

## 怎样写linux下的USB设备驱动程序

血殿pmvc | 浏览 188 次 | 问题未开放回答 | [① 举报](#)



### 最佳答案

推荐于2016-03-29 15:35:28

写一个USB的驱动程序最 基本的要做四件事：驱动程序要支持的设备、注册USB驱动程序、探测和断开、提交和控制urb（USB请求块）

驱动程序支持的设备：有一个结构体struct usb\_device\_id，这个结构体提供了一列不同类型的该驱动程序支持的USB设备，对于一个只控制一个特定的USB设备的驱动程序来说，struct usb\_device\_id表被定义为：

/\* 驱动程序支持的设备列表 \*/

```
static struct usb_device_id skel_table [] = {  
  
    { USB_DEVICE(USB_SKEL_VENDOR_ID, USB_SKEL_PRODUCT_ID) },  
  
    {} /* 终止入口 */  
};
```

MODULE\_DEVICE\_TABLE (usb, skel\_table);

对于PC驱动程序，MODULE\_DEVICE\_TABLE是必需的，而且usb必需为该宏的第一个值，而USB\_SKEL\_VENDOR\_ID和 USB\_SKEL\_PRODUCT\_ID就是这个特殊设备的制造商和产品的ID了，我们在程序中把定义的值改为我们这款USB的，如：

/\* 定义制造商和产品的ID号 \*/

```
#define USB_SKEL_VENDOR_ID    0x1234  
  
#define USB_SKEL_PRODUCT_ID    0x2345
```

这两个值可以通过命令lsusb，当然你得先把USB设备先插到主机上了。或者查看厂商的USB设备的手册也能得到，在我机器上运行lsusb是这样的结果：

Bus 004 Device 001: ID 0000:0000

Bus 003 Device 002: ID 1234:2345 Abc Corp.

Bus 002 Device 001: ID 0000:0000

Bus 001 Device 001: ID 0000:0000

得到这两个值后把它定义到程序里就可以了。

注册USB驱动程序：所有的USB驱动程序都必须创建的结构体是struct usb\_driver。这个结构体必须由USB驱动程序来填写，包括许多回调函数和变量，它们向USB核心代码描述USB驱动程序。创建一个有效的 struct usb\_driver结构体，只须要初始化五个字段就可以了，在框架程序中是这样的：

```
static struct usb_driver skel_driver = {
```

登录

还没有百度账号？  
[立即注册](#)

知道日报

[全部文章](#)



相关搜索



linux系统教程



世界上最好玩的游戏



网上最

分享



- 1

开家婚姻介绍所
- 2

干洗店成本
- 3

销售管理系统
- 4

洗衣液生产设备
- 5

毛坯房装修预算
- 6

干洗店设备价格
- 7

干洗店设备
- 8

全自动洗车设备
- 9

胸小怎么大
- 10

女生胸怎样变大
- 11

干洗店利润如何
- 12

花生豆腐设备

13

人工智能培

14

一键重装系统

15

泡沫混凝土

16

怎么大胸

17

开个干洗店多少

18

胸为什么那么大

19

商标转让

20

usb是什么

21

桶装水生产线

22

干洗店利润怎么

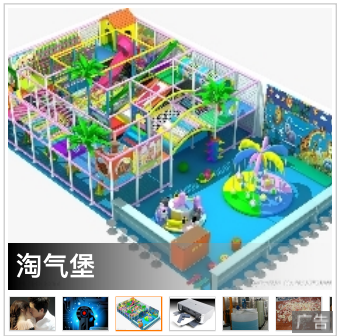
23

usb

24

一句话挽回爱情

广告



```
.name = "skeleton",
```

```
.probe = skel_probe,
```

```
.disconnect = skel_disconnect,
```

```
.id_table = skel_table,
```

```
};
```

探测和断开：当一个设备被安装而USB核心认为该驱动程序应该处理时，探测函数被调用，探测函数检查传递给它的设备信息，确定驱动程序是否真的适合该设备。当驱动程序因为某种原因不应该控制设备时，断开函数被调用，它可以做一些清理工作。探测回调函数中，USB驱动程序初始化任何可能用于控制USB设备的局部结构体，它还把所需的任何设备相关信息保存到一个局部结构体中，

提交和控制urb：当驱动程序有数据要发送到USB设备时（大多数情况是在驱动程序的写函数中），要分配一个urb来把数据传输给设备：

```
/* 创建一个urb,并且给它分配一个缓存*/
```

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

```
if (!urb) {
```

```
    retval = -ENOMEM;
```

```
    goto error;
```

```
}
```

当urb被成功分配后，还要创建一个DMA缓冲区来以高效的方式发送数据到设备，传递给驱动程序的数据要复制到这块缓冲中去：

```
buf = usb_buffer_alloc(dev->udev, count, GFP_KERNEL, &urb->transfer_dma);
```

```
if (!buf) {
```

```
    retval = -ENOMEM;
```

```
    goto error;
```

```
}
```

```
if (copy_from_user(buf, user_buffer, count)) {
```

```
    retval = -EFAULT;
```

```
    goto error;
```

```
}
```

当数据从用户空间正确复制到局部缓冲区后，urb必须在可以被提交给USB核心之前被正确初始化：

```
/* 初始化urb */
```

```
usb_fill_bulk_urb(urb, dev->udev,
```

```
    usb_sndbulkpipe(dev->udev, dev->bulk_out_endpointAddr),
```

快看，沓晃里有什么！

知道大数据，用数据解读生活点滴

团队7年，品牌升级由你决定！

分享

```
urb->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;
```

然后urb就可以被提交给USB核心以传输到设备了：

```
/* 把数据从批量OUT端口发出 */

retval = usb_submit_urb(urb, GFP_KERNEL);

if (retval) {

    err("%s - failed submitting write urb, error %d", __FUNCTION__, retval);

    goto error;

}
```

当urb被成功传输到USB设备之后，urb回调函数将被USB核心调用，在我们的例子中，我们初始化urb，使它指向skel\_write\_bulk\_callback函数，以下就是该函数：

```
static void skel_write_bulk_callback(struct urb *urb, struct pt_regs *regs)

{

    struct usb_skel *dev;

    dev = (struct usb_skel *)urb->context;

    if (urb->status &&

        !(urb->status == -ENOENT ||

          urb->status == -ECONNRESET ||

          urb->status == -ESHUTDOWN)) {

        dbg("%s - nonzero write bulk status received: %d",

            __FUNCTION__, urb->status);

    }

    /* 释放已分配的缓冲区 */

    usb_buffer_free(urb->dev, urb->transfer_buffer_length,

        urb->transfer_buffer, urb->transfer_dma);

}
```

有时候USB驱动程序只是要发送或者接收一些简单的数据，驱动程序也可以不用urb来进行数据的传输，这是涉及到两个简单的接口函数：usb\_bulk\_msg和usb\_control\_msg，在这个USB框架程序里读操作就是这样的一个应用：

```
/* 进行阻塞的批量读以从设备获取数据 */

retval = usb_bulk_msg(dev->udev,

    usb_rcvbulkpipe(dev->udev, dev->bulk_in_endpointAddr),

    dev->bulk_in_buffer,
```

分享

```
&count, HZ*10);
```

```
/*如果读成功，复制到用户空间 */
```

```
if (!retval) {

    if (copy_to_user(buffer, dev->bulk_in_buffer, count))

        retval = -EFAULT;

    else

        retval = count;

}
```

usb\_bulk\_msg接口函数的定义如下：

```
int usb_bulk_msg(struct usb_device *usb_dev,unsigned int pipe,

void *data,int len,int *actual_length,int timeout);
```

其参数为：

struct usb\_device \*usb\_dev：指向批量消息所发送的目标USB设备指针。

unsigned int pipe：批量消息所发送目标USB设备的特定端点，此值是调用usb\_sndbulkpipe或者usb\_rcvbulkpipe来创建的。

void \*data：如果是一个OUT端点，它是指向即将发送到设备的数据的指针。如果是IN端点，它是指向从设备读取的数据应该存放的位置的指针。

int len：data参数所指缓冲区的大小。

int \*actual\_length：指向保存实际传输字节数的位置的指针，至于传输到设备还是从设备接收取决于端点的方向。

int timeout：以Jiffies为单位的等待的超时时间，如果该值为0，该函数一直等待消息的结束。

如果该接口函数调用成功，返回值为0，否则返回一个负的错误值。

usb\_control\_msg接口函数定义如下：

```
int usb_control_msg(struct usb_device *dev,unsigned int pipe,__u8 request,__u8 requesttype,
__u16 value,__u16 index,void *data,__u16 size,int timeout)
```

除了允许驱动程序发送和接收USB控制消息之外，usb\_control\_msg函数的运作和usb\_bulk\_msg函数类似，其参数和usb\_bulk\_msg的参数有几个重要区别：

struct usb\_device \*dev：指向控制消息所发送的目标USB设备的指针。

unsigned int pipe：控制消息所发送的目标USB设备的特定端点，该值是调用usb\_sndctrlpipe或usb\_rcvctrlpipe来创建的。

\_\_u8 request：控制消息的USB请求值。

\_\_u8 requesttype：控制消息的USB请求类型值。

\_\_u16 value：控制消息的USB消息值。

分享

void \*data: 如果是一个OUT端点, 它是指身即将发送到设备的数据的指针。如果是一个IN端点, 它是指向从设备读取的数据应该存放的位置的指针。

\_\_u16 size: data参数所指缓冲区的大小。

int timeout: 以Jiffies为单位的应该等待的超时时间, 如果为0, 该函数将一直等待消息结束。

如果该接口函数调用成功, 返回传输到设备或者从设备读取的字节数; 如果不成功它返回一个负的错误值。

这两个接口函数都不能在一个中断上下文中或者持有自旋锁的情况下调用, 同样, 该函数也不能被任何其它函数取消, 使用时要谨慎。

我们要给未知的USB设备写驱动程序, 只需要把这个框架程序稍做修改就可以用了, 前面我们已经说过要修改制造商和产品的ID号, 把0xffff0这两个值改为未知USB的ID号。

```
#define USB_SKEL_VENDOR_ID    0xffff0

#define USB_SKEL_PRODUCT_ID    0xffff0
```

还有就是在探测函数中把需要探测的接口端点类型写好, 在这个框架程序中只探测了批量(USB\_ENDPOINT\_XFER\_BULK) IN和OUT端点, 可以在此处使用掩码(USB\_ENDPOINT\_XFERTYPE\_MASK) 让其探测其它的端点类型, 驱动程序会对USB设备的每一个接口进行一次探测, 当探测成功后, 驱动程序就被绑定到这个接口上。再有就是urb的初始化问题, 如果你只写简单的USB驱动, 这块不用多加考虑, 框架程序里的东西已经够用了, 这里我们简单介绍三个初始化urb的辅助函数:

usb\_fill\_int\_urb: 它的函数原型是这样的:

```
void usb_fill_int_urb(struct urb *urb, struct usb_device *dev,
unsigned int pipe, void *transfer_buff,
int buffer_length, usb_complete_t complete,
void *context, int interval);
```

这个函数用来正确的初始化即将被发送到USB设备的中断端点的urb。

usb\_fill\_bulk\_urb: 它的函数原型是这样的:

```
void usb_fill_bulk_urb(struct urb *urb, struct usb_device *dev,
unsigned int pipe, void *transfer_buffer,
int buffer_length, usb_complete_t complete)
```


这个函数是用来正确的初始化批量urb端点的。

usb\_fill\_control\_urb: 它的函数原型是这样的:




```
void usb_fill_control_urb(struct urb *urb, struct usb_device *dev, unsigned int pipe, unsigned char *setup_packet, void *transfer_buffer, int buffer_length, usb_complete_t complete, void *context);
```

这个函数是用来正确初始化控制urb端点的。

还有一个初始化等时urb的, 它现在还没有初始化函数, 所以它们在被提交到USB核心前, 必须在驱动程序中手工地进行初始化, 可以参考内核源代码树下的/usr/src/~/drivers/usb/media下的konicawc.c文件。

 为您推荐： [一句话挽回爱情](#) [一键重装系统](#) [泡沫混凝土设备](#) [女生胸怎样变大](#)






其他类似问题

- 怎样写linux下的USB设备驱动程序   12 2014-12-21
- 怎样写linux下的USB设备驱动程序 2016-08-01
- 怎样写Linux下的USB设备驱动程序  6 2015-01-12
- 怎样写linux下的USB设备驱动程序 2014-12-06
- 怎样写linux下的USB设备驱动程序 2016-03-20
- 怎样写linux下的USB设备驱动程序 2015-01-20
- 请问如何编写 Linux系统下USB设备驱动 2016-04-25
- 关于linux下USB设备驱动程序的最简单的问题 2014-12-31
- 更多类似问题 >

您可能关注的内容

- linux驱动,找阿里云云市场 广告
- click.aliyun.com ▼

设备驱动程序的相关知识

- Windows 无法加载这个硬件的设备驱动程序。驱动程序可能已损...  52 2008-12-13
- 键盘 ‘Windows 无法初始化这个硬件的设备驱动程序。(代码...  172 2010-06-18
- 设备驱动程序的开发  6 2008-12-16
- 电脑上显示： Windows 无法加载这个硬件的设备驱动程序。驱动...  35 2009-03-07
- Windows 无法初始化这个硬件的设备驱动程序。(代码 37  34 2008-10-05
- 更多关于设备驱动程序的知识 >

等待您来回答

- 成了友谊中的第三者，怎么办？ 10回答
- 有好听的QQ名字和好看的qq头像和个性签名推荐吗？ 6回答
- 你经历过最邪的事是什么？你相信有鬼吗？ 34回答
- 现在初中生谈恋爱的越来越多了！假如家里孩子上初中就开始谈... 25回答
- “不要找单亲家庭长大的孩子结婚”，这句话是否有道理？ 95回答
- 更多等待您来回答的问题 >



毛坯房装修预算



一键重装系统



游泳池水处理



怎样挽回老公



一句话挽回爱情



胸小怎么大



洗车加盟店 



新手帮助

- [如何答题](#)
- [获取采纳](#)
- [使用财富值](#)



玩法介绍

- [知道商城](#)
- [知道团队](#)
- [行家认证](#)
- [高质量问答](#)



投诉建议

- [意见反馈](#)
- [账号申诉](#)
- [智能咨询](#)