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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка загружаемого модуля ядра для изменения скорости перемещения курсора с помощью колесика мыши»

Студент ИУ7-74Б _____ А. В. Золотухин
(Группа) (Подпись, дата) (И.О.Фамилия)

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

2023 г.

РЕФЕРАТ

В работе представлена реализация программного обеспечения для изменения скорости перемещения курсора с помощью колесика USB-мыши. Ключевые слова: чувствительность, USB, драйвер, Linux.

Рассмотрены требования к реализации драйвера USB-устройств. Выбраны способы обработки прерываний от мыши. Приведены листинги кода. Реализован требуемый драйвер. Представлена демонстрация работы программы.

Расчетно-пояснительная записка к курсовой работе содержит 35 страницы, 3 иллюстраций, 11 источников.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Чувствительность мыши — это важная характеристика мыши. Она отвечает за то, насколько быстро перемещается указатель мыши при перемещении мыши. При более высокой чувствительности указатель движется быстрее и проходит большее расстояние на экране, чем пользователь физически перемещает мышь. При более низкой чувствительности указатель движется медленнее и требует больше усилий для перемещения по экрану, но обеспечивает лучшую точность.

Мышь – координатное устройство для управления курсором и отдачи различных команд компьютеру. Оно широко распространено среди пользователей компьютеров, поэтому для более гибкой настройки чувствительности имеет смысл добавить возможность ее с помощью колесика.

Тема курсовой работы – разработать программное обеспечение, позволяющее изменять чувствительность USB-мыши с помощью колесика.

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с заданием на курсовую работу по дисциплине «Операционные системы» требуется разработать программное обеспечение, позволяющее изменять чувствительность USB-мыши с помощью колесика.

Для выполнения задания требуется решить следующие задачи:

1. Провести анализ существующих подходов к изменению функциональности внешних устройств в Linux.
2. Разработать алгоритмы, необходимые для реализации программное обеспечение.
3. Разработать ПО, предоставляющая требуемую функциональность.
4. Провести исследование разработанного программное обеспечение.

При двойном нажатии на колесико мыши долена включиться настройка чувствительности и выключиться старая функциональность колесика. При прокрутке колесика мыши вниз чувствительность мыши должна понижаться, при прокрутке вверх — увеличиваться. Для отключения настройки и восстановления старой функциональности нужно один раз нажать на колесико мыши. Функциональность мыши должна сохраниться: с её помощью можно перемещать курсор и нажимать правую и левую кнопки мыши.

Для разработки и тестирования данной работы используется мышь Logitech B100 [1] и операционная система Ubuntu-22.04 [2].

1.2 Анализ драйверов устройства

Драйверы устройств являются одной из разновидностей модулей ядра. Драйверы полностью скрывают детали, касающиеся работы устройства и предоставляют четкий программный интерфейс для работы с аппаратурой. В Unix каждое аппаратное устройство представлено файлом устройства в каталоге /dev.

1.2.1 Анализ возможных типов драйвера

В Unix/Linux драйверы бывают трех типов:

— встроенные – выполнение этих драйверов инициализируется при запуске системы;

— драйверы, код которых поделен между ядром и специальной утилиты;

— драйверы, реализованные как загружаемые модули ядра.

Среди последних выделяют HID-драйверы. Класс HID является одним из наиболее часто используемых классов USB. Класс HID состоит в основном из устройств, предназначенных для интерактивного взаимодействия с компьютером.

Для изменения функциональности мыши требуется разработать именно HID-драйвер.

1.2.2 Анализ алгоритма регистрации USB-драйвера в системе

Для выполнения задания требуется разработать драйвер мыши. Регистрация USB-драйвера подразумевает [3]:

1. Заполнение структуры *usb_driver*.
2. Регистрация структуры в системе.

   Сначала требуется инициализировать поля структуры *usb_driver*.
   Структура usb_driver состоит из следующих полей [4]:

— **name** – имя драйвера, должно быть уникальным среди USB-драйверов.

— **id_table** – массив структур *usb_device_id*, который содержит список всех типов USB-устройств, которые обслуживает драйвер.

— **probe** – функция обратного вызова, является точкой входа драйвера. Она будет вызвана только для тех устройств, которые соответствуют параметрам, перечисленным в структуре *usb_device_id*.

— **disconnect** – функция обратного вызова, которая вызывается при отключении устройства от драйвера.

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.

1.3 Анализ подсистемы ввода/вывода USB

Подсистема ввода/вывода выполняет запросы файловой подсистемы и подсистемы управления процессами для доступа к периферийным устройствам (дискам, магнитным лентам, терминалам и т.д.). Она обеспечивает необходимую буферизацию данных и взаимодействует с драйверами устройств — специальными модулями ядра, непосредственно обслуживающими внешние устройства.

Для использования подсистемы ввода/вывода требуется инициализировать структуру *input_dev* [5]. Поле evbit этой структуры отвечает за то, какие события могут происходить на устройстве. Для мыши возможны два вида событий: *EV_KEY* [6] – нажатия кнопок мыши, и *EV_REL* – изменения относительного положения курсора на экране.

Для вызова событий, связанных с клавишами используется системный вызов *input_report_key* [7], который принимает устройство ввода, кнопку, на которую вызывается событие, и дополнительная информация о событии.

1.4 URB

Сообщение, передаваемое от драйвера USB-устройства системе, называется USB Request Block или URB [8]. Оно описывается структурой *struct urb*. URB используется для передачи или приёма информации от конечной точки на заданное USB-устройство в асинхронном режиме [9]. Каждая конечная

точка может обрабатывать очередь из URB, следовательно, на одну конечную точку может быть выслать множество URB. URB создаются динамически и содержат внутренний счётчик ссылок, что позволяет автоматически освобождать память, когда блок запроса больше никем не используется [8].

Существуют четыре типа URB.

1. control — используются для конфигурирования устройства во время подключения, для управления устройством и получения статусной информации в процессе работы.

2. bulk — применяются при необходимости обеспечения гарантированной доставки данных от хоста к функции или от функции к хосту, но время доставки не ограничено. Приоритет у таких передач самый низкий, они могут приостанавливаться при большой загрузке шины.

3. interrupt — используются в том случае, когда требуется передавать одиночные пакеты данных небольшого размера. Каждый пакет должен быть передан за определенное время. Операции передачи выполняются асинхронно и должны обслуживаться не медленнее, чем того требует устройство.

4. isochronous — применяются для обмена данными в «реальном времени», когда на каждом временном интервале требуется передавать строго определенное количество данных, но доставка информации не гарантирована. Изохронные URB обычно используются в мультимедийных устройствах для передачи аудио- и видеоданных.

1.5 Анализ способов изменения функциональности внешних устройств

Для изменения функциональности внешних устройств существует два основных подхода [12].

1. Report. Report descriptor определяет структуру report, содержащий всю информацию необходимую USB-хосту для определения формата данных и действий. HID report содержит фактические значения данных без

какой-либо дополнительной метаинформации. HID report могут отправляться с устройства («Input report», т.е. события ввода), на устройство («Output report», например, для изменения светодиодов) или использоваться для настройки устройства («Feature report»).

2. URB используется для передачи или приёма данных в или из заданной оконечной точки USB на заданное USB устройство в асинхронном режиме. В зависимости от потребностей, драйвер USB устройства может выделить для одной оконечной точке много URB или может повторно использовать один URB для множества разных оконечных точек. Каждая оконечная точка в устройстве может обрабатывать очередь URB, так что перед тем, как очередь опустеет, к одной оконечной точке может быть отправлено множество URB.

1.6 Анализ USB

1.6.1 Инициализация структуры usb_driver

Для создания USB-драйвера создается экземпляр структуры *usb_driver*. Создание экземпляра приведено на листинге 1.1.

Листинг 1.1 – Структура usb_driver

```
1  static struct usb_driver usb_mouse_driver = {
2      .name       = "accelerator",
3      .probe      = usb_mouse_probe,
4      .disconnect = usb_mouse_disconnect,
5      .id_table   = usb_mouse_id_table,
6  };
```

Для регистрации USB-драйвера в системе используется системный вызов *usb_register*.

1.6.2 Структура для хранения информации о мыши

Для передачи данных, связанных с мышью, была создана структура usb_mouse, приведенная на листинге 1.2.

Листинг 1.2 – Структура usb_mouse

```
1     struct usb_mouse {
2         char name[128];
3         char phys[64];
4         struct usb_device *usbdev;
5         struct input_dev *dev;
6         struct urb *irq;
7
8         signed char *data;
9         dma_addr_t data_dma;
10    };
```

— *data* — данные, передаваемые мышью при прерывании;

— *input_dev* — структура для использования подсистемы входа/выхода;

— *usb_device* — представление usb-устройства;

— *irq* — указатель на обработчик прерываний.

1.6.3 События мыши

Для реализации драйвера необходимо перехватывать следующие события мыши:

— нажатие правой кнопки мыши;

— нажатие левой кнопки мыши;

— нажатие колесика мыши;

— вращение колесика мыши;

— перемещение мыши.

Выводы

В результате проведенного анализа было решено:

— для выполнения задания разработать HID-драйвер, который должен быть реализован как загружаемый модуль ядра;

— для изменения функциональности USB-мыши использовать URB interrupt;

— для изменения чувствительности мыши будет использован коэффициент, на который будут умножаться координаты перемещения мыши. Каждый раз когда пользователь будет прокручивать колесико мыши вверх, коэффициент будет увеличиваться, тем самым будет увеличиваться расстояние, которое будет проходить курсор на экране.

2 КОНСТРУКТОРСКАЯ ЧАСТЬ

2.1 Диаграмма IDEF0

На рисунке 2.1 изображена диаграмма IDEF0 для требуемой задачи:

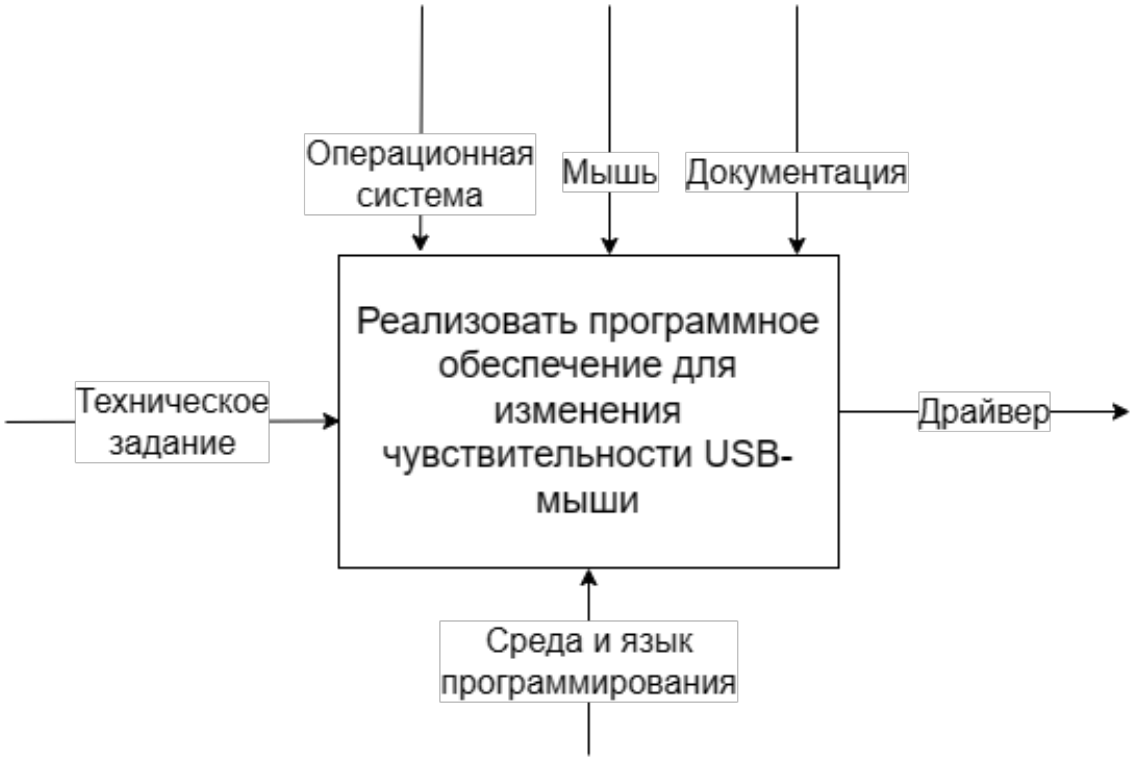

Рисунок 2.1 – Диаграмма IDEF0

2.2 Алгоритм перехвата событий мыши

На рисунке 2.2 изображена схема алгоритма перехвата событий мыши.

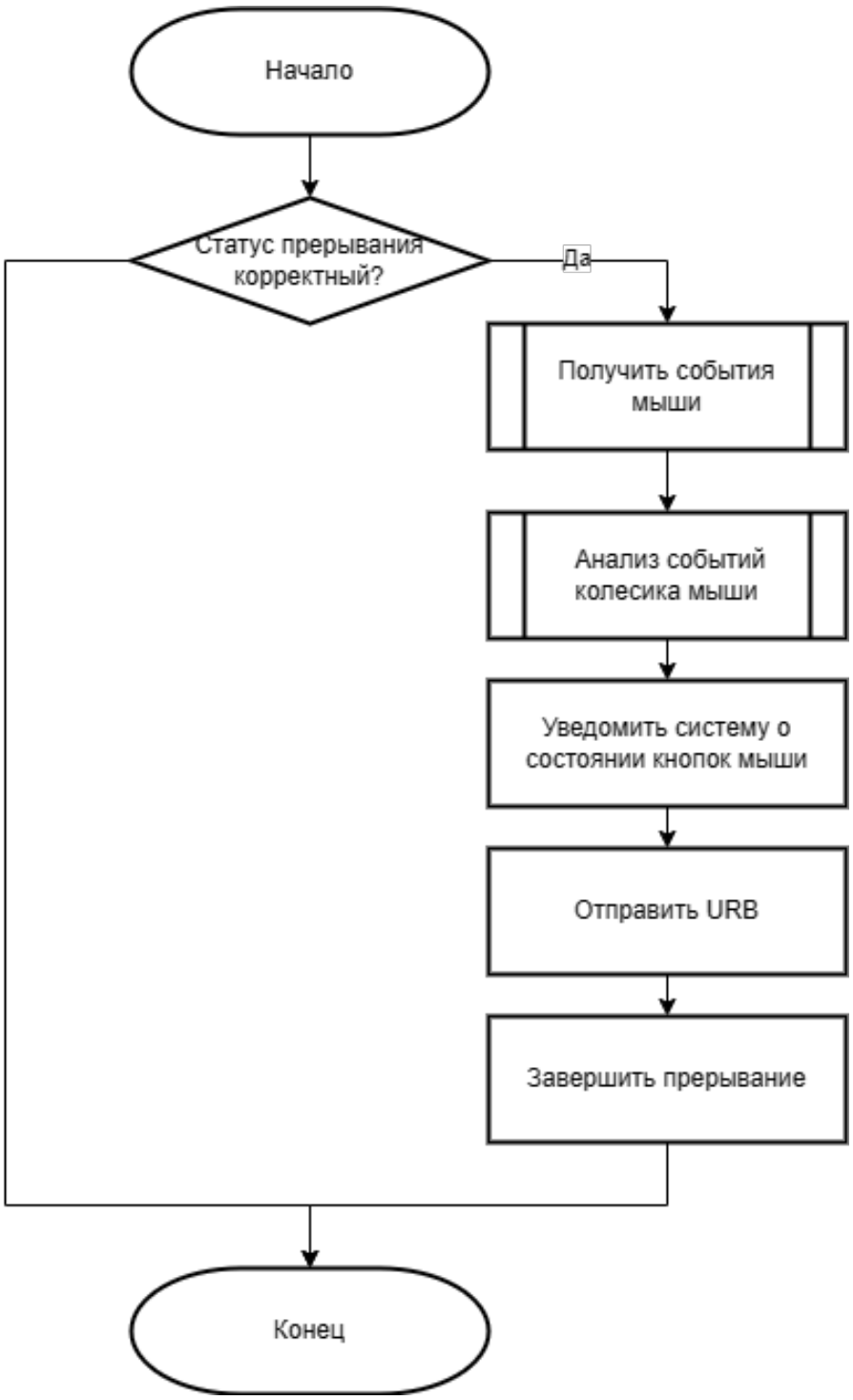

Рисунок 2.2 – Схема алгоритма перехвата событий мыши

2.3 Алгоритм анализа событий колесика мыши

На рисунке 2.3 изображена схема алгоритма анализа событий колесика мыши.

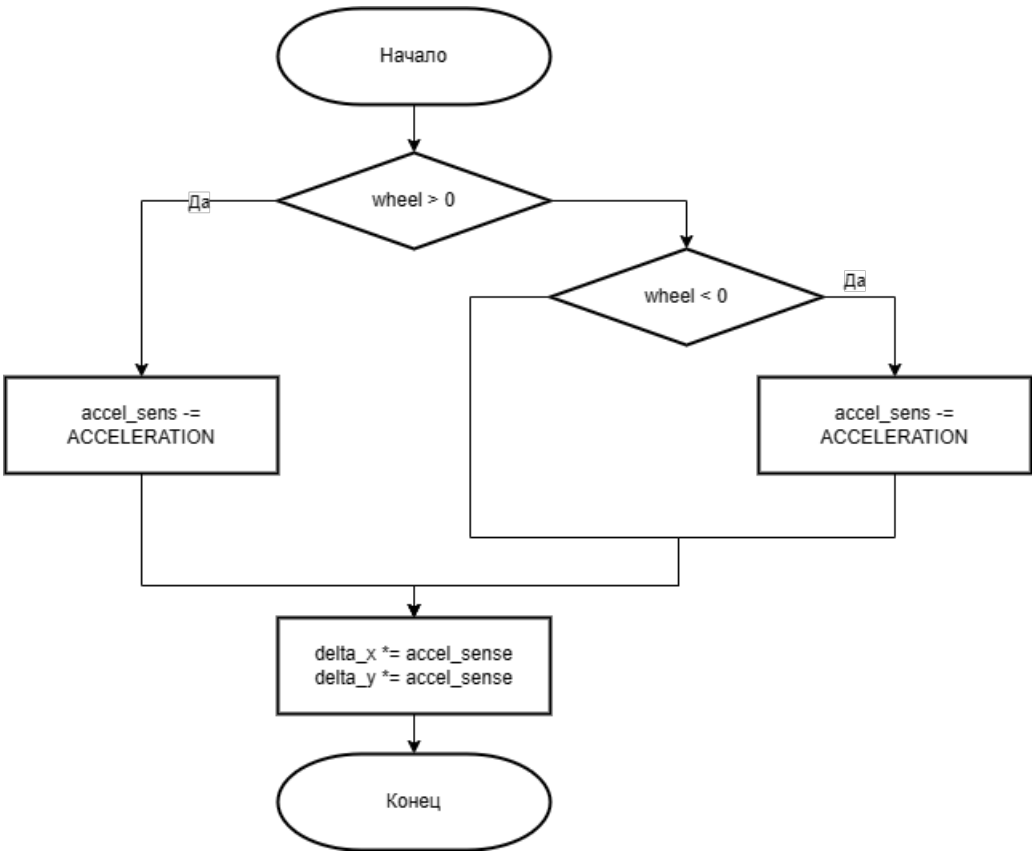

Рисунок 2.3 – Схема алгоритма анализа событий колесика мыши

3 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

3.1 Выбор средств реализации

В качестве языка программирования был выбран язык C. На этом языке реализованы все модули ядра и драйверы операционной системы Linux.

3.2 Реализация изменения чувствительности мыши

В листинге 3.1 представлена реализация алгоритма перехвата событий мыши.

Листинг 3.1 – Реализация алгоритма перехвата событий мыши

```
1  static void usb_mouse_irq(struct urb *urb)
2  {
3      struct usb_mouse *mouse = urb->context;
4      signed char *data = mouse->data;
5      struct input_dev *dev = mouse->dev;
6      int status;
7
8      float delta_x = data[1] * PRE_SCALE_X;
9      float delta_y = data[2] * PRE_SCALE_Y;
10
11     static float carry_x = 0.0f;
12     static float carry_y = 0.0f;
13
14     analyze_wheel(&delta_x, &delta_y, data[3]);
15
16     delta_x *= POST_SCALE_X;
17     delta_y *= POST_SCALE_Y;
18     delta_x += carry_x;
19     delta_y += carry_y;
20     carry_x = delta_x - my_round(delta_x);
21     carry_y = delta_y - my_round(delta_y);
22
23     switch (urb->status) {
24         case 0:
25         break;
26         case -ECONNRESET:
```

```
27          case -ENOENT:
28          case -ESHUTDOWN:
29          return;
30          default:
31          goto resubmit;
32      }
33
34      input_report_key(dev, BTN_LEFT,   data[0] & 0x01);
35      input_report_key(dev, BTN_RIGHT,  data[0] & 0x02);
36      input_report_key(dev, BTN_MIDDLE, data[0] & 0x04);
37      input_report_key(dev, BTN_SIDE,   data[0] & 0x08);
38      input_report_key(dev, BTN_EXTRA,  data[0] & 0x10);
39
40      input_report_rel(dev, REL_X,     Leet_round(delta_x));
41      input_report_rel(dev, REL_Y,     Leet_round(delta_y));
42      input_report_rel(dev, REL_WHEEL, data[3]);
43
44      input_sync(dev);
45      resubmit:
46      status = usb_submit_urb (urb, GFP_ATOMIC);
47      if (status)
48      dev_err(&mouse->usbdev->dev,
49      "can't resubmit intr, %s-%s/input0, status %d\n",
50      mouse->usbdev->bus->bus_name,
51      mouse->usbdev->devpath, status);
52  }
```

В листинге 3.2 представлена реализация алгоритма анализа событий колесика мыши.

Листинг 3.2 – Реализация анализа событий колесика мыши

```
1       static void analyze_wheel(float *delta_x, float *delta_y, int
            wheel)
2       {
3
4           if(wheel > 0)
5           {
6               printk("ACCELERATOR: Increase mouse speed. accel_sens
                    = %d", (int)(accel_sens*10));
7               accel_sens += ACCELERATION;
8               if (SENS_CAP > 0 && accel_sens >= SENS_CAP) {
9                   accel_sens = SENS_CAP;
```

```
10              }
11
12          }
13          else if(wheel < 0)
14          {
15              printk("ACCELERATOR: Decrease mouse speed. accel_sens
                    = %d", (int)(accel_sens*10));
16              accel_sens -= ACCELERATION;
17              if (accel_sens < 0.1f) {
18                  accel_sens = 0.1f;
19              }
20
21          }
22
23          accel_sens /= SENSITIVITY;
24          *delta_x *= accel_sens;
25          *delta_y *= accel_sens;
26      }
```

Для определения нажатой клавиши используется поле *data* из структуры usb_mouse. Биты байта *data[0]* отвечают за тип нажатой клавиши. Байты *data[1]* и *data[2]* отвечают за перемещение мыши горизонтали и вертикали соответственно. Байт *data[3*] отвечает за вращение колесика мыши, при вращении вверх он равен 1, при вращении вниз — -1, когда колесико не вращают — нулю.

3.3 Реализация функции регистрации драйвера

В приложении А представлена реализация функции probe.

В листинге 3.3 представлена реализация функции disconnect.

Листинг 3.3 – Реализация функции disconnect

```
1 static void usb_mouse_disconnect(struct usb_interface *intf)
2 {
3     struct usb_mouse *mouse = usb_get_intfdata (intf);
4
5     usb_set_intfdata(intf, NULL);
6     if (mouse) {
7         usb_kill_urb(mouse->irq);
```

```
8           input_unregister_device(mouse->dev);
9           usb_free_urb(mouse->irq);
10          usb_free_coherent(interface_to_usbdev(intf), 8,
                mouse->data, mouse->data_dma);
11          kfree(mouse);
12      }
13  }
```

В листинге 3.4 представлена реализация функции open.

Листинг 3.4 – Реализация функции open

```
1  static int usb_mouse_open(struct input_dev *dev)
2  {
3      struct usb_mouse *mouse = input_get_drvdata(dev);
4
5      mouse->irq->dev = mouse->usbdev;
6      if (usb_submit_urb(mouse->irq, GFP_KERNEL))
7          return -EIO;
8
9      return 0;
10 }
```

В листинге 3.5 представлена реализация функции close.

Листинг 3.5 – Реализация функции close

```
1  static void usb_mouse_close(struct input_dev *dev)
2  {
3      struct usb_mouse *mouse = input_get_drvdata(dev);
4
5      usb_kill_urb(mouse->irq);
6  }
```

3.4 Реализация Makefile

В листинге 3.6 представлена реализация Makefile.

Листинг 3.6 – Реализация Makefile

```
1  obj-m += accelerator.o
2  ccflags-y += -msse -mpreferred-stack-boundary=4
3
```

```
4 all:
5     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
6
7 clean:
8     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

4 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

4.1 Демонстрация работы программы

На рисунке 4.1 представлены логи при вращении колесика мыши.

Рисунок 4.1 – Логи при вращении колесика мыши

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было разработано ПО, позволяющее изменять чувствительность мыши с помощью колесика. Были выполнены следующие задачи:

— проведен анализ существующих подходов к изменению функциональности внешних устройств в Linux;

— проведен анализ существующих способов изменения скорости перемещения курсора;

— разработаны алгоритмы, необходимые для реализации ПО;

— разработано ПО, предоставляющая требуемую функциональность;

— проведен исследование разработанного ПО.

Было показано, что ПО отвечает поставленной задаче.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Logitech M190 Wireless Mouse [Электронный ресурс]. Режим доступа: https://www.logitech.com/en-us/products/mice/m190-wireless-mouse.910-005901.html (дата обращения: 10.12.2023).

2. Ubuntu 20.04.3 LTS (Focal Fossa) [Электронный ресурс]. Режим доступа: https://releases.ubuntu.com/20.04/ (дата обращения: 10.12.2023).

3. Writing USB Device Drivers. [Электронный ресурс]. Режим доступа: https://kernel.readthedocs.io/en/sphinx-samples/writing_usb_driver.html (дата обращения: 10.12.2023).

4. struct usb_driver. [Электронный ресурс]. Режим доступа: https://elixir.bootlin.com/linux/latest/source/include/linux/usb.h#L1188l (дата обращения: 10.12.2023).

5. struct input_dev. [Электронный ресурс]. Режим доступа: https://elixir.bootlin.com/linux/v5.10.1/source/include/linux/input.h#L131 (дата обращения: 10.12.2023).

6. EV_KEY. [Электронный ресурс]. Режим доступа: https://elixir.bootlin.com/linux/latest/source/include/uapi/linux/input-event-codes.h#L39 (дата обращения: 10.12.2023).

7. input_report_key. [Электронный ресурс]. Режим доступа: https://elixir.bootlin.com/linux/latest/source/include/linux/input.h#L415 (дата обращения: 10.12.2023).

8. Jonathan Corbet Alessandro Rubini Greg Kroah-Hartman. Linux Device Drivers. — 3 edition. – O'Reilly Media, 2005.

9. urb. [Электронный ресурс]. Режим доступа: https://docs.kernel.org/driver-api/usb/URB.html (дата обращения: 10.12.2023).

10. usb_fill_int_urb. [Электронный ресурс]. Режим доступа: https://manpages.debian.org/jessie-backports/linux-manual-4.8/usb_fill_int_urb.9.en.html (дата обращения: 10.12.2023).

11. submit. [Электронный ресурс]. Режим доступа: https://elixir.bootlin.com/linux/latest/source/include/linux/usb.h#L1723 (дата обращения: 10.12.2023).

12. Рязанова Н.Ю., Сикорский О.С. Метод изменения поведения HID-устройств под управлением ОС Linux // Новые информационные технологии в автоматизированных системах. 2018. №21. — С. 354–362.

ПРИЛОЖЕНИЕ А

В листинге 4.1 представлена реализация функции probe.

Листинг 4.1 – Реализация функции probe

```
1  static int usb_mouse_probe(struct usb_interface *intf, const
       struct usb_device_id *id)
2  {
3      struct usb_device *dev = interface_to_usbdev(intf);
4      struct usb_host_interface *interface;
5      struct usb_endpoint_descriptor *endpoint;
6      struct usb_mouse *mouse;
7      struct input_dev *input_dev;
8      int pipe, maxp;
9      int error = -ENOMEM;
10     interface = intf->cur_altsetting;
11     if (interface->desc.bNumEndpoints != 1)
12         return -ENODEV;
13     endpoint = &interface->endpoint[0].desc;
14     if (!usb_endpoint_is_int_in(endpoint))
15         return -ENODEV;
16     pipe = usb_rcvintpipe(dev, endpoint->bEndpointAddress);
17     maxp = usb_maxpacket(dev, pipe);
18     mouse = kzalloc(sizeof(struct usb_mouse), GFP_KERNEL);
19     input_dev = input_allocate_device();
20     if (!mouse || !input_dev)
21         goto fail1;
22     mouse->data = usb_alloc_coherent(dev, 8, GFP_ATOMIC,
         &mouse->data_dma);
23     if (!mouse->data)
24         goto fail1;
25     mouse->irq = usb_alloc_urb(0, GFP_KERNEL);
26     if (!mouse->irq)
27         goto fail2;
28     mouse->usbdev = dev;
29     mouse->dev = input_dev;
30     if (dev->manufacturer)
31         strlcpy(mouse->name, dev->manufacturer,
             sizeof(mouse->name));
32     if (dev->product) {
33         if (dev->manufacturer)
34             strlcat(mouse->name, " ", sizeof(mouse->name));
```

```
35          strlcat(mouse->name, dev->product, sizeof(mouse->name));
36      }
37      if (!strlen(mouse->name))
38          snprintf(mouse->name, sizeof(mouse->name),
39              "USB␣HIDBP␣Mouse␣%04x:%04x",
40              le16_to_cpu(dev->descriptor.idVendor),
41              le16_to_cpu(dev->descriptor.idProduct));
42      usb_make_path(dev, mouse->phys, sizeof(mouse->phys));
43      strlcat(mouse->phys, "/input0", sizeof(mouse->phys));
44      input_dev->name = mouse->name;
45      input_dev->phys = mouse->phys;
46      usb_to_input_id(dev, &input_dev->id);
47      input_dev->dev.parent = &intf->dev;
48      input_dev->evbit[0] = BIT_MASK(EV_KEY) | BIT_MASK(EV_REL);
49      input_dev->keybit[BIT_WORD(BTN_MOUSE)] = BIT_MASK(BTN_LEFT) |
50          BIT_MASK(BTN_RIGHT) | BIT_MASK(BTN_MIDDLE);
51      input_dev->relbit[0] = BIT_MASK(REL_X) | BIT_MASK(REL_Y);
52      input_dev->keybit[BIT_WORD(BTN_MOUSE)] |= BIT_MASK(BTN_SIDE) |
53          BIT_MASK(BTN_EXTRA);
54      input_dev->relbit[0] |= BIT_MASK(REL_WHEEL);
55      input_set_drvdata(input_dev, mouse);
56      input_dev->open = usb_mouse_open;
57      input_dev->close = usb_mouse_close;
58      usb_fill_int_urb(mouse->irq, dev, pipe, mouse->data,
59          (maxp > 8 ? 8 : maxp),
60          usb_mouse_irq, mouse, endpoint->bInterval);
61      mouse->irq->transfer_dma = mouse->data_dma;
62      mouse->irq->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;
63      error = input_register_device(mouse->dev);
64      if (error)
65          goto fail3;
66      usb_set_intfdata(intf, mouse);
67          return 0;
68  fail3:
69      usb_free_urb(mouse->irq);
70  fail2:
71      usb_free_coherent(dev, 8, mouse->data, mouse->data_dma);
72  fail1:
73      input_free_device(input_dev);
74      kfree(mouse);
75      return error;
```

```
76 }
```

ПРИЛОЖЕНИЕ Б

В листинге 4.2 представлен полный код драйвера.

Листинг 4.2 – Полный код драйвера

```
#define POLLING_RATE 125
#define ACCELERATION 0.1f
#define SENSITIVITY 1.0f
#define SENS_CAP 0.0f
#define OFFSET 0.0f
#define PRE_SCALE_X 1.0f
#define PRE_SCALE_Y 1.0f
#define POST_SCALE_X 1.0f
#define POST_SCALE_Y 1.0f
#define SPEED_CAP 0.0f


#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>

#define DRIVER_AUTHOR "azolotukhin"
#define DRIVER_DESC "USB HID Boot Protocol mouse driver with acceleration"
#define DRIVER_LICENSE "GPL"

MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(DRIVER_DESC);
MODULE_LICENSE(DRIVER_LICENSE);

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;
```

```
37 };
38
39 static inline int my_round(float x)
40 {
41     if (x >= 0) {
42         return (int)(x + 0.5f);
43     } else {
44         return (int)(x - 0.5f);
45     }
46 }
47
48 float accel_sens = SENSITIVITY;
49
50 static void analyze_wheel(float *delta_x, float *delta_y, int
       wheel)
51 {
52
53     if(wheel > 0)
54     {
55         printk("ACCELERATOR: Increase mouse speed. accel_sens =
              %d", (int)(accel_sens*10));
56         accel_sens += ACCELERATION;
57         if (SENS_CAP > 0 && accel_sens >= SENS_CAP) {
58             accel_sens = SENS_CAP;
59         }
60
61     }
62     else if(wheel < 0)
63     {
64         printk("ACCELERATOR: Decrease mouse speed. accel_sens =
              %d", (int)(accel_sens*10));
65         accel_sens -= ACCELERATION;
66         if (accel_sens < 0.1f) {
67             accel_sens = 0.1f;
68         }
69
70     }
71
72     accel_sens /= SENSITIVITY;
73     *delta_x *= accel_sens;
74     *delta_y *= accel_sens;
```

```
75 }
76
77 static void usb_mouse_irq(struct urb *urb)
78 {
79     struct usb_mouse *mouse = urb->context;
80     signed char *data = mouse->data;
81     struct input_dev *dev = mouse->dev;
82     int status;
83
84     float delta_x = data[1] * PRE_SCALE_X;
85     float delta_y = data[2] * PRE_SCALE_Y;
86
87     static float carry_x = 0.0f;
88     static float carry_y = 0.0f;
89
90     analyze_wheel(&delta_x, &delta_y, data[3]);
91
92     delta_x *= POST_SCALE_X;
93     delta_y *= POST_SCALE_Y;
94     delta_x += carry_x;
95     delta_y += carry_y;
96     carry_x = delta_x - my_round(delta_x);
97     carry_y = delta_y - my_round(delta_y);
98
99     switch (urb->status) {
100         case 0:
101         break;
102         case -ECONNRESET:
103         case -ENOENT:
104         case -ESHUTDOWN:
105         return;
106         default:
107         goto resubmit;
108     }
109
110     input_report_key(dev, BTN_LEFT,   data[0] & 0x01);
111     input_report_key(dev, BTN_RIGHT,  data[0] & 0x02);
112     input_report_key(dev, BTN_MIDDLE, data[0] & 0x04);
113     input_report_key(dev, BTN_SIDE,   data[0] & 0x08);
114     input_report_key(dev, BTN_EXTRA,  data[0] & 0x10);
115
```

```
116     input_report_rel(dev, REL_X,     Leet_round(delta_x));
117     input_report_rel(dev, REL_Y,     Leet_round(delta_y));
118     input_report_rel(dev, REL_WHEEL, data[3]);
119
120     input_sync(dev);
121     resubmit:
122     status = usb_submit_urb (urb, GFP_ATOMIC);
123     if (status)
124     dev_err(&mouse->usbdev->dev,
125     "can't␣resubmit␣intr,␣%s-%s/input0,␣status␣%d\n",
126     mouse->usbdev->bus->bus_name,
127     mouse->usbdev->devpath, status);
128 }
129
130 static int usb_mouse_open(struct input_dev *dev)
131 {
132     struct usb_mouse *mouse = input_get_drvdata(dev);
133
134     mouse->irq->dev = mouse->usbdev;
135     if (usb_submit_urb(mouse->irq, GFP_KERNEL))
136     return -EIO;
137
138     return 0;
139 }
140
141 static void usb_mouse_close(struct input_dev *dev)
142 {
143     struct usb_mouse *mouse = input_get_drvdata(dev);
144
145     usb_kill_urb(mouse->irq);
146 }
147
148 static int usb_mouse_probe(struct usb_interface *intf, const
       struct usb_device_id *id)
149 {
150     struct usb_device *dev = interface_to_usbdev(intf);
151     struct usb_host_interface *interface;
152     struct usb_endpoint_descriptor *endpoint;
153     struct usb_mouse *mouse;
154     struct input_dev *input_dev;
155     int pipe, maxp;
```

```
156     int error = -ENOMEM;
157
158     interface = intf->cur_altsetting;
159
160     if (interface->desc.bNumEndpoints != 1)
161     return -ENODEV;
162
163     endpoint = &interface->endpoint[0].desc;
164     if (!usb_endpoint_is_int_in(endpoint))
165     return -ENODEV;
166
167     pipe = usb_rcvintpipe(dev, endpoint->bEndpointAddress);
168     maxp = usb_maxpacket(dev, pipe);
169
170     mouse = kzalloc(sizeof(struct usb_mouse), GFP_KERNEL);
171     input_dev = input_allocate_device();
172     if (!mouse || !input_dev)
173     goto fail1;
174
175     mouse->data = usb_alloc_coherent(dev, 8, GFP_ATOMIC,
          &mouse->data_dma);
176     if (!mouse->data)
177     goto fail1;
178
179     mouse->irq = usb_alloc_urb(0, GFP_KERNEL);
180     if (!mouse->irq)
181     goto fail2;
182
183     mouse->usbdev = dev;
184     mouse->dev = input_dev;
185
186     if (dev->manufacturer)
187     strlcpy(mouse->name, dev->manufacturer, sizeof(mouse->name));
188
189     if (dev->product) {
190         if (dev->manufacturer)
191         strlcat(mouse->name, " ", sizeof(mouse->name));
192         strlcat(mouse->name, dev->product, sizeof(mouse->name));
193     }
194
195     if (!strlen(mouse->name))
```

```
196     snprintf(mouse->name, sizeof(mouse->name),
197     "USB␣HIDBP␣Mouse␣%04x:%04x",
198     le16_to_cpu(dev->descriptor.idVendor),
199     le16_to_cpu(dev->descriptor.idProduct));
200
201     usb_make_path(dev, mouse->phys, sizeof(mouse->phys));
202     strlcat(mouse->phys, "/input0", sizeof(mouse->phys));
203
204     input_dev->name = mouse->name;
205     input_dev->phys = mouse->phys;
206     usb_to_input_id(dev, &input_dev->id);
207     input_dev->dev.parent = &intf->dev;
208
209     input_dev->evbit[0] = BIT_MASK(EV_KEY) | BIT_MASK(EV_REL);
210     input_dev->keybit[BIT_WORD(BTN_MOUSE)] = BIT_MASK(BTN_LEFT) |
211     BIT_MASK(BTN_RIGHT) | BIT_MASK(BTN_MIDDLE);
212     input_dev->relbit[0] = BIT_MASK(REL_X) | BIT_MASK(REL_Y);
213     input_dev->keybit[BIT_WORD(BTN_MOUSE)] |= BIT_MASK(BTN_SIDE) |
214     BIT_MASK(BTN_EXTRA);
215     input_dev->relbit[0] |= BIT_MASK(REL_WHEEL);
216
217     input_set_drvdata(input_dev, mouse);
218
219     input_dev->open = usb_mouse_open;
220     input_dev->close = usb_mouse_close;
221
222     usb_fill_int_urb(mouse->irq, dev, pipe, mouse->data,
223     (maxp > 8 ? 8 : maxp),
224     usb_mouse_irq, mouse, endpoint->bInterval);
225     mouse->irq->transfer_dma = mouse->data_dma;
226     mouse->irq->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;
227
228     error = input_register_device(mouse->dev);
229     if (error)
230     goto fail3;
231
232     usb_set_intfdata(intf, mouse);
233     return 0;
234
235     fail3:
236     usb_free_urb(mouse->irq);
```

```
237     fail2:
238     usb_free_coherent(dev, 8, mouse->data, mouse->data_dma);
239     fail1:
240     input_free_device(input_dev);
241     kfree(mouse);
242     return error;
243 }
244
245 static void usb_mouse_disconnect(struct usb_interface *intf)
246 {
247     struct usb_mouse *mouse = usb_get_intfdata (intf);
248
249     usb_set_intfdata(intf, NULL);
250     if (mouse) {
251         usb_kill_urb(mouse->irq);
252         input_unregister_device(mouse->dev);
253         usb_free_urb(mouse->irq);
254         usb_free_coherent(interface_to_usbdev(intf), 8,
              mouse->data, mouse->data_dma);
255         kfree(mouse);
256     }
257 }
258
259 static struct usb_device_id usb_mouse_id_table [] = {
260     { USB_INTERFACE_INFO(USB_INTERFACE_CLASS_HID,
          USB_INTERFACE_SUBCLASS_BOOT,
261         USB_INTERFACE_PROTOCOL_MOUSE) },
262     { } /* Terminating entry */
263 };
264
265 MODULE_DEVICE_TABLE (usb, usb_mouse_id_table);
266
267 static struct usb_driver usb_mouse_driver = {
268     .name       = "accelerator",
269     .probe      = usb_mouse_probe,
270     .disconnect = usb_mouse_disconnect,
271     .id_table   = usb_mouse_id_table,
272 };
273
274 module_usb_driver(usb_mouse_driver);
```