**背景知识介绍：**

1、USB设备组成结构：

usb设备包括配置(configuration)、 接口（ interface） 和端点(endpoint)， 一个USB设备驱动程序对应一个USB接口 ， 而非整个USB设备。usb设备可能由一个或者多个配置构成，一个配置可能为多个接口构成，一个接口可能由一个或者多个端点构成。usb的一个接口就是usb设备的一个基本功能。如果一个usb设备拥有多个功能，那就需要多个接口构成。

2、主机与usb设备通信方式：

主机首先要发送请求给设备，设备才能给出相应的响应。

usb设备驱动（发起请求的源，发起usb请求）->usb核心->usb主控制器驱动（使用usb总线把指令、信息发给usb设备）

usb设备响应指令（返回信息）usb总线->usb主控制器->usb核心->usb设备驱动

3、usb请求信息使用urb结构体描述：

USB请求块（ USB request block-URB）是USB设备驱动中用来与USB设备通信所用的基本载体和核心数据结构，非常类似于网络设备驱动中的sk\_buff结构体， 是USB主机与设备通信的“电波”。  
urb使用步骤：

（1）USB 设备驱动程序创建并初始化一个访问特定端点的 urb，并提交给USB core；  
（2）USB core提交该urb到 USB主控制器驱动程序；  
（3）USB 主控制器驱动程序根据该urb描述的信息，来访问 USB设备；  
（4）当设备访问结束后，USB 主控制器驱动程序通知USB 设备驱动程序。

创建urb：

struct urb \*usb\_alloc\_urb(int iso\_packets, gfp\_t mem\_flags)  
参数：  
iso\_packets： urb所包含的等时数据包的个数，创建非批量urb填写0。  
mem\_flags：内 存分配标识(如GFP\_KERNEL)， 参考kmalloc。

初始化urb：

由端点类型分为四种不同的端点：

对于中断urb，使用 usb\_fill\_int\_urb函数来初始化  
对于批量urb，使用 usb\_fill\_bulk\_urb函数来初始化  
对于控制urb，使用 usb\_fill\_control\_urb函数来初始化  
对于等时urb， 只能手动地初始化urb。

提交urb：

在完成urb的创建和初始化后， USB驱动需要将urb提交给USB核心.  
int usb\_submit\_urb(struct urb \*urb, gfp\_t mem\_flags)  
参数：  
urb： 要提交urb的指针  
mem\_flags: 内存分配标识(如GFP\_KERNEL)， 参考kmallocURB被提交到 USB核心后， USB核心指定usb主控制器驱动程序来处理该urb， 处理完之后， urb完成函数将被调用。

4、在Linux内核中，使用 struct usb\_driver结构描述一个USB驱动。

struct usb\_device\_id \*id\_table：保存这个驱动所支持的设备的ID，由制造商ID和编号ID构成，查看usb设备的两个ID号使用lsusb命令查看，前一个为制造商ID后一个为编号ID。当有usb加载时，驱动首先会检查是否有usb设备的设备ID与usb驱动的id\_table里面的设备ID相同，如果有则调用此驱动的probe函数。

当设备拔出的时候就会调用disconnect函数。

HID协议：HID(Human Interface Device)， 属于人机交互类的设备，如USB鼠标， USB键盘， USB游戏操纵杆等。 该类设备必须遵循HID设计规范。

**实现流程及代码介绍：**

1、实现struct usb\_driver结构体：

1. **struct** usb\_driver dnw\_driver = {
2. .name       = "dnw",   /\* 驱动名 \*/
3. .probe      = dnw\_probe, /\* 捕获函数 \*/
4. .disconnect = dnw\_disconnect, /\* 卸载函数 \*/
5. .id\_table   = dnw\_id\_table, /\* 设备列表 \*/
6. };

2、使用lsusb命令查看usb设备号，并实现id\_table参数：

1. **static** **struct** usb\_device\_id dnw\_id\_table [] = {
2. { USB\_DEVICE(0x5345, 0x1234) },
3. { }
4. };

3、实现probe函数：

3.1、（应用程序需要读写操作）初始化字符设备，usb总线的字符设备用struct usb\_class\_driver结构体和usb\_register\_dev函数创建，与一般字符设备创建函数cdev实现效果一样：

1. **static** **struct** usb\_class\_driver dnw\_class = {
2. .name =     "secbulk%d",/\*创建的设备名，打开设备时的名字\*/
3. .fops =     &dnw\_ops,/\*设备的操作函数集名字\*/
4. .minor\_base =   100,/\*需要创建的设备的次设备号，从100开始\*/
5. };
6. **static** **struct** file\_operations dnw\_ops =
7. {
8. .owner =    THIS\_MODULE,
9. .write =    dnw\_write,
10. .open =     dnw\_open,
11. .release =  dnw\_release,
12. };

3.2、创建usb设备struct usb\_device \*udev，并利用probe函数中的struct usb\_interface \*intf接口信息转换成usb对对应的设备，转换函数：interface\_to\_usbdev，所需参数为struct usb\_interface \*intf，再使用usb\_get\_dev包含interface\_to\_usbdev函数，返回值为usb设备。

利用struct usb\_interface \*intf获取端点地址：

3.3.1、创建接口描述：

struct usb\_host\_interface \*interface;

3.3.2、获取当前接口设置：

interface = intf->cur\_altsetting;

3.3.3、利用接口设置拿到接口描述符，在接口描述符里面就有端点的数量：

端点描述符初始化：struct usb\_endpoint\_descriptor \*endpoint;

interface->desc.bNumEndpoints接口下端点的数量，利用for循环找出需要的端点，使用usb\_endpoint\_is\_bulk\_out函数判断端点是否为bulk\_out端点。

1. **static** **int** dnw\_probe(**struct** usb\_interface \*intf, **const** **struct** usb\_device\_id \*id)
2. {
3. /\* 接口设置描述 \*/
4. **struct** usb\_host\_interface \*interface;
5. **struct** usb\_endpoint\_descriptor \*endpoint;
6. **int** i;
8. interface = intf->cur\_altsetting;
10. **for**(i=0;i<interface->desc.bNumEndpoints;i++)
11. {
12. endpoint = &interface->endpoint[i].desc;
13. **if**(usb\_endpoint\_is\_bulk\_out(endpoint))
14. {
15. bulk\_out\_endaddr =  endpoint->bEndpointAddress;
16. **break**;
17. }
19. }
21. usb\_register\_dev(intf,&dnw\_class);
23. udev = usb\_get\_dev(interface\_to\_usbdev(intf));
25. }

4、实现write函数：

4.1、（分配urb、初始化urb、提交urb）可以使用usb\_bulk\_msg函数代替之前三个步骤

int usb\_bulk\_msg(struct usb\_device \*usb\_dev, unsigned int pipe, void \*data, int len, int \*actual\_length, int timeout)函数参数说明

第一个参数：usb设备名；

第二个参数：usb管道；（usb创建管道函数有usb\_sndbulkpipe、和usb\_rcvbulkpipe两种，分别是创建发送和接收函数）

这两个函数的第一个参数为设备名，第二个参数为端点地址

第三个参数：需要发送的数据；

第四个参数：需要发送的数据的数量；

第五个参数：（返回的）实际传输的数据数量

第六个参数：时间间隔

4.2、（下载为大数据传输）usb端点选择为批量型

1. **static** ssize\_t dnw\_write(**struct** file \*file, **const** **char** \_\_user \*buf, **size\_t** len, loff\_t \*pos)
2. {
3. **size\_t** to\_write;
4. **size\_t** total\_write = 0;
5. **size\_t** act\_len;
7. **while**(len>0)
8. {
9. to\_write = min(len,(**size\_t**)BULKOUT\_BUFFER\_SIZE);
11. copy\_from\_user(bulkout\_buffer,buf+total\_write,to\_write);
13. usb\_bulk\_msg(udev,usb\_sndbulkpipe(udev,bulk\_out\_endaddr),bulkout\_buffer,to\_write,&act\_len,3\*HZ);
15. len -= to\_write;
16. total\_write += to\_write;
17. }
19. **return** total\_write;
20. }