|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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, сопровождающий нажатия клавиш USB-мыши звуковыми сигналами»***

Студент \_\_\_\_\_\_ИУ7-75Б\_\_\_\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**О.С. Платонова**\_\_\_\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Н. Ю. Рязанова\_\_\_\_**\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*Москва, 2021 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)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по дисциплине \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Операционные системы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Загружаемый модуль ядра, сопровождающий нажатия клавиш USB-мыши звуковыми\_\_ сигналами.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Тема курсового проекта)

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Платонова О. С. гр. ИУ7-75Б\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, инициалы, индекс группы)

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

***1. Техническое задание***

Разработать загружаемый модуль ядра Linux, который устанавливает звуковые сигналы на клавиши USB-мыши, чтобы при нажатии воспроизводились звуки. Звук должен сопровождать нажатие и отпускание клавиш мыши. Реализовать для левой, правой и средней клавиш.

***2. Оформление курсового проекта***

2.1. Расчетно-пояснительная записка на 25-30 листах формата А4.

Расчетно-пояснительная записка должна содержать: введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.

2.2. Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

На защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс.

Дата выдачи задания «\_\_ \_\_» \_\_ \_\_ 2021 г.

**Руководитель курсового проекта**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_Н. Ю. Рязанова\_\_\_\_

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**О. С. Платонова**\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Оглавление

[Введение 4](#_Toc90723060)

[I. Аналитический раздел 5](#_Toc90723061)

[1.1. Постановка задачи 5](#_Toc90723062)

[1.2. Загружаемый модуль ядра 5](#_Toc90723063)

[1.3. USB драйвер 5](#_Toc90723064)

[1.4. USB ядро 6](#_Toc90723065)

[1.5. Конечная точка 7](#_Toc90723066)

[1.6. USB передачи данных 7](#_Toc90723067)

[II. Конструкторский раздел 9](#_Toc90723068)

[2.1. Требования к программному обеспечению 9](#_Toc90723069)

[2.2. Реализация драйвера 9](#_Toc90723070)

[2.3. Регистрация драйвера 9](#_Toc90723071)

[2.4. Обработка сообщений от устройства 11](#_Toc90723072)

[2.5. Дополнительный поток ядра 11](#_Toc90723073)

[2.6. Схемы алгоритмов работы функции обработки событий 14](#_Toc90723074)

[III. Технологический раздел 17](#_Toc90723075)

[3.1. Язык программирования и среда разработки 17](#_Toc90723076)

[3.2. Описание ключевых моментов реализации 17](#_Toc90723077)

[3.3. Makefile 22](#_Toc90723078)

[3.4. Привязка драйвера 22](#_Toc90723079)

[3.5. Результат выполнения 24](#_Toc90723080)

[Заключение 25](#_Toc90723081)

[Литература 26](#_Toc90723082)

[Приложение А 27](#_Toc90723083)

# Введение

Работа за компьютером для слабовидящих людей может вызывать большие трудности. Воспроизведение звукового сигнала по нажатию может служить хорошим дополнением к существующим приложениям для данной категории пользователей (дисплей Брайля, синтезатор речи).

Также разработанное ПО может быть использовано в образовательном процессе для детей и школьников. Звуковой сигнал позволит выполнять работу за компьютером в игровом режиме, тем самым облегчая процесс обучения и повышая к нему интерес.

Так как единственный существующий подход – загружаемый модуль ядра, то данная работа будет основана на анализе и модификации системного драйвера USB-мыши, который решает поставленную задачу частично.

# Аналитический раздел

## Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать и реализовать загружаемый модуль ядра для сопровождения нажатия клавиш USB-мыши звуковыми сигналами. В ходе реализации поставленного задания необходимо решить следующие задачи:

* проанализировать системный драйвер мыши;
* разработать загружаемый модуль, сопровождающий нажатие клавиш мыши звуковыми сигналами;
* реализовать программное обеспечение;
* выполнить тестирование разработанного модуля.

Разрабатываемое ПО должно позволять выполнять сопровождение нажатия и отпускания левой, средней и правой клавиш мыши.

## Загружаемый модуль ядра

Единственным способом решения поставленной задачи является загружаемый модуль ядра. Ядро Linux относится к классу монолитных. В архитектуре данного класса прикладные приложения выполняются посредством создания отдельной ветки кода в пространстве ядра. Поскольку в ранних версиях расширение функциональности требовало перекомпиляции ядра, что было недопустимо для систем промышленного уровня, позднее была добавлена технология модуля ядра.

## USB драйвер

USB (англ. Universal Serial Bus, Универсальная последовательная шина) является соединением между компьютером и несколькими периферийными устройствами [2]. Несмотря на то, что реализация шины очень проста, она имеет следующую отличительную особенность: USB является каналом связи между устройством и хостом и может запросить фиксированный канал пропускания для передачи данных (необходим для надежной передачи аудио и видео).

В ОС Linux реализована поддержка двух основных типов USB драйверов: драйверы на устройстве и драйверы на хост-системе. Последние управляют USB-устройствами, которые подключены к соответствующей хост-системе [3].

## USB ядро

В ядре Linux реализована подсистема, которая называется USB ядром (англ. USB core), созданная для поддержки USB устройств и контроллеров шины USB [4]. Драйверы основного ядра обращаются к прикладным интерфейсам USB ядра. В тоже время принято выделять два основных публичных прикладных интерфейса: один — реализует взаимодействие с драйверами общего назначения (символьное устройство), другой — взаимодействие с драйверами, являющимися частью ядра (драйвер хаба). Второй тип драйверов участвует в управлении USB шиной. На рисунке 1 представлена описанная выше конфигурация.

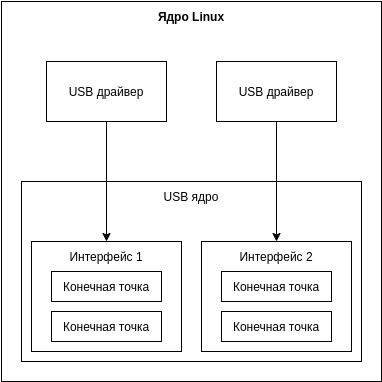


Рисунок 1. USB ядро.

## Конечная точка

Как видно из рисунка, интерфейс USB ядра состоит из так называемых конечных точек (англ. endpoint). Конечная точка является формой USB взаимодействия, способной переносить данные только в одном направлении (аналогия с однонаправленным каналом). Соответственно, точка IN переносит данные от устройства на хост-систему, OUT — с хост-системы на устройство. Драйвер мыши имеет только 1 конечную точку типа прерывания. Для конечных точек данного типа характерна передача небольшого объема данных с фиксированной частотой. Являются основными для клавиатуры и мыши. Передачи данного типа имеют зарезервированную пропускную способность. Также выделяют точки типа управляющая, поточная и изохронная. Управляющие и поточные точки используются для асинхронной передачи данных, когда драйвер решает их использовать. Используют всю пропускную способность. Изохорные точки не могут гарантировать доставку данных и должны обеспечивать заданную пропускную способность, однако они способны поддерживать обмен большого объема данных. Применяются для сбора данных в реальном времени (аудио, видео) [3].

## USB передачи данных

***Блоки запроса USB***

Для обмена данными между заданной конечной точкой USB и USB устройством в асинхронном режиме используется URB (англ. USB request block). URB содержит всю необходимую информация для выполнения USB-транзакции и доставки данных и статуса. Каждая конечная точка в устройстве может обрабатывать очередь из URB.

Жизненный цикл URB можно разделить на два этапа:

*I. Инициализация драйвером USB:*

Создание, назначение в определенную конечную точку устройства, передача в USB ядро.

*II. Обработка драйвером контроллера USB узла:*

Передача USB ядром в заданный драйвер контроллера USB узла, обработка драйвером, который выполняет передачу по USB в устройство. После завершения работы с URB драйвер уведомляет драйвер USB устройства.

***Передача без URB***

В Linux допустимы реализации USB драйверов, при которых отправка и прием данных происходит без создания и инициализации URB. Для этого в системе определены следующие функции:

1. *usb\_bulk\_msg*. Создает потоковый URB и отправляет его в указанное устройство. Перед возвратом к вызывающему устройству ожидает завершения URB.
2. *usb\_control\_msg*. Работа данной функции аналогична *usb\_bulk\_msg*, однако в данной реализации драйверу доступны отправка и получение управляющих сообщений USB [5].

# Конструкторский раздел

## Требования к программному обеспечению

Программное обеспечение состоит из драйвера, реализованного в виде загружаемого модуля ядра, который посредством создания потока ядра обрабатывает нажатия клавиш мыши и воспроизводит звуковые сигналы.

## Реализация драйвера

В основе данного ПО лежит реализация исходного USB драйвера мыши. Его работа заключается в регистрации своего объекта драйвера с USB подсистемой.

## Регистрация драйвера

Основной структурой, которую создают все драйверы, является *struct* *usb\_driver* (определена в /include/linux/usb.h). Эта структура создается и инициализируется драйвером посредством ряда функций. [5]

Рисунок 2. Структура usb\_driver.

Для регистрации USB драйвера мыши следует рассматривать следующие поля структуры:

* *name* — имя драйвера, должно быть уникальным в пространстве USB драйверов;
* *probe* — указатель на функцию, выполняющую проверку устройства, описанного структурой *usb\_interface*. В случае соответствия драйвера и указанного устройства, драйвер также инициализирует локальные структуры, необходимые для управления USB устройством. Так, в данной работе, в случае успешной проверки, будет выполнена инициализация структур *usb\_mouse*, *input\_dev*, выделена и проинициализирована память URB, создан поток ядра;
* *disconnect* — указатель на функцию, которая вызывается, когда USB интерфейс более не доступен (отключено устройство или выгружен модуль ядра). Внутри указанной функции выполняется освобождение памяти и отмена регистрации устройства;
* *id\_table* — структура *usb\_device\_id*, описывающая таблицу идентификаторов USB драйверов, необходимую для быстрого подключения устройств. В случае ее отсутствия, функция *probe* не сможет быть вызвана.

После инициализации структуры *usb\_device\_id* выполняется вызов макроса *MODULE\_DEVICE\_TABLE*. При компиляции процесс извлекает информацию из всех драйверов и инициализирует таблицу устройств. При подключении устройства, ядро обращается к таблице, где выполняется поиск записи, соответствующей идентификатору устройства. В случае нахождения такой записи, выполняется инициализация и загрузка модуля.

## Обработка сообщений от устройства

Также внутри драйвера реализована функция *usb\_mouse\_irq*, позволяющая обрабатывать сообщения, отправленные устройством.

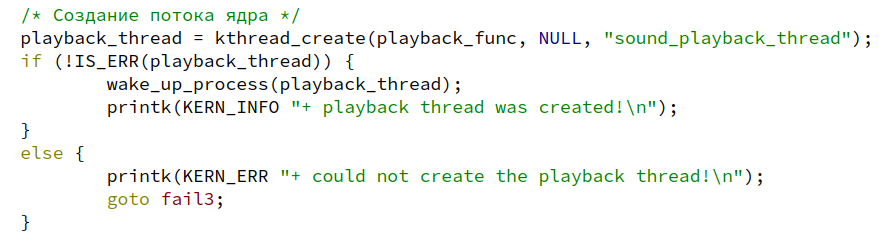


Внутри данной функции следует выполнить обновление текущего статуса кнопок мыши посредством вызова функции *set\_mouse\_status*. В драйвере мыши информация о нажатой кнопке хранится в поле data[0].

## Дополнительный поток ядра

***Создание потока***

В функции *probe* при успешной проверке подключенного устройства и инициализации локальных структур, следует выполнить создание потока ядра, задача которого асинхронно с помощью пользовательской функции выполнять воспроизведения звуковых сигналов. Необходимость создания дополнительного потока объясняется требованием воспроизведения музыки как фонового процесса, т. е. после нажатия клавиши мыши пользователь может продолжить работу, не дожидаясь окончания воспроизведения.



***Функция потока***

Получив сигнал от устройства, поток выполняет воспроизведение сигнала функцией *call\_usermodehelper.*

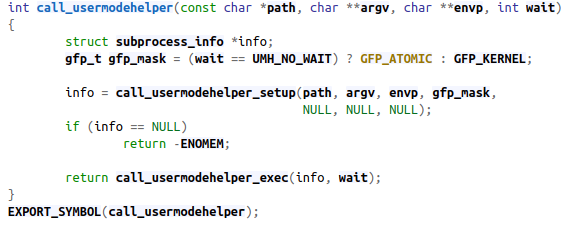
*.*

Рисунок 3. Функция *call\_usermodehelper.*

Данная функция выполняет подготовку и запуск приложения в пользовательском пространстве из режима ядра. Ее работа эквивалентна работе функции *execive*() в пользовательском режиме.

На вход функции подаются следующие аргументы:

* *path* — путь к исполняемому файлу;
* *argv* — вектор аргументов для процесса;
* *envp* — среда процесса;
* *wait* — указание процессу дождаться завершения приложения и вернуть статус.

В данной работе функция вызывается с параметром *UMH\_NO\_WAIT*, указывающим процессу не дожидаться окончания завершения. При такой работе не может быть получен результат исполнения. Это делает безопасным вызов из контекста прерывания.

На рисунке 3 представлена внутренняя реализация функции [6].



Рисунок 3. Внутренняя реализация API *usermodehelper.*

***Ожидание потока***

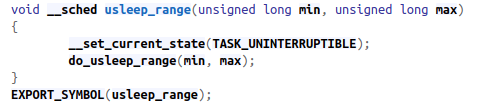
После запуска приложения в пользовательском режиме, поток засыпает, чтобы освободить процессор. Перевод потока в спящий режим осуществляется с помощью функции *usleep\_range*, параметры которой определяются пользователем в микросекундах.

Рисунок 4. Функция *usleep\_range.*

***Завершение работы потока***

Завершение работы потока происходит в функции *disconnect*, вызов которой происходит в случае отключения устройства или выгрузки модуля из ядра. Функция *kthread\_stop* выполняет пробуждение потока и ожидание его завершения.

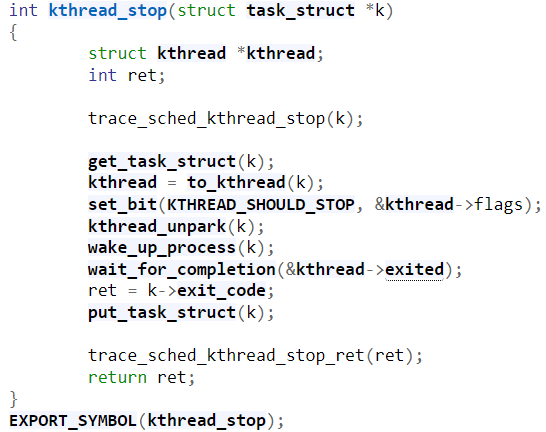


Рисунок 5. Функция *kthread\_stop*.

## Схемы алгоритмов работы функции обработки событий

На рисунке 6 представлена схема алгоритма работы функции *set\_mouse\_status,* выполняющей инициализацию текущего состояния мыши по нажатию клавиши.

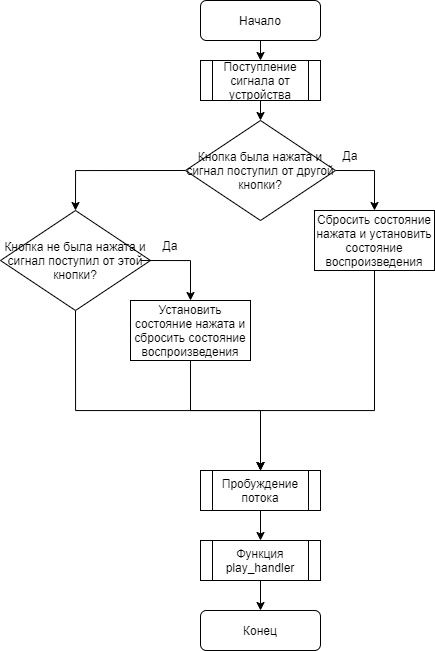


Рисунок 6. Схема алгоритма работы функции *set\_mouse\_status*.

На рисунке 7 представлена схема алгоритма работы функции *play\_handler,* реализованной в дополнительном потоке.

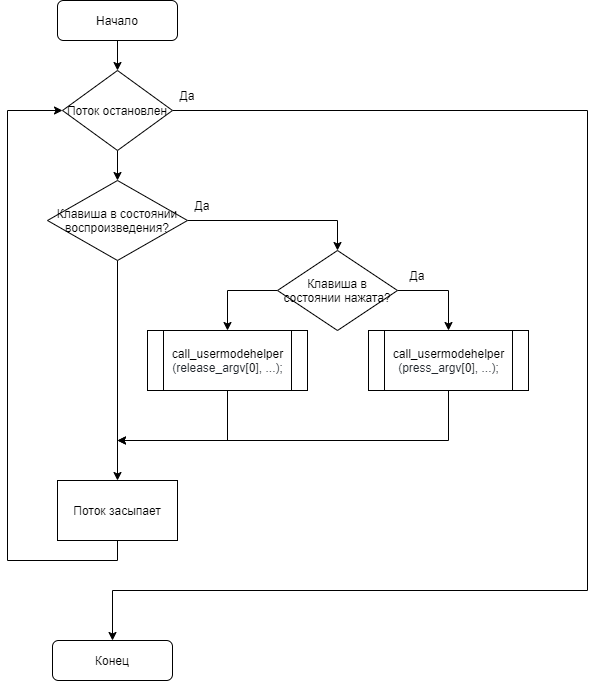


Рисунок 7. Схема алгоритма работы функции *play\_handler.*

# Технологический раздел

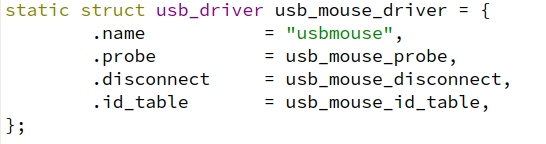
## Язык программирования и среда разработки

В качестве языка программирования для реализации поставленной задачи был выбран язык С. Он является языком реализации большинства модулей и драйверов ОС Linux. В качестве компилятора был использован компилятор gcc. Средой разработки был выбран текстовый редактор SublimeText.

## Описание ключевых моментов реализации

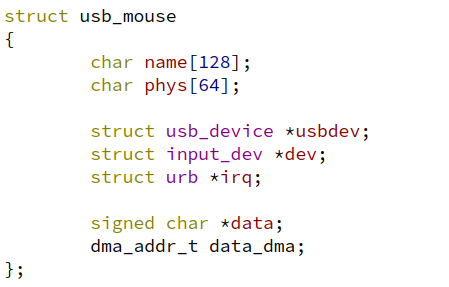
Основной структурой USB драйвера является *struct usb\_driver*.

Листинг 1 – Структура *struct usb\_driver.*



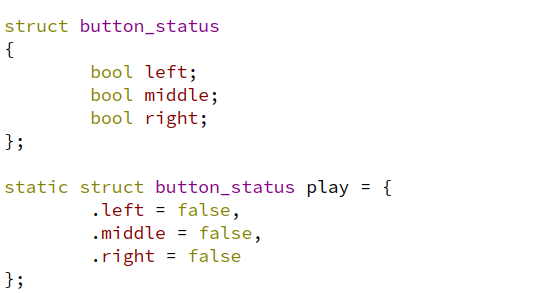
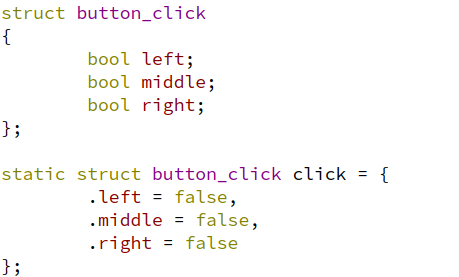
Локальной системной структурой является *usb\_mouse*.

Листинг 2 – Структура *struct* *usb\_mouse.*



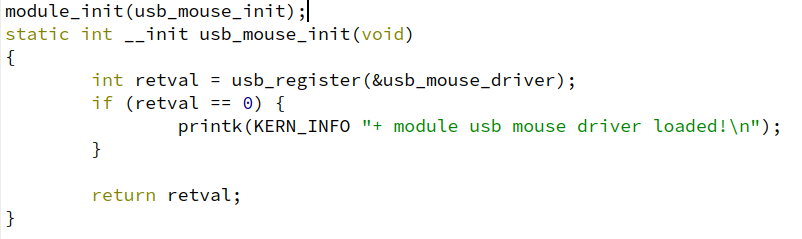
Разработанные структуры данных представлены *struct button\_click* и *struct button\_status.* Они описывают состояние каждой кнопки мыши (нажата/отпущена) и текущий статус звукового сигнала для каждой кнопки (воспроизводится/не воспроизводится).Начальное состояние клавиши: кнопка отпущена, звуковой сигнал не воспроизводится.

Листинг 3 – Структуры *struct button\_click* и *struct button\_status.*



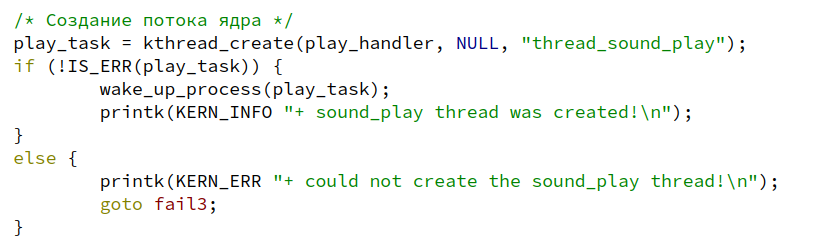
Работа ПО начинается при загрузке модуля в ядро. Функция инициализации модуля выполняет регистрацию *usb\_driver* в ядре.

Листинг 4 – Инициализация модуля.



После подключения USB мыши вызывается функция *probe*, где после успешной проверки соответствия драйвера устройству создается поток ядра.

Листинг 5 – Создание потока ядра.



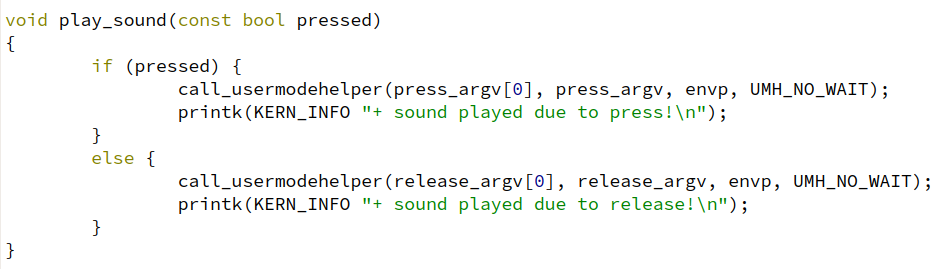
В качестве аргумента функции создания потока передается функция *play\_handler,* которая в зависимости от состояния каждой кнопки выполняет воспроизведение звукового сигнала. После обновления статуса каждой кнопки поток засыпает, тем самым освобождая процессор.

Листинг 6 – Функция *play\_handler.*



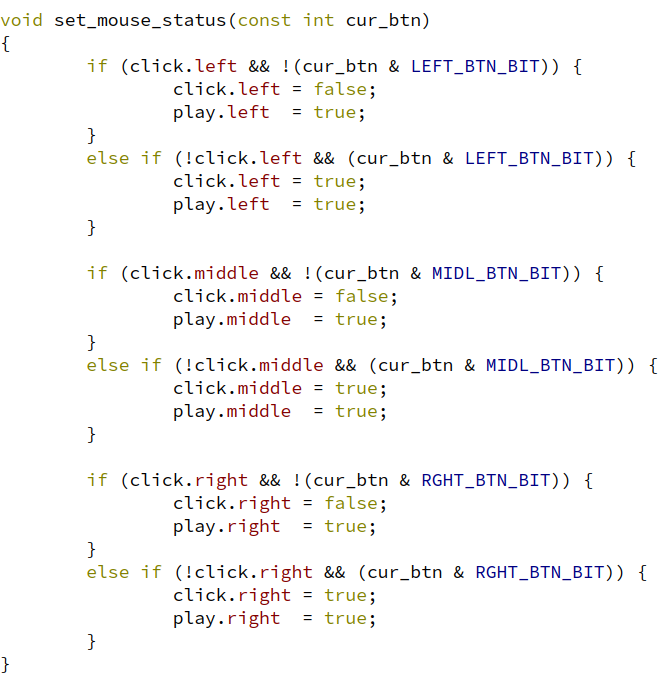
Для воспроизведения звукового сигнала, в функции дополнительного потока выполняется вызов функции *play\_sound.* В зависимости от состояния кнопки из ядра будет вызвана системная функция *call\_usermodehelper*. Если состояние соответствует нажатию, то системной функции на вход передается полный путь файла звукового сигнала нажатия. Аналогично для состояния «отпущена».

Листинг 7 - Функция *play\_sound.*



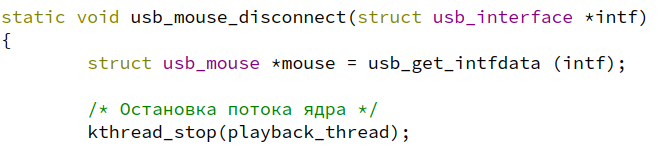
В системной функции *usb\_mouse\_irq*, обрабатывающей сообщения от устройства, выполняется корректировка текущего состояния мыши вызовом функции *set\_mouse\_status*. По нажатию одной из клавиш вызывается обработчик прерывания, в котором содержится информации о номере нажатой клавиши. На основе этого номера и предыдущего состояния формируется новое.

Листинг 8 – функция *set\_mouse\_status.*



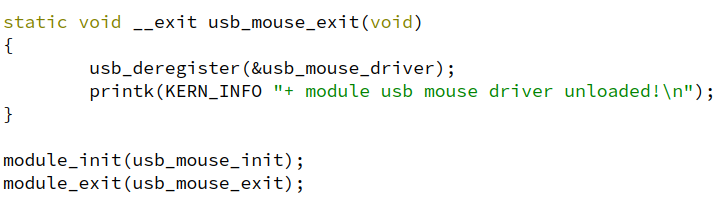
Завершается работа ПО функцией *disconnect*, в которой выполняется остановка потока ядра.

Листинг 9 – остановка потока ядра.



Выгрузка ядра выполняется с помощью макроса *module\_exit,* которыйобращается к функции *usb\_mouse\_exit,* где происходит отмена регистрации драйвера.

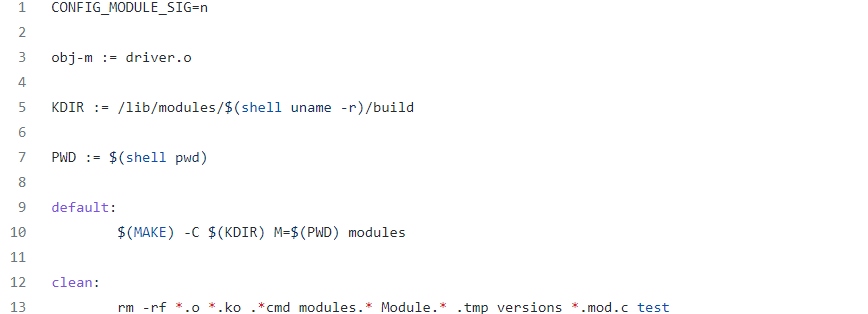
Листинг 10 – Функция, вызываемая при выгрузке ядра.



## Makefile

В листинге 11 приведено содержимое Makefile, содержащего набор инструкций, используемых утилитой make в инструментарии автоматизации сборки.

Листинг 11 – Makefile.



## Привязка драйвера

Для корректного функционирования разработанного ПО необходимо выполнить привязку реализованного драйвера. Привязка драйвера – это процесс связывания устройства с драйвером, который может управлять этим устройством. В ОС Linux определен модуль usbhid (/sys/bus/usb/drivers/usbhid), задача которого – автоматически регистрировать все стандартные драйверы в системе. Данный модуль связывает стандартный драйвер мыши с устройством, не позволяя установить собственный драйвер. С помощью команды *unbind* может быть выполнена отмена привязки драйвера к заданному устройству. Команда *bind* наоборот, выполняет привязку разработанного модуля. Для автоматизации работы ПО данные команды были написаны в bash-файлах.

Листинг 12 – bash-файл привязки собственного драйвера.

#!/bin/sh

# Usage:

# bash bind.sh <device\_id>

echo "Rebinding $1..."

echo -n "$1" > /sys/bus/usb/drivers/usbhid/unbind

echo -n "$1" > /sys/bus/usb/drivers/usbmouse/bind

Для того, что узнать id устройства, следует выполнить команду lsusb, которая отображает список подключенных USB устройств. ID USB устройства имеет следующий формат: Bus-Port1.Port2:SubDevice. На рисунке 6 представлен список подключенных USB устройств до загрузки модуля и после. В данном случае device\_id = 1-2:1.0.

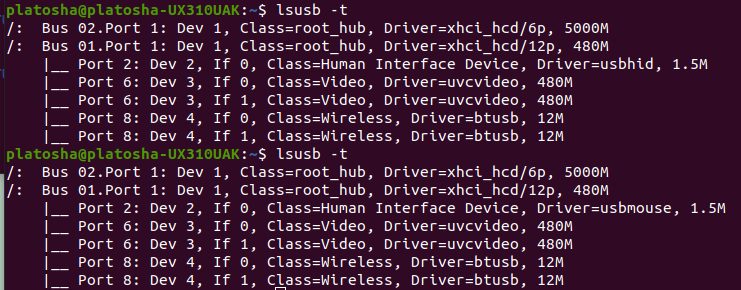


Рисунок 6. Список подключенных USB устройств.

## Результат выполнения

На рисунке 7 приведены этапы сборки и загрузки модуля в ядро.

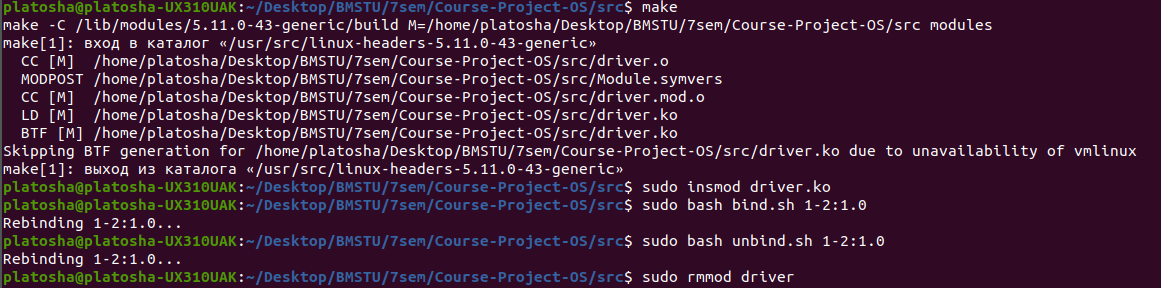


Рисунок 7. Загрузка и выгрузка модуля.

На рисунке 8 приведены сообщения ядра, полученные в результате выполнения ПО.

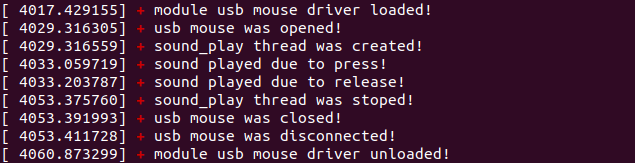


Рисунок 8. Сообщения ядра в результате выполнения ПО.

# Заключение

В результате данной работы были достигнуты поставленные цели и задачи: был выполнен анализ системного драйвера мыши, разработан и реализован загружаемый модуль ядра, выполняющий установку звуковых сигналов на клавиши USB-мыши.

Также в ходе работы был проведен анализ и определена целесообразность использования дополнительного потока ядра. Были разработаны пользовательские структуры, описывающие состояние клавиш мыши.

Было разработано программное обеспечение, представляющее из себя загружаемый модуль, включающий функции воспроизведения звуковых сигналов и анализа текущего состояния мыши. Реализованное ПО показало способность выполнять поставленные задачи.

# Литература

1. Usage of Linux for websites. W3 Techs. [Электронный ресурс]. – URL: https://w3techs.com/technologies/details/os-linux
2. Драйверы устройств. Драйверы USB. [Электронный ресурс]. – URL: http://rus-linux.net/MyLDP/BOOKS/drivers/linux-device-drivers-11.html
3. Corbet J., Rubini A., Kroah-Hartman G. Драйверы устройств Linux, Третья редакиця.
4. Introduction to USB on Linux. [Электронный ресурс]. – URL: https://web.archive.org/web/20090518100552/http://tali.admingilde.org/linux-docbook/usb/ch01.html
5. Исходные коды ядра Linux. [Электронный ресурс]. – URL: http://elixir.free-electrons.com.
6. API ядра Linux. [Электронный ресурс]. – URL: http://rus-linux.net/MyLDP/BOOKS/Moduli-yadra-Linux/kern-mod-index.html

# Приложение А

Листинги модуля

Листинг А.1 – Загружаемый модуль ядра

#include <linux/kernel.h>

#include <linux/slab.h>

#include <linux/module.h>

#include <linux/init.h>

#include <linux/usb/input.h>

#include <linux/hid.h>

#include <linux/kmod.h>

#include <linux/kthread.h>

#include <linux/delay.h>

#include "config.h"

MODULE\_AUTHOR(DRIVER\_AUTHOR);

MODULE\_DESCRIPTION(DRIVER\_DESC);

MODULE\_LICENSE("GPL");

struct usb\_mouse

{

char name[128];

char phys[64];

struct usb\_device \*usbdev;

struct input\_dev \*dev;

struct urb \*irq;

signed char \*data;

dma\_addr\_t data\_dma;

};

struct task\_struct \*play\_task;

struct button\_click

{

bool left;

bool middle;

bool right;

};

static struct button\_click click = {

.left = false,

.middle = false,

.right = false

};

struct button\_status

{

bool left;

bool middle;

bool right;

};

static struct button\_status play = {

.left = false,

.middle = false,

.right = false

};

void play\_sound(const bool pressed)

{

if (pressed) {

call\_usermodehelper(press\_argv[0], press\_argv, envp, UMH\_NO\_WAIT);

printk(KERN\_INFO "+ sound played due to press!\n");

}

else {

call\_usermodehelper(release\_argv[0], release\_argv, envp, UMH\_NO\_WAIT);

printk(KERN\_INFO "+ sound played due to release!\n");

}

}

static int play\_handler(void \*arg)

{

while (!kthread\_should\_stop()) {

if (play.left) {

play\_sound(click.left);

play.left = false;

}

if (play.middle) {

play\_sound(click.middle);

play.middle = false;

}

if (play.right) {

play\_sound(click.right);

play.right = false;

}

/\* Поток засыпает, чтобы освободить процессор \*/

usleep\_range(DELAY\_LO, DELAY\_HI);

}

return 0;

}

void set\_mouse\_status(const int cur\_btn)

{

if (click.left && !(cur\_btn & LEFT\_BTN\_BIT)) {

click.left = false;

play.left = true;

}

else if (!click.left && (cur\_btn & LEFT\_BTN\_BIT)) {

click.left = true;

play.left = true;

}

if (click.middle && !(cur\_btn & MIDL\_BTN\_BIT)) {

click.middle = false;

play.middle = true;

}

else if (!click.middle && (cur\_btn & MIDL\_BTN\_BIT)) {

click.middle = true;

play.middle = true;

}

if (click.right && !(cur\_btn & RGHT\_BTN\_BIT)) {

click.right = false;

play.right = true;

}

else if (!click.right && (cur\_btn & RGHT\_BTN\_BIT)) {

click.right = true;

play.right = true;

}

}

static void usb\_mouse\_irq(struct urb \*urb)

{

struct usb\_mouse \*mouse = urb->context;

signed char \*data = mouse->data;

struct input\_dev \*dev = mouse->dev;

int status = 0;

switch (urb->status) {

case 0:

break;

case -ECONNRESET:

case -ENOENT:

case -ESHUTDOWN:

return;

default:

goto resubmit;

}

set\_mouse\_status(data[0]);

input\_report\_key(dev, BTN\_LEFT, data[0] & LEFT\_BTN\_BIT);

input\_report\_key(dev, BTN\_RIGHT, data[0] & RGHT\_BTN\_BIT);

input\_report\_key(dev, BTN\_MIDDLE, data[0] & MIDL\_BTN\_BIT);

input\_report\_key(dev, BTN\_SIDE, data[0] & 0x08);

input\_report\_key(dev, BTN\_EXTRA, data[0] & 0x10);

input\_report\_rel(dev, REL\_X, data[1]);

input\_report\_rel(dev, REL\_Y, data[2]);

input\_report\_rel(dev, REL\_WHEEL, data[3]);

input\_sync(dev);

resubmit:

status = usb\_submit\_urb (urb, GFP\_ATOMIC);

if (status) {

dev\_err(&mouse->usbdev->dev, "can't resubmit intr, %s-%s/input0, status %d\n",

mouse->usbdev->bus->bus\_name,

mouse->usbdev->devpath, status);

}

}

static int usb\_mouse\_open(struct input\_dev \*dev)

{

struct usb\_mouse \*mouse = input\_get\_drvdata(dev);

mouse->irq->dev = mouse->usbdev;

if (usb\_submit\_urb(mouse->irq, GFP\_KERNEL)) {

return -EIO;

}

printk(KERN\_INFO "+ usb mouse was opened!\n");

return 0;

}

static void usb\_mouse\_close(struct input\_dev \*dev)

{

struct usb\_mouse \*mouse = input\_get\_drvdata(dev);

usb\_kill\_urb(mouse->irq);

printk(KERN\_INFO "+ usb mouse was closed!\n");

}

static int usb\_mouse\_probe(struct usb\_interface \*intf, const struct usb\_device\_id \*id)

{

struct usb\_device \*dev = interface\_to\_usbdev(intf);

struct usb\_host\_interface \*interface;

struct usb\_endpoint\_descriptor \*endpoint;

struct usb\_mouse \*mouse;

struct input\_dev \*input\_dev;

int pipe, maxp;

int error = -ENOMEM;

interface = intf->cur\_altsetting;

if (interface->desc.bNumEndpoints != 1) {

return -ENODEV;

}

/\* Получение информации о конечной точке\*/

endpoint = &interface->endpoint[0].desc;

if (!usb\_endpoint\_is\_int\_in(endpoint)) {

return -ENODEV;

}

/\* Получение максимального значения пакетных данных \*/

pipe = usb\_rcvintpipe(dev, endpoint->bEndpointAddress);

maxp = usb\_maxpacket(dev, pipe, usb\_pipeout(pipe));

/\* Аллокация структуры mouse и устройства ввода \*/

mouse = kzalloc(sizeof(struct usb\_mouse), GFP\_KERNEL);

input\_dev = input\_allocate\_device();

if (!mouse || !input\_dev) {

goto fail1;

}

/\* Выделение начальной буфферной памяти USB-данных мыши \*/

mouse->data = usb\_alloc\_coherent(dev, 8, GFP\_ATOMIC, &mouse->data\_dma);

if (!mouse->data) {

goto fail1;

}

/\* Аллокация urb \*/

mouse->irq = usb\_alloc\_urb(0, GFP\_KERNEL);

if (!mouse->irq) {

goto fail2;

}

/\* Заполнение полей структуры mouse \*/

mouse->usbdev = dev;

mouse->dev = input\_dev;

if (dev->manufacturer) {

strlcpy(mouse->name, dev->manufacturer, sizeof(mouse->name));

}

if (dev->product) {

if (dev->manufacturer) {

strlcat(mouse->name, " ", sizeof(mouse->name));

}

strlcat(mouse->name, dev->product, sizeof(mouse->name));

}

if (!strlen(mouse->name)) {

snprintf(mouse->name, sizeof(mouse->name),

"USB HIDBP Mouse %04x:%04x",

le16\_to\_cpu(dev->descriptor.idVendor),

le16\_to\_cpu(dev->descriptor.idProduct));

}

/\* Установка имени пути устройства \*/

usb\_make\_path(dev, mouse->phys, sizeof(mouse->phys));

strlcat(mouse->phys, "/input0", sizeof(mouse->phys));

input\_dev->name = mouse->name;

input\_dev->phys = mouse->phys;

usb\_to\_input\_id(dev, &input\_dev->id);

input\_dev->dev.parent = &intf->dev;

input\_dev->evbit[0] = BIT\_MASK(EV\_KEY) | BIT\_MASK(EV\_REL);

input\_dev->keybit[BIT\_WORD(BTN\_MOUSE)] = BIT\_MASK(BTN\_LEFT) |

BIT\_MASK(BTN\_RIGHT) | BIT\_MASK(BTN\_MIDDLE);

input\_dev->relbit[0] = BIT\_MASK(REL\_X) | BIT\_MASK(REL\_Y);

input\_dev->keybit[BIT\_WORD(BTN\_MOUSE)] |= BIT\_MASK(BTN\_SIDE) |

BIT\_MASK(BTN\_EXTRA);

input\_dev->relbit[0] |= BIT\_MASK(REL\_WHEEL);

input\_set\_drvdata(input\_dev, mouse);

input\_dev->open = usb\_mouse\_open;

input\_dev->close = usb\_mouse\_close;

/\* Инициализация urb \*/

usb\_fill\_int\_urb(mouse->irq, dev, pipe, mouse->data, (maxp > 8 ? 8 : maxp),

usb\_mouse\_irq, mouse, endpoint->bInterval);

mouse->irq->transfer\_dma = mouse->data\_dma;

mouse->irq->transfer\_flags |= URB\_NO\_TRANSFER\_DMA\_MAP;

error = input\_register\_device(mouse->dev);

if (error) {

goto fail3;

}

usb\_set\_intfdata(intf, mouse);

/\* Создание потока ядра \*/

play\_task = kthread\_create(play\_handler, NULL, "thread\_sound\_play");

if (!IS\_ERR(play\_task)) {

wake\_up\_process(play\_task);

printk(KERN\_INFO "+ sound\_play thread was created!\n");

}

else {

printk(KERN\_ERR "+ could not create the sound\_play thread!\n");

goto fail3;

}

return 0;

fail3:

usb\_free\_urb(mouse->irq);

fail2:

usb\_free\_coherent(dev, 8, mouse->data, mouse->data\_dma);

fail1:

input\_free\_device(input\_dev);

kfree(mouse);

return error;

}

static void usb\_mouse\_disconnect(struct usb\_interface \*intf)

{

struct usb\_mouse \*mouse = usb\_get\_intfdata (intf);

/\* Остановка потока ядра \*/

kthread\_stop(play\_task);

printk(KERN\_INFO "+ sound\_play thread was stoped!\n");

usb\_set\_intfdata(intf, NULL);

if (mouse) {

usb\_kill\_urb(mouse->irq);

input\_unregister\_device(mouse->dev);

usb\_free\_urb(mouse->irq);

usb\_free\_coherent(interface\_to\_usbdev(intf), 8, mouse->data, mouse->data\_dma);

kfree(mouse);

}

printk(KERN\_INFO "+ usb mouse was disconnected!\n");

}

static const struct usb\_device\_id usb\_mouse\_id\_table[] = {

{USB\_INTERFACE\_INFO(USB\_INTERFACE\_CLASS\_HID, USB\_INTERFACE\_SUBCLASS\_BOOT,

USB\_INTERFACE\_PROTOCOL\_MOUSE) },

{ } /\* Terminating entry \*/

};

MODULE\_DEVICE\_TABLE (usb, usb\_mouse\_id\_table);

static struct usb\_driver usb\_mouse\_driver = {

.name = "usbmouse",

.probe = usb\_mouse\_probe,

.disconnect = usb\_mouse\_disconnect,

.id\_table = usb\_mouse\_id\_table,

};

static int \_\_init usb\_mouse\_init(void)

{

int retval = usb\_register(&usb\_mouse\_driver);

if (retval == 0) {

printk(KERN\_INFO "+ module usb mouse driver loaded!\n");

}

return retval;

}

static void \_\_exit usb\_mouse\_exit(void)

{

usb\_deregister(&usb\_mouse\_driver);

printk(KERN\_INFO "+ module usb mouse driver unloaded!\n");

}

module\_init(usb\_mouse\_init);

module\_exit(usb\_mouse\_exit);

Листинг А.2 – Заголовочный файл *config.h*

#include <stddef.h>

#define DRIVER\_VERSION "v1.0"

#define DRIVER\_AUTHOR "Platonova Olga"

#define DRIVER\_DESC "USB Mouse Click Sound Driver"

/\* Button codes \*/

#define LEFT\_BTN\_BIT 0x01

#define RGHT\_BTN\_BIT 0X02

#define MIDL\_BTN\_BIT 0x04

/\* Thread delay range \*/

#define DELAY\_LO 1000

#define DELAY\_HI 2000

/\* aplay commands \*/

char \*press\_argv[] = {"/bin/aplay", "/home/platosha/Desktop/BMSTU/7sem/Course-Project-OS/audio/press.wav", NULL };

char \*release\_argv[] = {"/bin/aplay", "/home/platosha/Desktop/BMSTU/7sem/Course-Project-OS/audio/release.wav", NULL };

/\* aplay environment \*/

char \*envp[] = {"HOME=/", "PATH=/sbin:/usr/sbin:/bin:/usr/bin", NULL };

Листинг А.3 – Makefile

CONFIG\_MODULE\_SIG=n

obj-m := driver.o

KDIR := /lib/modules/$(shell uname -r)/build

PWD := $(shell pwd)

default:

$(MAKE) -C $(KDIR) M=$(PWD) modules

clean:

rm -rf \*.o \*.ko .\*cmd modules.\* Module.\* .tmp\_versions \*.mod.c test

Листинг А.4 – bash-файл bind.sh

#!/bin/sh

# Usage:

# bash bind.sh <device\_id>

echo "Rebinding $1..."

echo -n "$1" > /sys/bus/usb/drivers/usbhid/unbind

echo -n "$1" > /sys/bus/usb/drivers/usbmouse/bind

Листинг А.5 – bash-файл unbind.sh

#!/bin/sh

# Usage:

# bash bind.sh <device\_id>

echo "Rebinding $1..."

echo -n "$1" > /sys/bus/usb/drivers/usbmouse/unbind

echo -n "$1" > /sys/bus/usb/drivers/usbhid/bind