|  |  |
| --- | --- |
| 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 КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

***«Использование компьютерной мыши для ввода символов с использованием азбуки Морзе»***

Студент ИУ7-75Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Е.Б.Гришин

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

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

(Подпись, дата) (И.О.Фамилия)

*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\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_\_\_\_ИУ7-75Б\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Гришин Егор Борисович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Тема курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_Использование компьютерной мыши для ввода символов с использованием азбуки Морзе\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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\_\_\_\_\_****Разработать драйвер мыши позволяющий использовать её как телеграфный ключ, вводя данные азбукой Морзе.* ***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Оформление курсового проекта:***

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

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

На защиту проекта должна быть предоставлена презентация, состоящая из 15-20 слайдов.\_\_\_

На слайдах должны быть отражены: постановка задачи, использованные методы и\_\_\_\_\_\_\_\_\_

алгоритмы, расчетные соотношения, структура комплекса программ, интерфейс,\_\_\_\_\_\_\_\_\_\_\_

результаты проведенных исследований.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания « 11 » октября 2021 г.

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

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**Е.Б. Гришин**\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Оглавление

[Введение 4](#_Toc90727331)

[1. Аналитическая часть 5](#_Toc90727332)

[1.1 Формализация задачи 5](#_Toc90727333)

[1.2 /dev/input/event 5](#_Toc90727334)

[1.3 Драйвер устройства 6](#_Toc90727335)

[1.4 Измерение временных промежутков 9](#_Toc90727336)

[1.5 Таймер ядра 9](#_Toc90727337)

[1.6 Обработчик прерывания 10](#_Toc90727338)

[1.7 Выводы из аналитического раздела 11](#_Toc90727339)

[2. Конструкторская часть 12](#_Toc90727340)

[2.1 Требования к программе 12](#_Toc90727341)

[2.2 Драйвер USB мыши 12](#_Toc90727342)

[2.3 Модуль morze\_enter 13](#_Toc90727343)

[2.4 Схемы, демонстрирующие работу модуля morze\_enter 14](#_Toc90727344)

[2.5 Вывод 16](#_Toc90727345)

[3. Технологическая часть 17](#_Toc90727346)

[3.1 Выбор языка программирования и среды разработки 17](#_Toc90727347)

[3.2 Модуль morze\_enter 17](#_Toc90727348)

[3.3 Модуль driver 20](#_Toc90727349)

[3.4 Makefile 25](#_Toc90727350)

[3.5 Демонстрация работы 25](#_Toc90727351)

[3.6 Вывод 26](#_Toc90727352)

[Заключение 27](#_Toc90727353)

[Список использованной литературы 28](#_Toc90727354)

# Введение

Азбука Морзе – способ знакового кодирования, представление букв алфавита, цифр, знаков препинания и других символов последовательностью сигналов: длинных (тире) и коротких (точек). Передача кодов Морзе производится при помощи телеграфного ключа различных конструкций. Компьютерная мышь может быть использована в качестве телеграфного ключа при наличии соответствующего драйвера. Такой режим ввода может быть резервным на случай выхода из строя клавиатуры. Так как единственный эффективный способ решения – загружаемый модуль ядра, то данная работа будет основана на анализе и модификации системного драйвера USB-мыши.

# 1. Аналитическая часть

## 1.1 Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать и реализовать загружаемый модуль ядра, позволяющего интерпретировать короткие и длительные нажатия клавиши мыши как точки и тире, а их комбинации как вводимые символы. Программное обеспечение должно обеспечивать перехват нажатий кнопки мыши и измерение их длительности.

Из-за разной длины кодов Морзе отделять ввод одного символа от ввода следующего нужно длительным временным промежутком. Для этого нужно решить задачу откладывания выполнения задачи на фиксированный промежуток времени – изучить таймеры ядра.

Для достижения поставленной цели необходимо решить следующие задачи:

* проанализировать структуру драйвера мыши;
* проанализировать способы измерения времени в ядре;
* проанализировать таймеры ядра;
* проанализировать способы эмуляции нажатия клавиатуры;
* спроектировать и реализовать модуль ядра.

Разрабатываемое ПО должно обрабатывать нажатия только левой клавиши мыши.

## 1.2 /dev/input/event

Основными компонентами подсистемы ввода-вывода являются драйверы, управляющие внешними устройствами, и файловая система.

Драйвер взаимодействует, с одной стороны, с модулями ядра ОС, а с другой стороны — с контроллерами внешних устройств. Драйверы таких устройств, как мышь, клавиатура, и др. генерируют события, которые в виде структур input\_event записываются в соответствующий файл в директории “/dev/input”.

Листинг 1.2.1 – структура input\_event

|  |
| --- |
| struct input\_event {          struct timeval time; // время возникновения события          unsigned short type; // тип события (Например: EV\_REL, EV\_KEY)          unsigned short code; // код события (Например: REL\_X, KEY\_BACKSPACE)          unsigned int value; // статус события (Например для EV\_KEY: 0-release, 1-keypress, 2-autorepeat)  }; |

Чтобы увидеть это, нужно с помощью команды «cat /proc/bus/input/devices» узнать, какой файл “/dev/input/eventX” соответствует подключенной мыши, и с помощью команды «cat /dev/input/eventX» просмотреть его содержимое. При движении мышью или нажатии её кнопок на экране будут появляться символы.

Для работы с файлами в пространстве ядра нужно использовать специальные функции: filp\_open(), filp\_close(), kernel\_read(), kernel\_write().

С помощью /dev/input/event можно регистрировать нажатия клавиши мыши и эмулировать нажатия клавиш клавиатуры. Но в таком случае разрабатываемое ПО может работать в пространстве пользователя, что не подходит для курсовой работы по операционным системам. Поэтому данным способом будет эмулироваться только нажатие клавиш клавиатуры.

## 1.3 Драйвер устройства

Класс HID (Human Interface Device) драйверов предназначен для управления различными типами внешних устройств. Если устройство поддерживает HID интерфейс, т.е. написано в соответствии с его спецификацией, то его подключение не требует разработки нового драйвера. Однако при необходимости подключения устройства, не отвечающего HID-спецификации, необходим соответствующий HID-драйвер.

К классу USB HID принадлежат все устройства для взаимодействия с пользователем — клавиатуры, мыши, джойстики и т.д.

Регистрация USB-драйвера подразумевает:

1. заполнение структуры usb\_driver;
2. регистрация структуры в системе.

В ОС Linux код драйвера находится в файле usbmouse.c, а структура usb\_driver описана в includr/linux/usb.h.

Листинг 1.3.1 – структуры usb\_driver и usbdrv\_wrap (файл usb.h)

|  |
| --- |
| /\*\*   \* struct usbdrv\_wrap - wrapper for driver-model structure   \* @driver: The driver-model core driver structure.   \* @for\_devices: Non-zero for device drivers, 0 for interface drivers.   \*/  struct usbdrv\_wrap {      struct device\_driver driver;      int for\_devices;  };  struct usb\_driver {      const char \*name;      int (\*probe) (struct usb\_interface \*intf,                const struct usb\_device\_id \*id);      void (\*disconnect) (struct usb\_interface \*intf);      // ...      const struct usb\_device\_id \*id\_table;          // ...      struct usbdrv\_wrap drvwrap;          // ...  }; |

**Name** – это имя драйвера, должно быть уникальным среди USB-драйверов в ядре и таким же как имя модуля.

**id\_table** – это массив структур usb\_device\_id, который содержит список всех типов USB-устройств, которые обслуживает драйвер. В самом простом случае каждый элемент id\_table[i], который определяет интерфейс подключаемого устройства, содержит пару идентификаторов: идентификатор производителя и устройства.

Поле **drvwrap** – структура usbdrv\_wrap, которая является оберткой для device\_driver, которая говорит о том, что usb\_driver унаследован от device\_driver.

**probe** и **disconnect** – это функции обратного вызова (callbacks), вызываемые системой в контексте потока ядра USB-хаба. Probe() является точкой входа драйвера, которая инициализирует и регистрирует другие точки входа, она будет вызвана для каждого устройства, если список id\_table пуст, или только для тех устройств, которые соответствуют параметрам, перечисленным в списке.

Один зарегистрированный драйвер может "подключать" несколько устройств. Для установления связи устройства и драйвера система вызывает функцию драйвера probe(), которой передает 2 параметра:

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

**interface** – это интерфейс USB-устройства. Обычно USB-драйвер взаимодействует не с устройством напрямую, а с его интерфейсом. **id** - содержит информацию об устройстве. Если функция возвращает 0, то устройство успешно зарегистрировано, иначе - система попытается "привязать" устройство к какому-нибудь другому драйверу.

Для отключения устройства от драйвера система вызывает функцию disconnect, которой передается один параметр - интерфейс:

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

В общем случае, в функции probe для каждого подключаемого устройства выделяется структура в памяти, заполняется, затем регистрируется, например, символьное устройство, и проводится регистрация устройства в sysfs.

При установке собственного драйвера сначала необходимо выгрузить модуль usbhid, который автоматически регистрирует все стандартные драйверы в системе. Данный модуль устанавливает стандартный драйвер мыши и не позволяет установить свой. Если драйвер установлен, то его можно увидеть в ядре в sysfs, указав путь /sys/bus/usb/drivers/.

## 1.4 Измерение временных промежутков

Ядро следит за течением времени с помощью таймера прерываний. Прерывания таймера генерируются через постоянные интервалы системным аппаратным таймером; этот интервал программируется во время загрузки ядром в соответствии со значением HZ, которое является архитектурно-зависимой величиной, определённой в или файле подплатформы, подключаемом им. Оно определяет количество тиков в секунду.

Значение внутреннего счётчика ядра увеличивается каждый раз, когда происходит прерывание от таймера. Счётчик инициализируется 0 при загрузке системы, поэтому он представляет собой число тиков системных часов после последней загрузки. Счётчик является 64-х разрядной переменной (даже на 32-х разрядных архитектурах) и называется jiffies\_64. Однако, авторы драйверов обычно используют переменную jiffies типа unsigned long, которая то же самое, что и jiffies\_64 или её младшие биты.

Всякий раз, когда код должен запомнить текущее значение jiffies, он может просто обратится к переменной unsigned long, которая объявлена как volatile (нестабильная), чтобы компилятор не оптимизировал чтения памяти. Используя HZ, можно перевести промежуток времени в тиках в секунды.

## 1.5 Таймер ядра

Инструментом для планирования действия на позднее время без блокирования текущего процесса до наступления момента времени являются таймеры ядра. Такие таймеры используются для планирования выполнения функции в определённое время в будущем, основываясь на тактовых тиках. Таймер ядра описывается структурой timer\_list, которая инструктирует ядро для выполнения заданных пользователем функции с заданным пользователем аргументом в заданное пользователем время.

Листинг 1.5.1 – структура timer\_list

|  |
| --- |
| struct timer\_list {      /\* ... \*/      unsigned long expires;      void (\*function)(unsigned long);      unsigned long data;  }; |

Структура данных включает в себя больше полей, чем показано, но эти три предназначены для доступа снаружи кодом таймера. Поле expires представляет значение jiffies, которое таймер ожидает для запуска; в это время функция function вызывается с data в качестве аргумента. Если необходимо передать много объектов в аргументе, можно собрать их в единую структуру данных и передать указатель, приведя к unsigned long, это безопасная практика на всех поддерживаемых архитектурах и довольно распространена в управлении памятью.

## 1.6 Обработчик прерывания

При возникновении аппаратного прерывания вызванный обработчик запрещает прерывания на локальном процессоре. В результате другая работа выполняться не может. Такая ситуация в системе должна быть краткосрочной. Быстрые прерывания блокируют все другие прерывания, во время длинных IRQ могут обрабатываться другие прерывания (но не от того же устройства). Медленные прерывания разбивают на две части: исполняемую сразу при возникновении аппаратного прерывания и работу, которая может быть отложена на некоторое время. Эти части обработчиков прерываний получили название "верхняя" и "нижняя" половины.

“Верхняя половина“ занимается чтением и сохранением в буфере данных или передачей их из буфера в регистры контроллера и завершается постановкой “нижней половины“ в очередь на выполнение и разрешением прерываний.

“Нижняя половина“, как правило, выполняется сразу после завершения “верхней“ и запускает все необходимые операции, причем ей доступно всё то, что доступно обычным модулям ядра. Выполняется отдельным потоком ядра, имеет более низкий уровень приоритета и является отложенным действием, которое может быть прервано во время его выполнения.

Существует несколько способов реализации “нижней половины“ обработчика: гибкое прерывание (softirq), тасклет (tasklet) и очереди работ (workqueue).

## 1.7 Выводы из аналитического раздела

В данном разделе была формализована задача и рассмотрены способы ее решения. Выяснилось, что использование /dev/input/event и sys\_call\_table не подходят в рамках данной курсовой работы. Было принято решение написать драйвер для слежения за мышью и использовать /dev/input/event для эмуляции нажатий клавиатуры.

# 2. Конструкторская часть

В данном разделе будут рассмотрены требования к программе, основные сведения о реализуемых модулях, предоставлены схемы, описывающие работу модуля реализующего логику обработки нажатий клавиши мыши.

## 2.1 Требования к программе

Необходимо реализовать загружаемый модуль ядра, который будет получать данные о событиях мыши, интерпретировать их и писать соответствующие события в /dev/input/event, далее модуль morze\_enter.

Также необходимо реализовать драйвер usb мыши, который будет отсылать данные в модуль morze\_enter.

## 2.2 Драйвер USB мыши

За основу драйвера мыши стоит взять USB HID драйвер мыши, который можно найти в исходном коде linux: /drivers/hid/usbhid/usbmouse.c.

Внутри этого драйвера, сообщения, отправленные устройством, обрабатываются функцией usb\_mouse\_irq. Необходимо поместить в эту процедуру вызов экспортируемой функции из модуля morze\_enter, в которую будут передаваться данные, пришедшие от устройства.

Листинг 2.2.1 – заголовок usb\_mouse\_irq

|  |
| --- |
| static void usb\_mouse\_irq(struct urb \*urb); |

Листинг 2.2.2 – заголовок экспортируемой функции

|  |
| --- |
| extern void my\_mouse\_handler(char new\_mouse\_status); |

В листинге 2.2.2 предоставлен заголовок экспортируемой функции, вызов которой должен быть помещен в функцию usb\_mouse\_irq.

**new\_mouse\_status** – состояние кнопок мыши, стандартно: 0 – нейтральное; 1 – нажата левая, 2 – правая кнопка мыши, 3 – нажаты обе.

## 2.3 Модуль morze\_enter

При инициализации модуля инициализируются используемые структуры: тасклета, таймера, файла /dev/input/event. При завершении работы модуля используемые ресурсы освобождаются:

Функция my\_mouse\_handler при запуске проверяет изменение состояния клавиши мыши, т.к. запускается она при каждом прерывании от мыши, в том числе при её движении. Если состояние изменилось, т.е. клавишу нажали или отпустили, то таймер, если он был поставлен, сбрасывается.

Для хранения введённого кода выделяется массив символов entered\_code. Функция my\_mouse\_handler после отпускания клавиши мыши записывает в него точку или тире и, если entered\_code ещё не заполнен, т.е. не введён код максимально возможной длины, ставит таймер. Если же entered\_code заполнен, то его содержимое копируется в дополнительный массив entered\_code\_tasklet и ставится на выполнение тасклет.

Если клавиша находилась в одном состоянии достаточно долго (1 секунду), то успевает начать выполнение функция-обработчик таймера – timer\_check. Она производит те же самые действия, что и my\_mouse\_handler при заполнении entered\_code. Т.е. если нужно ввести код не максимальной длины, пользователь должен подождать 1 секунду.

В функции тасклета содержимое entered\_code\_tasklet сверяется со словарём и, если совпадение найдено, эмулируется нажатие соответствующей клавиши – в /dev/input/event записывается набор структур, сообщающий о её нажатии и отпускании. В данном случае entered\_code\_tasklet является разделяемой переменной. В неё пишут timer\_check и my\_mouse\_handler, а читает функция тасклета. Но, операциями записи в неё проходит не меньше 1 секунды, вероятность, что функция тасклета не успеет прочесть её за это время, пренебрежимо мала. Поэтому использовать средства синхронизации для её чтения смысла нет.

Но возможна одновременная запись в неё, а так же есть ещё несколько разделяемых переменных между timer\_check и my\_mouse\_handler. Для их синхронизации используется спин-блокировка.

## 2.4 Схемы, демонстрирующие работу модуля morze\_enter

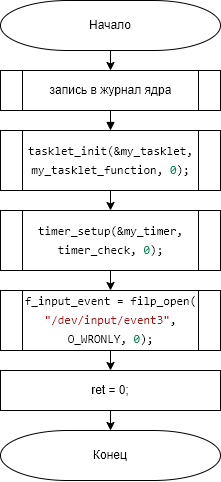
****

Рис. 2.4.1 – схема инициализации работы модуля

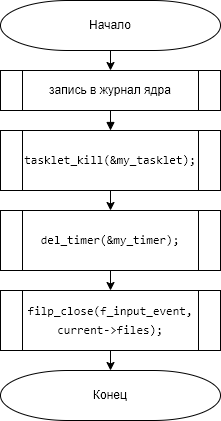


Рис. 2.4.2 – схема завершения работы модуля

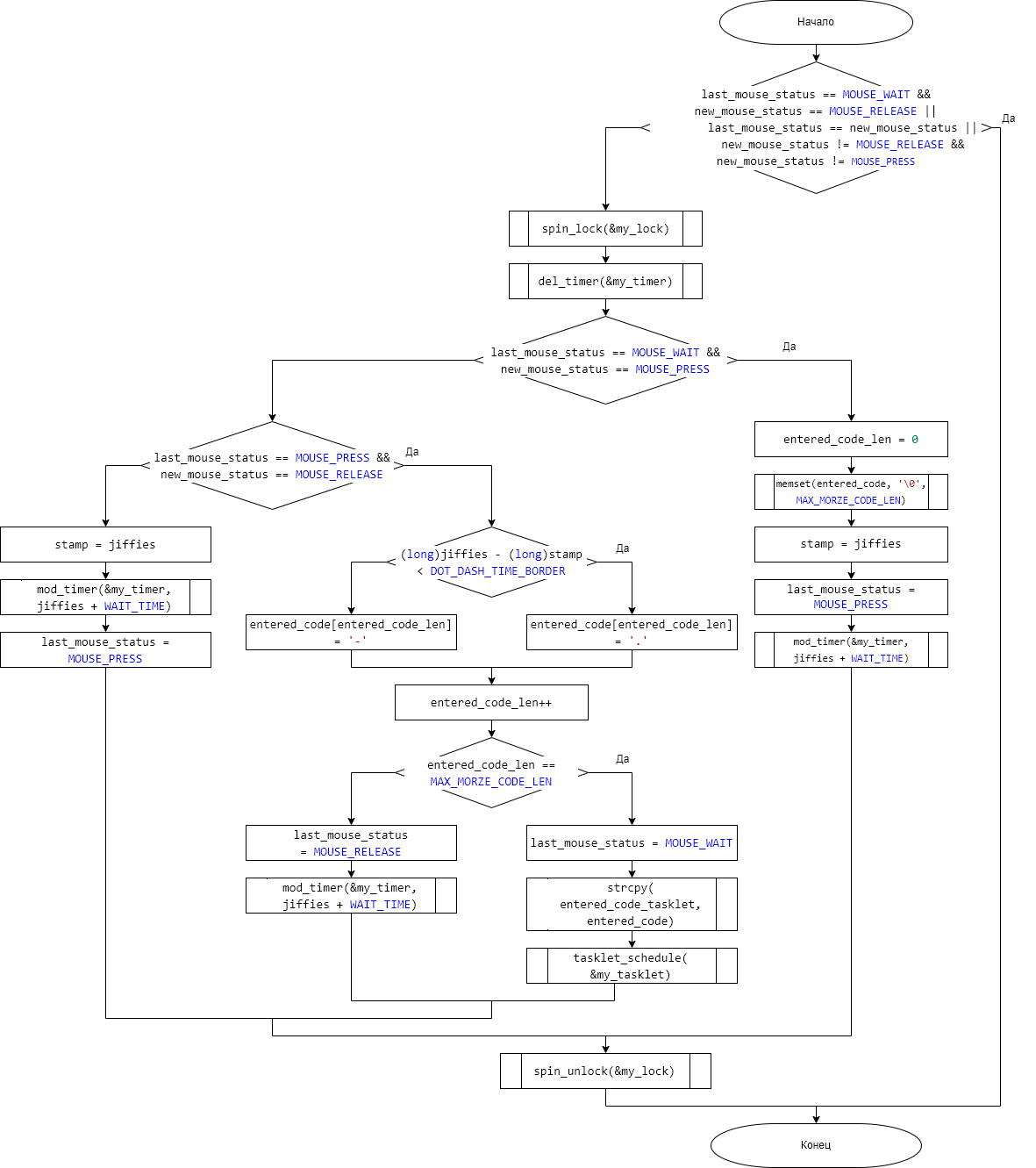


Рис. 2.4.3 – схема обработчика мыши

## 2.5 Вывод

В данном разделе были рассмотрены требования к программе, основные сведения о реализуемых модулях, предоставлены схемы, описывающие работу модуля morze\_enter.

# 3. Технологическая часть

В данном разделе будут предоставлены листинги реализованных модулей и демонстрация их работы.

## 3.1 Выбор языка программирования и среды разработки

В качестве языка программирования был выбран C.

В качестве среды разработки была выбрана «Code::Blocks» т.к. она бесплатная и кроссплатформенная.

ПО протестировано на ядре Linux v5.4.50.

## 3.2 Модуль morze\_enter

Листинг 3.2.1 – файл morze\_enter.c

|  |
| --- |
| #include <linux/module.h>  #include <linux/kernel.h>  #include <linux/jiffies.h>  #include <linux/timer.h>  #include <linux/interrupt.h>  #include <linux/spinlock.h>  #include <linux/input.h>  #include "morze\_enter.h"  #define MOUSE\_WAIT -1  #define MOUSE\_PRESS 1  #define MOUSE\_RELEASE 0  #define DOT\_DASH\_TIME\_BORDER HZ / 3  #define WAIT\_TIME HZ  static char last\_mouse\_status = MOUSE\_WAIT;  static unsigned long stamp = 0;  static char entered\_code[MAX\_MORZE\_CODE\_LEN + 1] = "\0\0\0\0\0\0";  static int entered\_code\_len = 0;  static char entered\_code\_tasklet[MAX\_MORZE\_CODE\_LEN + 1];  static struct tasklet\_struct my\_tasklet;  static struct timer\_list my\_timer;  DEFINE\_SPINLOCK(my\_lock);  static struct file \*f\_input\_event;  struct input\_event event\_kbd;  void my\_tasklet\_function(unsigned long data);  void timer\_check(struct timer\_list \*timer);  MODULE\_LICENSE("GPL");  MODULE\_AUTHOR("Grishin E.B");  MODULE\_DESCRIPTION("Entering by mouse morze");  static int \_\_init morze\_init( void )  {      printk(KERN\_INFO "[morze] in init\n");      tasklet\_init(&my\_tasklet, my\_tasklet\_function, 0);      timer\_setup(&my\_timer, timer\_check, 0);      f\_input\_event = filp\_open("/dev/input/event3", O\_WRONLY, 0);      return 0;  }  static void \_\_exit morze\_exit( void )  {      printk(KERN\_INFO "[morze] in exit\n");      tasklet\_kill(&my\_tasklet);      del\_timer(&my\_timer);      filp\_close(f\_input\_event, current->files);  }  void my\_tasklet\_function(unsigned long data)  {      int i = 0;      for (; i < MORZE\_DICT\_LEN && strcmp(morze\_codes[i], entered\_code\_tasklet); i++);      if (i < MORZE\_DICT\_LEN)      {          printk(KERN\_INFO "[morze] entered code: %s; entered symbol: %c\n", entered\_code\_tasklet, morze\_symbols[i]);          event\_kbd.type = EV\_KEY;          event\_kbd.code = morze\_keys[i];          event\_kbd.value = 1;          kernel\_write(f\_input\_event, (char\*)&event\_kbd, sizeof(event\_kbd), &f\_input\_event->f\_pos);          event\_kbd.type = EV\_KEY;          event\_kbd.code = morze\_keys[i];;          event\_kbd.value = 0;          kernel\_write(f\_input\_event, (char\*)&event\_kbd, sizeof(event\_kbd), &f\_input\_event->f\_pos);          event\_kbd.type = EV\_SYN;          event\_kbd.code = 0;          event\_kbd.value = 0;          kernel\_write(f\_input\_event, (char\*)&event\_kbd, sizeof(event\_kbd), &f\_input\_event->f\_pos);      }  }  void timer\_check(struct timer\_list \*timer)  {      spin\_lock(&my\_lock);      if (last\_mouse\_status != MOUSE\_WAIT)      {          last\_mouse\_status = MOUSE\_WAIT;          strcpy(entered\_code\_tasklet, entered\_code);          tasklet\_schedule(&my\_tasklet);      }      spin\_unlock(&my\_lock);  }  extern void my\_mouse\_handler(char new\_mouse\_status)  {      if (last\_mouse\_status == MOUSE\_WAIT && new\_mouse\_status == MOUSE\_RELEASE ||          last\_mouse\_status == new\_mouse\_status ||          new\_mouse\_status != MOUSE\_RELEASE && new\_mouse\_status != MOUSE\_PRESS)          return;      spin\_lock(&my\_lock);      del\_timer(&my\_timer);      if (last\_mouse\_status == MOUSE\_WAIT && new\_mouse\_status == MOUSE\_PRESS)      {          entered\_code\_len = 0;          memset(entered\_code, '\0', MAX\_MORZE\_CODE\_LEN);          stamp = jiffies;          last\_mouse\_status = MOUSE\_PRESS;          mod\_timer(&my\_timer, jiffies + WAIT\_TIME);      }      else if (last\_mouse\_status == MOUSE\_PRESS && new\_mouse\_status == MOUSE\_RELEASE)      {          if ((long)jiffies - (long)stamp < DOT\_DASH\_TIME\_BORDER)              entered\_code[entered\_code\_len] = '.';          else              entered\_code[entered\_code\_len] = '-';          entered\_code\_len++;          if (entered\_code\_len == MAX\_MORZE\_CODE\_LEN)          {              last\_mouse\_status = MOUSE\_WAIT;              strcpy(entered\_code\_tasklet, entered\_code);              tasklet\_schedule(&my\_tasklet);          }          else          {              last\_mouse\_status = MOUSE\_RELEASE;              mod\_timer(&my\_timer, jiffies + WAIT\_TIME);          }      }      else // if (last\_mouse\_status == MOUSE\_RELEASE && new\_mouse\_status == MOUSE\_PRESS)      {          stamp = jiffies;          mod\_timer(&my\_timer, jiffies + WAIT\_TIME);          last\_mouse\_status = MOUSE\_PRESS;      }      spin\_unlock(&my\_lock);  }  EXPORT\_SYMBOL(my\_mouse\_handler);  module\_init(morze\_init);  module\_exit(morze\_exit); |

## 3.3 Модуль driver

Листинг 3.3.1 – файл driver.c

|  |
| --- |
| //from https://github.com/torvalds/linux/blob/master/drivers/hid/usbhid/usbmouse.c  /// SPDX-License-Identifier: GPL-2.0-or-later  /\*   \*  Copyright (c) 1999-2001 Vojtech Pavlik   \*   \*  USB HIDBP Mouse support   \*/  /\*   \*   \* Should you need to contact me, the author, you can do so either by   \* e-mail - mail your message to <vojtech@ucw.cz>, or by paper mail:   \* Vojtech Pavlik, Simunkova 1594, Prague 8, 182 00 Czech Republic   \*/  #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>  /\*   \* Version Information   \*/  #define DRIVER\_VERSION "v1.6"  #define DRIVER\_AUTHOR "Grishin E.B."  #define DRIVER\_DESC "USB HID Boot Protocol mouse driver"  extern void my\_mouse\_handler(char new\_mouse\_status);  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;  };  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;      my\_mouse\_handler(data[0]);      switch (urb->status) {      case 0:         /\* success \*/          break;      case -ECONNRESET:   /\* unlink \*/      case -ENOENT:      case -ESHUTDOWN:          return;      /\* -EPIPE:  should clear the halt \*/      default:        /\* error \*/          goto resubmit;      }      input\_report\_key(dev, BTN\_LEFT,   data[0] & 0x01);      input\_report\_key(dev, BTN\_RIGHT,  data[0] & 0x02);      input\_report\_key(dev, BTN\_MIDDLE, data[0] & 0x04);      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);      printk(KERN\_INFO "[usbmouse] in usb\_mouse\_open\n");      mouse->irq->dev = mouse->usbdev;      if (usb\_submit\_urb(mouse->irq, GFP\_KERNEL))          return -EIO;      return 0;  }  static void usb\_mouse\_close(struct input\_dev \*dev)  {      struct usb\_mouse \*mouse = input\_get\_drvdata(dev);      printk(KERN\_INFO "[usbmouse] in usb\_mouse\_close\n");      usb\_kill\_urb(mouse->irq);  }  static int usb\_mouse\_probe(struct usb\_interface \*intf, const struct usb\_device\_id \*id)  {      printk(KERN\_INFO "[usbmouse] in usb\_mouse\_probe\n");      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;      endpoint = &interface->endpoint[0].desc;      if (!usb\_endpoint\_is\_int\_in(endpoint))          return -ENODEV;      pipe = usb\_rcvintpipe(dev, endpoint->bEndpointAddress);      maxp = usb\_maxpacket(dev, pipe, usb\_pipeout(pipe));      mouse = kzalloc(sizeof(struct usb\_mouse), GFP\_KERNEL);      input\_dev = input\_allocate\_device();      if (!mouse || !input\_dev)          goto fail1;      mouse->data = usb\_alloc\_coherent(dev, 8, GFP\_ATOMIC, &mouse->data\_dma);      if (!mouse->data)          goto fail1;      mouse->irq = usb\_alloc\_urb(0, GFP\_KERNEL);      if (!mouse->irq)          goto fail2;      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));      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;      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);      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)  {      printk(KERN\_INFO "[usbmouse] in usb\_mouse\_disconnect\n");      struct usb\_mouse \*mouse = usb\_get\_intfdata (intf);      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);      }  }  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       = "my\_usb\_mouse\_driver",      .probe      = usb\_mouse\_probe,      .disconnect = usb\_mouse\_disconnect,      .id\_table   = usb\_mouse\_id\_table,  };  module\_usb\_driver(usb\_mouse\_driver); |

## 3.4 Makefile

Листинг 3.4.1 – makefile для сборки модулей

|  |
| --- |
| KBUILD\_EXTRA\_SYMBOLS = $(shell pwd)/Module.symverscd  ifneq ($(KERNELRELEASE),)      obj-m := driver.o morze\_enter.o  else      CURRENT = $(shell uname -r)      KDIR = /lib/modules/$(CURRENT)/build      PWD = $(shell pwd)  default:      $(MAKE) -C $(KDIR) M=$(PWD) modules      make cleanHalf  cleanHalf:      rm -rf \*.o \*~ \*.mod \*.mod.c Module.\* \*.order  .tmp\_versions    clean:      make cleanHalf      rm -rf \*.ko  endif |

## 3.5 Демонстрация работы

На рис. 3.5.1 продемонстрирована последовательность загрузки драйвера мыши.

На рис. 3.5.2 продемонстрирована последовательность выгрузки драйвера мыши.

На рис 3.5.3 - 3.5.4 продемонстрирован ввод символов азбукой Морзе.

D:\Egor\Study\7_sem\CourseProjectOS\RPZ\Скриншоты\Снимок экрана от 2021-12-17 05-29-14.png

Рис. 3.5.1 – загрузка драйвера

D:\Egor\Study\7_sem\CourseProjectOS\RPZ\Скриншоты\Снимок экрана от 2021-12-17 05-29-27.png

Рис 3.5.2 – выгрузка драйвера

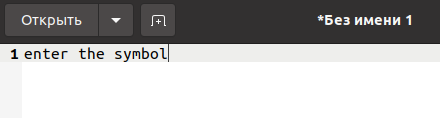


Рис 3.5.3 – введённые символы

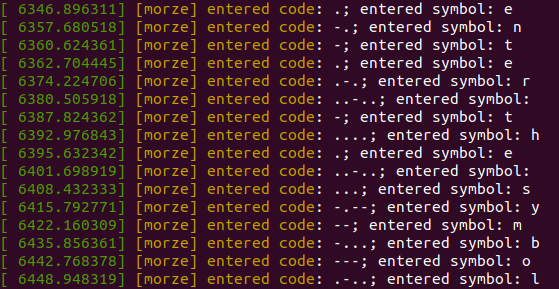


Рис 3.5.4 – сообщения ядра при вводе

## 3.6 Вывод

Были реализованы модуль драйвера usb мыши и модуль morze\_enter, листинг которых был предоставлен в данном разделе. Также была проведена демонстрация работы разработанного ПО.

# Заключение

Во время выполнения курсового проекта были достигнуты поставленные цель и задачи:

* была проанализирована структура драйвера мыши;
* были проанализированы способы измерения времени в ядре;
* были проанализированы таймеры ядра;
* были проанализированы способы эмуляции нажатия клавиатуры;
* был спроектирован и реализован модуль ядра и драйвер.

В ходе выполнения поставленных задач были изучены возможности языка C, получены знания в области написания загружаемых модулей ядра, драйверов.

# Список использованной литературы

1. Exploring /dev/input. Keerthi Vasan G.C, Suresh. B, 21.04.2017. The hacker Diary. [Электронный ресурс]. – Режим доступа: <https://thehackerdiary.wordpress.com/2017/04/21/exploring-devinput-1/>
2. Linux Input drivers v1.0. Vojtech Pavlik. [Электронный ресурс]. – Режим доступа: <https://www.kernel.org/doc/Documentation/input/input.txt>
3. А.Н. Васюнин, Н.Ю. Рязанова, канд. техн. наук, доц., Е.В. Тарасенко, С.В. Тарасенко. Анализ методов изменения функциональности внешних устройств в ОС Linux. В сб. «Автоматизация. Современные технологии», 2016 №10. С. 3-8.
4. Исходный код файла usb.h [Электронный ресурс]. – Режим доступа: <https://github.com/torvalds/linux/blob/master/include/linux/usb.h>
5. Corbet J., Rubini A., Kroan-Hartman G. Linux device drivers. O’Reilly Media, 2005.

# Приложение А