UEFI Development Exploration 89- YIE002USB Development Board (12 Linux Programming)

原创 luobing4365 Posted on 2021-05-13 12:05:05 Views: 686 Collection 4 Likes 1 copyright
Category Column: UEFI Development Article Tags: uefi USB usb hid Linux hid Low-level application development

UEFI Development This column includes this content 503 Subscribe 104 articles Subscribe to our column

(Please keep it-> Author: Luo Bing https://blog.csdn.net/luobing4365)

YIE002USB Development Board Linux **Programming**

As for USB HID programming, it has been discussed at the beginning of YIE002USB:

1. The lower computer is a USB HID device made using YIE002, which is an embedded development;
2. The host computer includes Windows system, UEFI system and Linux system.

Mac OSX is not included in the host computer software, mainly because I seldom develop on Apple systems, and it can also be regarded as equivalent to the Linux system (which belongs to the Unix-like system).

So far, we have completed the software writing of the host computer for Windows system and UEFI system. Now it is time to enter the USB HID host computer software development for Linux.

1 Overview

For HID under Linux, the website "The Linux Kernel " has a very detailed description, which is worth reading carefully. The URL is: https://www.kernel.org/doc/html/latest/hid/index.html.

The device we want to access is a USB HID device. In Linux systems, the commonly used USB open source library is libusb. This is a cross-platform USB device access interface library developed in C language.

After finding this open source library on Github, I read through its documentation. In theory, it can fully implement the functions we need. However, the documentation also suggests that if you want to access HID devices, you can use hidapi.

Hidapi is an open source C language library for operating HID devices, suitable for Windows, Linux and Mac OSX platforms. It is for HID devices and is not suitable for other USB devices.

Its source code repository is: https://github.com/libusb/hidapi, the main directories are as follows:

```
hidapi: 头文件（所有平台共用一份头文件）hidapi.h
libusb: Linux系统实现源码文件hid.c，使用libusb库实现的方式
linux: Linux系统实现源码文件hid.c，使用内核接口（hidraw）实现方式
windows: Windows系统实现源码文件hid.c
mac: Mac OSX 系统实现源码文件hid.c
hidtest: 测试代码 test.c
```

What we are going to develop now is the USB HID access program under Linux. From its code structure, we can see that there are two ways to use it, namely hidraw and libusb libraries.

Since hidapi is well encapsulated, there is not much difference between the two development methods. Most of my work is to consider how to write the Makefile file for compilation.

This article mainly uses the hidraw method to develop access code under Linux.

2. Preparation of HidAPI

During the development process, I used a virtual machine (Ubuntu 16.04) and an actual machine (Ubuntu 18.04) to conduct experiments. The former was used to develop the hidraw method code, and the latter was used to develop the libusb method code.

I use two development environments because I found that the code developed in libusb mode has various problems when running on a virtual machine. The specific reasons are unknown. However, the code developed in both modes runs well on the actual machine.

2.1 Install necessary components

Download the hidapi library to your local computer:

```
robin@robin-virtual-machine:~$ git clone https://github.com/libusb/hidapi.git
```

The hidapi library is relatively small and should be downloadable. If you still have problems, it is recommended to find the corresponding library on gitee and git it locally using the same method.

Install necessary components:

```
robin@robin-virtual-machine:~$ sudo apt-get install libudev-dev libusb-1.0-0-dev
```

For the required components, please refer to the requirements in the hidapi documentation. During the development process, there is no need to use the GUI, so libfox-1.6-dev does not need to be installed.

If the tools required for compilation are not installed, you also need to install them:

```
robin@robin-virtual-machine:~$ sudo apt-get install  build-essential pkg-config
```

build-essential contains the tools required for compilation, including gcc, make, etc. pkg-config is a command under Linux, which is used to obtain all compilation-related information of a library/module and is needed during the compilation process.

2.2 Hidapi main interface

Initialization and exit:

```
hid_init(): 初始化函数，无参数。
hid_exit(): 退出，实际上是销毁结构体等，在完成所有工作后必须调用，否则会造成内存泄漏。
```

enumerate:

```
hid_enumerate(): 枚举设备，返回的是hid_device_info链表。一般使用hid_enumerate(0, 0)枚举所有设备。枚举一般用于获取设备ID或者设备路径。如果提前知道这些信息，亦可不用枚举。
hid_free_enumeration(): 释放枚举所用到的链表。
```

Equipment opening and closing:

```
hid_open(): 打开指定 VID 和 PID 设备，返回设备结构体指针，如 hid_device *handle; handle = hid_open(0x8765, 0x4321, NULL)。设备读写和关闭函数时，均需要该结构体指针。
hid_open_path(): 根据设备路径打开设备，设备路径由hid_enumerate获取。如Linux下的 /dev/hidraw1。
hid_close(): 关闭设备。
```

Feature Report Sending and Receiving:

```
hid_get_feature_report(): 获取 Feature report，也即采用Feature报告进行通信。
hid_send_feature_report(): 发送 Feature report。
```

Read and write functions:

```
hid_read(): 读取数据。
hid_write(): 写数据。
```

It should be pointed out that many articles on the Internet regard hid_read() and hid_write() as reading input reports and sending output reports. This understanding is problematic. From the following experiments, it can be seen that hid_read() and hid_write() do not use the communication method of input report and output report. At least this understanding is not comprehensive.

From the experimental results, these two functions are similar to the ReadFile() and WriteFile() methods of the Windows system, while the lower computer corresponds to the endpoint communication method (see the 85th blog). That is, when there are multiple endpoints (including the required endpoint 0 and other endpoints), endpoint communication is used; only when there is endpoint 0, the Input report and Output report communication methods will be used.

In fact, in the source code of hidapi (hid.c in the inux folder), it is clearly stated that the hidraw method does not implement the acquisition of input report.

Of course, how to design it depends on the firmware of the lower computer.

3 Example of using the hidraw method of hidapi

After understanding the above background knowledge, it is relatively easy to implement the required code.

Hidapi provides a test program, located in the hidtest folder test.c. To compile the Linux version, you can use the Makefile file in the linux folder to compile:

```
robin@robin-virtual-machine:~/hidapi/linux$ make -f Makefile-manual
```

After compiling, the hidtest-hidraw executable file will be generated. We need to implement the required functions based on test.c.

3.1 Prepare the development directory

To facilitate subsequent development, you need to create a development folder yourself and include the required source files and header files.

Create a new folder hidraw and copy several files of hidapi here:

```
linux/hid.c
hidapi/hidapi.h
hidtest/test.c
```

In the folder linux, there is a file Makefile-manual for compilation. Since the file directory has changed, it cannot be used directly. You can create a new Makefile in hidraw with the following content:

```
all: hidtest-hidraw
CC       ?= gcc
```

```
CFLAGS   ?= -Wall -g -fpic

LDFLAGS  ?= -Wall -g

COBJS     = hid.o test.o
OBJS      = $(COBJS)
LIBS_UDEV = `pkg-config libudev --libs` -lrt
LIBS      = $(LIBS_UDEV)
INCLUDES ?= `pkg-config libusb-1.0 --cflags`


# Console Test Program
hidtest-hidraw: $(COBJS)
    $(CC) $(LDFLAGS) $^ $(LIBS_UDEV) -o $@

# Objects
$(COBJS): %.o: %.c
    $(CC) $(CFLAGS) -c $(INCLUDES) $< -o $@

clean:
    rm -f $(OBJS) hidtest-hidraw $(COBJS)

.PHONY: clean libs
```

Open the terminal in the hidraw folder and enter make to compile:

```
robin@robin-virtual-machine:~/luotest/hidapi$ make
cc -Wall -g -fpic -c `pkg-config libusb-1.0 --cflags` hid.c -o hid.o
cc -Wall -g -fpic -c `pkg-config libusb-1.0 --cflags` test.c -o test.o
cc -Wall -g hid.o test.o `pkg-config libudev --libs` -lrt -o hidtest-hidraw
```

It can be seen that the executable file hidtest-hidraw is generated.

Next, we will implement access to USB HID devices based on the original code in test.c.

3.2 USB HID hidraw code development

From the previous blog, we know that there are three ways for the host computer to communicate with the USB HID device. Currently, the hidapi code provides two methods: Feature report and hid_read()&hid_write(), but does not provide the communication method of Input report&Output report.
Mainly, in Linux hidraw, there is no corresponding interface.

As an aside, I really can't understand why there is no Input report & Output report interface in hidraw. If anyone knows, please leave a message in the comment section of the blog.

3.2.1 Feature report communication method

Feature report communication uses two functions provided in hid.c: hid_send_feature_report() and hid_get_feature_report(). The code is as follows:

```
    memset(yie_buf,0,sizeof(yie_buf));
    yie_buf[0] = 0x00;
    yie_buf[1] = 0xA0;
    yie_buf[2] = 0x0a;
    yie_buf[3] = 0x0b;
    yie_buf[4] = 0x0c;
    res = hid_send_feature_report(handle, yie_buf, 17);
    if (res < 0) {
        printf("Unable to send a feature report.\n");
    }

    memset(yie_buf,0,sizeof(yie_buf));

    // Read a Feature Report from the device
    // buf[0] = 0x2;
    res = hid_get_feature_report(handle, yie_buf, sizeof(yie_buf));
    if (res < 0) {
        printf("Unable to get a feature report.\n");
        printf("%ls", hid_error(handle));
    }
    else {
        // Print out the returned buffer.
        printf("Feature Report\n   ");
        printf("report number:%d\n   ",yie_buf[0]);
        for (i = 1; i < res; i++)
            printf("%02x ", yie_buf[i]);
        printf("\n");
    }
```

3.2.2 hid_read()&hid_write() communication method

hid.c provides hid_read() and hid_write() functions, which can implement functions similar to Windows' ReadFile() and WriteFile() (only for USB HID devices). The implementation code is as follows:

```
    memset(yie_buf,0,sizeof(yie_buf));
    yie_buf[0] = 0x00;
    yie_buf[1] = 0xA0;
    yie_buf[2] = 0x0a;
    yie_buf[3] = 0x0b;
    yie_buf[4] = 0x0c;

    res = hid_write(handle, yie_buf, 17);
    if (res < 0) {
        printf("Unable to write()\n");
        printf("Error: %ls\n", hid_error(handle));
    }

    memset(yie_buf,0,sizeof(yie_buf));
    res = hid_read(handle, yie_buf, sizeof(yie_buf));
    if (res < 0) {
        printf("Unable to Read data.\n");
        printf("%ls", hid_error(handle));
    }
    else {
        // Print out the returned buffer.
        printf("Read data\n   ");
        for (i = 0; i < res; i++)
            printf("%02x ", yie_buf[i]);
        printf("\n");
    }
```

After modification, compile using make. Plug in the self-made USB HID device and run the test code. The test results are as follows:

```
robin@robin-virtual-machine:~/luotest/hidapi$ sudo ./hidtest-hidraw
……//信息略
Manufacturer String: Robin
Product String: Robin's UEFI Explorer
Serial Number String: (77) My123
Unable to read indexed string 1
Indexed String 1:
Feature Report
   report number:0
   a0 03 0b 0c 00 00 00 00 00 00 00 00 00 00 00 00
Read data
   a0 01 0b 0c 00 00 00 00 00 00 00 00 00 00 00 00
```

Pay attention to the second byte of the returned data. From the content in the previous blog, we can know that two of the three communication methods have been implemented.

To achieve the complete three communication methods, the next article will use the libusb library to rebuild the code.

Gitee address: https://gitee.com/luobing4365/uefi-explorer
Project code is located at: /89 hidraw