USB 3G 网卡驱动流程

简介

首先介绍一下 linux 下的整体驱动模式：

本文基于的 linux kernel 版本为 2.6.36 (并且华为 EM770W 驱动，是由 FriendlyARM 公司定制的。所以该部分驱动可以在友善的官方网站上下载。其宏定义为 CONFIG_MACH_MINI6410)

总线，设备，驱动：三者是互相关联的，在总线上有设备列表，和驱动列表。当一个设备接入时，会在总线上遍历所有的驱动，知道找到支持他的驱动。然后就会将设备结构体下的 driver 指针指向该驱动。同样，驱动也会将该设备加入自己的设备指针列表。

3G 网卡是一个 USB 设备，那么就介绍一下 USB 驱动。

总线

Usb 总线结构体 struct bus_type usb_bus_type ={}; //那我们就来看看总线的结构体

[include/linux/device.h]文件中

```
struct bus_type {
    const char      *name;
    ……
    struct bus_type_private *p;
};
```

[drivers/base/base.h]文件中

```
struct bus_type_private {
    struct kset subsys;
    struct kset *drivers_kset;
    struct kset *devices_kset;
    struct klist klist_devices;
    struct klist klist_drivers;
    struct blocking_notifier_head bus_notifier;
    unsigned int drivers_autoprobe:1;
    struct bus_type *bus;
};
```

其中的 klist_devices, klist_drivers 分别为总线上的设备和驱动列表。当有新的设备接入，或加载新的驱动时。就会在这两个列表中遍历。

设备

这里就以 usb_device 为例

[include/linux/usb.h]文件中

```
struct usb_device {
    ……
    struct usb_bus *bus; //指向该设备的总线
    ……
```

```
    struce device dev;     //linux 下通用设备的属性
};
```

[include/linux/device.h]文件中

```
struct device {
    ……
    struct device_driver *driver;    /* which driver has allocated this device */
    ……
};
```

这里就有设备指向的总线和驱动指针了。一个设备之用一个驱动。下面的驱动，所支持的设备就是个列表了。驱动可以支持一系列的设备。

驱动

以 opton_driver 为例[drivers/usb/serial/option.c]

```
static struct usb_driver option_driver = {
    .name       = "option",
    .probe      = usb_serial_probe,
    .disconnect = usb_serial_disconnect,
#ifdef CONFIG_PM
    .suspend    = usb_serial_suspend,
    .resume     = usb_serial_resume,
    .supports_autosuspend = 1,
#endif
    .id_table   = option_ids,
    .no_dynamic_id =    1,
};
```

[include/linux/usb.h]文件中

```
struct usb_driver {
    const char *name;
    ……
    struct usbdrv_wrap drvwrap;
    ……
};
struct usbdrv_wrap {
    struct device_driver driver;
    int for_devices;
};
```

[include/linux/device.h]文件中

```
struct device_driver {
    ……
    struct bus_type     *bus;         //驱动所属的总线
```

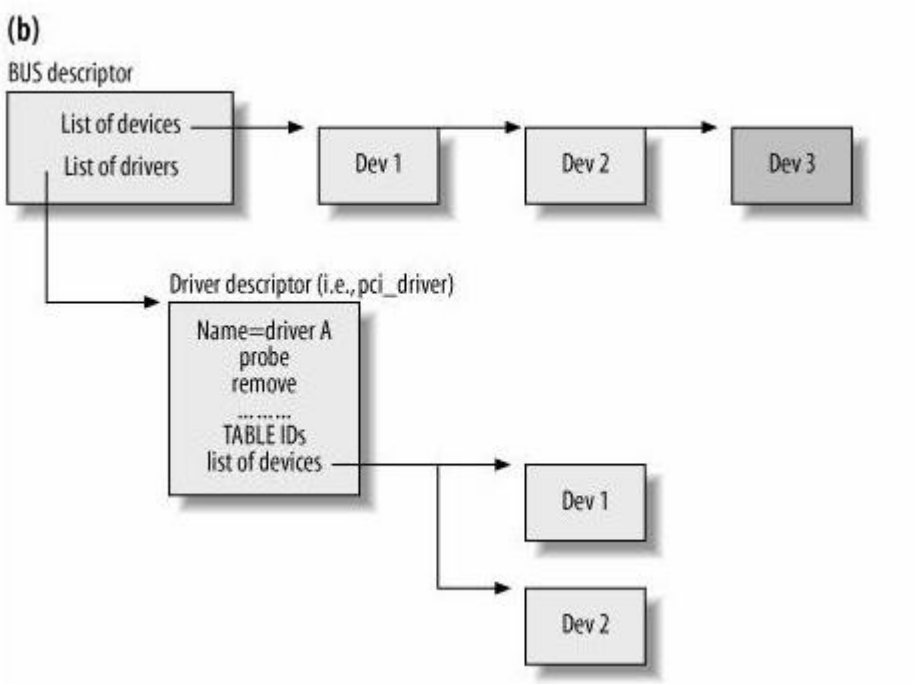
```
    ……
    struct driver_private *p;
    ……
};
[drivers/base/base.h]文件中
struct driver_private {
    struct kobject kobj;
    struct klist klist_devices;          //正在使用此驱动的设备列表
    struct klist_node knode_bus;
    struct module_kobject *mkobj;
    struct device_driver *driver;
};
```

option 驱动总线赋值

[drivers/usb/serial/option.c]中 option 初始化时，注册过程会将驱动所属总线赋值

```
static int __init option_init(void)
{
    retval = usb_register(&option_driver);
}
[include/linux/usb.h]中
static inline int usb_register(struct usb_driver *driver)

{

    return usb_register_driver(driver, THIS_MODULE, KBUILD_MODNAME);

}

[drivers/usb/core/driver.c]文件中
int usb_register_driver(struct usb_driver *new_driver, struct module *owner,
            const char *mod_name)
{
    new_driver->drvwrap.driver.bus = &usb_bus_type;
    new_driver->drvwrap.driver.probe = usb_probe_interface;
}
```

图例



以华为 **EM770W** 为例

以 HUAWEI EM770W WCDMA 3G 网卡为例。

在内核文件 linux/drivers/usb/serial/option.c 中。

```
static const struct usb_device_id option_ids[] = {
  { USB_DEVICE_AND_INTERFACE_INFO(HUAWEI_VENDOR_ID, HUAWEI_PRODUCT_E600, 0xff, 0xff, 0xff) },
}
```

有此行定义，在设备 EM770W 中读取出的.idProduct 与 HUAWEI_PRODUCT_E600 定义的值相同。

当一个 USB 设备插入时，会触发 hub_event 时间调用函数

一：添加 **usb** 设备

hub_events(); -->hub_port_connect_change();

--> udev = usb_alloc_dev(hdev, hdev->bus, port1); //将&usb_bus_type 分配给设备

usb_new_device(udev);

-->device_add(&udev->dev);(drivers/base/core.c)

-->bus_add_device(dev); //将设备加入总线设备列表

//klist_add_tail(&dev->p->knode_bus, &bus->p->klist_devices);

bus_probe_device(dev);

-->device_attach(dev);[drivers/base/dd.c]

-->bus_for_each_drv(dev->bus, NULL, dev, __device_attach); // 遍历驱动，找到支持此设备的驱动

-->__device_attach(drv, dev); // 找的驱动 usb 支持此设备

-->driver_probe_device(drv, dev);

-->really_probe(dev, drv); [drivers/base/dd.c]

-->drv->probe(dev); // 实际调用 usb_probe_device(dev);[drivers/usb/core/driver.c]

-->udriver->probe(udev); //实际上调用的 generic_probe(udev); [drivers/usb/core/generic.c]

-->usb_set_configuration(udev, c); [drivers/usb/core/message.c]

-->nintf = cp->desc.bNumInterfaces; // 这里首先得到设备的 Interface 数量,下面循环添加每个 Interface

```
for (i = 0; i < nintf; ++i) {
    ret = device_add(&intf->dev);
}
```

二: 添加 **interface**

下面进入每个 interface 的添加设备过程，在这里单独列出

device_add(&intf->dev);

-->bus_probe_device(dev); -->device_attach(dev); [drivers/base/dd.c]

-->bus_for_each_drv(dev->bus, NULL, dev, __device_attach); // 遍历驱动，找到支持此设备的驱动

-->__device_attach(drv, dev); // 遍历到 option 驱动时，在 option_ids[]结构体中，找到该驱动支持此设备
-->driver_probe_device(drv, dev);

-->really_probe(dev, drv); [drivers/base/dd.c]

-->drv->probe(dev); //这里实际调用的函数为 usb_probe_interface(dev);

-->driver->probe(inif, id); //这里实际调用的函数为 usb_serial_probe(interface, id); [drivers/usb/serial/usb-serial.c]

//这里检测到使用 pl2303 convertor, 并执行 pl2303 的 attach 函数

/*****************

*下面这部分做了许多事情。

* 创建一个 usb_serial 数据结构,增加一个 interface 接口设备,找到该接口的驱动(如 pl2303),设置 interface 的 endpoint 等

*****************/

-->type = search_serial_device(interface); //这里找到匹配的 pl2303 驱动 参看下一节【USB 检测 pl2303 驱动】

```
serial = create_serial(dev, interface, type); //创建 serial, 将 type 驱动赋值给新创建的 serial->type，即驱动指向为 pl2303
port = serial->port[i];
dev_set_name(&port->dev, "ttyUSB%d", port->number);
device_add(&port->dev); //调用增加设备 ttyUSB0
```

-->bus_probe_device(dev);

-->...如上...

-->really_probe(dev, drv); [drivers/base/dd.c]

-->dev->bus->probe(dev); // ***** 这里实际上调用的就是 usb_serial_device_probe(dev);

-->tty_register_device(usb_serial_tty_driver, minor, dev); //这里会再次调用 device_add(dev); 函数

USB 检测 pl2303 驱动

USB 的串行设备也分为很多种类。如 pl2303,ViVOpay
usb_serial_probe(interface, id);[drivers/usb/serial/usb-serial.c]

-->type = search_serial_device(interface); //根据提供的 interface 接口，

  //获得了 static struct usb_serial_driver pl2303_device ={} [drivers/usb/serial/pl2303.c]

-->list_for_each_entry(drv, &usb_serial_driver_list, driver_list) {

```
    id = get_iface_id(drv, iface);

    if (id)

        return drv;

}
```

pl2303 驱动的注册过程

```
static int __init pl2303_init(void)
{
    retval = usb_serial_register(&pl2303_device);
}
```

[drivers/usb/serial/usb-serial.c] 在注册函数中，将 pl2303 驱动加入 usb serial 驱动列表

```
int usb_serial_register(struct usb_serial_driver *driver)
{
    list_add(&driver->driver_list, &usb_serial_driver_list);
}
```

三: 添加 endpoint

// 至此，整个调用逐步返回！

最后在 usb_set_configuration(udev, c); [drivers/usb/core/message.c] 函数返回时调用

-->create_intf_ep_devs(intf); //增加 endpoint 设备

-->for (i = 0; i < alt->desc.bNumEndpoints; ++i)

    (void) usb_create_ep_devs(&intf->dev, &alt->endpoint[i], udev);

-->retval = device_register(&ep_dev->dev);

3G 网卡如何读写呢

在 pl2303 驱动加载完成后，生成的设备 ttyUSB0, ttyUSB1, ttyUSB2 后，即可以对这些设备进行操作。如 open,close,read,write,ioctl 等命令

pl2303.c 中有如下函数,即当打开网卡设备时，就会调用此函数。

```
static int pl2303_open(struct tty_struct *tty, struct usb_serial_port *port)
```

{

}

在板子的可使用 3G WCDMA 网卡上，因为开发板为 3G 网卡建立 3 个 USB 设备 ttyUSB0,ttyUSB1,ttyUSB2 所以在打开时，会调用 3 次该函数。

大概流程为

static int serial_open(struct tty_struct *tty, struct file *filp)

-->tty_port_open(&port->port, tty, filp);

-->port->ops->activate(port, tty); //此处调用的为 static int serial_activate(struct tty_port *tport, struct tty_struct *tty)

-->port->serial->type->open(tty, port); //这里即是调用 static int pl2303_open(struct tty_struct *tty, struct usb_serial_port *port)

从而对将设备打开，可以对设备进行读写，控制等操作。

设备识别

在设备插入后，以 HUAWEI ET127 3G 网卡为例。插入设备后会有如下的 log

scsi 0:0:0:0: CD-ROM HUAWEI Mobile CMCC CD 1.25 PQ: 0 ANSI: 0

大唐的 AirCard 901 插入后，同样会有 log 如下：

csi 1:0:0:0: CD-ROM AirCard SetupDisk 1.00 PQ: 0 ANSI: 2

这时，友善开发板将此设备识别成为一个光盘，即储存设备。这时，如果在 ubuntu 下，可以使用 usb_modeswitch 将 usb 设备进行模式转换。然后驱动 3G 网卡。我在网上搜索到一篇文章。说可以驱动大部分 3G 网卡，就此我问过胡菲菲。此前驱动大唐的 AirCard 901 用的就是此种方法。

http://linuxidc.com:81/Linux/2011-03/33430.htm

我昨天尝试按照这些步骤进行实现。但是遇到两个问题

1:arm 卡发板上使用的是 busybox， 和一些应用程序(如:dhcpcd)在 Android 系统下编译。我编译的 usb_modeswitch 无法运行

2:插入 usb 3G 设备后，arm 开发板识别为 sr0, sr1 设备。不知道该如何将此设备退出。或许 usb_modeswitch 转换完成后，即可以进行 AT 命令通信。还需要进行试验才能知道。

usb_modeswitch 作用

USB_ModeSwitch 是控制"flip flop"(多重设备)USB 装置的模式转换工具。作用有如下的几种：

1． Usb 闪存模式：提取和安装驱动

2． 转换模式：储存设备模式转换为所需（如 3G）设备模式

3． 所需设备模式：使用设备新的功能

详细的可以参看 usb_modeswitch 的 README 文档。

一些术语

USB 端点(endpoint)

端点是由厂商在 USB 设备内部建立的，因此是永久存在的。端点(endpoint)和主控制器(host controller)基于管道(pipe)连接。

一个端点只能单向的(in/out)传输数据。

端点是以 interface 来分组的。每一个 interface 对应着一个设备功能。

USB Interface

endpoint(端点)是以 interface 来分组的。比如华为的 WCDMA 是以每三个 ep 为一组，隶属于一个 interface。而且每一个 interface 对应着一个单独的设备功能。如华为 ET127 TD-CDMA 3G 网卡其中一个 interface 设备命名为 ttyUSB_utps_mms，对应的功能应该即为短信发送/接收。

USB 设备号

//WCDMA 华为 EM770W 在 ubuntu 10.04 下查看

crw-rw---- 1 root dialout 188, 1 2011-05-04 16:19 ttyUSB_utps_diag

crw-rw---- 1 root dialout 188, 0 2011-05-04 16:19 ttyUSB_utps_modem

crw-rw---- 1 root dialout 188, 2 2011-05-04 16:21 ttyUSB_utps_pcui

//TD-SCDMA 华为 ET127 在 ubuntu 10.04 下查看

crw-rw---- 1 root dialout 166, 1 2011-05-04 16:23 ttyUSB_utps_mms

crw-rw---- 1 root dialout 166, 0 2011-05-04 16:23 ttyUSB_utps_modem

crw-rw---- 1 root dialout 166, 2 2011-05-04 16:23 ttyUSB_utps_pcui

idVendor=12d1, idProduct=1da1

大唐 TD-SCDMA 设备参数

idVendor=1ab7, idProduct=0301

主设备号含义

ACM 调制解调器：主设备号 166

USB 打印机：主设备号 180（次设备号 0～15）

USB 串口：主设备号 188