The Research and Realization of USB Gadget Driver in Linux System

Sun Yufei, Fan Binwen, Ma Qianqian, Wang Panzhenzhuan

Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, Guangdong, 518000, China

773567844@qq.com

Keywords: Linux system; embedded system;driver;USB;gadget

Abstract. The USB devices are the most used embedded economical devices. To develop the driver for the devices in embedded Linux system better, USB basic knowledge and its driver model are introduced. This paper mainly concentrates on the USB driver on the slave side which is called gadget driver. The main data structures of gadget driver are analyzed. Then the principle of the UDC driver are introduced. As a result, how to design a gadget driver is demonstrated by the analysis of the file storage gadget driver.

1 Introduction

The USB (universal serial bus) is a serial bus standard which connects the computer system and the external equipment. And it is also an input and output interface specification. The USB device is widely used in electronic devices due to the following advantages:

(1) The hot plug characteristic；

(2) Easy to carry；

(3) The USB devices have a unified standard. The common electronic devices such as keyboard, mouse, printer, scanner and so on can connect with the host using the same standard.

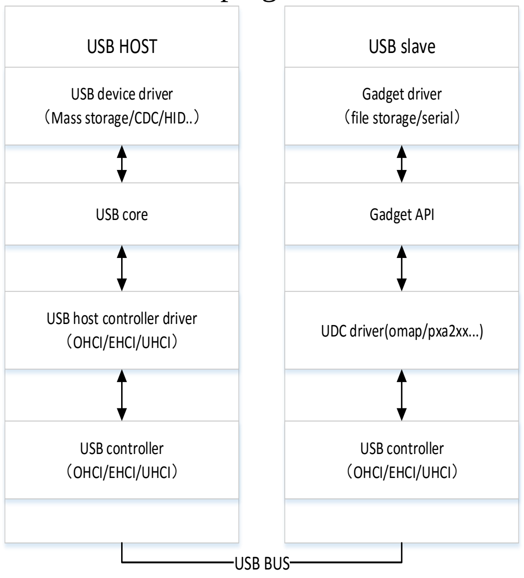
(4) Easy to extend. The USB port on the host can be extended by USB HUB.

In this paper ,we study the USB driver based on Linux system. The Linux system is widely used on the embedded system for the characteristic of opening source, portability and tailorable. Under the Linux system, the USB driver uses the tree topology structure. Physically, the USB device model can be divided into 2 parts: the master side is the USB Host Controller, and the slave side is the USB Device Controller (UDC). This paper emphatically introduces the USB driver on the slave side

2 The USB Driver in Linux System

The Linux system provides good support for the driver development of both the host and the slave devices.The USB driver structure under Linux system is showed as figure 1.

From the host side running on Linux system, the bottom of the driver is the USB host controller which directly operates the hardware. And then to is the USB Host Controller driver; again is USB core layer. The upper is the USB device driver layer such as U disk, mouse, USB to serial port device driver etc. It can be seen that: the host side driver mainly includes the Host Controller USB driver and the USB device driver. The former controls the USB device inserted in the host. And the latter controls the communication between the USB device and the host. The USB Core is responsible for the main task of the USB driver's management and the protocol processing. The task mainly includes 4 parts. First is the definition of some important data structure, macros and functions. Second is to provide programming interface for the upper device driver. Third is to maintain the USB device information of the whole system through some global variables. The last is to control the hot-plug of the devices and the data transmission on the USB bus.


USB HOST
USB device driver (Mass storage/CDC/HID..)
USB core
USB host controller driver (OHCI/EHCI/UHCI)
USB controller (OHCI/EHCI/UHCI)
USB slave
Gadget driver (file storage/serial)
Gadget API
UDC driver(omap/pxa2xx...)
USB controller (OHCI/EHCI/UHCI)
USB BUS
Fig.1. The USB driver structure under Linux system

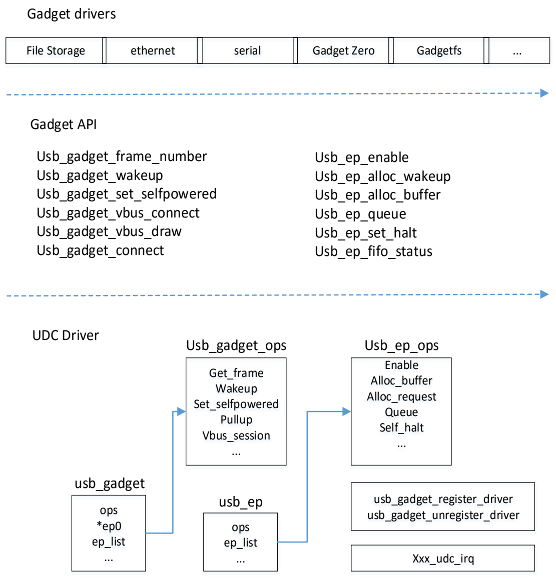


Fig.2. The structure of Linux gadget driver

In recent years, with the widely use of embedded devices, the slave devices based on Linux system also need USB protocol to communicate with the host devices. In order to avoid the confusion of the host and slave driver of USB, we call the driver running on the slave side USB gadget driver. The driver structure is showed as the right of figure 1. From the picture, we can conclude that the USB gadget driver has 3 levels: the UDC driver, the gadget API and the gadget driver. The UDC driver operates the hardware directly, which also controls the communication between the host and the slave devices. Moreover the hardware operating functions are provided for the upper level in the UDC driver. The gadget API is the package of the hardware operating functions. The top level is the gadget driver, which realizes the specific function of the devices such as network, printer, USB storage and so on.

3 The Framework of Linux Gadget Dirver

The structure of the USB gadget driver in Linux system is showed as figure 2:

3.1The UDC driver. The UDC driver mainly contains several core data structure:

usb_gadget_driver, which is to describe the USB device controller.
usb_gadget_driver, which is to describe the USB gadget driver.
usb_request, which is to describe the requests of data transmission.
usb_ep, which is to describe the endpoints.
usb_ep_ops, which is to describe the operations of endpoints.

The UDC and gadget driver is mainly about the definitions and the instance of the above data structure. All of these are included in the folder: *include/linux/usb/gadget.h*. Moreover, the UDC driver needs to provide the following 2 APIs for the gadget driver:
int usb_gadget_register_driver(struct usb_gadget_driver *driver)
int usb_gadget_unregister_driver(struct usb_gadget_driver *driver)

The register function will be called when driver models initialize to bind the device and the driver. In general, the following works will be done in the UDC driver :

(1) Package the standard data structure *usb_gadget*, *usb_ep***,** *usb_request* and definite the corresponding data structure., according to the specific platform and device characteristic.

(2) Instantiate the function of *plarform_driver* and register the UDC driver to the platform driver subsystem. Platform driver is a kind of virtual bus in the Linux driver.

Among all of the member functions, *s3c2410_udc_probe* and *s3c2410_udc_remove* are most important. The *s3c2410_udc_probe* function connects the *platform_device* object and the UDC object. In addition the initialization of the UDC device and driver is achieved in the function. Besides, the function applies necessary source such as endpoint section, interrupt number and it binds the interrupt number with the interrupt handler. The interrupt handler is the entrance of all the device operations. The remove function is the inverse operation of the probe function, which release the source applied in the probe function.

(3) Instance the *usb_gadget_ops* function sets, which directly operates the UDC devices.

(4) Instance the *usb_ep_ops* function sets, which operates the endpoints.

(5) Instance the interrupt handler, which is the core of the UDC driver.

(6) Instance the driver register interface of the upper function driver:

```
int usb_gadget_register_driver(struct usb_gadget_driver *driver)
int usb_gadget_unregister_driver(struct usb_gadget_driver *driver)
```

3.2 The Gadget Driver. The gadget driver calls the gadget API to realize the specific function of the USB devices. Common USB gadget driver includes that: gadget zero, Ethernet over USB, file-backed storage gadget and serial Gadget.

The following research analyzes the File-backed Storage Gadget driver. The driver is realized in the file: *drivers/usb/gadget/file_storage.c*. The main tasks completed by this driver are:

(1) To definite and instance several descriptors which are device descriptor, configuration descriptor, interface descriptor and endpoint descriptor.

(2) To instance the function usb_gadget_driver and its member functions: *bind(), unbind(), disconnect(), setup()* and so on.

The instance of the bind() function is the main task of the gadget driver. In this function, the devices function, endpoints, descriptor is installed. For instance, the logic units of the storage device are mapped to the virtual file system. The code analysis is as follows:

```
static int __init fsg_bind(struct usb_gadget *gadget)
{
        /*First is the definition of some necessary varibles*/
         struct fsg_dev         *fsg = the_fsg;
         int                    rc;
         int                    i;
         struct lun             *curlun;
         struct usb_ep          *ep;
         struct usb_request      *req;
         char                   *pathbuf, *p;
                  /*Second is the configuration of gadget function and endpoint*/
                  fsg->gadget = gadget;
                  set_gadget_data(gadget, fsg);
                  fsg->ep0 = gadget->ep0;
                  fsg->ep0->driver_data = fsg;
        /*Third is to find out the number of Logic Units and create the LUNs and open their backing
files. */
                  i = mod_data.nluns;
                          ...
                  fsg->luns = kmalloc(i * sizeof(struct lun), GFP_KERNEL);
                  memset(fsg->luns, 0, i * sizeof(struct lun));
         fsg->nluns = i;
                          ...
    /* Find all the endpoints we will use */
        usb_ep_autoconfig_reset(gadget);
        ep = usb_ep_autoconfig(gadget, &fs_bulk_in_desc);
        if (!ep)
```

```
            goto autoconf_fail;
        ep->driver_data = fsg;
        fsg->bulk_in = ep;
                        ...
    /*Then fix up the descriptors */
         device_desc.bMaxPacketSize0 = fsg->ep0->maxpacket;
         device_desc.idVendor = cpu_to_le16(mod_data.vendor);
         device_desc.idProduct = cpu_to_le16(mod_data.product);
         device_desc.bcdDevice = cpu_to_le16(mod_data.release);
         i = (transport_is_cbi() ? 3 : 2);      // Number of endpoints
         intf_desc.bNumEndpoints = i;
                        ....
    }
```

4 Conclusion

In Linux system, the USB gadget driver is based on the UDC driver and gadget API. The gadget API is the package of the operation functions of the UDC and endpoints. The gadget driver uses the API to control the communication process between the host and the devices.

In general, the framework of the gadget driver is not perfect enough. Because the gadget API does not realize the real isolation of the gadget driver from the UDC driver. The coupling problem exists either in the key data structure or the register function. All of this makes that, the gadget driver engineer needs to understand the bottom UDC driver to design the gadget driver.

Acknowledgment

This work was supported by the Shenzhen science and technology project (NO.JCYJ20150625142543448).Thanks to my laboratory: the Internet of Things Key Technology Laboratory of Harbin Institute of Technology Shenzhen Graduate School. Thanks for my tutor's guidance.

References

[1] Xiaobin Chu,Tiejun Lun,Yu Zong. Functional Verification of USB Mass Storage[J]. Lecture Notes in Engineering and Computer Science,2008,21691

[2] Axelson J. USB mass storage designing and programming devices and embedded hosts. Lakeview Research . 2006

[3] Detlef Fliegl.Programming Guide for Linux USB Device Drivers. http://usb.cs.tum.edu . 2000

[4] Jan Axelson.Serial Port Complete:COM Ports USB Virtual COM Ports,and Ports for Embedded Systems second edition. . 2007

[5] Linux-USB Community. Linux-USB GadgetAPI Frame-work[EB]. http://www. linux-usb. org /gadget, 2005.

[6] Tan Wool Ming,Solari Edward.Developing USB PC Peripherals[M]. Independent Pub Group, 2005