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 неизвестного 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 неизвестного 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 была произведена блокировка

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

Студент _____ С.Кононенко
(Подпись, дата) (И.О.Фамилия)

Оглавление

Введение

В настоящее время в мире остро стоит вопрос кибербезопасности. 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 недопустимого устройства можно отключать сетевые устройства системы. Таким образом исключается возможность передачи данных через интернет.

Цель работы — разработать загружаемый модуля ядра Linux для отключения сетевого оборудования системы при подключении неизвестного USB–устройства

1 Аналитическая часть

1.1 Постановка задачи

В соответствии с заданием необходимо разработать загружаемый модуля ядра Linux для отключения сетевого оборудования системы при подключении неизвестного USB–устройства. Для решения данной задачи необходимо:

- исследовать механизмы ядра для обработки событий подключения и отключения USB–устройств;
- исследовать структуры и функции ядра, предоставляющие информацию о подключенных USB–устройствах;
- спроектировать и реализовать загружаемый модуль ядра.

1.2 Уведомления в ядре Linux

1.2.1 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 используется в ядре; существуют цепочки для событий `hotplug` памяти, изменения политики частоты процессора, события `USB hotplug`, загрузки и выгрузки модулей, перезагрузки системы, изменения сетевых устройств [2].

В листинге 1.1 представлена структура `notifier_block` [3].

Листинг 1.1 – 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 1.2 представлена сигнатура функии `notifier_call`.

Листинг 1.2 – Тип `notifier_fn_t`

```
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`).

1.2.2 Уведомитель изменений на USB–портах

Для регистрации уведомителя для USB–портов используются функции регистрации и удаления уведомителя, представленные в листинге 1.3.

Листинг 1.3 – Уведомители на USB–портах

```
    /* Events from the usb core */
#define USB_DEVICE_ADD  0x0001
#define USB_DEVICE_REMOVE 0x0002
#define USB_BUS_ADD  0x0003
#define USB_BUS_REMOVE  0x0004
extern void usb_register_notify(struct notifier_block *nb);
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.

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 `/inlclude/linux/usb.h` [4].

Структура `usb_device` предствалена в листинге 1.4.

Листинг 1.4 – Структура `usb_device`

```
struct usb_device {
    int  devnum;
    char  devpath[16];
    u32  route;
    enum usb_device_state state;
    enum usb_device_speed speed;
    unsigned int  rx_lanes;
    unsigned int  tx_lanes;
    enum usb_ssp_rate ssp_rate;

    struct usb_tt *tt;
    int  ttport;

    unsigned int toggle[2];

    struct usb_device *parent;
    struct usb_bus *bus;
    struct usb_host_endpoint ep0;

    struct device dev;

    struct usb_device_descriptor descriptor;
    struct usb_host_bos *bos;
    struct usb_host_config *config;

    struct usb_host_config *actconfig;
    struct usb_host_endpoint *ep_in[16];
    struct usb_host_endpoint *ep_out[16];

    char **rawdescriptors;

    unsigned short bus_mA;
    u8 portnum;
    u8 level;
    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_besl_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 [5], одним из требовании 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 1.5 представлена структура дескриптора `usb_device_descriptor`, описанная в `/include/uapi/linux/usb/ch9.h`.

Листинг 1.5 – Структура `usb_device_descriptor`

```
/* USB_DT_DEVICE: Device descriptor */
struct usb_device_descriptor {
    __u8 bLength;
    __u8 bDescriptorType;

    __le16 bcdUSB;
    __u8 bDeviceClass;
    __u8 bDeviceSubClass;
    __u8 bDeviceProtocol;
    __u8 bMaxPacketSize0;
    __le16 idVendor;
    __le16 idProduct;
    __le16 bcdDevice;
    __u8 iManufacturer;
    __u8 iProduct;
    __u8 iSerialNumber;
    __u8 bNumConfigurations;
} __attribute__ ((packed));

#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` [?].

Структура `usb_device_id` представлена в листинге 1.6.

Листинг 1.6 – Структура `usb_device_id`

```
struct usb_device_id {
    /* which fields to match against? */
    __u16  match_flags;

    /* 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))));
27 };
```

1.4 Загружаемые модули ядра Linux

1.4.1 Особенности загружаемых модулей

Одной из особенностей ядра Linux является способность расширения функциональности во время работы, без необходимости компиляции ядра заново. Таким образом, существует возможность добавить (или убрать) функциональность в ядро когда система запущена и работает. Часть кода, которая может быть добавлена в ядро во время работы, называется модулем ядра. Ядро Linux предлагает поддержку большого числа классов модулей. Каждый модуль — это подготовленный объектный код, который может быть динамически подключен в работающее ядро, а позднее может быть выгружен из ядра.

Каждый модуль ядра сам регистрирует себя для того, чтобы обслуживать в будущем запросы, и его функция инициализации (`module_init`) немедленно прекращается. Задача инициализации модуля заключается в

подготовке функций модуля для последующего вызова. Функция выхода модуля вызывается перед выгрузкой модуля из ядра. Функция выхода (`module_exit`) должна отменить все изменения, сделанные функций инициализации, освободить захваченные в процессе работы модуля ресурсы.

Возможность выгрузить модуль помогает сократить время разработки — нет необходимости перезагрузки системы при последовательном тестировании новых версий разрабатываемого модуля ядра.

Модуль связан только с ядром и может вызывать только те функции, которые экспортированы ядром.

1.4.2 Пространства ядра и пользователя

Приложения работают в пользовательском пространстве, а ядро и его модули — в пространстве ядра. Такое разделение пространств — базовая концепция теории операционных систем.

Ролью операционной системы является обеспечение программ надёжным доступом к аппаратной части компьютера. Операционная система должна обеспечивать независимую работу программ и защиту от несанкционированного доступа к ресурсам. Решение этих задач становится возможным только в том случае, если процессор обеспечивает защиту системного программного обеспечения от прикладных программ.

Выбранный подход заключается в обеспечении разных режимов работы (или уровней) в самом центральном процессоре. Уровни играют разные роли и некоторые операции на более низких уровнях не допускаются; программный код может переключить один уровень на другой только ограниченным числом способов. Все современные процессоры имеют не менее двух уровней защиты, а некоторые, например семейство процессоров x86, имеют больше уровней; когда существует несколько уровней, используется самый высокий и самый низкий уровень защиты.

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 оборудованию 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.

1.4.3 Вызов приложений пространства пользователя из пространства ядра

Для вызова приложений 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 [6].

В листинге 1.7 представлена структура процесса, использующегося в `usermode-helper API` и сигнатура функции вызова [7].

Листинг 1.7 – `usermode-helper API`

```
#define UMH_NO_WAIT 0 /* don't wait at all */
#define UMH_WAIT_EXEC 1 /* wait for the exec, but not the process */
#define UMH_WAIT_PROC 2 /* wait for the process to complete */
#define UMH_KILLABLE 4 /* wait for EXEC/PROC killable */

struct subprocess_info {
    struct work_struct work;
    struct completion *complete;
    const char *path;
    char **argv;
    char **envp;
    int wait;
    int retval;
    int (*init)(struct subprocess_info *info, struct cred *new);
    void (*cleanup)(struct subprocess_info *info);
    void *data;
} __randomize_layout;
```

```
18
19 extern int call_usermodehelper(const char *path, char **argv, char **envp, int wait);
```

Вывод

В данном разделе были рассмотрены механизмы ядра для обработки событий подключения и отключения USB–устройств — уведомители (`notifiers`) —, структуры и функции ядра для работы с ними, а также особенности загружаемых модулей ядра и понятия пространств ядра и пользователя, способ вызова приложений пространства пользователя из пространства ядра.

2 Конструкторская часть

2.1 Архитектура приложения

В состав разрабатываемого программного обеспечения входит один загружаемый модуль ядра, который отслеживает подключенные USB–устройства и программно отключает сетевые устройства при наличии активного недопустимого устройства. Список допустимых устройств задается в исходном коде модуля.

2.2 Отслеживание событий

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

2.3 Хранение подключенных устройств

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

2.4 Алгоритм работы обработчика событий

На рисунках 2.1 и 2.2 представлены схемы работы обработчика события и алгоритма работы с сетевым драйвером соответственно.

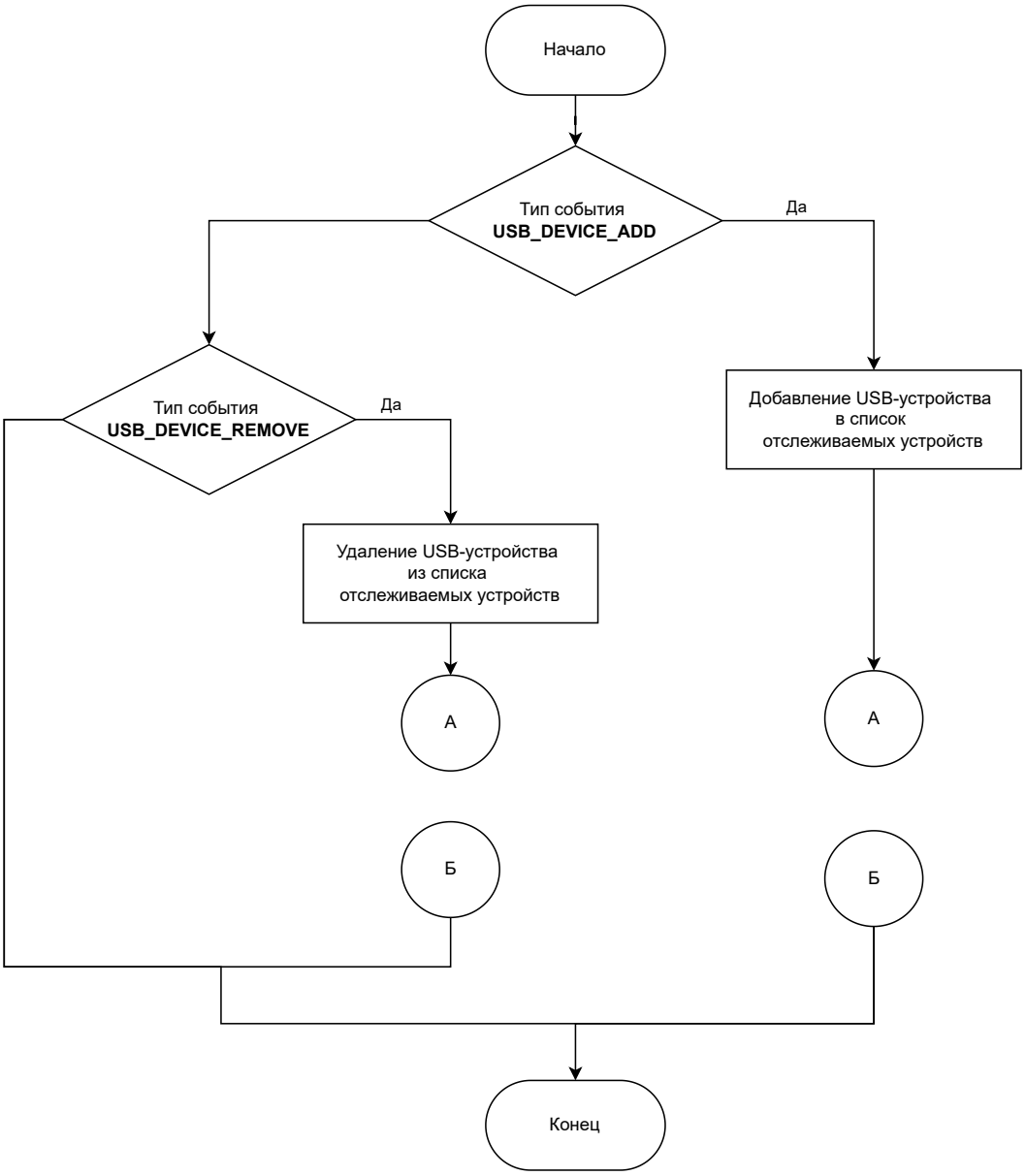

Рисунок 2.1 – Схема работы обработчика события

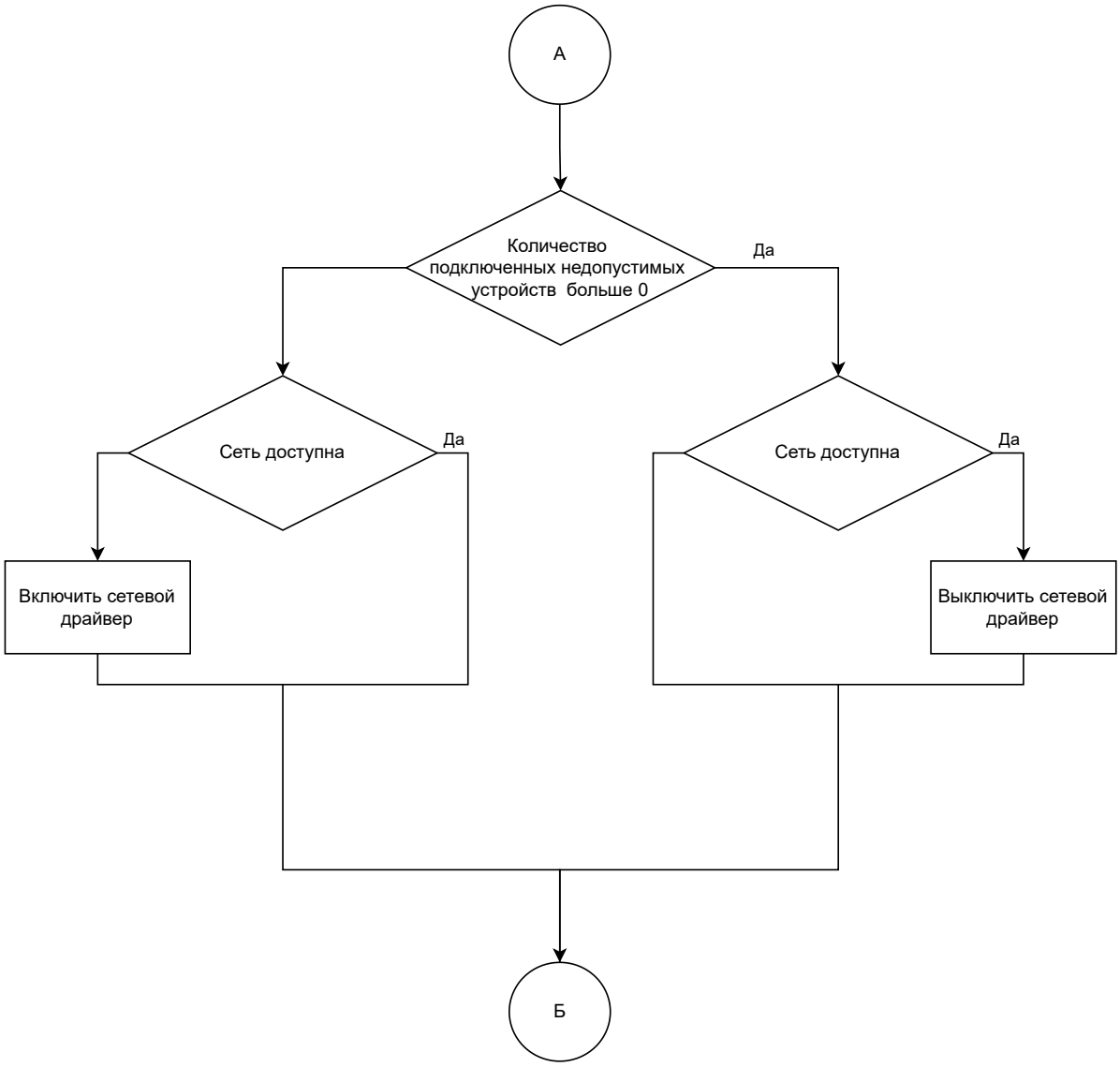

Рисунок 2.2 – Схема работы с сетевым драйвером

Вывод

В данном разделе была представлена архитектура разрабатываемого приложения, рассмотрены ключевые моменты работы и представлены схемы алгоритмов.

3 Технологическая часть

3.1 Выбор языка программирования

Разработанный модуль ядра написан на языке программирования `C` [8]. 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` [9].

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`

```
typedef struct int_usb_device
{
    struct usb_device_id dev_id;
    struct list_head list_node;
} 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

```
// Add connected device to list of tracked devices.
static void add_int_usb_dev(struct usb_device *dev)
{
    int_usb_device_t *new_usb_device = (int_usb_device_t
        *)kmalloc(sizeof(int_usb_device_t), GFP_KERNEL);
    struct usb_device_id new_id = {USB_DEVICE(dev->descriptor.idVendor,
        dev->descriptor.idProduct)};
    new_usb_device->dev_id = new_id;
```

```
    list_add_tail(&new_usb_device->list_node, &connected_devices);
}

// Delete device from list of tracked devices.
static void delete_int_usb_dev(struct usb_device *dev)
{
    int_usb_device_t *device, *temp;
    list_for_each_entry_safe(device, temp, &connected_devices, list_node)
    {
        if (is_dev_matched(dev, &device->dev_id))
        {
            list_del(&device->list_node);
            kfree(device);
        }
    }
}
```

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

```
struct usb_device_id allowed_devs[] = {
    {USB_DEVICE(0x13fe, 0x3e00)},
};

// Match device with device id.
static bool is_dev_matched(struct usb_device *dev, const struct usb_device_id *dev_id)
{
    // Check idVendor and idProduct, which are used.
    if (dev_id->idVendor != dev->descriptor.idVendor || dev_id->idProduct !=
        dev->descriptor.idProduct)
    {
        return false;
    }

    return true;
}

```

```
// Match device id with device id.
static bool is_dev_id_matched(struct usb_device_id *new_dev_id, const struct
    usb_device_id *dev_id)
{
    // Check idVendor and idProduct, which are used.
    if (dev_id->idVendor != new_dev_id->idVendor || dev_id->idProduct !=
        new_dev_id->idProduct)
    {
        return false;
    }

    return true;
}

// Check if device is in allowed devices list.
static bool *is_dev_allowed(struct usb_device_id *dev)
{
    unsigned long allowed_devs_len = sizeof(allowed_devs) / sizeof(struct usb_device_id);

    int i;
    for (i = 0; i < allowed_devs_len; i++)
    {
        if (is_dev_id_matched(dev, &allowed_devs[i]))
        {
            return true;
        }
    }

    return false;
}

// Check if changed device is acknowledged.
static int count_not_acked_devs(void)
{
    int_usb_device_t *temp;
    int count = 0;

    list_for_each_entry(temp, &connected_devices, list_node)
    {
        if (!is_dev_allowed(&temp->dev_id))
        {
            count++;
        }
    }

    return count;
}
```

3.4 Обработка событий подключения и отключения 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

```
static void usb_dev_insert(struct usb_device *dev)
{
    printk(KERN_INFO "netkiller: device connected with PID '%d' and VID '%d'\n",
           dev->descriptor.idProduct, dev->descriptor.idVendor);
    add_int_usb_dev(dev);
    int not_acked_devs = count_not_acked_devs();

    if (!not_acked_devs)
    {
        printk(KERN_INFO "netkiller: no not allowed devices connected, skipping network
            killing\n");
    }
    else
    {
        printk(KERN_INFO "netkiller: %d not allowed devices connected, killing
            network\n", not_acked_devs);
        if (!is_network_down)
        {
            char *argv[] = {"/sbin/modprobe", "-r", "virtio_net", NULL};
            char *envp[] = {"HOME=/", "TERM=linux", "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
                NULL};
            if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC > 0))
            {
                printk(KERN_WARNING "netkiller: unable to kill network\n");
            }
            else
            {
                printk(KERN_INFO "netkiller: network is killed\n");
                is_network_down = true;
            }
        }
    }
}
```

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

```
static void usb_dev_remove(struct usb_device *dev)
{
    printk(KERN_INFO "netkiller: device disconnected with PID '%d' and VID '%d'\n",
           dev->descriptor.idProduct, dev->descriptor.idVendor);
    delete_int_usb_dev(dev);
    int not_acked_devs = count_not_acked_devs();

    if (not_acked_devs)
    {
        printk(KERN_INFO "netkiller: %d not allowed devices connected, nothing to do\n",
            not_acked_devs);
    }
    else
    {
        if (is_network_down)
        {
            printk(KERN_INFO "netkiller: all not allowed devices are disconnected, 
                bringing network back\n");
            char *argv[] = {"/sbin/modprobe", "virtio_net", NULL};
            char *envp[] = {"HOME=/", "TERM=linux", "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
                NULL};
            if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC > 0))
            {
                printk(KERN_WARNING "netkiller: unable to bring network back\n");
            }
            else
            {
                printk(KERN_INFO "netkiller: network is available now\n");
                is_network_down = false;
            }
        }
    }
}
```

3.5 Регистрация уведомителя для USB–устройств

В листинге 3.7 представлено объявление уведомителя и его функции–обработчика.

Листинг 3.6 – Уведомитель для USB–устройств

```
// Handler for event's notifier.
static int notify(struct notifier_block *self, unsigned long action, void *dev)
{
    // Events, which our notifier react.
    switch (action)
    {
    case USB_DEVICE_ADD:
        usb_dev_insert(dev);
        break;
    case USB_DEVICE_REMOVE:
        usb_dev_remove(dev);
        break;
    default:
        break;
    }

    return 0;
}

// React on different notifies.
static struct notifier_block usb_notify = {
    .notifier_call = notify,
};
```

В листинге ?? представлены регистрация и дерегистрация уведомителя при загрузке и удалении модуля ядра соответственно.

Листинг 3.7 – Регистрация и дерегистрация уведомителя

```
// Module init function.
static int __init netkiller_init(void)
{
    usb_register_notify(&usb_notify);
    printk(KERN_INFO "netkiller: module loaded\n");
    return 0;
}

// Module exit function.
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

```
[  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

```
[  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

Вывод

В данном разделе был обоснован выбор языка программирования, рассмотрены листинги реализованного программного обеспечения и приведены результаты работы ПО.

Заключение

В ходе проделанной работы был разработан загружаемый модуль ядра Linux для отключения сетевого оборудования системы при подключении неизвестного USB–устройства.

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

На основе полученных знаний и проанализированных технологий реализован загружаемый модуль ядра.

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] 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).

[3] 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).

[4] 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).

[5] Document Library | USB-IF [Электронный ресурс]. Режим доступа: https://www.usb.org/documents?search=&type%5B0%5D=55&items_per_page=50 (дата обращения: 15.02.2022).

[6] 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).

[7] 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).

[8] C99 standard note [Электронный ресурс]. Режим доступа: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf (дата обращения: 15.02.2022).

[9] GCC, the GNU Compiler Collection [Электронный ресурс]. Режим доступа: https://gcc.gnu.org/ (дата обращения: 15.02.2022).

ПРИЛОЖЕНИЕ А

Листинг 3.8 – Исходный код программы

```c
#include <linux/module.h>
#include <linux/usb.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Sergey␣Kononenko");
MODULE_VERSION("1.0");

// Wrapper for usb_device_id with added list_head field to track devices.
typedef struct int_usb_device
{
    struct usb_device_id dev_id;
    struct list_head list_node;
} int_usb_device_t;

bool is_network_down = false;
struct usb_device_id allowed_devs[] = {
    {USB_DEVICE(0x13fe, 0x3e00)},
};

// Declare and init the head node of the linked list.
LIST_HEAD(connected_devices);

// Match device with device id.
static bool is_dev_matched(struct usb_device *dev, const struct usb_device_id *dev_id)
{
    // Check idVendor and idProduct, which are used.
    if (dev_id->idVendor != dev->descriptor.idVendor || dev_id->idProduct !=
        dev->descriptor.idProduct)
    {
        return false;
    }

    return true;
}

// Match device id with device id.
static bool is_dev_id_matched(struct usb_device_id *new_dev_id, const struct
    usb_device_id *dev_id)
{
    // Check idVendor and idProduct, which are used.
    if (dev_id->idVendor != new_dev_id->idVendor || dev_id->idProduct !=
        new_dev_id->idProduct)
    {
        return false;
```

```
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
            *)kmalloc(sizeof(int_usb_device_t), GFP_KERNEL);
85      struct usb_device_id new_id = {USB_DEVICE(dev->descriptor.idVendor,
            dev->descriptor.idProduct)};
86      new_usb_device->dev_id = new_id;
87      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",
                   NULL};
123            if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC > 0))
124            {
125                printk(KERN_WARNING "netkiller:␣unable␣to␣kill␣network\n");
126            }
127            else
128            {
129                printk(KERN_INFO "netkiller:␣network␣is␣killed\n");
130                is_network_down = true;
131            }
132        }
```

```
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",
              not_acked_devs);
147     }
148     else
149     {
150         if (is_network_down)
151         {
152             printk(KERN_INFO "netkiller: all not allowed devices are disconnected,
                  bringing network back\n");
153             char *argv[] = {"/sbin/modprobe", "virtio_net", NULL};
154             char *envp[] = {"HOME=/", "TERM=linux", "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
                  NULL};
155             if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC > 0))
156             {
157                 printk(KERN_WARNING "netkiller: unable to bring network back\n");
158             }
159             else
160             {
161                 printk(KERN_INFO "netkiller: network is available now\n");
162                 is_network_down = false;
163             }
164         }
165     }
166 }
167
168 // Handler for event's notifier.
169 static int notify(struct notifier_block *self, unsigned long action, void *dev)
170 {
171     // Events, which our notifier react.
172     switch (action)
173     {
174     case USB_DEVICE_ADD:
175         usb_dev_insert(dev);
176         break;
177     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);
```