Санкт-Петербургский политехнический университет Петра Великого
Кафедра компьютерных систем и программных технологий

Отчет по дисциплине
«Проектирование ОС и их компонентов»

Разработка драйверов под под Linux

Работу выполнил студент группы №: 13541/3 Чеботарёв М. М.
Работу принял преподаватель: _____ Душутина Е. В.

Санкт-Петербург
2017 г.

Используемая система и версия ядра

а) Windows

Процессор:	Intel(R) Core(TM) i5-2450 CPU @2.50GHz 2.50GHz
ОЗУ:	8,00 Гб
Тип системы:	Windows 7 Ultimate Compact (2009) Service Pack 1. x64.

б) Linux

```
michael@michael-LIFEBOOK-AH531:~$ lsb_release  -a
No LSB modules are available.
Distributor ID:         Ubuntu
Description:            Ubuntu 16.04.1 LTS
Release:                16.04
Codename:               xenial


michael@michael-LIFEBOOK-AH531:~$ cat /proc/version
Linux version 4.4.0-38-generic (buildd@lgw01-58) (gcc version 5.4.0 20160609
(Ubuntu 5.4.0-6ubuntu1~16.04.2) ) #57-Ubuntu SMP Tue Sep 6 15:42:33 UTC 2016
```

1. Драйвер символьного устройства

Драйвер (driver) — компьютерное программное обеспечение, с помощью которого другое программное обеспечение (операционная система) получает доступ к аппаратному обеспечению некоторого устройства. Обычно с операционными системами поставляются драйверы для ключевых компонентов аппаратного обеспечения, без которых система не сможет работать. Однако для некоторых устройств (таких, как видеокарта или принтер) могут потребоваться специальные драйверы, обычно предоставляемые производителем устройства.

Листинг 1.1. symbolDriver.c

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <asm/uaccess.h>

MODULE_LICENSE( "GPL" );
MODULE_AUTHOR( "Chebotarev Michael" );

#define SUCCESS 0
#define DEVICE_NAME "symbolDevice"

static int device_open( struct inode *, struct file * );
static int device_release( struct inode *, struct file * );
static ssize_t device_read(struct file *file, char *buf, size_t count, loff_t *off);
static ssize_t device_write(struct file *file, const char *buf, size_t count, loff_t
*off);

static int major_number; // senior number of device
static int is_device_open = 0;

static char text[100]= "It`s symbolDriver\n";
```

```
static char* text_ptr;

static struct file_operations fops =
 {
  .read = device_read,
  .write = device_write,
  .open = device_open,
  .release = device_release
 };

static int __init test_init( void )
{
 printk( "Driver \”symbolDriver\” is loaded!\n" );

 major_number = register_chrdev( 0, DEVICE_NAME, &fops );

 if ( major_number < 0 )
 {
  printk( "Registering the character device failed with %d\n", major_number );
  return major_number;
 }
 printk( "Major number: %d 0'.\n", major_number );
 return SUCCESS;
}

static void __exit test_exit( void )
{
 unregister_chrdev( major_number, DEVICE_NAME );
 printk( KERN_ALERT " Driver \”symbolDriver\” is unloaded!\n" );
}

module_init( test_init );
module_exit( test_exit );

static int device_open( struct inode *inode, struct file *file )
{
 text_ptr = text;
 if ( is_device_open )
  return -EBUSY;
 is_device_open++;
 return SUCCESS;
}

static int device_release( struct inode *inode, struct file *file )
{
 is_device_open--;
 return SUCCESS;
}

static ssize_t device_write(struct file *file, const char *buf, size_t count, loff_t
* off )
{

int write_read = 0;
 while ( count && *buf )
 {
  get_user( *( text_ptr++ ), buf++ );
  count--;
  write_read++;
  }
 return write_read;
```

```
}

static ssize_t device_read( struct file *file, char *buf, size_t count, loff_t *
off )
{
 int byte_read = 0;
 if ( *text_ptr == 0 )
  return 0;
 while ( count && *text_ptr )
 {
  put_user( *( text_ptr++ ), buf++ );
  count--;
  byte_read++;
 }

 return byte_read;
}
```

Листинг 1.2. Makefile

```
TARGET = symbolDriver
obj-m := $(TARGET).o

KERNELDIR ?= /lib/modules/$(shell uname -r)/build
PWD       := $(shell pwd)
CC          = gcc

all:
    $(MAKE) -C $(KERNELDIR) M=$(PWD)

clean:
     rm -rf *.o *~ core .depend .*.cmd *.ko *.mod.c .tmp_versions *.order *.symvers
```

Результат запуска Makefile

```
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ make
make -C /lib/modules/4.4.0-75-generic/build M=/home/michael/lab3-Driver
make[1]: Entering directory '/usr/src/linux-headers-4.4.0-75-generic'
  CC [M]  /home/michael/lab3-Driver/symbolDriver.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/michael/lab3-Driver/symbolDriver.mod.o
  LD [M]  /home/michael/lab3-Driver/symbolDriver.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.4.0-75-generic'
```

В директории ~/lab3-Driver появились следующие файлы:

```
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ ls -l
total 40
-rw-rw-r-- 1 michael michael    8 апр 29 17:57 built-in.o
-rw-rw-r-- 1 michael michael  260 апр 29 17:56 Makefile
-rw-rw-r-- 1 michael michael   72 апр 29 17:57 modules.order
-rw-rw-r-- 1 michael michael    0 апр 29 17:57 Module.symvers
-rw-rw-r-- 1 michael michael 2193 апр 29 17:16 symbolDriver.c
-rw-rw-r-- 1 michael michael 6968 апр 29 17:57 symbolDriver.ko
-rw-rw-r-- 1 michael michael  988 апр 29 17:57 symbolDriver.mod.c
-rw-rw-r-- 1 michael michael 3144 апр 29 17:57 symbolDriver.mod.o
-rw-rw-r-- 1 michael michael 6064 апр 29 17:57 symbolDriver.o
```

Описание файлов:

`built-in.o` – в документации сказано что-то про то, что файлы с данным расширением появляются в папках не указаны, как директория для для модуля ядра (в целом до конца не ясно что это и для чего).

Содержание файла: **`!<arch>`**

`modules.order` – файл определяет порядок подключения модулей, что по факту определяется порядком следования в Makefile. Команд modprobe использует файл, чтобы подключить все компоненты модуля (модуль может состоять из нескольких отдельных модулей, которые и нужно подключить в правильном порядке).

Содержание файла**: `kernel//home/michael/lab3-Driver/symbolDriver.ko`**

В данном случае модуль состоит всего из одного файла.

`Module.symvers [1]-` файл должен содержать список модулей, которые были скомпилированы. Представление информации происходит в следующем формате:

```
<CRC>       <Symbol>           <module>
0x2d036834  scsi_remove_host   drivers/scsi/scsi_mod
```

Однако в данном случае файл пуст.

`symbolDriver.ko` – модуль ядра.

Примечание: при компиляции пользовательской программы на C/C++ (используя gcc или g++) скомпилированный бинарный файл имеет расширение *.o; По аналогии, при компиляции модуля ядра создаются файлы *.ko (kernel object).

`symbolDriver.mod.c [2]` – инструкция для работы с модулем. Файла можно назвать Makefile`ом для ядра. **Файл содержит** имя модуля, название функций по инициализации и отключении модуля, CRC модуля. При подгрузке модуля в систему, производится пересчет контрольной суммы модуля, и сравнение с **CRC**, указанной в данном файле: если суммы не совпадают – система отказывается подгружать модуль, ссылась на его не соответствие. Секция **depends** комметариев не требует.

```
#include <linux/module.h>
#include <linux/vermagic.h>
#include <linux/compiler.h>

MODULE_INFO(vermagic, VERMAGIC_STRING);

__visible struct module __this_module
__attribute__((section(".gnu.linkonce.this_module"))) = {
	.name = KBUILD_MODNAME,
	.init = init_module,
#ifdef CONFIG_MODULE_UNLOAD
	.exit = cleanup_module,
#endif
	.arch = MODULE_ARCH_INIT,
};

static const struct modversion_info ____versions[]
__used
__attribute__((section("__versions"))) = {
```

```
      { 0x9d35aeec, __VMLINUX_SYMBOL_STR(module_layout) },
      { 0x6bc3fbc0, __VMLINUX_SYMBOL_STR(__unregister_chrdev) },
      { 0x177758bc, __VMLINUX_SYMBOL_STR(__register_chrdev) },
      { 0x27e1a049, __VMLINUX_SYMBOL_STR(printk) },
      { 0xc3aaf0a9, __VMLINUX_SYMBOL_STR(__put_user_1) },
      { 0x167e7f9d, __VMLINUX_SYMBOL_STR(__get_user_1) },
      { 0xbdfb6dbb, __VMLINUX_SYMBOL_STR(__fentry__) },
};

static const char __module_depends[]
__used
__attribute__((section(".modinfo"))) =
"depends=";


MODULE_INFO(srcversion, "0B8ECCDFC5195C9B86A8F8D");
```

***symbolDriver.mod.o* –** скомпилированная версия файла, описанного выше.

Подробнейшее описание создания make файла и описание процесса компиляции приведено с источнике[3].

Тестирование работы модуля

Подгрузим модуль в Likux ядро командой **insmod**:

```
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ sudo insmod symbolDriver.ko
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ lsmod
Module                  Size  Used by
symbolDriver           16384  0
drbg                   32768  1
ansi_cprng             16384  0
ctr                    16384  2
...
```

Для того, чтобы убедиться, что модуль действительно подгружен успешно используем команды lsmod (результат в листинге выше) и **dmesg | tail.**

```
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ dmesg | tail
[ 2238.232817] usb 2-1.3: Product: USB Device
[ 2238.232821] usb 2-1.3: Manufacturer: A4TECH
...
[17250.894812] Driver ”symbolDriver” is loaded!
[17250.894826] Major number: 244 0'.
```

Драйвер подгрузился и успешно инициализировался (сообщение “Driver ”symbolDriver” is loaded!” должно выводиться в ходе исполнения функции __init).

Видим, что драйвер загружен успешно. Создан драйвер со старшим номера устройства 244. **Создадим файл устройства** с указанным номером:

```
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ sudo mknod /dev/testDevice c 244 0
```

Проверим, что символьное устройство действительно создано:

```
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ ls -l /dev/testDevice
crw-r--r-- 1 root root 244, 0 апр 29 19:49 /dev/testDevice
```

Теперь созданное устройство можно читать, в него можно записывать. При этом будет использоваться разработанный драйвер.

Чтение устройства.

```
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ sudo cat /dev/testDevice
It`s symbolDriver
```

Запись в устройство и чтение.

При попытке записи возникла проблема с доступом к устройству (хотя действия производились от имени root-пользователя).

```
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ sudo echo Where is Bill? >
/dev/testDevice
bash: /dev/testDevice: Permission denied
```

Попытка изменить права доступа «chmod 775» пользы не принесли. Методом проб установлено, что предоставление полного доступа командой «chmod 777» доступ все-таки удается восстановить.

```
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ sudo chmod 777 /dev/testDevice
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ sudo echo Where is Bill? >
/dev/testDevice
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ sudo cat /dev/testDevice
Where is Bill?
er
```

Драйвер работает корректно. Чтобы выгрузить модуль из ядра используем следующую команду:

```
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ sudo rmmod symbolDriver
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ dmesg | tail
...
[  269.424886] Driver ”symbolDriver” is loaded!
[  269.424894] Major number: 244 0'.
[  671.020907]  Driver ”symbolDriver” is unloaded!
```

Для удаления файл созданного устройства выполним:

```
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ sudo rm -i /dev/testDevice
rm: remove character special file '/dev/testDevice'? y
```

Очистка каталога от созданных файлов

```
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ make clean rm -rf *.o *~
core .depend .*.cmd *.ko *.mod.c .tmp_versions *.order *.symvers
michael@michael-LIFEBOOK-AH531:~/lab3-Driver$ ls -l
total 8
-rw-rw-r-- 1 michael michael  260 апр 29 17:56 Makefile
-rw-rw-r-- 1 michael michael 2192 апр 29 19:31 symbolDriver.c
```

2. USB-драйвер “clickDriver”

2.1. Основные особенности работы USB устройств

Одна из главных концепций USB заключается в том, что в USB-системе может быть **только один мастер**. Им является host-компьютер. USB-**устройства** всегда отвечают на запросы host-компьютера - они никогда **не могу посылать информацию самостоятельно.**

Хост всегда является мастером, а обмен данными должен осуществляться в обоих направлениях:

* OUT - отсылая пакет с флагом OUT, хост отсылает данные устройству
* IN - отсылая пакет с флагом IN, хост отправляет запрос на прием данных из устройства.

Endpoint - источник/приемник данных. Спецификация USB определеят endpoint (EP), как источник или приемник данных. Устройство может иметь до 32 EP: 16 на прием и 16 на передачу. Обращение к тому или иному endpoint'у происходит по его адресу.

2.2. Регистрация/выгрузка драйвера

Регистрация USB-драйвера подразумевает:

1. заполнение структуры usb_driver
2. регистрацию структуры в системе

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

```
struct usb_driver {
        // ...
        const char *name;
        int (*probe) (struct usb_interface *intf,
                      const struct usb_device_id *id);
        void (*disconnect) (struct usb_interface *intf);
        const struct usb_device_id *id_table;
        struct device_driver driver;
        // ...
};
```

name - это имя драйвера.
driver - говорит о том, что usb_driver унаследован от device_driver.
id_table - это массив структур usb_device_id. Этот список предназначен для определения соответствия подключаемого устройства определенным параметрам. Только те устройства, которые соответствуют перечисленным параметрам, могут быть подключены к драйверу. Если массив пуст, система будет пытаться подключить каждое устройство к драйверу. В самом простом случае каждый элемент id_table[i] содержит пару идентификаторов:

* идентификатор производителя (Vendor ID)
* идентификатор устройства (Device ID).

probe и **disconnect** - это callback-функции, вызываемые системой при подключении и отключении USB-устройства. Функция probe будет вызвана для каждого устройства,

если список id_table пуст, или только для тех устройств, которые соответствуют параметрам, перечисленным в списке.

2.3. Регистрация устройства

Один зарегистрированный драйвер может "подключать" несколько. Для подключения устройства к драйверу система вызывает функцию драйвера probe, которой передает 2 параметра:

```
static int my_probe(struct usb_interface *interface,
                    const struct usb_device_id *id)
```

interface - это интерфейс USB-устройства. Обычно USB-драйвер взаимодействует не с устройством напрямую, а с его интерфейсом. id - содержит информацию об устройстве. Если функция возвращает 0, то устройство успешно зарегистрировано, иначе - система попытается привязать" устройство к какому-нибудь другому драйверу.

Для отключения устройства от драйвера система вызывает функцию , которой передается один параметр - интерфейс:

```
static void my_disconnect(struct usb_interface *interface)
```

В общем случае, в функции probe для каждого подключаемого устройства выделяется структура в памяти, заполняется, затем регистрируется, например, символьное устройство, и проводится регистрация устройства в sysfs.

2.4. Использование USB Major

Для регистрации символьного устройства необходимо получить число major - либо статически, либо динамически (вызвать register_chrdev с параметром major = 0). Когда символьное устройство зарегистрировано, необходимо создать файл в директории /dev. Для этого можно воспользоваться либо командой mknod (из user-space).

В программном интерфейсе USB для этих целей есть функция usb_register_dev. Функция usb_register_dev выполняет следующие действия:

* регистрирует символьное устройство с заданным major номером
* в зарезервированном диапазоне minor номеров выделяет один номер для данного устройства. Этот номер записывает в interface->minor.
* создает все необходимые файлы в sysfs: после этого udev создает файлы в /dev/

Вызов usb_deregister_dev выполняет обратные процедуры, поэтому должен вызываться в функции disconnect.

2.5. Установка функции-обработчика прерывания

```
usb_fill_int_urb(mouse→irq, dev, pipe, mouse→data, (maxp > 8 ? 8 : maxp),
                 usb_mouse_irq, mouse, endpoint->bInterval);
```

- **mouse->irq** – указатель на структуру urb (USB Request Block) – структура для передачи и приема данных с USB устройством;
- **dev** – указатель на usb_device
- **pipe** – конкретный endpoint, к которому должен быть отправлен urb;

- **mouse->data** – буфер, в которого поступают входящие данные или из которого забираются исходящие данные;
- **(maxp > 8 ? 8 : maxp)** – длина буфера (предыдущее поле);
- **usb_mouse_irq** – указатель на функцию-обработчик прерывания;
- **mouse** – указатель на дополнительный объект;
- **endpoint->bInterval – заданный интервал.**

Листинг 2. clickDriver.c

```
#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>
#include <linux/fs.h>
#include <asm/segment.h>
#include <asm/uaccess.h>
#include <linux/buffer_head.h>
#include <linux/device.h>
#include <linux/cdev.h>

#define DRIVER_AUTHOR "Chebotarev"
#define DRIVER_LICENSE "GPL"

MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_LICENSE(DRIVER_LICENSE);

static int registered = 0;

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;
};

/*функция обработки прерывания*/
static void usb_mouse_irq(struct urb *urb)
{
	struct usb_mouse *mouse = urb->context;
	signed char *data = mouse->data;
	int status;
	status = usb_submit_urb (urb, GFP_ATOMIC);
	if(!(data[0] & 0x01) && !(data[0] & 0x02))
		pr_info("No button pressed!\n");
	else if(data[0] & 0x01)
		pr_info("Left mouse button clicked!\n");
	else if(data[0] & 0x02)
		pr_info("Right mouse button clicked!\n");
}

static int my_open(struct inode *i, struct file *f)
{
  printk(KERN_INFO "Driver: open()\n");
  return 0;
}
  static int my_close(struct inode *i, struct file *f)
{
  printk(KERN_INFO "Driver: close()\n");
  return 0;
}
```

```
static struct file_operations pugs_fops =
{
  .open = my_open,
  .release = my_close
};

static int usb_mouse_open(struct input_dev *dev)
{
	struct usb_mouse *mouse = input_get_drvdata(dev);
	mouse->irq->dev = mouse->usbdev;
	/*
	заполнение структуры urb.
Эта структура используется для передачи или приема данных от USB в асинхронном режиме.
	*/
	if (usb_submit_urb(mouse->irq, GFP_KERNEL))
		return -1;
	return 0;
}

static void usb_mouse_close(struct input_dev *dev)
{
	struct usb_mouse *mouse = input_get_drvdata(dev);
	/*
	освобождение структуры urb.
*/
	usb_kill_urb(mouse->irq);
}

/* подключение устройства к драйверу */
static int usb_mouse_probe(struct usb_interface *intf, const struct usb_device_id *id)
{
	struct usb_device *dev = interface_to_usbdev(intf);
	struct usb_host_interface *interface;
	struct usb_endpoint_descriptor *endpoint;
	struct usb_mouse *mouse;
	struct input_dev *input_dev;
	int pipe, maxp;
	int t;

	interface = intf->cur_altsetting;
	endpoint = &interface->endpoint[0].desc;

	pipe = usb_rcvintpipe(dev, endpoint->bEndpointAddress);
	maxp = usb_maxpacket(dev, pipe, usb_pipeout(pipe));

	mouse = kzalloc(sizeof(struct usb_mouse), GFP_KERNEL);
	input_dev = input_allocate_device();
	if (!mouse || !input_dev)
		goto fail1;

	mouse->data = usb_alloc_coherent(dev, 8, GFP_ATOMIC, &mouse->data_dma);
	if (!mouse->data)
		goto fail1;

	mouse->irq = usb_alloc_urb(0, GFP_KERNEL);
	if (!mouse->irq)
		goto fail2;

	mouse->usbdev = dev;
	mouse->dev = input_dev;
	input_dev->name = mouse->name;
	input_dev->phys = mouse->phys;
	usb_to_input_id(dev, &input_dev->id);
	input_dev->dev.parent = &intf->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 = usb_mouse_open;
	input_dev->close = usb_mouse_close;

	usb_fill_int_urb(mouse->irq, dev, pipe, mouse->data,
			 (maxp > 8 ? 8 : maxp),
			 usb_mouse_irq, mouse, endpoint->bInterval);
	mouse->irq->transfer_dma = mouse->data_dma;
	mouse->irq->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;

	if (input_register_device(mouse->dev))
		goto fail3;

	usb_set_intfdata(intf, mouse);

	//register device
	t = register_chrdev(91, "mymouse", &pugs_fops);
	if(t<0)
		registered = 0;
	else
		registered = 1;
	return t;

fail3:
	usb_free_urb(mouse->irq);
fail2:
	usb_free_coherent(dev, 8, mouse->data, mouse->data_dma);
fail1:
	input_free_device(input_dev);
	kfree(mouse);
	return -1;
}

/*отключение устройства от драйвера*/
static void usb_mouse_disconnect(struct usb_interface *intf)
{
	struct usb_mouse *mouse = usb_get_intfdata (intf);
	usb_set_intfdata(intf, NULL);
	if (mouse) {
		usb_kill_urb(mouse->irq);
		input_unregister_device(mouse->dev);
		usb_free_urb(mouse->irq);
		usb_free_coherent(interface_to_usbdev(intf), 8, mouse->data, mouse->data_dma);
		kfree(mouse);
	}
	if(registered)
		unregister_chrdev(91, "mymouse");
	registered = 0;

}

static struct usb_device_id usb_mouse_id_table [] = {
	{ USB_INTERFACE_INFO(USB_INTERFACE_CLASS_HID, USB_INTERFACE_SUBCLASS_BOOT,
		USB_INTERFACE_PROTOCOL_MOUSE) },
	{ }
};

MODULE_DEVICE_TABLE (usb, usb_mouse_id_table);

/*	имя драйвера должно быть установлено перед вызовом функции регистрации драйвера;
	поле id содержит идентификатор шины  (PCI, USB, ...), vendor ID и device ID
```

```
устройства;
       probe  и  disconnect  -  это callback-функции, вызываемые системой при
       подключении  и  отключении  USB-устройства.  probe  будет  вызван для
       каждого  устройства,  если  список  id_table  пуст, или только для тех
       устройств, которые соответствуют параметрам, перечисленным в списке.
*/
static struct usb_driver usb_mouse_driver = {
       .name         = "usbmouse",
       .probe        = usb_mouse_probe,
       .disconnect   = usb_mouse_disconnect,
       .id_table     = usb_mouse_id_table,
};

module_usb_driver(usb_mouse_driver);
```

Компиляция модуля с помощью вышеописанного Makefile`a:

```
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-10-semester/lab3-
Driver/clickDriver$ make
make -C /lib/modules/4.4.0-75-generic/build M=/home/michael/Desktop/SP-10-
semester/lab3-Driver/clickDriver
make[1]: Entering directory '/usr/src/linux-headers-4.4.0-75-generic'
  LD      /home/michael/Desktop/SP-10-semester/lab3-Driver/clickDriver/built-in.o
  CC [M]  /home/michael/Desktop/SP-10-semester/lab3-Driver/clickDriver/clickDriver.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/michael/Desktop/SP-10-semester/lab3-
Driver/clickDriver/clickDriver.mod.o
  LD [M]  /home/michael/Desktop/SP-10-semester/lab3-
Driver/clickDriver/clickDriver.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.4.0-75-generic'
```

Зарегистрируем устройство с major number = 91, номер такой же, какой указан в коде.

```
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-10-semester/lab3-
Driver/clickDriver$ sudo mknod /dev/myDev c 91 1
```

Выгрузим родной модуль по управлению USB устройствами:

```
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-10-semester/lab3-Driver/clickDriver$ sudo rmmod usbhid
```

Примечание: это так же отключило подключенную через USB внешнюю аудио-карту.

Загрузим модуль clickDriver.ko

```
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-10-semester/lab3-
Driver/clickDriver$  sudo insmod clickDriver.ko
```

Совершим пару кликов и выведем лог:

```
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-10-semester/lab3-
Driver/clickDriver$ dmesg | tail
...
[ 1410.357589] usbcore: deregistering interface driver usbhid
[ 1463.051894] input:  as /devices/pci0000:00/0000:00:1d.0/usb2/2-1/2-1.3/2-
```

```
1.3:1.1/input/input22
[ 1463.052235] usbcore: registered new interface driver usbmouse
[ 1463.052657] No button pressed!
[ 1463.053644] No button pressed!
[ 1463.092853] No button pressed!
[ 1499.203356] Left mouse button clicked!
[ 1499.316340] No button pressed!
[ 1499.466330] Right mouse button clicked!
[ 1499.637292] No button pressed!
[ 1499.829269] Right mouse button clicked!
[ 1499.937258] No button pressed!
[ 1500.159230] Left mouse button clicked!
[ 1500.291213] No button pressed!
[ 1500.654177] No button pressed!
```

Выгрузим свой драйвер, и вернем на место исходный модуль:

```
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-10-semester/lab3-
Driver/clickDriver$ sudo rmmod clickDriver.ko
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-10-semester/lab3-
Driver/clickDriver$ sudo modprobe usbhid
```

Мышка вновь заработала – это самое верное доказательство того, что драйвер подключился, и проверив логи видно, что это действительно так:

```
michael@michael-LIFEBOOK-AH531:~/Desktop/SP-10-semester/lab3-
Driver/clickDriver$ dmesg | tail
[ 1537.600387] usbcore: deregistering interface driver usbmouse
[ 1537.602200] No button pressed!
[ 1553.180827] input: A4TECH USB Device as /devices/pci0000:00/0000:00:1d.0/usb2/2-
1/2-1.3/2-1.3:1.0/0003:09DA:054F.0005/input/input24
[ 1553.237313] hid-generic 0003:09DA:054F.0005: input,hiddev0,hidraw1: USB HID v1.11
Keyboard [A4TECH USB Device] on usb-0000:00:1d.0-1.3/input0
[ 1553.238833] input: A4TECH USB Device as /devices/pci0000:00/0000:00:1d.0/usb2/2-
1/2-1.3/2-1.3:1.1/0003:09DA:054F.0006/input/input25
[ 1553.239561] hid-generic 0003:09DA:054F.0006: input,hidraw2: USB HID v1.11 Mouse
[A4TECH USB Device] on usb-0000:00:1d.0-1.3/input1
[ 1553.239619] usbcore: registered new interface driver usbhid
[ 1553.239623] usbhid: USB HID core driver
```

ИСТОЧНИКИ

1. What is the purpose of "Module.symvers" in Linux?

https://www.quora.com/What-is-the-purpose-of-Module-symvers-in-Linux

2. Meaning of version info in .mod.c file in Linux kernel

http://stackoverflow.com/questions/17922234/meaning-of-version-info-in-mod-c-file-in-linux-kernel/17924341

3. Building External Modules

https://www.kernel.org/doc/Documentation/kbuild/modules.txt