[usb热插拔实现机制](http://blog.chinaunix.net/uid-25909619-id-3381683.html) 2012-10-22 18:58:54

分类：

原文地址：[usb热插拔实现机制](http://blog.chinaunix.net/uid-20727076-id-3279435.html) 作者：[jinxinxin163](http://blog.chinaunix.net/uid/20727076.html)

原文地址：<http://www.linux2web.net/?p=779>  
一.等待  
static struct usb\_driver hub\_driver = {  
    .name =        "hub",  
    .probe =    hub\_probe,  
    .disconnect =    hub\_disconnect,  
    .suspend =    hub\_suspend,  
    .resume =    hub\_resume,  
    .reset\_resume =    hub\_reset\_resume,  
    .pre\_reset =    hub\_pre\_reset,  
    .post\_reset =    hub\_post\_reset,  
    .ioctl =    hub\_ioctl,  
    .id\_table =    hub\_id\_table,  
    .supports\_autosuspend =    1,  
};  
int usb\_hub\_init(void)  
{  
    if (usb\_register(&hub\_driver) < 0) {  
        printk(KERN\_ERR "%s: can't register hub driver\n",  
            usbcore\_name);  
        return -1;  
    }  
  
    khubd\_task = kthread\_run(hub\_thread, NULL, "khubd");  
    if (!IS\_ERR(khubd\_task))  
        return 0;  
  
    /\* Fall through if kernel\_thread failed \*/  
    usb\_deregister(&hub\_driver);  
    printk(KERN\_ERR "%s: can't start khubd\n", usbcore\_name);  
  
    return -1;  
}  
  
static int hub\_thread(void \*\_\_unused)  
{  
    /\* khubd needs to be freezable to avoid intefering with USB-PERSIST  
     \* port handover.  Otherwise it might see that a full-speed device  
     \* was gone before the EHCI controller had handed its port over to  
     \* the companion full-speed controller.  
     \*/  
    set\_freezable();  
  
    do {  
        hub\_events();  
        wait\_event\_freezable(khubd\_wait,  
                !list\_empty(&hub\_event\_list) ||  
                kthread\_should\_stop());  
    } while (!kthread\_should\_stop() || !list\_empty(&hub\_event\_list));  
  
    pr\_debug("%s: khubd exiting\n", usbcore\_name);  
    return 0;  
}  
在hub\_thread执行时，会进入hub\_events,但是这时候hub\_event\_list队列为空，于是hub\_events退出并wait  
  
二.唤醒  
当主控制器初始化时，会初始化root hub，之后调用:  
static int hub\_probe(struct usb\_interface \*intf, const struct usb\_device\_id \*id)  
  
hub\_probe()所做的工作：  
1.为此root hub申请struct usb\_hub结构体并初始化它  
2.填充并提交中断in端点（由hub\_activate完成）  
usb\_fill\_int\_urb(hub->urb, hdev, pipe, \*hub->buffer, maxp, hub\_irq,  
        hub, endpoint->bInterval);  
usb\_submit\_urb(hub->urb, GFP\_NOIO);  
3.调用kick\_khubd(hub)  
  
static void kick\_khubd(struct usb\_hub \*hub)  
{  
    unsigned long    flags;  
  
    /\* Suppress autosuspend until khubd runs \*/  
    to\_usb\_interface(hub->intfdev)->pm\_usage\_cnt = 1;  
  
    spin\_lock\_irqsave(&hub\_event\_lock, flags);  
    if (!hub->disconnected && list\_empty(&hub->event\_list)) {  
        list\_add\_tail(&hub->event\_list, &hub\_event\_list);  
        wake\_up(&khubd\_wait);  
    }  
    spin\_unlock\_irqrestore(&hub\_event\_lock, flags);  
}  
由于这个时候root hub已经成功初始化了，所以kick\_khubd会将root hub的event\_list，添加到  
hub\_event\_list,表示root hub已经被识别了，同时wake\_up(&khubd\_wait)会唤醒上面的等待，  
于是hub\_events（）又一次执行了，但是这次，它是有备而来，因为hub\_event\_list不为空  
  
三.hub\_events  
  
hub\_events函数所做的工作：  
对每个端口号（共计bNbrPorts个端口，bNbrPorts这个值从hub描述符里边得到，因为此值描述了hub所用用的端口的情况），假如满足下列条件则调用hub\_port\_connect\_change()  
1.连接有变化  
2.端口本身重新使能,即所谓的enable,这种情况通常就是为了对付电磁干扰的,正如我们前面的判断中所说的那样  
3.在复位一个设备的时候发现其描述符变了,这通常对应的是硬件本身有了升级.很显然,第一种情况是真正的物理变化,后两者就算是逻辑变化  
代码模型如下：  
    for (i = 1; i <= hub->descriptor->bNbrPorts; i++) {  
        ...  
        if (connect\_change) //对root hub上活跃的port调用hub\_port\_connect\_change  
                hub\_port\_connect\_change(hub, i,  
                        portstatus, portchange);  
        ...      
    ｝  
  
hub\_port\_connect\_change()所做的工作：  
1.udev = usb\_alloc\_dev(hdev, hdev\_bus, port1);  
  原型：struct usb\_device \*usb\_alloc\_dev(struct usb\_device \*parent,  
                 struct usb\_bus \*bus, unsigned port1)  
  为探测到的usb设备（包括普通hub，u盘等）分配并初始化udev;  
2.status = hub\_port\_init(hub, udev, port1, i);  
  先进行两次新的策略(i=0和=1时),如果不行就再进行两次旧的策略(i=2和i=3时).所有这一切只有一个目的,就是为了获得设备的描述符，设置了udev->tt、udev->ttport和udev->ep0.desc.wMaxPacketSize,设置udev->status=       
  USB\_STATE\_ADDRESS  
3.usb\_new\_device(udev);  
(1)usb\_configure\_device(udev)->  
    usb\_get\_configuration(udev);  
        a.usb\_get\_descriptor()        //得到设备的描述符(包括设备描述符、配置描述符、接口描述符等)  
        b.usb\_parse\_configuration()    //分析以上描述符信息,提取出配置、接口等，并赋值给udev结构里相应的字段  
(2)device\_add(&udev->dev);  
   将usb设备注册到系统里，这个动作将触发驱动的匹配，由于这是个usb设备，所以万能usb驱动usb\_generic\_driver会匹配上，  
   从而generic\_probe会得到执行.关于 generic\_probe所做的工作，请参考：  
  http://blog.chinaunix.net/uid-20727076-id-3273535.html  
  
从上面可以看出来，这一次hub\_events()调用是由于主控制器初始化调用了hub\_probe,从而引发hub\_events调用。那root hub初始化完成以后hub\_events会如何触发呢?  
答案是通过中断！而这个中断的服务函数就是hub\_irq,也即是说,凡是真正的有端口变化事件发生,hub\_irq就会被调用,而hub\_irq()最终会调用kick\_khubd(),触发hub的event\_list,于是再次调用hub\_events().  
那hub\_irq是什么时候注册的呢？  
前面我们讲到：  
hub\_probe()所做的第二项工作是：填充并提交中断in端点（由hub\_activate完成）  
usb\_fill\_int\_urb(hub->urb, hdev, pipe, \*hub->buffer, maxp, hub\_irq,  
        hub, endpoint->bInterval);  
usb\_submit\_urb(hub->urb, GFP\_NOIO);  
hub\_irq作为参数传给了usb\_fill\_int\_urb,这样设定以后，只要root hub的端口有变化，hub\_irq就会执行到  
原文地址：<http://www.linux2web.net/?p=779>