



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 КУРСОВОЙ РАБОТЕ***  
***НА ТЕМУ:***

«Разработка загружаемого модуля ядра Linux для отключения  
сетевого оборудования системы при подключении  
USB-устройства»

Студент группы **ИУ7-73Б**

\_\_\_\_\_  
(Подпись, дата)

**С. Кононенко**  
(И.О. Фамилия)

Руководитель курсовой работы

\_\_\_\_\_  
(Подпись, дата)

**Н.Ю. Рязанова**  
(И.О. Фамилия)

Москва — 2022г.

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)  
« \_\_\_\_ » \_\_\_\_\_ 20 \_\_\_\_ г.

## З А Д А Н И Е на выполнение курсовой работы

по дисциплине Операционные системы

Студент группы ИУ7-73Б

Кононенко Сергей  
(Фамилия, имя, отчество)

Тема курсовой работы Разработка загружаемого модуля ядра Linux для отключения сетевого оборудования системы при подключении USB-устройства

Направленность КР (учебная, исследовательская, практическая, производственная, др.)

учебная

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения работы: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

**Задание:** разработать загружаемый модуль ядра Linux, позволяющий отключать сетевое оборудование системы при подключении USB-устройств, не находящихся в доверенном списке. При подключении USB-устройств из доверенного списка, производить включение сетевого оборудования в случае, если до момента включения было произведено отключение

**Оформление курсовой работы:** Расчетно-пояснительная записка на 25-30 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку задачи, введение, аналитическую часть, конструкторскую часть, технологическую часть, заключение, список литературы, приложения

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 « \_\_\_\_ » \_\_\_\_\_ 20\_\_ г.

Руководитель курсовой работы

\_\_\_\_\_  
(Подпись, дата)

Н.Ю. Рязанова  
(И.О.Фамилия)

Студент

\_\_\_\_\_  
(Подпись, дата)

С.Кононенко  
(И.О.Фамилия)

# Оглавление

<b>Введение</b>	<b>5</b>
<b>1 Аналитическая часть</b>	<b>6</b>
1.1 Постановка задачи . . . . .	6
1.2 Обработка событий от USB-устройств . . . . .	6
1.2.1 usbmon . . . . .	6
1.2.2 udevadm . . . . .	8
1.2.3 Уведомители . . . . .	9
1.3 USB-устройства в ядре Linux . . . . .	10
1.3.1 Структура <code>usb_device</code> . . . . .	10
1.3.2 Структура <code>usb_device_id</code> . . . . .	13
1.4 Особенности разработки загружаемых модулей ядра Linux .	14
1.4.1 Пользовательское пространство памяти и простран- ство памяти ядра . . . . .	14
1.4.2 Запуск программ пользовательского пространства в пространстве ядра . . . . .	15
<b>2 Конструкторская часть</b>	<b>17</b>
2.1 IDEF0 последовательность преобразований . . . . .	17
2.2 Структура программного обеспечения . . . . .	18
2.3 Алгоритм отслеживания событий . . . . .	18
2.4 Алгоритм добавления подключаемых устройств в список подключенных устройств . . . . .	19
2.5 Алгоритм работы обработчика событий . . . . .	20
<b>3 Технологическая часть</b>	<b>21</b>
3.1 Выбор языка и среды программирования . . . . .	21
3.2 Хранение информации об отслеживаемых устройствах . . .	21
3.3 Идентификация устройства как доверенного . . . . .	22
3.4 Обработка событий USB-устройства . . . . .	24
3.5 Регистрация уведомителя для USB-устройств . . . . .	26

3.6	Примеры работы разработанного ПО . . . . .	27
	<b>Заключение</b>	<b>28</b>
	<b>Литература</b>	<b>29</b>
	<b>Приложение А</b>	<b>31</b>

# Введение

Существует много способов для проведения кибератаки, одним из которых является атака при помощи USB-устройства. При проведении такой атаки при подключении устройства к компьютеру можно запустить вредоносный программный код на выполнение, который может как удалить важные данные из системы, так и отправить их третьему лицу через интернет [1].

Для того, чтобы предотвратить кибератаку, проводимую посредством подключенного USB-устройства, следует строго отслеживать активные устройства в системе. С точки зрения пользовательского опыта, нет возможности запретить подключать новые устройства к компьютеру, так как большинство устройств ввода-вывода подключаются через USB, поэтому мониторинг активных устройств, их анализ и последующее принятие решений являются хорошим способом для избежания кибератаки.

Чтобы обезопасить свои данные от утечки и передачи третьим лицам, при подключении устройства можно отключать сетевые устройства системы. Таким образом исключается возможность передачи данных через интернет.

Цель работы — разработать загружаемый модуль ядра Linux для отключения сетевого оборудования системы при подключении USB-устройства.

# 1 Аналитическая часть

## 1.1 Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать загружаемый модуль ядра Linux для отключения сетевого оборудования системы при подключении USB-устройства. Для решения данной задачи необходимо:

- проанализировать методы обработки событий, возникающих при взаимодействии с USB-устройствами;
- проанализировать структуры и функции ядра, предоставляющие информацию о USB-устройствах;
- разработать алгоритмы и структуру программного обеспечения;
- реализовать программное обеспечение;
- исследовать разработанное программное обеспечение.

## 1.2 Обработка событий от USB-устройств

Для обработки событий, возникающих при работе с USB-устройствами, например, таких как подключение или отключение устройства, необходимо узнать о возникновении события и выполнить необходимую обработку после возникновения события.

Далее будут рассмотрены существующие различные подходы к определению возникновения событий от USB-устройств и выбран наиболее подходящий для реализации в данной работе.

### 1.2.1 `usbmon`

`usbmon` [2] — средство ядра Linux, используемое для сбора информации о событиях, произошедших на устройствах ввода-вывода, подключенных

через USB.

**usbmon** предоставляет информацию о запросах, сделанных драйверами устройств к драйверам хост-контроллера (HCD). Если драйвера хост-контроллера неисправны, то данные, предоставленные **usbmon**, могут не соответствовать действительным переданным данным.

В настоящее время реализованы два программных интерфейса для взаимодействия с **usbmon**: текстовый и двоичный. Двоичный интерфейс доступен через символьное устройство в пространстве имен `/dev`. Текстовый интерфейс устарел, но сохраняется для совместимости.

В листинге 1.1 представлена структура ответа, полученного после события, случившегося на USB-устройстве (например, подключение к компьютеру).

Листинг 1.1 – Структура `usbmon_packet`

```
1 struct usbmon_packet {
2     u64 id; /* 0: URB ID - from submission to callback */
3     unsigned char type; /* 8: Same as text; extensible. */
4     unsigned char xfer_type; /* ISO (0), Intr, Control, Bulk (3) */
5     unsigned char epnum; /* Endpoint number and transfer direction */
6     unsigned char devnum; /* Device address */
7     u16 busnum; /* 12: Bus number */
8     char flag_setup; /* 14: Same as text */
9     char flag_data; /* 15: Same as text; Binary zero is OK. */
10    s64 ts_sec; /* 16: gettimeofday */
11    s32 ts_usec; /* 24: gettimeofday */
12    int status; /* 28: */
13    unsigned int length; /* 32: Length of data (submitted or actual) */
14    unsigned int len_cap; /* 36: Delivered length */
15    union { /* 40: */
16        unsigned char setup[SETUP_LEN]; /* Only for Control S-type */
17        struct iso_rec { /* Only for ISO */
18            int error_count;
19            int numdesc;
20        } iso;
21    } s;
22    int interval; /* 48: Only for Interrupt and ISO */
23    int start_frame; /* 52: For ISO */
24    unsigned int xfer_flags; /* 56: copy of URB's transfer_flags */
25    unsigned int ndesc; /* 60: Actual number of ISO descriptors */
26 }; /* 64 total length */
```

Особенности **usbmon**:

- возможность просматривать собранную информацию через специаль-

ное ПО (например, Wireshark);

- возможность отслеживать события на одном порте USB или на всех сразу;
- отсутствие возможности вызова обработчика при возникновении определенного события.

`usbmon` позволяет отслеживать события, но не позволяет реагировать на них без программной доработки для реализации обработчика.

## 1.2.2 udevadm

`udevadm` [3] — инструмент для управления устройствами `udev`. Структура `udev` описана в библиотеке `libudev` [4], которая не является системной библиотекой Linux. В данной библиотеке представлен программный интерфейс для мониторинга и взаимодействия с локальными устройствами.

При помощи `udevadm` можно получить полную информацию об устройстве, полученную из его представления в `sysfs`, чтобы создать корректные правила и обработчики событий для устройства. Кроме того можно получить список событий для устройства, установить наблюдение за ним.

В листинге 1.2 представлен пример правила обработки событий, задаваемого с помощью `udevadm`.

Листинг 1.2 – Правила `udevadm`

```
1 /* rules file */
2 SUBSYSTEM=="usb", ACTION=="add", ENV{DEVTYPE}=="usb_device", RUN+="/bin/device_added.sh"
3 SUBSYSTEM=="usb", ACTION=="remove", ENV{DEVTYPE}=="usb_device",
4     RUN+="/bin/device_removed.sh"
5
6 /* device_added.sh */
7 #!/bin/bash
8 echo "USB_device_added_at_$(date)" >>/tmp/scripts.log
9
10 /* device_removed.sh */
11 #!/bin/bash
12 echo "USB_device_removed_at_$(date)" >>/tmp/scripts.log
```

Особенности `udevadm`:



- возможность привязки своего обработчика к событию;
- невозможность использования интерфейса в ядре Linux;

### 1.2.3 Уведомители

Ядро Linux содержит механизм, называемый «уведомителями» (**notifiers**) или «цепочками уведомлений» (**notifiers chains**), который позволяет различным подсистемам подписываться на асинхронные события от других подсистем. Цепочки уведомлений в настоящее время активно используются в ядре; существуют цепочки для событий **hotplug** памяти, изменения политики частоты процессора, события **USB hotplug**, загрузки и выгрузки модулей, перезагрузки системы, изменения сетевых устройств [5].

В листинге 1.3 представлена структура **notifier\_block** [6].

Листинг 1.3 – Структура **notifier\_block**

```
1 struct notifier_block {
2     notifier_fn_t notifier_call;
3     struct notifier_block __rcu *next;
4     int priority;
5 };
```

Данная структура описана в `/include/linux/notifier.h`. Она содержит указатель на функцию–обработчик уведомления (**notifier\_call**), указатель на следующий уведомитель (**next**) и приоритет уведомителя (**priority**). Уведомители с более высоким значением приоритета выполняются первее.

В листинге 1.4 представлена сигнатура функции **notifier\_call**.

Листинг 1.4 – Тип **notifier\_fn\_t**

```
1 typedef int (*notifier_fn_t)(struct notifier_block *nb, unsigned long action, void
    *data);
```

Сигнатура содержит указатель на уведомитель (**nb**), действие, при котором срабатывает функция (**action**) и данные, которые передаются от действия в обработчик (**data**).

Для регистрации уведомителя для USB–портов используются функции регистрации и удаления уведомителя, представленные в листинге 1.5.

Листинг 1.5 – Уведомители на USB–портах

```

1 /* Events from the usb core */
2 #define USB_DEVICE_ADD 0x0001
3 #define USB_DEVICE_REMOVE 0x0002
4 #define USB_BUS_ADD 0x0003
5 #define USB_BUS_REMOVE 0x0004
6 extern void usb_register_notify(struct notifier_block *nb);
7 extern void usb_unregister_notify(struct notifier_block *nb);

```

Прототипы и константы для действий описаны в файле `/include/linux/notifier.h`, а реализации функций — в файле `/drivers/usb/core/notify.c`. Соответственно действие `USB_DEVICE_ADD` означает подключение нового устройства, а `USB_DEVICE_REMOVE` — удаление.

Особенности уведомителей:

- возможность привязки своего обработчика к событию;
- возможность добавления более чем одного обработчика событий;
- возможность использования интерфейса в загружаемом модуле ядра;

## 1.3 USB-устройства в ядре Linux

### 1.3.1 Структура `usb_device`

Для хранения информации о USB-устройстве в ядре используется структура `usb_device`, описанная в `/include/linux/usb.h` [7].

Структура `usb_device` представлена в листинге 1.6.

Листинг 1.6 – Структура `usb_device`

```

1 struct usb_device {
2     int devnum;
3     char devpath[16];
4     u32 route;
5     enum usb_device_state state;
6     enum usb_device_speed speed;
7     unsigned int rx_lanes;
8     unsigned int tx_lanes;
9     enum usb_ssp_rate ssp_rate;

```

```

10
11 struct usb_tt *tt;
12 int ttport;
13
14 unsigned int toggle[2];
15
16 struct usb_device *parent;
17 struct usb_bus *bus;
18 struct usb_host_endpoint ep0;
19
20 struct device dev;
21
22 struct usb_device_descriptor descriptor;
23 struct usb_host_bos *bos;
24 struct usb_host_config *config;
25
26 struct usb_host_config *actconfig;
27 struct usb_host_endpoint *ep_in[16];
28 struct usb_host_endpoint *ep_out[16];
29
30 char **rawdescriptors;
31
32 unsigned short bus_mA;
33 u8 portnum;
34 u8 level;
35 u8 devaddr;
36
37 unsigned can_submit:1;
38 unsigned persist_enabled:1;
39 unsigned have_langid:1;
40 unsigned authorized:1;
41 unsigned authenticated:1;
42 unsigned wusb:1;
43 unsigned lpm_capable:1;
44 unsigned usb2_hw_lpm_capable:1;
45 unsigned usb2_hw_lpm_bes1_capable:1;
46 unsigned usb2_hw_lpm_enabled:1;
47 unsigned usb2_hw_lpm_allowed:1;
48 unsigned usb3_lpm_u1_enabled:1;
49 unsigned usb3_lpm_u2_enabled:1;
50 int string_langid;
51
52 /* static strings from the device */
53 char *product;
54 char *manufacturer;
55 char *serial;
56
57 struct list_head filelist;

```

```

58
59     int maxchild;
60
61     u32 quirks;
62     atomic_t urbnum;
63
64     unsigned long active_duration;
65
66 #ifdef CONFIG_PM
67     unsigned long connect_time;
68
69     unsigned do_remote_wakeup:1;
70     unsigned reset_resume:1;
71     unsigned port_is_suspended:1;
72 #endif
73     struct wusb_dev *wusb_dev;
74     int slot_id;
75     enum usb_device_removable removable;
76     struct usb2_lpm_parameters l1_params;
77     struct usb3_lpm_parameters u1_params;
78     struct usb3_lpm_parameters u2_params;
79     unsigned lpm_disable_count;
80
81     u16 hub_delay;
82     unsigned use_generic_driver:1;
83 };

```

Каждое USB-устройство должно соответствовать спецификации USB-IF [8], одним из требований которой является наличие идентификатора поставщика (Vendor ID (VID)) и идентификатор продукта (Product ID (PID)). Эти данные присутствуют в поле **descriptor** структуры **usb\_device**. В листинге 1.7 представлена структура дескриптора **usb\_device\_descriptor**, описанная в `/include/uapi/linux/usb/ch9.h`.

Листинг 1.7 – Структура **usb\_device\_descriptor**

```

1  /* USB_DT_DEVICE: Device descriptor */
2  struct usb_device_descriptor {
3      __u8 bLength;
4      __u8 bDescriptorType;
5
6      __le16 bcdUSB;
7      __u8 bDeviceClass;
8      __u8 bDeviceSubClass;
9      __u8 bDeviceProtocol;
10     __u8 bMaxPacketSize0;
11     __le16 idVendor;

```

```

12     __le16 idProduct;
13     __le16 bcdDevice;
14     __u8 iManufacturer;
15     __u8 iProduct;
16     __u8 iSerialNumber;
17     __u8 bNumConfigurations;
18 } __attribute__((packed));
19
20 #define USB_DT_DEVICE_SIZE 18

```

### 1.3.2 Структура usb\_device\_id

При подключении USB-устройства к компьютеру, оно идентифицируется и идентификационная информация записывается в структуру `usb_device_id` [9].

Структура `usb_device_id` представлена в листинге 1.8.

Листинг 1.8 – Структура `usb_device_id`

```

1 struct usb_device_id {
2     /* which fields to match against? */
3     __u16 match_flags;
4
5     /* Used for product specific matches; range is inclusive */
6     __u16 idVendor;
7     __u16 idProduct;
8     __u16 bcdDevice_lo;
9     __u16 bcdDevice_hi;
10
11     /* Used for device class matches */
12     __u8 bDeviceClass;
13     __u8 bDeviceSubClass;
14     __u8 bDeviceProtocol;
15
16     /* Used for interface class matches */
17     __u8 bInterfaceClass;
18     __u8 bInterfaceSubClass;
19     __u8 bInterfaceProtocol;
20
21     /* Used for vendor-specific interface matches */
22     __u8 bInterfaceNumber;
23
24     /* not matched against */
25     kernel_ulong_t driver_info
26     __attribute__((aligned(sizeof(kernel_ulong_t))));

```

## 1.4 Особенности разработки загружаемых модулей ядра Linux

### 1.4.1 Пользовательское пространство памяти и пространство памяти ядра

Пользовательские программы работают в пользовательском пространстве, а ядро и его модули — в пространстве ядра.

Операционная система должна обеспечивать программы доступом к аппаратной части компьютера, независимую работу программ и защиту от несанкционированного доступа к ресурсам. Решение этих задач становится возможным в случае, если процессор обеспечивает защиту системного программного обеспечения от прикладных программ.

Ядро Linux выполняется на самом высоком уровне, где разрешено выполнение любых инструкций и доступ к произвольным участкам памяти, а приложения выполняются на самом низком уровне, в котором процессор регулирует прямой доступ к аппаратной части и несанкционированный доступ к памяти. Ядро выполняет переход из пользовательского пространства в пространство ядра, когда приложение делает системный вызов или приостанавливается аппаратным прерыванием. Код ядра, выполняя системный вызов, работает в контексте процесса — он действует от имени вызывающего процесса и в состоянии получить данные в адресном пространстве процесса. Код, который обрабатывает прерывания, является асинхронным по отношению к процессам и не связан с каким-либо определенным процессом.

Ролью модуля ядра является расширение функциональности ядра без его перекомпиляции. Код модулей выполняется в пространстве ядра.

## 1.4.2 Запуск программ пользовательского пространства в пространстве ядра

Для запуска программ пространства пользователя из пространства ядра используется `usermode-helper` API. Чтобы создать процесс из пространства пользователя необходимо указать имя исполняемого файла, аргументы, с которыми требуется запустить программу, и переменные окружения [10].

В листинге 1.9 представлена структура процесса, использующегося в `usermode-helper` API и сигнатура функции вызова [11].

Листинг 1.9 – `usermode-helper` API

```
1 #define UMH_NO_WAIT 0 /* don't wait at all */
2 #define UMH_WAIT_EXEC 1 /* wait for the exec, but not the process */
3 #define UMH_WAIT_PROC 2 /* wait for the process to complete */
4 #define UMH_KILLABLE 4 /* wait for EXEC/PROC killable */
5
6 struct subprocess_info {
7     struct work_struct work;
8     struct completion *complete;
9     const char *path;
10    char **argv;
11    char **envp;
12    int wait;
13    int retval;
14    int (*init)(struct subprocess_info *info, struct cred *new);
15    void (*cleanup)(struct subprocess_info *info);
16    void *data;
17 } __randomize_layout;
18
19 extern int call_usermodehelper(const char *path, char **argv, char **envp, int wait);
```

## Выводы

Были рассмотрены методы обработки событий, возникающих при взаимодействии с USB-устройствами. Среди рассмотренных методов был выбран механизм уведомителей, так как он позволяет привязать свой обработчик события, а также реализован на уровне ядра Linux. Были рассмот-

рены структуры и функции ядра для работы с уведомителями, а также особенности разработки загружаемых модулей ядра.



## 2 Конструкторская часть

### 2.1 IDEF0 последовательность преобразований

На рисунках 2.1 и 2.2 представлена IDEF0 последовательность преобразований.

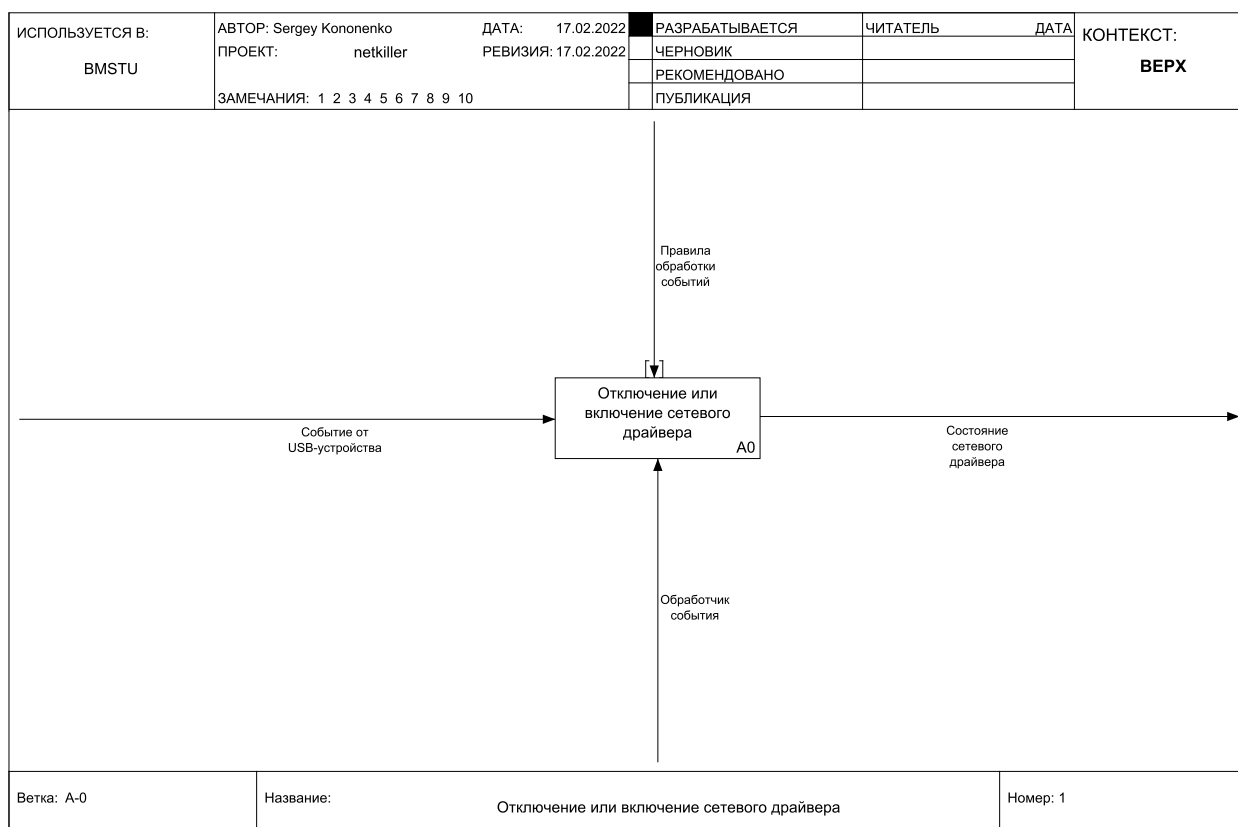


Рисунок 2.1 – Нулевой уровень преобразований

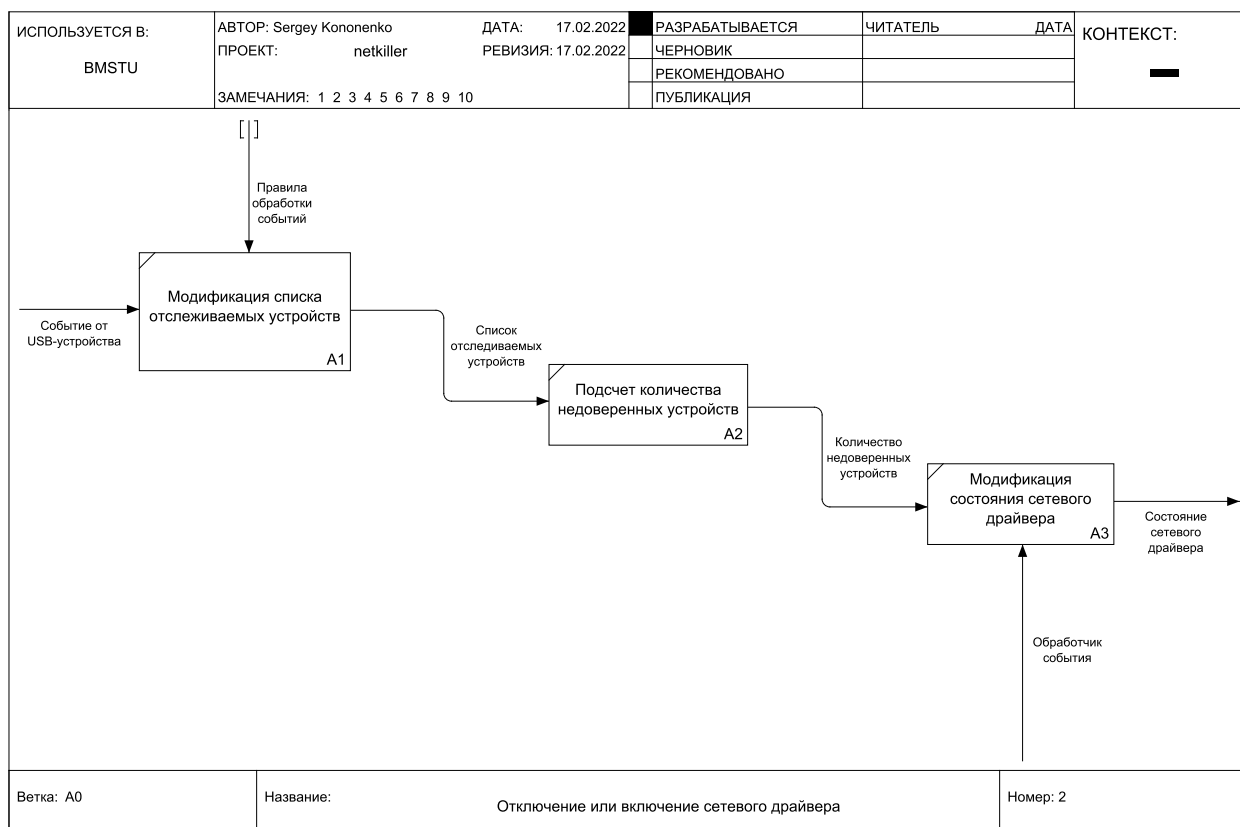


Рисунок 2.2 – Первый уровень преобразований

## 2.2 Структура программного обеспечения

В состав разрабатываемого программного обеспечения входит один загружаемый модуль ядра, который отслеживает подключенные USB-устройства и программно отключает сетевые устройства при наличии недоверенного устройства. Недоверенным устройством считается устройство, которое не идентифицируется в соответствии со списком допустимых устройств модуля. Список допустимых устройств задается в исходном коде модуля.

## 2.3 Алгоритм отслеживания событий

Для отслеживания событий подключения и отключения устройств в модуле ядра размещен соответствующий уведомитель, который будет зарегистрирован при загрузке модуля и удален при его удалении.

В листинге 2.1 представлена функция обработки событий.

## Листинг 2.1 – Обработка событий

```
1 // Handler for event's notifier.
2 static int notify(struct notifier_block *self, unsigned long action, void *dev)
3 {
4     // Events, which our notifier react.
5     switch (action)
6     {
7         case USB_DEVICE_ADD:
8             usb_dev_insert(dev);
9             break;
10        case USB_DEVICE_REMOVE:
11            usb_dev_remove(dev);
12            break;
13        default:
14            break;
15    }
16
17    return 0;
18 }
19
20 // React on different notifies.
21 static struct notifier_block usb_notify = {
22     .notifier_call = notify,
23 };
```

Для каждого события есть отдельный обработчик.

## 2.4 Алгоритм добавления подключаемых устройств в список подключенных устройств

Для хранения информации о подключенных устройствах будет использован связный список, хранящий информацию об идентификационных данных устройства.

Алгоритм модификации списка отслеживаемых устройств представлен в схеме на рисунке 2.3.

## 2.5 Алгоритм работы обработчика событий

На рисунке 2.3 представлен алгоритм работы обработчика событий .

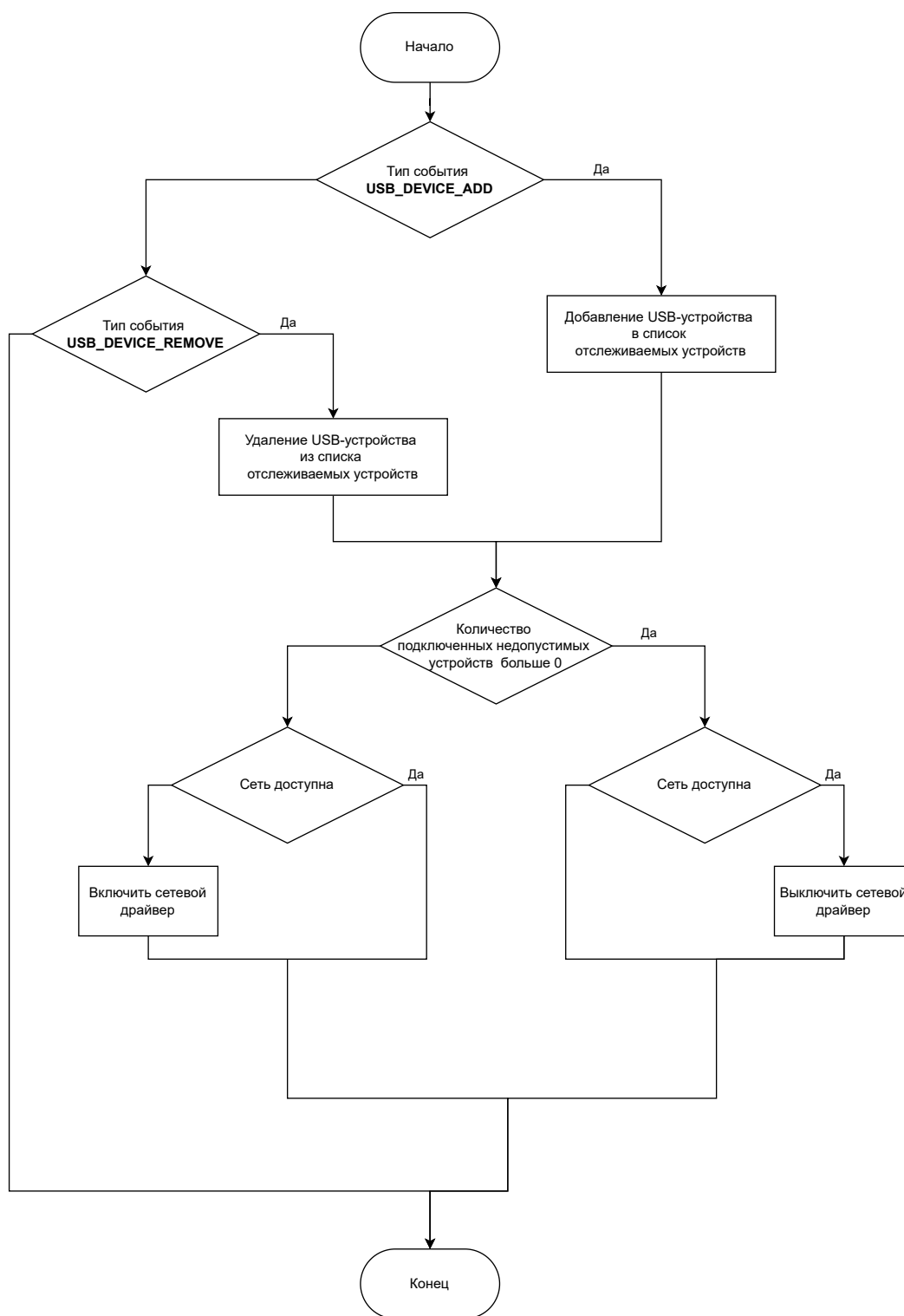


Рисунок 2.3 – Алгоритм обработчика событий

## 3 Технологическая часть

### 3.1 Выбор языка и среды программирования

Разработанный модуль ядра написан на языке программирования C [12]. Выбор языка программирования C основан на том, что исходный код ядра Linux, все его модули и драйверы написаны на данном языке.

В качестве компилятора выбран gcc [13].

В качестве среды разработки выбрана среда Visual Studio Code [14].

### 3.2 Хранение информации об отслеживаемых устройствах

Для хранения информации об отслеживаемых устройствах объявлена структура `int_usb_device`, которая хранит в себе идентификационные данные устройства (PID, VID) и указатель на элемент списка.

Структура `int_usb_device` представлена в листинге 3.1.

Листинг 3.1 – Структура `int_usb_device`

```
1 typedef struct int_usb_device
2 {
3     struct usb_device_id dev_id;
4     struct list_head list_node;
5 } int_usb_device_t;
```

При подключении или удалении устройства, создается экземпляр данной структуры и помещается в список отслеживаемых устройств.

В листинге 3.2 представлены функции для работы со списком отслеживаемых устройств.

Листинг 3.2 – Функции для работы со списком отслеживаемых устройств

```
1 // Add connected device to list of tracked devices.
2 static void add_int_usb_dev(struct usb_device *dev)
3 {
4     int_usb_device_t *new_usb_device = (int_usb_device_t
5         *)kmalloс(sizeof(int_usb_device_t), GFP_KERNEL);
```

```

5     struct usb_device_id new_id = {USB_DEVICE(dev->descriptor.idVendor,
        dev->descriptor.idProduct)};
6     new_usb_device->dev_id = new_id;
7     list_add_tail(&new_usb_device->list_node, &connected_devices);
8 }
9
10 // Delete device from list of tracked devices.
11 static void delete_int_usb_dev(struct usb_device *dev)
12 {
13     int_usb_device_t *device, *temp;
14     list_for_each_entry_safe(device, temp, &connected_devices, list_node)
15     {
16         if (is_dev_matched(dev, &device->dev_id))
17         {
18             list_del(&device->list_node);
19             kfree(device);
20         }
21     }
22 }

```

### 3.3 Идентификация устройства как доверенного

Для проверки устройства необходимо проверить его идентификационные данные с данными доверенных устройств. В листинге 3.3 представлены объявление списка доверенных устройств и функции для идентификации устройства.

Листинг 3.3 – Функции для идентификации устройств

```

1 struct usb_device_id allowed_devs[] = {
2     {USB_DEVICE(0x13fe, 0x3e00)},
3 };
4
5 // Match device with device id.
6 static bool is_dev_matched(struct usb_device *dev, const struct usb_device_id *dev_id)
7 {
8     // Check idVendor and idProduct, which are used.
9     if (dev_id->idVendor != dev->descriptor.idVendor || dev_id->idProduct !=
        dev->descriptor.idProduct)
10     {
11         return false;
12     }
13 }

```

```

14     return true;
15 }
16
17 // Match device id with device id.
18 static bool is_dev_id_matched(struct usb_device_id *new_dev_id, const struct
    usb_device_id *dev_id)
19 {
20     // Check idVendor and idProduct, which are used.
21     if (dev_id->idVendor != new_dev_id->idVendor || dev_id->idProduct !=
        new_dev_id->idProduct)
22     {
23         return false;
24     }
25
26     return true;
27 }
28
29 // Check if device is in allowed devices list.
30 static bool *is_dev_allowed(struct usb_device_id *dev)
31 {
32     unsigned long allowed_devs_len = sizeof(allowed_devs) / sizeof(struct usb_device_id);
33
34     int i;
35     for (i = 0; i < allowed_devs_len; i++)
36     {
37         if (is_dev_id_matched(dev, &allowed_devs[i]))
38         {
39             return true;
40         }
41     }
42
43     return false;
44 }
45
46 // Check if changed device is acknowledged.
47 static int count_not_acked_devs(void)
48 {
49     int_usb_device_t *temp;
50     int count = 0;
51
52     list_for_each_entry(temp, &connected_devices, list_node)
53     {
54         if (!is_dev_allowed(&temp->dev_id))
55         {
56             count++;
57         }
58     }
59

```

```
60     return count;
61 }
```

## 3.4 Обработка событий USB-устройства

При подключении устройство добавляется в список отслеживаемых устройств. После этого происходит проверка на наличие среди отслеживаемых устройств недоверенных, и, в случае если такие были найдены, происходит отключение драйвера сети. Отключение происходит путем вызова программы `modprobe` через `usermode-helper` API.

В листинге 3.4 представлен обработчик подключения USB-устройства.

Листинг 3.4 – Обработчик подключения USB-устройства

```
1 static void usb_dev_insert(struct usb_device *dev)
2 {
3     printk(KERN_INFO "netkiller: device connected with PID '%d' and VID '%d'\n",
4         dev->descriptor.idProduct, dev->descriptor.idVendor);
5     add_int_usb_dev(dev);
6     int not_acked_devs = count_not_acked_devs();
7
8     if (!not_acked_devs)
9     {
10         printk(KERN_INFO "netkiller: no not allowed devices connected, skipping network
11             killing\n");
12     }
13     else
14     {
15         printk(KERN_INFO "netkiller: %d not allowed devices connected, killing
16             network\n", not_acked_devs);
17         if (!is_network_down)
18         {
19             char *argv[] = {"/sbin/modprobe", "-r", "virtio_net", NULL};
20             char *envp[] = {"HOME=/", "TERM=linux", "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
21                 NULL};
22             if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC > 0))
23             {
24                 printk(KERN_WARNING "netkiller: unable to kill network\n");
25             }
26             else
27             {
28                 printk(KERN_INFO "netkiller: network is killed\n");
29                 is_network_down = true;
30             }
31         }
32     }
33 }
```



```

28     }
29 }
30 }

```

При отключении устройство удаляется из списка отслеживаемых устройств. После этого происходит проверка на наличие среди отслеживаемых устройств недоверенных, и, в случае если такие не были найдены, происходит включение драйвера сети. Включение также происходит путем вызова программы `modprobe` через `usermode-helper` API.

В листинге 3.5 представлен обработчик отключения USB-устройства.

Листинг 3.5 – Обработчик отключения USB-устройства

```

1 static void usb_dev_remove(struct usb_device *dev)
2 {
3     printk(KERN_INFO "netkiller:_device_disconnected_with_PID_'%d'_and_VID_'%d'\n",
4         dev->descriptor.idProduct, dev->descriptor.idVendor);
5     delete_int_usb_dev(dev);
6     int not_acked_devs = count_not_acked_devs();
7
8     if (not_acked_devs)
9     {
10         printk(KERN_INFO "netkiller:_%d_not_allowed_devices_connected,_nothing_to_do\n",
11             not_acked_devs);
12     }
13     else
14     {
15         if (is_network_down)
16         {
17             printk(KERN_INFO "netkiller:_all_not_allowed_devices_are_disconnected,_
18                 bringing_network_back\n");
19             char *argv[] = {"/sbin/modprobe", "virtio_net", NULL};
20             char *envp[] = {"HOME=/", "TERM=linux", "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
21                 NULL};
22             if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC > 0))
23             {
24                 printk(KERN_WARNING "netkiller:_unable_to_bring_network_back\n");
25             }
26             else
27             {
28                 printk(KERN_INFO "netkiller:_network_is_available_now\n");
29                 is_network_down = false;
30             }
31         }
32     }
33 }

```

## 3.5 Регистрация уведомителя для USB-устройств

В листинге 3.6 представлено объявление уведомителя и его функции-обработчика.

Листинг 3.6 – Уведомитель для USB-устройств

```
1 // Handler for event's notifier.
2 static int notify(struct notifier_block *self, unsigned long action, void *dev)
3 {
4     // Events, which our notifier react.
5     switch (action)
6     {
7         case USB_DEVICE_ADD:
8             usb_dev_insert(dev);
9             break;
10        case USB_DEVICE_REMOVE:
11            usb_dev_remove(dev);
12            break;
13        default:
14            break;
15    }
16
17    return 0;
18 }
19
20 // React on different notifies.
21 static struct notifier_block usb_notify = {
22     .notifier_call = notify,
23 };
```

В листинге 3.7 представлены регистрация и deregистрация уведомителя при загрузке и удалении модуля ядра соответственно.

Листинг 3.7 – Регистрация и deregистрация уведомителя

```
1 // Module init function.
2 static int __init netkiller_init(void)
3 {
4     usb_register_notify(&usb_notify);
5     printk(KERN_INFO "netkiller: module loaded\n");
6     return 0;
7 }
8
9 // Module exit function.
10 static void __exit netkiller_exit(void)
```

```

11 {
12     usb_unregister_notify(&usb_notify);
13     printk(KERN_INFO "netkiller: module unloaded\n");
14 }

```

## 3.6 Примеры работы разработанного ПО

На рисунках 3.1 — 3.3 представлены примеры работы разработанного модуля ядра.

```

[ 71.460298] netkiller: module loaded
[ 108.183874] netkiller: device connected with PID '15872' and VID '5118'
[ 108.183878] netkiller: no not allowed devices connected, skipping network killing
[ 133.276207] netkiller: device disconnected with PID '15872' and VID '5118'
parallels@ubuntu-linux-20-04-desktop:~/Desktop/Parallels Shared Folders/Home/Study/netkiller/src$ ping -c 3 google.com
PING google.com (64.233.162.113) 56(84) bytes of data.
64 bytes from li-in-f113.1e100.net (64.233.162.113): icmp_seq=1 ttl=128 time=18.3 ms
64 bytes from li-in-f113.1e100.net (64.233.162.113): icmp_seq=2 ttl=128 time=28.4 ms
64 bytes from li-in-f113.1e100.net (64.233.162.113): icmp_seq=3 ttl=128 time=26.5 ms

--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 18.336/24.431/28.416/4.377 ms
parallels@ubuntu-linux-20-04-desktop:~/Desktop/Parallels Shared Folders/Home/Study/netkiller/src$

```

Рисунок 3.1 – Пример подключения доверенного устройства и проверка сети

```

[ 252.877709] netkiller: device connected with PID '21879' and VID '1921'
[ 252.877710] netkiller: 1 not allowed devices connected, killing network
[ 252.877793] netkiller: network is killed
parallels@ubuntu-linux-20-04-desktop:~/Desktop/Parallels Shared Folders/Home/Study/netkiller/src$ ping -c 3 google.com
ping: google.com: Temporary failure in name resolution
parallels@ubuntu-linux-20-04-desktop:~/Desktop/Parallels Shared Folders/Home/Study/netkiller/src$

```

Рисунок 3.2 – Пример подключения недоверенного устройства и проверка сети

```

[ 284.569791] netkiller: device disconnected with PID '21879' and VID '1921'
[ 284.569797] netkiller: all not allowed devices are disconnected, bringing network back
[ 284.572162] netkiller: network is available now
parallels@ubuntu-linux-20-04-desktop:~/Desktop/Parallels Shared Folders/Home/Study/netkiller/src$ ping -c 3 google.com
PING google.com (64.233.162.102) 56(84) bytes of data.
64 bytes from li-in-f102.1e100.net (64.233.162.102): icmp_seq=1 ttl=128 time=19.5 ms
64 bytes from li-in-f102.1e100.net (64.233.162.102): icmp_seq=2 ttl=128 time=19.7 ms
64 bytes from li-in-f102.1e100.net (64.233.162.102): icmp_seq=3 ttl=128 time=18.3 ms

--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2009ms
rtt min/avg/max/mdev = 18.290/19.156/19.712/0.620 ms
parallels@ubuntu-linux-20-04-desktop:~/Desktop/Parallels Shared Folders/Home/Study/netkiller/src$

```

Рисунок 3.3 – Пример отключения недоверенного устройства и проверка сети

# Заключение

В процессе выполнения курсовой работы по операционным системам был разработан загружаемый модуль ядра Linux для отключения сетевого оборудования системы при подключении USB-устройства.

Проанализированы методы обработки событий, возникающих при взаимодействии с USB-устройствами.

Изучены структуры и функции ядра, которые предоставляют информацию о USB-устройствах, механизмы для обработки событий USB-устройств.

Разработан алгоритм отключения сетевого оборудования системы при подключении USB-устройства.

Реализовано и протестировано программное обеспечение, реализующее разработанный алгоритм. Программное обеспечение полностью удовлетворяет техническому заданию.

# Литература

- [1] Juice Jacking: Security Issues and Improvements in USB Technology / Debabrata Singh, Anil Kumar Biswal, Debabrata Samanta [и др.] // Sustainability. 2022. 01. Т. 14.
- [2] usbmon — The Linux Kernel documentation [Электронный ресурс]. Режим доступа: <https://www.kernel.org/doc/html/latest/usb/usbmon.html> (дата обращения: 16.02.2022).
- [3] udevadm(8) - Linux manual page [Электронный ресурс]. Режим доступа: <https://man7.org/linux/man-pages/man8/udevadm.8.html> (дата обращения: 16.02.2022).
- [4] libudev [Электронный ресурс]. Режим доступа: <https://www.freedesktop.org/software/systemd/man/libudev.html> (дата обращения: 16.02.2022).
- [5] Notification Chains in Linux Kernel [Электронный ресурс]. Режим доступа: <https://0xax.gitbooks.io/linux-insides/content/Concepts/linux-cpu-4.html> (дата обращения: 15.02.2022).
- [6] notifier.h - include/linux/notifier.h - Linux source code (v5.13) - Bootlin [Электронный ресурс]. Режим доступа: <https://elixir.bootlin.com/linux/v5.13/source/include/linux/notifier.h#L54> (дата обращения: 15.02.2022).
- [7] usb.h - include/linux/usb.h - Linux source code (v5.13) - Bootlin [Электронный ресурс]. Режим доступа: <https://elixir.bootlin.com/linux/v5.13/source/include/linux/usb.h#L632> (дата обращения: 15.02.2022).
- [8] Document Library | USB-IF [Электронный ресурс]. Режим доступа: [https://www.usb.org/documents?search=&type%5B0%5D=55&items\\_per\\_page=50](https://www.usb.org/documents?search=&type%5B0%5D=55&items_per_page=50) (дата обращения: 15.02.2022).
- [9] mod\_devicetable.h - include/linux/mod\_devicetable.h - Linux source code (v5.13) - Bootlin [Электронный ресурс]. Режим доступа:

[https://elixir.bootlin.com/linux/v5.13/source/include/linux/mod\\_devicetable.h#L121](https://elixir.bootlin.com/linux/v5.13/source/include/linux/mod_devicetable.h#L121) (дата обращения: 15.02.2022).

- [10] Invoking user-space applications from the kernel – IBM Developer [Электронный ресурс]. Режим доступа: <https://developer.ibm.com/articles/l-user-space-apps/> (дата обращения: 15.02.2022).
- [11] umh.h - include/linux/umh.h - Linux source code (v5.13) - Bootlin [Электронный ресурс]. Режим доступа: <https://elixir.bootlin.com/linux/v5.13/source/include/linux/umh.h#L42> (дата обращения: 15.02.2022).
- [12] C99 standard note [Электронный ресурс]. Режим доступа: <http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf> (дата обращения: 15.02.2022).
- [13] GCC, the GNU Compiler Collection [Электронный ресурс]. Режим доступа: <https://gcc.gnu.org/> (дата обращения: 15.02.2022).
- [14] Visual Studio Code - Code Editing. Redefined [Электронный ресурс]. Режим доступа: <https://code.visualstudio.com> (дата обращения: 16.02.2022).

# ПРИЛОЖЕНИЕ А

## Листинг 3.8 – Исходный код программы

```
1 #include <linux/module.h>
2 #include <linux/usb.h>
3
4 MODULE_LICENSE("GPL");
5 MODULE_AUTHOR("Sergey_Kononenko");
6 MODULE_VERSION("1.0");
7
8 // Wrapper for usb_device_id with added list_head field to track devices.
9 typedef struct int_usb_device
10 {
11     struct usb_device_id dev_id;
12     struct list_head list_node;
13 } int_usb_device_t;
14
15 bool is_network_down = false;
16 struct usb_device_id allowed_devs[] = {
17     {USB_DEVICE(0x13fe, 0x3e00)},
18 };
19
20 // Declare and init the head node of the linked list.
21 LIST_HEAD(connected_devices);
22
23 // Match device with device id.
24 static bool is_dev_matched(struct usb_device *dev, const struct usb_device_id *dev_id)
25 {
26     // Check idVendor and idProduct, which are used.
27     if (dev_id->idVendor != dev->descriptor.idVendor || dev_id->idProduct !=
28         dev->descriptor.idProduct)
29     {
30         return false;
31     }
32     return true;
33 }
34
35 // Match device id with device id.
36 static bool is_dev_id_matched(struct usb_device_id *new_dev_id, const struct
37     usb_device_id *dev_id)
38 {
39     // Check idVendor and idProduct, which are used.
40     if (dev_id->idVendor != new_dev_id->idVendor || dev_id->idProduct !=
41         new_dev_id->idProduct)
```

```

42     }
43
44     return true;
45 }
46
47 // Check if device is in allowed devices list.
48 static bool *is_dev_allowed(struct usb_device_id *dev)
49 {
50     unsigned long allowed_devs_len = sizeof(allowed_devs) / sizeof(struct usb_device_id);
51
52     int i;
53     for (i = 0; i < allowed_devs_len; i++)
54     {
55         if (is_dev_id_matched(dev, &allowed_devs[i]))
56         {
57             return true;
58         }
59     }
60
61     return false;
62 }
63
64 // Check if changed device is acknowledged.
65 static int count_not_acked_devs(void)
66 {
67     int_usb_device_t *temp;
68     int count = 0;
69
70     list_for_each_entry(temp, &connected_devices, list_node)
71     {
72         if (!is_dev_allowed(&temp->dev_id))
73         {
74             count++;
75         }
76     }
77
78     return count;
79 }
80
81 // Add connected device to list of tracked devices.
82 static void add_int_usb_dev(struct usb_device *dev)
83 {
84     int_usb_device_t *new_usb_device = (int_usb_device_t
85         *)kmalloc(sizeof(int_usb_device_t), GFP_KERNEL);
86     struct usb_device_id new_id = {USB_DEVICE(dev->descriptor.idVendor,
87         dev->descriptor.idProduct)};
88     new_usb_device->dev_id = new_id;
89     list_add_tail(&new_usb_device->list_node, &connected_devices);

```



```

88 }
89
90 // Delete device from list of tracked devices.
91 static void delete_int_usb_dev(struct usb_device *dev)
92 {
93     int_usb_device_t *device, *temp;
94     list_for_each_entry_safe(device, temp, &connected_devices, list_node)
95     {
96         if (is_dev_matched(dev, &device->dev_id))
97         {
98             list_del(&device->list_node);
99             kfree(device);
100         }
101     }
102 }
103
104 // Handler for USB insertion.
105 static void usb_dev_insert(struct usb_device *dev)
106 {
107     printk(KERN_INFO "netkiller:_device_connected_with_PID_%d_and_VID_%d\n",
108             dev->descriptor.idProduct, dev->descriptor.idVendor);
109     add_int_usb_dev(dev);
110     int not_acked_devs = count_not_acked_devs();
111
112     if (!not_acked_devs)
113     {
114         printk(KERN_INFO "netkiller:_no_not_allowed_devices_connected,_skipping_network_killing\n");
115     }
116     else
117     {
118         printk(KERN_INFO "netkiller:_%d_not_allowed_devices_connected,_killing_network\n", not_acked_devs);
119         if (!is_network_down)
120         {
121             char *argv[] = {"/sbin/modprobe", "-r", "virtio_net", NULL};
122             char *envp[] = {"HOME=/", "TERM=linux", "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
123                             NULL};
124             if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC > 0))
125             {
126                 printk(KERN_WARNING "netkiller:_unable_to_kill_network\n");
127             }
128             else
129             {
130                 printk(KERN_INFO "netkiller:_network_is_killed\n");
131                 is_network_down = true;
132             }
133         }
134     }
135 }

```

```

133     }
134 }
135
136 // Handler for USB removal.
137 static void usb_dev_remove(struct usb_device *dev)
138 {
139     printk(KERN_INFO "netkiller: device disconnected with PID '%d' and VID '%d'\n",
140         dev->descriptor.idProduct, dev->descriptor.idVendor);
141     delete_int_usb_dev(dev);
142     int not_acked_devs = count_not_acked_devs();
143
144     if (not_acked_devs)
145     {
146         printk(KERN_INFO "netkiller: %d not allowed devices connected, nothing to do\n",
147             not_acked_devs);
148     }
149     else
150     {
151         if (is_network_down)
152         {
153             printk(KERN_INFO "netkiller: all not allowed devices are disconnected, bringing network back\n");
154             char *argv[] = {"/sbin/modprobe", "virtio_net", NULL};
155             char *envp[] = {"HOME=", "TERM=linux", "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
156                 NULL};
157             if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC > 0))
158             {
159                 printk(KERN_WARNING "netkiller: unable to bring network back\n");
160             }
161             else
162             {
163                 printk(KERN_INFO "netkiller: network is available now\n");
164                 is_network_down = false;
165             }
166         }
167     }
168 }
169
170 // Handler for event's notifier.
171 static int notify(struct notifier_block *self, unsigned long action, void *dev)
172 {
173     // Events, which our notifier react.
174     switch (action)
175     {
176     case USB_DEVICE_ADD:
177         usb_dev_insert(dev);
178         break;
179     case USB_DEVICE_REMOVE:

```

```

178         usb_dev_remove(dev);
179         break;
180     default:
181         break;
182 }
183
184     return 0;
185 }
186
187 // React on different notifies.
188 static struct notifier_block usb_notify = {
189     .notifier_call = notify,
190 };
191
192 // Module init function.
193 static int __init netkiller_init(void)
194 {
195     usb_register_notify(&usb_notify);
196     printk(KERN_INFO "netkiller: module loaded\n");
197     return 0;
198 }
199
200 // Module exit function.
201 static void __exit netkiller_exit(void)
202 {
203     usb_unregister_notify(&usb_notify);
204     printk(KERN_INFO "netkiller: module unloaded\n");
205 }
206
207 module_init(netkiller_init);
208 module_exit(netkiller_exit);

```