# 1 libusb 移植

## 1.1 在ubuntu上面安装libusb并编译程序：

sudo apt-get install libudev-dev

./configure

sudo make

sudo make install

编译：

#include <libusb-1.0/libusb.h>//头文件必须如此包含libusb.h

gcc -o gps\_usb gps\_usb.c -lusb-1.0//编译必须加上-lusb-1.0

chao@chao:~/libusb/libusb-1.0.19/tests$ lsusb

Bus 001 Device 002: ID 0951:1642 Kingston Technology DT101 G2

chao@chao:~/libusb/libusb-1.0.19/tests$ lsusb -d 0951:1642 -v //查看设备的详细信息

注意在ubuntu上面执行程序以root用户身份，否则可能出现打不开设备情况

## 1.2 移植libusb到arm：

新建文件夹libusb，把下载的libusb-1.0.19.tar.bz2和libusb-compat-0.1.5.tar.bz2放在该目录，并解压

1、编译安装libusb

root@chao:/home/chao/libusb/ tar xjf libusb-1.0.19.tar.bz2

root@chao:/home/chao/libusb/ cd libusb-1.0.19/

root@chao:/home/chao/libusb/libusb-1.0.19/

./configure --host=arm-linux --prefix=`pwd`/install CC=arm-linux-gcc --disable-udev

root@chao:/home/chao/libusb/libusb-1.0.19/make

root@chao:/home/chao/libusb/libusb-1.0.19/ make install

2、编译安装libusb-compat(注意安装路径与libusb保持一致，便于不同版本兼容)

root@chao:/home/chao/libusb/libusb-1.0.19/ export PKG\_CONFIG\_PATH=/work/nfs\_root/fs\_mini\_mdev\_new/libusb/libusb-1.0.19/install/lib/pkgconfig:$PKG\_CONFIG\_PATH

root@chao:/home/chao/libusb/libusb-1.0.19/ echo $PKG\_CONFIG\_PATH //查看是否配置//成功，为了compat使用

book@book-desktop:/work/nfs\_root/fs\_mini\_mdev\_new/libusb$

tar xjf libusb-compat-0.1.5.tar.bz2

root@chao:/home/chao/libusb/ cd libusb-compat-0.1.5/

root@chao:/home/chao/libusb/libusb-compat-0.1.5/ ./configure --host=arm-linux CC=arm-linux-gcc --prefix=`pwd`/install

root@chao:/home/chao/libusb/libusb-compat-0.1.5/ make

root@chao:/home/chao/libusb/libusb-compat-0.1.5/ make install

3、拷贝库和头文件

root@chao:/home/chao/libusb mkdir libusb

root@chao:/home/chao/libusb# mkdir libusb/include /libusb/lib/

root@chao:/home/chao/libusb/libusb-compat-0.1.5/install# cp include/\* ../../libusb/include/

root@chao:/home/chao/libusb/libusb-compat-0.1.5/install# cp lib/\* ../../libusb/lib/

root@chao:/home/chao/libusb/libusb-1.0.19/install# cp include/libusb-1.0/\* ../../libusb/include

root@chao:/home/chao/libusb/libusb-1.0.19/install# cp lib/\* ../../libusb/lib/

4、拷贝库到板子

先将lib目录拷贝到work

板子上执行

cd /mnt/lib/

cp \* /usr/lib

5、交叉编译程序：

#include <libusb-1.0/libusb.h>改为#include <libusb.h>

arm-linux-gcc test.c -o test -I/home/chao/libusb/libusb/include -L/home/chao/libusb/libusb/lib -lusb-1.0

# 2 libusb基础

# 3 libusb 实践

## 3.1 hotplug 测试并打印描述符

#include <signal.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <libusb.h>

**static** **struct** libusb\_device\_handle \*devh = NULL;

**struct** libusb\_device \*\*devs;

**static** **void** dump\_endpoint(**const** **struct** libusb\_endpoint\_descriptor \*endpoint)

{

**static** **const** **char** \* **const** typeattr[] = {

        "Control",

        "Isochronous",

        "Bulk",

        "Interrupt"

    };

**static** **const** **char** \* **const** syncattr[] = {

        "None",

        "Asynchronous",

        "Adaptive",

        "Synchronous"

    };

**static** **const** **char** \* **const** usage[] = {

        "Data",

        "Feedback",

        "Implicit feedback Data",

        "(reserved)"

};

**static** **const** **char** \* **const** hb[] = { "1x", "2x", "3x", "(?\?)" };

    unsigned **int** wmax=endpoint->wMaxPacketSize;

**printf**("      Endpoint Descriptor:\n"

           "        bLength             %5u\n"

           "        bDescriptorType     %5u\n"

            "        bEndpointAddress     0x%02x  EP %u %s\n"

           "        bmAttributes        %5u\n"

           "          Transfer Type            %s\n"

           "          Synch Type               %s\n"

           "          Usage Type               %s\n"

           "        wMaxPacketSize     0x%04x  %s %d bytes\n"

           "        bInterval           %5u\n",

           endpoint->bLength,

           endpoint->bDescriptorType,

            endpoint->bEndpointAddress,

           endpoint->bEndpointAddress & 0x0f,

            (endpoint->bEndpointAddress & 0x80) ? "IN" : "OUT",

           endpoint->bmAttributes,

           typeattr[endpoint->bmAttributes & 3],

           syncattr[(endpoint->bmAttributes >> 2) & 3],

           usage[(endpoint->bmAttributes >> 4) & 3],

           wmax, hb[(wmax >> 11) & 3], wmax & 0x7ff,

           endpoint->bInterval);

    /\* only for audio endpoints \*/

**if** (endpoint->bLength == 9)

**printf**("        bRefresh            %5u\n"

               "        bSynchAddress       %5u\n",

               endpoint->bRefresh, endpoint->bSynchAddress);

}

**static** **void** dump\_altsetting(**const** **struct** libusb\_interface\_descriptor \*interface){

**const** **struct** libusb\_endpoint\_descriptor \*end\_desc;

**int** m;

**printf**("Interface Descriptor:\n"

    "bLength             %5u\n"

    "bDescriptorType     %5u\n"

    "bInterfaceNumber    %5u\n"

    "bAlternateSetting   %5u\n"

    "bNumEndpoints       %5u\n"

    "bInterfaceClass     %5u\n"

    "bInterfaceSubClass  %5u\n"

    "bInterfaceProtocol  %5u\n",

    interface->bLength, interface->bDescriptorType,

    interface->bInterfaceNumber,interface->bAlternateSetting,

    interface->bNumEndpoints,interface->bInterfaceClass,

    interface->bInterfaceSubClass,interface->bInterfaceProtocol);

    //每个设置多个端点

**for** (m = 0; m < interface->bNumEndpoints; m++)

      {

        end\_desc = &interface->endpoint[m];

           dump\_endpoint(end\_desc );

        }

}

**static** **void** dump\_interface( **const** **struct** libusb\_interface \*interface)

{

**int** i;

    //每个接口有多个设置,可以设置成不同的接口

**for** (i = 0; i < interface->num\_altsetting; i++)

    dump\_altsetting(&interface->altsetting[i]);

}

**static** **struct** libusb\_device \*libusb\_search\_device\_with\_id(libusb\_context \*ctx, uint16\_t vendor\_id, uint16\_t product\_id)

{

**struct** libusb\_device \*found = NULL;

**struct** libusb\_device \*dev=NULL;

**struct** libusb\_device\_descriptor desc;

**size\_t** i = 0;

**int** r;

**if** (libusb\_get\_device\_list(ctx, &devs) < 0)

**return** NULL;

**while** ((dev = devs[i++]) != NULL) {

        r = libusb\_get\_device\_descriptor(dev, &desc);

**if** (r < 0)

**goto** out;

**if** (desc.idVendor == vendor\_id && desc.idProduct == product\_id) {

            found = dev;

**printf**("VID:PID: %04X:%04X\n", desc.idVendor,desc.idProduct);

**break**;

        }

    }

out:

    libusb\_free\_device\_list(devs, 1);

**return** found;

}

**static** **struct** libusb\_device \*find\_device()

{

**struct** libusb\_device \*dev;

    dev= libusb\_search\_device\_with\_id(NULL, 0x0951, 0x1642);

**return** dev;

}

**static** **int** LIBUSB\_CALL hotplug\_callback(libusb\_context \*ctx, libusb\_device \*dev, libusb\_hotplug\_event event, **void** \*user\_data)

{

**int** r;

    sleep(1);

    dev= find\_device();

**if** (dev) {

**printf**("find device \n");

        r=libusb\_open(dev,&devh);

**if**(r)

**printf**("open device error\n");

**else** **printf**("open device ok\n");

    }

**printf** ("attch device\n");

**return** 0;

}

**static** **int** LIBUSB\_CALL hotplug\_callback\_detach(libusb\_context \*ctx, libusb\_device \*dev, libusb\_hotplug\_event event, **void** \*user\_data)

{

**if** (devh)

    {

        libusb\_close (devh);

        devh=NULL;

        dev=NULL;

**printf** ("Device detached\n");

    }

**return** 0;

}

**int** main(**void**)

{

**int** r,k;

    libusb\_hotplug\_callback\_handle hp[2];

**struct** libusb\_device \*dev = NULL;

**struct** libusb\_device\_descriptor dev\_desc;

**struct** libusb\_config\_descriptor \*conf\_desc;

    r = libusb\_init(NULL);

**if** (r < 0) {

**printf**("failed to initialise libusb\n");

**goto** out;

    }

**if** (!libusb\_has\_capability (LIBUSB\_CAP\_HAS\_HOTPLUG)) {

**printf** ("Hotplug capabilites are not supported on this platform\n");

        libusb\_exit (NULL);

    }

**printf** ("Hotplug capabilites are  supported on this platform\n");

    r = libusb\_hotplug\_register\_callback (NULL, LIBUSB\_HOTPLUG\_EVENT\_DEVICE\_ARRIVED,LIBUSB\_HOTPLUG\_ENUMERATE, 0x0951,

        0x1642, LIBUSB\_HOTPLUG\_MATCH\_ANY, hotplug\_callback, NULL, &hp[0]);

**if** (r < 0) {

**printf** ( "Error registering callback 0\n");

        libusb\_exit (NULL);

    }

    r = libusb\_hotplug\_register\_callback (NULL, LIBUSB\_HOTPLUG\_EVENT\_DEVICE\_LEFT, LIBUSB\_HOTPLUG\_ENUMERATE,  0x0951,

        0x1642,LIBUSB\_HOTPLUG\_MATCH\_ANY, hotplug\_callback\_detach, NULL, &hp[1]);

**if** (r < 0) {

**printf** ( "Error registering callback 1\n");

        libusb\_exit (NULL);

    }

 /\* 注释掉的代码是用来支持打印描述符的

    libusb\_get\_device\_descriptor(dev, &dev\_desc);

    printf("VID:PID: %04X:%04X\n", dev\_desc.idVendor, dev\_desc.idProduct);

    printf("num of confs: %d\n", dev\_desc.bNumConfigurations);

    libusb\_get\_config\_descriptor(dev, 0, &conf\_desc);

    printf("num of interfaces: %d\n",conf\_desc->bNumInterfaces);

    for (k = 0 ; k < conf\_desc->bNumInterfaces; k++)

        dump\_interface(&conf\_desc->interface[k]);

    \*/

**while** (1) {

        libusb\_handle\_events (NULL);

    }

out:

**if** (devh)

    {

        libusb\_close (devh);

        devh=NULL;

        dev=NULL;

    }

     libusb\_hotplug\_deregister\_callback(NULL,hp[0]);

    libusb\_hotplug\_deregister\_callback(NULL,hp[1]);

    libusb\_exit(NULL);

}

# 4 问题汇总

**1、在板子上执行热插拔程序发现无法打开设备：**

猜测是热插拔函数执行快于设备节点的形成，要加延时，经过实验证明猜测是正确的。

**static** **int** LIBUSB\_CALL hotplug\_callback(libusb\_context \*ctx, libusb\_device \*dev, libusb\_hotplug\_event event, **void** \*user\_data)

{

**int** r;

    sleep(1);

    dev= find\_device();

**if** (dev) {

**printf**("find device \n");

        r=libusb\_open(dev,&devh);

**if**(r)

**printf**("open device error\n");

**else** **printf**("open device ok\n");

    }

**printf** ("attch device\n");

**return** 0;

}

实验结果：

/mnt/libusb/test # ./hotplug

Hotplug capabilites are supported on this platform

VID:PID: 0951:1642

find device

open device ok

attch device

registering callback

usb 1-1: USB disconnect, device number 23

Device detached