**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

a) 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 С/С++ (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$ s**udo 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`а:

|  |
| --- |
| 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>