Лекция 5.3 Драйверы USB

Разработал: Максимов А.Н.

Реализация драйверов USB.

Обзор особенностей шины USB

Обзор реализации USB в Linux

Представление на уровне пользователя

Основные структуры данных и функции

Пример структура драйвера.

Классы USB устройств.

Пример драйвера USB-serial.

Шина USB

Шина USB является иерархической и контролируется одним хостом. Хост использует мастер/слейв протокол для общения с подключенными USB устройствами. Слейв устройства не могут обмениваться между собой на прямую. Использование м.с. Протокола упрощает протокол и позволяет избегать коллизий. Текущая реализация протокола позволяет подключать к мастеру до 127 устройств.

Типы хост контроллеров

Используется несколько основных типов хост контроллеров:

(USB 1.0)

Universal Host Controller Interface (UHCI)

(USB 1.1)

Open Host Controller Interface (OHCI)

(USB 2.0)

Enhanced Host Controller Interface(EHCI).

(USB 3.0) – *поддержка в Linux начиная с ядра 2.6.31*

xtensible Host Controller Interface (xHCI)

Можно проверить тип USB контроллера при поимощи /proc/pci

Типы USB устройств

Существует несколько типов USB устройст которые могут использоваться для различных целей.

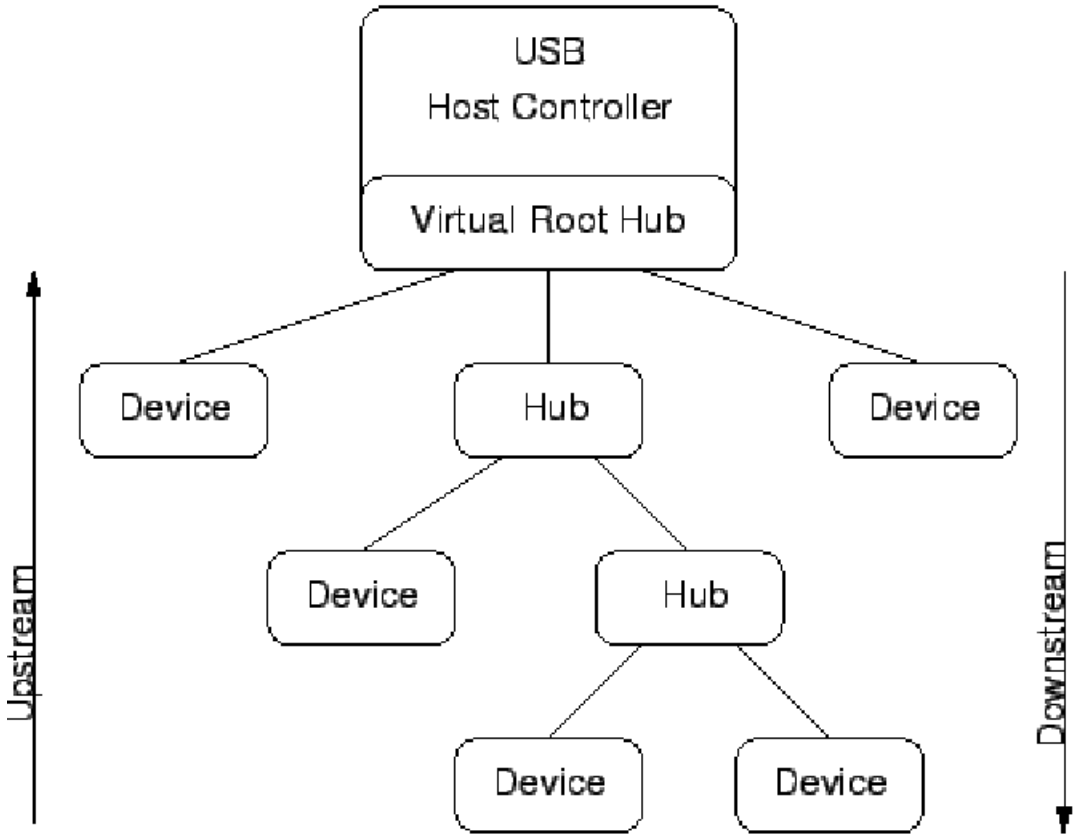
Устройства могут питатся от шины, самостоятельно или и использовать оба типа питания. USB обеспечивает до 500mA для устройств на шине

Скорость обмена данными может быть различной. USB спецификация определяет low speed и full speed устройства.

Low speed devices мышь, клавиатура,джойстик etc. 1.5MBit/s и обладают ограниченными возиожностями

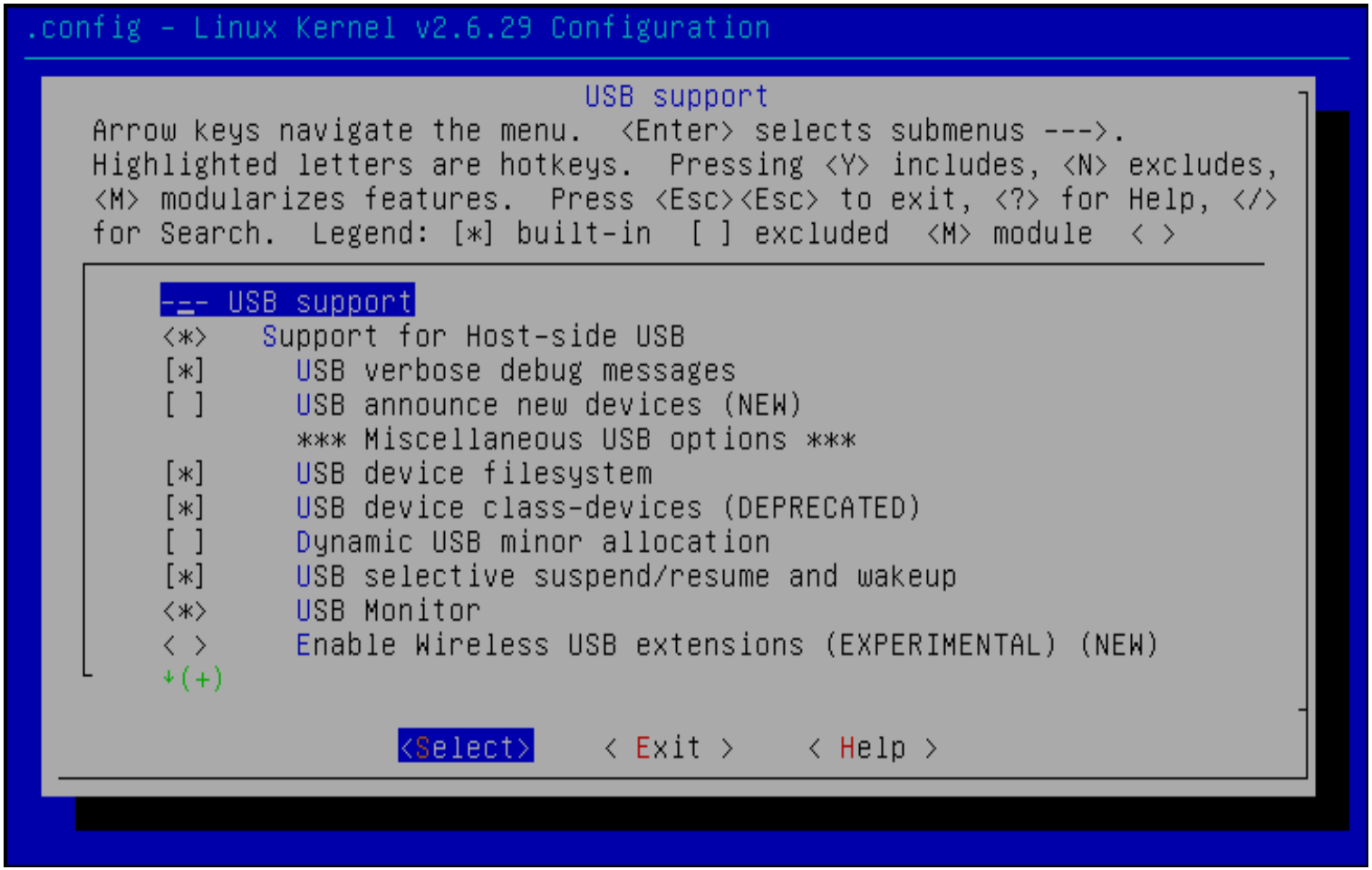
Full speed devices такие как аудио и видео могут использовать до 90% от 12Mbit/s.

Архитектура шины USB



Allan B. Cruse Advanced Systems Programming. Lecture 7. University of San Francisco. (lect_usfca1)

Подключение USB хоста в ядре



Allan B. Cruse Advanced Systems Programming. Lecture 7. University of San Francisco. (lect_usfca1)

Хабы

В компьютере может быть несколько USB портов. Порты используются для подключения слейв устройств или хабов. Хабы увеличивают кол-во устройств которые можно подключить к шине. Обычно физические порты на хост контроллере обрабатываются виртуальным хабом. Этот хаб эмулируется драйвером хост контроллера для унификации топологии сети.

Greg Kroah-Hartman How to Write a Linux USB Device Driver.
http://www.linuxjournal.com/article/4786

Типы драйверов USB

Модули подистемы USB — архитектурно независимая подсистема ядра, которая реализует USB спецификацию

Драйверы USB хостов — драйверы, которые релизуют взаимодействи с аппаратурой usb хоста на конкретной аппаратной платформе;

Драйверы USB устройст на шине USB (HMI, камеры, накопители и т.д.)

Драйверы USB гаджетов

Типы передачи данных.

USB позволяет передавать данные в двух направлениях и использовать три типа передачи данных.

От хоста к устройству - downstream или OUT transfer.

От устройства к хосту - upstream или IN transfer.

В зависимость от типа устройства различные типы передачи данных используются.

Control transfers – используется для запроса и надежной пересыкли коротких пакетов данных. Используется для конфигурирования устройств. Все должны поддерживать минимальный набор команд:

- GET STATUS
- CLEAR FEATURE
- SET FEATURE
- SET ADDRESS
- GET DESCRIPTOR
- SET DESCRIPTOR
- GET CONFIGURATION
- SET CONFIGURATION
- GET INTERFACE
- SET INTERFACE
- SYNCH FRAME

Производители могут использовать дополнительные команды.

Bulk transfers используются для запроса и надежно пересылки пакетов на полной скорости. Данный режим используют такие устройства как scannerы or scsi адаптеры.

Interrupt transfers аналогичен bulk transfer, который производится периодически. Если используется interrupt transfer, то драйвер контроллера хоста автоматически повторяет запросы с определенным интервалом (1ms - 255ms).

Isochronous transfers send or receive data streams in realtime with guaranteed bus bandwidth but without any reliability. In general these transfer types are used for audio and video devices.

Конечные точки

Одна из наиболее выжных абстракций — конечная точки. Могут быть рассматриваться как однонаправленные каналы.

Передают данные

OUT — с компьютре на устройство

IN — от устройства

Есть три типа:

CONTROL

INTERRUPT

BULK

ISOCHRONOS

Конечные точки

```
struct usb_host_endpoint {
  struct usb_endpoint_descriptor desc;
  struct list_head urb_list;
  void * hcpriv;
  struct ep_device * ep_dev;
  unsigned char * extra;
  int extralen;
  int enabled;
};
```

Desc — дескриптор endpoint, wMaxPacketSize в родном порядке байт

urb_list - urbs queued to this endpoint; maintained by usbcore

hcpriv for use by HCD; typically holds hardware dma queue head (QH) with one or more transfer descriptors (TDs) per urb

ep_dev ep_device for sysfs info

extra descriptors following this endpoint in the configuration

extralen how many bytes of “extra” are valid

enabled URBs may be submitted to this endpoint

Description

USB запросы всего помещаются передаются через указанную конечную точку, которая идентифицируется дескриптором.

urb

В linux взаимодействие с USB устройствами осуществляется при помощи urb (USB request block).

```
struct urb  {
// (IN) device and pipe specify the endpoint queue
   struct usb_device *dev;         // pointer to associated USB device
   unsigned int pipe;              // endpoint information
   unsigned int transfer_flags;    // ISO_ASAP, SHORT_NOT_OK, etc.
// (IN) all urbs need completion routines
   void *context;                  // context for completion routine
   void (*complete)(struct urb *); // pointer to completion routine
// (OUT) status after each completion
   int status;                     // returned status
// (IN) buffer used for data transfers
   void *transfer_buffer;          // associated data buffer
   int transfer_buffer_length;     // data buffer length
   int number_of_packets;          // size of iso_frame_desc
// (OUT) sometimes only part of CTRL/BULK/INTR transfer_buffer is used
   int actual_length;              // actual data buffer length
// (IN) setup stage for CTRL (pass a struct usb_ctrlrequest)
   unsigned char* setup_packet;    // setup packet (control only)
// Only for PERIODIC transfers (ISO, INTERRUPT)
    // (IN/OUT) start_frame is set unless ISO_ASAP isn't set
  int start_frame;                 // start frame
  int interval;                    // polling interval
    // ISO only: packets are only "best effort"; each can have errors
   int error_count;                // number of errors
   struct usb_iso_packet_descriptor iso_frame_desc[0];
};
```

Функции для работы с urb

Выделение и удаление:

struct urb *usb_alloc_urb(int isoframes, int mem_flags)

void usb_free_urb(struct urb *urb)

Пример:

```
urb = usb_alloc_urb(0, GFP_KERNEL);
```

Инициализация

<linux/usb.h>

fill_control_urb() and fill_bulk_urb().

Необходимые параметры

usb device pointer,

pipe (usual format from usb.h),

transfer buffer,

transfer length,

completion handler,

И контекстand its context

Функции для работы с urb

Запуск

int usb_submit_urb(struct urb *urb, int mem_flags)

Завершение запроса:

Асинхронное int usb_unlink_urb(struct urb *urb)

Синхронное void usb_kill_urb(struct urb *urb)

completion handler - функция формата:

typedef void (*usb_complete_t)(struct urb *, struct pt_regs *)

Пример драйвера

```
static struct usb_device_id id_table [] = {{USB_DEVICE(0x4242,0x0001)},{}};
MODULE_DEVICETABLE (usb,id_table)
static struct usb_driver led_driver = {
.owner =     THIS_MODULE,
.name =              "usbled",
.probe =     led_probe,
.disconnect =       led_disconnect,
.id_table =  id_table,
};
static int __init usb_led_init(void){
  int retval = 0;
  retval = usb_register(&led_driver);
  if (retval)
   err("usb_register failed. Error number %d", retval);
  return retval;
}
static void __exit usb_led_exit(void) {
   usb_deregister(&led_driver);
}
module_init (usb_led_init);
module_exit (usb_led_exit);
```

Пример драйвера

```
static void led_disconnect(struct usb_interface *interface)
{
struct usb_led *dev;

dev = usb_get_intfdata (interface);
usb_set_intfdata (interface, NULL);

device_remove_file(&interface->dev, &dev_attr_blue);
device_remove_file(&interface->dev, &dev_attr_red);
device_remove_file(&interface->dev, &dev_attr_green);

usb_put_dev(dev->udev);

kfree(dev);

dev_info(&interface->dev, "USB LED now disconnected\n");
}
```

Пример драйвера

```
static int led_probe(struct usb_interface *interface, const struct usb_device_id *id) {
struct usb_device *udev = interface_to_usbdev(interface);
struct usb_led *dev = NULL;
int retval = -ENOMEM;
dev = kmalloc(sizeof(struct usb_led), GFP_KERNEL);
if (dev == NULL) {
dev_err(&interface->dev, "Out of memory\n");
goto error;
}
memset (dev, 0x00, sizeof (*dev));

dev->udev = usb_get_dev(udev);
usb_set_intfdata (interface, dev);
device_create_file(&interface->dev, &dev_attr_blue);
device_create_file(&interface->dev, &dev_attr_red);
device_create_file(&interface->dev, &dev_attr_green);

dev_info(&interface->dev, "USB LED device now attached\n");
return 0;

error:
kfree(dev);
return retval;
}
```

Пример драйвера

```
#define show_set(value)       \
static ssize_t show_##value(struct device *dev, char *buf)          \
{                                                                   \
struct usb_interface *intf = to_usb_interface(dev);             \
struct usb_led *led = usb_get_intfdata(intf);                   \
\
return sprintf(buf, "%d\n", led->value);                    \
}                                                                   \
static ssize_t set_##value(struct device *dev, const char *buf, size_t count)   \
{                                                                   \
struct usb_interface *intf = to_usb_interface(dev);             \
struct usb_led *led = usb_get_intfdata(intf);                   \
int temp = simple_strtoul(buf, NULL, 10);                       \
\
led->value = temp;                                              \
change_color(led);                                              \
return count;                                                   \
}                                                                   \
static DEVICE_ATTR(value, S_IWUGO | S_IRUGO, show_##value, set_##value);
show_set(blue);
show_set(red);
show_set(green);
```

Пример драйвера

```
#define BLUE    0x04
#define RED     0x02
#define GREEN  0x01
static void change_color(struct usb_led *led){
int retval;
unsigned char color = 0x07;unsigned char *buffer;

buffer = kmalloc(8, GFP_KERNEL);
if (!buffer) {dev_err(&led->udev->dev, "out of memory\n");return;}
if (led->blue)  color &= ~(BLUE);
if (led->red)    color &= ~(RED);
if (led->green)
color &= ~(GREEN);
dev_dbg(&led->udev->dev,
"blue = %d, red = %d, green = %d, color = %.2x\n",
led->blue, led->red, led->green, color);

retval = usb_control_msg(led->udev,
usb_sndctrlpipe(led->udev, 0),
0x12,0xc8,(0x02 * 0x100) + 0x0a,(0x00 * 0x100) + color,
buffer,       8,2 * HZ);
if (retval) dev_dbg(&led->udev->dev, "retval = %d\n", retval);
   kfree(buffer);
}
```

Пример драйвера

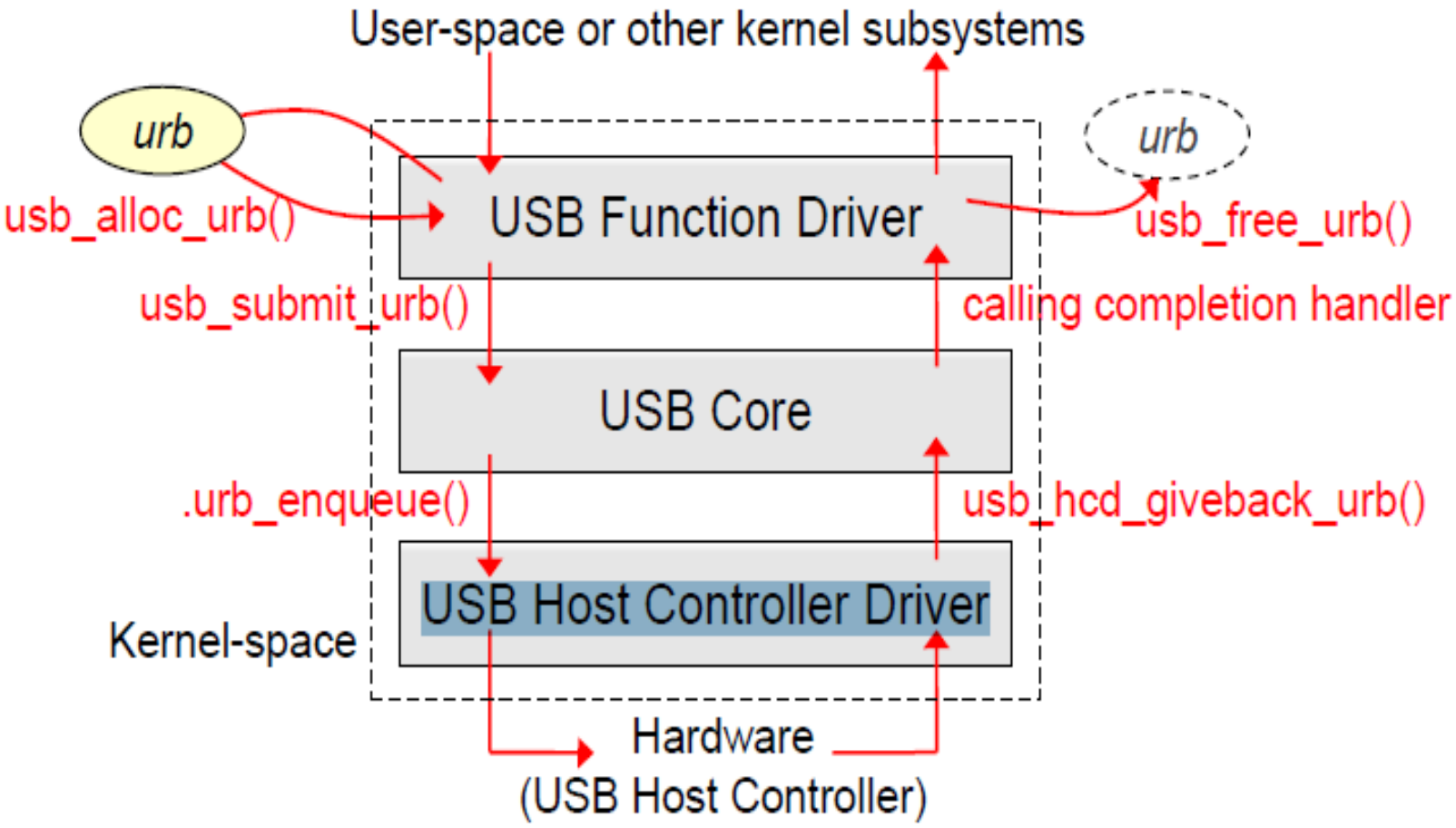
```
#include <linux/config.h>
#include <linux/kernel.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/module.h>
#include <linux/usb.h>

#define DRIVER_AUTHOR "Greg Kroah-Hartman, greg@kroah.com"
#define DRIVER_DESC "USB LED Driver"
#define VENDOR_ID  0x0fc5
#define PRODUCT_ID        0x1223
/* table of devices that work with this driver */
static struct usb_device_id id_table [] = {
{ USB_DEVICE(VENDOR_ID, PRODUCT_ID) },{ },};
MODULE_DEVICE_TABLE (usb, id_table);

struct usb_led {
struct usb_device *      udev;
unsigned char            blue;
unsigned char            red;
unsigned char            green;
};
```

Часть 2. Обзор архитектуры хост контроллера USB

Место драйвер usb хост контроллера в стеке USB.



Какие модули используются.

Файлы исходного кода для USB хост контроллера.

UHCI:

uhci-hcd.c uhci-hub.c (всегда на PCI)

OHCI:

ohci-hcd.c ohci-hub.c ohci-mem.c

ohci-pci.c

EHCI:

ehci-hcd.c ehci-hub.c ehci-mem.c (общие функции для реализации работы драйвера echi контроллер)

ehci-pci.c (реализацию echi контроллера для шины PCI)

xHCI:

xhci-hcd.c xhci-hub.c xhci-mem.c xhci-ring.c

xhci-pci.c

Как Linux находит USB хост контроллер.

USB хост контроллер на шине PCI находится по base_class:

UHCI,OHCI,EHCI,xHCI:

Пример. (ehci-pci.c)

```
// Обрабатываем любой  USB 2.0 EHCI контроллер
static const struct pci_device_id pci_ids [] = { {
PCI_DEVICE_CLASS(PCI_CLASS_SERIAL_USB_EHCI, ~0),
.driver_data =      (unsigned long) &ehci_pci_hc_driver,
},{  }
};
MODULE_DEVICE_TABLE(pci, pci_ids);
static struct pci_driver ehci_pci_driver = {
.name =             (char *) hcd_name,
.id_table =  pci_ids,
.probe =     usb_hcd_pci_probe,
.remove =  usb_hcd_pci_remove,
.shutdown =        usb_hcd_pci_shutdown,
};
```

Определение PIC base class для для usb хост контроллера.

В спецификации PCI 3.0 для usb хост контроллера определено следующее.

Base Class	Sub-Class	Interface	Meaning
0Ch	00	00h	IEEE 1394 (FireWire)
		10h	IEEE 1394 following the 1394 OpenHCI specification
	01h	00h	ACCESS.bus
	02h	00h	SSA
	03h	00h	Universal Serial Bus (USB) following the Universal Host Controller Specification
		10h	Universal Serial Bus (USB) following the Open Host Controller Specification
		20h	USB2 host controller following the Intel Enhanced Host Controller Interface
		80h	Universal Serial Bus with no specific programming interface
		FEh	USB device (not host controller)

Практическо задание. Найти usb хост контроллер на шине PCI.

Так выглядит в VMWare

>lspci

00:00.0 Host bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX Host bridge (rev 01)

00:01.0 PCI bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX AGP bridge (re

:.

02:02.0 USB Controller: VMware Inc Abstract USB2 EHCI Controller

Ключевые структуры данных драйвера хост контроллера USB..

Одной из ключевых структур данных является hc_driver. Данная структура содержит указатели на операции, которые должен реализовать хост контроллер.

```
static const struct hc_driver ehci_pci_hc_driver = {
.description =              hcd_name,
.product_desc =             "EHCI Host Controller",
.hcd_priv_size =      sizeof(struct ehci_hcd),
// generic hardware linkage
.irq =                      ehci_irq,
.flags =                    HCD_MEMORY | HCD_USB2,
// basic lifecycle operations
.reset =                    ehci_pci_setup,
.start =                    ehci_run,
#ifdef  CONFIG_PM
.pci_suspend =              ehci_pci_suspend,
.pci_resume =               ehci_pci_resume,
#endif
.stop =                     ehci_stop,
.shutdown =           ehci_shutdown,
// managing i/o requests and associated device resources
.urb_enqueue =              ehci_urb_enqueue,
.urb_dequeue =              ehci_urb_dequeue,
.endpoint_disable = ehci_endpoint_disable,
.endpoint_reset =     ehci_endpoint_reset,
// scheduling support
.get_frame_number =         ehci_get_frame,
// root hub support
.hub_status_data = ehci_hub_status_data,
.hub_control =              ehci_hub_control,
.bus_suspend =              ehci_bus_suspend,
.bus_resume =               ehci_bus_resume,
.relinquish_port =    ehci_relinquish_port,
.port_handed_over =         ehci_port_handed_over,
.clear_tt_buffer_complete  = ehci_clear_tt_buffer_complete,
};
```

Usb. Практический пример (ч.1).

```
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/module.h>
#include <linux/kref.h>
#include <linux/poll.h>
#include <asm/uaccess.h>
#include <linux/usb.h>

#define VENDOR_ID   0x0547
#define PRODUCT_ID  0x1002
// Define the vendor commands supported by OSR USB FX2 device.
#define OSRFX2_READ_SWITCHES                    0xD6
#define OSRFX2_READ_BARGRAPH_DISPLAY    0xD7
#define OSRFX2_SET_BARGRAPH_DISPLAY     0xD8
#define OSRFX2_IS_HIGH_SPEED                    0xD9
#define OSRFX2_REENUMERATE                      0xDA

struct bargraph_state {
    union {         struct {        // Верхние две лампы индикатора не горят
            unsigned char Bar4 : 1;
            unsigned char Bar5 : 1;
            unsigned char Bar6 : 1;
            unsigned char Bar7 : 1;
            unsigned char Bar8 : 1;
            unsigned char Bar1 : 1;
            unsigned char Bar2 : 1;
            unsigned char Bar3 : 1;
        };
        unsigned char BarsOctet; // The state of all eight bars as a single octet.
    };
} __attribute__ ((packed));
static struct usb_device_id id_table [] = {    { USB_DEVICE( VENDOR_ID, PRODUCT_ID ) },    { },};
MODULE_DEVICE_TABLE(usb, id_table);
```

Практический пример (ч.2).

```
struct osrfx2 {                    // This is the private device context structure.
    struct usb_device    * udev;
    struct usb_interface * interface;
    unsigned char * int_in_buffer; // Transfer Buffer
    size_t int_in_size;            // Buffer size
    __u8  int_in_endpointAddr;
    __u8  int_in_endpointInterval;
    struct urb * int_in_urb;
    struct kref      kref;
    struct semaphore sem;
};
static struct usb_driver osrfx2_driver; // Forward declaration for our usb_driver definition later.
static ssize_t show_bargraph(struct device * dev,struct device_attribute * attr,char * buf)
{
    struct usb_interface  * intf   = to_usb_interface(dev);
    struct osrfx2         * fx2dev = usb_get_intfdata(intf);
    struct bargraph_state * packet;
    int retval;

    packet = kmalloc(sizeof(*packet), GFP_KERNEL);
    if (!packet) {
        return -ENOMEM;
    }
    packet->BarsOctet = 0;
    retval = usb_control_msg(fx2dev->udev,  usb_rcvctrlpipe(fx2dev->udev, 0),  OSRFX2_READ_BARGRAPH_DISPLAY,  USB_DIR_IN | USB_TYPE_VENDOR, 0,
          0, packet,eof(*packet),
                          USB_CTRL_GET_TIMEOUT);
    if (retval < 0) {        dev_err(&fx2dev->udev->dev, "%s - retval=%d\n",    __FUNCTION__, retval);
        kfree(packet);
        return retval;
    }
    retval = sprintf(buf, "%s%s%s%s%s%s%s%s",    /* bottom LED --> top LED */ (packet->Bar1) ? "*" : ".", (packet->Bar2) ? "*" : ".",  (packet->Bar3) ? "*" : ".",
            (packet->Bar4) ? "*" : ".",  (packet->Bar5) ? "*" : ".",      (packet->Bar6) ? "*" : ".",   (packet->Bar7) ? "*" : ".",                (packet->Bar8) ? "*" : "." );
    kfree(packet);
    return retval;
}
```

Практический пример (ч.3).

```
static ssize_t set_bargraph(struct device*dev,struct device_attribute*attr,const char* buf,size_t count) {
    struct usb_interface  * intf   = to_usb_interface(dev);
    struct osrfx2         * fx2dev = usb_get_intfdata(intf);
    struct bargraph_state * packet;

    unsigned int value;
    int retval;
    char * end;

    packet = kmalloc(sizeof(*packet), GFP_KERNEL);
    if (!packet) {
        return -ENOMEM;
    }
    packet->BarsOctet = 0;
    value = (simple_strtoul(buf, &end, 10) & 0xFF);
    if (buf == end) {
        value = 0;
    }
    packet->Bar1 = (value & 0x01) ? 1 : 0;
    packet->Bar2 = (value & 0x02) ? 1 : 0;
    packet->Bar3 = (value & 0x04) ? 1 : 0;
    packet->Bar4 = (value & 0x08) ? 1 : 0;
    packet->Bar5 = (value & 0x10) ? 1 : 0;
    packet->Bar6 = (value & 0x20) ? 1 : 0;
    packet->Bar7 = (value & 0x40) ? 1 : 0;
    packet->Bar8 = (value & 0x80) ? 1 : 0;

    retval = usb_control_msg(fx2dev->udev, usb_sndctrlpipe(fx2dev->udev, 0),
                        OSRFX2_SET_BARGRAPH_DISPLAY, USB_DIR_OUT | USB_TYPE_VENDOR,
                        0, 0, packet,  sizeof(*packet),   USB_CTRL_GET_TIMEOUT);

    if (retval < 0) {
        dev_err(&fx2dev->udev->dev, "%s - retval=%d\n",
                __FUNCTION__, retval);
    }
    kfree(packet);
    return count;
}
```

Практический пример (ч.4).

```
static void osrfx2_delete(struct kref * kref) {
    struct osrfx2 * fx2dev = container_of(kref, struct osrfx2, kref);
    usb_put_dev( fx2dev->udev );
    if (fx2dev->int_in_urb) {        usb_free_urb(fx2dev->int_in_urb);    }
    if (fx2dev->int_in_buffer) {
        kfree(fx2dev->int_in_buffer);
    }
    kfree( fx2dev );
}
static int osrfx2_probe(struct usb_interface * interface, const struct usb_device_id * id) {
    struct usb_device * udev = interface_to_usbdev(interface);
    struct osrfx2 * fx2dev = NULL;
    int retval;

    fx2dev = kmalloc(sizeof(struct osrfx2), GFP_KERNEL);
    if (fx2dev == NULL) {       retval = -ENOMEM;        goto error;
    }
    memset(fx2dev, 0, sizeof(*fx2dev));
    kref_init( &fx2dev->kref );

    fx2dev->udev = usb_get_dev(udev);
    fx2dev->interface = interface;

    usb_set_intfdata(interface, fx2dev);

    device_create_file(&interface->dev, &dev_attr_bargraph);

    retval = find_endpoints( fx2dev );
    if (retval != 0)
        goto error;
    dev_info(&interface->dev, "OSR USB-FX2 device now attached.\n");
    return 0;
error:
    dev_err(&interface->dev, "OSR USB-FX2 device probe failed: %d.\n", retval);
    if (fx2dev) {
        kref_put( &fx2dev->kref, osrfx2_delete );
    }
    return retval;
}
```

Практический пример (ч.5).

```
// Event: device instance is being disconnected (deleted)
static void osrfx2_disconnect(struct usb_interface * interface) {
    struct osrfx2 * fx2dev;

    lock_kernel();
    fx2dev = usb_get_intfdata(interface);
    usb_kill_urb(fx2dev->int_in_urb);

    usb_set_intfdata(interface, NULL);
    device_remove_file(&interface->dev, &dev_attr_bargraph);
    unlock_kernel();

    kref_put( &fx2dev->kref, osrfx2_delete );
    dev_info(&interface->dev, "OSR USB-FX2 now disconnected.\n");
}

static struct usb_driver osrfx2_driver = {
    .name        = "osrfx2",
    .probe       = osrfx2_probe,
    .disconnect  = osrfx2_disconnect,
    .id_table    = id_table,
};

static int __init osrfx2_init(void) {
    return usb_register(&osrfx2_driver);
}

static void __exit osrfx2_exit(void) {
    usb_deregister( &osrfx2_driver );
}
module_init( osrfx2_init );
module_exit( osrfx2_exit );
```

Дополнительная информация

1. Documentation/usb/URB.txt

2. Detlef Fliegl. Programming Guide for Linux USB Device Drivers http://swarm.cs.pub.ro/~razvan/books/lin-kernel/Programming%20Guide%20for%20Linux%20USB%20Device%20Drivers.pdf

3. Alessandro Rubini. Usb Device Drivers. http://www.linux.it/~rubini/docs/usb/usb.html

4. USB in a NutShell. http://www.beyondlogic.org/usbnutshell/usb1.htm

5. Greg Kroah-Hartman How to Write a Linux USB Device Driver. http://www.linuxjournal.com/article/4786

6. USB FAQ: Introductory Level http://www.microsoft.com/whdc/connect/usb/usbfaq_intro.mspx#EBEAC

(перечень классов usb устройств поддерживаемых в windows)

7.USB Class Codes http://www.usb.org/developers/defined_class (общий перечень классов устройств)

8. Approved Class Specification Documents http://www.usb.org/developers/devclass_docs

(спецификации для классы устройств usb)

9. Device Driver Support http://www.linux-usb.org/devices.html (перечень классов устройст, поддеживаемых в Linux)

10. Paravirtualized USB Supportfor Xenhttp://www.xen.org/files/xensummit_tokyo/26_NoboruIwamatsu-en.pdf (показана структура стека USB)

11.Paravirtualized USB Support for Xen http://www.xen.org/files/xensummit_tokyo/26_NoboruIwamatsu-en.pdf

(Показана USB стек и точки входа/выхода)

!!!! Что происходит при подключении USB устройства.

12. http://www.technovelty.org/code/linux/plugging-in-usb.html