华为数据卡/模块Linux第三方集成指导文档

Huawei Technologies Co., Ltd.

华为技术有限公司

Revision record 修订记录

Date 日期	Revision Version 修订 版本	CR ID / Defect ID CR号	Section Number 修改 章节	Change Description 修改描述	Author 作者
2009-4-15	V1.0.0			初稿	方晓志 00110321
2009-8-18	V1.1.0			修改了usb_stor_huawei_init函数的返回值	方晓志 00110321

华为数据卡/模块Linux第三方集成指导文档

Catalog 目 录

1 目的

本文档主要针对华为数据卡/模块设备基于Linux系统的集成开发活动进行相关的指导说明。主要面向第三方软件开发商的集成开发人员。

2 范围

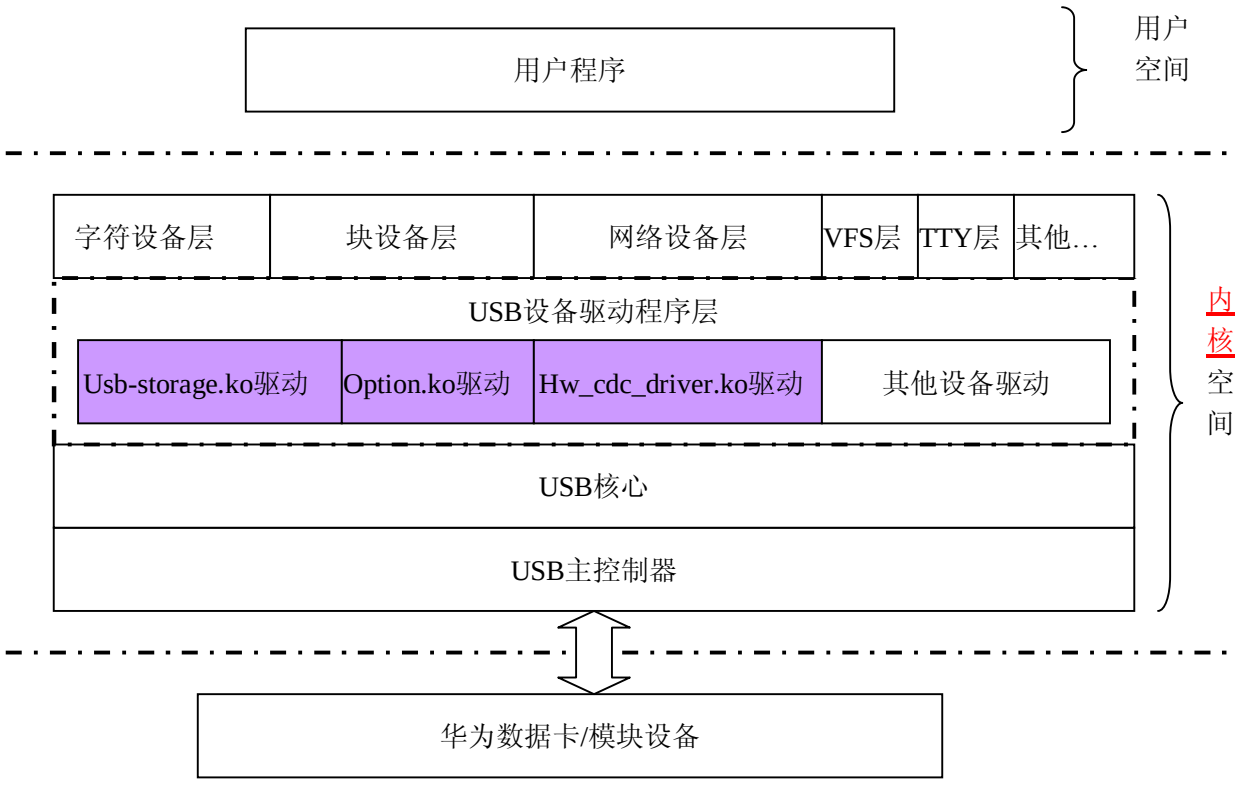
本文档主要说明了在Linux系统上支持华为数据卡/模块设备的相关开发工作，及其注意事项。

3 总体概述

3.1 Linux 内核支持华为数据卡/模块的驱动架构

对于华为数据卡/模块和模块来说，不管是什么类型的接口，其上报的复合设备接口，最终均是加载为Linux USB设备驱动层中的相关驱动模块。

对于华为数据卡/模块设备的支持，Linux系统相关的的USB驱动架构如下图所示：



如上图所示，在Linux系统中的USB驱动架构中，跟华为数据卡/模块设备相关的驱动模

块，主要是USB设备驱动程序层中的usb-storage驱动模块，option串口驱动模块，hw_cdc_driver网卡驱动模块。其中，hw_cdc_driver网卡驱动模块用于适配华为数据卡/模块的NDIS网口，该驱动模块目前还没有合入Linux内核版本中；而usb-storage驱动模块和option串口驱动模块，则是Linux内核版本中内置的驱动模块。

3.2 Linux 数据卡/模块支持的集成流程

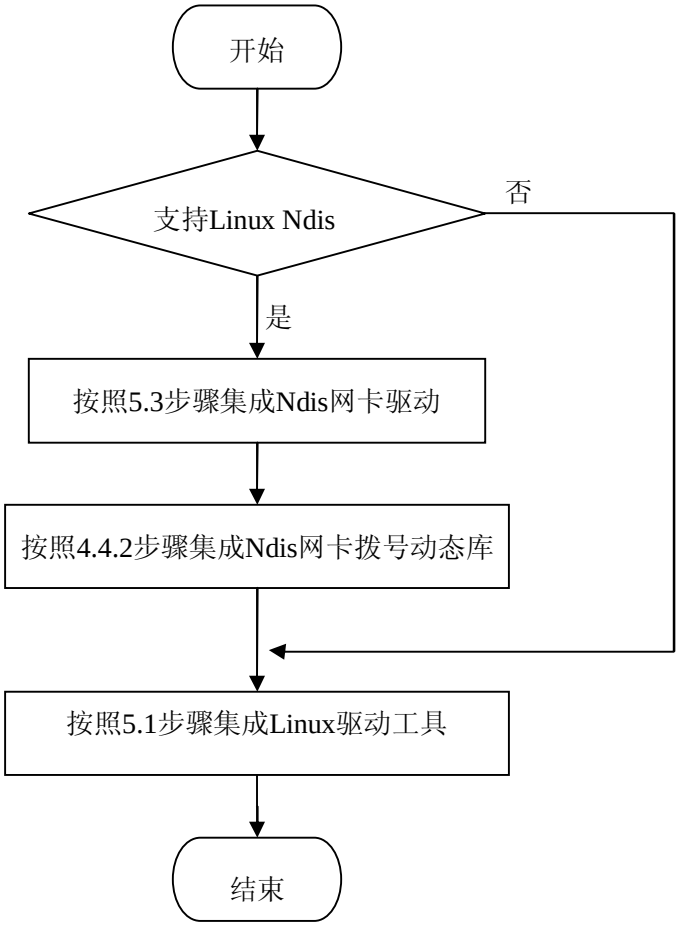
集成之前，请先确认需要在第三方Linux后台软件中支持华为的数据卡/模块设备，还是需要在Linux系统上直接支持华为的数据卡/模块设备。如果是想在第三方后台软件中支持，则按照3.2.1说明的集成流程进行集成；如果想直接在Linux系统上支持，则按照3.2.2说明的集成流程进行集成。

支持Linux Ndis的判断方法：在Windows系统上查找华为数据卡/模块的Modem端口的Product ID， 如果为0x14AC，则说明需要支持；否则不需要支持。

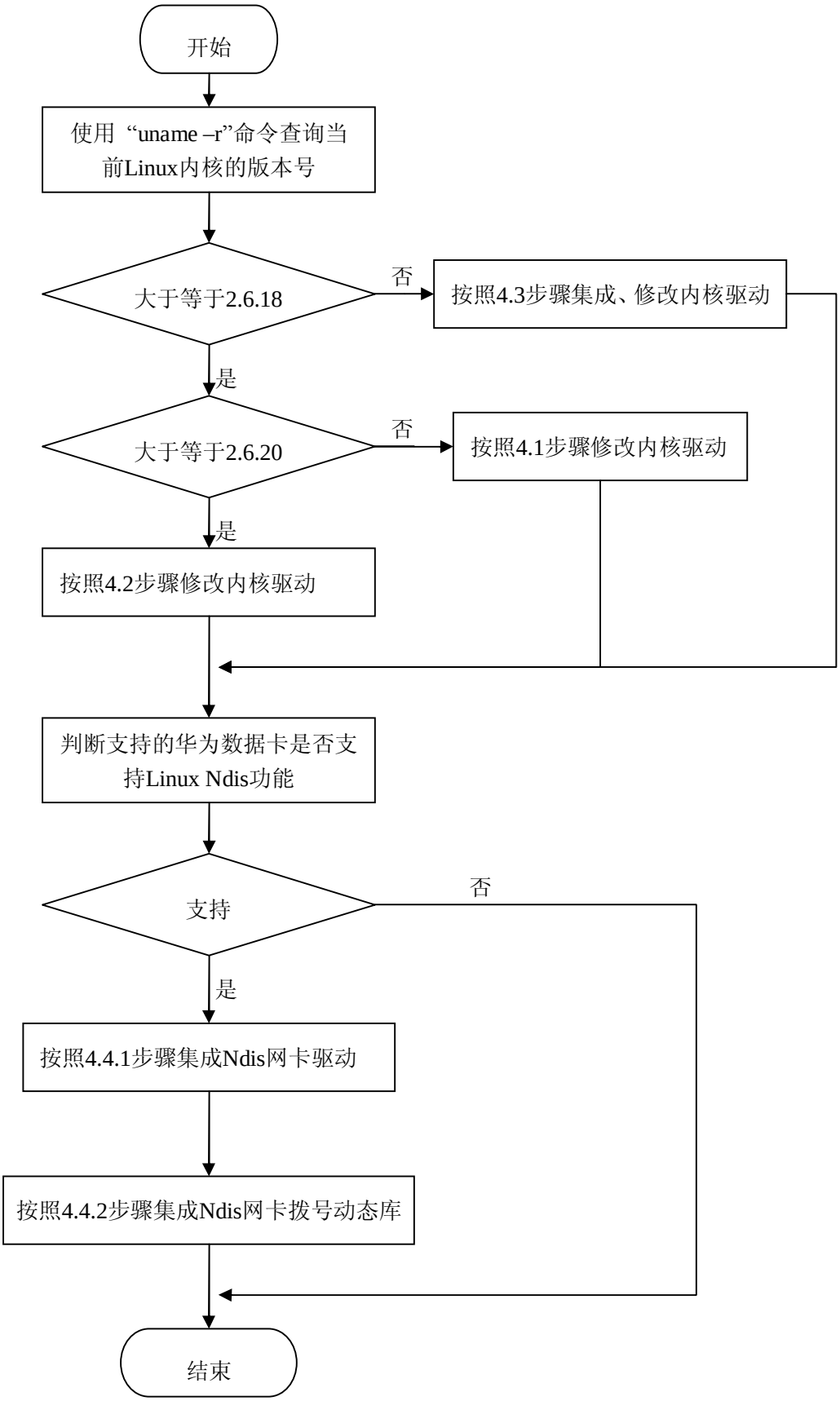
注意：当前仅有PID 14AC支持NIDS，之后华为有可能新增其他PID支持NDIS，若有更新，会刷新该文档，请留意和华为支持人员获取该文档得最新版本。

若集成过程中发现问题，也请及时和华为支持人员取得联系。

3.2.1 第三方后台集成流程：



3.2.2 Linux 内核集成流程


开始
使用 “uname –r”命令查询当前Linux内核的版本号
大于等于2.6.18
否
按照4.3步骤集成、修改内核驱动
是
大于等于2.6.20
否
按照4.1步骤修改内核驱动
是
按照4.2步骤修改内核驱动
判断支持的华为数据卡是否支持Linux Ndis功能
支持
否
是
按照4.4.1步骤集成Ndis网卡驱动
按照4.4.2步骤集成Ndis网卡拨号动态库
结束

4 Linux 驱动模块集成方案

本节主要说明通过修改Linux内核的方案来支持华为数据卡/模块的集成方案。而Linux内核源代码可以从下面的网站链接上获取：

http://www.kernel.org/pub/linux/kernel/v2.6/

4.1 针对 2.6.18 或是 2.6.19 内核版本的集成步骤

4.1.1 usb-storage 驱动模块的集成步骤

涉及的Linux内核源码文件有：

\drivers\usb\storage\libusual.c

\drivers\usb\storage\usb.c

\drivers\usb\storage\unusual_devs.h

\drivers\usb\storage\initializers.c

\drivers\usb\storage\initializers.h

具体的修改步骤如下：

1、 修改libusual.c文件

a) 在文件中“#define UNUSUAL_DEV(...)”宏定义后面增加一个新的宏定义：

```
#define UNUSUAL_DEV_HW(id_vendor, id_product, bcdDeviceMin, bcdDeviceMax, \
        bIntClass, vendorName, productName, useProtocol, useTransport, \
        initFunction, flags) \
        { .match_flags = (USB_DEVICE_ID_MATCH_DEVICE_AND_VERSION |
USB_DEVICE_ID_MATCH_INT_CLASS), \
        .idVendor = (id_vendor), .idProduct = (id_product), \
        .bcdDevice_lo = (bcdDeviceMin), .bcdDevice_hi = (bcdDeviceMax), \
        .bInterfaceClass = (bIntClass), \
        .driver_info = (flags)|(USB_US_TYPE_STOR<<24)}
```

b) 在文件中“#undef UNUSUAL_DEV”宏定义后面增加下面的宏定义：

```
#undef UNUSUAL_DEV_HW
```

2、 修改usb.c文件

a) 在第一个“#define UNUSUAL_DEV(...)”宏定义和“static struct usb_device_id storage_usb_ids [] = ...”之间，增加如下的宏定义：

```
#define UNUSUAL_DEV_HW(id_vendor, id_product, bcdDeviceMin, bcdDeviceMax, \
    bIntClass, vendorName, productName, useProtocol, useTransport, \
    initFunction, flags) \
    { .match_flags = (USB_DEVICE_ID_MATCH_DEVICE_AND_VERSION |
USB_DEVICE_ID_MATCH_INT_CLASS), \
    .idVendor = (id_vendor), .idProduct = (id_product), \
    .bcdDevice_lo = (bcdDeviceMin), .bcdDevice_hi = (bcdDeviceMax), \
    .bInterfaceClass = (bIntClass), \
    .driver_info = (flags)|(USB_US_TYPE_STOR<<24)}
```

b) 在“static struct usb_device_id storage_usb_ids [] = ...”语句块中的宏定义“#undef UNUSUAL_DEV”之后，增加如下的宏定义：

```
#undef UNUSUAL_DEV_HW
```

c) 在第二个宏定义“#define UNUSUAL_DEV(...)”和“static struct us_unusual_dev us_unusual_dev_list [] = ...”语句之间，增加如下宏定义：

```
#define UNUSUAL_DEV_HW(id_vendor, id_product, bcdDeviceMin, bcdDeviceMax, \
    bInterfaceClass, vendor_Name, product_Name, use_Protocol, use_Transport, \
    init_Function, flags) \
    { \
    .vendorName = vendor_Name,      \
    .productName = product_Name,    \
    .useProtocol = use_Protocol,  \
    .useTransport = use_Transport,      \
    .initFunction = init_Function, \
    }
```

d) 在“static struct us_unusual_dev us_unusual_dev_list [] = ...”语句块中的“#undef UNUSUAL_DEV”宏定义之后，增加如下宏定义：

"#undef UNUSUAL_DEV_HW

3、 修改unusual_devs.h文件

unusual_devs.h文件是用于声明需要特殊处理的设备，其顺序为根据Vendor ID进行升序排序的。因此需要根据其排列顺序，在文件中找出华为Vendor ID 0x12D1处于中间的两个相邻的声明位置，并在该位置加入对应的声明：

a) 先增加如下的声明：

```
UNUSUAL_DEV_HW( 0x12d1, 0x1001, 0x0000, 0x0000,
        0x08,
        "HUAWEI",
        "HUAWEI MOBILE Mass Storage",
        US_SC_DEVICE, US_PR_DEVICE, usb_stor_huawei_init,
        0),
```

注意：以上的示例，仅是加入了Product ID为0x1001的华为数据卡/模块设备的支持。根据以上的声明格式，增加下表所有Product ID的声明：

0x1003、0x1004、0x1401－0x143F

b) 接着增加下面的声明：

```
UNUSUAL_DEV_HW( 0x12d1, 0x1446, 0x0000, 0x0000,
        0x08,
        "HUAWEI",
        "HUAWEI MOBILE Mass Storage",
        US_SC_DEVICE, US_PR_DEVICE, usb_stor_huawei_scsi_init,
        0),
```

4、 修改initializers.c文件

a) 先在文件中增加usb_stor_huawei_init函数的定义：

```
int usb_stor_huawei_init(struct us_data *us)
{
        int result = 0;
        us->iobuf[0] = 0x0;
        result = usb_stor_control_msg(us, us->send_ctrl_pipe,
```

```
                USB_REQ_SET_FEATURE, USB_TYPE_STANDARD |
USB_RECIP_DEVICE,
                0x01, 0x0, us->iobuf, 0x0, 1000);
        US_DEBUGP("usb_control_msg performing result is %d\n", result);
        return 0;
    }
```

b) 接着在文件中增加usb_stor_huawei_scsi_init函数的定义：

```
    int usb_stor_huawei_scsi_init(struct us_data *us)
    {
        int result = 0;
        int act_len = 0;
        unsigned char cmd[32] = {0x55, 0x53, 0x42, 0x43, 0xEE, 0x00, 0x00, 0x00,
                                 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x11,
                                 0x06, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
        result = usb_stor_bulk_transfer_buf(us, us->send_bulk_pipe, cmd, 31, &act_len);
        US_DEBUGP("usb_stor_bulk_transfer_buf performing result is %d, transfer the
actual length=%d\n", result, act_len);
        return result;
    }
```

5、 修改initializers.h文件

a) 先在文件中增加usb_stor_huawei_init函数的声明：

```
int usb_stor_huawei_init(struct us_data *us);
```

b) 接着在文件中增加usb_stor_huawei_scsi_init函数的声明：

```
int usb_stor_huawei_scsi_init(struct us_data *us);
```

4.1.2 Linux 串口驱动模块的集成步骤

本集成方案涉及到的Linux内核源码文件有：

\drivers\usb\serial\option.c

\drivers\usb\serial\pl2303.c

其具体的修改步骤如下：

1、 修改option.c文件

a) 在“static struct usb_device_id option_ids[] = {”语句之前增加下面的宏定义：

```
#define USB_DEVICE_AND_INTERFACE_INFO(vend, prod, cl, sc, pr) \
        .match_flags = USB_DEVICE_ID_MATCH_INT_INFO |
USB_DEVICE_ID_MATCH_DEVICE, \
        .idVendor = (vend), \
        .idProduct = (prod), \
        .bInterfaceClass = (cl), \
        .bInterfaceSubClass = (sc), \
        .bInterfaceProtocol = (pr)
```

b) 在/* Vendor and product IDs */稍后的位置上，增加对华为数据卡/模块设备的Vendor ID和Product ID的声明，如：

```
#define HUAWEI_VENDOR_ID            0x12D1
#define HUAWEI_PRODUCT_E1001        0x1001
#define HUAWEI_PRODUCT_E1003        0x1003
#define HUAWEI_PRODUCT_E1004        0x1004
#define HUAWEI_PRODUCT_E1401        0x1401
#define HUAWEI_PRODUCT_E1402        0x1402
…
#define HUAWEI_PRODUCT_E143E        0x143E
#define HUAWEI_PRODUCT_E143F        0x143F
…
#define HUAWEI_PRODUCT_E1440        0x1440
#define HUAWEI_PRODUCT_E1441        0x1441
#define HUAWEI_PRODUCT_E1448        0x1448
#define HUAWEI_PRODUCT_E14AC        0x14AC
```

注意：该示例仅包含华为的部分Product ID，如果还需要支持华为新的数据卡/模块

的Product ID，则需要增加对应的定义声明，包括：0x1401—0x143F，0x1440，0x1441，0x1448，0x14AC

c) 在“static struct usb_device_id option_ids[] = {”语句块中，增加对华为数据卡/模块的声明，例如对于Product ID为0x1003的支持声明：

```
static struct usb_device_id option_ids[] = {
    ...
    { USB_DEVICE_AND_INTERFACE_INFO(HUAWEI_VENDOR_ID,
HUAWEI_PRODUCT_E1003, 0xff, 0xff, 0xff) },
    ...
    { } /* Terminating entry */
};
```

d) 在“static struct usb_device_id option_ids1[] = {”语句块中，增加对华为数据卡/模块的声明，例如对于Product ID为0x1003的支持声明：

```
static struct usb_device_id option_ids1[] = {
    ...
    { USB_DEVICE_AND_INTERFACE_INFO(HUAWEI_VENDOR_ID,
HUAWEI_PRODUCT_E1003, 0xff, 0xff, 0xff) },
    ...
    { } /* Terminating entry */
};
```

e) 删除“static struct usb_device_id option_ids[] = {”和“static struct usb_device_id option_ids1[] = {”中，使用USB_DEVICE宏定义声明的华为Product ID的支持声明，如：

```
{ USB_DEVICE(HUAWEI_VENDOR_ID, HUAWEI_PRODUCT_E600) },
```

2、 修改pl2303.c文件

屏蔽或是删除“static struct usb_device_id id_table [] = {”语句块中的关于华为数据卡/模块设备声明的语句：

```
{ USB_DEVICE(HUAWEI_VENDOR_ID, HUAWEI_PRODUCT_ID) },
```

4.2 针对 2.6.20 或是更高的内核版本的集成方案说明

4.2.1 usb-storage 驱动模块的集成步骤

涉及的Linux内核源码文件有：

\drivers\usb\storage\unusual_devs.h

\drivers\usb\storage\initializers.c

\drivers\usb\storage\initializers.h

具体的修改步骤如下：

1、 修改unusual_devs.h文件

a) 把文件中原来的声明：

UNUSUAL_DEV(0x12d1, 0x1003, 0x0000, 0xffff,

"HUAWEI",

"E220 USB-UMTS Install",

US_SC_DEVICE, US_PR_DEVICE, NULL,

US_FL_IGNORE_DEVICE),

修改为下面的声明：

UNUSUAL_DEV(0x12d1, 0x1003, 0x0000, 0x0000,

"HUAWEI",

"HUAWEI MOBILE Mass Storage",

US_SC_DEVICE, US_PR_DEVICE, usb_stor_huawei_init,

0),

注意：在2.6.24及更高的Linux内核版本的unusual_devs.h文件中，其声明的格式已经修改正确了，那么可以跳过本步骤。

b) 在对应的位置上，增加新的Product ID声明语句。建议根据以上的声明格式，增加下表所有Product ID的声明：

0x1001、0x1003、0x1004、0x1401－0x143F

c) 接着在0x143F的声明语句之后，增加下面的声明：

UNUSUAL_DEV (0x12d1, 0x1446, 0x0000, 0x0000,

"HUAWEI",

```
        "HUAWEI MOBILE Mass Storage",
        US_SC_DEVICE, US_PR_DEVICE, usb_stor_huawei_scsi_init,
        0),
```

2、 修改initializers.c文件

a) 首先在文件中增加usb_stor_huawei_init函数的定义：

```
int usb_stor_huawei_init(struct us_data *us)
{
    int result = 0;
    us->iobuf[0] = 0x0;
    result = usb_stor_control_msg(us, us->send_ctrl_pipe,
        USB_REQ_SET_FEATURE, USB_TYPE_STANDARD |
USB_RECIP_DEVICE,
        0x01, 0x0, us->iobuf, 0x0, 1000);
    US_DEBUGP("usb_control_msg performing result is %d\n", result);
    return 0;
}
```

b) 接着在文件中增加usb_stor_huawei_scsi_init函数的定义：

```
int usb_stor_huawei_scsi_init(struct us_data *us)
{
    int result = 0;
    int act_len = 0;
    unsigned char cmd[32] = {0x55, 0x53, 0x42, 0x43, 0xEE, 0x00, 0x00, 0x00,
                             0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x11,
                             0x06, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                             0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
    result = usb_stor_bulk_transfer_buf (us, us->send_bulk_pipe, cmd, 31, &act_len);
    US_DEBUGP("usb_stor_bulk_transfer_buf performing result is %d, transfer the
actual length=%d\n", result, act_len);
    return result;
```

}

3、 修改initializers.h文件

a) 首先在文件中usb_stor_huawei_init函数的声明：

int usb_stor_huawei_init(struct us_data *us);

b) 接着在文件中增加usb_stor_huawei_scsi_init函数的声明：

int usb_stor_huawei_scsi_init(struct us_data *us);

4.2.2 Linux 串口驱动模块的集成步骤

本集成方案涉及到的Linux内核源码文件有：

\drivers\usb\serial\option.c

\drivers\usb\serial\pl2303.c

其具体的修改步骤如下：

1、 修改option.c文件

a) 在“static struct usb_device_id option_ids[] = {”语句之前增加下面的宏定义：

```
#define USB_DEVICE_AND_INTERFACE_INFO(vend, prod, cl, sc, pr) \
        .match_flags = USB_DEVICE_ID_MATCH_INT_INFO |
USB_DEVICE_ID_MATCH_DEVICE, \
        .idVendor = (vend), \
        .idProduct = (prod), \
        .bInterfaceClass = (cl), \
        .bInterfaceSubClass = (sc), \
        .bInterfaceProtocol = (pr)
```

b) 在/* Vendor and product IDs */稍后的位置上，增加对华为数据卡/模块设备的Vendor ID和Product ID的声明，如：

```
#define HUAWEI_VENDOR_ID            0x12D1
#define HUAWEI_PRODUCT_E1001        0x1001
#define HUAWEI_PRODUCT_E1003        0x1003
#define HUAWEI_PRODUCT_E1004        0x1004
#define HUAWEI_PRODUCT_E1401        0x1401
```

```
#define HUAWEI_PRODUCT_E1402            0x1402
…
#define HUAWEI_PRODUCT_E143E            0x143E
#define HUAWEI_PRODUCT_E143F            0x143F
…
#define HUAWEI_PRODUCT_E1440            0x1440
#define HUAWEI_PRODUCT_E1441            0x1441
#define HUAWEI_PRODUCT_E1448            0x1448
#define HUAWEI_PRODUCT_E14AC            0x14AC
```

注意：该示例仅包含华为的部分Product ID，如果还需要支持华为新的数据卡/模块的Product ID，则需要增加对应的定义声明。包括：0x1401—0x143F，0x1440，0x1441，0x1448，0x14AC

c) 在“static struct usb_device_id option_ids[] = {”语句块中，增加对华为数据卡/模块的声明，例如对于Product ID为0x1003的支持声明：

```
static struct usb_device_id option_ids[] = {
    ...
    { USB_DEVICE_AND_INTERFACE_INFO(HUAWEI_VENDOR_ID,
HUAWEI_PRODUCT_E1003, 0xff, 0xff, 0xff) },
    ...
    { } /* Terminating entry */
};
```

d) 在“static struct usb_device_id option_ids1[] = {”语句块中，增加对华为数据卡/模块的声明，例如对于Product ID为0x1003的支持声明：

```
static struct usb_device_id option_ids1[] = {
    ...
    { USB_DEVICE_AND_INTERFACE_INFO(HUAWEI_VENDOR_ID,
HUAWEI_PRODUCT_E1003, 0xff, 0xff, 0xff) },
    ...
    { } /* Terminating entry */
```

};

注意：该变量如果在更高的Linux内核版本的option.c文件中已经不存在了，在可以跳过该步骤。

e) 删除“static struct usb_device_id option_ids[] = {”和“static struct usb_device_id option_ids1[] = {”中，使用USB_DEVICE宏定义声明的华为Product ID的支持声明，如：

{ USB_DEVICE(HUAWEI_VENDOR_ID, HUAWEI_PRODUCT_E600) },

2、 修改pl2303.c文件

屏蔽或是删除“static struct usb_device_id id_table [] = {”语句块中的关于华为数据卡/模块设备声明的语句：

{ USB_DEVICE(HUAWEI_VENDOR_ID, HUAWEI_PRODUCT_ID) },

4.3 针对低于 2.6.18 的内核版本的集成步骤

低于2.6.18的Linux内核版本，其内核驱动不支持华为的数据卡/模块设备。对于这些内核版本，华为仅能提供一个概要的集成方案说明，而且该方案没有经过实际验证，因此在集成时，可能会存在一些语法、接口或是结构的兼容性问题，这些都需要开发商自己进行解决，华为无法提供技术支持。

对于这些内核版本，其集成思路如下：

1) 从2.6.18或是更高的内核版本的源码包中获取option.c源码文件，拷贝到目标版本的源码drivers/usb/serial目录下。

2) 修改serial目录下的Kconfig和Makefile文件。具体操作请参考4.4节中集成hw_cdc_driver.c文件的操作步骤。完成option.c的集成。

3) 参考4.1的集成方案说明，修改对应的文件。完成集成。

4.4 hw_cdc_driver 网卡驱动模块的集成方案

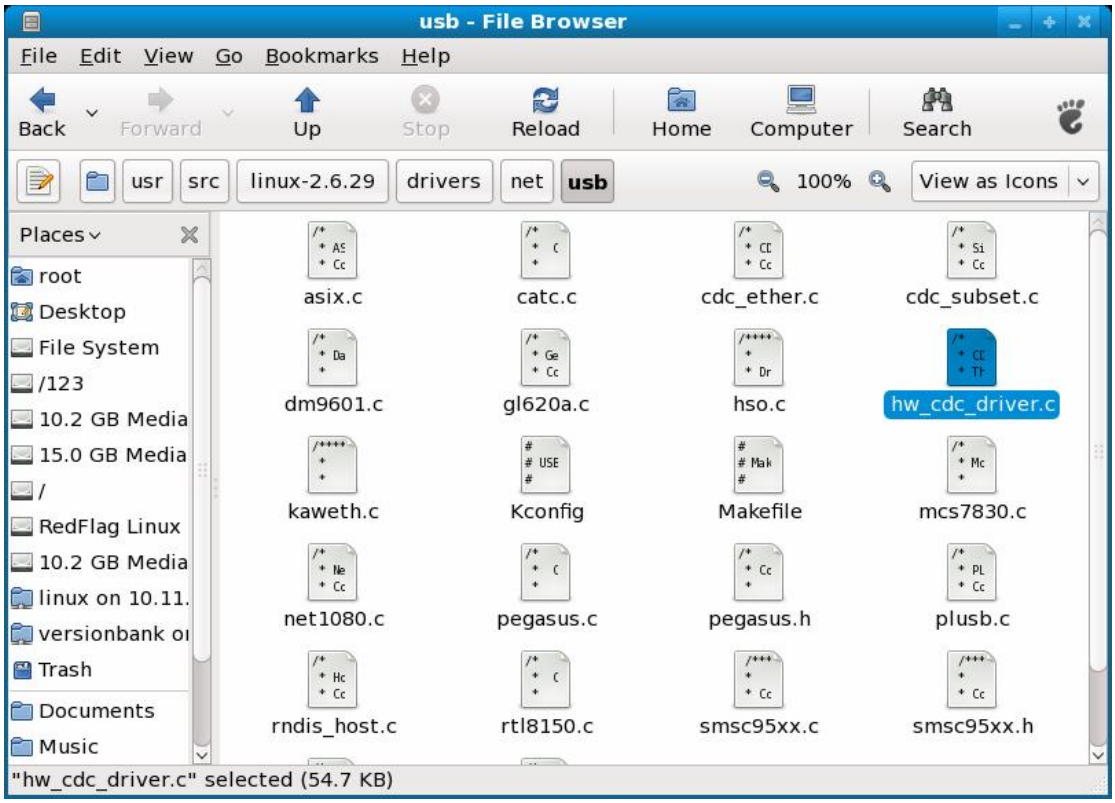
对于华为的Linux Ndis网卡集成，必须包括网卡驱动集成和网卡拨号动态库集成。

4.4.1 hw_cdc_driver 网卡驱动集成说明

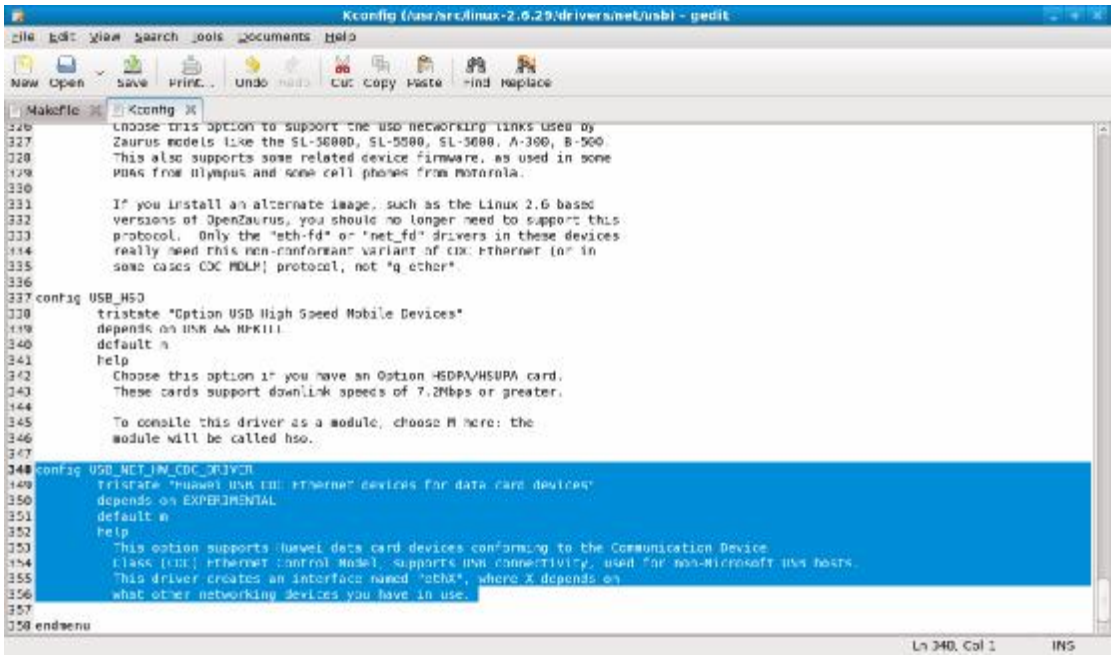
支持华为数据卡/模块的Linux Ndis网卡驱动，是华为基于Linux内核的usbnet和cdc_ether

自己开发的一个内核驱动模块，该模块的源码文件名称为hw_cdc_driver.c。其集成步骤如下：

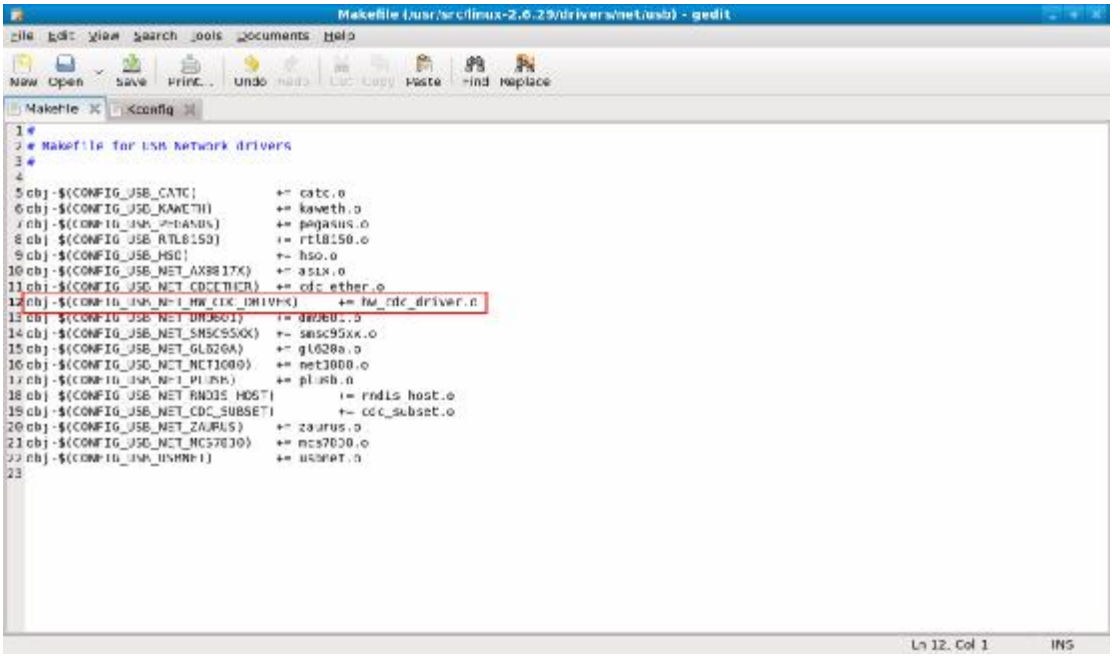
1） 把华为提供的 Linux 网卡驱动的源码文件 hw_cdc_driver.c 文件拷贝到内核源码的 drivers/net/usb/目录下。



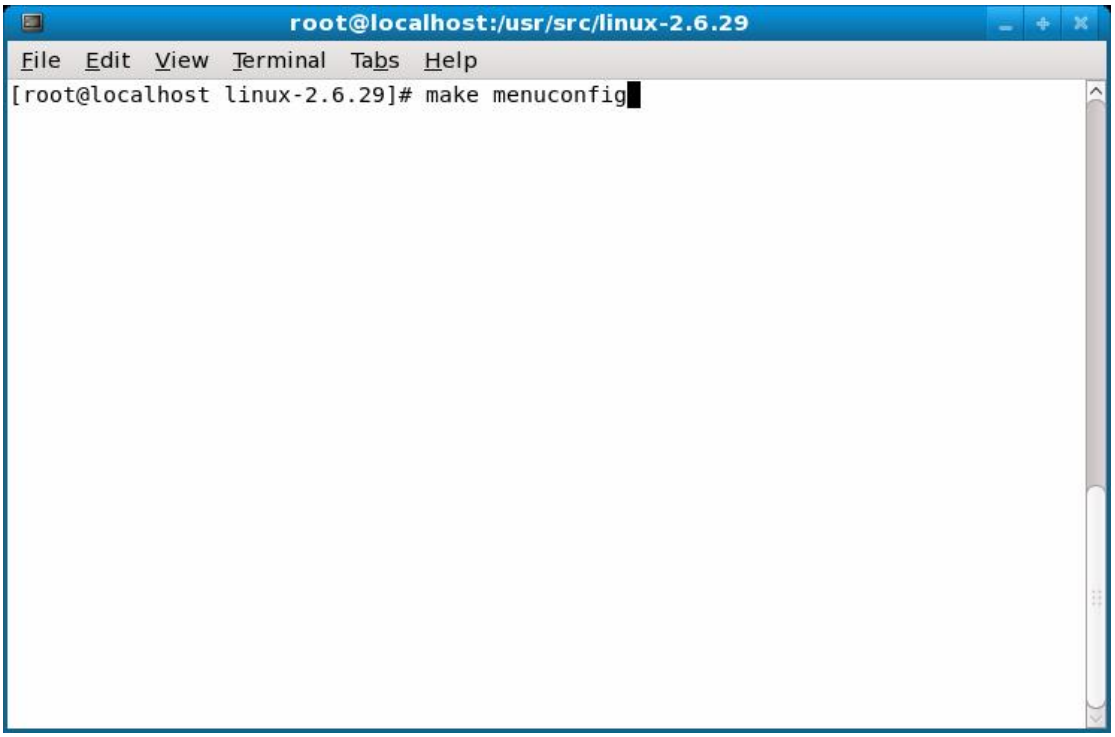
2） 编辑 drivers/net/usb 目录下的 Kconfig 文件，在所示位置增加 config USB_NET_HW_CDC_DRIVER 的配置项内容，如下图所示。



3） 编辑 drivers/net/usb 目录下的 Makefile 文件，增加如下图所示的内容。

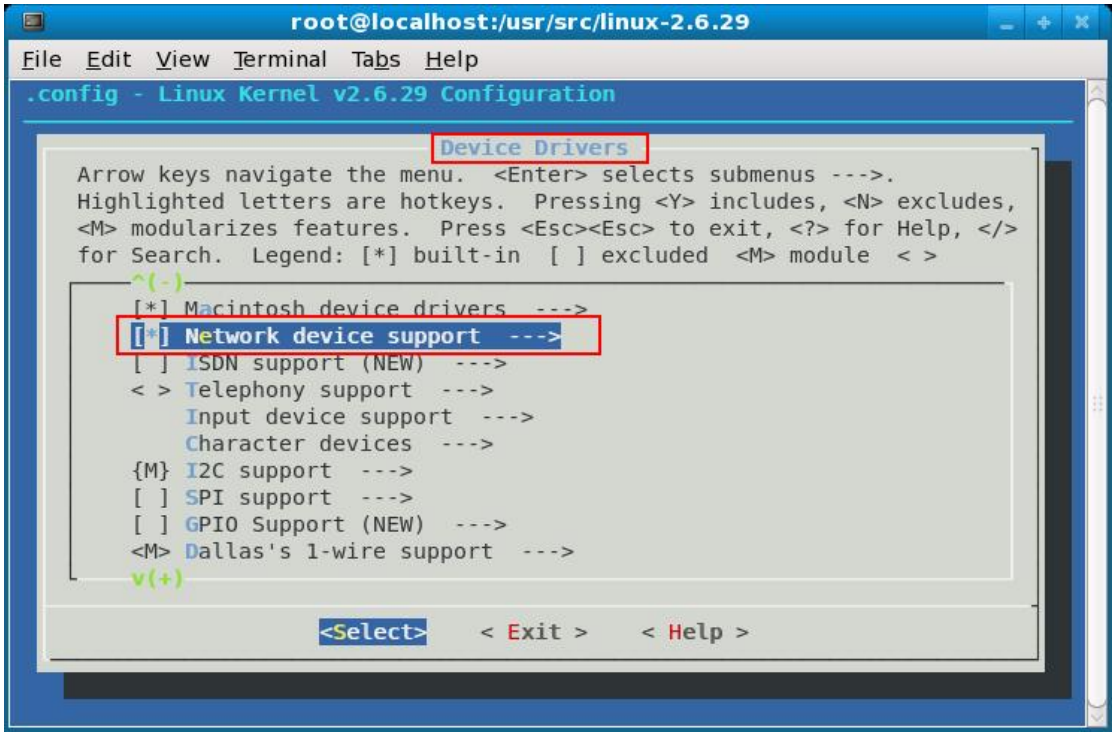


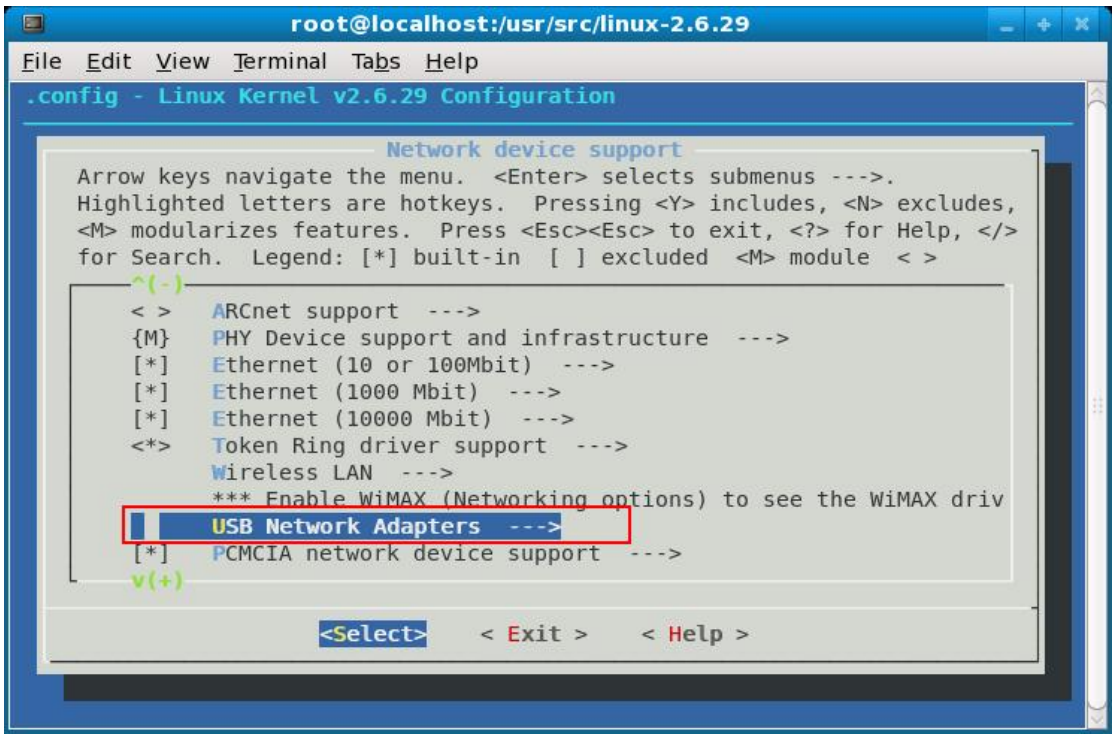
4） 打开 Terminal 工具，进入/usr/src/linux-2.6.29 目录，然后执行 make menuconfig 命令。

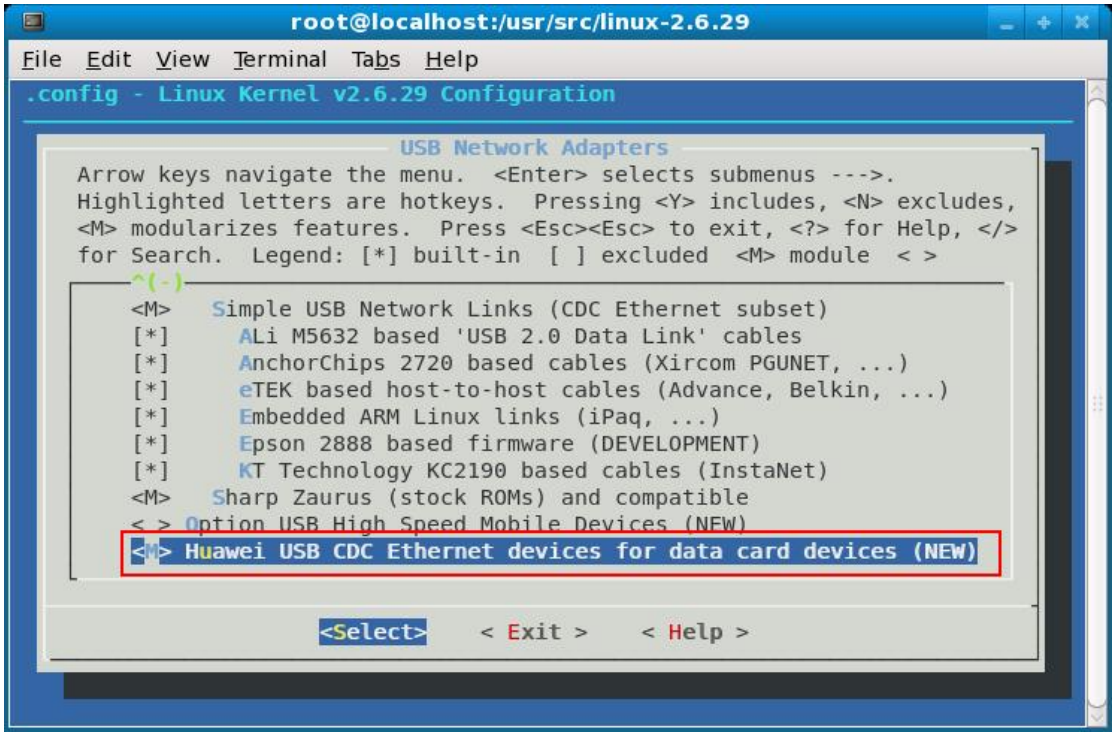


5） 按照如下示意图的步骤，检查 Linux Ndis 驱动的编译选项是否已经在配置菜单中了。

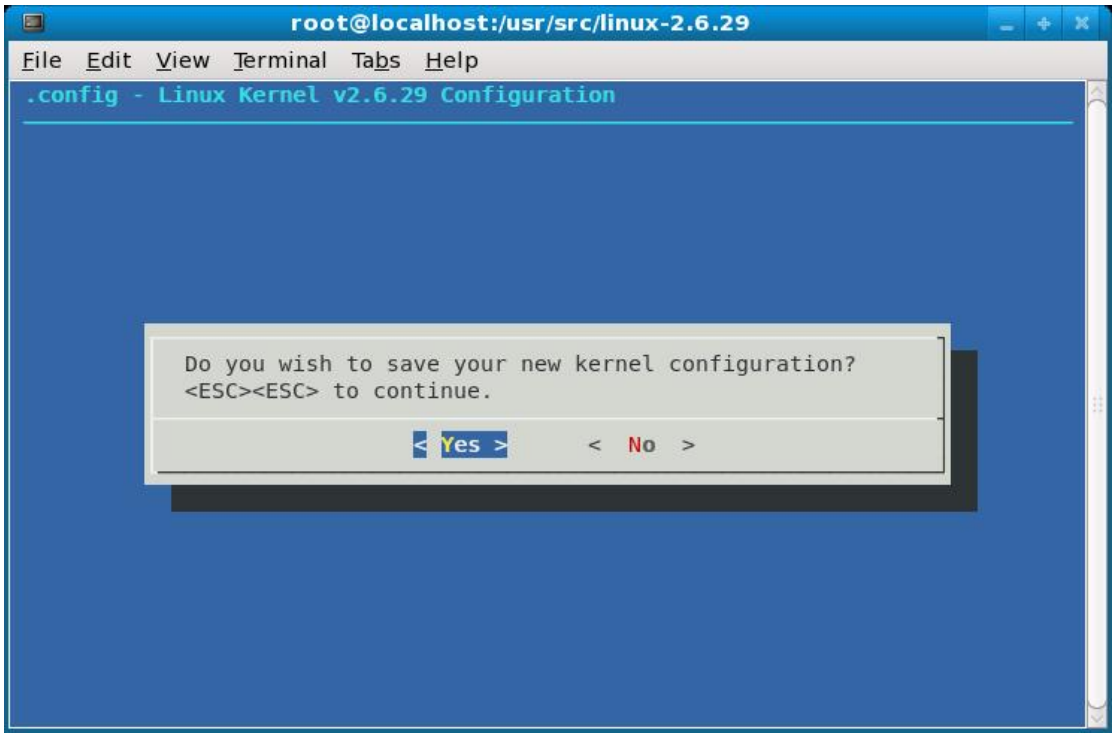
Device Drivers- **à** Network device support- **à** USB Network Adapters


root@localhost:/usr/src/linux-2.6.29
File Edit View Terminal Tabs Help
.config - Linux Kernel v2.6.29 Configuration
Device Drivers
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >
[*] Macintosh device drivers --->
[*] Network device support --->
[ ] ISDN support (NEW) --->
< > Telephony support --->
Input device support --->
Character devices --->
{M} I2C support --->
[ ] SPI support --->
[ ] GPIO Support (NEW) --->
<M> Dallas's 1-wire support --->
<Select> < Exit > < Help >


root@localhost:/usr/src/linux-2.6.29
File Edit View Terminal Tabs Help
.config - Linux Kernel v2.6.29 Configuration
Network device support
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >
< > ARCnet support --->
{M} PHY Device support and infrastructure --->
[*] Ethernet (10 or 100Mbit) --->
[*] Ethernet (1000 Mbit) --->
[*] Ethernet (10000 Mbit) --->
<*> Token Ring driver support --->
Wireless LAN --->
*** Enable WiMAX (Networking options) to see the WiMAX driv
USB Network Adapters --->
[*] PCMCIA network device support --->
<Select> < Exit > < Help >



6） 通过选择<Exit>按钮，退出各个配置界面。最后在保存配置界面中，选择”Yes”选项并退出。



7） 完成配置后，即可运行 make 命令，开始编译修改后的内核版本。

4.4.2 Linux 网卡设备拨号集成

对于华为数据卡/模块在Linux系统上的网卡拨号功能，可以直接使用华为提供的c语言动态库lib_ndis.so提供的api来实现。

该动态库的头文件为：ndis_qmi_service.h

该动态库提供的API接口如下：

3、 char *get_lib_version();

该接口用于获取动态库的当前版本号。

4、 int ndis_connect(char *apn, char *username, char *passwd, char *auth);

该接口为网卡设备的数据业务拨号链接函数，其四个参数均为字符串，其中，

- **第一个参数：**表示 APN，默认为空字符串
- **第二个参数：**表示用户名，默认为空字符串
- **第三个参数：**表示秘密，默认为空字符串
- **第四个参数：**表示鉴权协议，"1"表示 PAP，"2"表示 CHAP，"3"表示 PAP 和 CHAP 一起使用，默认为空字符串。

5、 int ndis_disconnect();

该接口为网卡设备的数据业务拨号链接断开函数。

- int ndis_get_status();

该接口用于获取网卡设备数据业务的链接状态。

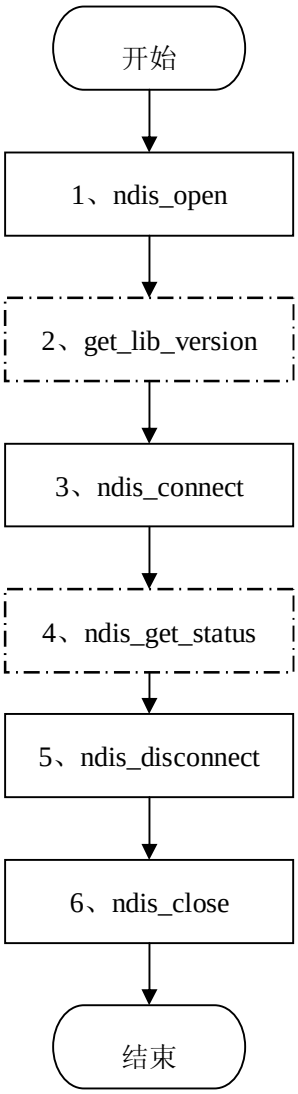
- boolean ndis_open();

该接口用于打开网卡设备的端口文件。

- boolean ndis_close();

该接口用于关闭已经被打开的网卡设备的端口文件。

一般来说，华为数据卡/模块在Linux系统上的网卡数据业务拨号处理流程如下：（2、4步骤可选）

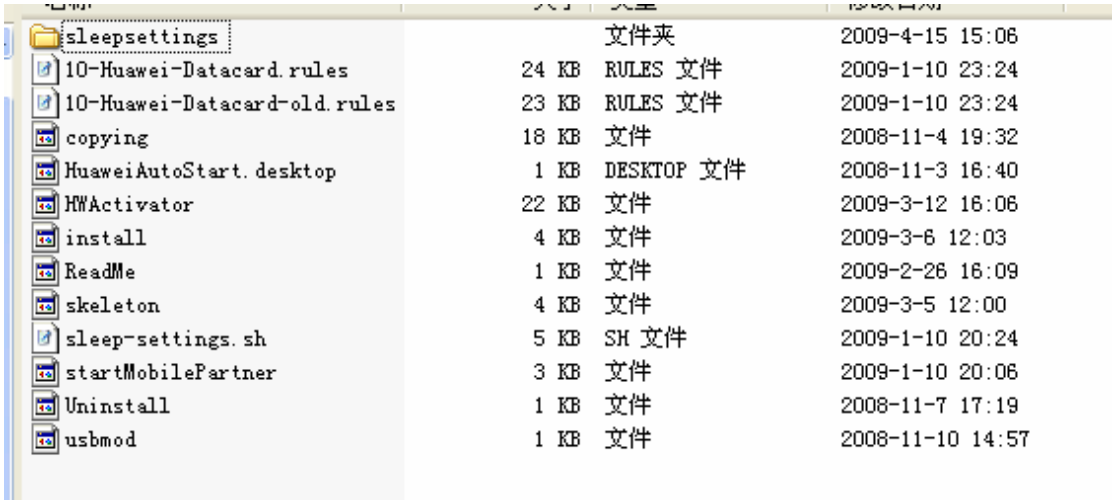


5 Linux 驱动工具的集成方案

5.1 Linux 驱动工具介绍

Linux驱动工具，并不是内核驱动程序，而事实上是Linux应用层的程序和配置集，由一个可执行文件和若干个配置脚本文件组成，主要的文件有：

- **可执行文件**：HWActivator
- **脚本文件**：10-Huawei-Datacard.rules、10-Huawei-Datacard-old.rules、HuaweiAutoStart.desktop、skeleton、startMobilePartner、usbmod、休眠/待机事件通知文件等。



sleepsettings		文件夹	2009-4-15 15:06
10-Huawei-Datacard.rules	24 KB	RULES 文件	2009-1-10 23:24
10-Huawei-Datacard-old.rules	23 KB	RULES 文件	2009-1-10 23:24
copying	18 KB	文件	2008-11-4 19:32
HuaweiAutoStart.desktop	1 KB	DESKTOP 文件	2008-11-3 16:40
HWActivator	22 KB	文件	2009-3-12 16:06
install	4 KB	文件	2009-3-6 12:03
ReadMe	1 KB	文件	2009-2-26 16:09
skeleton	4 KB	文件	2009-3-5 12:00
sleep-settings.sh	5 KB	SH 文件	2009-1-10 20:24
startMobilePartner	3 KB	文件	2009-1-10 20:06
Uninstall	1 KB	文件	2008-11-7 17:19
usbmod	1 KB	文件	2008-11-10 14:57

在整个Linux驱动工具中，可执行文件HWActivator是核心，并被实现为一个Linux的精灵进程，其主要目的是用于确保华为数据卡/模块设备在连接到Linux系统后，能够上报一个Linux当前内核驱动模块能够支持的PID端口模式，使用数据卡/模块设备能够在Linux系统上正常使用。

Linux驱动工具最新版本的主要功能有：

- PID端口模式管理功能
- 端口名称固化功能

5.2 Linux 驱动工具的集成操作

第三方后台软件开发商可以把华为的Linux驱动工具集成到他们的Linux后台软件安装包中，以便在通用的Linux系统上配套华为的数据卡/模块设备使用。其具体的集成如下：

1、在后台软件的安装目录下，生成一个driver的子目录。

2、后台软件安装时，把华为Linux驱动工具的所有文件都拷贝到该driver目录下。

3、调用driver目录下的install安装脚本文件，并传入driver目录所在的路径，例如，其运行的命令格式如下所示，即可完成华为Linux驱动工具的安装。

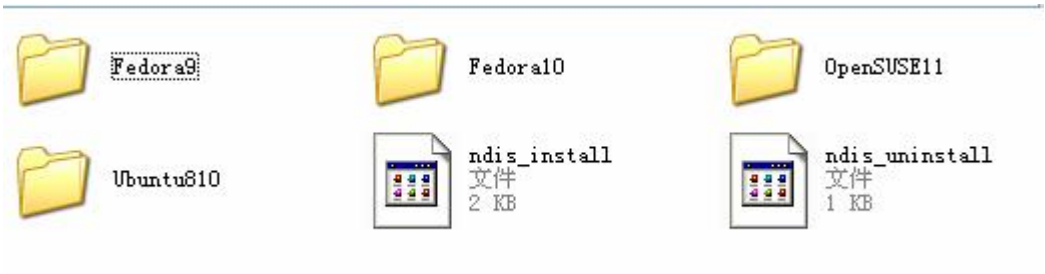
$DASHBOARD_PATH/driver/install $DASHBOARD_PATH

4、 在完成Linux驱动工具之后，当华为数据卡/模块连接到Linux系统之后，第三方后台软件需要使用如下名称的端口来跟华为的数据卡/模块进行交互：

- ttyUSB_utps_modem：后台软件使用该端口名称来进行modem拨号。
- ttyUSB_utps_pcui：后台软件通过该端口名称向华为数据卡/模块下发AT命令。

5.3 第三方后台的 Ndis 驱动集成步骤

对于第三方后台软件，华为直接提供已经编译好的Linux Ndis驱动包进行集成。Linux Ndis的驱动包包含的文件如下所示：



其集成步骤如下：

1、在后台软件的安装目录下，生成一个linux_ndis的子目录。

2、后台软件安装时，把华为Linux Ndis驱动的所有文件夹及文件都拷贝到该linux_ndis目录下。

3、调用linux_ndis目录下的ndis_install安装脚本文件，并传入linux_ndis目录所在的路径，例如，其运行的命令格式如下所示，即可完成华为Linux Ndis驱动的安装。

$DASHBOARD_PATH/linux_ndis/ndis_install $DASHBOARD_PATH/linux_ndis

注意：集成Linux Ndis驱动时，后台软件安装时，先执行Linux Ndis驱动的安装脚本ndis_install；然后在执行Linux驱动工具的安装脚本install。

6 附录

6.1 获取单板当前的端口映射情况信息

- 单板当前的端口映射信息，是存放在/proc/bus/usb目录下的devices文件中。在一些Linux系统中，如Debian，该devices文件默认就存在；但是在一些Linux系统中，如Ubuntu，该devices文件默认是不存在的。
- 可以通过如下命令来查询该文件是否存在：

  ls /proc/bus/usb/devices
- 如果通过以上命令没有查询到devices文件，则需要通过如下命令来生成该文件：

  mount -t usbfs none /proc/bus/usb
- 通过如下命令查看devices文件的信息

cat /proc/bus/usb/devices

Ø 查询当前的数据卡/模块设备的PID信息：

lsusb

Ø 当前数据卡/模块设备的modem、pcui等端口的设备文件名称 查询命令：

ls /dev/ttyUSB*

6.2 Linux 驱动工具 HWActivator 的查询操作

Ø Linux驱动工具HWActivator是否正在运行的查看命令：

ps –ef |grep HWA

Ø Linux驱动工具HWActivator的版本号查询命令：

$PATH/HWActivator –version

6.3 在 Linux 系统上的 modem 拨号

在Linux系统上对华为数据卡/模块设备的modem进行拨号，可以使用Linux系统上的wvdial模块来进行拨号。Wvdial.conf的配置，可以参考网上的示例。