106 free(buf);
107 close(fd);
108
109 return EXIT_SUCCESS;
110 }
111
112 int write_rule(struct fw_comm *comm)
113 {
114     int fd;
115     int count_byte;
116
117     fd = open("/dev/fw_file", O_WRONLY | O_APPEND);
118     if (fd < 0)

```

```

119         return DEVICE_NOT_AVAILABLE;
120
121     write(fd, comm, sizeof(*comm));
122
123     close(fd);
124
125     return EXIT_SUCCESS;
126 }
127
128 void init_comm(struct fw_comm *comm)
129 {
130     comm->action = NONE;
131     comm->rule.in = NOT_STATED;
132     comm->rule.src_ip = NOT_STATED;
133     comm->rule.src_port = NOT_STATED;
134     comm->rule.dest_ip = NOT_STATED;
135     comm->rule.dest_port = NOT_STATED;
136     comm->rule.protocol = NOT_STATED;
137 }
138
139 uint64_t parse_add_arg(const char *str, int min_value, int max_value)
140 {
141     int num;
142     char *end;
143
144     num = strtol(str, &end, 10);
145     if (num < min_value || num > max_value || str == end)
146         return EXIT_FAILURE;
147     return num;
148 }
149
150 int parse_add_prot(const char *protocol)
151 {
152     if (strcmp(protocol, TCP_PROTOCOL) == 0)
153         return IPPROTO_TCP;
154     if (strcmp(protocol, UDP_PROTOCOL) == 0)
155         return IPPROTO_UDP;
156     return EXIT_FAILURE;
157 }
158

```

```

159 int get_ip_from_domain(const char *str)
160 {
161     struct hostent* host = NULL;
162     char tmpIp[INET_ADDRSTRLEN];
163
164     host = gethostbyname(str);
165     if (host == NULL)
166         return INCORRECT_DOMAIN;
167
168     for (int i = 0; host->h_addr_list[i] != NULL; i++)
169     {
170         memset(tmpIp, 0, sizeof(tmpIp));
171         inet_ntop(host->h_addrtype, host->h_addr_list[i], tmpIp,
INET_ADDRSTRLEN);
172
173         if (strlen(tmpIp) > 0)
174             strcpy(ip_arr[i], tmpIp);
175     }
176     return EXIT_SUCCESS;
177 }
178
179 int parse_comm(int argc, char **argv, struct fw_comm *res_comm)
180 {
181     int res, comm_ind, protocol;
182     int64_t param;
183     const char* short_comm = "ad:Aiop:s:r:m:t:e:M:h10";
184     struct in_addr addr;
185     struct fw_comm comm;
186
187     if (argc == 1)
188     {
189         show_info();
190         return LACK_ARGUMENTS;
191     }
192
193     struct option long_comm[] =
194     {
195         {"add", no_argument, 0, 'a'},
196         {"delete", no_argument, 0, 'd'},
197         {"all", no_argument, 0, 'A'},

```

```

198     {"in", no_argument, 0, 'i'},
199     {"out", no_argument, 0, 'o'},
200     {"protocol", required_argument, 0, 'p'},
201     {"src_ip", required_argument, 0, 's'},
202     {"src_port", required_argument, 0, 'r'},
203     {"src_domain", required_argument, 0, 'm'},
204     {"dest_ip", required_argument, 0, 't'},
205     {"dest_port", required_argument, 0, 'e'},
206     {"dest_domain", required_argument, 0, 'M'},
207     {"help", no_argument, 0, 'h'},
208     {"hide", no_argument, 0, '1'},
209     {"unhide", no_argument, 0, '0'},
210     {NULL, 0, NULL, 0}
211 };
212
213 init_comm(&comm);
214
215 while (1)
216 {
217     res = getopt_long(argc, argv, short_comm, long_comm, &comm_ind);
218     if (res < 0)
219         break;
220
221     switch (res)
222     {
223         case 'a':
224             if (comm.action != NONE)
225                 return ACTION_MENTIONED;
226             comm.action = ADD;
227             break;
228         case 'd':
229             if (comm.action != NONE)
230                 return ACTION_MENTIONED;
231             comm.action = DELETE;
232             break;
233         case 'A':
234             if (comm.action != NONE)
235                 return ACTION_MENTIONED;
236             comm.action = SHOW;
237             break;

```

```

238     case 'i':
239         if (comm.rule.in == OUT)
240             return DIRECTION_MENTIONED;
241         comm.rule.in = IN;
242         break;
243     case 'o':
244         if (comm.rule.in == IN)
245             return DIRECTION_MENTIONED;
246         comm.rule.in = OUT;
247         break;
248     case 'p':
249         if (comm.rule.protocol != NOT_STATED)
250             return PROTOCOL_MENTIONED;
251         protocol = parse_add_prot(optarg);
252         if (protocol == EXIT_FAILURE)
253             return WRONG_PROTOCOL;
254         comm.rule.protocol = protocol;
255         break;
256     case 's':
257         if (comm.rule.src_ip != NOT_STATED)
258             return SRC_IP_MENTIONED;
259         if (!inet_aton(optarg, &addr))
260             return INCORRECT_SRC_IP;
261         comm.rule.src_ip = addr.s_addr;
262         break;
263     case 't':
264         if (comm.rule.dest_ip != NOT_STATED)
265             return DEST_IP_MENTIONED;
266         if (!inet_aton(optarg, &addr))
267             return INCORRECT_DEST_IP;
268         comm.rule.dest_ip = addr.s_addr;
269         break;
270     case 'r':
271         if (comm.rule.src_port != NOT_STATED)
272             return SRC_PORT_MENTIONED;
273         param = parse_add_arg(optarg, 0, USHRT_MAX);
274         if (param == EXIT_FAILURE)
275             return INCORRECT_SRC_PORT;
276         comm.rule.src_port = htons((uint16_t)param);
277         break;

```

```

278     case 'm':
279         if (comm.rule.src_ip != NOT_STATED)
280             return SRC_IP_MENTIONED;
281         if (domain_flag != NOT_STATED)
282             return DOMAIN_MENTIONED;
283         param = get_ip_from_domain(optarg);
284         if (param == INCORRECT_DOMAIN)
285             return INCORRECT_DOMAIN;
286         if (!inet_aton(ip_arr[0], &addr))
287             return INCORRECT_DEST_IP;
288         comm.rule.src_ip = addr.s_addr;
289         domain_flag = SRC;
290         break;
291     case 'M':
292         if (comm.rule.dest_ip != NOT_STATED)
293             return DEST_IP_MENTIONED;
294         if (domain_flag != NOT_STATED)
295             return DOMAIN_MENTIONED;
296         param = get_ip_from_domain(optarg);
297         if (param == INCORRECT_DOMAIN)
298             return INCORRECT_DOMAIN;
299         if (!inet_aton(ip_arr[0], &addr))
300             return INCORRECT_DEST_IP;
301         comm.rule.dest_ip = addr.s_addr;
302         domain_flag = DEST;
303         break;
304     case 'e':
305         if (comm.rule.dest_port != NOT_STATED)
306             return DEST_PORT_MENTIONED;
307         param = parse_add_arg(optarg, 0, USHRT_MAX);
308         if (param == EXIT_FAILURE)
309             return INCORRECT_DEST_PORT;
310         comm.rule.dest_port = htons((uint16_t)param);
311         break;
312     case '1':
313         if (comm.action != NONE)
314             return ACTION_MENTIONED;
315         comm.action = HIDE;
316         break;
317     case '0':

```



```

318         if (comm.action != NONE)
319             return ACTION_MENTIONED;
320         comm.action = UNHIDE;
321         break;
322     default:
323         show_info();
324         return EXIT_FAILURE;
325     }
326 }
327
328 if (comm.action == NONE)
329     return ACTION_NOT_MENTIONED;
330
331 if (comm.action == SHOW || comm.action == HIDE || comm.action == UNHIDE)
332 {
333     *res_comm = comm;
334     return EXIT_SUCCESS;
335 }
336
337 if (comm.rule.in == NOT_STATED)
338     return DIRECTION_NOT_MENTIONED;
339
340 if (comm.rule.src_ip == NOT_STATED && comm.rule.src_port == NOT_STATED &&
341     \
342     comm.rule.dest_ip == NOT_STATED && comm.rule.dest_port == NOT_STATED
343     && \
344     comm.rule.protocol == NOT_STATED)
345     return KEYS_NOT_MENTIONED;
346
347 *res_comm = comm;
348
349 return EXIT_SUCCESS;
350 }
351
352 int main(int argc, char *argv[])
353 {
354     struct fw_comm comm;
355     struct in_addr addr;
356     int res, ip_ind = 1;

```

```

356     res = parse_comm(argc, argv, &comm);
357
358     if (res)
359     {
360         switch (res)
361         {
362             case LACK_ARGUMENTS:
363                 printf("ERROR: not enough arguments.\n");
364                 break;
365             case ACTION_MENTIONED:
366                 printf("ERROR: action is already mentioned\n");
367                 break;
368             case DIRECTION_MENTIONED:
369                 printf("ERROR: direction is already mentioned\n");
370                 break;
371             case PROTOCOL_MENTIONED:
372                 printf("ERROR: protocol is already mentioned\n");
373                 break;
374             case WRONG_PROTOCOL:
375                 printf("ERROR: wrong parameter of protocol\n");
376                 break;
377             case SRC_IP_MENTIONED:
378                 printf("ERROR: source IP is already mentioned\n");
379                 break;
380             case INCORRECT_SRC_IP:
381                 printf("ERROR: incorrect source IP\n");
382                 break;
383             case DEST_IP_MENTIONED:
384                 printf("ERROR: destination IP is already mentioned\n");
385                 break;
386             case INCORRECT_DEST_IP:
387                 printf("ERROR: incorrect destination IP\n");
388                 break;
389             case SRC_PORT_MENTIONED:
390                 printf("ERROR: source port is already mentioned\n");
391                 break;
392             case INCORRECT_SRC_PORT:
393                 printf("ERROR: incorrect source port\n");
394                 break;
395             case DEST_PORT_MENTIONED:

```

```

396         printf("ERROR: destination port is already mentioned\n");
397         break;
398     case INCORRECT_DEST_PORT:
399         printf("ERROR: incorrect destination port\n");
400         break;
401     case ACTION_NOT_MENTIONED:
402         printf("ERROR: action (add/delete) is not mentioned\n");
403         break;
404     case DIRECTION_NOT_MENTIONED:
405         printf("ERROR: direction (in/out) is not mentioned\n");
406         break;
407     case INCORRECT_INDEX_RULE:
408         printf("ERROR: incorrect index of rule\n");
409         break;
410     case KEYS_NOT_MENTIONED:
411         printf("ERROR: keys are not mentioned\n");
412         break;
413     case INCORRECT_DOMAIN:
414         printf("ERROR: domain name is wrong\n");
415         break;
416     case DOMAIN_MENTIONED:
417         printf("ERROR: prohibited to mention more than one domain
name\n");
418         break;
419     default:
420         break;
421 }
422 return res;
423 }
424
425 do
426 {
427     switch (comm.action)
428     {
429         case ADD:
430         case DELETE:
431         case HIDE:
432         case UNHIDE:
433             res = write_rule(&comm);
434

```

```

435         switch (res)
436         {
437             case DEVICE_NOT_AVAILABLE:
438                 printf("ERROR: denied access to the device\n");
439                 break;
440             case RULE_ADDITION_FAILED:
441                 printf("ERROR: operation was failed.\n");
442                 break;
443             case EXIT_SUCCESS:
444                 break;
445             default:
446                 break;
447         }
448         break;
449     case SHOW:
450         res = show_rules();
451
452         switch (res)
453         {
454             case DEVICE_NOT_AVAILABLE:
455                 printf("ERROR: denied access to the device\n");
456                 break;
457             case MEMORY_ERROR:
458                 printf("ERROR: problems with memory allocation");
459                 break;
460             default:
461                 break;
462         }
463         break;
464     }
465
466     if (ip_ind < strlen(*ip_arr) && strcmp(ip_arr[ip_ind], ""))
467     {
468         if (!inet_aton(ip_arr[ip_ind], &addr))
469             return INCORRECT_SRC_IP;
470
471         if (domain_flag == SRC)
472             comm.rule.src_ip = addr.s_addr;
473         else if (domain_flag == DEST)
474             comm.rule.dest_ip = addr.s_addr;

```

```
475         ip_ind++;
476     }
477     else
478     {
479         for (int i = 0; i < strlen(*ip_arr) && strcmp(ip_arr[i], ""); i
480         ++)
481             strcpy(ip_arr[ip_ind], "");
482         break;
483     }
484 } while (1);
485
486 domain_flag = NOT_STATED;
487 return EXIT_SUCCESS;
488 }
```

Листинг 12: errors.c

```
1 #define LACK_ARGUMENTS 1000
2 #define DEVICE_NOT_AVAILABLE 1001
3 #define RULE_ADDITION_FAILED 1002
4 #define MEMORY_ERROR 1003
5
6 #define ACTION_MENTIONED 2000
7 #define DIRECTION_MENTIONED 2001
8 #define PROTOCOL_MENTIONED 2002
9 #define DOMAIN_MENTIONED 2003
10 #define WRONG_PROTOCOL 2004
11 #define SRC_IP_MENTIONED 2005
12 #define DEST_IP_MENTIONED 2006
13 #define SRC_PORT_MENTIONED 2007
14 #define DEST_PORT_MENTIONED 2008
15
16 #define ACTION_NOT_MENTIONED 2010
17 #define DIRECTION_NOT_MENTIONED 2011
18 #define KEYS_NOT_MENTIONED 2012
19
20 #define INCORRECT_SRC_IP 3000
21 #define INCORRECT_DEST_IP 3001
22 #define INCORRECT_SRC_PORT 3002
23 #define INCORRECT_DEST_PORT 3003
24 #define INCORRECT_INDEX_RULE 3004
25 #define INCORRECT_DOMAIN 3005
```