Государственное образовательное учреждение высшего профессионального образования

|  |  |
| --- | --- |
| Описание: Gerb-BMSTU_01 | ***«Московский государственный технический университет  имени Н.Э. Баумана»***  ***(МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** Информатика и системы управления

**КАФЕДРА** Программное обеспечение ЭВМ и информационные технологии

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту по операционным системам на тему:**

Перехват сообщений USB-мыши и их журналирование

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Копьёва М.В. )

(ф.и.о.)

Руководитель курсового проекта\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Алексеев Ю.Е. )

(ф.и.о.)

Москва, 2019

МГТУ им. Н.Э. Баумана

**Реферат**

РПЗ 31 с., 7 рис., 5 источников

Объектом исследования является часть ядра Linux версии 4.4, отвечающая за управление USB-мышью.

Цель работы - разработка программного комплекса для перехвата сообщений USB-мыши и их журналирования.

В процессе работы разработан программный комплекс, состоящий из загружаемого модуля ядра, дополненного системного драйвера Linux и приложения пользовательского уровня.

Дополнения драйвера реализуют передачу данных загружаемому модулю.

Загружаемый модуль ядра реализует получение данных от драйвера мыши и предоставляет их пользователю посредством файловой системы /proc.

Приложение уровня пользователя позволяет визуализировать полученные данные и реализует ведение журнала.

Оглавление

[Введение 6](#_Toc405736601)

[1. Аналитический раздел 7](#_Toc405736602)

[1.1. Анализ задачи 7](#_Toc405736603)

[1.2. Анализ подходов к реализации 7](#_Toc405736604)

[1.3. USB-драйверы в Linux 8](#_Toc405736605)

[1.3.1. Поддержка USB в ядре Linux 8](#_Toc405736606)

[1.3.2. Регистрация/выгрузка драйвера 8](#_Toc405736607)

[1.3.3. Регистрация устройства 9](#_Toc405736608)

[1.3.4. Выводы 10](#_Toc405736609)

[1.4. Перехват сообщений USB-мыши 10](#_Toc405736610)

[1.4.1. Анализ usbmouse.c 10](#_Toc405736611)

[1.4.2. Анализ функции usb\_mouse\_irq 12](#_Toc405736612)

[1.5. Передача данных в пространство пользователя 13](#_Toc405736613)

[2. Конструкторский раздел 14](#_Toc405736614)

[2.1. Структура программного обеспечения 14](#_Toc405736615)

[2.2. Перехват сообщений 14](#_Toc405736616)

[2.3. Хранение информации в модуле ядра 14](#_Toc405736617)

[2.4. Разработка загружаемого модуля ядра 15](#_Toc405736618)

[2.4.1. Алгоритм загрузки модуля 15](#_Toc405736619)

[2.4.2. Алгоритм записи перехваченных данных 15](#_Toc405736620)

[2.4.3. Алгоритм открытия файла в /proc 15](#_Toc405736621)

[2.4.4. Алгоритм чтения данных из файла в /proc 16](#_Toc405736622)

[2.5. Разработка пользовательского приложения 18](#_Toc405736623)

[2.5.1. Диаграмма классов 19](#_Toc405736624)

[2.5.2. Алгоритм чтения данных из файла в /proc 19](#_Toc405736625)

[3. Технологический раздел 21](#_Toc405736626)

[3.1. Выбор языка программирования и средств разработки 21](#_Toc405736627)

[3.2. Модуль ядра 21](#_Toc405736628)

[3.3. Исходный код программы 22](#_Toc405736629)

[3.3.1. Файл mouseListener.h 22](#_Toc405736630)

[3.3.2. Файл mouseListener.c 23](#_Toc405736631)

[3.3.3. Файл mainwindow.h 25](#_Toc405736632)

[3.3.4. Файл mainwindow.cpp 25](#_Toc405736633)

[3.3.5. Файл canvas.h 27](#_Toc405736634)

[3.3.6. Файл canvas.cpp 28](#_Toc405736635)

[3.3.7. Файл datataker.h 29](#_Toc405736636)

[3.3.8. Файл datataker.cpp 29](#_Toc405736637)

[Заключение 30](#_Toc405736638)

[Список литературы 31](#_Toc405736639)

# Введение

Мышь - интерактивное средство ввода-вывода. Прерывания от мыши возникают очень часто. Важно получать информацию о событиях, инициируемых мышью. Эта информация может использоваться для оптимизации механизмов обработки прерываний в системе.

Кроме того, полученные низкоуровневые данные в дальнейшем могут использоваться для восстановления последовательности действий определённого пользователя и использования их в качестве системного тестирования приложений (запомнить и повторить действия). Также можно тестировать и интерфейс приложения: если пользователь двигает мышку слишком много и «размашисто», можно заключить, что интерфейс не удобный, и расположить соответствующие компоненты графического интерфейса ближе друг к другу.

Также анализ этих данных может дать информацию о психологическом состоянии пользователя: насколько резко он дёргает мышь, как часть крутит колёсико, как часто нажимает на кнопки.

Проект посвящен фиксации перемещений и нажатий кнопок мыши в журнале и передаче собранной информации из пространства ядра в пространство пользователя.

Целью работы является разработка программного комплекса для перехвата сообщений USB мыши и их журналирования.

# Аналитический раздел

## Анализ задачи

В соответствии с заданием на курсовой проект необходимо разработать программное обеспечение, фиксирующее события в системе, инициирующиеся средством ввода информации – мышью.

Мышь формирует события: нажатий кнопок, перемещений и движения колёсика.

Программное обеспечение должно обеспечивать перехват всех действий мыши и фиксировать эту информацию в файле, для того, чтобы в дальнейшем было возможно произвести анализ этой информации.

Для решения этой цели необходимо решить следующие задачи:

* Проанализировать реализацию USB интерфейса в системе Linux.
* Проанализировать способы перехвата сообщений от USB устройств.
* Проанализировать структуру драйвера USB мыши.
* Проанализировать методы передачи информации из модулей ядра в пространство пользователя.
* Спроектировать и реализовать модуль ядра.
* Спроектировать и реализовать программное обеспечение уровня пользователя.

## Анализ подходов к реализации

Известно несколько способов решения данной задачи:

* чтение информации из системного файла устройства “/dev/input/event\*”;
* перехват сообщений мыши в пространстве ядра.

Чтение файла “/dev/input/event\*” возможно реализовать в пространстве пользователя. Второй вариант подразумевает под собой написание модуля ядра и исследование драйвера USB мыши. Также перехват сообщений в модуле предоставляет более низкоуровневый доступ к данным, приходящим от мыши. Именно поэтому предпочтение отдается второму варианту.

## USB-драйверы в Linux

### Поддержка USB в ядре Linux

Программный интерфейс для взаимодействия с USB устройствами в ядре Linux очень прост. За простым интерфейсом скрываются все алгоритмы отсылки запросов, отслеживания подтверждений, контроля ошибок и т.п.

Драйвер, взаимодействующий с USB-устройством(-ами), как правило, выполняет следующие действия:

1. регистрация/выгрузка драйвера;
2. регистрация/удаление устройства;
3. обмен данными: управляющий и информационный.

### Регистрация/выгрузка драйвера

По USB может передаваться несколько типов пакетов:

Регистрация USB-драйвера подразумевает:

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

Рассмотрим наиболее важные поля cтруктура usb\_driver.

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 device\_driver driver**;**

// ...

**};**

Очевидно, что name - это имя драйвера. id\_table - это массив структур usb\_device\_id. Этот список предназначен для определения cоответствия подключаемого устройства определенным параметрам. Только те устройства, которые соответствуют перечисленным параметрам, могут быть подключены к драйверу. Если массив пуст, система будет пытаться каждое устройство к драйверу. Поле driver говорит о том, что usb\_driver унаследован от device\_driver. В самом простом случае каждый элемент id\_table[i] содержит пару идентификаторов:

* идентификатор производителя (Vendor ID);
* идентификатор устройства (Device ID).

probe и disconnect - это callback-функции, вызываемые системой при подключении и отключении USB-устройства. probe будет вызвана для каждого устройства, если список id\_table пуст, или только для тех устройств, которые соответствуют параметрам, перечисленным в списке.

### Регистрация устройства

Один зарегистрированный драйвер может "подключать" несколько устройств. Для подключения устройства к драйверу система вызывает функцию драйвера probe, которой передает 2 параметра:

static int my\_probe**(**struct usb\_interface **\***interface**,**

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 my\_disconnect**(**struct usb\_interface **\***interface**)**

**{**

// ...

**}**

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.

### Выводы

При регистрации устройства должна указываться функция обработки данных, приходящих от этого устройства. В эту функцию может быть вставлен перехватчик данных, которые в дальнейшем можно передать в пространство пользователя.

## Перехват сообщений USB-мыши

За управление USB-устройствами в Linux отвечает модуль usbhid. Проанализируем файл Linux/drivers/hid/usbhid/Makefile:

#

# Makefile for the USB input drivers

#

# Multipart objects.

usbhid-y **:=** hid-core.o hid-quirks.o

# Optional parts of multipart objects.

ifeq ($(CONFIG\_USB\_HIDDEV),y)

usbhid-y +**=** hiddev.o

endif

ifeq ($(CONFIG\_HID\_PID),y)

usbhid-y +**=** hid-pidff.o

endif

obj-$(CONFIG\_USB\_HID) +**=** usbhid.o

obj-$(CONFIG\_USB\_KBD) +**=** usbkbd.o

obj-$(CONFIG\_USB\_MOUSE) +**=** usbmouse.o

Модуль собирается из объектных файлов usbhid.o, usbkbd.o и usbmouse.o. Соответственно код драйвера мыши можно найти в файле usbmouse.c.

### Анализ usbmouse.c

Как было решено ранее, нужно в функции (\*probe) найти функцию, в которой данные, пришедшие от мыши, будут обрабатываться и передаваться в пользовательское пространство, и вставить в эту функцию свой перехватчик.

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**;**

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**;**

**}**

Данные передаются блоками URB (USB Request Block). Интерес представляет строка

usb\_fill\_int\_urb**(**mouse**->**irq**,** dev**,** pipe**,** mouse**->**data**,**

**(**maxp **>** 8 **?** 8 **:** maxp**),**

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

Анализ заголовка этой функции показывает, что обработка данных, приходящих от мыши происходит в функции usb\_mouse\_irq:

static inline void usb\_fill\_int\_urb**(**struct urb **\***urb**,**

struct usb\_device **\***dev**,**

unsigned int pipe**,**

void **\***transfer\_buffer**,**

int buffer\_length**,**

**usb\_complete\_t complete\_fn,**

void **\***context**,**

int interval**)**

### Анализ функции usb\_mouse\_irq

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**;**

**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**);**

**}**

Если URB принят без ошибок, то идёт передача данных в пользовательское пространство с помощью функций input\_report\_key и input\_report\_rel, которые вызывают input\_event.

## Передача данных в пространство пользователя

Поставленная задача подразумевает под собой передачу данных из пространства ядра в пространство пользователя для дальнейшей обработки. Для передачи данных в пространство пользователя существует файловая система procfs. Она предоставляет все ресурсы для реализации интерфейса между пространством пользователя и пространством ядра. Часто используется в исходных кодах Linux.

# Конструкторский раздел

## Структура программного обеспечения

В соответствии с анализом задачи в состав программного обеспечения будет входить измененный драйвер USB-мыши, подлежащие разработке модуль ядра для перехвата и промежуточного хранения сообщений, и программа пространства пользователя для вывода перехваченных сообщений на экране (рис. 1).

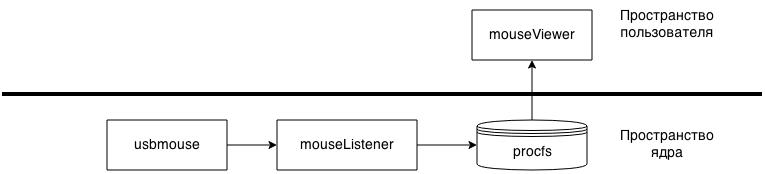


Рисунок 1 - Структура программного обеспечения.

## Перехват сообщений

Как было выяснено в аналитическом разделе, сообщения, отправленные устройством, обрабатываются функцией usb\_mouse\_irq. Необходимо поместить в эту процедуру вызов экспортируемой процедуры из разрабатываемого модуля ядра, в которую будут передаваться данные, пришедшие от мыши.

Данный драйвер мыши принимает стандартный набор данных. Это состояние кнопок мыши, относительное перемещение мыши по осям X и Y, и смещение колёсика мыши. Все эти данные хранятся в переменной data. Прототип функции решено сделать на основе существующего способа хранения в драйвере мыши.

extern bool mouseListenerSendCoordinates**(**char buttons**,** char dx**,** char dy**,** char wheel**);**

## Хранение информации в модуле ядра

Для промежуточного хранения данных, перехваченных в драйвере мыши, разработана структура данных.

struct mouseInfoItem

**{**

int time**;**

char btns**;**

char dx**;**

char dy**;**

char wheel**;**

**};**

Необходимо будет предусмотреть хранение фиксированного массива типа этой структуры.

## Разработка загружаемого модуля ядра

В соответствии с аналитическим разделом модуль ядра должен решать следующие задачи:

* инициализация файла в файловой системе /proc;
* приём и хранение в буфере данных, принятых с помощью экспортируемой функции;
* определение функции открытия файла файловой системы /proc;
* определение функции чтения из файла файловой системы /proc;
* определение функции закрытия файла файловой системы /proc.

### Алгоритм загрузки модуля

На рисунке 2 представлена структура алгоритма инициализации загружаемого модуля ядра.

### Алгоритм записи перехваченных данных

На рисунке 3 представлена структура алгоритма получения и хранения перехваченных в драйвере данных от USB-мыши.

### Алгоритм открытия файла в /proc

На рисунке 4 представлена структура алгоритма открытия созданного файла в файловой системе /proc.

### Алгоритм чтения данных из файла в /proc

На рисунке 5 представлена схема алгоритма чтения из созданного файла в файловой системе /proc.

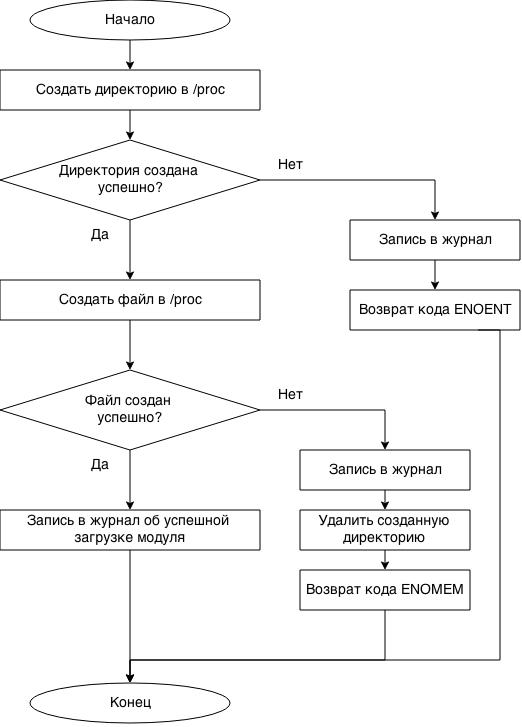


Рисунок 2 - Схема алгоритма загрузки модуля

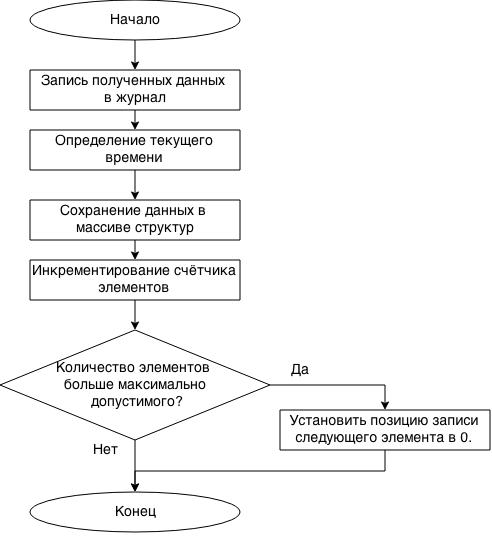


Рисунок 3 - Схема алгоритма записи в буфер

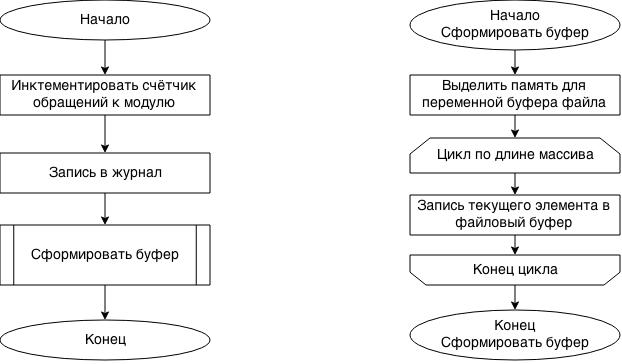


Рисунок 4 - Схема алгоритма открытия файла в /proc

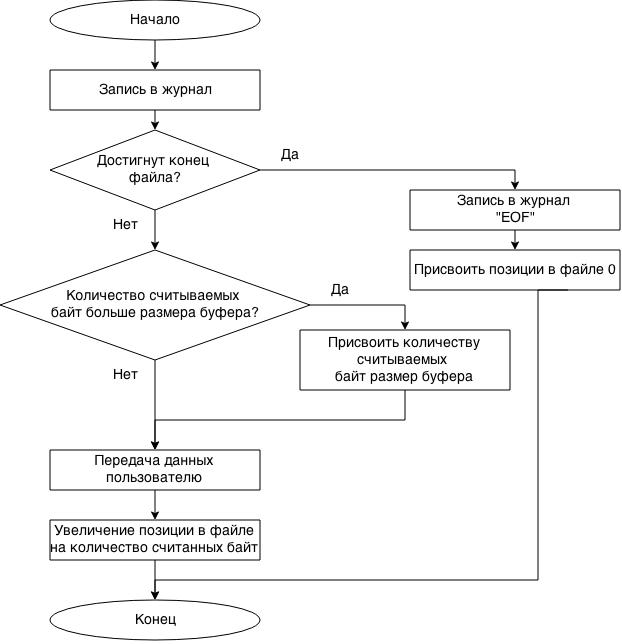


Рисунок 5 - Схема алгоритма чтения файла из /proc

## Разработка пользовательского приложения

В пространстве пользователя необходимо решать 3 задачи:

* получение данных;
* демонстрация полученных данных пользователю;
* ведение журнала.

Решено использовать объектно-ориентированный подход для реализации пользовательского приложения. Это оправдано гибкостью, расширяемостью и быстротой реализации ПО.

### Диаграмма классов

Для реализации поставленных задач необходимо разработать следующие классы.

* MainWindow – управляющий. Отвечает за логику работы пользовательской программы.
* Canvas – класс для демонстрации пользователю перехваченных данных.
* DataTaker – класс, отвечающий за приём и передачу данных из файла в файловой системе /proc.

На рисунке 6 представлена диаграмма классов.

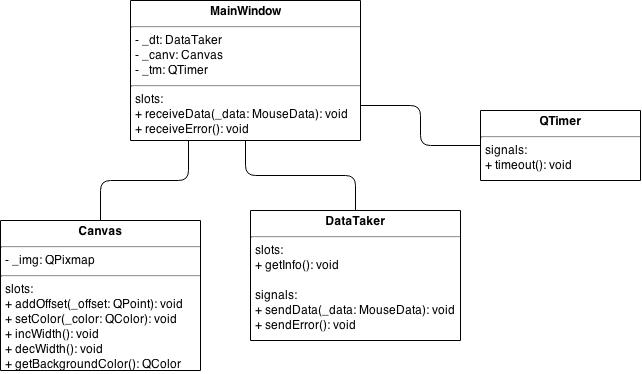


Рисунок 6 - UML-диаграмма классов

### Алгоритм чтения данных из файла в /proc

На рисунке 7 представлена структура алгоритма чтения данных из файла в файловой системе /proc.

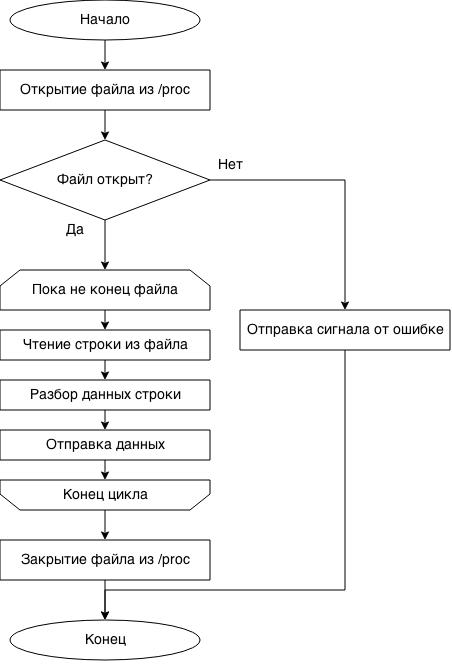


Рисунок 7 - Схема алгоритма чтения из /proc

# Технологический раздел

## Выбор языка программирования и средств разработки

Для сборки модулей ядра используется специальный компилятор gcc.

Для разработки приложения уровня пользователя выбран язык С++.

Среда разработки Qt. Это обусловлено большим набором библиотек ускоряющих разработку.

## Модуль ядра

В соответствии с алгоритмами и решениями, принятыми в конструкторсоком разделе был написан программный код пользовательского приложения.

Модуль ядра будет собираться из 3х файлов.

* mouseListenerExtern.h // определяет экспортируемую функцию mouseListenerSendCoordinates
* mouseListener.h // заголовочный файл
* mouseListener.c // файл реализации

Makefile для сборки модуля:

ifneq ($(KERNELRELEASE),)

obj-m := mouseListener.o

else

CURRENT = $(shell uname -r)

KDIR = /lib/modules/$(CURRENT)/build

PWD = $(shell pwd)

all:

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

clean:

@rm -f \*.o .\*.cmd .\*.flags \*.mod.c \*.order

@rm -f .\*.\*.cmd \*~ \*.\*~ TODO.\*

@rm -fR .tmp\*

@rm -rf .tmp\_versions

disclean: clean

@rm \*.ko \*.symvers

endif

Makefile для сборки дополненного перехватчиком драйвера:

ifneq ($(KERNELRELEASE),)

obj-m := usbmouse.o

else

CURRENT = $(shell uname -r)

KDIR = /lib/modules/$(CURRENT)/build

PWD = $(shell pwd)

all:

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

clean:

@rm -f \*.o .\*.cmd .\*.flags \*.mod.c \*.order

@rm -f .\*.\*.cmd \*~ \*.\*~ TODO.\*

@rm -fR .tmp\*

@rm -rf .tmp\_versions

disclean: clean

@rm \*.ko \*.symvers

endif

clean**:**

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

Для использования файловой системы /proc необходимо создать в ней файл и определить функции для работы с этим файлом в структуре:

static const struct file\_operations nodeFops **=**

**{**

**.**owner **=** THIS\_MODULE**,**

**.**open **=** nodeOpen**,**

**.**read **=** nodeRead**,**

**.**release **=** nodeClose

**};**

Функции для создания директории и файла в /proc для ядра версии 4.4:

procDir **=** proc\_mkdir\_mode**(**DIR\_NAME**,** S\_IFDIR **|** S\_IRWXUGO**,** **NULL);**

procNode **=** proc\_create\_data**(**NODE\_NAME**,** S\_IFREG **|** S\_IRUGO **|** S\_IWUGO**,**

procDir**,** **&**nodeFops**,** **NULL)**

## Исходный код программы

### Файл mouseListener.h

#include <linux/module.h>

#include <linux/proc\_fs.h>

#include <linux/stat.h>

#include <linux/slab.h>

#include <linux/init.h>

#include <asm/uaccess.h>

#include <linux/cdev.h>

#include "mouseListenerExtern.h"

#define DIR\_NAME "mouseListener"

#define NODE\_NAME "info"

MODULE\_LICENSE**(**"GPL"**);**

MODULE\_AUTHOR**(**"Kopyova Margarita IU7-72"**);**

static struct proc\_dir\_entry **\***procDir**;**

static int \_\_init procInit**(** void **);**

static void \_\_exit procExit**(** void **);**

static int nodeOpen**(**struct inode **\***inode**,** struct file **\***file**);**

static ssize\_t nodeRead**(**struct file **\***file**,** char **\***buf**,** size\_t count**,** loff\_t **\***ppos**);**

static int nodeClose**(**struct inode **\***inode**,** struct file **\***file**);**

void makeMessage**(**void**);**

static const struct file\_operations nodeFops **=**

**{**

**.**owner **=** THIS\_MODULE**,**

**.**open **=** nodeOpen**,**

**.**read **=** nodeRead**,**

**.**release **=** nodeClose

**};**

struct mouseInfoItem

**{**

int time**;**

char btns**;**

char dx**;**

char dy**;**

char wheel**;**

**};**

struct mouseInfoItem **\***mouseInfo**;**

int mouseInfoMaxLen **=** 100**;**

int mouseInfoLen **=** 0**;**

char **\***mouseInfoMsg**;**

module\_init**(**procInit**);**

module\_exit**(**procExit**);**

### Файл mouseListener.c

#include "mouseListener.h"

static int \_\_init procInit**(** void **)**

**{**

struct proc\_dir\_entry **\***procNode**;**

int ret**;**

printk**(**KERN\_CRIT "Starting mouse listener\n"**);**

printk**(**KERN\_CRIT "Memory allocated\n"**);**

**if((**procDir **=** proc\_mkdir\_mode**(**DIR\_NAME**,** S\_IFDIR **|** S\_IRWXUGO**,** **NULL))** **==** **NULL)**

**{**

ret **=** **-**ENOENT**;**

printk**(**KERN\_CRIT "Can't create dir /proc/%s\n"**,** DIR\_NAME**);**

**return** ret**;**

**}**

printk**(**KERN\_CRIT "Directory /proc/%s created"**,** DIR\_NAME**);**

**if((**procNode **=** proc\_create\_data**(**NODE\_NAME**,** S\_IFREG **|** S\_IRUGO **|** S\_IWUGO**,**

procDir**,** **&**nodeFops**,** **NULL))** **==** **NULL)**

**{**

ret **=** **-**ENOMEM**;**

printk**(**KERN\_CRIT "Can't create node /proc/%s/%s\n"**,** DIR\_NAME**,** NODE\_NAME**);**

remove\_proc\_entry**(**DIR\_NAME**,** **NULL);**

**return** ret**;**

**}**

mouseInfo **=** **(**struct mouseInfoItem**\*)**

kmalloc**(sizeof(**struct mouseInfoItem**)\***mouseInfoMaxLen**,** GFP\_KERNEL**);**

printk**(**KERN\_CRIT "Node /proc/%s/%s installed"**,** DIR\_NAME**,** NODE\_NAME**);**

printk**(**KERN\_CRIT "Init finished\n"**);**

**return** 0**;**

**}**

static void \_\_exit procExit**(** void **)**

**{**

kfree**(**mouseInfoMsg**);**

kfree**(**mouseInfo**);**

remove\_proc\_entry**(**NODE\_NAME**,** procDir**);**

remove\_proc\_entry**(**DIR\_NAME**,** **NULL);**

**}**

extern bool mouseListenerSendCoordinates**(**char \_btns**,** char \_dx**,** char \_dy**,** char \_wheel**)**

**{**

struct timeval**\*** time**;**

printk**(**KERN\_CRIT "Received %d (%d, %d), wheel %d\n"**,** \_btns**,** \_dx**,** \_dy**,** \_wheel**);**

time **=** **(**struct timeval**\*)**kmalloc**(sizeof(**struct timeval**),** GFP\_KERNEL**);**

do\_gettimeofday**(**time**);**

mouseInfo**[**mouseInfoLen**].**time **=** time**->**tv\_sec**;**

mouseInfo**[**mouseInfoLen**].**btns **=** \_btns**;**

mouseInfo**[**mouseInfoLen**].**dx **=** \_dx**;**

mouseInfo**[**mouseInfoLen**].**dy **=** \_dy**;**

mouseInfo**[**mouseInfoLen**].**wheel **=** \_wheel**;**

**++**mouseInfoLen**;**

**if(**mouseInfoLen **>=** mouseInfoMaxLen**)**

mouseInfoLen **=** 0**;**

kfree**(**time**);**

**return** 1**;**

**}**

EXPORT\_SYMBOL**(**mouseListenerSendCoordinates**);**

static int nodeOpen**(**struct inode**\*** inode**,** struct file**\*** file**)**

**{**

try\_module\_get**(**THIS\_MODULE**);**

printk**(**KERN\_CRIT "File opened\n"**);**

makeMessage**();**

**return** 0**;**

**}**

void makeMessage**(**void**)**

**{**

int i**;**

mouseInfoMsg **=** **(**char**\*)**kmalloc**(**1 **+** mouseInfoMaxLen**\*(**5 **+** **sizeof(**struct mouseInfoItem**)),** GFP\_KERNEL**);**

sprintf**(**mouseInfoMsg**,** ""**);**

**for(**i **=** 0**;** i **<** mouseInfoLen**;** **++**i**)**

**{**

sprintf**(**mouseInfoMsg**,** "%s%d\t%d\t%d\t%d\t%d\n"**,** mouseInfoMsg**,**

mouseInfo**[**i**].**time**,** mouseInfo**[**i**].**btns**,**

mouseInfo**[**i**].**dx**,** mouseInfo**[**i**].**dy**,** mouseInfo**[**i**].**wheel**);**

**}**

mouseInfoLen **=** 0**;**

**}**

static ssize\_t nodeRead**(**struct file**\*** file**,** char**\*** buf**,** size\_t count**,** loff\_t**\*** ppos**)**

**{**

int res**;**

printk**(**KERN\_CRIT "reading %d bytes (ppos = %d)\n"**,** count**,** **\***ppos**);**

**if(\***ppos **>=** strlen**(**mouseInfoMsg**))**

**{**

**\***ppos **=** 0**;**

printk**(**KERN\_CRIT "EOF"**);**

**return** 0**;**

**}**

**if(**count **>** strlen**(**mouseInfoMsg**)** **-** **\***ppos**)**

count **=** strlen**(**mouseInfoMsg**)** **-** **\***ppos**;**

res **=** copy\_to\_user**((**void**\*)**buf**,** mouseInfoMsg **+** **\***ppos**,** count**);**

**\***ppos **+=** count**;**

printk**(**KERN\_CRIT "returned %d bytes"**,** count**);**

**return** count**;**

**}**

static int nodeClose**(**struct inode**\*** inode**,** struct file**\*** file**)**

**{**

kfree**(**mouseInfoMsg**);**

printk**(**KERN\_CRIT "File closed\n"**);**

module\_put**(**THIS\_MODULE**);**

**return** 0**;**

**}**

### Файл mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTimer>

#include "datataker.h"

#include "canvas.h"

static const char**\*** logFilename **=** "../mouseViewer/loginfo.log"**;**

class MainWindow **:** public QMainWindow

**{**

Q\_OBJECT

public**:**

explicit MainWindow**(**QWidget **\***parent **=** 0**);**

void getInformation**();**

**~**MainWindow**();**

public slots**:**

void receiveData**(**MouseData**);**

void receiveError**();**

private**:**

void writeLog**(**MouseData**);**

private**:**

DataTaker dt**;**

Canvas **\***\_canv**;**

FILE**\*** logging**;**

QTimer **\***m\_tm**;**

**};**

#endif // MAINWINDOW\_H

### Файл mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow**::**MainWindow**(**QWidget **\***parent**)** **:**

QMainWindow**(**parent**)**

**{**

logging **=** fopen**(**logFilename**,** "w"**);**

fclose**(**logging**);**

m\_tm **=** **new** QTimer**(this);**

connect**(**m\_tm**,** SIGNAL**(**timeout**()),** **&**dt**,** SLOT**(**getInfo**()));**

connect**(&**dt**,** SIGNAL**(**sendData**(**MouseData**)),** **this,** SLOT**(**receiveData**(**MouseData**)));**

connect**(&**dt**,** SIGNAL**(**sendError**()),** **this,** SLOT**(**receiveError**()));**

m\_tm**->**start**(**1**);**

\_canv **=** **new** Canvas**(**0**);**

\_canv**->**move**(**20**,** 34**);**

\_canv**->**setWindowTitle**(**tr**(**"Ваша мышка"**));**

\_canv**->**showMaximized**();**

**}**

MainWindow**::~**MainWindow**()**

**{**

**delete** m\_tm**;**

**}**

void MainWindow**::**receiveError**()**

**{**

logging **=** fopen**(**logFilename**,** "w"**);**

fprintf**(**logging**,** "Файл %s не найден.\n"**,** procFilename**);**

fclose**(**logging**);**

**}**

void MainWindow**::**writeLog**(**MouseData data**)**

**{**

logging **=** fopen**(**logFilename**,** "a"**);**

char str**[**100**]** **=** ""**;**

**if(**data**.**left**)** sprintf**(**str**,** "%sMOUSE1"**,** str**);**

**else**

**if(**data**.**right**)** sprintf**(**str**,** "%sMOUSE2"**,** str**);**

**else**

**if(**data**.**middle**)** sprintf**(**str**,** "%sMOUSE3"**,** str**);**

**else** sprintf**(**str**,** "%s%6s"**,** str**,** " "**);**

sprintf**(**str**,** "%s\t"**,** str**);**

**if(**data**.**offset**.**x**()** **!=** 0**)**

sprintf**(**str**,** "%sdx=%-7d"**,** str**,** data**.**offset**.**x**());**

**else**

sprintf**(**str**,** "%s%-10s"**,** str**,** " "**);**

**if(**data**.**offset**.**y**()** **!=** 0**)**

sprintf**(**str**,** "%sdy=%-7d"**,** str**,** data**.**offset**.**y**());**

**else**

sprintf**(**str**,** "%s%-10s"**,** str**,** " "**);**

**if(**data**.**down**)** sprintf**(**str**,** "%sWHEEL\_DOWN"**,** str**);**

**else**

**if(**data**.**up**)** sprintf**(**str**,** "%sWHEEL\_UP"**,** str**);**

**else** sprintf**(**str**,** "%s%10s"**,** str**,** " "**);**

sprintf**(**str**,** "%s\t"**,** str**);**

**if(**strlen**(**str**))**

**{**

QByteArray ba **=** data**.**time**.**toString**().**toUtf8**();**

sprintf**(**str**,** "%s\t%s\n"**,** str**,** ba**.**data**());**

fprintf**(**logging**,** str**);**

**}**

fclose**(**logging**);**

**}**

void MainWindow**::**receiveData**(**MouseData data**)**

**{**

writeLog**(**data**);**

**if(**data**.**left**)** **{**

\_canv**->**setColor**(**QColor**(**200**,** 0**,** 0**,** 100**));**

\_canv**->**addOffset**(**data**.**offset**);**

**}** **else** **if(**data**.**middle**){**

\_canv**->**setColor**(**\_canv**->**getBackgroundColor**());**

\_canv**->**addOffset**(**data**.**offset**);**

**}** **else** **if(!**data**.**right**)** **{**

\_canv**->**setColor**(**QColor**(**70**,** 200**,** 20**,** 200**));**

\_canv**->**addOffset**(**data**.**offset**);**

**}**

**if(**data**.**down**)**

\_canv**->**decWidth**();**

**else** **if(**data**.**up**)**

\_canv**->**incWidth**();**

\_canv**->**repaint**();**

**}**

### Файл canvas.h

#ifndef CANVAS\_H

#define CANVAS\_H

#include <QMainWindow>

#include <QWidget>

#include <QPainter>

#include <QPixmap>

class Canvas **:** public QWidget

**{**

Q\_OBJECT

public**:**

explicit Canvas**(**QWidget **\***parent **=** 0**);**

**~**Canvas**();**

signals**:**

public slots**:**

void addOffset**(**QPoint \_Offset**);**

void setColor**(**QColor \_Color**);**

void incWidth**()** **{** **++**\_widthOffset**;** **}**

void decWidth**()** **{** **--**\_widthOffset**;** **}**

QColor getBackgroundColor**();**

protected**:**

void resizeEvent**(**QResizeEvent **\*);**

void paintEvent**(**QPaintEvent **\*);**

private**:**

QPixmap \_img**;**

QPoint \_pos**;**

QPoint \_prevPos**;**

QColor \_color**;**

int \_widthOffset**;**

int \_maxWidth**,** \_minWidth**;**

**};**

#endif // CANVAS\_H

### Файл canvas.cpp

#include "canvas.h"

Canvas**::**Canvas**(**QWidget **\***parent**)** **:**

QWidget**(**parent**),**

\_pos**(**QCursor**::**pos**()),**

\_prevPos**(**\_pos**),**

\_color**(**QColor**(**170**,** 70**,** 180**,** 200**)),**

\_widthOffset**(**0**),**

\_maxWidth**(**10**),**

\_minWidth**(**1**)**

**{**

\_img **=** QPixmap**(**size**());**

**}**

Canvas**::~**Canvas**()**

**{**

**}**

void Canvas**::**resizeEvent**(**QResizeEvent **\*)**

**{**

\_img **=** QPixmap**(**size**());**

QPainter p**(&**\_img**);**

p**.**setPen**(**QColor**(**90**,** 90**,** 90**));**

p**.**setBrush**(**QColor**(**245**,** 245**,** 245**));**

p**.**drawRect**(**rect**());**

repaint**();**

**}**

void Canvas**::**paintEvent**(**QPaintEvent **\*)** **{**

QPainter p**(this);**

p**.**setRenderHint**(**QPainter**::**Antialiasing**);**

p**.**drawPixmap**(**0**,** 0**,** \_img**);**

p**.**setPen**(**QPen**(**QColor**(**110**,** 110**,** 110**,** 130**),** 1**));**

p**.**setBrush**(**Qt**::**NoBrush**);**

**}**

void Canvas**::**addOffset**(**QPoint offset**)** **{**

\_pos **+=** offset**;**

**if** **(**\_pos**.**x**()** **<** 0**)** \_pos**.**setX**(**0**);**

**if** **(**\_pos**.**y**()** **<** 0**)** \_pos**.**setY**(**0**);**

**if** **(**\_pos**.**x**()** **>** width**())** \_pos**.**setX**(**width**());**

**if** **(**\_pos**.**y**()** **>** height**())** \_pos**.**setY**(**height**());**

QPainter p**(&**\_img**);**

p**.**setRenderHint**(**QPainter**::**Antialiasing**);**

int penWidth **=** 2 **+** \_widthOffset**;**

**if(**penWidth **>** \_maxWidth**)** penWidth **=** \_maxWidth**;**

**if(**penWidth **<** \_minWidth**)** penWidth **=** \_minWidth**;**

p**.**setPen**(**QPen**(**\_color**,** penWidth **));**

p**.**setBrush**(**\_color**);**

p**.**drawLine**(**\_pos**,** \_prevPos**);**

\_prevPos **=** \_pos**;**

**}**

void Canvas**::**setColor**(**QColor color**)** **{**

\_color **=** color**;**

**}**

QColor Canvas**::**getBackgroundColor**()**

**{**

**return** QWidget**::**palette**().**color**(**QWidget**::**backgroundRole**());**

**}**

### Файл datataker.h

#ifndef DATATAKER\_H

#define DATATAKER\_H

#include <QThread>

#include <QString>

#include <QStringList>

#include <QDateTime>

#include <QPoint>

#include <QObject>

#include <stdio.h>

int cleanFile**();**

static const char**\*** procFilename **=** "/proc/mouseListener/info"**;**

struct MouseData

**{**

QDateTime time**;**

bool left**;**

bool right**;**

bool middle**;**

QPoint offset**;**

bool up**;**

bool down**;**

**};**

class DataTaker **:** public QObject

**{**

Q\_OBJECT

public**:**

DataTaker**()** **{}**

public slots**:**

void getInfo**();**

signals**:**

void sendData**(**MouseData**);**

void sendError**();**

**};**

#endif // DATATAKER\_H

### Файл datataker.cpp

#include "datataker.h"

void DataTaker**::**getInfo**()**

**{**

char buf**[**100**];**

FILE**\*** fp **=** fopen**(**procFilename**,** "r"**);**

**if(!**fp**)**

**{**

emit sendError**();**

**return;**

**}**

**while(**fgets**(**buf**,** 100**,** fp**)** **!=** **NULL)**

**{**

QString str**(**buf**);**

QStringList data **=** str**.**split**(**"\t"**);**

**if(**data**.**length**()** **==** 5**)**

**{**

MouseData md**;**

md**.**time**.**setTime\_t**(**data**[**0**].**toUInt**());**

md**.**offset **=** QPoint**(**data**[**2**].**toInt**(),** data**[**3**].**toInt**());**

md**.**left **=** data**[**1**].**toInt**()** **&** 0x01**;**

md**.**right **=** data**[**1**].**toInt**()** **&** 0x02**;**

md**.**middle **=** data**[**1**].**toInt**()** **&** 0x04**;**

md**.**down **=** data**[**4**].**toInt**()** **<** 0**;**

md**.**up **=** data**[**4**].**toInt**()** **>** 0**;**

emit sendData**(**md**);**

**}**

**}**

fclose**(**fp**);**

**}**

# Заключение

Написан рабочий модуль ядра и программа пользовательского уровня, которая получает из него информацию. В процессе написания проекта изучена структура драйверов USB-устройств в Linux, в частности USB-мыши, получены навыки быстро ориентироваться в чужих исходных кодах. Также изучены новые возможности файловой системы /proc в ядре версии 4.4. В процессе работы программы ведётся подробный журнал, удобный как для чтения человеком, так и для парсинга программно, на основании которого могут быть реализованы цели, описанные во введении.

# Список литературы

* 1. URL: <https://github.com/torvalds>
  2. URL: [http://www.stackoverflow.com](http://www.stackoverflow.com/)
  3. <http://lxr.free-electrons.com/source/drivers>
  4. URL: <http://lxr.free-electrons.com/source/drivers>
  5. Разработка модулей ядра Linux: Часть 1. Первые шаги.

<http://www.ibm.com/developerworks/ru/library/l-linux_kernel_01/>

URL: <http://www.ibm.com/developerworks/ru/library/l-linux_kernel_01/>

* 1. <http://doc.crossplatform.ru/qt/4.8.x/html-qt>URL: http://doc.crossplatform.ru/qt/4.8.x/html-qt