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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Разработка ПО для изменения яркости экрана с помощью USB-мыши»

Студент ИУ7-75Б _____ А. Н. Прянишников
(Группа) (Подпись, дата) (И.О.Фамилия)

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

2023 г.

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

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-7
_____ И. В. Рудаков
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине «Операционные системы»

Студент группы ИУ7-75Б

Прянишников Александр Николаевич
(Фамилия, имя, отчество)

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

учебный

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание Разработать ПО для изменения яркости экрана с помощью USB-мыши

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на ___ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
Презентация на 8-10 слайдах.

Дата выдачи задания « » _____ 2022 г.

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

Студент _____ А. Н. Прянишников
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

РЕФЕРАТ

Расчетно-пояснительная записка 28 с., 5 рис., 13 лист., 13 ист.

В работе представлена реализация ПО для изменения яркости экрана с помощью USB-мыши.

Ключевые слова: яркость, USB, драйвер, Linux.

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

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Яркость экрана – одна из важнейших характеристик, влияющих на работу человека за компьютером. Далеко не все компьютеры оснащены клавишей, отвечающей за изменение яркости экрана. В то же время яркость необходимо регулировать в разные промежутки дня.

Мышь – координатное устройство для управления курсором и отдачи различных команд компьютеру. Оно широко распространено среди пользователей компьютеров, поэтому имеет смысл добавить возможность изменять яркость экрана через мышь.

Тема курсовой работы – разработать ПО, позволяющее изменять яркость экрана с помощью USB-мыши.

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с заданием на курсовую работу по дисциплине «Операционные системы» требуется разработать ПО, позволяющее изменять яркость экрана с помощью USB-мыши.

Для выполнения задания требуется решить следующие задачи:

1) Провести анализ существующих подходов к изменению функциональности внешних устройств в Linux.
2) Провести анализ существующих способов изменения яркости экрана.
3) Разработать алгоритмы, необходимые для реализации ПО.
4) Разработать ПО, предоставляющая требуемую функциональность.
5) Провести исследование разработанного ПО.

Одновременное нажатие SHIFT-R на клавиатуре и правой кнопки мыши на клавиатуре должно увеличивать яркость экрана, а SHIFT-R и левой кнопки мыши – уменьшать. 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 M190 [1] и операционная система Ubuntu [2].

1.2 Драйвер 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 Алгоритм 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 Подсистема ввода/вывода

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 со-

бытий: *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].

Структура *struct urb* не может быть создана статически в драйвере или внутри другой структуры [9]. Для её создания нужно воспользоваться системным вызовом *usb_alloc_urb*. Для завершения работы драйвера с URB, драйвер должен вызвать функцию *usb_free_urb*.

Чтобы правильно инициализировать URB перед отправкой на конечную точку при прерывании используется системный вызов *usb_fill_int_urb* [10].

Как только URB был должным образом создан и инициализирован драйвером, он готов к отправке.

Для отправки используется системный вызов *usb_submit_urb* [11].

1.5 Анализ способов изменения функциональности внешних устройств

Прерывание – сигнал к процессору, испускаемый аппаратными средствами или программным обеспечением, и указывающий на событие, которое требует немедленного внимания. Процессор отвечает, приостанавливая свои текущие действия, сохраняя свое состояние и выполняя функцию, называемую обработчиком прерываний, для обработки события.

В ОС Linux различают быстрые и медленные прерывания. Обработка быстрых прерываний выполняется от начала до конца. В современных системах осталось только одно быстрое прерывание – от таймера.

Чтобы сократить время выполнения обработчиков прерываний обработчики медленных аппаратных прерываний делятся на две части, которые назы-

ваются верхняя (top) и нижняя (bottom) половины (half). Выполнение нижних половин инициируется верхними половинами.

В современных ОС Linux имеется три типа нижних половин:

1) Softirq – отложенные прерывания.
2) Tasklet – тасклеты.
3) Workqueue – очереди работ.

Так как требуется обрабатывать события от мыши и клавиатуры, нужно определиться с тем, какой механизм использовать.

1.5.1 Очереди работ

Очередь работ является еще одной концепцией для обработки отложенных функций. Функции рабочих очередей выполняются в контексте процесса ядра. Это означает, что функции очереди задач не должны быть атомарными, как функции тасклета [13].

Очередь работ будет использоваться для обработки событий, связанных с мышью. Это связано с тем, что в обработчике прерывания будет происходить изменение яркости экрана, поэтому процесс может перейти в состояние ожидания.

1.6 Анализ подходов к изменению яркости экрана

Большинство ноутбуков предоставляют способ изменения яркости экрана с помощью горячих клавиш. Но для реализации изменения яркости с помощью мыши этот способ не работает. Поэтому разработчиками Ubuntu предусмотрен другой метод.

Максимально возможное значение яркости экрана содержится в файле */sys/class/backlight/intel_backlight/max_brightness*. Текущее же значение яркости экрана содержится в файле */sys/class/backlight/intel_backlight/brightness*. Для изменения текущей яркости экрана достаточно изменить значение, хранящееся в файле.

Для чтения и записи в файл нового значения яркости используется статический буфер. Для буфера определен размер - *MAX_BRINGHTNESS_CLASS*, равный 6. Поскольку буфер используется для чтения и записи числа, количество элементов выбрано таким образом, чтобы в буфер можно было записать значения в диапазоне от 0 до 10000.

1.7 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 очереди работ;

— для изменения яркости экрана требуется изменить значение, хранящееся в файле */sys/class/backlight/intel_backlight/brightness*;

2 Конструкторский раздел

2.1 Диаграмма IDEF0

На рисунке 2.1 изображена диаграмма IDEF0 для требуемой задачи:

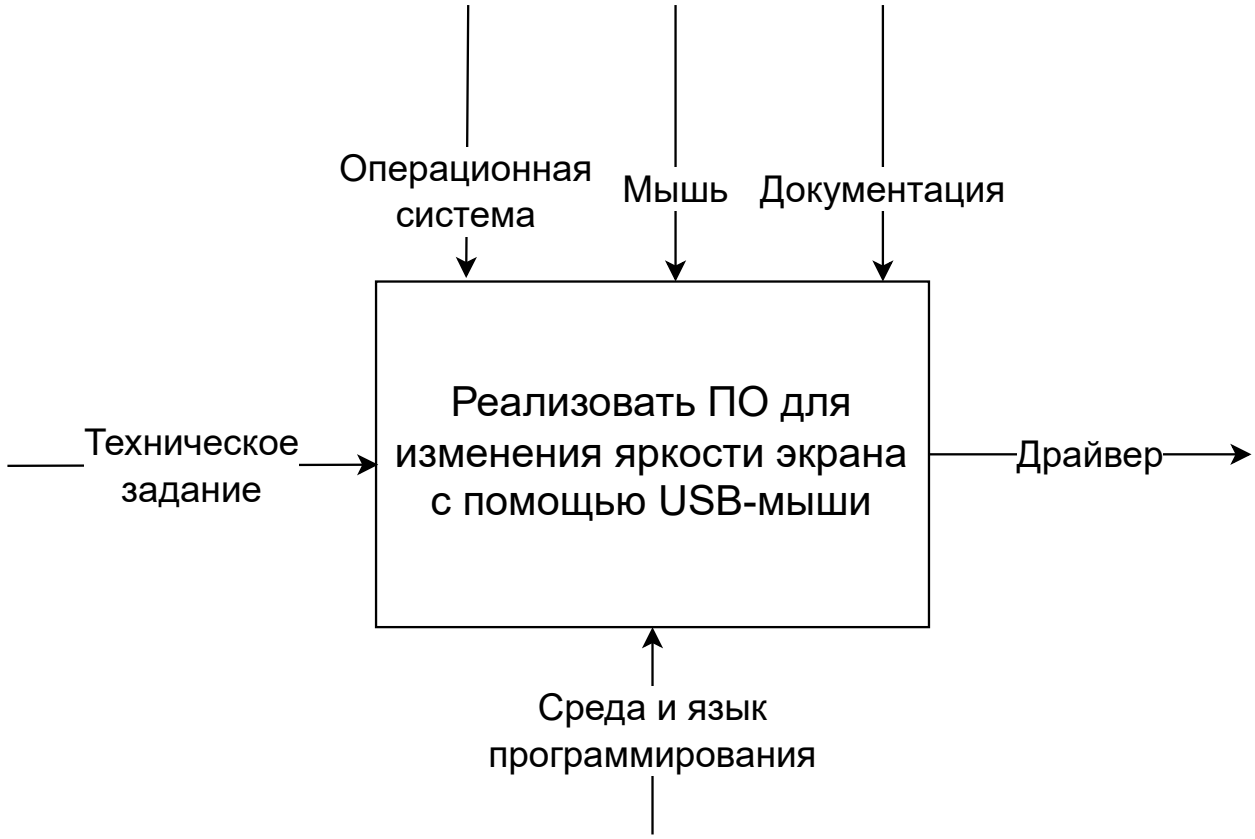

Рисунок 2.1 – Диаграмма IDEF0

2.2 Структура для хранения информации о мыши

Для передачи данных, связанных с мышью, была создана структура *mouse_t*, приведенная на листинге 1.

Листинг 1 – Структура mouse_t

```
struct mouse_t {
    unsigned char     *data;
    struct input_dev  *input_dev;
    struct usb_device *usb_dev;
    struct urb        *irq;
};
```

Поля этой структуры:

— data – данные, передаваемые мышью при прерывании;
— input_dev – структура для использования подсистемы входа/выхода;
— usb_device – представление usb-устройства;
— irq – указатель на обработчик прерываний.

2.3 Инициализация структуры usb_driver

Для создания USB-драйвера создается экземпляр структуры *usb_driver*. Создание экземпляра приведено на листинге 2.

Листинг 2 – Инициализация структуры usb_driver

```
static struct usb_driver my_mouse_driver = {
    .name       = DRIVER_NAME,
    .probe      = my_mouse_probe,
    .disconnect = my_mouse_disconnect,
    .id_table   = mouse_table,
};
```

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

2.4 Структура для передачи данных о прерывании

Для того, чтобы передавать в работу из очереди данные о прерывании была создана структура urb_workqueue, представленная на листинге 3.

Листинг 3 – Структура urb_workqueue

```
struct urb_workqueue {
    struct urb *urb;
    struct work_struct work;
};

typedef struct urb_workqueue urb_workqueue_t;
```

2.5 Алгоритм работы обработчика прерывания от мыши

На рисунке 2.2 представлена схема работы алгоритма обработчика прерываний мыши.

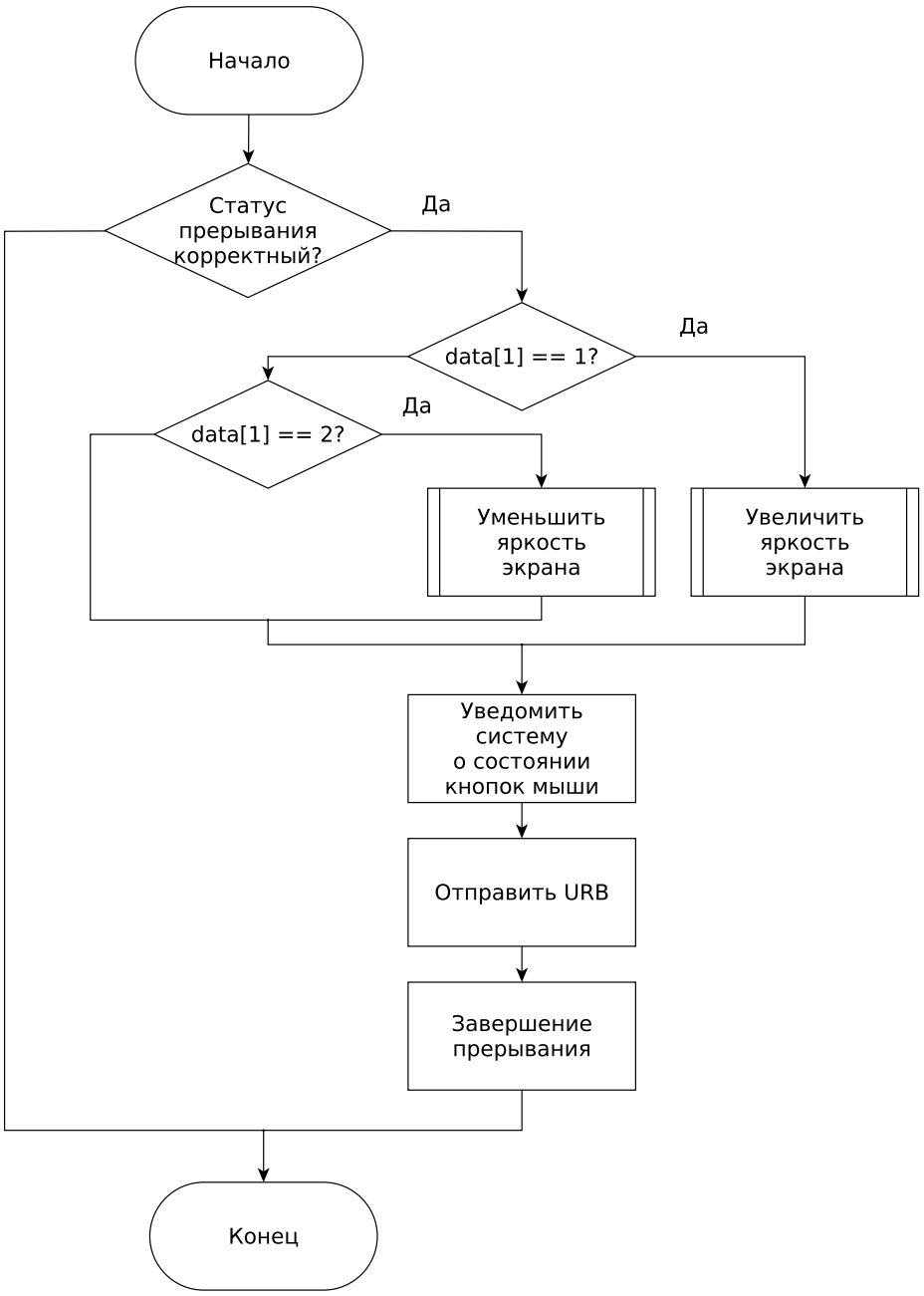

Рисунок 2.2 – Схема работы алгоритма обработчика прерываний мыши

2.6 Структура для хранения информации об яркости экрана

Для хранении информации об яркости экрана была создана структура data. Она и некоторые макросы для корректной работы системы представлены на листинге 4.

Листинг 4 – Структура data для яркости

```
#define BACKLIGHT_PATH "/sys/class/backlight/intel_backlight/brightness"
#define MAX_BACKLIGHT_PATH "/sys/class/backlight/intel_backlight/max_brightness"
#define MAX_BUFFER_LEN 6
#define STEP      10


struct bright_data {
        int max;
        int cur;
        int step;
}

static struct bright_data *data;
static char brightness_str[MAX_BUFFER_LEN];
```

2.7 Структура ПО

Структура ПО состоит из:

— загружаемый модуль ядра, реализующий драйвер USB-мыши;
— вспомогательный модуль, осуществляющий регулировку яркости экрана;
— вспомогательный модуль для работы с файлами из пространства ядра.

3 Технологический раздел

3.1 Выбор языка и среды программирования

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

В качестве среды разработки была выбрана Visual Studio Code.

3.2 Реализация изменения яркости

На листинге 5 представлена реализация изменения яркости.

Функция *brightness_update* записывает новое значение яркости в файл, а *backlight_change* рассчитывает в соответствии с наступившем событием это значение.

Листинг 5 – Реализация изменения яркости

```
void backlight_change(int flag) {

        if (flag == 0)
                return;

        if (flag == 1) {
                data->cur += data->step;
                if (data->cur  > data->max)
                        data->cur = data->max;
                printk(KERN_INFO "brightness.c: Backlight tmp_brightness was
                   increased!\n");
        }
        else if (flag == -1) {
                data->cur -= data->step;
                if (data->cur <= 8)
                        data->cur = 8;
                printk(KERN_INFO "brightness.c: Backlight tmp_brightness was
                   decreased!\n");
        }
        brightness_update();
}
```

3.3 Реализация обработчика прерываний от мыши

В приложении А представлена реализация обработчика прерываний от мыши.

Для определения нажатой клавиши используется поле *data* из структуры *mouse_t*. Второй байт поля отвечает за тип нажатой клавиши, поля с третьего по шестой включительно используются для вычисления новых координат.

3.4 Реализация драйвера

В листинге 6 представлена реализация функции *probe*.

Листинг 6 – Реализация функции probe

```
static int my_mouse_probe(struct usb_interface *interface, const struct
    usb_device_id *id) {
     struct usb_device *usb_device = interface_to_usbdev(interface);
     mouse_t *mouse;
     struct input_dev *input_dev;
     struct usb_endpoint_descriptor *endpoint;
     int error = -ENOMEM;

     printk(KERN_INFO "%s: probe checking my_mouse\n", DRIVER_NAME);

     mouse = kzalloc(sizeof(mouse_t), GFP_KERNEL);
     input_dev = input_allocate_device();
     if (!mouse || !input_dev) {
          input_free_device(input_dev);
          kfree(mouse);

          printk(KERN_ERR "%s: error when allocate device\n", DRIVER_NAME);
          return error;
     }

     mouse->data = (unsigned char *)usb_alloc_coherent(usb_device, USB_PACKET_LEN
         , GFP_KERNEL, &mouse->data_dma);
     if (!mouse->data) {
          input_free_device(input_dev);
          kfree(mouse);

          printk(KERN_ERR "%s: error when allocate coherent\n", DRIVER_NAME);
          return error;
     }

     mouse->irq = usb_alloc_urb(0, GFP_KERNEL);
     if (!mouse->irq) {
          usb_free_coherent(usb_device, USB_PACKET_LEN, mouse->data, mouse->
              data_dma);
          input_free_device(input_dev);
          kfree(mouse);

          printk(KERN_ERR "%s: error when allocate urb\n", DRIVER_NAME);
          return error;
     }

     mouse->usb_dev = usb_device;
     mouse->input_dev = input_dev;
     usb_make_path(usb_device, mouse->phys, sizeof(mouse->phys));
```

```
    strlcat(mouse->phys, "/input0", sizeof(mouse->phys));
    input_dev->name = DRIVER_NAME;
    input_dev->phys = mouse->phys;
    usb_to_input_id(usb_device, &input_dev->id);
    input_dev->dev.parent = &interface->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 = my_mouse_open;
    input_dev->close = my_mouse_close;
    endpoint = &interface->cur_altsetting->endpoint[0].desc;
    usb_fill_int_urb(
        mouse->irq, usb_device,
        usb_rcvintpipe(usb_device, endpoint->bEndpointAddress),
        mouse->data, USB_PACKET_LEN,
        my_mouse_irq, mouse, endpoint->bInterval
    );
    usb_submit_urb(mouse->irq, GFP_ATOMIC);
    mouse->irq->transfer_dma = mouse->data_dma;
    mouse->irq->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;
    error = input_register_device(mouse->input_dev);
    if (error) {
        usb_free_urb(mouse->irq);
        usb_free_coherent(usb_device, USB_PACKET_LEN, mouse->data, mouse->
           data_dma);
        input_free_device(input_dev);
        kfree(mouse);

        printk(KERN_ERR "%s: error when register device\n", DRIVER_NAME);
        return error;
    }
    usb_set_intfdata(interface, mouse);
    printk(KERN_INFO "%s: device is connected\n", DRIVER_NAME);
    return 0;
}
```

На листинге 7 представлена реализация функции *disconnect*.

Листинг 7 – Реализация функции disconnect

```
static void my_mouse_disconnect(struct usb_interface *interface) {
    mouse_t *mouse = usb_get_intfdata(interface);
    usb_set_intfdata(interface, NULL);

    if (mouse) {
        usb_kill_urb(mouse->irq);
        input_unregister_device(mouse->input_dev);
        usb_free_urb(mouse->irq);
        usb_free_coherent(interface_to_usbdev(interface), USB_PACKET_LEN, mouse
           ->data, mouse->data_dma);
        kfree(mouse);

        printk(KERN_INFO "%s: device was disconected\n", DRIVER_NAME);
    }
}
```

На листинге 8 представлена реализация функции *init* для драйвера.

Листинг 8 – Реализация функции init

```
static void my_mouse_irq(struct urb *urb) {
    urb_workqueue_t *container = kzalloc(sizeof(urb_workqueue_t), GFP_KERNEL);
    container->urb = urb;
    INIT_WORK(&container->work, work_irq);
    queue_work(workq, &container->work);
}

static int __init mouse_bright_init(void) {
    int result = usb_register(&my_mouse_driver);
    int i, j, ret;

    int return_val = request_irq(KEYB_IRQ, (irq_handler_t)irq_handler,
       IRQF_SHARED, "interrupt", &tmp);
        if (return_val)
        {
                printk (KERN_DEBUG "request irq failed\n");
                return -1;
        }

    if (result < 0) {
        printk(KERN_ERR "%s: usb register error\n", DRIVER_NAME);
        return result;
    }
    workq = create_workqueue("workqueue_mouse");
    if (workq == NULL) {
        printk(KERN_ERR "%s: allocation workqueue error\n", DRIVER_NAME);
        return -1;
    }

    work_k = create_workqueue("workqueue_keyboard");
    if (workq_k == NULL) {
        printk(KERN_ERR "%s: allocation workqueue error\n", DRIVER_NAME);
        return -1;
    }
    virtual_mouse = input_allocate_device();
    if (virtual_mouse == NULL) {
        printk(KERN_ERR "%s: allocation device error\n", DRIVER_NAME);
```

На листинге 9 представлена реализация функции *exit* для драйвера.

Листинг 9 – Реализация функции exit

```
    }

    INIT_WORK(&work_k, workHandler);
    virtual_mouse->name = "virtual mouse";
    set_bit(EV_REL, virtual_mouse->evbit);
        set_bit(REL_X, virtual_mouse->relbit);
        set_bit(REL_Y, virtual_mouse->relbit);
    set_bit(EV_KEY, virtual_mouse->evbit);
```

3.5 Makefile

На листинге 10 представлена реализация Makefile.

Листинг 10 – Makefile

```
obj-m := brightness_module.o

brightness_module-objs += brightness_control_ops.o
brightness_module-objs += brightness_keyboard_controller.o
brightness_module-objs += kernel_space_fops.o

all default:
        make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules

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

4 Исследовательский раздел

4.1 Демонстрация работы программы

На рисунке 4.1 изображены логи при загрузке драйвера в систему:

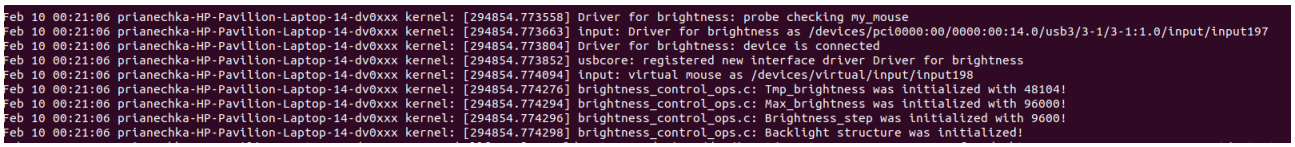

Рисунок 4.1 – Логи при загрузке драйвера в систему

На рисунке 4.2 представлены логи при изменении яркости экрана:

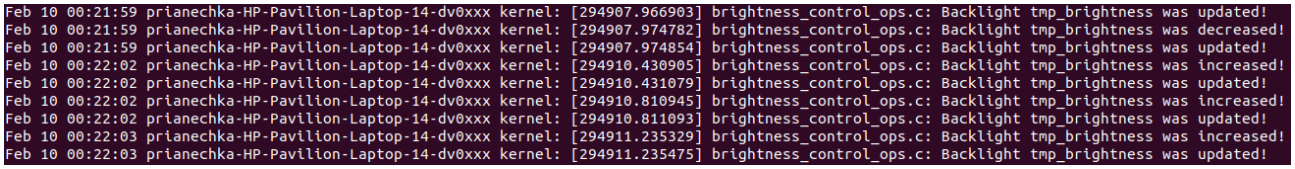

Рисунок 4.2 – Логи при изменении яркости экрана

На рисунке 4.3 представлены логи при выгрузке драйвера из системы:

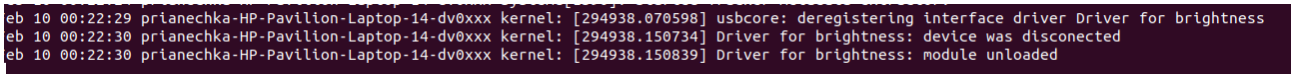

Рисунок 4.3 – Логи при выгрузке драйвера из системы

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было разработано ПО, позволяющее изменять яркость экрана с помощью USB-мыши. Были выполнены следующие задачи:

— проведен анализ существующих подходов к изменению функциональности внешних устройств в Linux;
— проведен анализ существующих способов изменения яркости экрана;
— 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 свободный – (10.02.2023).

2. Ubuntu 20.04.3 LTS (Focal Fossa) [Электронный ресурс]. – Режим доступа: https://releases.ubuntu.com/20.04/ свободный – (10.02.2023).

3. Writing USB Device Drivers. [Электронный ресурс]. – Access mode: https://kernel.readthedocs.io/en/sphinx-samples/writing_usb_driver.html online – (10.02.2023).

4. struct usb_driver. [Электронный ресурс]. – Access mode: https://elixir.bootlin.com/linux/latest/source/include/linux/usb.h#L1188l online – (10.02.2023).

5. struct input_dev. [Электронный ресурс]. – Access mode: https://elixir.bootlin.com/linux/v5.10.1/source/include/linux/input.h#L131 online – (10.02.2023).

6. EV_KEY. [Электронный ресурс]. – Access mode: https://elixir.bootlin.com/linux/latest/source/include/uapi/linux/input-event-codes.h#L39 online – (10.02.2023).

7. input_report_key. [Электронный ресурс]. – Access mode: https://elixir.bootlin.com/linux/latest/source/include/linux/input.h#L415 online – (10.02.2023).

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

9. urb. [Электронный ресурс]. – Access mode: https://docs.kernel.org/driver-api/usb/URB.html online – (10.02.2023).

10. usb_fill_int_urb. [Электронный ресурс]. – Access mode: https://manpages.debian.org/jessie-backports/linux-manual-4.8/usb_fill_int_urb.9.en.html online – (10.02.2023).

11. usb_submit_urb. [Электронный ресурс]. – Access mode: https://elixir.bootlin.com/linux/latest/source/include/linux/usb.h#L1723 online – (10.02.2023).

12. Цилюрик О.И. Модули ядра Linux. — 4 издание. — 2011.

13. workqueue. [Электронный ресурс]. – Access mode: https://docs.kernel.org/core-api/workqueue.html online – (10.02.2023).

ПРИЛОЖЕНИЕ А

В листинге 11 представлен код загружаемого модуля ядра, реализующий драйвер мыши.

Листинг 11 – Загружаемый модуль ядра реализующий драйвер мыши

```
#include "inc/controller.h"
#include "inc/brightness.h"
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/usb/input.h>
#include <linux/slab.h>
#include <linux/workqueue.h>

#define DRIVER_NAME    "Driver for brightness"
#define DRIVER_AUTHOR  "Alexander Pryanishnikov"
#define DRIVER_DESC    "Driver for my mouse"
#define DRIVER_LICENSE "GPL"

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

#define ID_VENDOR_MOUSE 0x046d
#define ID_PRODUCT_MOUSE 0xc542
#define USB_PACKET_LEN  10
#define WHEEL_THRESHOLD 4

struct my_mouse {
    unsigned char     *data;
    dma_addr_t         data_dma;
    struct input_dev  *input_dev;
    struct usb_device *usb_dev;
    struct urb        *irq;
    int                old_wheel_pos;
    char               phys[32];
};

typedef struct my_mouse mouse_t;

struct urb_workqueue {
    struct urb *urb;
    struct work_struct work;
};

typedef struct urb_workqueue urb_workqueue_t;
```

```
static int shift = 0;
static int bright_flag = 0;
static int pressed_key;
static struct workqueue_struct *workq;
static struct input_dev *virtual_mouse;

static void work_irq(struct work_struct *work) {
    urb_workqueue_t *container = container_of(work, urb_workqueue_t, work);
    struct urb *urb;
    int retval;
    mouse_t *mouse;
    unsigned char *data;

    if (container == NULL) {
        printk(KERN_ERR "%s: %s - container is NULL\n", DRIVER_NAME, __func__);
        return;
    }

    urb = container->urb;
    mouse = urb->context;
    data = mouse->data;

    if (urb->status != 0) {
        printk(KERN_ERR "%s: %s - urb status is %d\n", DRIVER_NAME, __func__,
            urb->status);
        kfree(container);
        return;
    }

    switch(data[1]) {
        case 1:
            if (shift)
                bright_flag = 1;
            break;
        case 2:
            if (shift)
                bright_flag = -1;
            break;
        default:
            break;
    }
    backlight_change(bright_flag);
    if (!bright_flag) {
        input_report_key(virtual_mouse, BTN_LEFT, data[1] & 0x0001);
        input_report_key(virtual_mouse, BTN_RIGHT, data[1] & 0x0002);
    }
    bright_flag = 0;
```

```
        input_report_rel(virtual_mouse, REL_X,     data[2] - data[3]);
    input_report_rel(virtual_mouse, REL_Y,     data[4] - data[5]);
        input_report_rel(virtual_mouse, REL_WHEEL, data[6]);

    input_sync(virtual_mouse);

    retval = usb_submit_urb(urb, GFP_ATOMIC);
    if (retval)
        printk(KERN_ERR "%s: %s - usb_submit_urb failed with result %d\n",
            DRIVER_NAME, __func__, retval);

    kfree(container);
}

static void my_mouse_irq(struct urb *urb) {
    urb_workqueue_t *container = kzalloc(sizeof(urb_workqueue_t), GFP_KERNEL);
    container->urb = urb;
    INIT_WORK(&container->work, work_irq);
    queue_work(workq, &container->work);
}

static int my_mouse_open(struct input_dev *dev) {
    mouse_t *mouse = input_get_drvdata(dev);

    mouse->old_wheel_pos = -WHEEL_THRESHOLD - 1;
    mouse->irq->dev = mouse->usb_dev;
    if (usb_submit_urb(mouse->irq, GFP_KERNEL))
        return -EIO;

    return 0;
}

static void my_mouse_close(struct input_dev *dev) {
    mouse_t *mouse = input_get_drvdata(dev);
    usb_kill_urb(mouse->irq);
}

static int my_mouse_probe(struct usb_interface *interface, const struct
   usb_device_id *id) {
    struct usb_device *usb_device = interface_to_usbdev(interface);
    mouse_t *mouse;
    struct input_dev *input_dev;
    struct usb_endpoint_descriptor *endpoint;
    int error = -ENOMEM;

    printk(KERN_INFO "%s: probe checking my_mouse\n", DRIVER_NAME);
```

```
mouse = kzalloc(sizeof(mouse_t), GFP_KERNEL);
input_dev = input_allocate_device();
if (!mouse || !input_dev) {
    input_free_device(input_dev);
    kfree(mouse);

    printk(KERN_ERR "%s: error when allocate device\n", DRIVER_NAME);
    return error;
}

mouse->data = (unsigned char *)usb_alloc_coherent(usb_device, USB_PACKET_LEN
    , GFP_KERNEL, &mouse->data_dma);
if (!mouse->data) {
    input_free_device(input_dev);
    kfree(mouse);

    printk(KERN_ERR "%s: error when allocate coherent\n", DRIVER_NAME);
    return error;
}

mouse->irq = usb_alloc_urb(0, GFP_KERNEL);
if (!mouse->irq) {
    usb_free_coherent(usb_device, USB_PACKET_LEN, mouse->data, mouse->
        data_dma);
    input_free_device(input_dev);
    kfree(mouse);

    printk(KERN_ERR "%s: error when allocate urb\n", DRIVER_NAME);
    return error;
}

mouse->usb_dev = usb_device;
mouse->input_dev = input_dev;
usb_make_path(usb_device, mouse->phys, sizeof(mouse->phys));
strlcat(mouse->phys, "/input0", sizeof(mouse->phys));
input_dev->name = DRIVER_NAME;
input_dev->phys = mouse->phys;
usb_to_input_id(usb_device, &input_dev->id);
input_dev->dev.parent = &interface->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 = my_mouse_open;
    input_dev->close = my_mouse_close;
    endpoint = &interface->cur_altsetting->endpoint[0].desc;
    usb_fill_int_urb(
        mouse->irq, usb_device,
        usb_rcvintpipe(usb_device, endpoint->bEndpointAddress),
        mouse->data, USB_PACKET_LEN,
        my_mouse_irq, mouse, endpoint->bInterval
    );
    usb_submit_urb(mouse->irq, GFP_ATOMIC);
    mouse->irq->transfer_dma = mouse->data_dma;
    mouse->irq->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;
    error = input_register_device(mouse->input_dev);
    if (error) {
        usb_free_urb(mouse->irq);
        usb_free_coherent(usb_device, USB_PACKET_LEN, mouse->data, mouse->
            data_dma);
        input_free_device(input_dev);
        kfree(mouse);

        printk(KERN_ERR "%s: error when register device\n", DRIVER_NAME);
        return error;
    }
    usb_set_intfdata(interface, mouse);
    printk(KERN_INFO "%s: device is connected\n", DRIVER_NAME);
    return 0;
}

static void my_mouse_disconnect(struct usb_interface *interface) {
    mouse_t *mouse = usb_get_intfdata(interface);
    usb_set_intfdata(interface, NULL);

    if (mouse) {
        usb_kill_urb(mouse->irq);
        input_unregister_device(mouse->input_dev);
        usb_free_urb(mouse->irq);
        usb_free_coherent(interface_to_usbdev(interface), USB_PACKET_LEN, mouse
            ->data, mouse->data_dma);
        kfree(mouse);

        printk(KERN_INFO "%s: device was disconected\n", DRIVER_NAME);
    }
}

static struct usb_device_id mouse_table [] = {
    { USB_DEVICE(ID_VENDOR_MOUSE, ID_PRODUCT_MOUSE) },
    { },
};
```

```
MODULE_DEVICE_TABLE(usb, mouse_table);

static struct usb_driver my_mouse_driver = {
    .name       = DRIVER_NAME,
    .probe      = my_mouse_probe,
    .disconnect = my_mouse_disconnect,
    .id_table   = mouse_table,
};

void workHandler(struct work_struct *work)
{
        scancode = inb(KBD_DATA_REG);
    switch (kb_data.scancode) {
                case SHIFT_PR: shift = 1; break;
        case SHIFT_REL: shift = 0; break;
                default:
                        break;
    }
        return;
}

irqreturn_t irq_handler(int irq, void *dev_id)
{
        if (irq == KEYB_IRQ)
        {
                queue_work(workqueue, &workHandler);
                return IRQ_HANDLED;
        }
        else
                return IRQ_NONE;
}

static void my_mouse_irq(struct urb *urb) {
    urb_workqueue_t *container = kzalloc(sizeof(urb_workqueue_t), GFP_KERNEL);
    container->urb = urb;
    INIT_WORK(&container->work, work_irq);
    queue_work(workq, &container->work);
}

static int __init mouse_bright_init(void) {
    int result = usb_register(&my_mouse_driver);
    int i, j, ret;

    int return_val = request_irq(KEYB_IRQ, (irq_handler_t)irq_handler,
       IRQF_SHARED, "interrupt", &tmp);
        if (return_val)
        {
```

```
                printk (KERN_DEBUG "request irq failed\n");
                return -1;
        }

    if (result < 0) {
        printk(KERN_ERR "%s: usb register error\n", DRIVER_NAME);
        return result;
    }
    workq = create_workqueue("workqueue_mouse");
    if (workq == NULL) {
        printk(KERN_ERR "%s: allocation workqueue error\n", DRIVER_NAME);
        return -1;
    }

    work_k = create_workqueue("workqueue_keyboard");
    if (workq_k == NULL) {
        printk(KERN_ERR "%s: allocation workqueue error\n", DRIVER_NAME);
        return -1;
    }
    virtual_mouse = input_allocate_device();
    if (virtual_mouse == NULL) {
        printk(KERN_ERR "%s: allocation device error\n", DRIVER_NAME);
        return -1;
    }

    INIT_WORK(&work_k, workHandler);
    virtual_mouse->name = "virtual mouse";
    set_bit(EV_REL, virtual_mouse->evbit);
        set_bit(REL_X, virtual_mouse->relbit);
        set_bit(REL_Y, virtual_mouse->relbit);
    set_bit(EV_KEY, virtual_mouse->evbit);
    set_bit(BTN_LEFT, virtual_mouse->keybit);
        set_bit(BTN_RIGHT, virtual_mouse->keybit);

    result = input_register_device(virtual_mouse);
    if (result != 0) {
        printk(KERN_ERR "%s: registration device error\n", DRIVER_NAME);
        return result;
    }

        backlight_init();
    printk(KERN_INFO "brightness_controller.c: initialization complete.");
    return ret;
}

static void __exit mouse_bright_exit(void) {
    flush_workqueue(workq);
    destroy_workqueue(workq);
```

```
    input_unregister_device(virtual_mouse);
    usb_deregister(&my_mouse_driver);
    free_irq(KB_IRQ, &data);
    printk(KERN_INFO "%s: module unloaded\n", DRIVER_NAME);
}

module_init(mouse_bright_init);
module_exit(mouse_bright_exit);
```

ПРИЛОЖЕНИЕ Б

В листинге 12 представлена реализация модуля, использующегося для регулировки яркости экрана.

Листинг 12 – Реализация функции probe

```
#include "inc/brightness.h"

int read_light_value(struct file *file){

        int cur_brightness = 0;
        file_read((struct file *) file, 0, brightness_str, MAX_BUFFER_LEN);
        brightness_str[MAX_BUFFER_LEN-1] = '\0';
        kstrtoint(brightness_str, 10, &cur_brightness);
        memset(brightness_str, 0, sizeof(brightness_str));

        return cur_brightness;
}

void backlight_init(void) {
        data = kmalloc(sizeof(struct backlight_data), GFP_KERNEL);
        memset(brightness_str, 0, sizeof(brightness_str));

        backlight_file = file_open(BACKLIGHT_PATH, O_RDWR, 0);
        data->cur = backlight_read_value(backlight_file);
        printk(KERN_INFO "brightness.c: Tmp_brightness was initialized with %d!\
            n", data->tmp_brightness);

        struct file *max_backlight_file = file_open(MAX_BACKLIGHT_PATH, O_RDONLY
            , 0);
        data->max = backlight_read_value(max_backlight_file);
        file_close((struct file *) max_backlight_file);
        printk(KERN_INFO "brightness.c: Max_brightness was initialized with %d!\
            n", data->max_brightness);
        data->step = data->max / STEP;
        printk(KERN_INFO "brightness.c: Backlight structure was initialized!\n")
            ;
}

void brightness_update(void){
        snprintf(brightness_str, MAX_BRINGHTNESS_CLASS, "%d", data->
            tmp_brightness);
        file_write(backlight_file, 0, brightness_str, MAX_BRINGHTNESS_CLASS);
        memset(brightness_str, 0, sizeof(brightness_str));
        printk(KERN_INFO "brightness.c: Backlight tmp_brightness was updated!\n"
            );
}
```

```
void backlight_change(int flag) {

        if (flag == 0)
                return;

        if (flag == 1) {
                data->cur += data->step;
                if (data->cur  > data->max)
                        data->cur = data->max;
                printk(KERN_INFO "brightness.c: Backlight tmp_brightness was
                    increased!\n");
        }
        else if (flag == -1) {
                data->cur -= data->step;
                if (data->cur <= 8)
                        data->cur = 8;
                printk(KERN_INFO "brightness.c: Backlight tmp_brightness was
                    decreased!\n");
        }
        brightness_update();
}


void brightness_exit(void){
        kfree(data);
        printk(KERN_INFO "brightness.c: Backlight structure was uninitialized!\n
            ");
}
```

ПРИЛОЖЕНИЕ В

В листинге 13 представлен код вспомогательного модуля для работы с файлами из пространства ядра.

Листинг 13 – Вспомогательный модуль для работы с файлами из пространства ядра

```
#include "file_ops.h"
#include <asm/uaccess.h>
#include <asm/segment.h>


struct file *file_open(const char *path, int flags, int rights) {
        struct file *fp = NULL;

        fp = filp_open(path, flags, rights);
        int error = 0;

        if(IS_ERR(fp)){
                error = PTR_ERR(fp);
                printk("tmp brightness: %d", error);
                return NULL;
        }

        return fp;
}


ssize_t file_read(struct file *fp, unsigned long long offset, unsigned char *
    data, unsigned int size) {
     return kernel_read(fp, data, size, &offset);
}


ssize_t file_write(struct file *fp, unsigned long long offset, unsigned char *
    data, unsigned int size) {
        return kernel_write(fp, data, size, &offset);
}


void file_close(struct file *file) {
     filp_close(file, NULL);
}
```