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:

«Linux Драйвер контроллера Dualshock 4 v2 для управления курсором мыши и ввода ограниченного текста»

Студент ИУ7-72Б _____ _____ Е.С. Тимонин
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ А.А. Павельев
(Подпись, дата) (И.О.Фамилия)

Москва, 2020 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 __ИУ7____
(Индекс)
_____ ___И.В._Рудаков____
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине _____Операционные системы_____

Студент группы ____ИУ7-72Б_____

_____Тимонин Егор Сергеевич_____
(Фамилия, имя, отчество)

Тема курсового проекта _____
_____Linux Драйвер контроллера Dualshock 4 v2 для управления курсором мыши и ввода ограниченного текста_____
_____

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 программный комплекс, позволяющий осуществить управление курсором мыши и ввод ограниченного текста в ОС Linux с помощью контроллера Sony Dualshock 4 v2. Нажатие на кнопки контроллера генерирует следующие события: нажатие клавиш, перемещение курсора. Разрабатываемый комплекс должен обрабатывать эмуляцию этих событий._____
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на _20-30_ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
На защиту проекта должна быть предоставлена презентация, состоящая из 15-20 слайдов.___
На слайдах должны быть отражены: постановка задачи, использованные методы и_____
алгоритмы, расчетные соотношения, структура комплекса программ, интерфейс,_____

Дата выдачи задания «» _____ 2020г.

Руководитель курсового проекта _____ ___А.А. Павельев _____
(Подпись, дата) (И.О.Фамилия)

Студент _____ ___Е.С. Тимонин_____
(Подпись, дата) (И.О.Фамилия)

Введение

USB (англ. Universal Serial Bus) — последовательный интерфейс для подключения периферийных устройств к вычислительной технике. Для того, чтобы работать с USB-устройствами под Linux должна быть включена поддержка соответствующих устройств. Например, для клавиатуры, мыши, джойстика, для интерактивных устройств должна быть включена поддержка USB (англ. Human Interface Device). Зачастую реализация протокола HID для таких сложных устройств, как игровой контроллер, не предусматривает реализацию поведения для всех компонентов устройства, чтобы обеспечить каждый компонент устройства поведением, необходимо написать собственный HID-драйвер.

Целью данного курсового проекта является разработка HID-драйвера контроллера Sony Dualshock 4 v2, для управления курсором мыши и ввода ограниченного текста.

1 Аналитический раздел

В данном разделе будет проанализированы варианты реализация драйвера, рассмотрены особенности выбранного варианта реализации.

В соответствии с заданием на курсовую работу необходимо разработать драйвер контроллера Sony Dualshock 4 v2, чтобы с его помощью можно было управлять курсором мыши и вводить ограниченный текст.

1.1 Выбор типа драйвера

Sony DualShock 4 v2 подключается через USB и принадлежит классу HID, то это задание можно сделать двумя способами либо написать USB драйвер, либо написать HID драйвер.

Написание USB драйвера потенциально является гибким подходом, так позволяет взаимодействовать с устройством на низком уровне, но при этом придется заново реализовывать протокол класса HID и взаимодействие с подсистемой Input.

В подсистеме HID реализовано взаимодействие с устройством и с подсистемой Input, поэтому в данном курсовом проекте была выбрана реализация HID драйвера.

1.2 HID

Класс устройств HID позволяет современным операционным системам единообразно обрабатывать все интерактивные устройства и не писать новый драйвер для каждого нового устройства. Информация о USB-устройстве хранится в сегментах его ПЗУ (только для чтения). Эти сегменты называются дескрипторами. Дескриптор интерфейса может идентифицировать устройство как принадлежащее к одному из конечного числа классов. Класс HID является основным объектом внимания этого документа.

Устройство класса USB/HID использует соответствующий драйвер класса HID для извлечения и маршрутизации всех данных.

Маршрутизация и извлечение данных осуществляется путем изучения дескрипторов устройства и данных, которые оно предоставляет.

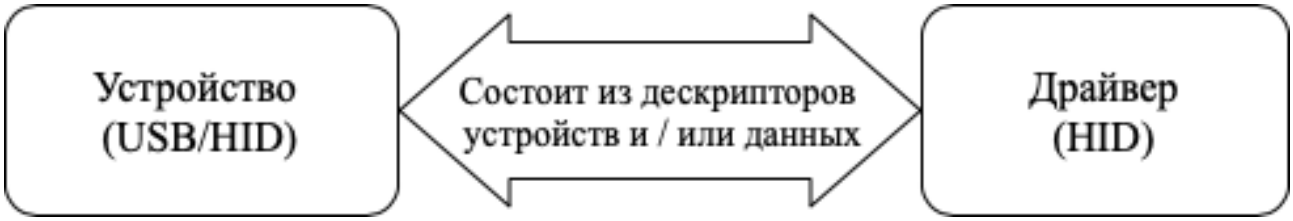

Рисунок 1.1 — Маршрутизация данных

Дескриптор устройства класса HID определяет, какие другие дескрипторы класса HID присутствуют, и указывает их размеры. Например, отчет и физические дескрипторы.

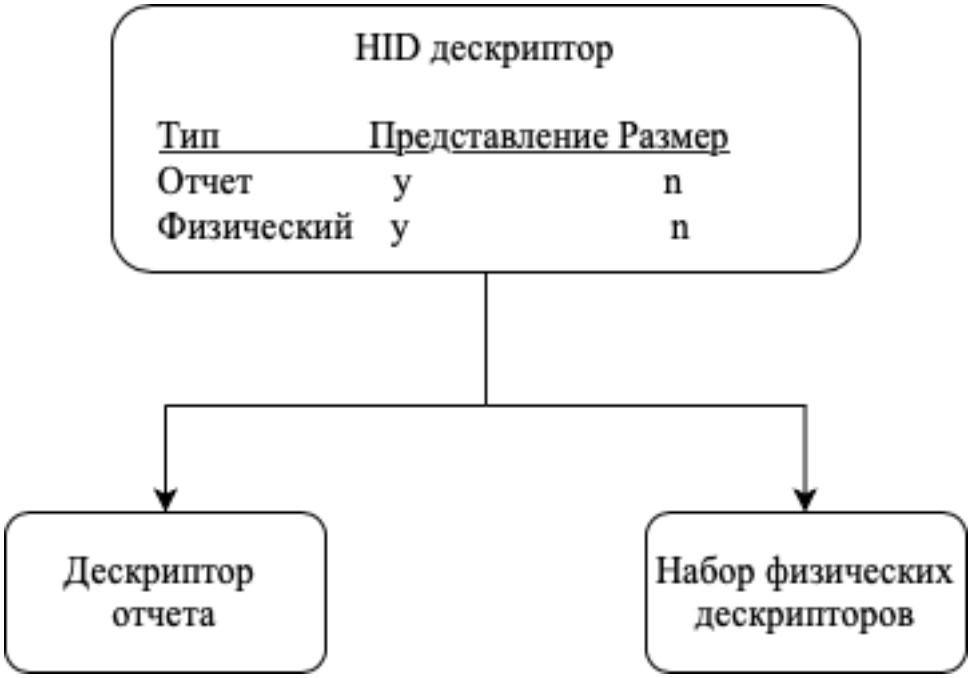

Рисунок 1.2 — Дескриптор устройства

Дескриптор отчета описывает каждый фрагмент данных, генерируемых устройством, и то, что эти данные фактически измеряют.

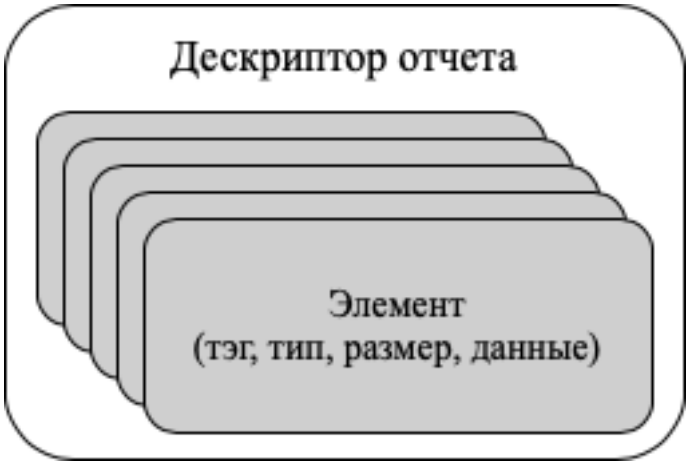

Рисунок 1.3 — Дескриптор отчета

Например, дескриптор отчета определяет элементы, описывающие положение или состояние кнопки. Информация о элементе используется для:

- Определения, куда направлять входные данные—например, отправьте входные данные в API мыши или джойстика.
- Того, чтобы разрешить программному обеспечению назначать функциональность входным данным—например, использовать джойстик для позиционирования чего-либо.

Исследуя элементы (совместно называемые дескриптором отчета), драйвер класса HID может определить размер и состав отчетов данных с устройства класса HID.

Дескрипторы отчетов состоят из фрагментов информации. Каждая часть информации называется элементом.

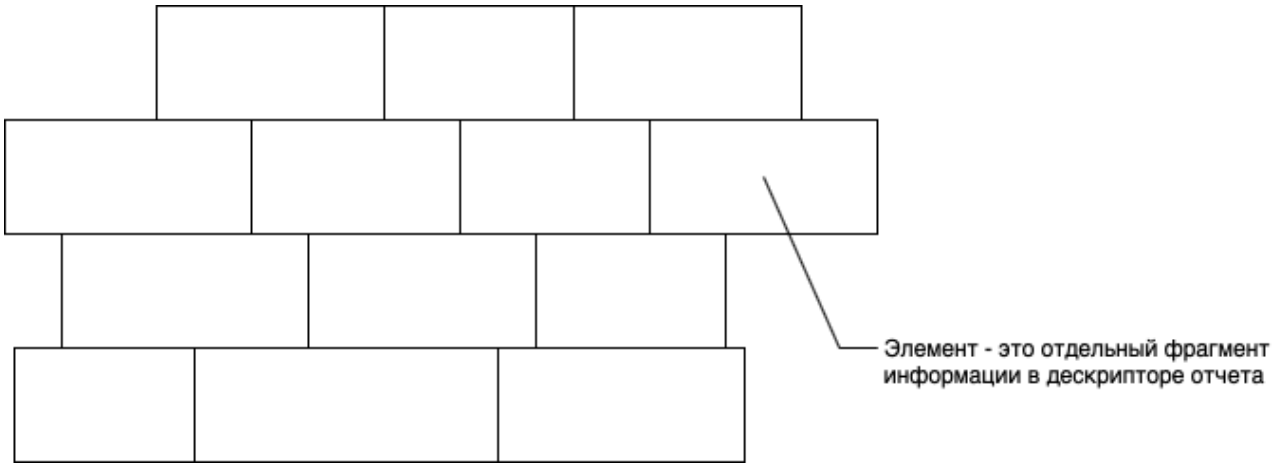

Рисунок 1.4 — Дескриптор отчета

Элемент — это часть информации об устройстве. Все элементы имеют однобайтовый префикс, содержащий тег элемента, тип элемента и размер элемента.

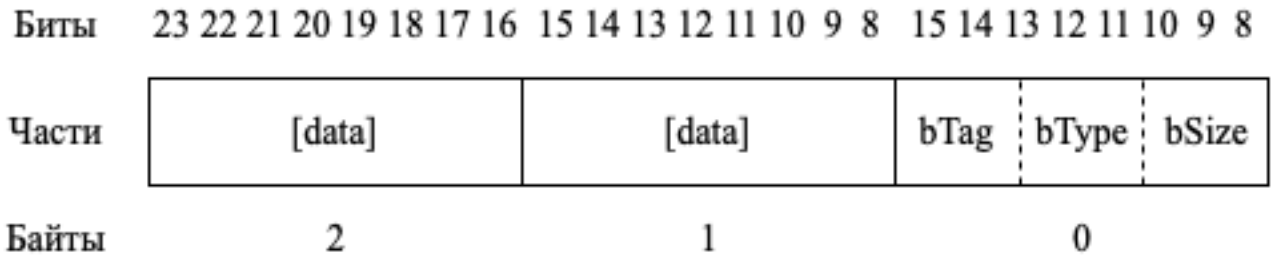

Рисунок 1.5 — Структура элемента дескриптора отчета

Дескриптор отчета состоит из элементов, которые предоставляют информацию об устройстве и описание данных.

Таким образом с помощью дескриптора отчета можно определить, какое устройство подключено и как оно передает данные.

1.3 Подсистема HID в Linux

Класс HID не зависит от механизма передачи данных. Ее основу составляет модуль hid, основные структуры данных которого находятся в файле /linux/hid.h, а код модуля расположен в файлах /drivers/hid/hid-core.c/ и /drivers/ hid/hid-input.c.

За USB взаимодействие отвечает модуль usbhid, код которого находится в файле /drivers/hid/usbhid/hid-core.c. Этот модуль является USB драйвером для USB HID устройств и все действия с дескрипторами отчетов и самими отчетами он перенаправляет модулю hid.

Драйвер класса HID содержит парсер, используемый для анализа элементов, найденных в дескрипторе отчета. Парсер извлекает информацию из дескриптора линейным способом. Парсер собирает состояние каждого известного элемента по мере прохождения через дескриптор и сохраняет их в таблице состояний элемента. Затем регистрирует устройство в системе и перенаправляет данные от устройства другим подсистемам.

Если стандартный драйвер устройства не подходит, то можно написать собственный HID драйвер и переопределить его стандартное поведение.

1.4 Драйвер для Dualchock 4 v2 в Linux

Для контроллеров Sony в операционной системе Linux уже есть драйвер hid-sony.c, код которого находится в /drivers/hid/hid-sony.c. В нем присутствует поддержка всех элементов, которые находятся в контроллерах (акселерометры, светодиоды и т. д.). Но поведение драйвера Dualshock 4 v2 не предусматривает взаимодействие с кнопками, можно взаимодействовать только с сенсорной панелью. Поэтому необходимо написать собственный драйвер, который определит поведение для кнопок контроллера.

2 Конструкторский раздел

В данном разделе рассмотрена структура драйвера в Linux и основные этапы его разработки.

2.1 Драйвер USB HID устройства

Драйвер ОС Linux представляет собой загружаемый модуль ядра, который определяет точки входа для работы с устройством. Чтобы загрузить собственный HID драйвер, необходимо заполнить структуру hid_driver и зарегистрироваться в подсистеме hid.

Листинг 2.1 — Структура hid_driver

```
static struct hid_driver ds4v2_driver =
{
    .name = "hid_ds4",
    .id_table = ds4v2_table,
    .probe = ds4v2_probe,
    .remove = ds4v2_remove,
    .raw_event = ds4v2_raw_event,
};
```

Где .name — отображаемое имя драйвера.

.id_table — массив структур hid_device_id, где структура hid_device_id описывает информацию относительно ID устройств поддерживаемых драйвером hid и передается в макрос MODULE_DEVICE_TABLE.

.probe — callback-функция, которая вызывается при подключении устройства.

.remove - callback-функция, которая вызывается при отключении устройства.

.raw_event - callback-функция, которая вызывается при получении отчета от устройства.

Листинг 2.2 — Регистрация в подсистеме hid

```
hid_register_driver(&ds4_driver);
```

Аналогичная функция используется, когда необходимо удалить собственный драйвер из hid-подсистемы.

Данная часть модуля используется для того, чтобы принимать отчеты от устройства.

2.2 Символьный драйвер

Чтобы загрузить собственный символьный драйвер, необходимо заполнить структуру file_operations и зарегистрироваться в таблице ядра.

Листинг 2.3 — Регистрация в таблице ядра

```
static struct file_operations simple_driver_fops =
{
    .owner = THIS_MODULE,
    .read  = device_file_read,
};
```

Где .owner — указывает на модуль, который владеет данной структурой, поле используется ядром для получения счетчика использования модуля.

.read — используется для копирования данных.

Листинг 2.4 — Регистрация в таблице ядра

```
RES = register_chrdev( 0, device_name, &simple_driver_fops );
```

Аналогичная функция используется, когда необходимо удалить драйвер из таблицы ядра.

Данная часть модуля используется для того, чтобы записывать данные, полученные от контроллера, в пространство пользователя.

2.3 ПО в пространстве пользователя

ПО считывает данные о нажатой кнопке с файла, в который символьный драйвер записывает коды от устройства. Полученные коды используются для того, чтобы с использованием xdotool произвести действие с клавиатурой или мышью, в зависимости от того, какое действие мы назначили коду.

Листинг 2.5 — Коды кнопок контроллера

```
#define UP_BTN        0x0
#define RIGHT_BTN     0x2
#define DOWN_BTN      0x4
#define LEFT_BTN      0x6
#define SQUARE_BTN    0x18
#define X_BTN         0x28
#define CIRCLE_BTN    0x48
#define TRIANGLE_BTN 0x88

#define L1_BTN        0x1
#define R1_BTN        0x2
#define L2_BTN        0x4
#define R2_BTN        0x8
#define SHARE_BTN     0x10
#define OPTIONS_BTN   0x20
#define L3_BTN        0x40
#define R3_BTN        0x80
```

В итоге взаимодействие драйвера и пользовательского приложения можно описать следующим образом: при нажатии на кнопки контроллера приходят отчеты с кодами в наш HID драйвер, который обрабатывает их так, чтобы символьный драйвер мог записать данные в пространство пользователя, где пользовательское приложение интерпретирует коды из отчетов и выполняет соответсвующее действие.

3 Технологический раздел

В данном разделе будут описаны инструменты, использованные в ходе работы.

3.1 Выбор языка программирования и среды разработки

В операционной системе Linux модули ядра пишутся на языке C и компилируются с помощью компилятора GCC. Сборка модуля драйвера производится автоматически с помощью утилиты make.

Листинг 3.1 — Makefile для модуля ядра

```
obj-m += ds4v2.o

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

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

3.2 Регистрация HID-драйвера

Чтобы зарегистрировать драйвер в подсистеме hid, необходимо сначала заполнить массив hid_device_id, в котором описываем информацию об устройстве, чтобы система выбрала правильное USB-устройство.

Листинг 3.2 — Заполнение массива hid_device_id

```
#define VENDOR_SONY_ID      0x054c
#define DS4_ID              0x09cc

static struct hid_device_id ds4v2_table[] =
{
    { HID_USB_DEVICE(VENDOR_SONY_ID, DS4V2_ID) },
    {}
};
```

```
MODULE_DEVICE_TABLE (hid, ds4v2_table);
```

Затем инициализируем структуру hid_driver.

Листинг 3.3 — Инициализация структуры hid_driver

```
static struct hid_driver ds4v2_driver =
{
    .name = "hid_ds4",
    .id_table = ds4v2_table,
    .probe = ds4v2_probe,
    .remove = ds4v2_remove,
    .raw_event = ds4v2_raw_event,
};
```

Регистрация драйвера в подсистеме hid происходит при инициализации модуля ядра.

Листинг 3.4 — Регистрация драйвера в подсистеме hid

```
static int __init ds4v2_init(void)
{
    return hid_register_driver(&ds4_driver);
}
```

Удаление драйвера из подсистемы hid происходит во время выгрузки модуля из ядра.

Листинг 3.5 — Удаление драйвера из подсистемы hid

```
static void __exit ds4v2_exit(void)
{
    hid_unregister_driver(&ds4_driver);
}
```

3.3 Регистрация символьного драйвера

Чтобы зарегистрировать символьный драйвер в таблице ядра необходимо заполнить структуру file_operations.

Листинг 3.6 — Инициализация структуры file_operations

```
static struct file_operations simple_driver_fops =
{
    .owner = THIS_MODULE,
    .read  = device_file_read,
};
```

Регистрация символьного устройства происходит при инициализации модуля ядра, для этого была написана отдельная функция, которая обрабатывает регистрацию.

Листинг 3.7 — Регистрация символьного драйвера в таблице ядра

```
int register_character_device(void) {
    int RES = 0;

    RES = register_chrdev( 0, device_name,
&simple_driver_fops );
    if( RES < 0 ) {
        printk( KERN_WARNING "DS4V2:  fail register character
device ERRCODE = %i", RES );
        return RES;
    }

    device_file_major_number = RES;
    printk( KERN_NOTICE "DS4V2: register character device
MAJOR_NUMBER = %i MINOR_NUMBER 0...255",
device_file_major_number );

    return 0;
}
```

Удаление символьного устройства из таблицы ядра происходит во время выгрузки модуля из ядра.

Листинг 3.8 — Удаление символьного драйвера из таблицы ядра

```
void unregister_character_device(void)
{
    if(device_file_major_number != 0)
    {
        unregister_chrdev(device_file_major_number,
device_name);
    }
}
```

3.4 Структура ds4_data

Чтобы использовать HID драйвер выделяется отдельная структура, которая будет инициализироваться при подключении устройства, и удаляться при его отключении. В ней находится указатель на hid_device, а так же дополнительная информация об устройстве.

Листинг 3.8 — Структура ds4_data

```
struct ds4v2_data {
    struct hid_device *hdev;
    char *name;
};
```

3.5 Функция probe и remove

Функция probe активируется при подключении устройства к системе, в ней выделяется память под структуру ds4_data с помощью функции devm_kzalloc и происходит инициализация этой структуры, с помощью hid_parse посылается запрос дескриптора отчетов устройства, а функция hid_hw_start инициализирует получение отчетов от устройства.

Функция remove активируется, когда устройство было отключено от системы, освобождение памяти автоматически, так как была использована

функция devm_kzalloc, чтобы устройство перестало посылать отчеты вызывается функция hid_hw_stop.

3.6 Функция raw_event

Функция raw_event активируется, когда контроллер получает необработанное событие, в ней происходит передача данных в символьный драйвер.

3.7 Загрузка модуля ядра

Так как для контроллера по умолчанию устанавливается драйвер hid_sony, перед загрузкой нашего драйвер, необходимо выгрузить из ядра конкурирующий драйвер. Для этого необходимо выполнить следующую команду:

```
sudo rmmod hid_sony
```

Стандартный драйвер выгружен, теперь нужно скомпилировать модуль ядра с помощью утилиты make.

Дальше необходимо загрузить драйвер в пространство ядра с помощью следующей команды:

```
sudo insmod hid-ds4.ko
```

Чтобы обеспечить взаимодействие между драйвером и пользовательской программой, необходимо создать символьный файл, куда будут записаны отчеты от устройства и откуда будет читать информацию пользовательская программа. Это может быть реализована с помощью следующей команды:

```
mknod /dev/ds4-device c echo "cat /proc/devices | grep DS4" |
cut -d" " -f1 0
```

Заключение

В ходе курсового проекта был написан драйвер для устройства Dualshock 4 v2 , с помощью которого управлять курсором мыши и вводить ограниченный текст.

Был изучен стандарт HID, написан HID драйвер для получения отчетов от устройства, символьный драйвер для обработки отчетов, пользовательская программа для интерпретации кодов из отчетов.

Список литературы

[1] Linux Device Drivers, Third Edition - Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman

[2] Human Interface Devices (HID) Information [Электронный ресурс. - Режим досутпа — https://www.usb.org/hid]

[3] Разработка драйверов для USB-устройств под Linux [Электронный ресурс. - Режим досутпа — https://www.opennet.ru/base/dev/write_linux_driver.txt.html]

Приложение А Код загружаемого модуля

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/usb.h>
#include <linux/hid.h>
#include <linux/input.h>
#include <linux/sysfs.h>
#include <asm/uaccess.h>

#define VENDOR_SONY_ID   0x054c
#define DS4V2_ID    0x09cc
#define DS4V2_NAME "Dualshock 4 v2"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Timonin Egor");
MODULE_DESCRIPTION("DualShock 4 v2 driver");

struct ds4v2_data {
    struct hid_device *hdev;
    struct input_dev *input_dev;

    char *name;
};

static struct attribute *ds4v2_attrs[] = {
    NULL,
};

static struct attribute_group ds4v2_attr_group = {
    .attrs = ds4v2_attrs,
};

u8 *raw_input = NULL;
static ssize_t raw_input_size = sizeof(raw_input) * 2;

static int device_file_major_number = 0;
```

```
static const char device_name[] = "DS4V2";

static ssize_t device_file_read(struct file *file_ptr, char
__user *user_buffer, size_t count, loff_t *position) {
    if( *position >= raw_input_size )
        return 0;

    if( *position + count > raw_input_size )
        count = raw_input_size - *position;

    if( raw_copy_to_user(user_buffer, raw_input + *position,
count) != 0 ) {
        printk(KERN_WARNING "DS4V2: fail raw_copy_to_user()");
        return -EFAULT;
    }

    *position += count;
    return count;
}

static struct file_operations simple_driver_fops = {
    .owner     = THIS_MODULE,
    .read  = device_file_read,
};

int register_character_device(void) {
    int RES = 0;

    RES = register_chrdev( 0, device_name,
&simple_driver_fops );
    if( RES < 0 ) {
        printk( KERN_WARNING "DS4V2:  fail register character
device ERRCODE = %i", RES );
        return RES;
    }

    device_file_major_number = RES;
```

```
    printk( KERN_NOTICE "DS4V2: register character device
MAJOR_NUMBER = %i MINOR_NUMBER 0...255",
device_file_major_number );

    return 0;
}

void unregister_character_device(void) {
    if(device_file_major_number != 0) {
        unregister_chrdev(device_file_major_number,
device_name);
    }
}

static int ds4v2_probe(struct hid_device *hdev, const struct
hid_device_id *id) {
    struct ds4v2_data *data;
    struct usb_interface *intf;
    struct usb_device *usbdev;
    int ERR;

    intf = to_usb_interface(hdev->dev.parent);
    usbdev = interface_to_usbdev(intf);

    data = kzalloc(sizeof(struct ds4v2_data), GFP_KERNEL);
    if (data == NULL) {
        dev_err(&hdev->dev, "fail allocate\n");
        ERR = -ENOMEM;
        goto err_no_cleanup;
    }

    data->hdev = hdev;
    hid_set_drvdata(hdev,data);

    printk(KERN_NOTICE "DS4V2: plugged\n");

    ERR = hid_parse(hdev);
    if(ERR) {
```

```
        dev_err(&hdev->dev, DS4V2_NAME " fail device parse\n");
        ERR = -EINVAL;
        goto err_cleanup_data;
    }

    ERR = hid_hw_start(hdev, HID_CONNECT_DEFAULT |
HID_CONNECT_HIDINPUT_FORCE);
    if(ERR) {
        dev_err(&hdev->dev, DS4V2_NAME " fail hardware
start\n");
    }

    ERR = hdev->ll_driver->open(hdev);
    if(ERR) {
        dev_err(&hdev->dev, DS4V2_NAME " failed open input
interrupt pipe\n");
        ERR = -EINVAL;
        goto err_cleanup_data;
    }

    data->input_dev = input_allocate_device();
    if (data->input_dev == NULL) {
        dev_err(&hdev->dev, DS4V2_NAME " fail initialize input
device");
        ERR = -ENOMEM;
        goto err_cleanup_data;
    }

    input_set_drvdata(data->input_dev, hdev);
    ERR = input_register_device(data->input_dev);
    if (ERR) {
        dev_err(&hdev->dev, DS4V2_NAME " fail register input
device");
        ERR = -EINVAL;
        goto err_cleanup_input_dev;
    }

    sysfs_create_group(&(hdev->dev.kobj), &ds4v2_attr_group);
    return 0;
```

```
    err_cleanup_input_dev:
        input_free_device(data->input_dev);
    err_cleanup_data:
        kfree(data);
    err_no_cleanup:
        hid_set_drvdata(hdev, NULL);
    return ERR;
}

static void ds4v2_remove(struct hid_device *hdev) {
    struct ds4v2_data *data;
    hdev->ll_driver->close(hdev);
    hid_hw_stop(hdev);
    sysfs_remove_group(&(hdev->dev.kobj), &ds4v2_attr_group);
    data = hid_get_drvdata(hdev);
    input_unregister_device(data->input_dev);
    kfree(data->name);
    kfree(data);
    printk(KERN_INFO "DS4V2: unplugged\n");
}

static int ds4v2_raw_event(struct hid_device *hdev, struct
hid_report *report, u8 *raw_data, int size) {
    raw_input = raw_data;
    return 0;
}

static struct hid_device_id ds4v2_table[] = {
    { HID_USB_DEVICE(VENDOR_SONY_ID, DS4V2_ID) },
    {}
};

MODULE_DEVICE_TABLE (hid, ds4v2_table);

static struct hid_driver ds4v2_driver = {
    .name = "hid_ds4",
```

```
    .id_table = ds4v2_table,
    .probe = ds4v2_probe,
    .remove = ds4v2_remove,
    .raw_event = ds4v2_raw_event,
};

static int __init ds4v2_init(void) {
    register_character_device();
    return hid_register_driver(&ds4v2_driver);
}

static void __exit ds4v2_exit(void) {
    unregister_character_device();
    hid_unregister_driver(&ds4v2_driver);
}

module_init(ds4v2_init);
module_exit(ds4v2_exit);
```