[usb接口驱动加载流程分析](http://blog.chinaunix.net/uid-25909619-id-3381916.html) 2012-10-23 09:28:41

分类：

原文地址：[usb接口驱动加载流程分析](http://blog.chinaunix.net/uid-20727076-id-3273535.html) 作者：[jinxinxin163](http://blog.chinaunix.net/uid/20727076.html)

usb接口驱动加载流程分析  
struct usb\_device\_driver usb\_generic\_driver = {  
    .name =    "usb",  
    .probe = generic\_probe,  
    .disconnect = generic\_disconnect,  
#ifdef    CONFIG\_PM  
    .suspend = generic\_suspend,  
    .resume = generic\_resume,  
#endif  
    .supports\_autosuspend = 1,  
};  
  
struct bus\_type usb\_bus\_type = {  
    .name =        "usb",  
    .match =    usb\_device\_match,  
    .uevent =    usb\_uevent,  
};  
系统初始化时usb core会调用:  
retval = usb\_register\_device\_driver(&usb\_generic\_driver, THIS\_MODULE);  
此动作会：  
usb\_generic\_driver.drvwrap.driver.bus = &usb\_bus\_type;  
当USB设备(只有设备先被注册之后才会分析接口，才会注册接口) 被探测并被注册到系统后(用device\_add)，会调用usb\_bus\_type.mach()（只要是usb设备，都会跟 usb\_generic\_driver匹配上）,之后会调用usb\_probe\_device(),从而引发usb\_generic\_driver的 probe()调用.  
  
note:  
usb设备首先以设备的身份与usb\_generic\_driver匹配，成功之后，会分裂出接口，当对接口调用device\_add()后，会引起接口 和接口驱动的匹配，这个匹配还是用usb\_bus\_type.mach()函数。因为接口的device->bus=& usb\_bus\_type, 这跟usb设备是一样的，所以，都会调用到usb\_bus\_type.mach(),但设备和接口的处理流程是不一样的  
  
static int generic\_probe(struct usb\_device \*udev)  
{  
    int err, c;  
  
    if (udev->authorized == 0)  
        dev\_err(&udev->dev, "Device is not authorized for usage\n");  
    else {  
        c = usb\_choose\_configuration(udev);  
        if (c >= 0) {  
            err = usb\_set\_configuration(udev, c);  
            if (err) {  
                dev\_err(&udev->dev, "can't set config #%d, error %d\n",  
                    c, err);  
                /\* This need not be fatal.  The user can try to  
                 \* set other configurations. \*/  
            }  
        }  
    }  
    usb\_notify\_add\_device(udev);  
    return 0;  
}  
generic\_probe所做的工作：  
从设备可能的众多配置中选择一个合适的，然后去配置设备，从而让设备进入期待已久的Configured状态。  
  
int usb\_choose\_configuration(struct usb\_device \*udev);  
从udev->descriptor.bNumConfigurations个配置里选择一个合适的配置(struct usb\_host\_config)，并返回该配置的索引值  
例如：我机器上的的 usb 驱动加载时，输出：  
usb 1-1: configuration #1 chosen from 3 choices  
表示：此设备有3个配置，而驱动最终选择了索引号为1的配置，至于选择策略是怎样的，请看usb\_choose\_configuration()函数  
  
int usb\_set\_configuration(struct usb\_device \*dev, int configuration)  
{  
    int i, ret;  
    struct usb\_host\_config \*cp = NULL;  
    struct usb\_interface \*\*new\_interfaces = NULL;  
    int n, nintf;  
  
    if (dev->authorized == 0 || configuration == -1)  
        configuration = 0;  
    else {  
        for (i = 0; i < dev->descriptor.bNumConfigurations; i++) {  
            if (dev->config[i].desc.bConfigurationValue ==  
                    configuration) {  
                cp = &dev->config[i];            //取得索引号为configuration的配置的结构体  
                break;  
            }  
        }  
    }  
    if ((!cp && configuration != 0))  
        return -EINVAL;  
  
    if (cp && configuration == 0)  
        dev\_warn(&dev->dev, "config 0 descriptor??\n");        //假如配置的索引号为0，打出警告  
  
    n = nintf = 0;  
    if (cp) {  
        nintf = cp->desc.bNumInterfaces;  
        new\_interfaces = kmalloc(nintf \* sizeof(\*new\_interfaces),  
                GFP\_KERNEL);  
        if (!new\_interfaces) {  
            dev\_err(&dev->dev, "Out of memory\n");  
            return -ENOMEM;  
        }  
  
        for (; n < nintf; ++n) {  
            new\_interfaces[n] = kzalloc(  
                    sizeof(struct usb\_interface),  
                    GFP\_KERNEL);  
            if (!new\_interfaces[n]) {  
                dev\_err(&dev->dev, "Out of memory\n");  
                ret = -ENOMEM;  
free\_interfaces:  
                while (--n >= 0)  
                    kfree(new\_interfaces[n]);  
                kfree(new\_interfaces);  
                return ret;  
            }  
        }  
                                //以上代码为配置的bNumInterfaces个接口分配空间  
        i = dev->bus\_mA - cp->desc.bMaxPower \* 2;  
        if (i < 0)  
            dev\_warn(&dev->dev, "new config #%d exceeds power "  
                    "limit by %dmA\n",  
                    configuration, -i);  
    }  
  
    ret = usb\_control\_msg(dev, usb\_sndctrlpipe(dev, 0),  
                  USB\_REQ\_SET\_CONFIGURATION, 0, configuration, 0,  
                  NULL, 0, USB\_CTRL\_SET\_TIMEOUT);  
    if (ret < 0) {  
        /\* All the old state is gone, so what else can we do?  
         \* The device is probably useless now anyway.  
         \*/  
        cp = NULL;  
    }  
  
    dev->actconfig = cp;                    //发送USB\_REQ\_SET\_CONFIGURATION的urb信息来设置设备的配置为cp,同时记录在dev->actconfig里  
  
    usb\_set\_device\_state(dev, USB\_STATE\_CONFIGURED);    //设置设备的status 为USB\_STATE\_CONFIGURED  
  
    for (i = 0; i < nintf; ++i) {  
        struct usb\_interface\_cache \*intfc;  
        struct usb\_interface \*intf;  
        struct usb\_host\_interface \*alt;  
  
        cp->interface[i] = intf = new\_interfaces[i];  
        intfc = cp->intf\_cache[i];  
        intf->altsetting = intfc->altsetting;  
        intf->num\_altsetting = intfc->num\_altsetting;  
        intf->intf\_assoc = find\_iad(dev, cp, i);  
        kref\_get(&intfc->ref);  
  
        alt = usb\_altnum\_to\_altsetting(intf, 0);  
  
        /\* No altsetting 0?  We'll assume the first altsetting.  
         \* We could use a GetInterface call, but if a device is  
         \* so non-compliant that it doesn't have altsetting 0  
         \* then I wouldn't trust its reply anyway.  
         \*/  
        if (!alt)  
            alt = &intf->altsetting[0];  
  
        intf->cur\_altsetting = alt;  
        usb\_enable\_interface(dev, intf, true);  
        intf->dev.parent = &dev->dev;  
        intf->dev.driver = NULL;  
        intf->dev.bus = &usb\_bus\_type;  
        intf->dev.type = &usb\_if\_device\_type;  
        intf->dev.groups = usb\_interface\_groups;  
        intf->dev.dma\_mask = dev->dev.dma\_mask;  
        INIT\_WORK(&intf->reset\_ws, \_\_usb\_queue\_reset\_device);  
        device\_initialize(&intf->dev);  
        mark\_quiesced(intf);  
        dev\_set\_name(&intf->dev, "%d-%s:%d.%d",  
            dev->bus->busnum, dev->devpath,  
            configuration, alt->desc.bInterfaceNumber);    //初始化各个接口struct以及接口的dev结构  
    }  
    kfree(new\_interfaces);  
  
    for (i = 0; i < nintf; ++i) {  
        struct usb\_interface \*intf = cp->interface[i];  
  
        dev\_dbg(&dev->dev,  
            "adding %s (config #%d, interface %d)\n",  
            dev\_name(&intf->dev), configuration,  
            intf->cur\_altsetting->desc.bInterfaceNumber);  
        ret = device\_add(&intf->dev);                //将接口所对应的设备添加到系统，此动作将引发接口设备和接口驱动的匹配，从而引发接口设备驱动你的probe函数  
        if (ret != 0) {  
            dev\_err(&dev->dev, "device\_add(%s) --> %d\n",  
                dev\_name(&intf->dev), ret);  
            continue;  
        }  
        create\_intf\_ep\_devs(intf);  
    }  
  
    usb\_autosuspend\_device(dev);  
    return 0;  
}