Государственное образовательное учреждение высшего профессионального образования

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

ФАКУЛЬТЕТ \_\_\_Информатика и системы управления \_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_Програмное обеспечение ЭВМ и инфомационные технологии\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

\_\_ Программное обеспечение, предоставляющее пользователю возможность работать\_\_\_\_\_\_\_\_

в терминале ОС Linux, используя только мышь\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_**Захаров М.М.\_**\_\_**

(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта \_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_** Тассов К.Л. **\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Москва, 2017

**Содержание**

Введение...............................................................................................................................3

1. Аналитическая часть.................................................................................................4
   1. Анализ задачи.......................................................................................................4
   2. Анализ подходов к реализации...........................................................................4
   3. USB-драйверы в Linux.........................................................................................5
      1. Поддержка USB в ядре Linux........................................................................5
      2. Регистрация и выгрузка драйвера................................................................5
      3. Регистрация устройства................................................................................6
      4. Выводы...........................................................................................................6
   4. Перехват сообщений USB-мыши.......................................................................7
      1. Анализ файла usbmouse.c..............................................................................7
   5. Передача данных в пространство пользователя................................................7
   6. Принцип работы приложения, имитирующего действия клавиатуры............8
2. Конструкторская часть..............................................................................................9
   1. Структура программного обеспечения..............................................................9
   2. Передача действий мыши из драйвера usbmouse в модуль mouseListener.....9
   3. Сопоставление данных, полученных от драйвера, и кнопок мыши.............10
   4. Разработка пользовательского приложения....................................................11
      1. Анализ задач пользовательского приложения..........................................11
      2. Диаграмма классов......................................................................................11
3. Технологическая часть............................................................................................13
   1. Выбор языка программирования и средств разработки.................................13
   2. Сборка проекта...................................................................................................13
   3. Очистка...............................................................................................................13
   4. Запуск проекта...................................................................................................14
   5. Остановка проекта.............................................................................................14
   6. Пример работы программы...............................................................................14
      1. Запуск программы.......................................................................................14
      2. Работа программы.......................................................................................15

Заключение.........................................................................................................................16

Список использованной литературы................................................................................17

Приложение........................................................................................................................18

# Введение

Сегодня клавиатура и мышь – это основные устройства, которые пользователи компьютера используют для взаимодействия с ним. Управление компьютером с помощью мыши привычно большинству пользователей. Мышь используется для запуска программ и открытия папок на рабочем столе и в файловом менеджере, для выделения текста, копирования файлов путем перетаскивания. Мышь является основным средством управления в видеоиграх и других развлекательных приложениях, а также в графических редакторах и в системах автоматизированного проектирования. Однако есть ситуации, в которых мышь практически не нужна. Это те ситуации, в которых управление компьютером осуществляется преимущественно через клавиатуру. Одна из таких ситуаций – работа в командной строке или терминале, в частности, в терминале операционной системы семейства Linux. В Linux, в отличие от операционных систем Windows, терминал – основное средство управления системой. Иногда даже, например, при подключении к удаленному серверу через протокол SSH, управление системой возможно только через терминал. В таких ситуациях без клавиатуры обойтись невозможно. Но от технических проблем не застрахован никто, и что же делать, если есть необходимость работать в терминале Linux, а клавиатуры нет? Цель моего проекта – разработать возможное решение этой проблемы: программное обеспечение, предоставляющее пользователю возможность работать в терминале Linux, используя только мышь.

# Аналитическая часть

* 1. Анализ задачи

В соответствии с техническим заданием на курсовой проект необходимо разработать программное обеспечение, фиксирующее события в системе, инициирующиеся средством ввода информации – мышью, и имитирующее события клавиатуры. Обычная мышь, помимо перемещения курсора, позволяет совершать 5 действий: нажатие левой кнопки, нажатие правой кнопки, прокрутка колесика вниз, прокрутка колесика вверх, нажатие на колесико. При совершении каждого из этих действий мышь формирует событие в системе. Программное обеспечение должно перехватывать эти события, и интерпретировать их, как вызов определенных функций приложения, имитирующего действия клавиатуры.

Для достижения 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.
  1. 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.

* 1. USB-драйверы в Linux
     1. 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.

В ядре Linux файлы программ располагаются в drivers/usb/, а заголовочные файлы – в include/linux/. Информации, представленной в этих директориях, достаточно, чтобы самостоятельно написать драйвер для любого USB-устройства.

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

1. Регистрация/выгрузка драйвера;
2. Регистрация/удаление устройства;
3. Обмен данными: управляющий и информационный.
   * 1. Регистрация и выгрузка драйвера

По USB может передаваться несколько типов пакетов:

Регистрация USB-драйвера подразумевает:

1. Заполнение структуры usb\_driver;
2. Регистрацию структуры в системе

Структура usb\_driver описана в include/linux/usb.h. Рассмотрим наиболее важные поля этой структуры.

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 соответствия подключаемого устройства определенным параметрам. Только те устройства, которые соответствуют перечисленным параметрам, могут быть подключены к драйверу. Если массив пуст, система будет пытаться каждое устройство подключить 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).

Определение структуры usb\_device\_id находится в include/linux/mod\_devicetable.h

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.

* + 1. 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.

* + 1. 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.

* 1. Перехват сообщений USB-мыши

За управление USB-устройствами в Linux отвечает модуль usbhid. Проанализировав файл Linux/drivers/hid/usbhid/ Kconfig в исходном коде Linux, можно сделать вывод, что модуль usbhid обеспечивает подключение различных устройств USB для взаимодействия с человеком: клавиатуры, мыши, джойстики, графические планшеты. При этом этот драйвер не может использоваться одновременно с драйвером мыши, поэтому его придется отключить. Драйвер мыши представлен модулем usbmouse. Соответственно исходный код драйвера мыши можно найти в файле usbmouse.c.

* + 1. Анализ файла usbmouse.c

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

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. Событие мыши функция обрабатывает в виде массива data:

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

signed char \*data = mouse->data;

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.

* 1. Передача данных в пространство пользователя

Для реализации поставленной задачи необходимо передавать данные из пространства ядра в пространство пользователя для дальнейшей обработки. Для передачи данных в пространство пользователя можно воспользоваться файловой системой procfs. Она предоставляет все ресурсы для реализации интерфейса между пространством пользователя и пространством ядра.

* 1. Принцип работы приложения, имитирующего действия клавиатуры

Из-за того, что на клавиатуре кнопок намного больше, чем на мыши, а также принимая во внимание ограниченные графические возможности терминала, необходимо, чтобы программа, получая от пользователя только 5 различных команд, могла выполнять все действия клавиатуры, необходимые для работы в терминале, а также предоставлять пользователю интерфейс, необходимый для использования программы. Было решено реализовать эти задачи следующим образом:

1. Выделить категории действий клавиатуры, необходимых для работы с терминалом:
   * Строчные буквы
   * Прописные буквы
   * Цифры
   * Символы
   * Управляющие клавиши
2. Для переключения между категориями использовать правую кнопку мыши
3. Для переключения между действиями выбранной категории использовать прокрутку колесика мыши вниз и вверх.
4. По нажатию на левую кнопку мыши добавлять действие в буфер.
5. По нажатию на колесико мыши выполнять действия, находящиеся в буфере.
6. После каждого действия выводить выбранную категорию действий, выбранное действие и буфер.

# Конструкторская часть

* 1. Структура программного обеспечения

В соответствии с анализом задачи, в состав программного обеспечения будут входить:

* usbmouse – измененный драйвер USB-мыши, передающий действия мыши в mouseListener
* mouseListener – модуль ядра, сопоставляющий данные, полученные от драйвера, нажатым кнопкам мыши и передающий информацию о нажатых кнопках мыши в procfs
* kb – программа пространства пользователя, получающая из procfs информацию о нажатых кнопках мыши и вызывающая соответствующие им функции для имитации действий клавиатуры.

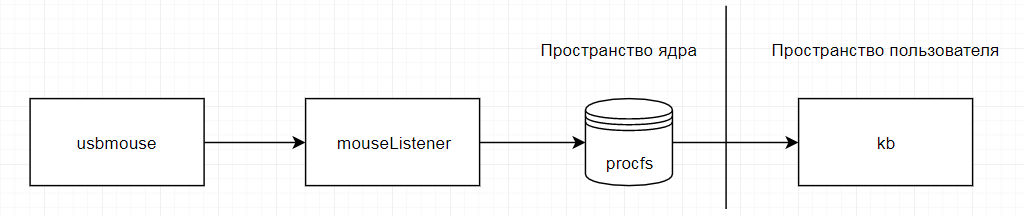


Рисунок 2.1. – Структура программного обеспечения

* 1. Передача действий мыши из драйвера usbmouse в модуль mouseListener

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(signed char \*data);

* 1. Сопоставление данных, полученных от драйвера, и кнопок мыши

Для представления кнопок мыши создан тип данных:

|  |
| --- |
| typedef enum mouseButton |
| { |
| NONE, |
| LEFT, |
| RIGHT, |
| MIDDLE, |
| WHEELUP, |
| WHEELDOWN |
| } MouseButton; |

Функция, преобразующая данные, полученные от драйвера в тип MouseButon:

|  |
| --- |
| static MouseButton dataToButton(signed char \*data) |
| { |
| if (data[0] & 0x01) |
| { |
| return LEFT; |
| } |
| if (data[0] & 0x02) |
| { |
| return RIGHT; |
| } |
| if (data[0] & 0x04) |
| { |
| return MIDDLE; |
| } |
| if (data[6] == 1) |
| { |
| return WHEELUP; |
| } |
| if (data[6] == -1) |
| { |
| return WHEELDOWN; |
| } |
| return NONE; |
| } |

* 1. Разработка пользовательского приложения
     1. Анализ задач пользовательского приложения

Пользовательское приложение должно выполнять следующие функции:

1. Читать данные из procfs
2. Выполнять в соответствии с полученными данными действия, описанные в аналитическом разделе
3. Имитировать действия клавиатуры

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

* + 1. Диаграмма классов

Для реализации поставленных задач необходимо разработать следующие классы:

* Model – управляющий, отвечает за логику работы пользовательской программы
* Device – выполняет имитацию нажатия клавиши
* Action – действие клавиатуры
* Click – нажатие клавиши
* Combo – одновременное нажатие нескольких клавиш
* Shift – комбинация нажатия клавиши с нажатием клавиши Shift
* Control – комбинация нажатия клавиши с нажатием клавиши Control
* ActionSet – категория действий клавиатуры
* Letters – строчные буквы
* CapitalLetters – прописные буквы
* Digits – цифры
* Symbols – символы
* Controls – управляющие действия
* ActionSetCycle – циклический список категорий действий клавиатуры
* ActionBuffer – буфер действий клавиатуры

Диаграмма классов представлена на рисунке 2.2.

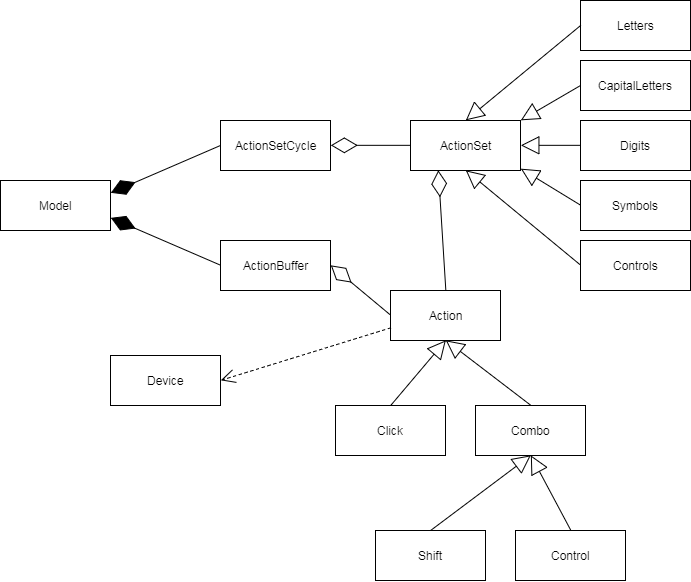


Рисунок 2.2. – Диаграмма классов

# Технологическая часть

* 1. Выбор языка программирования и средств разработки

Для разработки пользовательского приложения выбран язык программирования Python. Синтаксис языка Python очень прост, и многие небольшие задачи, решение которых в других языках требовало бы написание собственных функций, реализованы в стандартных библиотеках Python. Для имитации действий клавиатуры используется библиотеку uinput-python, представляющая собой интерфейс к библиотеке uinput языка C.

В качестве среды разработки пользовательского приложения был выбран JetBrains PyCharm. Как и другие среды разработки от JetBrains, PyCharm предоставляет мощные средства для рефакторинга и динамического анализа кода.

Для сборки модулей ядра используется компилятор gcc.

* 1. Сборка проекта

Для сборки проекта нужно в корневой директории проекта запустить скрипт make.sh:

|  |
| --- |
| cd mouseListener |
| make |
| cd ../usbmouse |
| make |

* 1. Очистка

Для удаления файлов, созданных в результате сборки нужно в корневой директории проекта запустить скрипт clean.sh:

|  |
| --- |
| cd mouseListener |
| make clean |
| cd ../usbmouse |
| make clean |

* 1. Запуск проекта

Для запуска проекта написан скрипт start.sh, который останавливает драйвер usbhid, загружает драйвер usbmouse и модуль ядра mouseListener, и запускает пользовательское приложение:

|  |
| --- |
| sudo rmmod usbhid |
| sudo insmod mouseListener/mouseListener.ko |
| sudo insmod usbmouse/usbmouse.ko |
| sudo `which python` kb/main.py & |

* 1. Остановка проекта

Для остановки проекта написан скрипт stop.sh, который останавливает пользовательское приложение, драйвер usbmouse и модуль ядра mouseListener, и загружает драйвер usbhid:

|  |
| --- |
| sudo kill `pgrep "$sudo.\*python"` |
| sudo rmmod usbmouse |
| sudo rmmod mouseListener |
| sudo modprobe usbhid |

* 1. Пример работы программы
     1. Запуск программы

После выполнения скрипта start.sh программа запускается, и выводит пользователю текущее состояние: выбранное действие, буфер и выбранную категорию. Окно терминала, в котором выполнен запуск программы, представлено на рисунке 3.1.

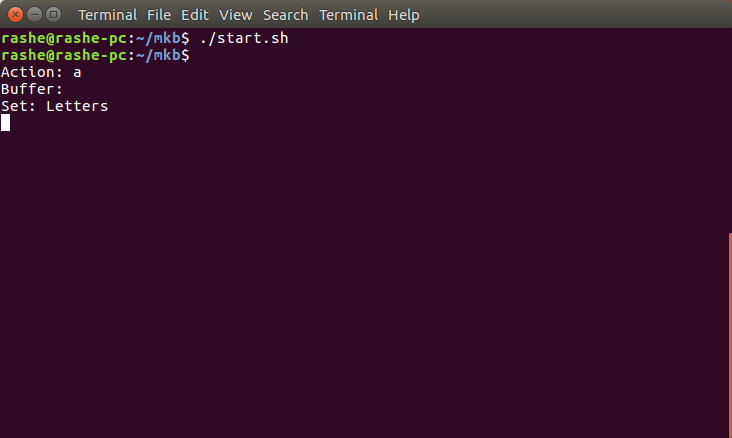


Рисунок 3.1. Запуск программы

* + 1. Работа программы

Окно терминала, в котором с помощью программы набрана и выполнена команда

“ls -la” представлено на рисунке 3.2.

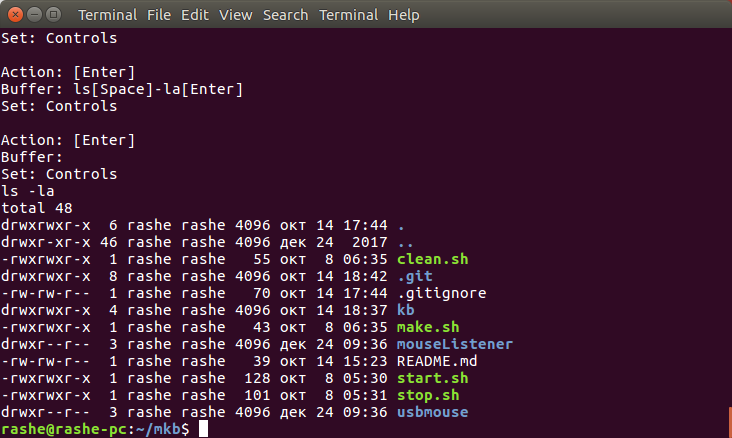


Рисунок 3.1. Выполнение команды

# Заключение

Разработанное программное обеспечение реализует все функции, необходимые для работы в терминале Linux с использованием только мыши. Сборка, запуск и остановка программного обеспечения вынесены в отдельны скрипты для удобства пользователя. Интерфейс, несмотря на отсутствие графики и ограничения командной строки, обеспечивает удобное взаимодействие с программой. Таким образом, разработанное программное обеспечение удовлетворяет всем требованиям технического задания.

# Список использованной литературы

1. Вахалия Ю. UNIX изнутри – Санкт-Петербург: Питер, 2003
2. Джонс, М. Анатомия загружаемых модулей ядра Linux / М. Джонс – <https://www.ibm.com/developerworks/ru/library/l-lkm/index.html>, 2008
3. Лав, Роберт, Ядро Linux: описание процесса разработки, 3-е изд.: пер. с англ. – М.: ООО «И. Д. Вильямс», 2015
4. Free electronics [Электронный ресурс] – URL:

<https://elexir.free-electrons.com/linux/latest/source/>

1. Kernel,org [Электронный ресурс] – URL: https://www.kernel.org/doc/Documentation

# Приложение

Исходный код разработанного модуля ядра

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" |
| #define MAX\_ID 99 |
| #define MESSAGE\_SIZE 14 |
|  |
| MODULE\_LICENSE("GPL"); |
| MODULE\_AUTHOR("Mikhail Zakharov"); |
|  |
| static struct proc\_dir\_entry \*procDir; |
|  |
| static int \_\_init moduleInit(void); |
| static void \_\_exit moduleExit(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); |
|  |
| static const struct file\_operations nodeFops = |
| { |
| .owner = THIS\_MODULE, |
| .open = nodeOpen, |
| .read = nodeRead, |
| .release = nodeClose |
| }; |
|  |
| typedef enum mouseButton |
| { |
| NONE, |
| LEFT, |
| RIGHT, |
| MIDDLE, |
| WHEELUP, |
| WHEELDOWN |
| } MouseButton; |
|  |
| MouseButton button; |
| int id = 0; |
| char \*mouseInfoMsg; |
|  |
| module\_init(moduleInit); |
| module\_exit(moduleExit); |

mouseListener.c

|  |
| --- |
| #include "mouseListener.h" |
|  |
| static int \_\_init moduleInit(void) |
| { |
| if ((procDir = proc\_mkdir\_mode(DIR\_NAME, S\_IFDIR | S\_IRWXUGO, NULL)) == NULL) |
| { |
| printk(KERN\_CRIT "Can't create dir /proc/%s", DIR\_NAME); |
| return -ENOENT; |
| } |
| if (proc\_create\_data(NODE\_NAME, S\_IFREG | S\_IRUGO | S\_IWUGO, procDir, &nodeFops, NULL) == NULL) |
| { |
| printk(KERN\_CRIT "Can't create node /proc/%s/%s", DIR\_NAME, NODE\_NAME); |
| remove\_proc\_entry(DIR\_NAME, NULL); |
| return -ENOMEM; |
| } |
| printk(KERN\_CRIT "Node /proc/%s/%s installed", DIR\_NAME, NODE\_NAME); |
| mouseInfoMsg = (char\*) kmalloc(MESSAGE\_SIZE, GFP\_KERNEL); |
| return 0; |
| } |
|  |
| static void \_\_exit moduleExit(void) |
| { |
| kfree(mouseInfoMsg); |
| remove\_proc\_entry(NODE\_NAME, procDir); |
| remove\_proc\_entry(DIR\_NAME, NULL); |
| } |
|  |
| static MouseButton dataToButton(signed char \*data) |
| { |
| if (data[0] & 0x01) |
| { |
| return LEFT; |
| } |
| if (data[0] & 0x02) |
| { |
| return RIGHT; |
| } |
| if (data[0] & 0x04) |
| { |
| return MIDDLE; |
| } |
| if (data[6] == 1) |
| { |
| return WHEELUP; |
| } |
| if (data[6] == -1) |
| { |
| return WHEELDOWN; |
| } |
| return NONE; |
| } |
|  |
| static char \*buttonToString(void) |
| { |
| switch (button) |
| { |
| case LEFT: |
| return "LEFT"; |
| case RIGHT: |
| return "RIGHT"; |
| case MIDDLE: |
| return "MIDDLE"; |
| case WHEELUP: |
| return "WHEELUP"; |
| case WHEELDOWN: |
| return "WHEELDOWN"; |
| default: |
| return "NONE"; |
| } |
| } |
|  |
| extern bool mouseListenerSendCoordinates(signed char \*data) |
| { |
| MouseButton mb = dataToButton(data); |
| if (mb != NONE) |
| { |
| button = mb; |
| if (++id > MAX\_ID) { |
| id = 0; |
| } |
| } |
| return 1; |
| } |
| EXPORT\_SYMBOL(mouseListenerSendCoordinates); |
|  |
| static int nodeOpen(struct inode\* inode, struct file\* file) |
| { |
| try\_module\_get(THIS\_MODULE); |
| sprintf(mouseInfoMsg, "%d\n%s\n", id, buttonToString()); |
| return 0; |
| } |
|  |
| static ssize\_t nodeRead(struct file\* file, char\* buf, size\_t count, loff\_t\* ppos) |
| { |
| if(\*ppos >= strlen(mouseInfoMsg)) |
| { |
| \*ppos = 0; |
| return 0; |
| } |
| if(count > strlen(mouseInfoMsg) - \*ppos) |
| { |
| count = strlen(mouseInfoMsg) - \*ppos; |
| } |
| copy\_to\_user((void\*) buf, mouseInfoMsg + \*ppos, count); |
| \*ppos += count; |
| return count; |
| } |
|  |
| static int nodeClose(struct inode\* inode, struct file\* file) |
| { |
| module\_put(THIS\_MODULE); |
| return 0; |
| } |

mouseListenerExtern.h

extern bool mouseListenerSendCoordinates(signed char \*data);