



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:

*«Загружаемый модуль ядра для мониторинга
действий пользователя»*

Студент ИУ7-75Б
(Группа)

Т.М.Оберган
(Подпись, дата) (И.О.Фамилия)

Руководитель

Н.Ю.Рязанова
(Подпись, дата) (И.О.Фамилия)

2020 г.

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

УТВЕРЖДАЮ
Заведующий кафедрой _____ ИУ7 _____
(Индекс)
_____ И. В. Рудаков
(И.О.Фамилия)
« _____ » _____ 20 _____ г.

З А Д А Н И Е на выполнение курсового проекта

по дисциплине _____ Операционные системы

Студент группы _____ ИУ-75Б

Оберган Татьяна Максимовна
(Фамилия, имя, отчество)

Тема курсового проекта _____ Загружаемый модуль ядра для мониторинга действий пользователя

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____ Учебная

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 загружаемый модуль ядра для мониторинга различных действий пользователя.
Предоставить возможность просмотра и выгрузку лога отслеживаемых событий.

Оформление курсового проекта:

Расчетно-пояснительная записка на 25-35 листах формата А4.

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

На защиту работы должна быть представлена презентация, состоящая из 10-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО.

Дата выдачи задания « 22 » _____ сентября _____ 2020 г.

Руководитель курсового проекта

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

Студент

_____ Оберган Т.М.
(Подпись, дата) (И.О.Фамилия)

Оглавление

Введение.....	4
1. Аналитическая часть.....	5
1.1 Формализация задачи.....	5
1.2 Анализ способов решения поставленной задачи	5
1.3 /dev/input/event	6
1.4 Keyboard notifier.....	6
1.5 Драйвер устройства.....	7
1.6 Обработчик прерывания	9
1.7 sys_call_table	10
1.8 Передача данных в пространство пользователя.....	11
1.9 Выводы из аналитического раздела	12
2. Конструкторская часть	13
2.1 Требования к программе.....	13
2.2 Драйвер USB мыши	13
2.3 Модуль - логгер	14
2.4 Схемы, демонстрирующие работу модуля	16
2.5 Вывод.....	18
3. Технологическая часть.....	19
3.1 Выбор языка программирования и среды разработки.....	19
3.2 Модуль – логгер.....	19
3.3 Драйвер мыши	22
3.4 Makefile.....	26
3.5 Взаимодействие с модулями	26
3.6 Вывод.....	29
Заключение	30
Список использованной литературы.....	31

Введение

Мышь и клавиатура - интерактивные средства ввода-вывода. Прерывания от мыши возникают очень часто.

Информацию о событиях клавиатуры и мыши можно использовать как во вред, так и во благо: для тестирования приложений и интерфейса, для слежения за пользователем. При помощи подобных программ можно получить персональные данные: пароли, номера банковских карт, и т.д.

Целью данной работы является реализация программного обеспечения для перехвата сообщений USB мыши и клавиатуры и их последующего журналирования.

1. Аналитическая часть

В данном разделе будет формализована задача и рассмотрены способы ее решения.

1.1 Формализация задачи

Целью данного курсового проекта является реализация программного обеспечения, фиксирующего события в системе, инициирующиеся действиями пользователя – взаимодействие с мышью и клавиатурой.

Программное обеспечение должно обеспечивать перехват нажатий кнопок, перемещения мыши и движения колесика, фиксировать эту информацию для того, чтобы в дальнейшем было возможно произвести анализ этой информации.

Для достижения поставленной цели необходимо решить следующие задачи:

- проанализировать способы перехвата сообщений от мыши и клавиатуры;
- проанализировать структуру драйвера мыши;
- проанализировать методы передачи информации из модулей ядра в пространство пользователя;
- спроектировать и реализовать модуль ядра;

1.2 Анализ способов решения поставленной задачи

Существует несколько способов решения поставленной задачи:

- чтение информации из системного файла устройства `“/dev/input/event*”`[1];
- использование keyboard notifier;
- разработка драйвера устройства в виде загружаемого модуля ядра;

- установка на линию прерывания нескольких обработчиков прерываний;
- изменение соответствующих указателей в `sys_call_table`.

1.3 /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 можно посмотреть в директории “/dev/input”. Например, чтобы проверить, что мышь эмулирована, необходимо ввести в консоли «`cat /dev/input/mouse0`», при движении мышью на экране должны появляться символы. [2]

Листинг 1.3.1 – структура `input_event` [2]

```
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)
};
```

Преимуществом данного подхода является простота реализации, однако, ввиду того что считывание данных подобным образом возможно реализовать в пространстве пользователя, данный способ не подходит для курсовой работы по операционным системам.

1.4 Keyboard notifier

При помощи вызова `register_keyboard_notifier` можно “подписаться” на события клавиатуры, в качестве аргумента передается структура `notifier_block`. В этой структуре поле `notifier_call` – указатель на функцию обработки поступающего события. Описание структур и заголовки функций предоставлены на листингах 1.4.1 и 1.4.2.

Листинг 1.4.1 – содержимое файла `notifier.h` [3]

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

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

Листинг 1.4.2 – содержимое файла keyboard.h [4]

```
struct notifier_block;
extern unsigned short *key_maps[MAX_NR_KEYMAPS];
extern unsigned short plain_map[NR_KEYS];

struct keyboard_notifier_param {
    struct vc_data *vc;      /* VC on which the keyboard press was done */
    int down;                /* Pressure of the key? */
    int shift;               /* Current shift mask */
    int ledstate;            /* Current led state */
    unsigned int value;      /* keycode, unicode value or keysym */
};

extern int register_keyboard_notifier(struct notifier_block *nb);
extern int unregister_keyboard_notifier(struct notifier_block *nb);
```

Данный подход подразумевает написание модуля ядра, внутри которого будут получены и обработаны события клавиатуры.

1.5 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. [5]

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 `include/linux/usb.h`.

Листинг 1.5.1 – структуры `usb_driver` и `usbdrv_wrap` (файл `usb.h`) [6]

```
/**
 * 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/`. [5]

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. [5]

“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. [7]

“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. [7]

Существует несколько способов реализации “нижней половины“ обработчика: гибкое прерывание (softirq), tasklet (tasklet) и очереди работ (workqueue). [5]

1.7 sys_call_table

Когда процесс запрашивает какую-либо услугу, используется механизм системных вызовов. Чтобы исполнить системный вызов, процесс заполняет регистры микропроцессора соответствующими значениями и выполняет специальную инструкцию, которая производит переход в predetermined место в пространстве ядра. Микропроцессор воспринимает это как переход из ограниченного пользовательского режима в защищенный режим ядра.

Если необходимо изменить поведение некоторого системного вызова, то первое, что необходимо сделать – это написать функцию, которая выполняет требуемые действия и вызывает первоначальную функцию, реализующую системный вызов, затем – изменить указатель в sys_call_table так, чтобы он указывал на новую функцию. Поскольку модуль впоследствии может быть

выгружен, то следует предусмотреть восстановление системы в ее первоначальное состояние, чтобы не оставлять ее в нестабильном состоянии. [8]

С целью предотвращения потенциальной опасности, связанной с подменой адресов системных вызовов, **ядро более не экспортирует sys_call_table** [8]. Поэтому, надлежит наложить "заплату" на ядро.

Подмена системных вызовов потенциально опасна и может стать причиной потери данных.

1.8 Передача данных в пространство пользователя

Поставленная задача подразумевает под собой передачу данных из пространства ядра в пространство пользователя. Для этой цели можно использовать файловую систему procfs. Она предоставляет все ресурсы для реализации интерфейса между пространством пользователя и пространством ядра. Часто используется в исходных кодах Linux.

Листинг 1.8.1 – основные поля структуры file_operations (/include/linux/fs.h) [9]

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
    ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
    // ...
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    // ...
}
```

Copy_to_user – функция копирования данных из пространства ядра в пространство пользователя. Возвращает количество нескопированных байт, т.е. 0 – успешное завершение функции. [10]

Листинг 1.8.2 – заголовок функции copy_to_user

```
#include <asm/uaccess.h>
int copy_to_user(void *dst, const void *src, unsigned int size);
```

1.9 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 использовать `keyboard notifier` для слежения за клавиатурой пользователя и написание драйвера для слежения за мышью.

2. Конструкторская часть

В данном разделе будут рассмотрены требования к программе, основные сведения о реализуемых модулях, предоставлены схемы, описывающие работу модуля - логгера.

2.1 Требования к программе

Необходимо реализовать загружаемый модуль ядра, который будет получать данные о событиях мыши и клавиатуры и предоставлять доступ к журналу этих событий через procfs.

Также необходимо реализовать драйвер usb мыши, который будет отсылать данные в модуль – логгер.

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-логгера, 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 int send_mouse_coordinates(char buttons, char dx, char dy, char wheel);
```

В листинге 2.2.2 предоставлен заголовок экспортируемой функции, вызов которой должен быть помещен в функцию `usb_mouse_irq`.

Buttons – состояние кнопок мыши, стандартно: 0 – нейтральное; 1 – нажата левая, 2 – правая кнопка мыши.

Dx, dy – смещение мыши по осям x и y соответственно.

Wheel – смещение колесика мыши.

В драйвере мыши эта информация хранится в `data`. На листинге 2.2.3 подробнее показан способ получения этой информации.

Листинг 2.2.3 – получение информации о событии мыши

```
struct usb_mouse *mouse = urb->context;
signed char *data = mouse->data;
char buttons = data[0];
char dx = data[1];
char dy = data[2];
char wheel = data[3];
```

2.3 Модуль - логгер

Для хранения журнала выделяется массив символов. При записи в лог очень важно не допускать переполнение. Для этих целей вводится переменная `my_log_len`, которая хранит количество занятых символов в логе на данный момент. При полном заполнении журнала, он сбрасывается и отсчет начинается с 0.

Листинг 2.3.1 – массив символов для журналирования

```
static int my_log_len;
static char my_log[MY_LOG_SIZE];
```

Как уже было выяснено выше, загружаемый модуль ядра для журналирования должен реализовывать экспортируемую драйвером мыши функцию. Полученные данные от драйвера должны сохраняться в лог.

Листинг 2.3.2 – функция обработки события мыши

```
extern int send_mouse_coordinates(char buttons, char dx, char dy, char wheel);
```

Также необходимо реализовать функцию `keylogger_notify`, указатель на которую устанавливается в структуру `notifier_block`, которая используется в качестве аргументов в функциях `register_keyboard_notifier` и `unregister_keyboard_notifier`.

Листинг 2.3.3 – заполнение структуры `notifier_block`

```
static struct notifier_block keylogger_nb = {
    .notifier_call = keylogger_notify
};
```

Также необходимо заполнить структуру `file_operations`, самое нужное поле в рамках поставленной задачи – `read`. В функции `procfs_read` нужно использовать функцию `copy_to_user` для копирования журнала модуля ядра в пространство пользователя.

Листинг 2.3.4 – заполнение структуры `file_operations`

```
int procfs_open(struct inode *inode, struct file *file);

static ssize_t procfs_read(struct file *filp, char *buffer, size_t length,
loff_t * offset);

static ssize_t procfs_write(struct file *file, const char *buffer, size_t
length, loff_t * off);

int procfs_close(struct inode *inode, struct file *file);

static const struct file_operations fops = {
    .owner = THIS_MODULE,
    .read = procfs_read,
    .write = procfs_write,
    .open = procfs_open,
    .release = procfs_close,
};
```

Листинг 2.3.5 – ключевые моменты функций инициализации и завершения работы модуля

```
static int __init procInit( void )
{
    // ...
    our_proc_file = proc_create(DEVICE_NAME, 0644 , NULL, &fops);
    if (!our_proc_file)
        return -ENOMEM; // out of memory
    register_keyboard_notifier(&keylogger_nb);
    // ...
    return 0;
}

static void __exit procExit( void )
{
    // ...
    unregister_keyboard_notifier(&keylogger_nb);
    remove_proc_entry(DEVICE_NAME, NULL);
    // ...
}
```

2.4 Схемы, демонстрирующие работу модуля

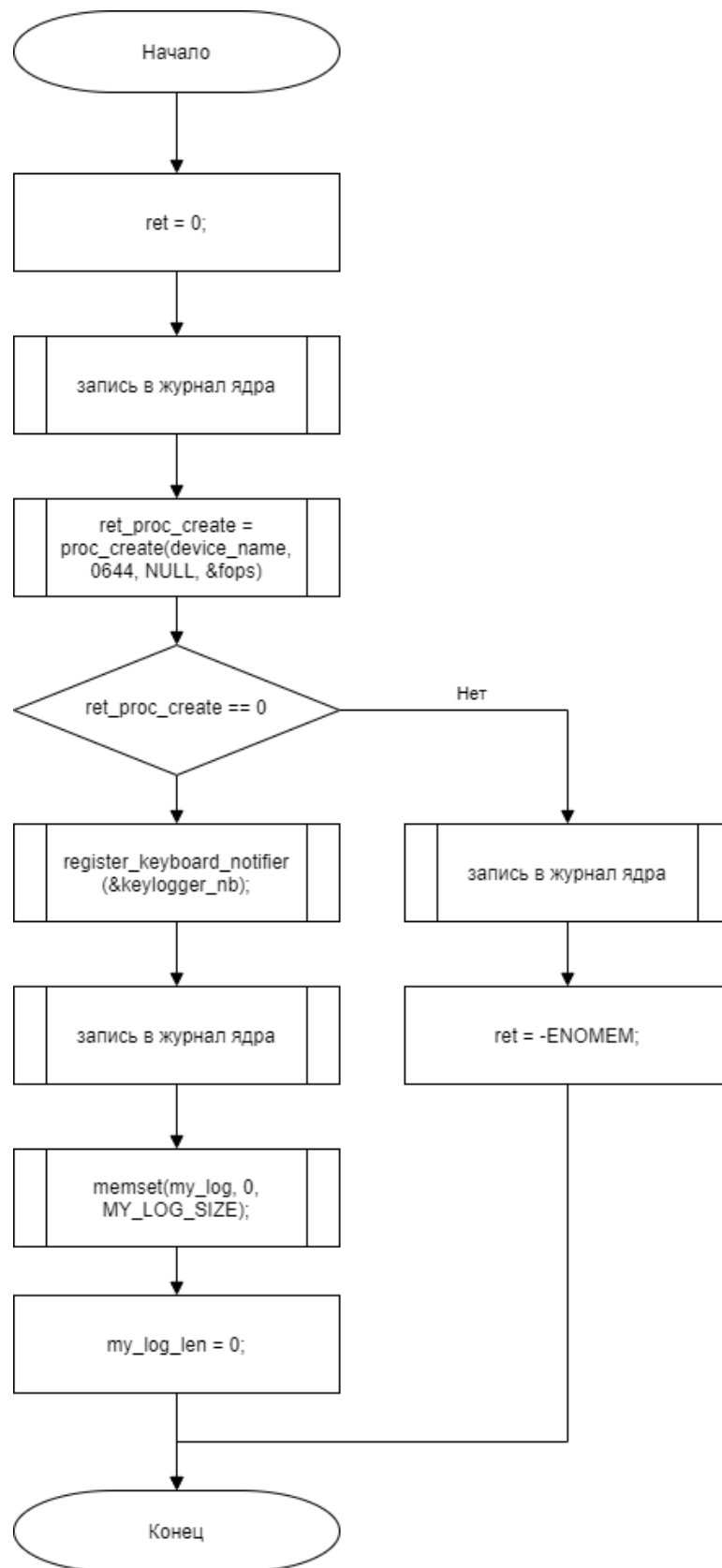


Рис. 2.4.1 – схема инициализации работы модуля

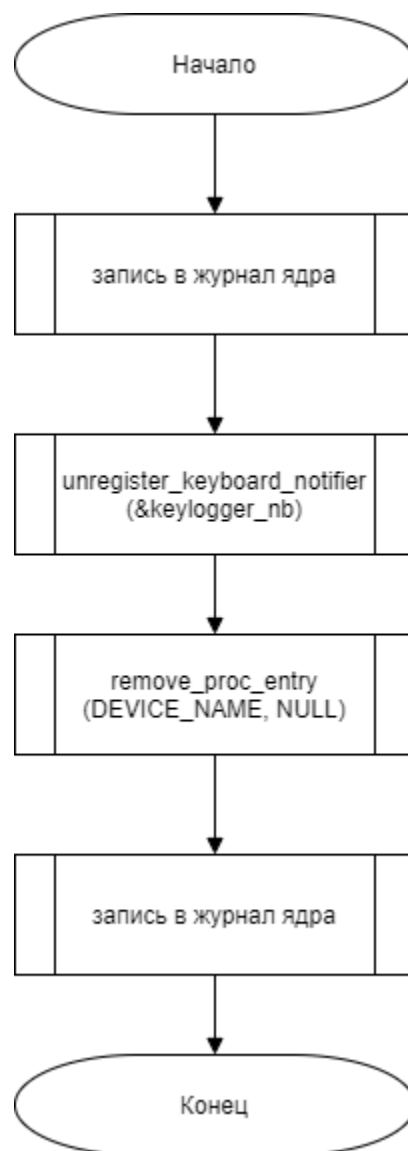


Рис. 2.4.2 – схема завершения работы модуля

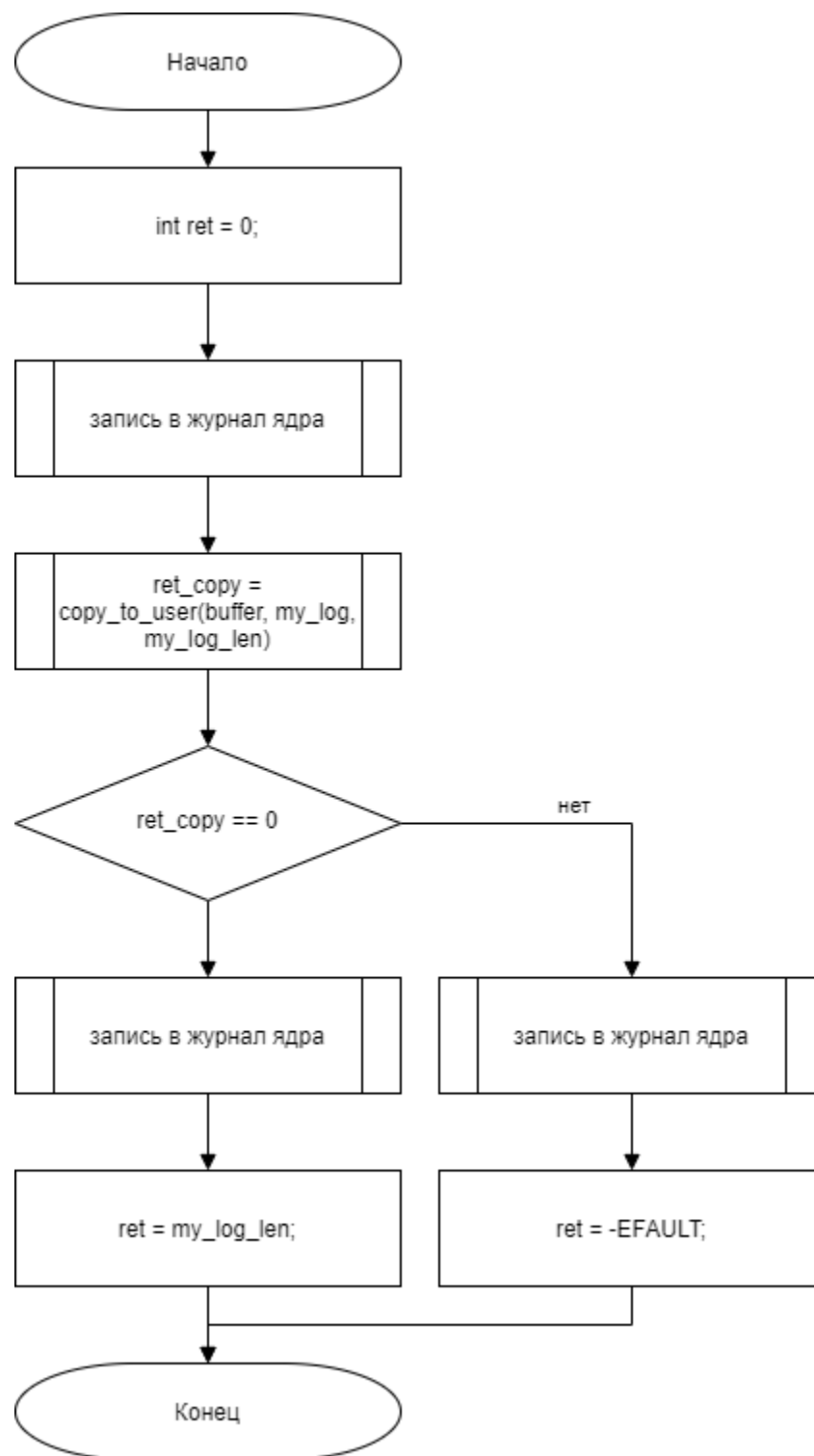


Рис. 2.4.3 – схема обработки чтения из /proc

2.5 Вывод

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

3. Технологическая часть

В данном разделе будет предоставлен листинг реализованных модулей и проведена апробация.

3.1 Выбор языка программирования и среды разработки

В качестве языка программирования был выбран С.

В качестве среды разработки была выбрана «Visual Studio Code» т.к. она бесплатна, кроссплатформенная, имеет множество плагинов для расширения функционала.

3.2 Модуль – логгер

Листинг 3.2.1 – файл my_logger.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/keyboard.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include "my_logger.h"

#define MODULE_INFO_PREFIX "myLogger"
extern int send_mouse_coordinates(char buttons, char dx, char dy, char wheel);

#define DEVICE_NAME "my_logger"
#define MY_LOG_SIZE 2048

static struct proc_dir_entry* our_proc_file;

static int my_log_len;
static char my_log[MY_LOG_SIZE];
static int isShiftKey = 0;

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Obergan T.M");
MODULE_DESCRIPTION("Logging mouse and keyboard");

/* keylogger_notify start */
int keylogger_notify(struct notifier_block *nblock, unsigned long code, void
*_param) {
    struct keyboard_notifier_param *param = _param;
    char buf[128];
    int len;

    if ( code == KBD_KEYCODE )
    {
        if( param->value==42 || param->value==54 )
        {
            if( param->down )
                isShiftKey = 1;
            else
                isShiftKey = 0;
        }
    }
}
```

```

        return NOTIFY_OK;
    }

    if( param->down )
    {
        // если не предусмотреть выход за границы массива,
        // то система может попросту зависнуть
        if (param->value > MyKeysMax)
            sprintf(buf, "keyboard: id-%d\n", param->value);
        if( isShiftKey == 0 )
            sprintf(buf, "keyboard: %s\n", MyKeys[param->value]);
        else
            sprintf(buf, "keyboard: shift + %s\n", MyKeys[param->value]);

        len = strlen(buf);
        if(len + my_log_len >= MY_LOG_SIZE)
        {
            memset(my_log, 0, MY_LOG_SIZE);
            my_log_len = 0;
        }
        strcat(my_log, buf);
        my_log_len += len;
    }

    return NOTIFY_OK;
}

static struct notifier_block keylogger_nb = {
    .notifier_call = keylogger_notify
};
/* keylogger_notify end */

/* procfs start */
int procfs_open(struct inode *inode, struct file *file)
{
    printk(KERN_INFO "%s in procfs_open\n", MODULE_INFO_PREFIX);
    try_module_get(THIS_MODULE);
    return 0;
}

static ssize_t procfs_read(struct file *filp, char *buffer, size_t length,
loff_t * offset)
{
    static int ret = 0;
    printk(KERN_INFO "%s in procfs_read\n", MODULE_INFO_PREFIX);

    if (ret)
        ret = 0;
    else
    {
        if ( raw_copy_to_user(buffer, my_log, my_log_len) )
            return -EFAULT;

        printk(KERN_INFO "%s read %lu bytes\n", MODULE_INFO_PREFIX, my_log_len);
        ret = my_log_len;
    }
    return ret;
}

static ssize_t procfs_write(struct file *file, const char *buffer, size_t
length, loff_t * off)
{

```

```

    printk(KERN_INFO "%s in procfs_write\n", MODULE_INFO_PREFIX);
    return 0;
}

int procfs_close(struct inode *inode, struct file *file)
{
    printk(KERN_INFO "%s in procfs_close\n", MODULE_INFO_PREFIX);
    module_put(THIS_MODULE);
    return 0;
}

static const struct file_operations fops = {
    .owner = THIS_MODULE,
    .read = procfs_read,
    .write = procfs_write,
    .open = procfs_open,
    .release = procfs_close,
};
/* procfs end */

static int __init procInit( void )
{
    printk(KERN_INFO "%s in init\n", MODULE_INFO_PREFIX);

    our_proc_file = proc_create(DEVICE_NAME, 0644 , NULL, &fops);
    if (!our_proc_file)
    {
        printk(KERN_INFO "%s proc_create %s failed\n", MODULE_INFO_PREFIX,
DEVICE_NAME);
        return -ENOMEM;
    }
    register_keyboard_notifier(&keylogger_nb);
    printk(KERN_INFO "%s logger registered\n", MODULE_INFO_PREFIX);

    memset(my_log, 0, MY_LOG_SIZE);
    my_log_len = 0;

    return 0;
}

static void __exit procExit( void )
{
    printk(KERN_INFO "%s in exit\n", MODULE_INFO_PREFIX);

    unregister_keyboard_notifier(&keylogger_nb);
    remove_proc_entry(DEVICE_NAME, NULL);
    printk(KERN_INFO "%s unregistered\n", MODULE_INFO_PREFIX);
}

extern int send_mouse_coordinates(char buttons, char dx, char dy, char wheel)
{
    printk(KERN_INFO "%s received send_mouse_coordinates %d %d %d %d\n",
MODULE_INFO_PREFIX, buttons, dx, dy, wheel);

    char buf[32];
    int len;

    sprintf(buf, "mouse: %d %d %d %d \n", buttons, dx, dy, wheel);

    len = strlen(buf);
    if(len + my_log_len >= MY_LOG_SIZE)
    {
        memset(my_log, 0, MY_LOG_SIZE);
    }
}

```

```

        my_log_len = 0;
    }
    strcat(my_log, buf);
    my_log_len += len;

    return 0;
}
EXPORT_SYMBOL(send_mouse_coordinates);

module_init(procInit);
module_exit(procExit);

```

3.3 Драйвер мыши

Листинг 3.3.1 – файл my_usb_mouse_driver.c

```

//основа кода взята из
https://github.com/torvalds/linux/blob/master/drivers/hid/usbhid/usbmouse.c

#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 "Obergan T.M"
#define DRIVER_DESC "USB HID Boot Protocol mouse driver"

#define MODULE_INFO_PREFIX "usbmouse:"
extern int send_mouse_coordinates(char buttons, char dx, char dy, char wheel);

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;

    if(send_mouse_coordinates(data[0], data[1], data[2], data[3]) != 0)
        printk(KERN_INFO "%s usb_mouse_irq can't send data\n",
MODULE_INFO_PREFIX);
}

```

```

        else
            printk(KERN_INFO "%s usb_mouse_irq success\n",
MODULE_INFO_PREFIX);

        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)
    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));

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 := my_logger.o my_usb_mouse_driver.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 Взаимодействие с модулями

На рис. 3.5.1 продемонстрирована загрузка модуля логгера при помощи команды `insmod` и показано содержимое `/proc/my_logger`.

На рис. 3.5.2 продемонстрирована выгрузка модуля логгера при помощи команды `rmmod` и показано содержимое, что `/proc/my_logger` более недоступна.

На рис. 3.5.3 продемонстрирована последовательность загрузки драйвера мыши. Модуль логгера должен быть загружен раньше драйвера.

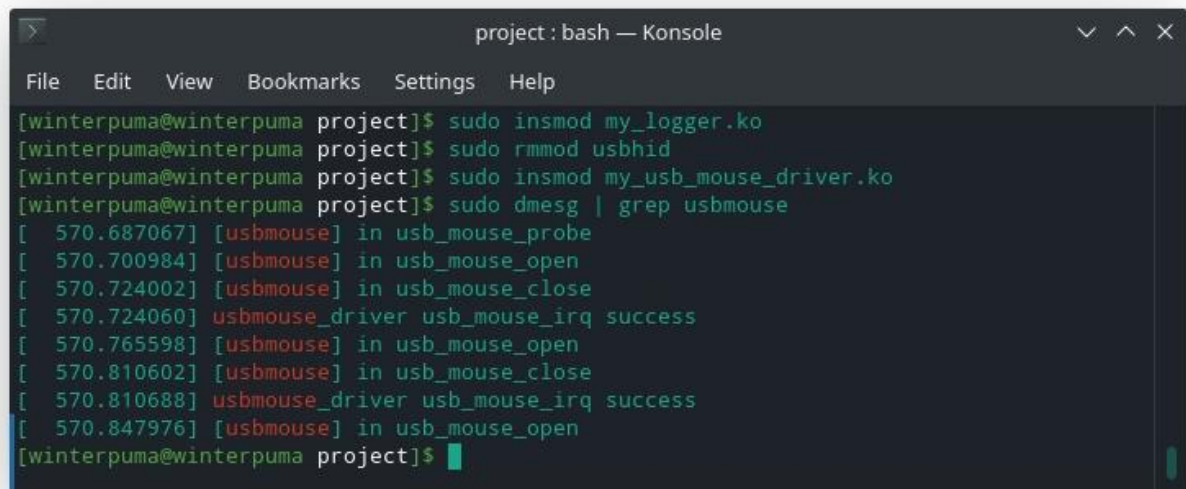
На рис 3.5.4 - 3.5.6 продемонстрировано соответствие событий пользовательского взаимодействия с мышью и соответствующих записей в журнале.

```
project : bash — Konsole
File Edit View Bookmarks Settings Help
[winterpuma@winterpuma project]$ sudo insmod my_logger.ko
[sudo] password for winterpuma:
[winterpuma@winterpuma project]$ sudo dmesg | grep myLogger
[ 296.155655] myLogger in init
[ 296.155658] myLogger logger registered
[winterpuma@winterpuma project]$ cat /proc/my_logger
keyboard: s
keyboard: u
keyboard: d
keyboard: o
keyboard: SPACE
keyboard: d
keyboard: m
keyboard: e
keyboard: s
keyboard: g
keyboard: SPACE
keyboard: shift + \
keyboard: SPACE
keyboard: g
keyboard: r
keyboard: e
keyboard: p
keyboard: SPACE
keyboard: m
keyboard: y
keyboard: shift + l
keyboard: o
keyboard: g
keyboard: g
keyboard: e
keyboard: r
keyboard: ENTER
keyboard: /
keyboard: BACKSPACE
```

Рис. 3.5.1 – загрузка модуля логгера

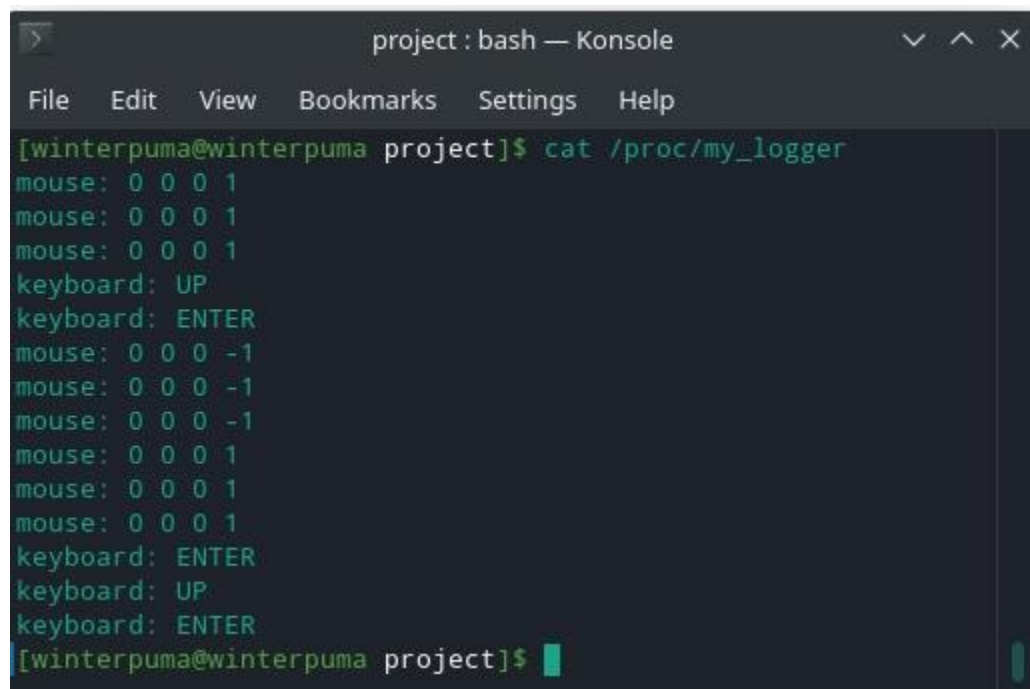
```
project : bash — Konsole
File Edit View Bookmarks Settings Help
[winterpuma@winterpuma project]$ sudo rmmod my_logger
[winterpuma@winterpuma project]$ sudo dmesg | grep myLogger
[ 296.155655] myLogger in init
[ 296.155658] myLogger logger registered
[ 325.810013] myLogger in procfs_open
[ 325.810039] myLogger in procfs_read
[ 325.810044] myLogger read 648 bytes
[ 325.810111] myLogger in procfs_read
[ 325.810133] myLogger in procfs_close
[ 418.915009] myLogger in exit
[ 418.933938] myLogger unregistered
[winterpuma@winterpuma project]$ cat /proc/my_logger
cat: /proc/my_logger: No such file or directory
[winterpuma@winterpuma project]$
```

Рис 3.5.2 – выгрузка модуля логгера



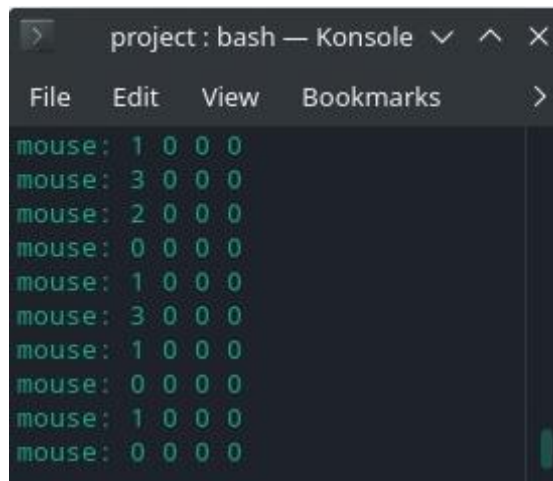
```
project : bash — Konsole
File Edit View Bookmarks Settings Help
[winterpuma@winterpuma project]$ sudo insmod my_logger.ko
[winterpuma@winterpuma project]$ sudo rmmod usbhid
[winterpuma@winterpuma project]$ sudo insmod my_usb_mouse_driver.ko
[winterpuma@winterpuma project]$ sudo dmesg | grep usbmouse
[ 570.687067] [usbmouse] in usb_mouse_probe
[ 570.700984] [usbmouse] in usb_mouse_open
[ 570.724002] [usbmouse] in usb_mouse_close
[ 570.724060] usbmouse_driver usb_mouse_irq success
[ 570.765598] [usbmouse] in usb_mouse_open
[ 570.810602] [usbmouse] in usb_mouse_close
[ 570.810688] usbmouse_driver usb_mouse_irq success
[ 570.847976] [usbmouse] in usb_mouse_open
[winterpuma@winterpuma project]$
```

Рис 3.5.3 – загрузка модуля драйвера



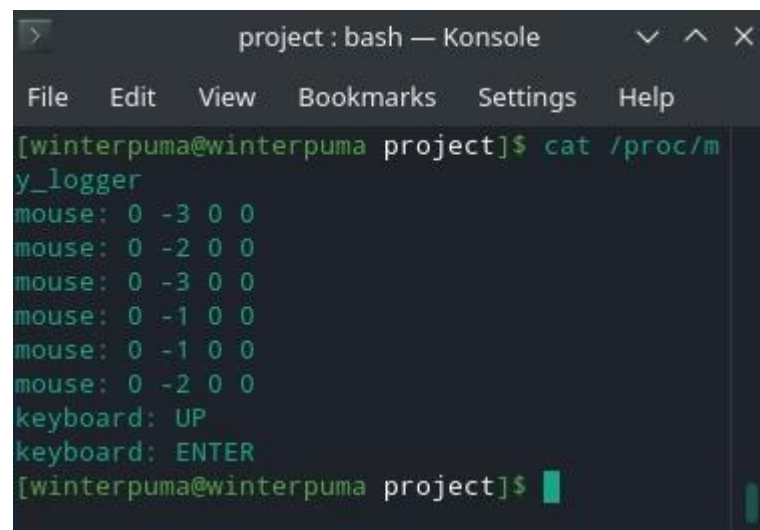
```
project : bash — Konsole
File Edit View Bookmarks Settings Help
[winterpuma@winterpuma project]$ cat /proc/my_logger
mouse: 0 0 0 1
mouse: 0 0 0 1
mouse: 0 0 0 1
keyboard: UP
keyboard: ENTER
mouse: 0 0 0 -1
mouse: 0 0 0 -1
mouse: 0 0 0 -1
mouse: 0 0 0 1
mouse: 0 0 0 1
mouse: 0 0 0 1
keyboard: ENTER
keyboard: UP
keyboard: ENTER
[winterpuma@winterpuma project]$
```

Рис 3.5.4 – движение колесиком мыши



```
project : bash — Konsole
File Edit View Bookmarks
mouse: 1 0 0 0
mouse: 3 0 0 0
mouse: 2 0 0 0
mouse: 0 0 0 0
mouse: 1 0 0 0
mouse: 3 0 0 0
mouse: 1 0 0 0
mouse: 0 0 0 0
mouse: 1 0 0 0
mouse: 0 0 0 0
```

Рис. 3.5.5 – нажатие клавиш мыши



```
project : bash — Konsole
File Edit View Bookmarks Settings Help
[winterpuma@winterpuma project]$ cat /proc/mouse
y_logger
mouse: 0 -3 0 0
mouse: 0 -2 0 0
mouse: 0 -3 0 0
mouse: 0 -1 0 0
mouse: 0 -1 0 0
mouse: 0 -2 0 0
keyboard: UP
keyboard: ENTER
[winterpuma@winterpuma project]$
```

Рис. 3.5.6 – движение мыши влево

3.6 Вывод

Были реализованы модуль драйвера usb мыши и модуль-логгер, листинг которых был предоставлен в данном разделе. Также была проведена апробация различных сценариев использования устройств: нажатие клавиш клавиатуры и мыши, движение мыши и ее колесика.

Заключение

Во время выполнения курсового проекта были достигнуты поставленные цель и задачи: проанализированы способы перехвата сообщений от мыши и клавиатуры; проанализирована структура драйвера мыши; проанализированы методы передачи информации из модулей ядра в пространство пользователя; спроектированы и реализованы модуль ядра и драйвер.

Были реализованы загрузаемый модуль ядра, выполняющий перехват сообщений клавиатуры и мыши и драйвер USB мыши.

В ходе выполнения поставленных задач были изучены возможности языка С, получены знания в области написания загрузаемых 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. Исходный код файла notifier.h [Электронный ресурс]. – Режим доступа: <https://github.com/torvalds/linux/blob/master/include/linux/notifier.h>
4. Исходный код файла keyboard.h [Электронный ресурс]. – Режим доступа: <https://github.com/torvalds/linux/blob/master/include/linux/keyboard.h>
5. 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.
6. Исходный код файла usb.h [Электронный ресурс]. – Режим доступа: <https://github.com/torvalds/linux/blob/master/include/linux/usb.h>
7. Corbet J., Rubini A., Kroan-Hartman G. Linux device drivers. O'Reilly Media, 2005. 567 p.
8. Peter Jay Salzman, Michael Burian, Ori Pomerantz. The Linux Kernel Module Programming Guide. 2007, ver. 2.6.4.
9. Исходный код файла fs.h [Электронный ресурс]. – Режим доступа: <https://github.com/torvalds/linux/blob/master/include/linux/fs.h>
10. Robert Love. Linux Kernel: How does copy_to_user work? [Электронный ресурс]. – Режим доступа: https://www.quora.com/Linux-Kernel-How-does-copy_to_user-work