

Содержание

Реферат	3
Введение	4
Аналитическая часть	5
Формализация задачи	5
netfilter	5
Утилита iptables	8
Загружаемый модуль ядра	9
Вывод	10
Конструкторская часть	11
Требования к программе	11
Модуль ядра	11
Клиент-приложение	15
Схемы, демонстрирующие работу модуля	16
Вывод	19
Технологическая часть	20
Выбор языка программирования и среды разработки	20
Содержимое Makefile	20
Описание некоторых моментов реализации	20
Пример работы загружаемого модуля	24
Вывод	25
Заключение	26
Список литературы	27
Приложение А	28

Реферат

Ключевые слова: Linux, загружаемый модуль, TCP, UDP, транспортный протокол.

Курсовой проект представляет собой загружаемый модуль ядра для управления входящими TCP/UDP пакетами (принятие или отклонение) в зависимости от выбранной политики (черный или белый список). Используется язык программирования C.

Отчёт содержит 27 страниц, 17 листингов, 8 рисунков и 7 таблиц.

Введение

Транспортные протоколы предписывают способ передачи сообщений между узлами сети. Наиболее популярными из транспортных протоколов являются протокол управления передачей (TCP) и протокол пользовательских датаграмм (UDP) [1].

Входящие соединения могут использоваться злоумышленниками в корыстных целях, например, когда пользователь некорректно настроил брандмауэр в системе.

Целью данной работы является реализация программного обеспечения для управления входящими TCP/UDP пакетами в зависимости от выбранной политики.

Аналитическая часть

В данном разделе будет произведена формализация задачи, а также будут рассмотрены способы ее решения.

Формализация задачи

Целью данного курсового проекта является реализация программного обеспечения, которое бы позволило управлять входящими TCP/UDP пакетами в зависимости от выбранной нами политики.

Программное обеспечение должно отслеживать и корректно идентифицировать входящие соединения, чтобы впоследствии была возможность отклонить или принять входящий пакет.

Помимо этого, необходимо разработать клиент-приложение, которое позволяло бы менять политику в реальном времени.

Таким образом, для достижения поставленной цели необходимо решить следующие задачи:

- 1) проанализировать способы перехвата входящих TCP/UDP пакетов;
- 2) разработать алгоритмы, позволяющие управлять входящими пакетами;
- 3) спроектировать и реализовать модуль ядра.

netfilter

netfilter – межсетевой экран (брандмауэр), встроен в ядро Linux с версии 2.4.

Ключевыми понятиями netfilter являются:

- 1) правило – состоит из критерия, действия и счетчика;
- 2) цепочка – упорядоченная последовательность правил. Стандартные цепочки указаны в таблице 1;
- 3) таблица – совокупность базовых и пользовательских цепочек, объединенных общим функциональным назначением. Существующие таблицы указаны в таблице 2.

Стоит также отметить, что каждый пакет может иметь одно из четырёх возможных состояний, указанных в таблице 3.

Таблица 1: Типы стандартных цепочек, встроенных в систему

Название	Назначение
PREROUTING	Для изначальной обработки входящих пакетов
INPUT	Для входящих пакетов адресованных непосредственно локальному процессу (клиенту или серверу)
FORWARD	Для входящих пакетов, перенаправленных на выход
OUTPUT	Для пакетов, генерируемых локальными процессами
POSTROUTING	Для окончательной обработки исходящих пакетов

Таблица 2: Таблицы для организации цепочек

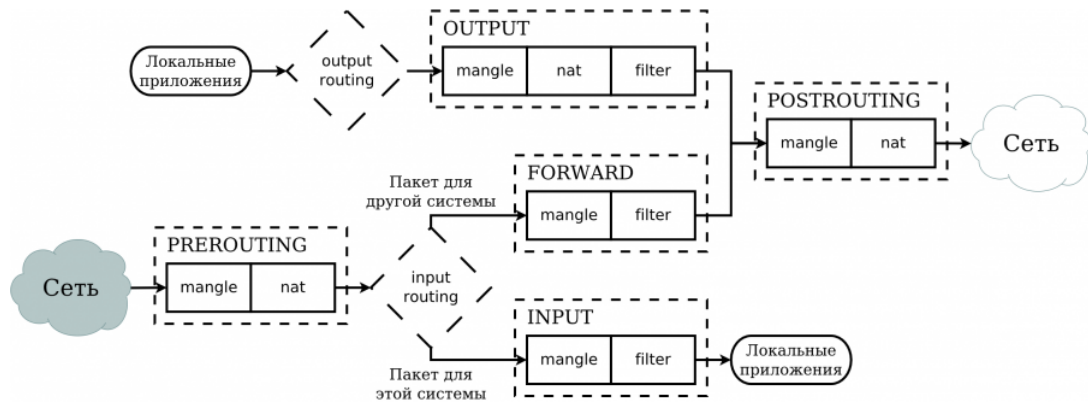
Название	Назначение
raw	Просматривается до передачи пакета системе определения состояний. Содержит цепочки PREROUTING и OUTPUT.
mangle	Содержит правила модификации (обычно заголовка) IP пакетов.
nat	Просматривает только пакеты, создающие новое соединение (согласно системе определения состояний)
filter	Основная таблица, используется по умолчанию если название таблицы не указано. Содержит цепочки INPUT, FORWARD, и OUTPUT.

Таблица 3: Возможные состояния пакета

Название	Значение
NEW	Пакет открывает новый сеанс.
ESTABLISHED	Пакет является частью уже существующего сеанса
RELATED	Пакет открывает новый сеанс, связанный с уже открытым сеансом.
INVALID	Все прочие пакеты.

В целом, диаграмма прохождения пакета таблиц и цепочек принимает следующий вид (рисунок 1): netfilter предоставляет набор хуков (netfilter hooks) в

Рис. 1: Диаграмма прохождения таблиц и цепочек



пространстве ядра для каждого протокола (для IPv4 - 5), которые позволяют управлять пакетами соединений. Виды хуков представлены в таблице 4.

Таблица 4: Разновидности netfilter hooks

Название	Назначение
NF_IP_PER_ROUTING	Вызывается, когда прибывает новый пакет
NF_IP_LOCAL_IN	Вызывается в том случае, если пакет предназначен для текущей машины
NF_IP_FORWARD	Вызывается в случае, если пакет предназначен для другого интерфейса
NF_IP_POST_ROUTING	Вызывается, когда пакет выходит за пределы машины
NF_IP_LOCAL_OUT	Вызывается, когда пакет создается локально и предназначается для отправки

Чтобы использовать хуки netfilter внутри ядра, необходимо вызывать функцию `nf_register_hook`, которая получает на вход `struct nf_hooks_ops`.

Описание структуры `nf_hooks_ops` выглядит так:

Листинг 1: `struct nf_hooks_ops`

```

1 struct nf_hook_ops {
2     /* User fills in from here down. */
3     nf_hookfn    *hook;
4     struct net_device *dev;
5     void         *priv;

```

```

6     u_int8_t      pf;
7     unsigned int   hooknum;
8     /* Hooks are ordered in ascending priority. */
9     int            priority;
10 };

```

Поля структуры `nf_hooks_ops` описаны в таблице 5.

Таблица 5: Структура `nf_hooks_ops`

Поле	Значение
<code>hook</code>	Указатель на функцию, которая вызывается при срабатывании хука. Эта функция относится к типу <code>nf_hookfn</code> и возвращает <code>NF_DROP</code> (отбросить пакет), <code>NF_ACCEPT</code> (позволить пакету пройти дальше) или <code>NF_QUEUE</code> (используется в том случае, если необходимо поставить пакет в очередь для обработки в пространстве пользователя)
<code>hooknum</code>	Один из идентификаторов хука (например, <code>NF_IP_POST_ROUTING</code>).
<code>pf</code>	Идентификатор семейства протоколов (например, <code>PF_INET</code> для IPv4).
<code>priority</code>	Приоритет хука (в случае, если в системе зарегистрированы другие хуки). Может принимать значение, определенное в <code>nf_ip_hook_priorities</code> , который определен в <code>netfilter_ipv4.h</code> (например, <code>NF_IP_PRI_FIRST</code> или <code>NF_IP_PRI_RAW</code>).

Таким образом, `netfilter hooks` могут быть использованы для решения поставленной задачи.

Утилита `iptables`

`iptables` — утилита командной строки, является стандартным интерфейсом управления работой межсетевого экрана (брандмауэра) `netfilter` для ядер Linux, начиная с версии 2.4 [2].

Утилиту `iptables` можно использовать для контроля сетевых соединений. Например, для отклонения всех входящих TCP соединений, можно использовать команду:

```
1 iptables -A INPUT -p tcp DROP
```

Однако из-за того, что для использования данной утилиты достаточно прав суперпользователя, данный способ не подходит для курсовой работы по операционным системам.

Загружаемый модуль ядра

Ядро Linux динамически изменяемое – это означает, что вы можете загружать в ядро дополнительную функциональность, выгружать функции из ядра и даже добавлять новые модули, использующие другие модули ядра. Преимущество загружаемых модулей заключается в возможности сократить расход памяти для ядра, загружая только необходимые модули (это может оказаться важным для встроенных систем).

Загружаемый модуль представляет собой специальный объектный файл в формате ELF (Executable and Linkable Format). Обычно объектные файлы обрабатываются компоновщиком, который разрешает символы и формирует исполняемый файл. Однако в связи с тем, что загружаемый модуль не может разрешить символы до загрузки в ядро, он остается ELF-объектом. Для работы с загружаемыми модулями можно использовать стандартные средства работы с объектными файлами (имеют суффикс `.ko`, от `kernel object`) [3].

В ОС Linux существуют специальные команды для работы с загружаемыми модулями ядра.

- 1) **insmod** загружает модуль в ядро из конкретного файла, если модуль зависит от других модулей. Только суперпользователь может загрузить модуль в ядро;
- 2) **lsmod** выводит список модулей, загруженных в ядро;
- 3) **modinfo** извлекает информацию из модулей ядра (лицензия, автор, описание и т.д.);
- 4) **rmmod** используется для выгрузки модуля из ядра, в качестве параметра

передается имя файла модуля. Только суперпользователь может выгрузить модуль из ядра;

- 5) **dmesg** - команда для вывода буфера сообщений ядра в стандартный поток вывода. Сообщения содержат информацию о драйверах устройств, загружаемых в ядро во время загрузки системы, а также при подключении аппаратного обеспечения к системе.

Помимо этого, загружаемые модули ядра должны содержать два макроса: `module_init` и `module_exit`.

Вывод

В данном разделе была произведена формализация задачи и рассмотрены способы ее решения. Выяснилось, что утилита `iptables` не подходит для решения данной задачи, поэтому мы будем использовать `netfilter hooks`.

Конструкторская часть

В данном разделе будут рассмотрены требования к программе, основные сведения о реализуемых модулях, а также представлена схемы работы алгоритма.

Требования к программе

Программное обеспечение должно состоять из загружаемого модуля ядра, который бы позволил управлять входящими TCP/UDP пакетами с помощью хуков netfilter и непосредственно клиента, позволяющего выбирать политику управления пакетами.

Модуль ядра

Для начала нам необходимо инициализировать символьный драйвер, который бы позволил получать информацию с символьного устройства.

Для регистрации устройства, нужно задать специальные номера, а именно:

- 1) MAJOR – старший номер (является уникальным в системе);
- 2) MINOR – младший номер (не является уникальным в системе).

Для задания старшего номера вызывается функция `register_chrdev`.

Листинг 2: func register_chrdev

```
1 static inline int register_chrdev(unsigned int major, const char *name
2 ,
3 const struct file_operations *fops)
4 {
5     return __register_chrdev(major, 0, 256, name, fops);
6 }
```

Также необходимо заполнить структуру `file_operations` как представлено в листинге 3. Для данной работы наиболее значимой функцией является `dev_write`, которая позволяет получать информацию из пространства пользователя.

Листинг 3: struct file_operations для текущей задачи

```
1 static struct file_operations fops =
2 {
3     .open = devel_open,
```

```

4  .write = dev_write ,
5  .release = dev_release ,
6  };

```

Листинг 4: функции struct file_operations

```

1 static int      devel_open(struct inode *, struct file *);
2 static int      dev_release(struct inode *, struct file *);
3 static ssize_t  dev_write(struct file *, const char *, size_t, loff_t
    *);

```

Для хранения информации о текущей политике выделяется двумерный массив символов, который также позволяет журналировать предыдущие изменения политики.

Листинг 5: Массив для хранения изменений политики

```

1 char list[100][6];
2 int listiterator = 0;

```

Как уже было упомянуто ранее, следующим шагом будет необходимо вызвать функцию `nf_register_hook` и передать в нее инициализированную структуру `nf_hooks_ops`.

Поскольку наша задача заключается в том, чтобы перехватить все входящие пакеты, поле `hooknum` должно принимать значение `NF_IP_LOCAL_IN` (в нашем случае `NF_INET_LOCAL_IN`, поскольку мы работаем только с протоколами TCP, UDP). Тогда структура `nf_hooks_ops` в нашем случае примет следующий вид:

Листинг 6: struct nf_hooks_ops для текущей задачи

```

1 static struct nf_hook_ops drop __read_mostly = {
2  .pf = NFPROTO_IPV4,
3  .priority = NF_IP_PRI_FIRST,
4  .hooknum = NF_INET_LOCAL_IN,
5  .hook = (nf_hookfn *) icmp_hook
6  };

```

Листинг вышеупомянутой функции представлен ниже.

Листинг 7: func nf_register_hook

```
1 int nf_register_net_hook(struct net *net, const struct nf_hook_ops *  
    reg)  
2 {  
3 int err;  
4  
5 if (reg->pf == NFPROTO_INET) {  
6 ...  
7 } else {  
8 err = __nf_register_net_hook(net, reg->pf, reg);  
9 if (err < 0)  
10 return err;  
11 }  
12  
13 return 0;  
14 }
```

Функция `icmpr_hook`, которая вызывается при срабатывании хука, будет представлена дальше (Технологическая часть, листинг 17).

Для корректного журналирования полезно будет определить IP адрес и порт приходящего пакета для того, чтобы принять или отклонить его в зависимости от установленной политики. Нам понадобятся функции `skb_transport_header` и `skb_network_header`, в которые мы передадим `struct sk_buff`.

Листинг 8: struct sk_buff

```
1 struct sk_buff {  
2     union {unnamed_union};  
3     ...  
4     sk_buff_data_t tail;  
5     sk_buff_data_t end;  
6     unsigned char * head;  
7     unsigned char * data;  
8     ...  
9 };
```

Некоторые поля структуры `sk_buff` описаны в таблице 6.

Таблица 6: Структура sk_buff

Поле	Значение
head	Указывает на начало выделенного пространства
data	Указывает на начало достоверных октетов
tail	Является окончанием достоверных октетов
end	Указывает на максимальный адрес, который может достичь tail

Ниже представлены листинги вышеупомянутых функций.

Листинг 9: func skb_transport_header

```

1 static inline unsigned char *skb_transport_header(const struct sk_buff
    *skb)
2 {
3     return skb->head + skb->transport_header;
4 }
```

Листинг 10: func skb_network_header

```

1 static inline unsigned char *skb_network_header(const struct sk_buff *
    skb)
2 {
3     return skb->head + skb->network_header;
4 }
```

Наконец, останется только принять или отклонить пакет с помощью необходимого флага.

В рамках данной курсовой работы достаточно использовать NF_ACCEPT и NF_DROP.

Таблица 7: Флаги netfilter

Флаг	Значение
NF_ACCEPT	Позволить пакету продолжить перемещение
NF_DROP	Отбросить пакет
NF_STOLEN	Перехватить перемещение
NF_QUEUE	Поместить пакет в очередь
NF_REPEAT	Вызвать хук снова

Клиент-приложение

Клиент-приложение должно обращаться непосредственно к созданному нами символьному устройству.

```
1 fd = open("/dev/pd", O_RDWR);  
2 if (fd < 0)  
3 {  
4     printf("failed to open the device...");  
5     return -1;  
6 }
```

Запись в файл производится с помощью команд lseek и write.

Схемы, демонстрирующие работу модуля

Ниже представлены схемы, демонстрирующие работу модуля.

Рис. 2: Схема инициализации работы модуля

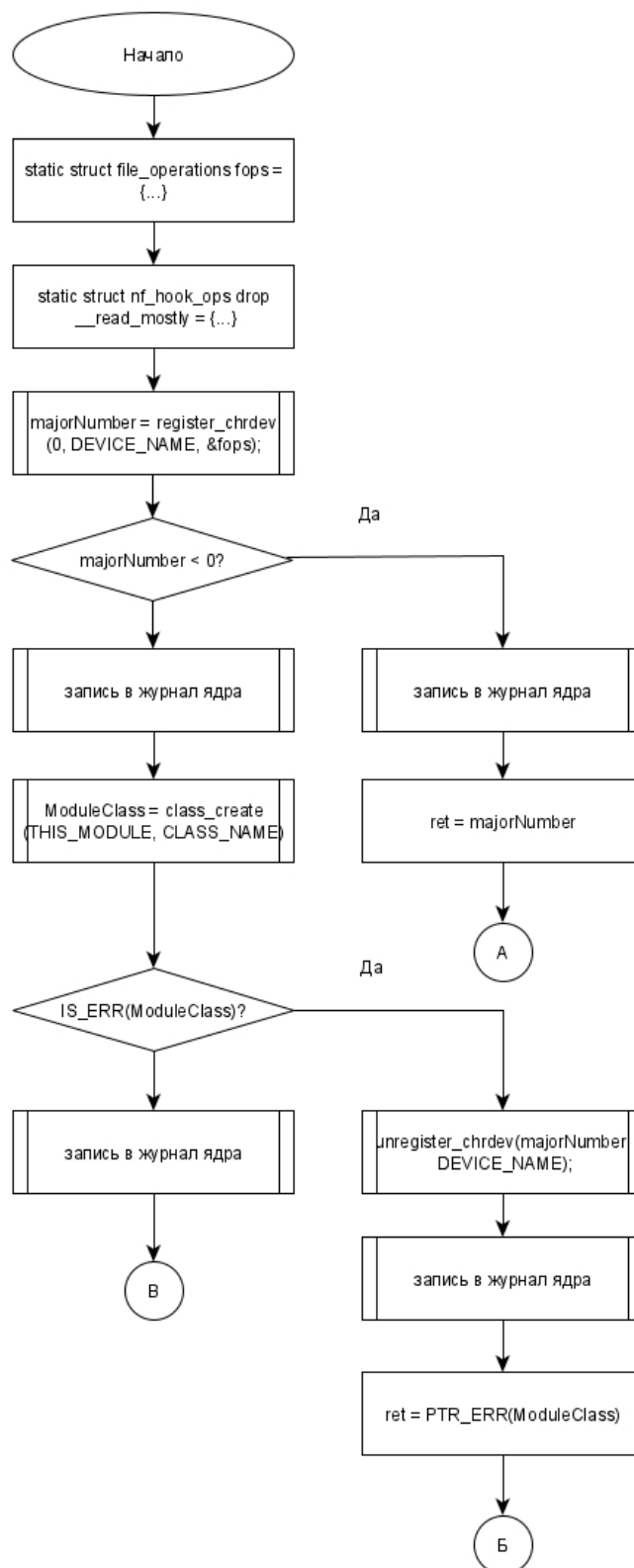


Рис. 3: Схема инициализации работы модуля

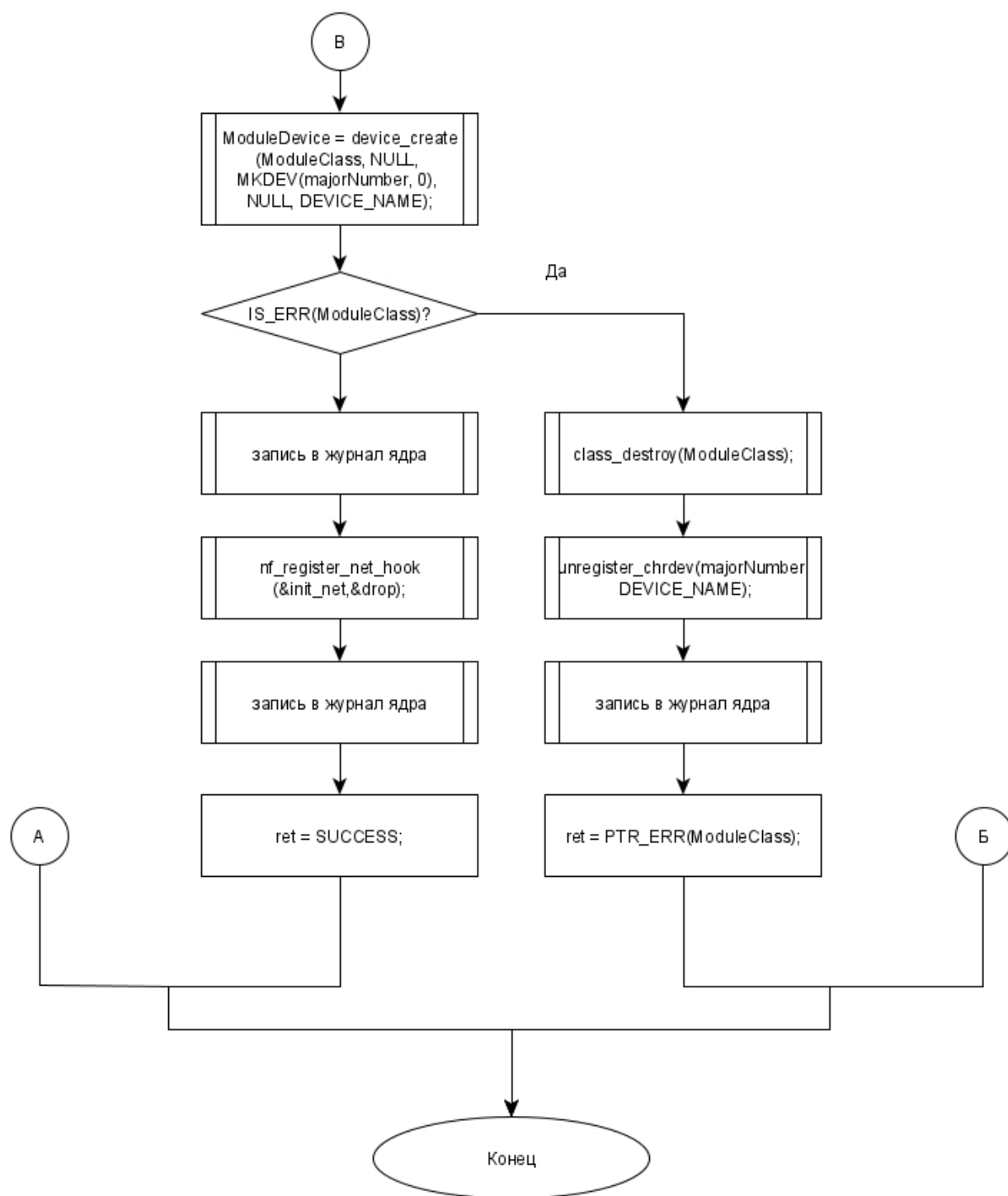


Рис. 4: Схема обработки пакетов

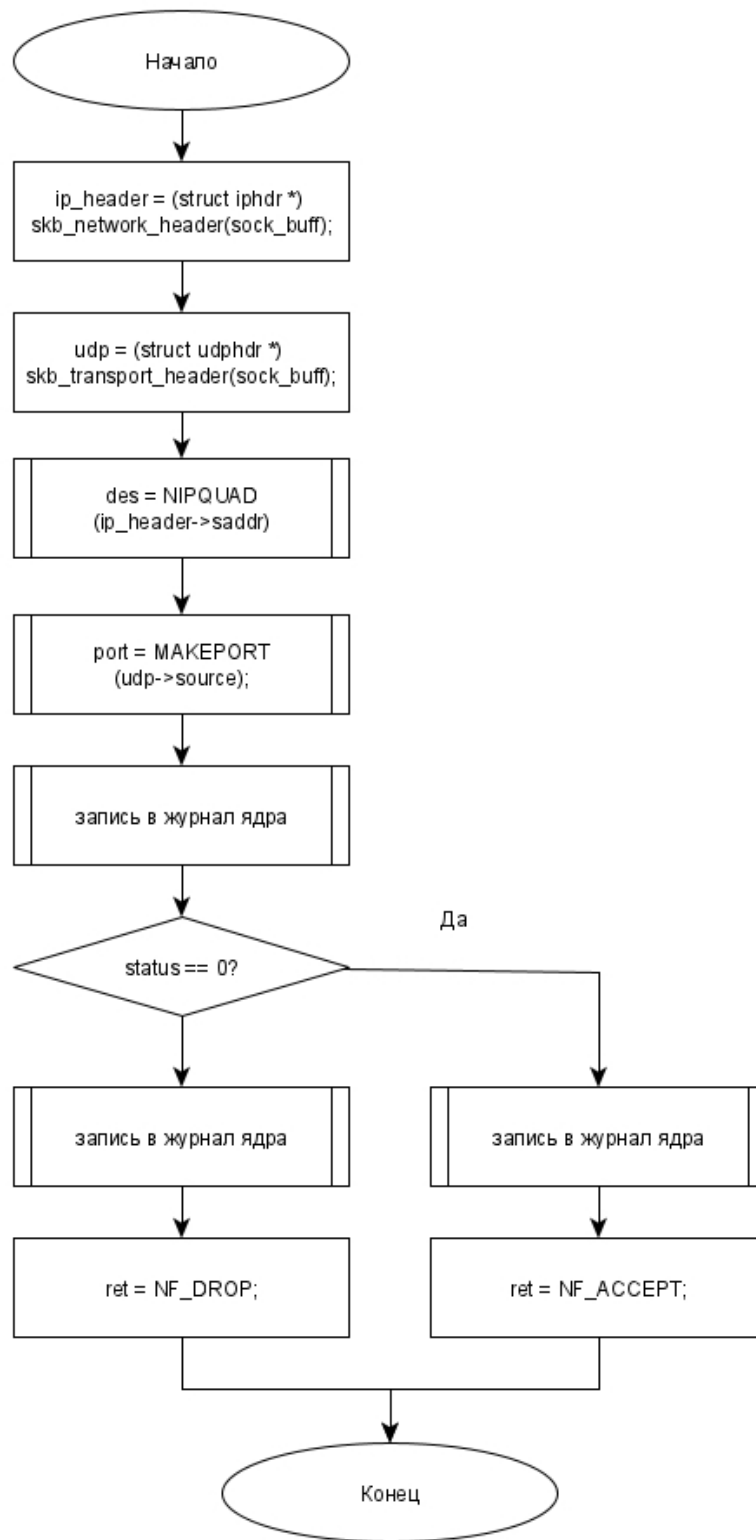
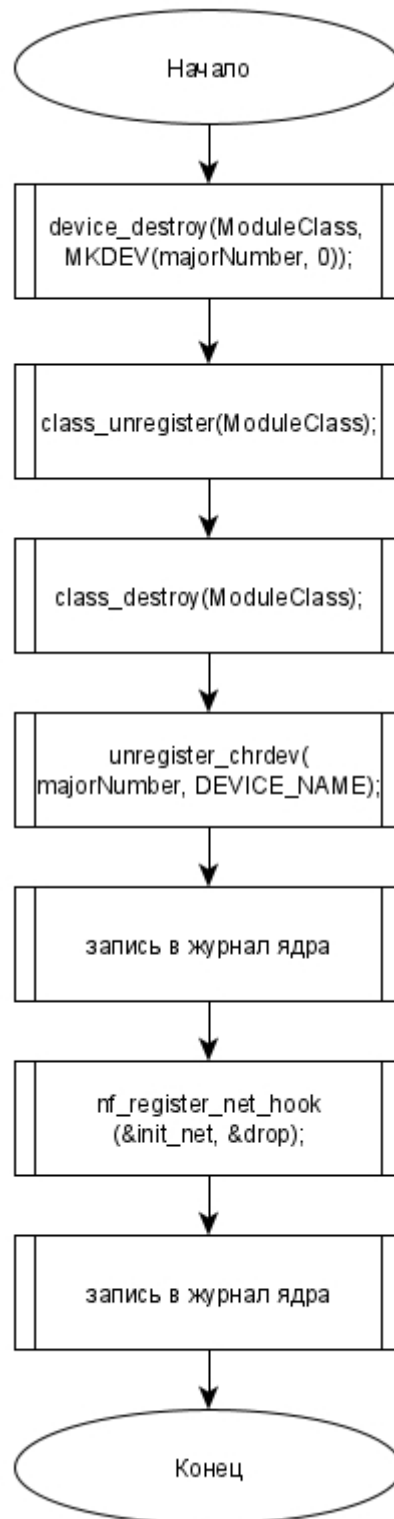


Рис. 5: Схема завершения работы модуля



Вывод

В данном разделе были рассмотрены требования к программе, основные сведения о реализуемых модулях, предоставлены схемы, описывающие работу модуля.

Технологическая часть

В данном разделе будет предоставлен листинг реализованных модулей и проведена апробация.

Выбор языка программирования и среды разработки

В качестве языка программирования был выбран C, поскольку при помощи этого языка реализованы все модули ядра и драйверы в ОС Linux.

В качестве среды разработки был выбран стандартный текстовый редактор ОС Linux.

Содержимое Makefile

Ниже представлено содержимое Makefile.

Листинг 11: Содержимое Makefile

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

Описание некоторых моментов реализации

В листинге 12 производится инициализация модуля ядра с помощью макросов входа/выхода, в листинге 13 показаны макросы модуля, а в листингах 14 и 15 продемонстрированы конструктор и деструктор модуля.

Листинг 12: Инициализация модуля ядра

```
1 module_init(pd_init);
2 module_exit(pd_exit);
```

Листинг 13: Макросы модуля

```
1 MODULE_LICENSE("GPL");
2 MODULE_AUTHOR("Furdik Nikita");
3 MODULE_DESCRIPTION("TCP/UDP packet manager");
4 MODULE_VERSION("1.0");
```

Листинг 14: Конструктор модуля

```

1 static int __init pd_init(void) {
2     printk(KERN_INFO "packet manager :: initializing the device driver\n
3         ");
4
5     majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
6     if (majorNumber < 0){
7         printk(KERN_ALERT "packet manager :: device failed to register a
8             major number\n");
9         return majorNumber;
10    }
11    printk(KERN_INFO "packet manager :: device registered correctly with
12        a major number %d\n", majorNumber);
13    ModuleClass = class_create(THIS_MODULE, CLASS_NAME);
14    if (IS_ERR(ModuleClass)){
15        unregister_chrdev(majorNumber, DEVICE_NAME);
16        printk(KERN_ALERT "packet manager :: failed to register device
17            class\n");
18        return PTR_ERR(ModuleClass);
19    }
20    printk(KERN_INFO "packet manager :: device class registered
21        correctly\n");
22    ModuleDevice = device_create(ModuleClass, NULL, MKDEV(majorNumber,
23        0), NULL, DEVICE_NAME);
24    if (IS_ERR(ModuleDevice)){
25        class_destroy(ModuleClass);
26        unregister_chrdev(majorNumber, DEVICE_NAME);
27        printk(KERN_ALERT "packet manager :: Failed to create the device\n
28            ");
29        return PTR_ERR(ModuleDevice);
30    }
31    printk(KERN_INFO "packet manager :: device class created correctly\n
32        ");
33    printk(KERN_INFO "starting the device!\n");
34    nf_register_net_hook(&init_net,&drop);
35    return 0;
36 }

```

Листинг 15: Деструктор модуля

```

1 static void __exit pd_exit(void) {
2     device_destroy(ModuleClass, MKDEV(majorNumber, 0));

```

```

3 class _unregister (ModuleClass);
4 class _destroy (ModuleClass);
5 unregister_chrdev (majorNumber, DEVICE_NAME);
6 printk (KERN_INFO "packet manager :: stopping the device!\n");
7 nf_unregister_net_hook (&init_net, &drop);
8
9 }

```

Далее покажем функции структуры file_operations.

Листинг 16: Функции структуры file_operations

```

1 static int devel_open (struct inode *inodep, struct file *filep){
2     numberOpens++;
3     printk (KERN_INFO "packet manager :: device has been opened %d time(s)
4         )\n", numberOpens);
5     return 0;
6 }
7
8 static ssize_t dev_write (struct file *filep, const char *buffer,
9     size_t len, loff_t *offset){
10     sprintf (list[listliterator], "%s", buffer);
11     printk (KERN_INFO "packet manager :: received from the user: %s mode
12         \n", list[listliterator]);
13     if (!strcmp (list[listliterator], "white")){
14         mode = 1;
15         listliterator = 0;
16     }else if (!strcmp (list[listliterator], "black")){
17         mode = 0;
18         listliterator = 0;
19     }else{
20         listliterator++;
21     }
22     return sizeof (list[listliterator]);
23 }
24
25 static int dev_release (struct inode *inodep, struct file *filep){
26     printk (KERN_INFO "packet manager :: device successfully closed\n");
27     return 0;
28 }

```

И, наконец, опишем функцию `icmp_hook`, которая вызывается при срабатывании хука.

Листинг 17: func `icmp_hook`

```
1 unsigned int icmp_hook(unsigned int hooknum, struct sk_buff *skb,
2 const struct net_device *in, const struct net_device *out, int(*okfn)(
3     struct sk_buff *)) {
4     int port;
5     char des[100];
6
7     sock_buff = skb;
8
9     ip_header = (struct iphdr *) skb_network_header(sock_buff);
10    udp = (struct udphdr *) skb_transport_header(sock_buff);
11
12    port = MAKEPORT(udp->source);
13
14    sprintf(des, "%d.%d.%d.%d:%d", NIPQUAD(ip_header->saddr), port);
15    printk(KERN_INFO "packet manager :: a packet was recieved from %s\n",
16        des);
17    printk(KERN_INFO "packet manager :: we are in %s mode\n", mode == 0
18        ? "black list" : "white list");
19
20
21    int status = 0;
22    if(mode == 1)
23        status = 1;
24
25    if(status == 1){
26        printk(KERN_INFO "packet manager :: packet has been accepted!");
27        return NF_ACCEPT;
28    }
29    else{
30        printk(KERN_INFO "packet manager :: dropping the packet!");
31        return NF_DROP;
32    }
33 }
```

Пример работы загружаемого модуля

Загрузим модуль ядра и проверим его успешную загрузку.

Рис. 6: Загрузка модуля

```
schoolboychik@schoolboychik-VirtualBox:~/coursework$ sudo insmod pd.ko
schoolboychik@schoolboychik-VirtualBox:~/coursework$ lsmod | grep pd
pd                24576  0
ppdev             24576  0
parport           53248  3 parport_pc,lp,ppdev
schoolboychik@schoolboychik-VirtualBox:~/coursework$ sudo dmesg
[ 8798.047587] packet manager :: initializing the device driver
[ 8798.047589] packet manager :: device registered correctly with a major number
240
[ 8798.047593] packet manager :: device class registered correctly
[ 8798.049014] packet manager :: device class created correctly
[ 8798.049014] starting the device!
[ 8800.205535] packet manager :: a packet was recieved from 192.168.0.1:4452
[ 8800.205552] packet manager :: we are in black list mode
[ 8800.205556] packet manager :: dropping the packet!
[ 8802.406585] packet manager :: a packet was recieved from 192.168.0.104:5632
[ 8802.406602] packet manager :: we are in black list mode
[ 8802.406607] packet manager :: dropping the packet!
[ 8806.837161] packet manager :: a packet was recieved from 13.33.240.122:443
[ 8806.837179] packet manager :: we are in black list mode
[ 8806.837183] packet manager :: dropping the packet!
```

Изначально устанавливается политика черного списка, т.е. все входящие пакеты отклоняются, как видно из рисунка 6.

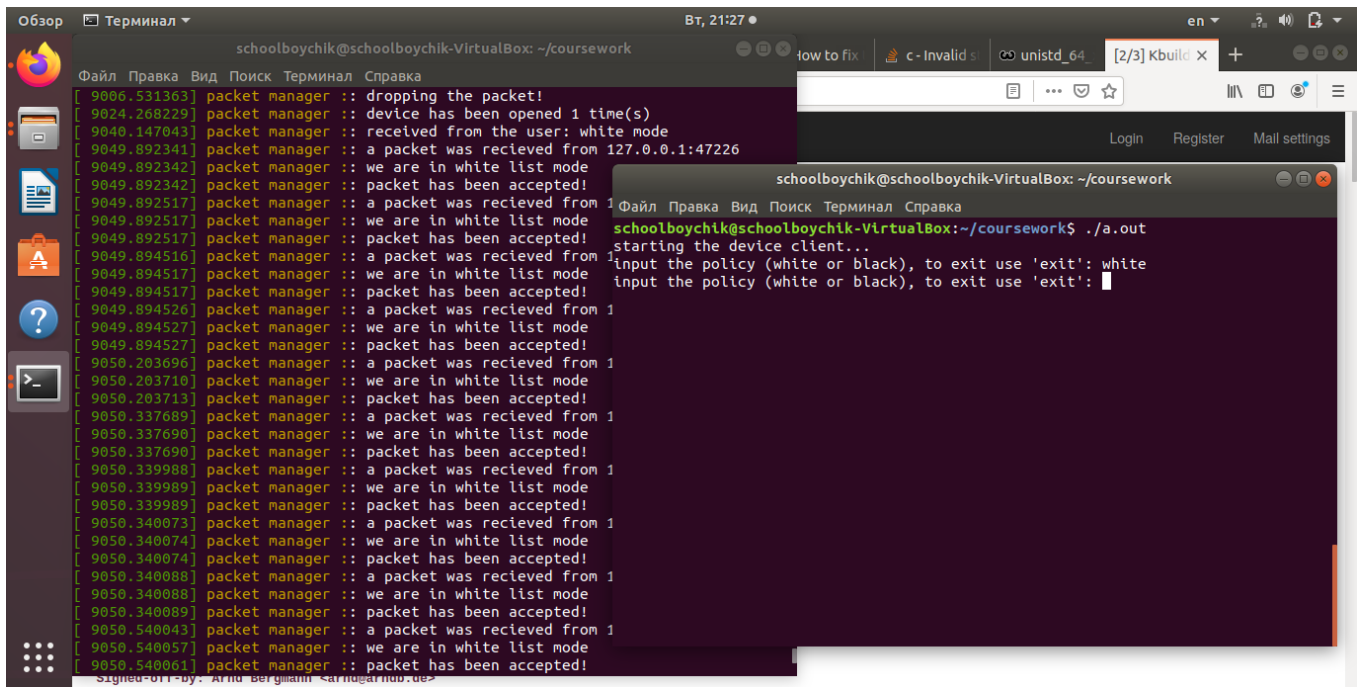
Запустим клиент и изменим политику. В случае неудачного открытия драйвера устройства, мы увидим сообщение об ошибке (рисунок 7).

Рис. 7: Неудачный запуск клиента

```
schoolboychik@schoolboychik-VirtualBox:~/coursework$ ./a.out
starting the device client...
failed to open the device...schoolboychik@schoolboychik-Virtual
```

Успешное изменение политики представлено на рисунке 8.

Рис. 8: Смена политики принятия пакетов



```
schoolboychik@schoolboychik-VirtualBox: ~/coursework
packet manager :: dropping the packet!
[9024.268229] packet manager :: device has been opened 1 time(s)
[9040.147043] packet manager :: received from the user: white mode
[9049.892341] packet manager :: a packet was recieved from 127.0.0.1:47226
[9049.892342] packet manager :: we are in white list mode
[9049.892342] packet manager :: packet has been accepted!
[9049.892517] packet manager :: a packet was recieved from 1
[9049.892517] packet manager :: we are in white list mode
[9049.894516] packet manager :: a packet was recieved from 1
[9049.894516] packet manager :: packet has been accepted!
[9049.894517] packet manager :: we are in white list mode
[9049.894517] packet manager :: packet has been accepted!
[9049.894526] packet manager :: a packet was recieved from 1
[9049.894527] packet manager :: we are in white list mode
[9049.894527] packet manager :: packet has been accepted!
[9050.203696] packet manager :: a packet was recieved from 1
[9050.203710] packet manager :: we are in white list mode
[9050.203713] packet manager :: packet has been accepted!
[9050.337689] packet manager :: a packet was recieved from 1
[9050.337690] packet manager :: we are in white list mode
[9050.337690] packet manager :: packet has been accepted!
[9050.339988] packet manager :: a packet was recieved from 1
[9050.339989] packet manager :: we are in white list mode
[9050.339989] packet manager :: packet has been accepted!
[9050.340073] packet manager :: a packet was recieved from 1
[9050.340074] packet manager :: we are in white list mode
[9050.340074] packet manager :: packet has been accepted!
[9050.340088] packet manager :: a packet was recieved from 1
[9050.340088] packet manager :: we are in white list mode
[9050.340089] packet manager :: packet has been accepted!
[9050.540043] packet manager :: a packet was recieved from 1
[9050.540057] packet manager :: we are in white list mode
[9050.540061] packet manager :: packet has been accepted!

schoolboychik@schoolboychik-VirtualBox: ~/coursework$ ./a.out
starting the device client...
input the policy (white or black), to exit use 'exit': white
input the policy (white or black), to exit use 'exit':
```

Вывод

Были реализованы модуль ядра и приложение-клиент, листинги которых были предоставлены в данном разделе. Помимо этого, программное обеспечение было протестировано на наличие ошибок.

Заключение

Во время выполнения курсового проекта были достигнуты поставленные цель и задачи: проанализированы способы перехвата входящих TCP/UDP пакетов, разработаны алгоритмы, позволяющие управлять входящими пакетами, а также спроектирован и реализован загружаемый модуль ядра.

В ходе выполнения поставленных задач были изучены возможности языка С, получены знания в области написания загружаемых модулей ядра и драйверов.

Список литературы

1. Транспортные протоколы TCP и UDP [Электронный ресурс]. Режим доступа: <http://osnovy-setei.ru/transportnye-protokoly-tcp-i-udp.html> (дата обращения 05.12.2020).
2. Рязанова Н.Ю. - Курс лекций по "Операционным системам"[Текст], Москва 2020 год.
3. У. Ричард Стивенс, Стивен А. Раго - UNIX. Профессиональное программирование. 3-е изд. - Москва: Питер, 2018. - 944 с.
4. Iptables [Электронный ресурс]. Режим доступа: <https://help.ubuntu.ru/wiki/iptables> (дата обращения 18.12.2020).
5. Вахалия Ю. UNIX изнутри. – СПб.: Питер, 2003. – 844 с.
6. Major and Minor Numbers [Электронный ресурс]. Режим доступа: <https://www.oreilly.com/library/view/linux-device-drivers/0596000081/ch03s02.html> (дата обращения 18.12.2020).
7. Linux Kernel Module Programming: Hello World Program [Электронный ресурс]. Режим доступа: <https://www.geeksforgeeks.org/linux-kernel-module-programming-hello-world-program/> (дата обращения 18.12.2020).
8. Как написать свой первый Linux device driver [Электронный ресурс]. Режим доступа: <https://habr.com/ru/post/337946/> (дата обращения 19.12.2020).

Приложение А

```
1 #include <linux/kernel.h>
2 #include <linux/module.h>
3 #include <linux/netfilter.h>
4 #include <linux/netfilter_ipv4.h>
5 #include <linux/skbuff.h>
6 #include <linux/udp.h>
7 #include <linux/icmp.h>
8 #include <linux/ip.h>
9 #include <linux/inet.h>
10 #include <linux/init.h>
11 #include <linux/device.h>
12 #include <linux/fs.h>
13 #include <linux/uaccess.h>
14
15
16 #define DEVICE_NAME "pd"
17 #define CLASS_NAME "pd"
18 #define NIPQUAD(addr) ((unsigned char *)&addr)[0], ((unsigned char *)&
    addr)[1], ((unsigned char *)&addr)[2], ((unsigned char *)&addr)[3]
19 #define MAKEPORT(port) ((unsigned char *)&port)[0] * 256 + ((unsigned
    char *)&port)[1]
20
21 MODULE_LICENSE("GPL");
22 MODULE_AUTHOR("Furdik Nikita");
23 MODULE_DESCRIPTION("TCP/UDP packet manager");
24 MODULE_VERSION("1.0");
25
26
27 char list[100][100];
28 int listIterator = 0;
29 int mode = 0; //0 == black and 1 == white
30
31 static int    majorNumber;
32 static int    numberOpens = 0;
33 static struct class * ModuleClass = NULL;
34 static struct device * ModuleDevice = NULL;
35
36 static int    devel_open(struct inode *, struct file *);
37 static int    dev_release(struct inode *, struct file *);
```

```

38 static ssize_t dev_write(struct file *, const char *, size_t, loff_t
    *);
39
40 char buffer[48];
41 struct sk_buff * sock_buff;
42 struct iphdr * ip_header;
43 struct udphdr * udp;
44
45 unsigned int icmp_hook(unsigned int hooknum, struct sk_buff *skb,
    const struct net_device *in, const struct net_device *out, int(*
    okfn)(struct sk_buff *));
46
47 static struct file_operations fops =
48 {
49     .open = devel_open,
50     .write = dev_write,
51     .release = dev_release,
52 };
53
54 static struct nf_hook_ops drop __read_mostly = {
55     .pf = NFPROTO_IPV4,
56     .priority= NF_IP_PRI_FIRST,
57     .hooknum = NF_INET_LOCAL_IN,
58     .hook = (nf_hookfn *) icmp_hook
59 };
60
61 static int __init ICMPDrop_init(void) {
62     printk(KERN_INFO "packet manager :: initializing the device driver\n
        ");
63
64     majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
65     if (majorNumber < 0){
66         printk(KERN_ALERT "packet manager :: device failed to register a
            major number\n");
67         return majorNumber;
68     }
69     printk(KERN_INFO "packet manager :: device registered correctly with
        a major number %d\n", majorNumber);
70     ModuleClass = class_create(THIS_MODULE, CLASS_NAME);
71     if (IS_ERR(ModuleClass)){
72         unregister_chrdev(majorNumber, DEVICE_NAME);

```

```

73     printk(KERN_ALERT "packet manager :: failed to register device
        class\n");
74     return PTR_ERR(ModuleClass);
75 }
76 printk(KERN_INFO "packet manager :: device class registered
        correctly\n");
77 ModuleDevice = device_create(ModuleClass, NULL, MKDEV(majorNumber,
        0), NULL, DEVICE_NAME);
78 if (IS_ERR(ModuleDevice)){
79     class_destroy(ModuleClass);
80     unregister_chrdev(majorNumber, DEVICE_NAME);
81     printk(KERN_ALERT "packet manager :: Failed to create the device\n
        ");
82     return PTR_ERR(ModuleDevice);
83 }
84 printk(KERN_INFO "packet manager :: device class created correctly\n
        ");
85 printk(KERN_INFO "starting the device!\n");
86 nf_register_net_hook(&init_net,&drop);
87 return 0;
88 }
89
90
91 static void __exit ICMPDrop_exit(void) {
92     device_destroy(ModuleClass, MKDEV(majorNumber, 0));
93     class_unregister(ModuleClass);
94     class_destroy(ModuleClass);
95     unregister_chrdev(majorNumber, DEVICE_NAME);
96     printk(KERN_INFO "packet manager :: stopping the device!\n");
97     nf_unregister_net_hook(&init_net, &drop);
98 }
99
100
101
102 unsigned int icmp_hook(unsigned int hooknum, struct sk_buff *skb,
        const struct net_device *in, const struct net_device *out, int(*
        okfn)(struct sk_buff *)) {
103     sock_buff = skb;
104
105     ip_header = (struct iphdr *) skb_network_header(sock_buff);
106     udp = (struct udphdr *) skb_transport_header(sock_buff);

```

```

107
108  int port = MAKEPORT(udp->source);
109
110  char des[100];
111  sprintf(des, "%d.%d.%d.%d:%d", NIPQUAD(ip_header->saddr), port);
112  printk(KERN_INFO "packet manager :: a packet was recieved from %s\n",
        des);
113  printk(KERN_INFO "packet manager :: we are in %s mode\n", mode == 0
        ? "black list" : "white list");
114
115  int status = 0;
116  if(mode == 1)
117  status = 1;
118
119  for(int i = 0; i < listliterator; i++)
120  {
121  if(mode == 0)
122  {
123      if(!strcmp(des, list[i]))
124      {
125          status = 1;
126          break;
127      }
128  }
129  else
130  {
131      if(!strcmp(des, list[i]))
132      {
133          status = 0;
134          break;
135      }
136  }
137  }
138
139  if(status == 1){
140      printk(KERN_INFO "packet manager :: packet has been accepted!");
141      return NF_ACCEPT;
142  }else{
143      printk(KERN_INFO "packet manager :: dropping the packet!");
144      return NF_DROP;
145  }

```

```

146 }
147
148
149 static int devel_open(struct inode *inodep, struct file *filep){
150     numberOpens++;
151     printk(KERN_INFO "packet manager :: device has been opened %d time(s)
152         )\n", numberOpens);
153     return 0;
154 }
155
156 static ssize_t dev_write(struct file *filep, const char *buffer,
157     size_t len, loff_t *offset){
158     sprintf(list[listliterator], "%s", buffer);
159     printk(KERN_INFO "packet manager :: received from the user: %s mode
160         \n", list[listliterator]);
161     if(!strcmp(list[listliterator], "white")){
162         mode = 1;
163         listliterator = 0;
164     }else if(!strcmp(list[listliterator], "black")){
165         mode = 0;
166         listliterator = 0;
167     }else{
168         listliterator++;
169     }
170     return sizeof(list[listliterator]);
171 }
172
173 static int dev_release(struct inode *inodep, struct file *filep){
174     printk(KERN_INFO "packet manager :: device successfully closed\n");
175     return 0;
176 }
177
178 module_init(ICMPDrop_init);
179 module_exit(ICMPDrop_exit);

```