



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  
(Группа)

О.С. Платонова  
(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.

**З А Д А Н И Е**  
**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.....	5
I. Аналитический раздел .....	6
1.1. Постановка задачи .....	6
1.2. Загружаемый модуль ядра Linux.....	6
1.3. USB драйвер .....	7
1.4. USB ядро .....	8
1.5. Конечная точка.....	9
1.6. USB передачи данных .....	10
1.7. Поток ядра .....	11
1.8. Запуск процессов пользователя из пространства ядра .....	12
II. Конструкторский раздел.....	13
2.1. Требования к программному обеспечению.....	13
2.2. Регистрация драйвера .....	13
2.3. Подключение мыши .....	14
2.4. Работа потока .....	15
2.5. Обработка сообщений от устройства.....	17
2.6. Схемы алгоритмов работы функции обработки событий .....	17
III. Технологический раздел.....	20
3.1. Выбор языка и среды программирования .....	20
3.2. Описание ключевых моментов реализации .....	20
3.3. Makefile .....	25
3.4. Привязка драйвера .....	25
3.5. Результат выполнения .....	27
Заключение .....	28

Литература .....	29
Приложение А .....	30

## **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 эффективный способ решения – загружаемый модуль ядра, то данная работа будет основана на анализе и модификации системного драйвера USB-мыши, который решает поставленную задачу частично.

## **I. Аналитический раздел**

### **1.1. 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.

### **1.2. Загружаемый модуль ядра Linux**

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 [1]. Модуль ядра может реализовывать драйвер устройства, файловую систему или сетевой протокол.

К преимуществам такого подхода следует отнести сокращение неиспользуемого кода в базовом ядре и, как следствие, уменьшение занимаемой памяти. Недостатком является фрагментация ядра, после загрузки модулей, несмотря на то, что код базового ядра не фрагментируем. Фрагментация ведет к незначительному снижению производительности из-за увеличения пропусков записей TLB. Также использование загружаемого

модуля снижает безопасность системы, поскольку злоумышленники, имея доступ в пространство ядра, могут скрывать процессы и файлы.

### ***Компоненты модуля ядра***

Загрузка модуля осуществляется с помощью команд *insmod* или *modprobe*. Отличие заключается в том, что *modprobe* разрешает зависимости модулей. При загрузке вызывается функция входа, определенная в модуле. Для указания ее назначения используется макрос *module\_init*. При корректной загрузке функция входа возвращает 0. В ином случае, модуль не будет загружен.

Для просмотра списка загруженных модулей и информации об отдельном модуле используются команды *lsmod* и *modinfo* соответственно.

Командой *rmmod* осуществляется выгрузка модуля, при которой вызывается функция выхода. Для нее используется макрос *module\_exit*.

### ***Драйвер устройства***

Одной из разновидностей модуля ядра являются драйверы устройств. В этом случае компонента регистрации драйверов позволяет модулю сообщить ядру о том, что доступен новый драйвер. При этом базовое ядро содержит информацию, организованную в виде таблицы, об известных драйверах и предоставляет набор программ для добавления или удаления информации из этой таблицы.

## **1.3. 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].

#### **1.4. 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.



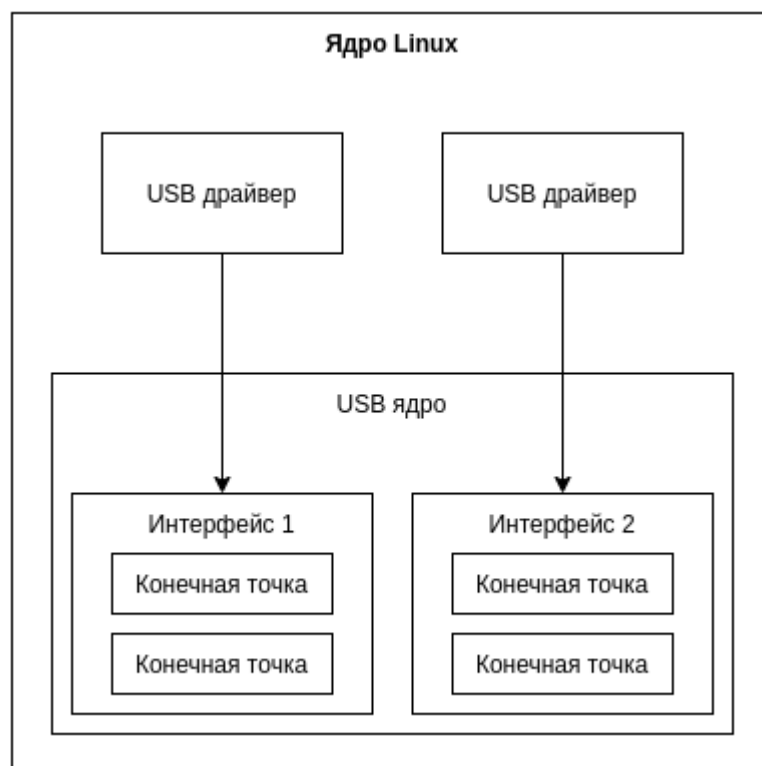


Рисунок 1. USB ядро.

### 1.5. 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].

## **1.6. 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 информацию 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].

## 1.7. Поток ядра

Потоки или нити ядра работают внутри пространства ядра и не имеют своего собственного адресного пространства. Они могут независимо назначаться на выполнение. Потоки используют стандартные механизмы синхронизации ядра, такие как *sleep* или *wakeup*.

Потоки ядра используются для выполнения асинхронного ввода-вывода. Ядро создает поток для обработки запросов каждой такой операции. Потоки также могут быть использованы для обработки прерываний.

### *Переключение потоков*

Переключение потоков является основной задачей планировщика. В ОС Linux механизм планирования основывается на приоритетах. Когда процесс просыпается, ядро устанавливает значение текущего приоритета, равное значению приоритета сна события или ресурса, на котором он был заблокирован. Такой процесс будет назначен на выполнение раньше, чем другие процессы в режиме задачи.

Когда процесс завершил выполнение системного вызова и находится в состоянии возврата в режим задачи, его приоритет сбрасывается обратно в значение текущего приоритета в режиме задачи.

## 1.8. Запуск процессов пользователя из пространства ядра

Использование функции `call_usermodehelper` позволяет выполнять процесс пользователя из процесса ядра. Работа этой функции аналогична работе вызова `exec()` в пространстве пользователя. Особенность данного метода заключается в том, что процесс запускается без управляющего терминала и с нестандартным окружением. Внутренняя реализация функции представлена на рисунке 2 [6].

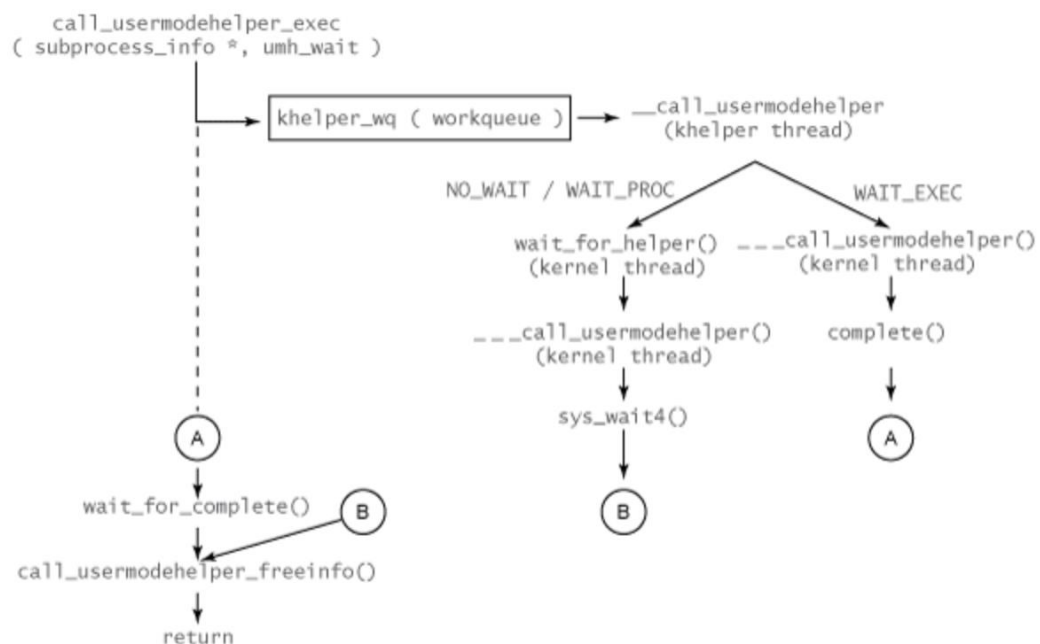


Рисунок 2. Внутренняя реализация функции `call_usermodehelper`.

Функция использует для своей работы структуру `struct subprocess_info`.

```
/linux/include/kmod.h

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;
};
```

## II. Конструкторский раздел

### 2.1. Требования к программному обеспечению

Программное обеспечение состоит из драйвера, реализованного в виде загружаемого модуля ядра, который посредством создания потока ядра сопровождает нажатия клавиш мыши звуковыми сигналами.

### 2.2. Регистрация драйвера

Функция входа, вызываемая при загрузке ядра, выполняет регистрацию 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]

```
struct usb_driver {
    const char *name;

    int (*probe) (struct usb_interface *intf,
                  const struct usb_device_id *id);

    void (*disconnect) (struct usb_interface *intf);

    int (*unlocked_ioctl) (struct usb_interface *intf, unsigned int code,
                           void *buf);

    int (*suspend) (struct usb_interface *intf, pm_message_t message);
    int (*resume) (struct usb_interface *intf);
    int (*reset_resume)(struct usb_interface *intf);

    int (*pre_reset)(struct usb_interface *intf);
    int (*post_reset)(struct usb_interface *intf);

    const struct usb_device_id *id_table;
    const struct attribute_group **dev_groups;

    struct usb_dynids dynids;
    struct usbdrv_wrap drvwrap;
    unsigned int no_dynamic_id:1;
    unsigned int supports_autosuspend:1;
    unsigned int disable_hub_initiated_lpm:1;
    unsigned int soft_unbind:1;
};
```

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, 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.

### 2.3. Подключение мыши

В результате успешной регистрации структуры *usb\_driver* ядро вызывает функцию *probe*, когда установлено устройство. Задача данной функции заключается в проверке информации, переданной ей об устройстве, и принятии решения о том, действительно ли этот драйвер подходит для

указанного устройства. В случае успешной проверки выполняется инициализация локальных структур, которые драйвер использует для управления USB устройством. Инициализация подразумевает заполнение структур необходимой информацией об устройстве. В данной работе извлекается информация об адресе конечной точки и размере буферов, поскольку она необходима для обмена сообщениями с устройством.

Помимо заполнения локальных структур в функции *probe* должен быть создан и проинициализирован надлежащим образом URB для передачи данных на устройство.

## **2.4. Работа потока**

### ***Создание***

В функции *probe* при успешной проверке устройства и инициализации всех 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.

### ***Пробуждение потока***

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

```

int call_usermodehelper(const char *path, char **argv, char **envp, int wait)
{
    struct subprocess_info *info;
    gfp_t gfp_mask = (wait == UMH_NO_WAIT) ? GFP_ATOMIC : GFP_KERNEL;

    info = call_usermodehelper_setup(path, argv, envp, gfp_mask,
                                    NULL, NULL, NULL);
    if (info == NULL)
        return -ENOMEM;

    return call_usermodehelper_exec(info, wait);
}
EXPORT_SYMBOL(call_usermodehelper);

```

Как было показано в аналитическом разделе, данная функция выполняет подготовку и запуск приложения в пользовательском пространстве из режима ядра.

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.

### ***Ожидание потоком сигнала***

После запуска приложения в пользовательском режиме, поток засыпает, чтобы освободить процессор. Поток будет «спать», ожидая нажатия клавиши мыши. Перевод потока в спящий режим осуществляется с помощью функции *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.

```
int kthread_stop(struct task_struct *k)
{
    struct kthread *kthread;
    int ret;

    trace_sched_kthread_stop(k);

    get_task_struct(k);
    kthread = to_kthread(k);
    set_bit(KTHREAD_SHOULD_STOP, &kthread->flags);
    kthread_unpark(k);
    wake_up_process(k);
    wait_for_completion(&kthread->exited);
    ret = k->exit_code;
    put_task_struct(k);

    trace_sched_kthread_stop_ret(ret);
    return ret;
}
EXPORT_SYMBOL(kthread_stop);
```

## 2.5. Обработка сообщений от устройства

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

```
static void usb_mouse_irq(struct urb *urb)
```

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]*.

## 2.6. Схемы алгоритмов работы функции обработки событий

На рисунке 3 представлена схема алгоритма работы функции *set\_mouse\_status*, выполняющей инициализацию текущего состояния мыши по нажатию клавиши.

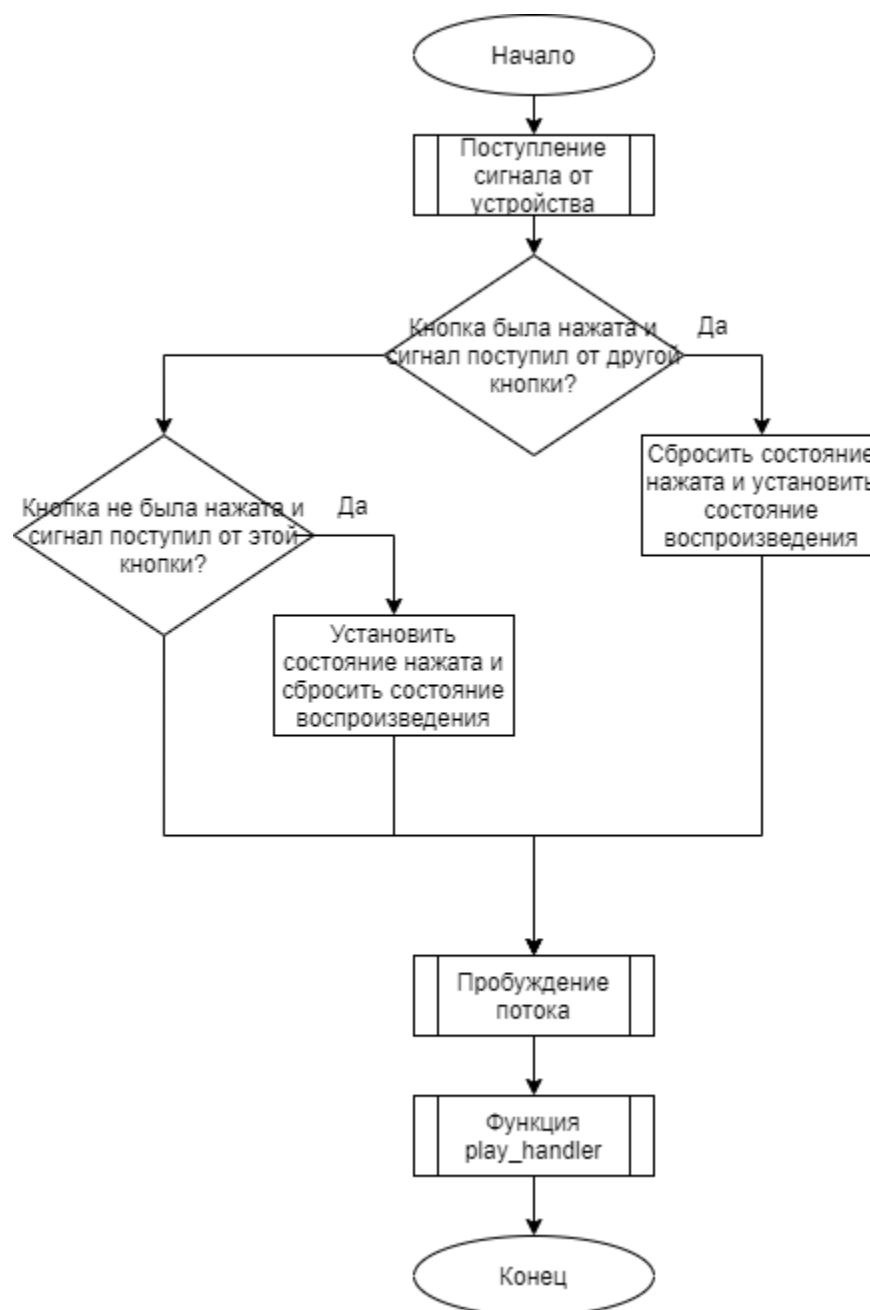


Рисунок 3. Схема алгоритма работы функции *set\_mouse\_status*.

На рисунке 4 представлена схема алгоритма работы функции *play\_handler*, реализованной в дополнительном потоке.

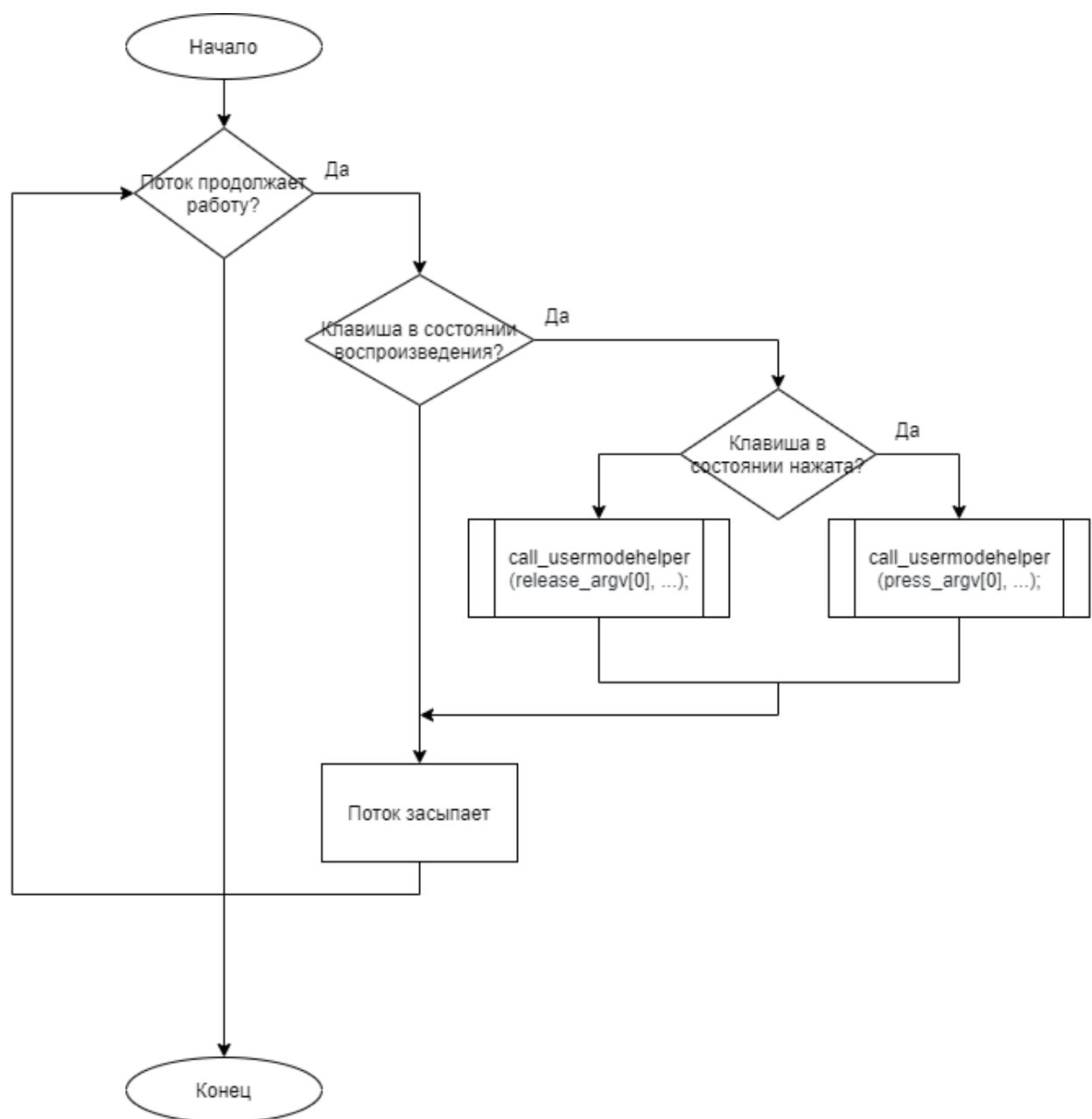


Рисунок 4. Схема алгоритма работы функции *play\_handler*.

## III. Технологический раздел

### 3.1. Выбор языка и среды программирования

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

### 3.2. Описание ключевых моментов реализации

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

Листинг 1 – Структура *struct usb\_driver*.

```
static struct usb_driver usb_mouse_driver = {
    .name          = "usbmouse",
    .probe         = usb_mouse_probe,
    .disconnect    = usb_mouse_disconnect,
    .id_table      = usb_mouse_id_table,
};
```

Локальной системной структурой является *usb\_mouse*.

Листинг 2 – Структура *struct usb\_mouse*.

```
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;
};
```

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*.

```
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
};
```

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

Листинг 4 – Инициализация модуля.

```
module_init(usb_mouse_init);|
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;
}
```

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

## Листинг 5 – Создание потока ядра.

```
/* 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");
    usb_free_urb(mouse->irq);
    free_devices(input_dev, mouse, dev);
    return -ENOMEM;
}
```

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*.

```
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;
}
```

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*.

```
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");
    }
}
```

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*.

```
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;
    }
}
```

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

Листинг 9 – остановка потока ядра.

```
static void usb_mouse_disconnect(struct usb_interface *intf)
{
    struct usb_mouse *mouse = usb_get_intfdata (intf);

    /* Остановка потока ядра */
    kthread_stop(playback_thread);
}
```

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



Листинг 10 – Функция, вызываемая при выгрузке ядра.

```
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);
```

### 3.3. Makefile

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

Листинг 11 – Makefile.

```
1  CONFIG_MODULE_SIG=n
2
3  obj-m := driver.o
4
5  KDIR := /lib/modules/$(shell uname -r)/build
6
7  PWD := $(shell pwd)
8
9  default:
10     $(MAKE) -C $(KDIR) M=$(PWD) modules
11
12  clean:
13     rm -rf *.o *.ko *.cmd modules.* Module.* .tmp versions *.mod.c test
```

### 3.4. 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.

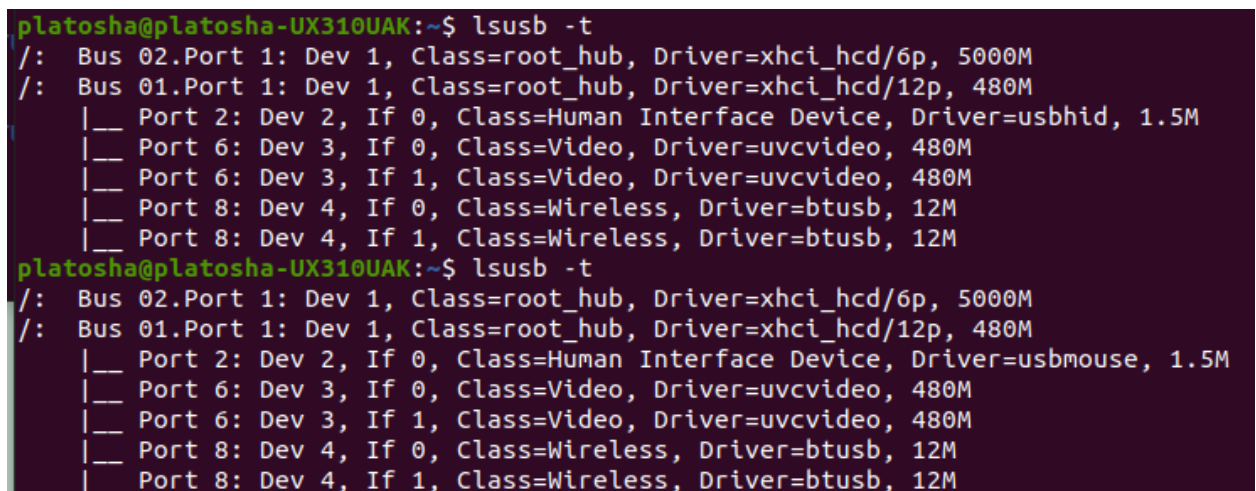
Листинг 12 – bash-файл привязки собственного драйвера.

```
#!/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 5 представлен список подключенных USB устройств до загрузки модуля и после. В данном случае *device\_id* = 1-2:1.0.



```
platosha@platosha-UX310UAK:~$ lsusb -t
/: Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/6p, 5000M
/: Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/12p, 480M
   |__ Port 2: Dev 2, If 0, Class=Human Interface Device, Driver=usbhid, 1.5M
   |__ Port 6: Dev 3, If 0, Class=Video, Driver=uvcvideo, 480M
   |__ Port 6: Dev 3, If 1, Class=Video, Driver=uvcvideo, 480M
   |__ Port 8: Dev 4, If 0, Class=Wireless, Driver=btusb, 12M
   |__ Port 8: Dev 4, If 1, Class=Wireless, Driver=btusb, 12M
platosha@platosha-UX310UAK:~$ lsusb -t
/: Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/6p, 5000M
/: Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/12p, 480M
   |__ Port 2: Dev 2, If 0, Class=Human Interface Device, Driver=usbmouse, 1.5M
   |__ Port 6: Dev 3, If 0, Class=Video, Driver=uvcvideo, 480M
   |__ Port 6: Dev 3, If 1, Class=Video, Driver=uvcvideo, 480M
   |__ Port 8: Dev 4, If 0, Class=Wireless, Driver=btusb, 12M
   |__ Port 8: Dev 4, If 1, Class=Wireless, Driver=btusb, 12M
```

Рисунок 5. Список подключенных USB устройств.

### 3.5. Результат выполнения

На рисунке 6 приведены этапы сборки и загрузки модуля в ядро.

```
platosh@platosh-UX310UAK:~/Desktop/BMSTU/7sem/Course-Project-05/src$ make
make -C /lib/modules/5.11.0-43-generic/build M=/home/platosh/Desktop/BMSTU/7sem/Course-Project-05/src modules
make[1]: вход в каталог «/usr/src/linux-headers-5.11.0-43-generic»
CC [M] /home/platosh/Desktop/BMSTU/7sem/Course-Project-05/src/driver.o
MODPOST /home/platosh/Desktop/BMSTU/7sem/Course-Project-05/src/Module.symvers
CC [M] /home/platosh/Desktop/BMSTU/7sem/Course-Project-05/src/driver.mod.o
LD [M] /home/platosh/Desktop/BMSTU/7sem/Course-Project-05/src/driver.ko
BTF [M] /home/platosh/Desktop/BMSTU/7sem/Course-Project-05/src/driver.ko
Skipping BTF generation for /home/platosh/Desktop/BMSTU/7sem/Course-Project-05/src/driver.ko due to unavailability of vmlinux
make[1]: выход из каталога «/usr/src/linux-headers-5.11.0-43-generic»
platosh@platosh-UX310UAK:~/Desktop/BMSTU/7sem/Course-Project-05/src$ sudo insmod driver.ko
platosh@platosh-UX310UAK:~/Desktop/BMSTU/7sem/Course-Project-05/src$ sudo bash bind.sh 1-2:1.0
Rebinding 1-2:1.0...
platosh@platosh-UX310UAK:~/Desktop/BMSTU/7sem/Course-Project-05/src$ sudo bash unbind.sh 1-2:1.0
Rebinding 1-2:1.0...
platosh@platosh-UX310UAK:~/Desktop/BMSTU/7sem/Course-Project-05/src$ sudo rmmod driver
```

Рисунок 6. Загрузка и выгрузка модуля.

На рисунке 7 приведены сообщения ядра, полученные в результате выполнения ПО.

```
[ 4017.429155] + module usb mouse driver loaded!
[ 4029.316305] + usb mouse was opened!
[ 4029.316559] + sound_play thread was created!
[ 4033.059719] + sound played due to press!
[ 4033.203787] + sound played due to release!
[ 4053.375760] + sound_play thread was stoped!
[ 4053.391993] + usb mouse was closed!
[ 4053.411728] + usb mouse was disconnected!
[ 4060.873299] + module usb mouse driver unloaded!
```

Рисунок 7. 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 сопровождение нажатия клавиш USB-мыши звуковыми сигналами.

В ходе работы был исследован системный драйвер мыши и способы реализации собственного. Были разработаны пользовательские структуры, описывающие состояние клавиш мыши. Также была определена целесообразность использования дополнительного потока ядра.

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

## Литература

1. Циллорик О. Модули ядра Linux. [Электронный ресурс]. – Режим доступа: <http://rus-linux.net/MyLDP/BOOKS/Moduli-yadra-Linux/03/kern-mod-03-01.html>.
2. Usage of Linux for websites. W3 Techs. [Электронный ресурс]. – URL: <https://w3techs.com/technologies/details/os-linux>
3. Драйверы устройств. Драйверы USB. [Электронный ресурс]. – URL: <http://rus-linux.net/MyLDP/BOOKS/drivers/linux-device-drivers-11.html>
4. Corbet J., Rubini A., Kroah-Hartman G. Драйверы устройств Linux, Третья редакция.
5. Introduction to USB on Linux. [Электронный ресурс]. – URL: <https://web.archive.org/web/20090518100552/http://tali.admingilde.org/linux-docbook/usb/ch01.html>
6. Исходные коды ядра Linux. [Электронный ресурс]. – URL: <http://elixir.free-electrons.com>.
7. 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 буферной 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) {
    strcpy(mouse->name, dev->manufacturer, sizeof(mouse->name));
}

if (dev->product) {
    if (dev->manufacturer) {
        strcat(mouse->name, " ", sizeof(mouse->name));
    }
    strcat(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));
strcat(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 A.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 A.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 A.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
```

