



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 (ИУ)»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии (ИУ7)»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

*«Разработка драйвера для настройки направления
скроллинга USB-мыши»*

Студент ИУ7-75Б
(Группа)

(Подпись, дата)

И. О. Артемьев
(И. О. Фамилия)

Руководитель курсовой работы

(Подпись, дата)

Н. Ю. Рязанова
(И. О. Фамилия)

2023 г.

РЕФЕРАТ

В рамках данной курсовой работы был разработан драйвер для настройки направления скроллинга USB-мыши.

Ключевые слова: скроллинг, USB-мышь, драйвер.

Рассчетно-пояснительная записка к курсовой работе содержит 24 страницы, 6 иллюстраций, 3 источника, 1 приложение.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 Аналитический раздел	5
1.1 Постановка задачи	5
1.2 Анализ основных принципов работы USB-мыши	5
1.3 Анализ возможных типов драйвера	6
1.4 Анализ алгоритма регистрации HID-драйвера в системе	6
1.5 Анализ способов изменения направления скроллинга мыши	8
1.6 Вывод	9
2 Конструкторский раздел	10
2.1 Диаграмма IDEF0	10
2.2 Инициализация структуры hid_driver	10
2.3 Структура для хранения информации о мыши	11
2.4 Алгоритм обработки сырых данных	11
3 Технологический раздел	13
3.1 Выбор языка и среды программирования	13
3.2 Реализация парсинга настроечного файла	13
3.3 Реализация драйвера	14
3.4 Makefile	16
4 Исследовательский раздел	17
4.1 Демонстрация работы программы	17
ЗАКЛЮЧЕНИЕ	19
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	20
ПРИЛОЖЕНИЕ А	21

ВВЕДЕНИЕ

Разработка драйверов для устройств является важной составляющей в обеспечении правильной работы аппаратных средств и расширении функциональности компьютерных систем. Одним из таких устройств является USB-мышь, стандартное устройство в компьютерном арсенале, которое обеспечивает пользовательский интерфейс для взаимодействия с операционной системой.

Одной из ключевых возможностей USB-мыши является скроллинг – способ перемещения содержимого экрана вверх или вниз. Направление этого скроллинга может оказаться важным для конечного пользователя, и в различных сценариях использования может потребоваться переключение направления скроллинга в соответствии с предпочтениями пользователя или специфическими требованиями.

Данная курсовая работа посвящена разработке драйвера для настройки направления скроллинга USB-мыши.

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать драйвер для настройки направления скроллинга USB-мыши.

Для решения поставленной задачи необходимо:

- провести анализ основных принципов работы мыши;
- провести анализ структуры драйвера мыши;
- провести анализ существующих способов изменения направления скроллинга мыши.
- разработать алгоритмы и структуру программного обеспечения;
- провести анализ работы разработанного программного обеспечения.

1.2 Анализ основных принципов работы USB-мыши

USB-мышь представляет собой устройство ввода, используемое для управления курсором на компьютерном экране. Она соединяется с компьютером через USB-порт и основана на использовании оптического или лазерного сенсора для определения движения. Этот сенсор сканирует поверхность, на которой находится мышь, и передает данные о движении курсора в операционную систему.

Основными компонентами USB-мыши являются кнопки (обычно левая, правая и средняя) и колесо прокрутки. Она использует протоколы, такие как HID, для взаимодействия с компьютером и передачи информации о движении, нажатиях и вращении колеса. Эти данные интерпретируются операционной системой, что позволяет пользователям управлять курсором, выполнять клики и скроллинг.

Технологии USB-мышей постоянно развиваются, добавляя новые функции, улучшая точность определения движения и обогащая возможности настройки для конечного пользователя. Эволюция включает в себя беспроводные соединения, расширенную функциональность кнопок и дополнительные опции скроллинга.

Основные принципы работы USB-мыши лежат в основе ее функциональности как удобного и надежного устройства для управления компьютером, обеспечивая пользователю широкий спектр возможностей взаимодействия с интерфейсом и выполнения задач.

1.3 Анализ возможных типов драйвера

В операционных системах Unix/Linux существуют три типа драйверов:

1. встроенные – выполнение этих драйверов инициализируется при запуске системы;
2. драйверы, код которых поделен между ядром и специальной утилитой;
3. драйверы, реализованные как загружаемые модули ядра.

Среди загружаемых модулей ядра выделяются HID-драйверы. HID (Human Interface Device) – это один из наиболее часто используемых классов устройств USB. Он включает в себя устройства, предназначенные для взаимодействия пользователя с компьютером.

Для изменения функциональности USB-мыши в системе Unix/Linux требуется разработать именно HID-драйвер. HID-драйверы специализируются на работе с устройствами класса HID, что позволяет изменять и настраивать функции и возможности устройств, таких как мыши, клавиатуры и другие устройства ввода, используемые для взаимодействия пользователя с компьютером.

1.4 Анализ алгоритма регистрации HID-драйвера в системе

Регистрация HID-драйвера подразумевает:

1. заполнение структуры `hid_driver`;
2. регистрация структуры в системе.

Сначала требуется инициализировать поля структуры `hid_driver`, представленной на листинге 1.1.

Листинг 1.1 – struct hid_driver

```
1 struct hid_driver
2 {
3     char *name;
4     const struct hid_device_id *id_table;
5     ...
6     int (*probe)(struct hid_device *dev, const struct hid_device_id *id);
7     void (*remove)(struct hid_device *dev);
8     ...
9     int (*raw_event)(struct hid_device *hdev, struct hid_report *report, u8
10         *data, int size);
11     ...
12 }
```

- `char *name` – это поле определяет имя драйвера. Имя драйвера представляет собой строку символов, идентифицирующую данный драйвер;
- `const struct hid_device_id *id_table` – это поле указывает на таблицу идентификаторов устройств, для которых применяется данный драйвер. Эта таблица содержит информацию о поддерживаемых устройствах и их идентификаторах;
- `int (*probe)(struct hid_device *dev, const struct hid_device_id *id)` – это указатель на функцию `probe`, которая вызывается при обнаружении нового устройства. Она инициализирует устройство и осуществляет его подключение к драйверу;
- `void (*remove)(struct hid_device *dev)` – это указатель на функцию `remove`, которая вызывается при отключении устройства от драйвера. Она отвечает за корректное завершение работы с устройством;
- `int (*raw_event)(struct hid_device *hdev, struct hid_report *report, u8 *data, int size)` – это указатель на функцию `raw_event`, которая обрабатывает низкоуровневые события от устройства, такие как получение отчетов (reports) от устройства HID. Она разбирает полученные данные от устройства и выполняет необходимые действия в соответствии с протоколом работы HID-устройства.

Регистрация и deregистрация драйвера в системе выполняется в макросах `module_init` и `module_exit` с помощью функций `hid_register_driver()`

и `hid_unregister_driver()`, которым передается адрес инициализированной структуры `struct hid_driver`.

1.5 Анализ способов изменения направления скроллинга мыши

Модификация дескриптора отчета (Report Descriptor Modification) и перехват сырых событий (Raw Event Interception) – это два подхода к изменению поведения USB-устройств, таких как мышь, с целью переопределения направления скроллинга.

Модификация дескриптора отчета работает путем изменения формата данных, которые устройство отправляет компьютеру. Это позволяет создать пользовательский дескриптор, что обеспечивает гибкость в настройке скроллинга. Замена отчетного дескриптора осуществляется с помощью поля `report_fixup` в структуре драйвера [1], куда передается указатель на функцию для замены дескриптора. Пример замены отчетного дескриптора представлен на листинге 1.2.

Листинг 1.2 – Функция замены отчетного дескриптора

```
1 static u8* report_fixup(struct hid_device *hdev, u8 *rdesc, unsigned int *  
    rsize)  
2 {  
3     *rsize = sizeof(my_rdesc);  
4     return my_rdesc;  
5 }
```

Изменение данных в сырых событиях представляет собой метод модификации информации, получаемой от устройства. Этот подход обладает высокой гибкостью, позволяя пользователю настраивать скроллинг в соответствии с индивидуальными предпочтениями. Благодаря реализации на уровне программного обеспечения, он обеспечивает относительно легкий доступ для пользователей. Изменение данных происходит с помощью поля `raw_event` в структуре драйвера, куда передается указатель на функцию для обработки событий. Пример обработки событий скроллинга мыши представлен на листинге 1.3.

Листинг 1.3 – Функция обработки данных от устройства

```
1 static int mouse_hid_input(struct hid_device* hdev, struct hid_report*  
   report, u8* data, int size)  
2 {  
3     struct mouse_sc* sc = hid_get_drvdata(hdev);  
4  
5     if (sc->is_natural_scrolling == 1 && size > 0) {  
6         if (data[size - 1] == 0xFF) {  
7             data[size - 1] = 0x01;  
8         }  
9         else if (data[size - 1] == 0x01) {  
10            data[size - 1] = 0xFF;  
11        }  
12    }  
13  
14    return 0;  
15 }
```

Именно изменение данных в сырых событиях будет использоваться в качестве подхода для изменения направления скроллинга мыши, в виду своей простоты и эффективности реализации.

1.6 Вывод

В результате проведенного анализа было решено:

- для выполнения задания разработать HID-драйвер, который должен быть реализован как загружаемый модуль ядра;
- для изменения направления скроллинга мыши использовать изменение данных в сырых событиях.

2 Конструкторский раздел

2.1 Диаграмма IDEF0

На рисунке 2.1 приведена диаграмма состояний IDEF0 нулевого уровня, а на рисунке 2.2 – диаграмма состояний IDEF0 первого уровня.

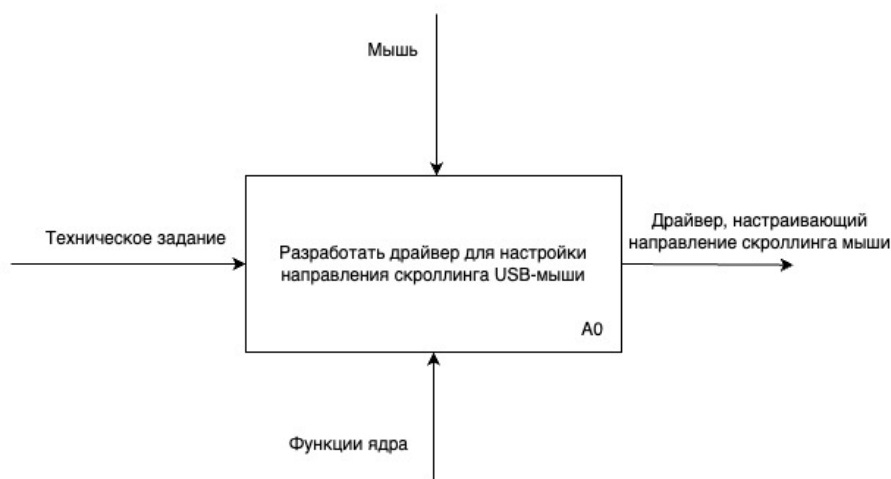


Рисунок 2.1 – Диаграмма состояний IDEF0 нулевого уровня

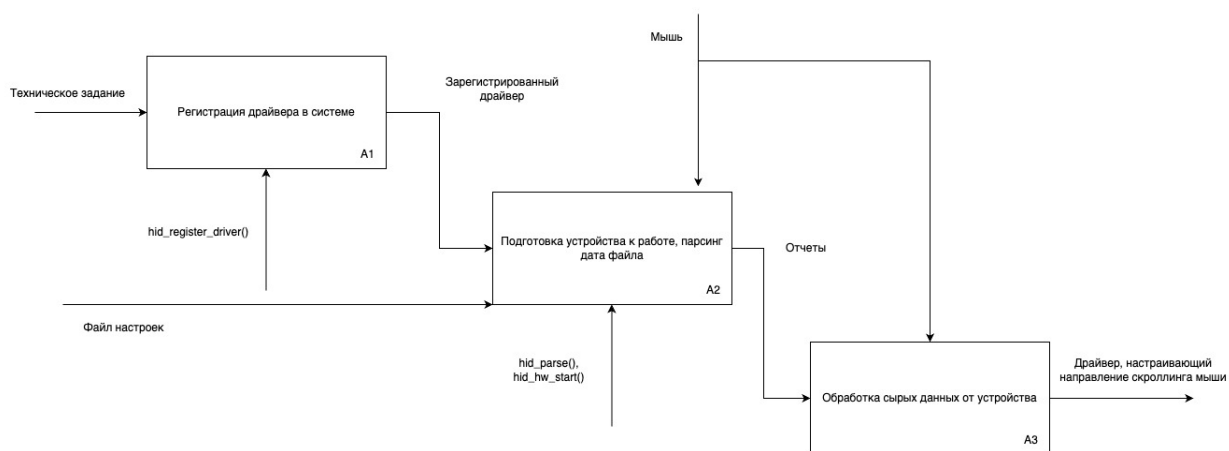


Рисунок 2.2 – Диаграмма состояний IDEF0 первого уровня

2.2 Инициализация структуры hid_driver

Для создания HID-драйвера создается экземпляр структуры `hid_driver`. Создание экземпляра приведено на листинге 2.1.

Листинг 2.1 – Инициализация структуры `hid_driver`

```
1 static struct hid_driver mouse_driver = {
2     .name = "custom_mouse_driver",
3     .id_table = mouse_table,
4     .probe = mouse_probe,
5     .remove = mouse_disconnect,
6     .raw_event = mouse_hid_input,
7 };
```

Для регистрации HID-драйвера используется функция `hid_register_driver`.

2.3 Структура для хранения информации о мыши

Для передачи данных, связанных с мышью, была создана структура `mouse_sc`, приведенная на листинге 2.2.

Листинг 2.2 – Структура `mouse_sc`

```
1 struct mouse_sc {
2     struct hid_device* hdev;
3     int is_natural_scrolling;
4 };
```

Поля этой структуры:

- `hdev` – указатель на структуру устройства;
- `is_natural_scrolling` – флаг, указывающий на то, включен ли режим естественного скроллинга или нет.

2.4 Алгоритм обработки сырых данных

На рисунке 2.3 показана схема алгоритма обработки сырых данных для скроллинга.

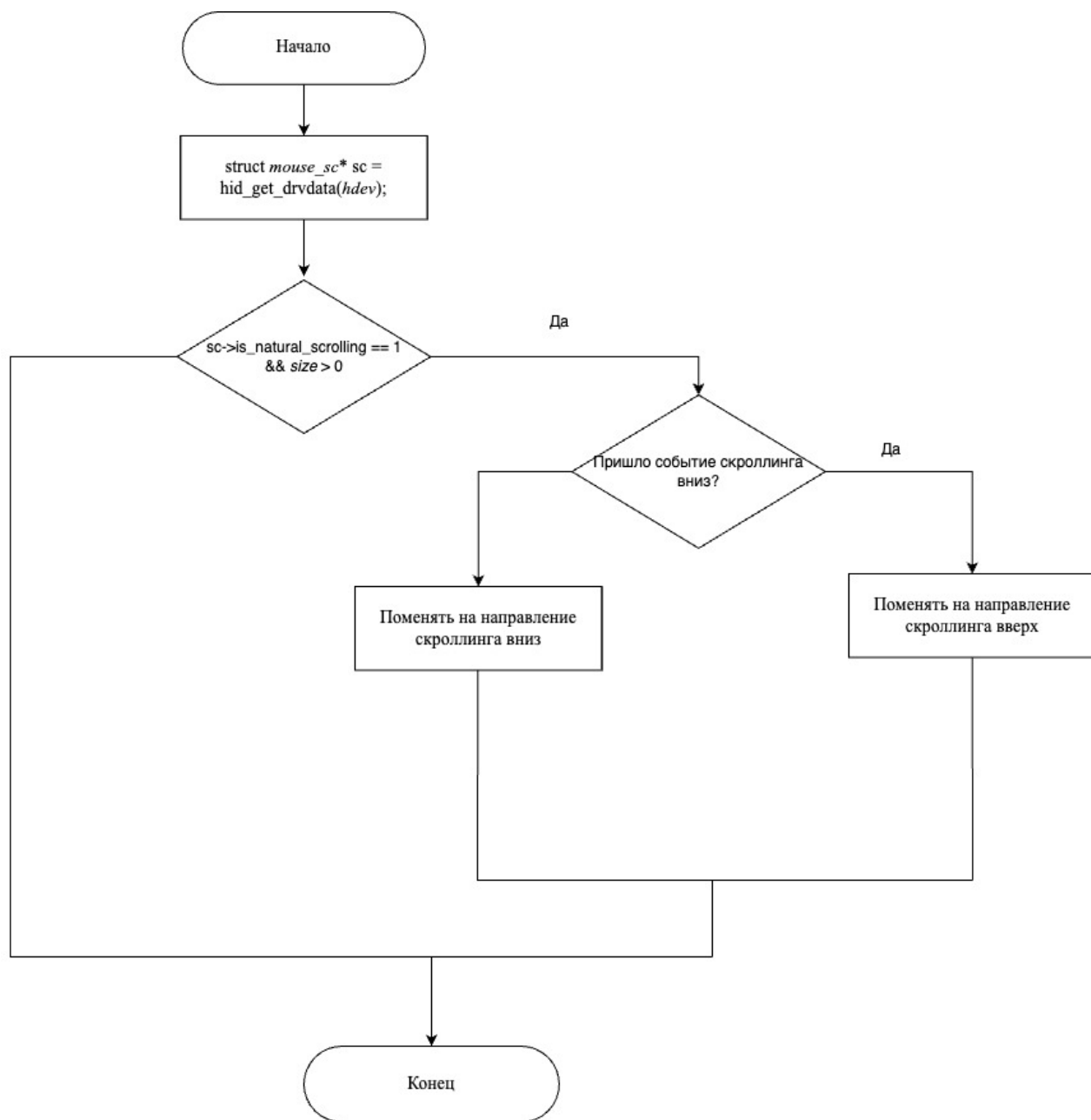


Рисунок 2.3 – Алгоритм обработки сырых данных

3 Технологический раздел

3.1 Выбор языка и среды программирования

В качестве языка программирования был выбран язык C [2], так как большая часть исходного кода ядра Linux, его модулей и драйверов написана на C.

В качестве среды разработки была выбрана Visual Studio Code [3] в связи с простотой и удобством использования.

3.2 Реализация парсинга настроечного файла

На листинге 3.1 представлена реализация парсинга настроечного файла.

Функция проверяет установлен ли флаг `is_natural_scrolling` в файле настроек. На вход функции подается абсолютный путь к файлу, а на выходе возвращается распаршенное значение.

Листинг 3.1 – Реализация функции парсинга настроечного файла

```
1 static int get_scrolling_value(const char* filepath)
2 {
3     char settings[SETTINGS_SIZE];
4     struct file* file;
5     loff_t pos = 0;
6     int bytes_read;
7
8     file = filp_open(filepath, O_RDONLY, 0);
9     if (IS_ERR(file)) {
10         printk(KERN_ERR "file can't open\n");
11         return -ENOENT;
12     }
13
14     vfs_llseek(file, pos, SEEK_SET);
15     bytes_read = kernel_read(file, settings, sizeof(settings), &pos);
16     filp_close(file, NULL);
17
18     int is_natural_scrolling = -1;
19
20     if (bytes_read > 0) {
21         char* found = strstr(settings, "Natural scrolling: ");
22         if (found != NULL) {
23             found += strlen("Natural scrolling: ");
24             is_natural_scrolling = *found - '0';
25         }
26     }
```

```

27
28     printk(KERN_INFO "is_natural_scrolling: %d\n", is_natural_scrolling);
29
30     return is_natural_scrolling;
31 }

```

3.3 Реализация драйвера

На листинге 3.2 представлена реализация функции probe.

Листинг 3.2 – Реализация функции probe

```

1  static int mouse_probe(struct hid_device* hdev, const struct hid_device_id*
    id)
2  {
3      int ret;
4      struct mouse_sc* sc;
5
6      sc = devm_kzalloc(&hdev->dev, sizeof(*sc), GFP_KERNEL);
7      if (sc == NULL) {
8          return -ENOMEM;
9      }
10
11     sc->hdev = hdev;
12     sc->is_natural_scrolling = get_scrolling_value(SETTINGS_PATH);
13
14     hid_set_drvdata(hdev, sc);
15
16     ret = hid_parse(hdev);
17     if (ret)
18     {
19         hid_err(hdev, "parse failed\n");
20         return ret;
21     }
22
23     ret = hid_hw_start(hdev, HID_CONNECT_DEFAULT);
24     if (ret) {
25         hid_err(hdev, "hw start failed\n");
26         return ret;
27     }
28
29     printk(KERN_WARNING "Connect driver\n");
30
31     return 0;
32 }

```

На листинге 3.3 представлена реализация функции remove.

Листинг 3.3 – Реализация функции remove

```
1 static void mouse_disconnect(struct hid_device* hdev)
2 {
3     hid_hw_stop(hdev);
4 }
```

На листинге 3.4 представлена реализация функции raw_event.

Листинг 3.4 – Реализация функции raw_event

```
1 static int mouse_hid_input(struct hid_device* hdev, struct hid_report*
   report, u8* data, int size)
2 {
3     struct mouse_sc* sc = hid_get_drvdata(hdev);
4
5     if (sc->is_natural_scrolling == 1 && size > 0) {
6         if (data[size - 1] == 0xFF) {
7             data[size - 1] = 0x01;
8         }
9         else if (data[size - 1] == 0x01) {
10             data[size - 1] = 0xFF;
11         }
12     }
13
14     return 0;
15 }
```

На листинге 3.5 представлена реализация функции init для драйвера.

Листинг 3.5 – Реализация функции init

```
1 static int __init custom_mouse_init(void)
2 {
3     printk(KERN_INFO "Mouse module loaded\n");
4
5     int ret = hid_register_driver(&mouse_driver);
6     if (ret)
7         printk(KERN_ERR "Failed to register mouse driver: %d\n", ret);
8
9     return ret;
10 }
```

На листинге 3.6 представлена реализация функции `exit` для драйвера.

Листинг 3.6 – Реализация функции `exit`

```
1 static void __exit custom_mouse_exit(void)
2 {
3     printk(KERN_INFO "Mouse module unloaded\n");
4
5     hid_unregister_driver(&mouse_driver);
6 }
```

3.4 Makefile

На листинге 3.7 представлена реализация Makefile.

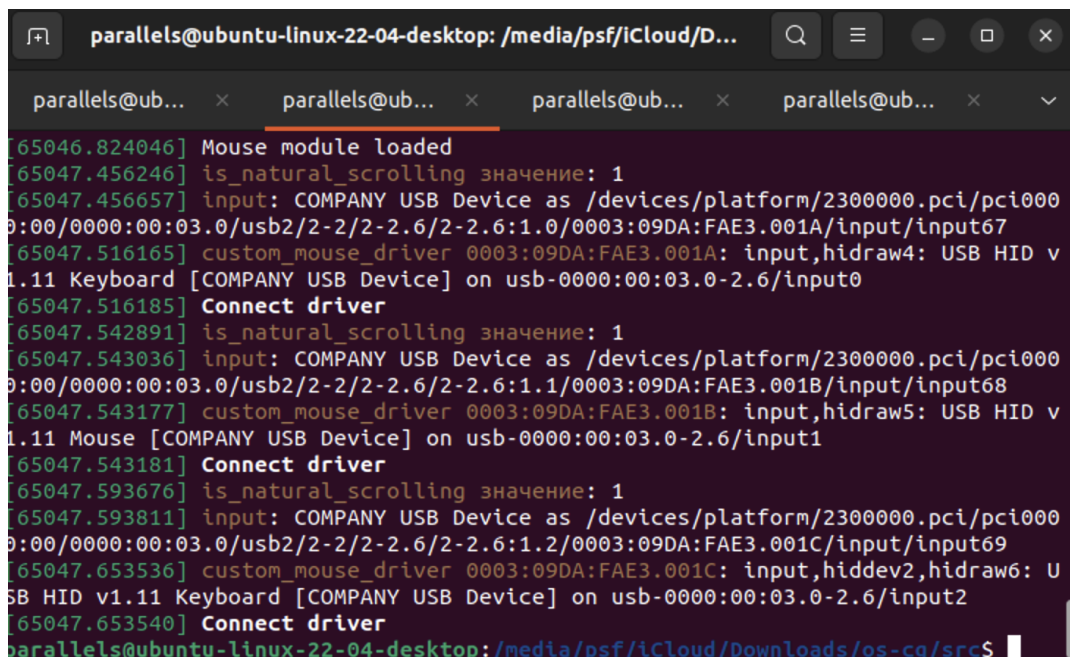
Листинг 3.7 – Makefile

```
1 CONFIG_MODULE_SIG=n
2
3 CURRENT = $(shell uname -r)
4 KDIR = /lib/modules/$(CURRENT)/build
5 PWD = $(shell pwd)
6
7 obj-m := mouse_scrolling_module.o
8 default:
9
10                                make -C $(KDIR) M=$(PWD) modules
11 clean:
12
13                                @rm -f *.o *.cmd *.flags *.mod.c *.order
14                                *.mod *.ko *.symvers
15                                @rm -f *.*.cmd *~ *.*~ TODO.* *.d
16                                @rm -fR .tmp*
17                                @rm -rf .tmp_versions
18
19 disclean: clean
20
21                                @rm *.ko *.symversS
```


4 Исследовательский раздел

4.1 Демонстрация работы программы

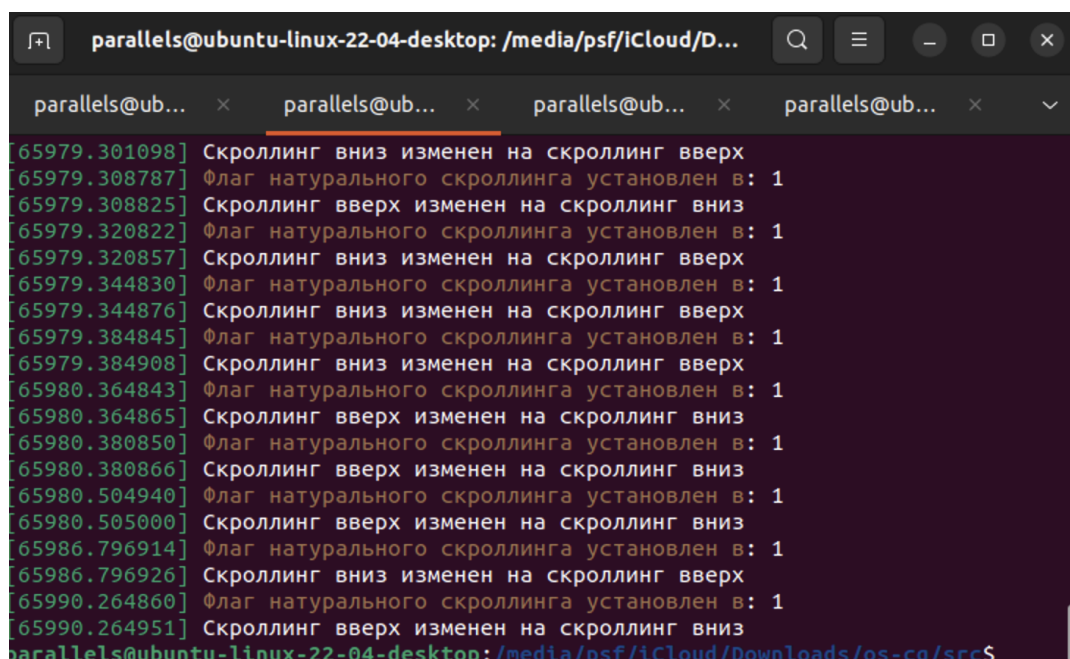
На рисунке 4.1 представлены логи загрузки драйвера в систему:



```
parallels@ubuntu-linux-22-04-desktop: /media/psf/iCloud/D...
parallels@ub... x parallels@ub... x parallels@ub... x parallels@ub... x
[65046.824046] Mouse module loaded
[65047.456246] is_natural_scrolling значение: 1
[65047.456657] input: COMPANY USB Device as /devices/platform/2300000.pci/pci000
0:00/0000:00:03.0/usb2/2-2/2-2.6/2-2.6:1.0/0003:09DA:FAE3.001A/input/input67
[65047.516165] custom_mouse_driver 0003:09DA:FAE3.001A: input,hidraw4: USB HID v
1.11 Keyboard [COMPANY USB Device] on usb-0000:00:03.0-2.6/input0
[65047.516185] Connect driver
[65047.542891] is_natural_scrolling значение: 1
[65047.543036] input: COMPANY USB Device as /devices/platform/2300000.pci/pci000
0:00/0000:00:03.0/usb2/2-2/2-2.6/2-2.6:1.1/0003:09DA:FAE3.001B/input/input68
[65047.543177] custom_mouse_driver 0003:09DA:FAE3.001B: input,hidraw5: USB HID v
1.11 Mouse [COMPANY USB Device] on usb-0000:00:03.0-2.6/input1
[65047.543181] Connect driver
[65047.593676] is_natural_scrolling значение: 1
[65047.593811] input: COMPANY USB Device as /devices/platform/2300000.pci/pci000
0:00/0000:00:03.0/usb2/2-2/2-2.6/2-2.6:1.2/0003:09DA:FAE3.001C/input/input69
[65047.653536] custom_mouse_driver 0003:09DA:FAE3.001C: input,hiddev2,hidraw6: U
SB HID v1.11 Keyboard [COMPANY USB Device] on usb-0000:00:03.0-2.6/input2
[65047.653540] Connect driver
parallels@ubuntu-linux-22-04-desktop: /media/psf/iCloud/Downloads/os-cg/src$
```

Рисунок 4.1 – Логи при загрузке драйвера в систему

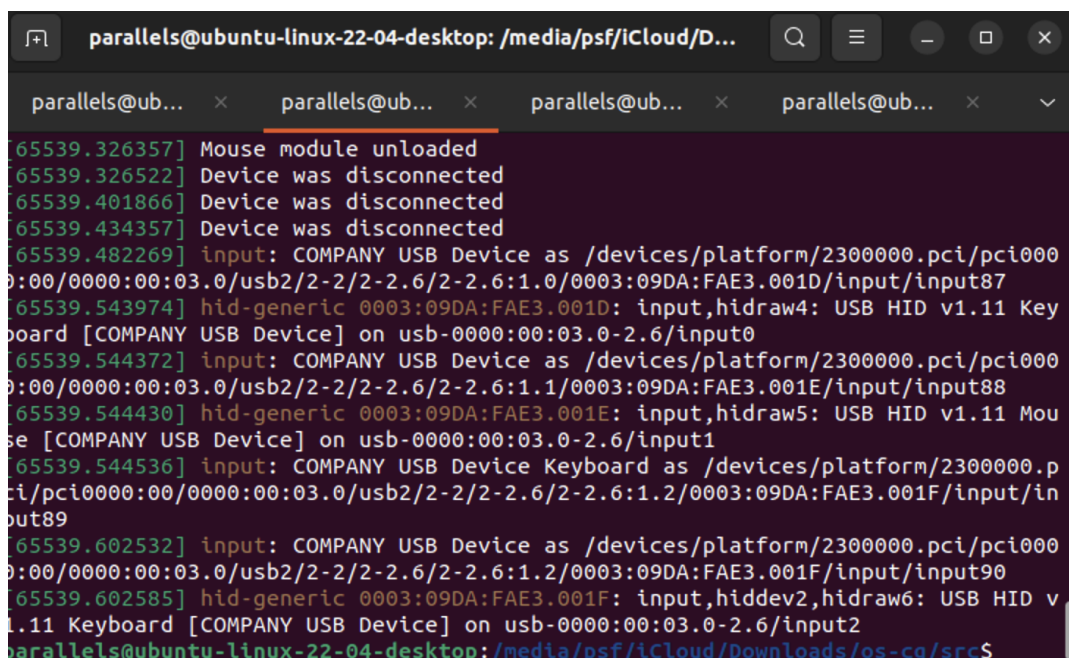
На рисунке 4.2 представлены логи при скроллинге мыши:



```
parallels@ubuntu-linux-22-04-desktop: /media/psf/iCloud/D...
parallels@ub... x parallels@ub... x parallels@ub... x parallels@ub... x
[65979.301098] Скроллинг вниз изменен на скроллинг вверх
[65979.308787] Флаг натурального скроллинга установлен в: 1
[65979.308825] Скроллинг вверх изменен на скроллинг вниз
[65979.320822] Флаг натурального скроллинга установлен в: 1
[65979.320857] Скроллинг вниз изменен на скроллинг вверх
[65979.344830] Флаг натурального скроллинга установлен в: 1
[65979.344876] Скроллинг вниз изменен на скроллинг вверх
[65979.384845] Флаг натурального скроллинга установлен в: 1
[65979.384908] Скроллинг вниз изменен на скроллинг вверх
[65980.364843] Флаг натурального скроллинга установлен в: 1
[65980.364865] Скроллинг вверх изменен на скроллинг вниз
[65980.380850] Флаг натурального скроллинга установлен в: 1
[65980.380866] Скроллинг вверх изменен на скроллинг вниз
[65980.504940] Флаг натурального скроллинга установлен в: 1
[65980.505000] Скроллинг вверх изменен на скроллинг вниз
[65986.796914] Флаг натурального скроллинга установлен в: 1
[65986.796926] Скроллинг вниз изменен на скроллинг вверх
[65990.264860] Флаг натурального скроллинга установлен в: 1
[65990.264951] Скроллинг вверх изменен на скроллинг вниз
parallels@ubuntu-linux-22-04-desktop: /media/psf/iCloud/Downloads/os-cg/src$
```

Рисунок 4.2 – Логи при скроллинге

На рисунке 4.3 представлены логи при выгрузке драйвера из системы:



```
parallels@ubuntu-linux-22-04-desktop: /media/psf/iCloud/D...
parallels@ub... x parallels@ub... x parallels@ub... x parallels@ub... x
[65539.326357] Mouse module unloaded
[65539.326522] Device was disconnected
[65539.401866] Device was disconnected
[65539.434357] Device was disconnected
[65539.482269] input: COMPANY USB Device as /devices/platform/23000000.pci/pci000
0:00/0000:00:03.0/usb2/2-2/2-2.6/2-2.6:1.0/0003:09DA:FAE3.001D/input/input87
[65539.543974] hid-generic 0003:09DA:FAE3.001D: input,hidraw4: USB HID v1.11 Key
board [COMPANY USB Device] on usb-0000:00:03.0-2.6/input0
[65539.544372] input: COMPANY USB Device as /devices/platform/23000000.pci/pci000
0:00/0000:00:03.0/usb2/2-2/2-2.6/2-2.6:1.1/0003:09DA:FAE3.001E/input/input88
[65539.544430] hid-generic 0003:09DA:FAE3.001E: input,hidraw5: USB HID v1.11 Mou
se [COMPANY USB Device] on usb-0000:00:03.0-2.6/input1
[65539.544536] input: COMPANY USB Device Keyboard as /devices/platform/23000000.p
ci/pci0000:00/0000:00:03.0/usb2/2-2/2-2.6/2-2.6:1.2/0003:09DA:FAE3.001F/input/in
put89
[65539.602532] input: COMPANY USB Device as /devices/platform/23000000.pci/pci000
0:00/0000:00:03.0/usb2/2-2/2-2.6/2-2.6:1.2/0003:09DA:FAE3.001F/input/input90
[65539.602585] hid-generic 0003:09DA:FAE3.001F: input,hiddev2,hidraw6: USB HID v
1.11 Keyboard [COMPANY USB Device] on usb-0000:00:03.0-2.6/input2
parallels@ubuntu-linux-22-04-desktop: /media/psf/iCloud/Downloads/os-cg/src$
```

Рисунок 4.3 – Логи при выгрузке драйвера из системы

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был разработан драйвер, позволяющий настраивать направление скроллинга USB-мыши.

В ходе написания работы были решены поставленные задачи:

- проведен анализ основных принципов работы мыши;
- проведен анализ структуры драйвера мыши;
- проведен анализ существующих способов изменения направления скроллинга мыши;
- разработаны алгоритмы и структура программного обеспечения;
- проведен анализ работы разработанного программного обеспечения.

Цель работы достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Рязанова Н., Сикорский О.* Метод изменения поведения HID-устройств под управлением ОС Linux // Новые информационные технологии в автоматизированных системах. — 2018.
2. ISO/IEC 9899:1990 Programming languages — C [Электронный ресурс]. — Режим доступа: <https://www.iso.org/standard/17782.html> (дата обращения: 25.12.2022).
3. Visual Studio Code [Электронный ресурс]. — Режим доступа: <https://code.visualstudio.com/> (дата обращения: 25.12.2022).

ПРИЛОЖЕНИЕ А

На листинге 4.1 представлен код загружаемого модуля ядра, реализующий драйвер мыши.

Листинг 4.1 – Загружаемый модуль ядра реализующий драйвер мыши

```
1 #include <linux/module.h>
2 #include <linux/kernel.h>
3 #include <linux/usb.h>
4 #include <linux/hid.h>
5 #include <linux/hiddev.h>
6 #include <linux/input.h>
7
8 MODULE_LICENSE("GPL");
9 MODULE_AUTHOR("Artemev Ilya Olegovich");
10 MODULE_DESCRIPTION("Module for changing the scrolling direction of a USB
    mouse");
11
12 #define VENDOR_A4TECH      0x09da
13 #define PRODUCT_A4TECH    0xfae3
14
15 #define SETTINGS_SIZE      20
16 #define SETTINGS_PATH      "/media/psf/iCloud/Downloads/os-cg/src/settings.
    txt"
17
18 struct mouse_sc {
19     struct hid_device* hdev;
20     int is_natural_scrolling;
21 };
22
23 static int get_scrolling_value(const char* filepath)
24 {
25     char settings[SETTINGS_SIZE];
26     struct file* file;
27     loff_t pos = 0;
28     int bytes_read;
29
30
31     file = filp_open(filepath, O_RDONLY, 0);
32     if (IS_ERR(file)) {
33         printk(KERN_ERR "Can't open file\n");
34         return -ENOENT;
35     }
36
37     vfs_llseek(file, pos, SEEK_SET);
38     bytes_read = kernel_read(file, settings, sizeof(settings), &pos);
39     filp_close(file, NULL);
40 }
```

```

41     int is_natural_scrolling = -1;
42
43     if (bytes_read > 0) {
44         char* found = strstr(settings, "Natural scrolling: ");
45         if (found != NULL) {
46             found += strlen("Natural scrolling: ");
47             is_natural_scrolling = *found - '0';
48         }
49     }
50
51     printk(KERN_INFO "is_natural_scrolling: %d\n", is_natural_scrolling);
52
53     return is_natural_scrolling;
54 }
55
56 static int mouse_hid_input(struct hid_device* hdev, struct hid_report*
57     report, u8* data, int size)
58 {
59     struct mouse_sc* sc = hid_get_drvdata(hdev);
60
61     printk(KERN_INFO "is_natural_scrolling: %d\n", sc->is_natural_scrolling)
62         ;
63
64     if (sc->is_natural_scrolling == 1 && size > 0) {
65         if (data[size - 1] == 0xFF) {
66             printk(KERN_INFO "Scrolling down changed to scrolling up\n");
67             data[size - 1] = 0x01;
68         }
69         else if (data[size - 1] == 0x01) {
70             printk(KERN_INFO "Scrolling up changed to scrolling down\n");
71             data[size - 1] = 0xFF;
72         }
73     }
74
75     return 0;
76 }
77
78 static int mouse_probe(struct hid_device* hdev, const struct hid_device_id*
79     id)
80 {
81     int ret;
82     struct mouse_sc* sc;
83
84     sc = devm_kzalloc(&hdev->dev, sizeof(*sc), GFP_KERNEL);
85     if (sc == NULL) {
86         return -ENOMEM;
87     }

```

```

86     sc->hdev = hdev;
87     sc->is_natural_scrolling = get_scrolling_value(SETTINGS_PATH);
88
89     hid_set_drvdata(hdev, sc);
90
91     ret = hid_parse(hdev);
92     if (ret)
93     {
94         hid_err(hdev, "parse failed\n");
95         return ret;
96     }
97
98     ret = hid_hw_start(hdev, HID_CONNECT_DEFAULT);
99     if (ret) {
100         hid_err(hdev, "hw start failed\n");
101         return ret;
102     }
103
104     printk(KERN_WARNING "Connect driver\n");
105
106     return 0;
107 }
108
109 static void mouse_disconnect(struct hid_device* hdev)
110 {
111     printk(KERN_INFO "Device was disconnected\n");
112     hid_hw_stop(hdev);
113 }
114
115 static struct hid_device_id mouse_table[] = {
116     { HID_USB_DEVICE(VENDOR_A4TECH, PRODUCT_A4TECH) },
117     {}
118 };
119 MODULE_DEVICE_TABLE(hid, mouse_table);
120
121 static struct hid_driver mouse_driver = {
122     .name = "custom_mouse_driver",
123     .id_table = mouse_table,
124     .probe = mouse_probe,
125     .remove = mouse_disconnect,
126     .raw_event = mouse_hid_input,
127 };
128
129 static int __init custom_mouse_init(void)
130 {
131     printk(KERN_INFO "Mouse module loaded\n");
132
133     int ret = hid_register_driver(&mouse_driver);

```

```
134     if (ret)
135         printk(KERN_ERR "Failed to register mouse driver: %d\n", ret);
136
137     return ret;
138 }
139
140 static void __exit custom_mouse_exit(void)
141 {
142     printk(KERN_INFO "Mouse module unloaded\n");
143
144     hid_unregister_driver(&mouse_driver);
145 }
146
147 module_init(custom_mouse_init);
148 module_exit(custom_mouse_exit);
```