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» (ИУ7)_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

Загружаемый модуль ядра для отслеживания USB-устройств, являющихся ключом для доступа к приложению.

Студент __ИУ7-73Б_____ _____ ____**Сиденко А. Г.**____
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ ____**Тассов К. Л.**___
(Подпись, дата) (И.О.Фамилия)

2020 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ___ИУ7__
(Индекс)
_____ __И.В.Рудаков___
(И.О.Фамилия)
« _____ » _____ 20_ __ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _Операционные системы_____

Студент группы __ИУ7-73Б____

_____Сиденко Анастасия Генадьевна_____
(Фамилия, имя, отчество)

Тема курсового проекта _Загружаемый модуль ядра для отслеживания USB-устройств, являющихся ключом для доступа к приложению. _____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_учебный_____
Источник тематики (кафедра, предприятие, НИР) ___кафедра_____

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

Задание __ Реализовать загружаемый модуль ядра для отслеживания изменений на usb-портах и проверка на наличие допуска. При наличии допуска, предоставление доступа к приложению.
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на _20-25__ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
К защите должны быть подготовлены презентация и доклад, отражающие суть выполненной работы, содержание и методы решения основных задач, а также полученные результаты.

_____

Дата выдачи задания « _ _ » _____ 20__ г.

Руководитель курсового проекта _____ _____ ___К.Л. Тассов_____
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____ ___А.Г. Сиденко____
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

РЕФЕРАТ

Курсовой проект представляет собой загружаемого модуля ядра для отслеживания USB-устройств, являющихся ключом для доступа к приложению.

Ключевые слова: загружаемый модуль, Linux, USB устройство, уведомитель(notify), шифрование.

Отчёт содержит 32 страниц, 4 рисунка, 7 источников, 1 прил.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время кража персональных данных – это большая проблема во всем мире. Одним из способов защиты ваших персональных данных на компьютере, это доступ к ним по USB-устройству [1].

Если вы не хотите, чтобы кто-то извлекал документы с Вашего компьютера через USB или устанавливал вредоносные программы, необходимо отслеживать USB-устройства, их идентифицировать и предоставлять или отказывать в доступе.

Linux – это операционная система с монолитным ядром. Для того, чтобы избежать перекомпиляции ядра при добавлении нового функционала, используются загружаемые модули ядра. Целью данной работы является реализация загружаемого модуля ядра для отслеживания USB-устройств, являющихся ключом для доступа к приложению.

Необходимая функциональность.

1. Список разрешенных устройств.
2. Список секретных файлов, приложений.
3. Предоставление или отказ в доступе, при наличии различных USB-устройств.

Для достижения поставленной цели необходимо решить следующие задачи.

1. Определение основных понятий.
2. Разработка алгоритмов.
3. Реализация загружаемого модуля.

1 Аналитический раздел

Целью данной работы является реализация загружаемого модуля ядра для отслеживания USB-устройств, являющихся ключом для доступа к приложению. В данном разделе производится постановка задачи и рассмотрение основных понятий.

1.1 Постановка задачи

Требуется разработать программное обеспечение для отслеживания USB-устройств, который обладает следующей функциональностью:

- список разрешенных устройств;
- список путей к секретным файлам;
- отслеживание появления новых USB-устройств;
  - если устройство известно, происходит расшифровка файла;
  - если устройство не известно, происходит зашифровка файла.

На вход подается USB устройство с паролем. На выходе получаем зашифрованный или расшифрованный файл.

1.2 Загружаемый модуль ядра

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

Загружаемый модуль представляет собой специальный объектный файл в формате ELF (Executable and Linkable Format). Обычно объектные файлы обрабатываются компоновщиком, который разрешает символы и

формирует исполняемый файл. Однако в связи с тем, что загружаемый модуль не может разрешить символы до загрузки в ядро, он остается ELF-объектом. Для работы с загружаемыми модулями можно использовать стандартные средства работы с объектными файлами (имеют суффикс .ko, от kernel object). [2]

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

insmod – Загружает модуль в ядро из конкретного файла, если модуль зависит от других модулей. Только суперпользователь может загрузить модуль в ядро.

lsmod – Выводит список модулей, загруженных в ядро.

modinfo – Извлекает информацию из модулей ядра (лицензия, автор, описание и т.д.).

rmmod – Команда используется для выгрузки модуля из ядра, в качестве параметра передается имя файла модуля. Только суперпользователь может выгрузить модуль из ядра.

Загружаемые модули ядра должны содержать два макроса module_init и module_exit.

1.3 **Уведомления в ядре Linux**

1.3.1 **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, загрузка и выгрузка модулей, перезагрузки системы, изменения сетевых устройств и т. д. [3]

Основной является структура notifier_block, листинг которой представлен в 1.1.

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 };
```

Структура определен в #include/linux/notifier.h. Эта структура содержит указатель на функцию обратного – notifier_call, ссылку на следующий notifier_block и приоритет функции, функции с более высоким приоритетом выполняются первыми.

1.3.2 Уведомитель изменений на USB портах

Существует уведомитель, позволяющий отслеживать изменения на usb портах. [4]

*void usb_register_notify(struct notifier_block *nb);*

*void usb_unregister_notify(struct notifier_block *nb);*

Существующие события: *USB_DEVICE_ADD* – добавление нового устройства, *USB_DEVICE_REMOVE* – удаление устройства.

1.4 Хранение информации о доступных USB устройствах

Для хранения устройств будет использовать двусвязный список ядра Linux, реализованный в файле #include/linux/list.h. [5]

LIST_HEAD – объявление и инициализация головы списка.

list_for_each_entry(temp, &connected_devices, list_node) – проход по списку.

list_for_each_entry_safe(device, temp, &connected_devices, list_node) – «защищенный» проход по всем элементам списка, используется для удаления записей списка.

*list_add_tail(struct list_head * new, struct list_head * head)* – добавление нового элемента.

1.5 Вызов приложений пользовательского пространства из ядра

Usermode-helper API – это простой API с известным набором опций. Например, чтобы создать процесс из пользовательского пространства, обычно необходимо указать имя исполняемого файла, параметры исполняемого файла и набор переменных среды. [6]

*int call_usermodehelper(const char *path, char **argv, char **envp, int wait)* – подготовить и запустить приложение пользовательского режима.

*const char * path* – путь к исполняемому файлц пользовательского режима.

*char ** argv* – параметры.

*char ** envp* – переменные среды.

int wait – дождитесь завершения работы приложения и возврата статуса.

Реализация usermodehelper проста и понятна, рисунок 1.1.

UMH_WAIT_PROC – если запрашивающий хочет дождаться завершения всего процесса, включая вызов пользовательского пространства, *UMH_NO_WAIT* – вообще не ждать, *UMH_WAIT_EXEC* – дождаться вызова приложения пользовательского пространства, но не завершения.

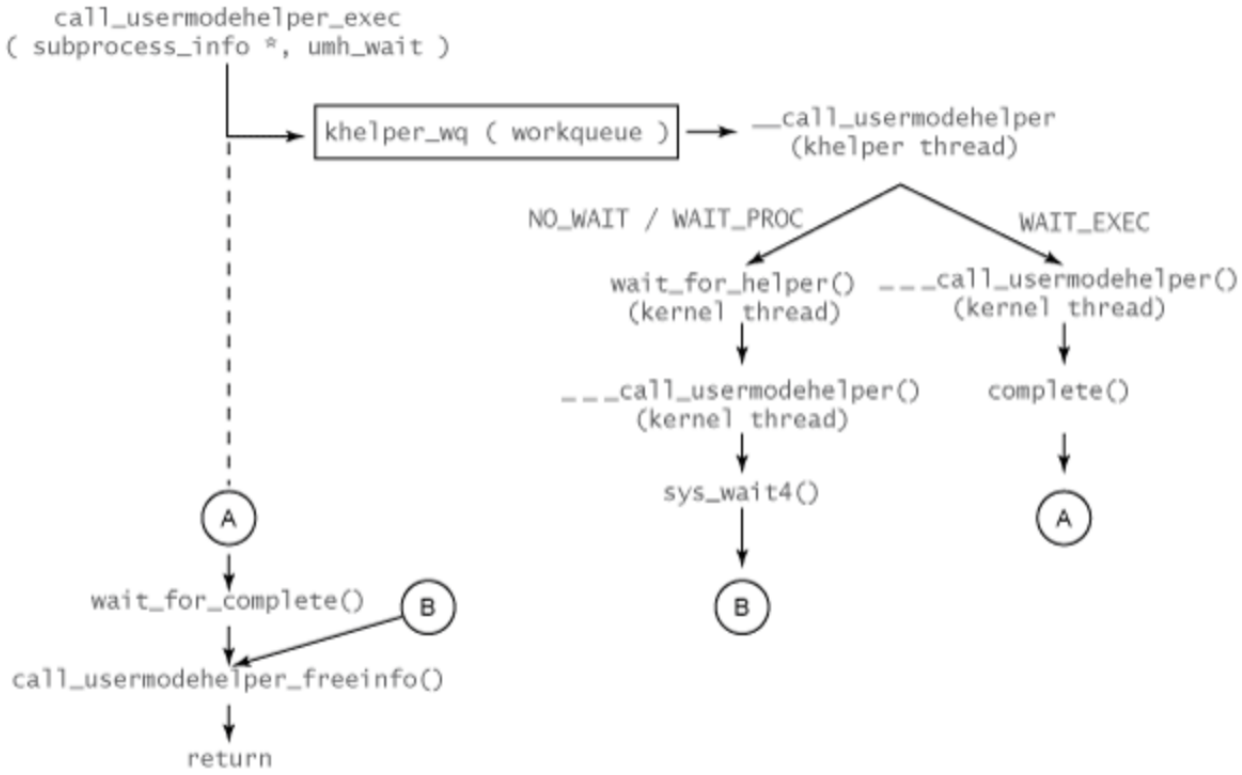

Рисунок 1.1 — Внутренняя реализация API usermodehelper.

1.6 **Чтение и запись файлов в пространстве ядра**

Иногда необходимо читать и записывать файловые данные в ядре Linux. [7]

В основном это функции:

struct file filp_open(const char* filename, int open_mode, int mode)* – открытие файла в ядре. filename – имя файла, который может быть создан или открыт, включает путь до файла; open_mode – режим открытия файла O_CREAT, O_RDWR, O_RDONLY, mode – используется при создании файла, установите разрешения на чтение и запись созданного файла, в противном случае он может быть установлен в 0.

*int filp_close(struct file*filp, fl_owner_t id)* – закрытие файла.

ssize_t vfs_read(struct file filp, char __user* buffer, size_t len, loff_t* pos)*, *ssize_t vfs_write(struct file* filp, const char __user* buffer, size_t len, loff_t* pos)* – чтение и запись файлов в ядре.

Второй параметр этих двух функций имеет перед собой модификатор __user, который требует, чтобы оба указателя буфера указывали на память пространства пользователя. Чтобы эти две функции чтения и записи правильно работали с указателем буфера в пространстве ядра, вам нужно использовать функцию set_fs(). Ее функция состоит в том, чтобы изменить способ, которым ядро обрабатывает проверку адресов памяти. На самом деле параметр fs этой функции имеет только два значения: USER_DS и KERNEL_DS, которые представляют пространство пользователя и пространство ядра соответственно.

void set_fs(mm_segment_t fs)

mm_segment_t get_fs ()

1.7 Основные используемые структуры

В данной работе происходит отслеживание изменений на usb портах, основными структурами являются usb_device и usb_device_id.

1.7.1 usb_device

Структура usb_device приведена в листинге А.1 – представление USB-устройста в ядре.

Используемые поля:

descriptor – дескриптор USB устройства.

Каждое продающееся устройство с USB требует сертификации на соответствие требованиям USB, для чего ему необходимо иметь ID поставщика (vendor ID) и ID изделия (product ID). Эти поля присутствуют в descriptor, используются для идентификации USB устройства.

1.7.2 usb_device_id

Структура usb_device_id приведена в листинге А.2 – идентификация USB устройств для отслеживания и подключения.

Используемые поля:

idVendor – ID поставщика;

idProduct – ID изделия.

1.8 **Вывод**

В данном разделе была постановлена задача и рассмотрены основные понятия.

2 Конструкторский раздел

Разрабатываемое программное обеспечения можно разделить на подзадачи:

— загружаемый модуль ядра;

— приложение для шифрования файлов.

2.1 Перехват сообщений

Для перехвата сообщений добавление нового USB устройства и удаление USB устройства необходимо в загружаемом модуле ядра разместить уведомитель, принимающий в качества параметра функцию обратного вызова нашей обработки данного события.

Для этого была создана следующая структура представленная в листинге 2.1.

Листинг 2.1 — Структура usb_notify

```
1 static struct notifier_block usb_notify = {
2     .notifier_call = notify,
3 };
```

В этой структуре содержится указатель на прототип нашей функции обработки:

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

Для создания уведомителя передаем созданную структуру в функцию:

usb_register_notify(&usb_notify);

Для удаления уведомителя передаем структуру в функцию:

usb_unregister_notify(&usb_notify);

2.2 Хранение информации

Для хранения информации о подключенных USB устройствах создадим структуру, листинг 2.2.

Листинг 2.2 — Структура our_usb_device

```
1 typedef struct our_usb_device {
2     struct usb_device_id dev_id;
3     struct list_head list_node;
4 } our_usb_device_t;
```

Инициализируем список: *LIST_HEAD(connected_devices);*

Для добавления нового подключенного устройства используется функция А.3, для удаления – А.4.

2.3 Алгоритм работы функции-обработчика

На рисунке 2.1 представлен алгоритм работы функции обратного вызова добавления или удаления USB устройства.

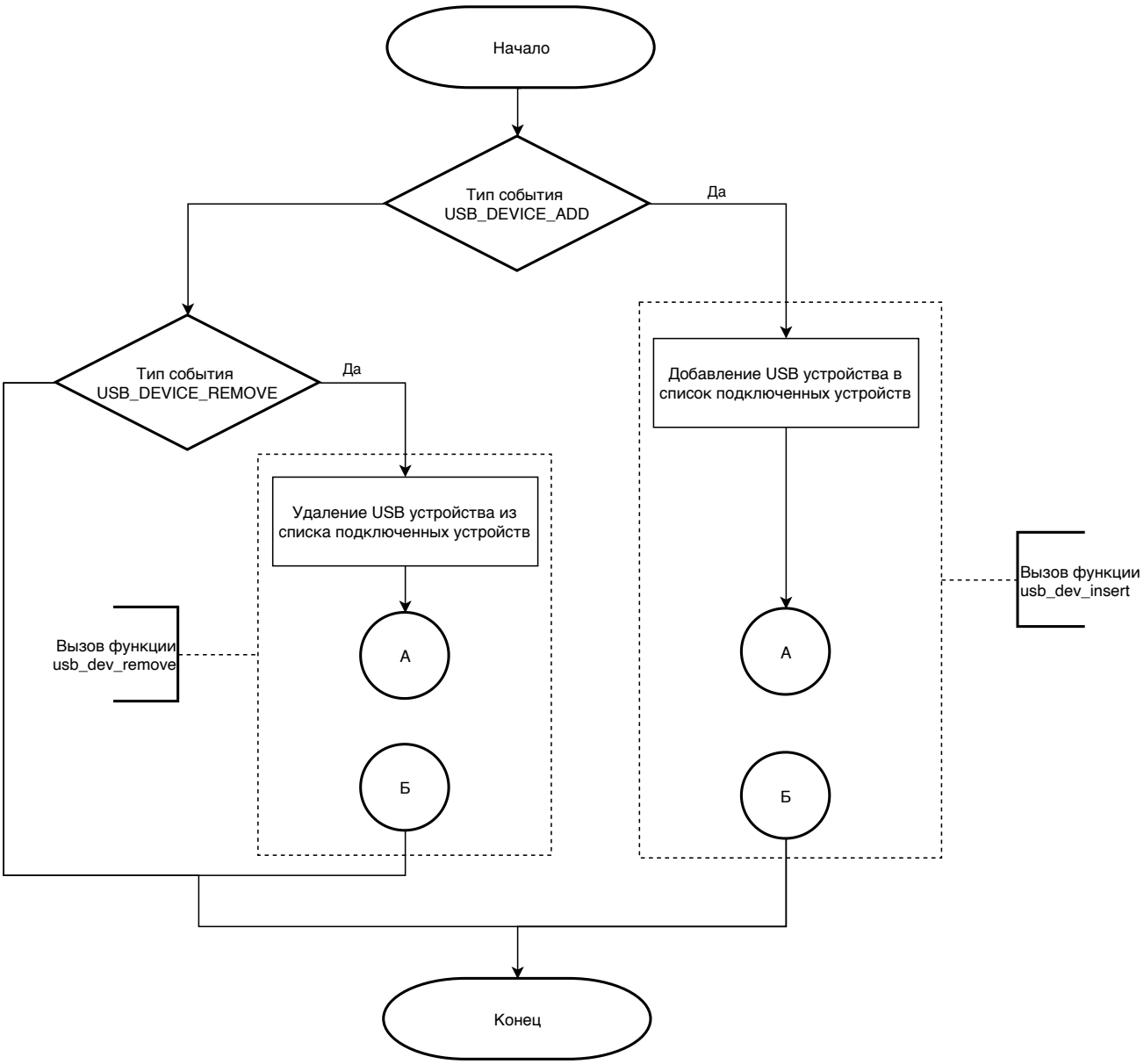

Рисунок 2.1 — Алгоритм работы функции-обработчика.

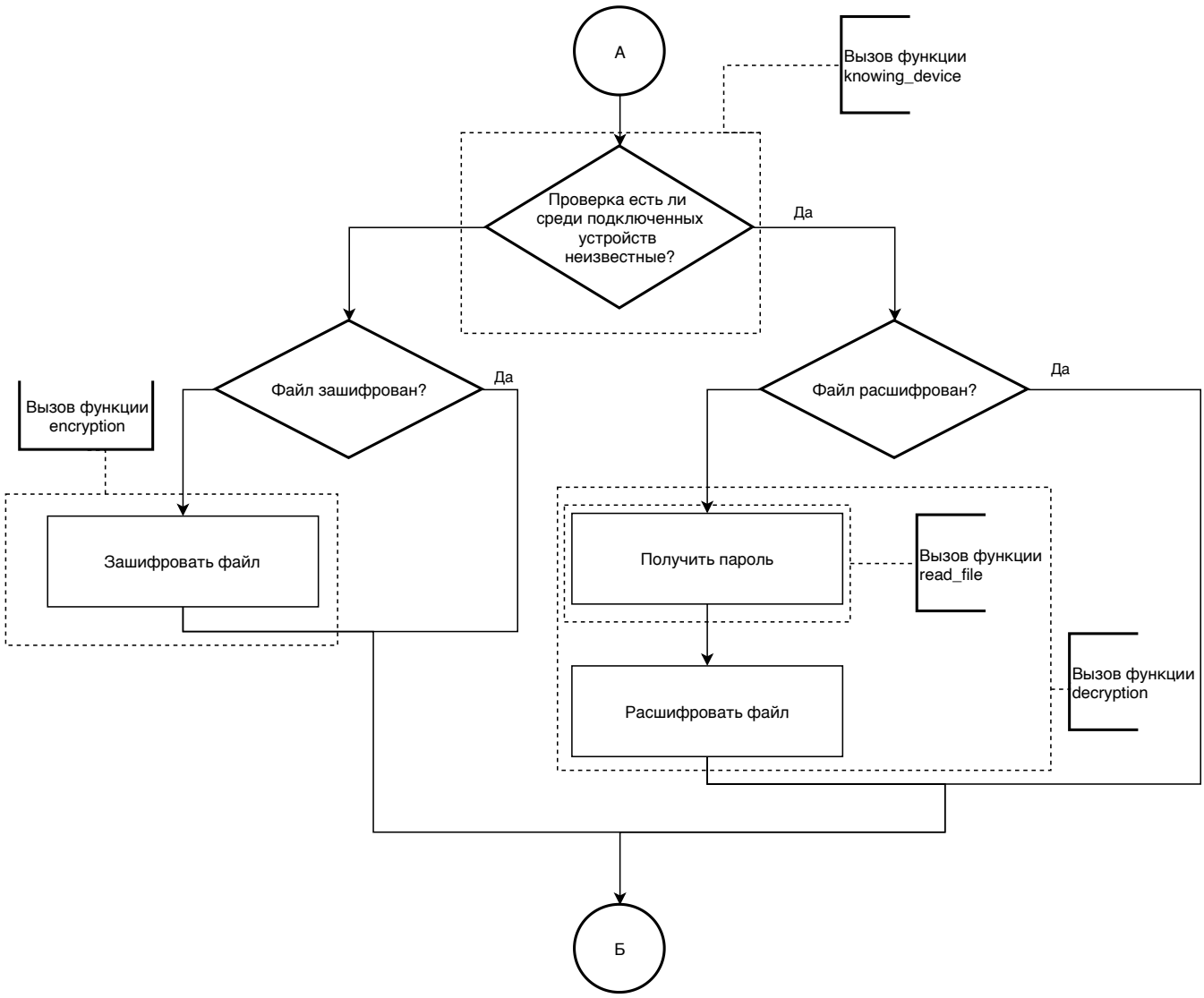

Рисунок 2.2 — Алгоритм работы функции-обработчика.

2.4 Алгоритм шифрования файла

Для каждого файла из списка секретных.

1. Побайтовое считывание символов из файла.
2. Применение операции XOR для данных с паролем.
3. Побайтовая запись символов в файл.

2.5 Вывод

Структура программного обеспечения разработана, переход к реализации.

3 Технологический раздел

В соответствии с выбранной задачей – реализация загружаемого модуля ядра для отслеживания USB-устройств, являющихся ключом для доступа к приложению. Необходимо выбрать средства реализации, создать модули и интерфейс, описать ограничения и порядок работы программы.

3.1 Выбор технологий

Для реализации был выбран язык программирования C. Компилятор – gcc. Для облегчения сборки был написан Makefile, позволяющий запускать сборку одной командой, листинг А.5.

3.2 Хранение данных

Параметры USB устройств, идентификатор поставщика и изделия, а также список секретных файлов и приложений хранятся в конфигурационном файле. Пример конфигурационного файла USB устройств представлен в листинге А.6. Конфигурационный файл секретных файлов и приложений – листинг А.7.

Пароль для доступа к зашифрованным данным хранится на разрешенном USB устройстве в файле *password.txt*.

3.3 Загружаемый модуль

Цель работы создание загружаемого модуля, реализация загрузки и удаления представлена в листинге А.8.

После компиляции загружаемого модуля объектный файл может быть загружен в ядро с помощью команды *insmod* с правами суперпользователя, для выгрузки используется команда *rmmod*.

3.4 Функция-обработчик

В листинге А.9 представлена реализация функции обратного вызова добавления или удаления USB устройства *static int notify(struct notifier_ block *self, unsigned long action, void *dev).*

С последующим вызовом, в зависимости от события *static void usb_ dev_ remove(struct usb_ device *dev)*, *static void usb_ dev_ insert(struct usb_ device *dev).*

3.5 Проверка принадлежности устройства известным

Чтобы узнать можно ли расшифровать файл, необходимо узнать принадлежит ли устройство списку разрешенных устройств. Каждое устройство имеет уникальную пару идентификатор поставщика и идентификатор изделия, по ней и будет происходит поиск. Также в известных устройствах хранится файл с паролем для расшифровки секретных данных.

Реализация данной проверки представлена в листинге А.10.

Считывание пароля представлено в листинге А.11.

3.6 Проверка принадлежности устройства известным

После проверки принадлежности, при необходимости вызываются функции шифровки и расшифровки файлов, которые вызывают исполняемый файл пользовательского пространства.

Реализация этих функций представлена в листинге А.12.

3.7 Пример вывода в dmesg

На рисунке представлен пример работы загружаемого модуля.

Рисунок 3.1 — Пример работы загружаемого модуля.

3.8 Вывод

Реализовано спроектированное ПО, представлены результаты.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы выполнены следующие задачи.

1. Определены основные понятия, такие как загружаемый модуль ядра, уведомления и уведомители. Рассмотрены структуры usb_device, usb_device_id.

2. Разработаны алгоритмы работы функции-обработчика и шифрования файлов.

3. Реализован загружаемого модуля.

Достигнута цель проекта – реализация загружаемого модуля ядра для отслеживания USB-устройств, являющихся ключом для доступа к приложению.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Утечки данных 2019: статистика. — Режим доступа: https://vc.ru/services/103616-utechki-dannyh-2019-statistika-tendencii-kiberbezopasnosti-i- (дата обращения: 10.12.2020).

2. Анатомия загружаемых модулей ядра Linux. — Режим доступа: https://www.ibm.com/developerworks/ru/library/l-lkm/index.html" (дата обращения: 10.12.2020).

3. Notification Chains in Linux Kernel. — Режим доступа: https://0xax.gitbooks.io/linux-insides/content/Concepts/linux-cpu-4.html (дата обращения: 10.12.2020).

4. include/linux/usb.h. — Режим доступа: https://elixir.bootlin.com/linux/latest/source/include/linux/usb.h#L2020 (дата обращения: 10.12.2020).

5. Doubly Linked Lists. — Режим доступа: https://www.kernel.org/doc/html/v4.14/core-api/kernel-api.html (дата обращения: 10.12.2020).

6. Invoking user-space applications from the kernel. — Режим доступа: https://developer.ibm.com/technologies/linux/articles/l-user-space-apps/ (дата обращения: 10.12.2020).

7. Reading and writing of files in Linux kernel driver. — Режим доступа: https://www.programmersought.com/article/83015124510/ (дата обращения: 10.12.2020).

ПРИЛОЖЕНИЕ А

ЛИСТИНГИ

Листинг А.1 — 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;

    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;

    unsigned can_submit:1;
    unsigned persist_enabled:1;
    unsigned have_langid:1;
    unsigned authorized:1;
    unsigned authenticated:1;
    unsigned wusb:1;
```

```
42      unsigned lpm_capable:1;
43      unsigned usb2_hw_lpm_capable:1;
44      unsigned usb2_hw_lpm_besl_capable:1;
45      unsigned usb2_hw_lpm_enabled:1;
46      unsigned usb2_hw_lpm_allowed:1;
47      unsigned usb3_lpm_u1_enabled:1;
48      unsigned usb3_lpm_u2_enabled:1;
49      int string_langid;
50
51      /* static strings from the device */
52      char *product;
53      char *manufacturer;
54      char *serial;
55
56      struct list_head filelist;
57
58      int maxchild;
59
60      u32 quirks;
61      atomic_t urbnum;
62
63      unsigned long active_duration;
64
65  #ifdef CONFIG_PM
66      unsigned long connect_time;
67
68      unsigned do_remote_wakeup:1;
69      unsigned reset_resume:1;
70      unsigned port_is_suspended:1;
71  #endif
72      struct wusb_dev *wusb_dev;
73      int slot_id;
74      enum usb_device_removable removable;
75      struct usb2_lpm_parameters l1_params;
76      struct usb3_lpm_parameters u1_params;
77      struct usb3_lpm_parameters u2_params;
78      unsigned lpm_disable_count;
79
80      u16 hub_delay;
81      unsigned use_generic_driver:1;
82  };
```

Листинг A.2 — 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))));
27  };
```

Листинг А.3 — Добавление usb устройства

```
1  static void add_our_usb_device(struct usb_device *dev)
2  {
3      our_usb_device_t* new_usb_device = (our_usb_device_t
           *)kmalloc(sizeof(our_usb_device_t), GFP_KERNEL);
4      struct usb_device_id new_id = { USB_DEVICE(dev->descriptor.idVendor,
           dev->descriptor.idProduct) };
5      new_usb_device->dev_id = new_id;
6      list_add_tail(&new_usb_device->list_node, &connected_devices);
7  }
```

Листинг А.4 — Удаление usb устройства

```
1  static void delete_our_usb_device(struct usb_device *dev)
2  {
3      our_usb_device_t *device, *temp;
4      list_for_each_entry_safe(device, temp, &connected_devices, list_node)
5      {
6          if (device_match_device_id(dev, &device->dev_id))
7          {
8              list_del(&device->list_node);
```

```
9                 kfree(device);
10            }
11        }
12    }
```

Листинг A.5 — Makefile

```
1  ifneq ($(KERNELRELEASE),)
2      obj-m := md.o
3  else
4      CURRENT = $(shell uname -r)
5      KDIR = /lib/modules/$(CURRENT)/build
6      PWD = $(shell pwd)
7
8  default:
9      $(MAKE) -C $(KDIR) M=$(PWD) modules
10
11 clean:
12     rm -rf .tmp_versions
13     rm *.ko
14     rm *.o
15     rm *.mod.c
16     rm *.symvers
17     rm *.order
18
19 endif
```

Листинг A.6 — Конфигурационный файл USB устройств

```
1  struct known_usb_device {
2      struct usb_device_id dev_id;
3      char *name;
4  };
5
6  // List of all USB devices you know
7  static const struct known_usb_device known_devices[] = {
8      { .dev_id = { USB_DEVICE(0x058f, 0x6387) }, .name = "SAG" },
9  };
```

Листинг A.7 — Конфигурационный файл секретных файлов и приложений

```
1    static char *secret_apps[] = {
2      "/home/parallels/Desktop/Operating_systems_coursework/file.txt",
3      "/home/parallels/Desktop/Operating_systems_coursework/xor",
4      "/usr/bin/firefox",
5      NULL,
6  };
```

Листинг А.8 — Загрузка и удаление модуля ядра

```
static int __init my_module_init(void)
{
    usb_register_notify(&usb_notify);
    call_encryption();
    printk(KERN_INFO "USB MODULE: loaded.\n");
    return 0;
}

static void __exit my_module_exit(void)
{
    usb_unregister_notify(&usb_notify);
    printk(KERN_INFO "USB MODULE: unloaded.\n");
}

module_init(my_module_init);
module_exit(my_module_exit);
```

Листинг А.9 — Функция-обработчик

```
// If usb device inserted.
static void usb_dev_insert(struct usb_device *dev)
{
    add_our_usb_device(dev);
    char *name = knowing_device();

    if (name)
    {
        if (state_encrypt)
            call_decryption(name);
        state_encrypt = false;
        printk(KERN_INFO "USB MODULE: New device we can encrypt.\n");
    }
    else
   {
        if (!state_encrypt)
            call_encryption();
        state_encrypt = true;
        printk(KERN_INFO "USB MODULE: New device, we can't encrypt.\n");
    }
}

// If usb device removed.
static void usb_dev_remove(struct usb_device *dev)
{
    delete_our_usb_device(dev);
    char *name = knowing_device();
```

```
28
29      if (name)
30      {
31          if (state_encrypt)
32              call_decryption(name);
33          state_encrypt = false;
34          printk(KERN_INFO "USB MODULE: Delete device, we can encrypt.\n");
35      }
36      else
37      {
38          if (!state_encrypt)
39              call_encryption();
40          state_encrypt = true;
41          printk(KERN_INFO "USB MODULE: Delete device, we can't encrypt.\n");
42      }
43  }
44
45  // New notify.
46  static int notify(struct notifier_block *self, unsigned long action, void
        *dev)
47  {
48      // Events, which our notifier react.
49      switch (action)
50      {
51          case USB_DEVICE_ADD:
52              usb_dev_insert(dev);
53              break;
54          case USB_DEVICE_REMOVE:
55              usb_dev_remove(dev);
56              break;
57          default:
58              break;
59      }
60      return 0;
61  }
```

Листинг A.10 — Функции для проверки разрешенных устройств

```
1  // Match device id with device id.
2  static bool device_id_match_device_id(struct usb_device_id *new_dev_id,
       const struct usb_device_id *dev_id)
3  {
4      // Check idVendor and idProduct, which are used.
5      if (dev_id->idVendor != new_dev_id->idVendor)
6          return false;
7      if (dev_id->idProduct != new_dev_id->idProduct)
8          return false;
```

```
9       return true;
10  }
11
12  // Check our list of devices, if we know device.
13  static char *usb_device_id_is_known(struct usb_device_id *dev)
14  {
15      unsigned long known_devices_len = sizeof(known_devices) /
            sizeof(known_devices[0]);
16      int i = 0;
17      for (i = 0; i < known_devices_len; i++)
18      {
19          if (device_id_match_device_id(dev, &known_devices[i].dev_id))
20          {
21              int size = sizeof(known_devices[i].name);
22              char *name = (char *)kmalloc(size + 1, GFP_KERNEL);
23              int j = 0;
24              for (j = 0; j < size; j++)
25                  name[j] = known_devices[i].name[j];
26              name[size + 1] = '\0';
27
28              return name;
29          }
30      }
31      return NULL;
32  }
33
34  static char *knowing_device(void)
35  {
36      our_usb_device_t *temp;
37      int count = 0;
38      char *name;
39
40      list_for_each_entry(temp, &connected_devices, list_node) {
41          name = usb_device_id_is_known(&temp->dev_id);
42          if (!name)
43              return NULL;
44          count++;
45      }
46      if (0 == count)
47          return NULL;
48      return name;
49  }
```

Листинг A.11 — Считывание пароля из файла USB устройства

```
1   static char *read_file(char *filename)
2   {
```

```
3       struct kstat *stat;
4       struct file *fp;
5       mm_segment_t fs;
6       loff_t pos = 0;
7       char *buf;
8       int size;
9
10      fp = filp_open(filename, O_RDWR, 0644);
11      if (IS_ERR(fp))
12      {
13          return NULL;
14      }
15
16      fs = get_fs();
17      set_fs(KERNEL_DS);
18
19      stat = (struct kstat *)kmalloc(sizeof(struct kstat), GFP_KERNEL);
20      if (!stat)
21      {
22          return NULL;
23      }
24
25      vfs_stat(filename, stat);
26      size = stat->size;
27
28      buf = kmalloc(size, GFP_KERNEL);
29      if (!buf)
30      {
31          kfree(stat);
32          return NULL;
33      }
34
35      kernel_read(fp, buf, size, &pos);
36
37      filp_close(fp, NULL);
38      set_fs(fs);
39      kfree(stat);
40      buf[size]='\0';
41      return buf;
42  }
```

Листинг А.12 — Функции вызывающие исполняемый файл пользовательского пространства

```
1   static int call_decryption(char *name_device) {
2       printk(KERN_INFO "USB MODULE: Call_decrypt\n");
3
```

```
4       char path[80];
5       strcpy(path, USB_FOLDER);
6       strcat(path, name_device);
7       strcat(path, "/");
8       strcat(path, PASSWORD_FILE);
9       char *data = read_file(path);
10
11      char *argv[] = {
12       "/home/parallels/Desktop/Operating_systems_coursework/crypto",
13      data,
14      NULL };
15
16      static char *envp[] = {
17      "HOME=/",
18      "TERM=linux",
19      "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
20       NULL };
21
22      if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC) < 0)
23      {
24          return -1;
25      }
26
27      return 0;
28  }
29
30  static int call_encryption(void) {
31      printk(KERN_INFO "USB MODULE: Call_encrypt\n");
32      char *argv[] = {
33       "/home/parallels/Desktop/Operating_systems_coursework/crypto",
34      NULL };
35
36      static char *envp[] = {
37      "HOME=/",
38      "TERM=linux",
39      "PATH=/sbin:/bin:/usr/sbin:/usr/bin",
40      NULL };
41
42      if (call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC) < 0)
43      {
44          return -1;
45      }
46
47      return 0;
48  }
```