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:

«Загружаемый модуль ядра ОС Linux для отключения системы от сети при подключении незарегистрированного USB–устройства»

Студент ИУ7-73Б (Группа) _____ (Подпись, дата) Миронов Г. А. (И. О. Фамилия)

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

2022 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одним из способов получения доступа к конфидециальной инфрмации являестся подключение через USB–устройство, на котором находится вредоносное ПО, задачей которого является передача конфидециальной информации по сети третьим лицам [1].

Для устранения такой возможности предлагается создавать список доверенных USB–устройств и блокировать доступ к сети при подкючении неизвестного устройства [2].

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

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать загружаемый модуля ядра ОС 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 подходы к определению возникновения событий от USB–устройств и выбран наиболее подходящий для реализации в данной работе

1.2.1 usbmon

`usbmon` [3] — это средство ядра Linux, которое используется для сбора информации о событиях, произошедших на устройствах ввода–вывода, подключенных посредством USB.

`usbmon` предоставляет информацию о запросах, сделанных драйверами устройств к драйверам хост–контроллера (HCD). В случае, когда драйвера

хост–контроллера неисправны, данные, предоставленные usbmon, могут не соответствовать действительным переданным данным.

В настоящее время реализованы два программных интерфейса для взаимодействия с usbmon:

— текстовый — данный интерфейс устарел, но сохраняется для совместимости;

— бинарный — доступен через символьное устройство в пространстве имен /dev.

К особенностям usbmon относятся:

— возможность просматривать собранную информацию через специальное ПО (например, Wireshark [4]);

— возможность отслеживать события на одном порте USB или на всех сразу;

— отсутствие возможности вызова обработчика при возникновении определенного события.

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

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

Листинг 1.1 – Структура usbmon_packet. Часть 1

```
struct usbmon_packet {
    u64 id;                 //  0: URB ID - from submission to
      callback
    unsigned char type;     //  8: Same as text; extensible.
    unsigned char xfer_type; //    ISO (0), Intr, Control, Bulk (3)
    unsigned char epnum;    //     Endpoint number and transfer
      direction
    unsigned char devnum;   //     Device address
    u16 busnum;             // 12: Bus number
    char flag_setup;        // 14: Same as text
    char flag_data;         // 15: Same as text; Binary zero is OK.
```

Листинг 1.2 – Структура usbmon_packet. Часть 2

```
    s64 ts_sec;             // 16: gettimeofday
    s32 ts_usec;            // 24: gettimeofday
    int status;             // 28:
    unsigned int length;    // 32: Length of data (submitted or
       actual)
    unsigned int len_cap;   // 36: Delivered length
    union {                 // 40:
        unsigned char setup[SETUP_LEN]; // Only for Control S-type
        struct iso_rec {                // Only for ISO
            int error_count;
            int numdesc;
        } iso;
    } s;
    int interval;           // 48: Only for Interrupt and ISO
    int start_frame;        // 52: For ISO
    unsigned int xfer_flags; // 56: copy of URB's transfer_flags
    unsigned int ndesc;     // 60: Actual number of ISO descriptors
};                          // 64 total length
```

1.2.2 udevadm

udevadm [5] — инструмент для управления устройствами udev. Структура udev описана в библиотеке libudev [6], 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.

Особенности udevadm:

— возможность привязки своего обработчика к событию;

— невозможность использования интерфейса в ядре Linux;

В листинге 1.3 представлен пример правила обработки событий, задаваемого с помощью udevadm.

Листинг 1.3 – Правила udevadm

```
/* rules file */
SUBSYSTEM=="usb", ACTION=="add", ENV{DEVTYPE}=="usb_device",  RUN+=
   "/bin/device_added.sh"
SUBSYSTEM=="usb", ACTION=="remove", ENV{DEVTYPE}=="usb_device", RUN
   +="/bin/device_removed.sh"

/* device_added.sh */
#!/bin/bash
echo "USB device added at $(date)" >>/tmp/scripts.log

/* device_removed.sh */
#!/bin/bash
echo "USB device removed  at $(date)" >>/tmp/scripts.log
```

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.

В настоящее время цепочки уведомлений 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 [7].

В листинге 1.4 представлена структура `notifier_block` [8].

Листинг 1.4 – Структура `notifier_block`

```
struct notifier_block {
    notifier_fn_t notifier_call;
    struct notifier_block __rcu *next;
    int priority;
};
```

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 1.5 представлена сигнатура функии `notifier_call`.

Листинг 1.5 – Структура `notifier_fn_t`

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

/* 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);
```

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 1.6.

Листинг 1.6 – Уведомители на USB–портах

```
#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. Действие USB_DEVICE_ADD означает подключение нового устройства, а USB_DEVICE_REMOVE — удаление, соответственно.

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 `/inlclude/linux/usb.h` [9]. Данная структура представлена в листинге 1.7.

Листинг 1.7 – Структура usb_device

```
struct usb_device {
    ...
    struct usb_device_descriptor descriptor;
    ...
    /* static strings from the device */
    char *product;
    char *manufacturer;
    char *serial;
    ...
};
```

Каждое USB–устройство должно соответствовать спецификации USB–IF [10], одним из требований которой является наличие идентификатора поставщика (`Vendor ID (VID)`) и идентификатора продукта (`Product ID (PID)`).

Эти данные присутствуют в поле `descriptor` структуры `usb_device`. Структура дескриптора `usb_device_descriptor`, описанная в `/include/uapi/linux/usb/ch9.h`, представлена в листинге 1.8.

Листинг 1.8 – Структура usb_device_descriptor

```
struct usb_device_descriptor {
    __le16 idVendor;
    __le16 idProduct;
    ...
} __attribute__ ((packed));
```

1.3.2 Структура usb_device_id

При подключении USB–устройства к компьютеру, оно идентифицируется и идентификационная информация записывается в структуру `usb_device_id` [11]. Данная структура представлена в листинге **??**.

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 [12].

В листинге 1.9 представлена структура процесса, использующегося в

usermode-helper API и сигнатура функции вызова [13].

Листинг 1.9 – 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;

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

2 Конструкторский раздел

2.1 Последовательность преобразований

На рисунках 2.1 и 2.2 представлена последовательность преобразований.

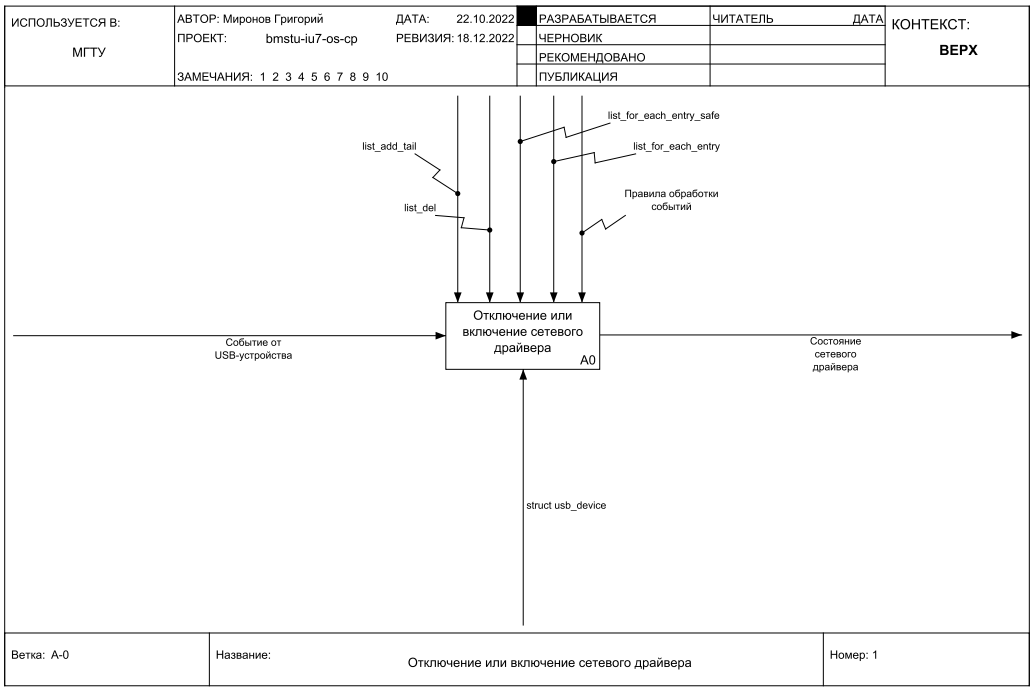

Рисунок 2.1 – Нулевой уровень преобразований

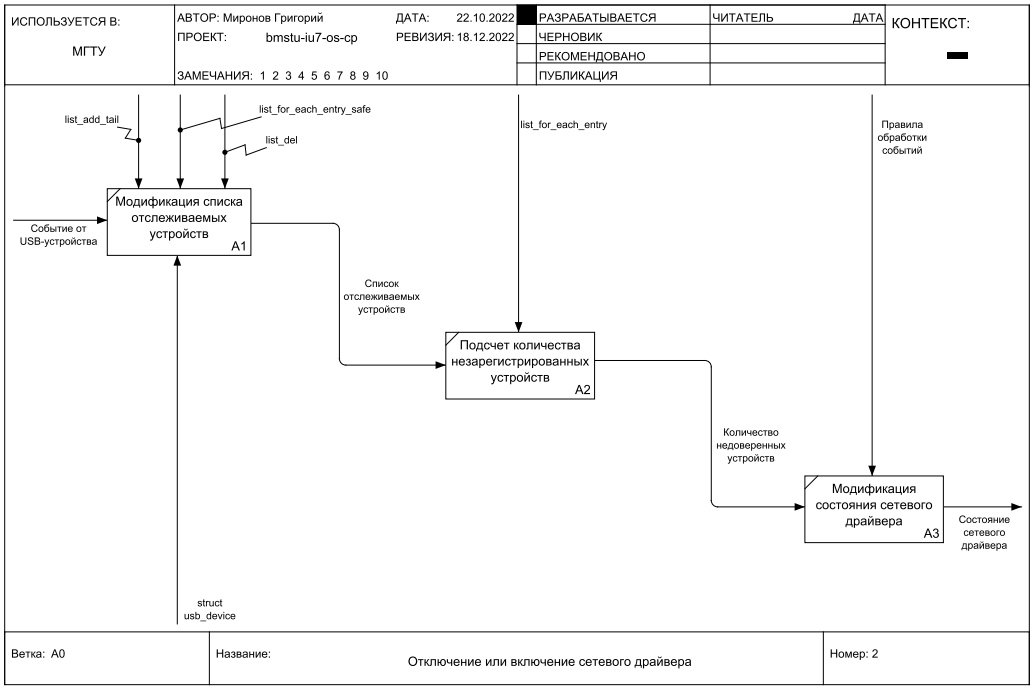

Рисунок 2.2 – Первый уровень преобразований

2.2 Загружаемый модуль ядра

Для отслеживания событий подключения и отключения устройств, а так же отслеживания событий ввода с клавиатуры, в модуле ядра размещаются соответствующие уведомители, которые будут зарегистрированы при загрузке модуля и удалены при его выгрузке.

Схема алгоритма загружаемого модуля представлена на рисунке 2.3. Выполняется регистрация уведомителей для обработки событий от USB и клавиатуры.

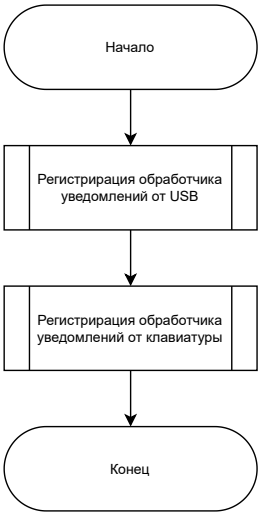

Рисунок 2.3 – Схема алгоритма загружаемого модуля

2.3 Обработчик событий от USB

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

На рисунке 2.4 представлена схема алгоритма работы обработчика события подключения USB–устройства.

В случае подключения USB–устройства производится проверка зарегистрированности данного устройства как доверенного. Если устройство является незарегистрированным, производится отключение сети.

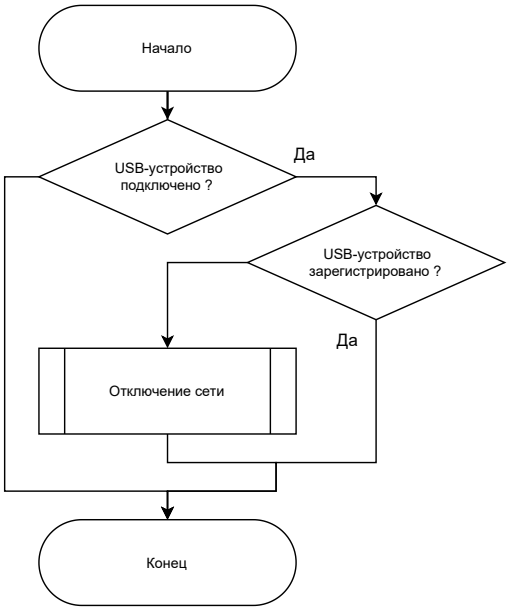

Рисунок 2.4 – Схема алгоритма работы обработчика события подключения USB–устройства

На рисунке 2.5 представлена схема алгоритма работы обработчика события удаления USB–устройства.

В случае удаления USB–устройства производится проверка зарегистрированности подключенных устройств как доверенных. Если все устройства являются зарегистрированными, производится подключение сети.

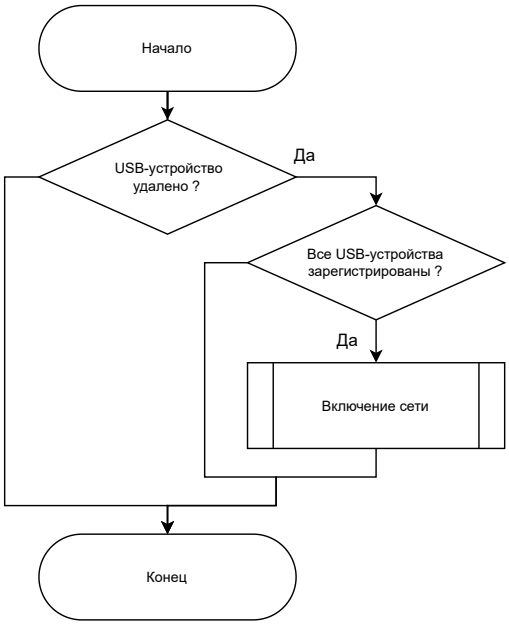

Рисунок 2.5 – Схема алгоритма работы обработчика события удаления USB–устройства

2.4 Обработчик событий от клавиатуры

На рисунке 2.6 представлен алгоритм работы обработчика событий от клавиатуры.

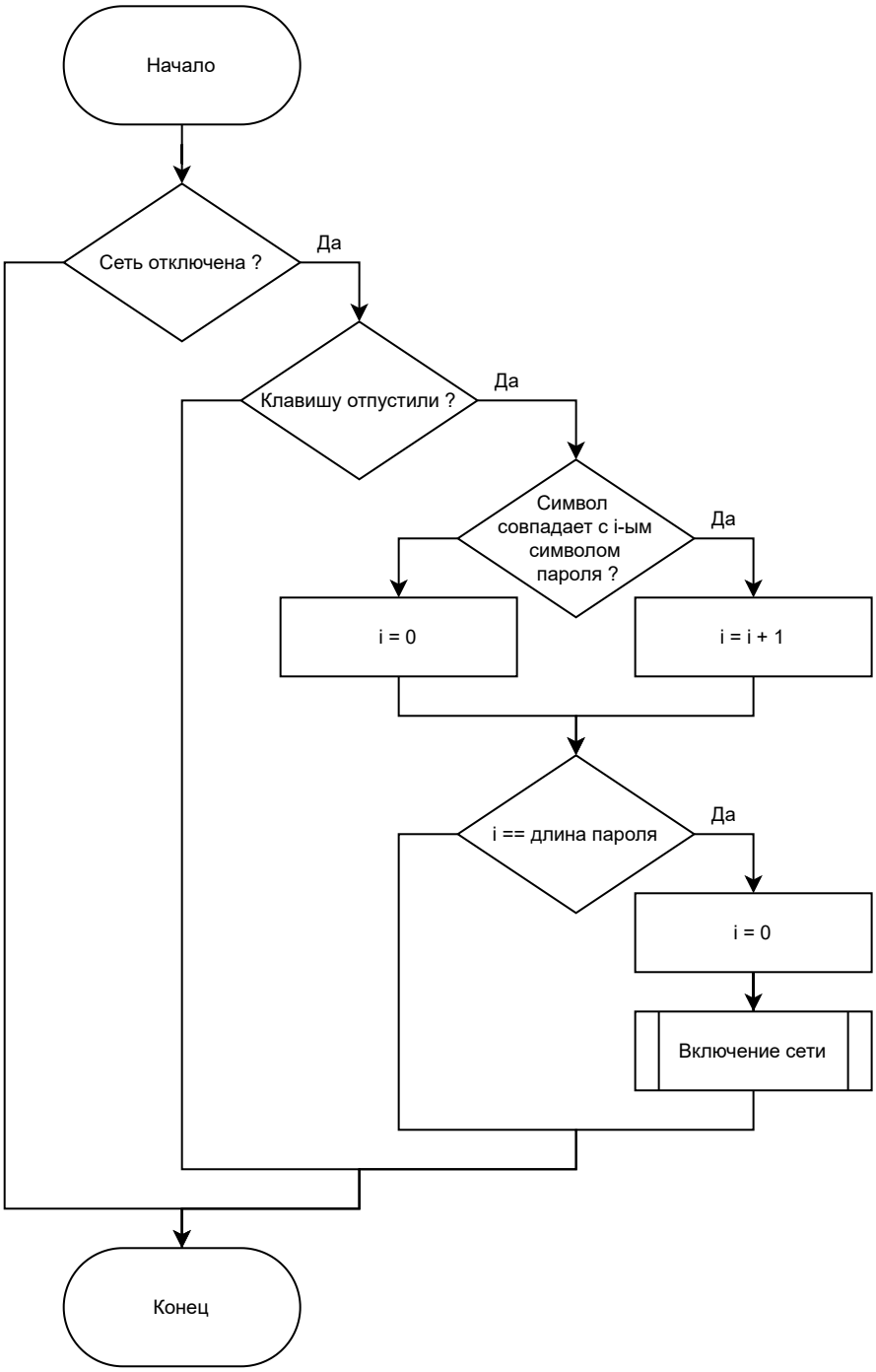

Рисунок 2.6 – Схема алгоритма обработчика событий от клавиатуры

В начале проверяется состояние сети устройства – если сеть не отключена, дальнейшая обработка не требуется. В противном случае, введенный символ сравнивается с очередным символом пароля. В случае, если символы не совпадают, попытка ввода считается неудачной. Если успешно введен весь

пароль, производится включение сети.

2.5 Структура программного обеспечения

Составляющие проекта приведены на рисунке 2.7.

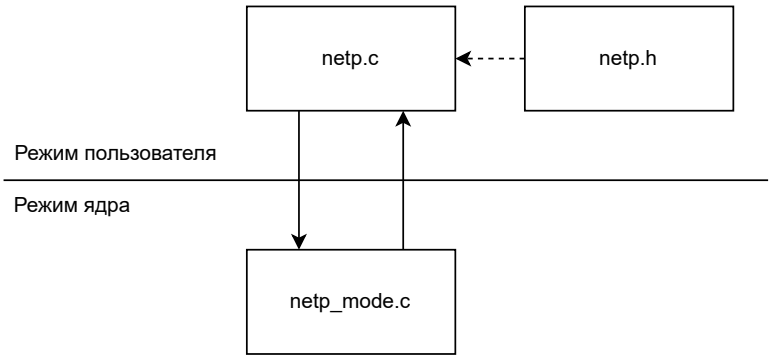

Рисунок 2.7 – Структура ПО

3 Технологический раздел

3.1 Выбор языка и среды программирования

Разработанный модуль ядра написан на языке программирования `C` [14]. Выбор языка программирования `C` основан на том, что в настоящий момент большая часть исходного кода ядра Linux, его модулей и драйверов написана на данном языке [15].

В качестве компилятора выбран `gcc` [16]. Выбор обоснован тем, что данный компилятор является предпочтительным для сборки Linux [17].

В качестве среды разработки выбрана среда `Visual Studio Code` [18].

3.2 Загружаемый модуль ядра

3.2.1 Уведомитель для USB–устройств

В листинге 3.1 представлено объявление уведомителя и его функции–обработчика.

Листинг 3.1 – Уведомитель для USB–устройств. Часть 1

```
static int
usb_notifier_call(struct notifier_block *self, unsigned long action
    , void *dev);

static struct notifier_block usb_notify = {
  .notifier_call = usb_notifier_call,
};
```

В листингах 3.2 и 3.3 представлено определение функции–обработчика уведомителя для USB–устройств. Функции, используемые в теле данного обработчика описаны в разделе 3.3.3.

Листинг 3.2 – Уведомитель для USB–устройств. Часть 2

```
static int
usb_notifier_call(struct notifier_block *self, unsigned long action
    , void *dev)
{
  // Events, which our notifier react.
  switch (action)
  {
  case USB_DEVICE_ADD:
```

Листинг 3.3 – Уведомитель для USB–устройств. Часть 3

```
    usb_dev_insert(dev);
    break;
  case USB_DEVICE_REMOVE:
    usb_dev_remove(dev);
    break;
  default:
    break;
  }

  return NOTIFY_OK;
}
```

3.2.2 Уведомитель для клавиатуры

В листинге 3.4 представлено объявление уведомителя и его функции–обработчика.

Листинг 3.4 – Уведомитель для клавиатуры

```
static int
kbd_notifier_call(struct notifier_block *self, unsigned long action
   , void *_param);

static struct notifier_block kbd_notify = {
  .notifier_call = kbd_notifier_call,
};
```

В листингах 3.5 и 3.6 представлено определение функции–обработчика уведомителя. Функции, используемые в теле данного обработчика описаны в разделе 3.4.1.

Листинг 3.5 – Обработчик событий от клавиатуры

```
static int
kbd_notifier_call(struct notifier_block *self, unsigned long action
   , void *_param)
{
  if (!kbd_notifier_verify_action(action, _param))
    return NOTIFY_OK;

  if (!kbd_notifier_verify_pwd_len())
    return NOTIFY_OK;
```

Листинг 3.6 – Обработчик событий от клавиатуры

```
  struct keyboard_notifier_param *param = _param;
  char symbol = param->value;

  kbd_notifier_process_action(symbol);

  return NOTIFY_OK;
}
```

3.2.3 Регистрация уведомителей

Реализация алгоритма из пункта 2.2, приведена в листинге 3.7.

Листинг 3.7 – Регистрация и дерегистрация уведомителей

```
__init netpmod_init(void)
{
  usb_register_notify(&usb_notify);
  register_keyboard_notifier(&kbd_notify);

  pr_info("netpmod: module loaded\n");

  return 0;
}

// Module exit function.
static void
__exit netpmod_exit(void)
{
  unregister_keyboard_notifier(&kbd_notify);
  usb_unregister_notify(&usb_notify);

  pr_info("netpmod: module unloaded\n");
}

module_init(netpmod_init);
module_exit(netpmod_exit);
```

3.3 Обработчик событий от USB

3.3.1 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, SERIAL`), а так же указатель на элемент списка.

Структура `int_usb_device`, а так же инициализация списка, в котором будут храниться данные структуры представлены в листинге 3.8.

Листинг 3.8 – Структура `int_usb_device`

```
typedef struct int_usb_device_id {
  struct usb_device_id id;
  char                 *serial;
} int_usb_device_id_t;

#define INT_USB_DEVICE(v, p, s)\
  .id={USB_DEVICE(v, p)},\
  .serial=(s)

typedef struct int_usb_device
{
  int_usb_device_id_t dev_id;
  struct list_head    list_node;
} int_usb_device_t;

LIST_HEAD(connected_devices);
```

Список отслеживаемых устройств должен модифицироваться при подключении и удалении USB–устройств. Для этого, при подключении или удалении устройства, создается экземпляр структуры `int_usb_device` и помещается в список отслеживаемых устройств или удаляется из него.

В листинге 3.9 представлены функции для работы со списком отслеживаемых устройств.

Листинг 3.9 – Функции для работы со списком устройств

```
static void
add_int_usb_dev(const struct usb_device * const dev)
{
  int_usb_device_t *new_usb_device = (int_usb_device_t *)kmalloc(
     sizeof(int_usb_device_t), GFP_KERNEL);
  int_usb_device_id_t new_id = {
    INT_USB_DEVICE(dev->descriptor.idVendor, dev->descriptor.
       idProduct, dev->serial),
  };

  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(const struct usb_device * const 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);
    }
}
```

3.3.2 Идентификация устройства как доверенного

Для проверки устройства необходимо проверить его идентификационные данные с данными доверенных устройств.

В листингах 3.10–3.12 представлены объявление списка доверенных устройств и функции для идентификации устройства.

Листинг 3.10 – Функции идентификации устройств. Часть 1

```
static int_usb_device_id_t allowed_devs[] = {
  {INT_USB_DEVICE(0x0781, 0x5571, "03021524050621080032")},
};
```

Листинг 3.11 – Функции идентификации устройств. Часть 2

```
static bool
is_dev_matched(const struct usb_device * const dev, const
   int_usb_device_id_t *const dev_id)
{
  return dev_id->id.idVendor == dev->descriptor.idVendor
      && dev_id->id.idProduct == dev->descriptor.idProduct
      && !strcmp(dev_id->serial, dev->serial);
}

static bool
is_dev_id_matched(const int_usb_device_id_t * const new_dev_id,
   const int_usb_device_id_t * const dev_id)
{
  return dev_id->id.idVendor == new_dev_id->id.idVendor
      && dev_id->id.idProduct == new_dev_id->id.idProduct
      && !strcmp(dev_id->serial, new_dev_id->serial);
}

static bool
is_dev_allowed(const int_usb_device_id_t * const dev)
{
  unsigned long allowed_devs_len = sizeof(allowed_devs) / sizeof(
     int_usb_device_id_t);

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

  return false;
}

static int
count_not_acked_devs(void)
{
  int count = 0;

  int_usb_device_t *temp;
  list_for_each_entry(temp, &connected_devices, list_node)
    if (!is_dev_allowed(&temp->dev_id))
```

Листинг 3.12 – Функции идентификации устройств. Часть 3

```
      count++;

  return count;
}
```

3.3.3 Обработка событий USB–устройства

Реализация алгоритмов из пункта 2.3, приведена в листингах 3.13 и 3.14.

Отключение и восстановление сети происходит путем вызова программы `modprobe` через `usermode-helper API`.

Листинг 3.13 – Обработчик подключения USB–устройства

```
static void
usb_dev_insert(const struct usb_device * const dev)
{
  pr_info("netpmod: dev connected with PID '%d' and VID '%d' and
     SERIAL '%s'\n",
       dev->descriptor.idProduct, dev->descriptor.idVendor, dev->
          serial);

  add_int_usb_dev(dev);

  int not_acked_devs = count_not_acked_devs();
  if (!not_acked_devs)
  {
    pr_info("netpmod: allowed dev connected, skipping network
       killing\n");
    return;
  }

  pr_info("netpmod: %d not allowed devs connected, killing network\
     n", not_acked_devs);

  if (is_network_disabled())
    return;

  disable_network();
}
```

Листинг 3.14 – Обработчик удаления USB–устройства

```
static void
usb_dev_remove(const struct usb_device * const dev)
{
  pr_info("netpmod: dev disconnected with PID '%d' and VID '%d' and
      SERIAL '%s'\n",
       dev->descriptor.idProduct, dev->descriptor.idVendor, dev->
          serial);
  delete_int_usb_dev(dev);

  if (!is_network_disabled())
    return;

  int not_acked_devs = count_not_acked_devs();
  if (not_acked_devs)
  {
    pr_info("netpmod: %d not allowed devs connected, nothing to do\
       n", not_acked_devs);
    return;
  }

  pr_info("netpmod: all not allowed devs are disconnected, bringing
      network back\n");

  enable_network();
}
```

3.4 Обработчик событий от клавиатуры

3.4.1 Обработка событий клавиатуры

Уведомители от клавиатуры поддерживают пять типов событий: KBD_KEYCODE, KBD_UNBOUND_KEYCODE, KBD_UNICODE, KBD_KEYSYM и KBD_POST_KEYSYM. Каждый из обработчиков событий клавиатуры получает все пять типов событий.

Событие KBD_KEYSYM позволяет получить информацию о введенном сиволе из таблицы ASCII, в связи с чем будет использоваться обработчик именно этого события.

Листинг 3.15 – Объявление используемых значений

```
static size_t matched_password_len = 0;
static size_t password_len = 0;
```

В листинге 3.16 представлена функция валидации событий от клавиатуры.

Листинг 3.16 – Функция валидации события

```
static int
kbd_notifier_verify_action(unsigned long action, void *_param)
{
  if (!is_network_disabled())
    return 0;

  struct keyboard_notifier_param *param = _param;
  if (action != KBD_KEYSYM || !param->down)
    return 0;

  return 1;
}
```

Если не задан пароль, следует полностью исключить возможность включения сети без удаления незарегистрированных USB–устройств.

В листинге 3.17 представлена функция валидации пароля, указанного в параметрах загружаемого модуля.

Листинг 3.17 – Функция валидации пароля

```
static int
kbd_notifier_verify_pwd_len(void)
{
  if (!password_len)
    password_len = strlen(password);

  if (!password_len)
    return 0;

  return 1;
}
```

Наконец, в листинге 3.18 представлена функция обработки введенного символа.

Листинг 3.18 – Функция обработки символа

```
static void
kbd_notifier_process_action(char symbol)
{
  if (symbol < ' ' || symbol > '~')
    return;

  if (symbol != password[matched_password_len])
  {
    matched_password_len = 0;
    return;
  }

  if (++matched_password_len == password_len)
  {
    pr_info("netpmod: password matched, bringing network back\n");

    matched_password_len = 0;
    enable_network();
  }
}
```

Реализация алгоритма из пункта 2.4, приведена в листинге 3.19.

Листинг 3.19 – Обработчик событий клавиатуры

```
static int
kbd_notifier_call(struct notifier_block *self, unsigned long action
   , void *_param)
{
  if (!kbd_notifier_verify_action(action, _param))
    return NOTIFY_OK;

  if (!kbd_notifier_verify_pwd_len())
    return NOTIFY_OK;

  struct keyboard_notifier_param *param = _param;
  char symbol = param->value;

  kbd_notifier_process_action(symbol);

  return NOTIFY_OK;
}
```

4 Исследовательский раздел

4.1 Примеры работы разработанного ПО

В листингах 4.1 и 4.2 представлены примеры подключения и удаления USB–устройства не из списка доверенных, а так же последующее отключение и включение сетевого драйвера соответственно.

Листинг 4.1 – Подключение недоверенного устройства и проверка сети

```
bmstu-iu7-os-cp/src [dev] $ sudo dmesg | grep 'netpmod:'
[  498.489270] netpmod: module loaded
[  504.583702] netpmod: device connected with PID '21873' and VID '
   1921' and SERIAL '03021524050621080032'
[  504.583706] netpmod: 1 not allowed devices connected, killing
   network
[  504.583974] netpmod: network is killed
bmstu-iu7-os-cp/src [dev]  ping -c 3 google.com
ping: google.com: Temporary failure in name resolution
```

Листинг 4.2 – Отключение недоверенного устройства и проверка сети

```
bmstu-iu7-os-cp/src [dev] $ sudo dmesg | grep 'netpmod:'
...
[  589.893378] netpmod: device disconnected with PID '21873' and
   VID '1921' and SERIAL '03021524050621080032'
[  589.893383] netpmod: all not allowed devices are disconnected,
   bringing network back
[  589.893782] netpmod: network is available now
bmstu-iu7-os-cp/src [dev] $ ping -c 3 google.com
PING google.com (108.177.14.102) 56(84) bytes of data.
64 bytes from lt-in-f102.1e100.net (108.177.14.102): icmp_seq=1 ttl
   =59 time=18.9 ms
64 bytes from lt-in-f102.1e100.net (108.177.14.102): icmp_seq=2 ttl
   =59 time=19.3 ms
64 bytes from lt-in-f102.1e100.net (108.177.14.102): icmp_seq=3 ttl
   =59 time=19.5 ms

--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 18.905/19.225/19.463/0.235 ms
```

В листинге 4.3 представлен пример подключения USB–устройства из списка доверенных.

Листинг 4.3 – Подключение доверенного устройства и проверка сети

```
bmstu-iu7-os-cp/src [dev] $ sudo dmesg | grep 'netpmod:'
...
[  734.312735] netpmod: module loaded
[  739.479982] netpmod: device connected with PID '21873' and VID '
   1921' and SERIAL '03021524050621080032'
[  739.479987] netpmod: allowed device connected, skipping network
   killing
bmstu-iu7-os-cp/src [dev]  ping -c 3 google.com
PING google.com(lu-in-x71.1e100.net (2a00:1450:4010:c0e::71)) 56
   data bytes
64 bytes from lu-in-f113.1e100.net (2a00:1450:4010:c0e::71):
   icmp_seq=1 ttl=60 time=19.1 ms
64 bytes from lu-in-f113.1e100.net (2a00:1450:4010:c0e::71):
   icmp_seq=2 ttl=60 time=21.6 ms
64 bytes from lu-in-x71.1e100.net (2a00:1450:4010:c0e::71):
   icmp_seq=3 ttl=60 time=21.7 ms

--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 19.098/20.805/21.688/1.207 ms
```

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были выполнены поставленные задачи:

— проанализированы методы обработки событий, возникающих при взаимодействии с USB–устройствами;

— проанализированы структуры и функции ядра, предоставляющие информацию о USB–устройствах;

— разработаны алгоритмы и структуру программного обеспечения;

— реализовано программное обеспечение;

— исследовано разработанное программное обеспечение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Juice Jacking: Security Issues and Improvements in USB Technology / D. Singh [и др.] // Sustainability. — 2022. — Янв. — Т. 14. — DOI: 10.3390/su14020939.
2. usbmon — USB Drop Attacks: The Danger Of 'Lost And Found' Thumb Drives [Электронный ресурс]. — Режим доступа: https://www.redteamsecure.com/blog/usb-drop-attacks-the-danger-of-lost-and-found-thumb-drives (дата обращения: 23.09.2022).
3. usbmon — The Linux Kernel documentation [Электронный ресурс]. — Режим доступа: https://www.kernel.org/doc/html/latest/usb/usbmon.html (дата обращения: 23.09.2022).
4. Wireshark · Go Deep. [Электронный ресурс]. — Режим доступа: https://www.wireshark.org/ (дата обращения: 23.09.2022).
5. udevadm(8) - Linux manual page [Электронный ресурс]. — Режим доступа: https://man7.org/linux/man-pages/man8/udevadm.8.html (дата обращения: 23.09.2022).
6. libudev [Электронный ресурс]. — Режим доступа: https://www.freedesktop.org/software/systemd/man/libudev.html (дата обращения: 23.09.2022).
7. Notification Chains in Linux Kernel [Электронный ресурс]. — Режим доступа: https://0xax.gitbooks.io/linux-insides/content/Concepts/linux-cpu-4.html (дата обращения: 25.09.2022).
8. notifier.h - include/linux/notifier.h - Linux source code (v5.19) - Bootlin [Электронный ресурс]. — Режим доступа: https://elixir.bootlin.com/linux/v5.19/source/include/linux/notifier.h (дата обращения: 25.09.2022).
9. usb.h - include/linux/usb.h - Linux source code (v5.19) - Bootlin [Электронный ресурс]. — Режим доступа: https://elixir.bootlin.com/linux/v5.19/source/include/linux/usb.h (дата обращения: 26.09.2022).
10. Document Library | USB-IF [Электронный ресурс]. — Режим доступа: https://www.usb.org/documents (дата обращения: 25.09.2022).

11. mod_devicetable.h - include/linux/mod_devicetable.h - Linux source code (v5.19) - Bootlin [Электронный ресурс]. — Режим доступа: https://elixir.bootlin.com/linux/v5.19/source/include/linux/mod_devicetable.h (дата обращения: 26.09.2022).

12. Invoking user-space applications from the kernel – IBM Developer [Электронный ресурс]. — Режим доступа: https://developer.ibm.com/articles/l-user-space-apps/ (дата обращения: 27.09.2022).

13. umh.h - include/linux/umh.h - Linux source code (v5.19) - Bootlin [Электронный ресурс]. — Режим доступа: https://elixir.bootlin.com/linux/v5.19/source/include/linux/umh.h (дата обращения: 27.09.2022).

14. ISO/IEC 9899:1990 Programming languages — C [Электронный ресурс]. — Режим доступа: https://www.iso.org/standard/17782.html (дата обращения: 14.10.2022).

15. Линус Торвальдс запланировал внедрение Rust в Linux 6.1 [Электронный ресурс]. — Режим доступа: https://www.linux.org.ru/news/kernel/16978439 (дата обращения: 14.10.2022).

16. GCC, the GNU Compiler Collection [Электронный ресурс]. — Режим доступа: https://gcc.gnu.org/ (дата обращения: 14.10.2022).

17. How to Compile a Linux Kernel [Электронный ресурс]. — Режим доступа: https://www.linux.com/topic/desktop/how-compile-linux-kernel-0/ (дата обращения: 14.10.2022).

18. Visual Studio Code - Code Editing. Redefined [Электронный ресурс]. — Режим доступа: https://code.visualstudio.com (дата обращения: 14.10.2022).

ПРИЛОЖЕНИЕ А
Исходный код загружаемого модуля

Листинг А.1 – Исходный код netp.h

```
#ifndef __NETP_H__
#define __NETP_H__

bool is_network_disabled(void);

void disable_network(void);

void enable_network(void);

#endif
```

Листинг А.2 – Исходный код netp.c. Часть 1

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

#include "netp.h"

// params ----------------------------------------

static char *net_modules[16] = {"iwlwifi"};
static int net_modules_n = 1;
module_param_array(net_modules, charp, &net_modules_n, S_IRUGO);
MODULE_PARM_DESC(net_modules, "List of modules to manipulate with")
   ;

// params ^--------------------------------------

static bool is_network_down = false;

bool
is_network_disabled(void)
{
  return is_network_down;
}

static char *envp[] = {"HOME=/", "TERM=linux", "PATH=/sbin:/bin:/
   usr/sbin:/usr/bin", NULL};
```

Листинг А.3 – Исходный код netp.c. Часть 2

```
void
disable_network(void)
{
  int i = 0;
  for (; i < net_modules_n; i++)
  {
    char *argv[] = {"/sbin/modprobe", "-r", net_modules[i], NULL};
    if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC > 0)
       )
    {
      pr_warn("netpmod: unable to kill network\n");
    }
    else
    {
      pr_info("netpmod: network is killed\n");
      is_network_down = true;
    }
  }
}

void
enable_network(void)
{
  int i = 0;
  for (; i < net_modules_n; i++)
  {
    char *argv[] = {"/sbin/modprobe", net_modules[i], NULL};
    if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC > 0)
       )
    {
      pr_warn("netpmod: unable to bring network back\n");
    }
    else
    {
      pr_info("netpmod: network is available now\n");
      is_network_down = false;
    }
  }
}
```

Листинг А.4 – Исходный код netp_mod.c. Часть 1

```
#include <linux/module.h>
#include <linux/usb.h>
#include <linux/keyboard.h>
#include <linux/slab.h> // for kmalloc, kfree
#include <linux/string.h>

#include "netp.h"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gregory Mironov");
MODULE_VERSION("1.0.0");

// params ----------------------------------------

static char *password = {"qwery"};
module_param(password, charp, 0000);
MODULE_PARM_DESC(password, "Password to reenable network manually")
    ;

// params ^--------------------------------------

// init ------------------------------------------

static int
usb_notifier_call(struct notifier_block *self, unsigned long action
    , void *dev);

static struct notifier_block usb_notify = {
  .notifier_call = usb_notifier_call,
};

static int
kbd_notifier_call(struct notifier_block *self, unsigned long action
    , void *_param);

static struct notifier_block kbd_notify = {
  .notifier_call = kbd_notifier_call,
};
```

Листинг А.5 – Исходный код netp_mod.c. Часть 2

```
// Module init function.
static int
__init netpmod_init(void)
{
  usb_register_notify(&usb_notify);
  register_keyboard_notifier(&kbd_notify);

  pr_info("netpmod: module loaded\n");

  return 0;
}

// Module exit function.
static void
__exit netpmod_exit(void)
{
  unregister_keyboard_notifier(&kbd_notify);
  usb_unregister_notify(&usb_notify);

  pr_info("netpmod: module unloaded\n");
}

module_init(netpmod_init);
module_exit(netpmod_exit);

// init ^-----------------------------------

// usb handler ------------------------------

typedef struct int_usb_device_id {
  struct usb_device_id id;
  char                 *serial;
} int_usb_device_id_t;

#define INT_USB_DEVICE(v, p, s)\
  .id={USB_DEVICE(v, p)},\
  .serial=(s)
```

Листинг А.6 – Исходный код netp_mod.c. Часть 3

```
typedef struct int_usb_device
{
  int_usb_device_id_t dev_id;
  struct list_head    list_node;
} int_usb_device_t;

LIST_HEAD(connected_devices);

static int_usb_device_id_t allowed_devs[] = {
  {INT_USB_DEVICE(0x0781, 0x5571, "03021524050621080032")},
};

static bool
is_dev_matched(const struct usb_device * const dev, const
   int_usb_device_id_t *const dev_id)
{
  return dev_id->id.idVendor == dev->descriptor.idVendor
      && dev_id->id.idProduct == dev->descriptor.idProduct
      && !strcmp(dev_id->serial, dev->serial);
}

static bool
is_dev_id_matched(const int_usb_device_id_t * const new_dev_id,
   const int_usb_device_id_t * const dev_id)
{
  return dev_id->id.idVendor == new_dev_id->id.idVendor
      && dev_id->id.idProduct == new_dev_id->id.idProduct
      && !strcmp(dev_id->serial, new_dev_id->serial);
}

static bool
is_dev_allowed(const int_usb_device_id_t * const dev)
{
  unsigned long allowed_devs_len = sizeof(allowed_devs) / sizeof(
     int_usb_device_id_t);

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

Листинг А.7 – Исходный код netp_mod.c. Часть 4

```
  return false;
}

static int
count_not_acked_devs(void)
{
  int count = 0;

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

  return count;
}

static void
add_int_usb_dev(const struct usb_device * const dev)
{
  int_usb_device_t *new_usb_device = (int_usb_device_t *)kmalloc(
     sizeof(int_usb_device_t), GFP_KERNEL);
  int_usb_device_id_t new_id = {
    INT_USB_DEVICE(dev->descriptor.idVendor, dev->descriptor.
       idProduct, dev->serial),
  };

  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(const struct usb_device * const 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);
```

Листинг А.8 – Исходный код netp_mod.c. Часть 5

```
      kfree(device);
    }
}

// Handler for USB insertion.
static void
usb_dev_insert(const struct usb_device * const dev)
{
  pr_info("netpmod: dev connected with PID '%d' and VID '%d' and
     SERIAL '%s'\n",
       dev->descriptor.idProduct, dev->descriptor.idVendor, dev->
          serial);

  add_int_usb_dev(dev);

  int not_acked_devs = count_not_acked_devs();
  if (!not_acked_devs)
  {
    pr_info("netpmod: allowed dev connected, skipping network
       killing\n");
    return;
  }

  pr_info("netpmod: %d not allowed devs connected, killing network\
     n", not_acked_devs);

  if (is_network_disabled())
    return;

  disable_network();
}

// Handler for USB removal.
static void
usb_dev_remove(const struct usb_device * const dev)
{
  pr_info("netpmod: dev disconnected with PID '%d' and VID '%d' and
      SERIAL '%s'\n",
       dev->descriptor.idProduct, dev->descriptor.idVendor, dev->
          serial);
```

Листинг А.9 – Исходный код netp_mod.c. Часть 6

```
  delete_int_usb_dev(dev);

  if (!is_network_disabled())
    return;

  int not_acked_devs = count_not_acked_devs();
  if (not_acked_devs)
  {
    pr_info("netpmod: %d not allowed devs connected, nothing to do\
       n", not_acked_devs);
    return;
  }

  pr_info("netpmod: all not allowed devs are disconnected, bringing
      network back\n");

  enable_network();
}

// Handler for event's notifier.
static int
usb_notifier_call(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 NOTIFY_OK;
}
```

Листинг А.10 – Исходный код netp_mod.c. Часть 7

```
// usb handler ^----------------------------------

// keyboard handler ------------------------------

static size_t matched_password_len = 0;
static size_t password_len = 0;

static int
kbd_notifier_verify_action(unsigned long action, void *_param)
{
  if (!is_network_disabled())
    return 0;

  struct keyboard_notifier_param *param = _param;
  if (action != KBD_KEYSYM || !param->down)
    return 0;

  return 1;
}

static int
kbd_notifier_verify_pwd_len(void)
{
  if (!password_len)
    password_len = strlen(password);

  if (!password_len)
    return 0;

  return 1;
}

static void
kbd_notifier_process_action(char symbol)
{
  if (symbol < ' ' || symbol > '~')
    return;

  if (symbol != password[matched_password_len])
```

Листинг А.11 – Исходный код netp_mod.c. Часть 8

```
  {
    matched_password_len = 0;
    return;
  }

  if (++matched_password_len == password_len)
  {
    pr_info("netpmod: password matched, bringing network back\n");

    matched_password_len = 0;
    enable_network();
  }
}

static int
kbd_notifier_call(struct notifier_block *self, unsigned long action
   , void *_param)
{
  if (!kbd_notifier_verify_action(action, _param))
    return NOTIFY_OK;

  if (!kbd_notifier_verify_pwd_len())
    return NOTIFY_OK;

  struct keyboard_notifier_param *param = _param;
  char symbol = param->value;

  kbd_notifier_process_action(symbol);

  return NOTIFY_OK;
}

// keyboard handler ^-------------------------
```