# Integration of Linux USB Device Drivers into a Component-Based Embedded Operating System

Chung-Wei Tsai
Department of Information Management
National Chi Nan University
Email: s96213516@ncnu.edu.tw

Mei-Ling Chiang
Department of Information Management
National Chi Nan University
Email: joanna@ncnu.edu.tw

Yu-Chen Yeh
Department of Information Management
National Chi Nan University
Email: s97213527@ncnu.edu.tw

*Abstract*--**In recent years, the booming of embedded systems provides more and more applications, and the corresponding device drivers must be developed in time when numerous peripheral devices have been promoted. Some of those devices support Universal Serial Bus (USB) which is a serial bus standard designed for connecting devices to a host computer and is widely used in devices.**

**This paper discusses how we transplant Linux USB device drivers into a component-based embedded operating system. We take the source codes in Linux 2.6 kernel and integrate them into the component-based embedded operating system named LyraOS after making the source codes wrapped with wrappers. Our major tasks include (1) transplanting drivers of USB host controller, USB Core, USB keyboard and USB mouse into LyraOS, (2) implementing the required data structures and functions of device drivers in wrappers, and (3) integrating the transplanted USB device drivers into LyraOS's device driver model.**

*Index Terms:* **Embedded Operating Systems, Device Driver, Transplantation, Linux, USB**

## 1. Introduction

With the rapid development of information technology in recent years, the embedded systems become the daily applications for human beings. Many embedded operating systems have been developed with the booming of embedded systems. However, device drivers must be redesigned due to the difference between varieties of embedded operating systems. For pushing the hardware products to market quickly, how to efficiently reduce the development time and costs becomes the significant issue for vendors.

LyraOS [1-4] is a component-based embedded operating system which is created as a research platform for operating systems and providing a set of well-designed and clear-interface system software components. The ideas of component-based design and component reuse let us think over the feasibility of reusing the existent device drivers deliberately. However, we need a universal and open source library to help us continuously to make a further study over the device drivers. Due to the feature of open source, Linux is the biggest library of open source and device drivers.

Universal Serial Bus (USB) is a serial bus standard for connecting devices to a host computer. Owing to the widely-used USB devices, our research focuses on implementing USB device drivers in LyraOS for supporting USB devices.

Currently, we have undertaken the transplantation of USB device drivers from Linux 2.6 kernel into LyraOS. The major tasks of our research are transplanting USB host controller drivers, USB

Core, USB keyboard and mouse drivers into LyraOS, and implementing necessary data structures and functions for supporting these transplanted drivers, as well as integrating the transplanted USB device drivers into LyraOS's device driver model.

## 2. Background and Related Work

This section briefly introduces the device driver models of LyraOS and Linux. The structure of USB is introduced as well.

### 2.1 LyraOS

LyraOS [1-4] is designed to abstract the hardware resources such that low-level machine dependent layer is clearly divided from higher-level system semantics. Thus, it can be easily ported to different hardware architectures [2,4].

Figure 1 shows system architecture of LyraOS. Each system component is completely separate, self-contained, and highly modular. LyraFILE component [7,8], a light-weight VFAT-based file system, supports both RAM-based and disk-based storages. Besides, LyraOS provides the Linux device driver emulation environment [9-11] to make use of Linux device drivers. Under this emulation environment, Linux device driver codes can be integrated into LyraOS without modification. Furthermore, the LyraNET component [12,14], a TCP/IP protocol stack derived from Linux TCP/IP codes is implemented with the zero-copy mechanism to reduce protocol processing overhead and memory usage. Recently, LyraOS uses a sensor-side prelinking mechanism to support dynamic component update to efficiently update its components on-the-fly without rebooting. Besides, a memory protection mechanism [15] is implemented to safely update LyraOS's components dynamically.

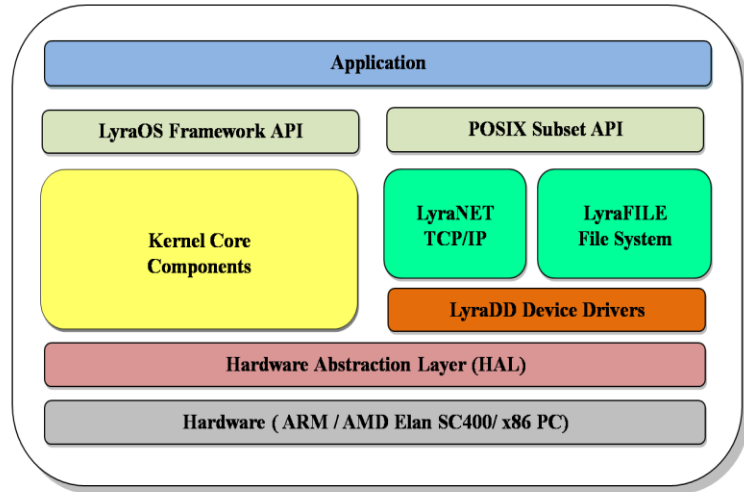


**Figure 1. LyraOS system architecture**

LyraOS supports three classes of devices, i.e. character devices, block devices, and network devices. LyraOS adopts object-oriented design concept and is implemented with C++ language. Figure 2 shows device types of LyraOS.

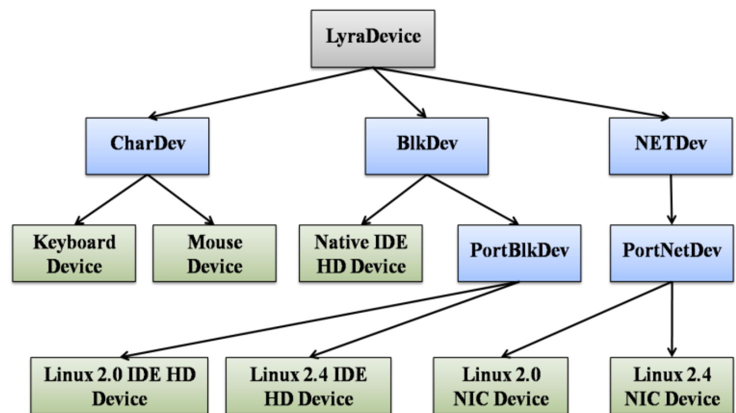


**Figure 2. Device types of LyraOS**

Figure 3 shows the base class structure for hardware devices. Each device has *Device_ID* and *Instance_Number* fields to identify hardware device. *Native_Device* filed is used to differentiate Linux drivers from Native drivers. If the device uses a Linux driver, then the *Port_Device_Struct*

field will point to the Linux device data structure.

```
class LyraDevice
{
        public:
                LyraDevice(){};
                ~LyraDevice(){};
                friend class DeviceMgr;
                void init(void);

         protected:
                LyraDevice *Prev, *Next;
                unsigned int Device_Type;
                unsigned int Device_ID;
                unsigned int Instance_Number;
                bool Native_Device;
                void * Port_Device_Struct;

                virtual void open(void){};
                virtual void close(void){};
                virtual void ioctl(void){};
                virtual int read(void * Parms){return 0;};
                virtual int write(void *Parms){return 0;};
};
```

**Figure 3. The base class structure for hardware devices**

LyraOS device driver model [9] contains four components, including Device Manager, LyraOS wrapper, native device drivers, Linux device drivers. Device Manager is used to manage all devices. LyraOS wrapper includes OS service wrapper and driver specific wrapper. There are two device driver types in LyraOS, including native device drivers and Linux device drivers. Linux device drivers were transplanted from Linux. Nevertheless, both types of the drivers support different functions. In order to let each OS kernel component access all device drivers conveniently, the Device Manager component provides a unified management interface and each component of OS kernel invokes functions of device drivers through the Device Manager component.

Device Manager component maintains a linked list named DEVICE_LIST which stores the information of all controllable hardware devices in system. When a device driver begins to initiate, it invokes functions of the Device Manager and adds information of device to the DEVICE_LIST. The Device Manager also includes functions for registering I/O port for device drivers, I/O management, and IRQ Management. Table 1 shows functions of the Device Manager component.

| Functions of Device Manager Component | Description |
|---|---|
| DMgr.read() | Read data from hardware |
| DMgr.write | Write data to hardware |
| DMgr.Set_Device() | Register hardware information to DEVICE_LIST |
| DMgr.Get_Device_Instance_Number() | Get the instance number of hardware |
| DMgr.FindDev() | Get a pointer of device structure |

**Table 1. Functions of Device Manager**

Before an OS kernel component invokes device driver, it must use *FindDev()* function of Device Manager component to get a pointer of hardware device structure. Then the OS kernel component could use *read()* and *write()* functions of Device Manager component to invoke device driver. Figure 4 shows Device Manager component of LyraDD [9].

LyraOS wrapper includes OS service wrapper and device specific wrapper. Developers can transplant Linux device driver codes into LyraOS and need not major modification. OS service wrapper provides wrapper functions to map Linux kernel functions to OS service in LyraOS and implement some functions which are not provided in LyraOS. Device specific wrapper provides data structures needed in Linux device drivers and supports specific device functions including Net driver wrapper and IDE driver wrapper (for Linux kernel 2.0/2.4).

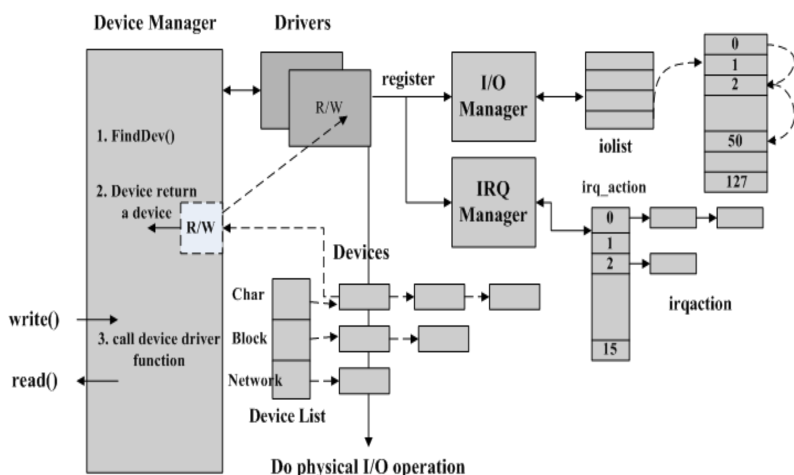


**Figure 4. Device management of LyraDD**

### 2.2 Linux Device Drivers

Linux's source codes are open and available for everyone to trace and research [13]. Therefore, it results in having many developers devoting themselves to test, modify, and develop applications and programs in Linux. Device drivers play an important role in Linux kernel [16], they could control and communicate with hardware. Developing device drivers can be separated from Linux kernel while they could be developed modularly. Because of the modular design, the kernel would not need to be updated frequently for supporting new hardware. Traditionally, Unix systems classify hardware devices into three types while the drivers could be three types as well. These types are character device, block device, and Network interface.

### 2.3 USB Architecture

USB, as known as Universal Serial Bus, is a bridge to connect a computer and affiliated devices. Topologically, USB could not be counted as bus, instead it is a tree structure which is constructed of multiple point-to-point lines. USB devices connect to USB hub via four-wire cables. The USB host controller shall query each USB device in rotation to check if any of these devices has data to send. Therefore, USB devices would not transmit a bit of data before receiving the first call from host controller. The purpose of the design is to support plug and play and let the system configure the device just plugged in easily and automatically. Another feature of USB is that it is only a communication tunnel between computers. USB specification has defined a set of standard protocols for any type of devices to follow, which means that there is no need to develop drivers for particular devices because the same type of devices could use the same driver. USB has defined many device classes such as storage media, keyboard, mouse, network device, modem, and printer, et al.

Figure 5 shows USB device overview, the USB driver is located among different kernel subsystems (i.e. block, character, network). USB core provides a platform to USB drivers to access and control USB hardware without dealing with the format of the USB hardware host controller.

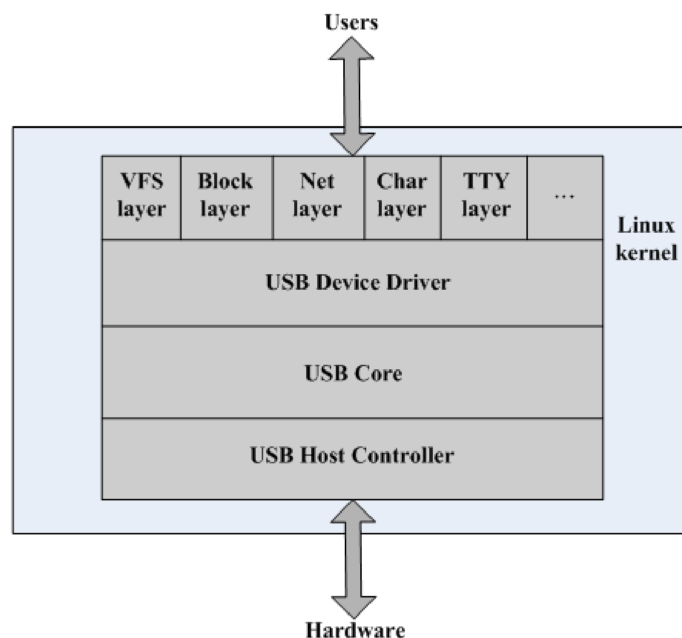


**Figure 5. USB device overview**

The main task of the USB host controller is to communicate with hardware. USB host controller has three types: OHCI (Open Host Controller Interface), UHCI (Universal Host Controller), and EHCI (Enhanced Host Controller Interface). OHCI driver supports non-PC and chipset that belongs to SiS, ALi USB chipset. UHCI driver supports most PCs including Intel and Via USB chipset. EHCI issued in USB 2.0 standard is compatible with both OHCI and UHCI.

USB devices are complicated, but Linux kernel has provided USB core subsystem that could take care of the complicated parts. Figure 6 shows the USB device descriptors [17]. The communication of USB is through so-called endpoint. One USB endpoint could only send data in one direction that is called one-way transmission. OUT endpoint is used to send data from computer to device's node. IN endpoint is used to send data from device to computer. Configurations are composed of the operating status of USB interface. An USB interface can have multiple modes, but the device can only stay under one kind of the modes at anytime. The USB devices can change status by switching to other modes, for example, firmware update mode.

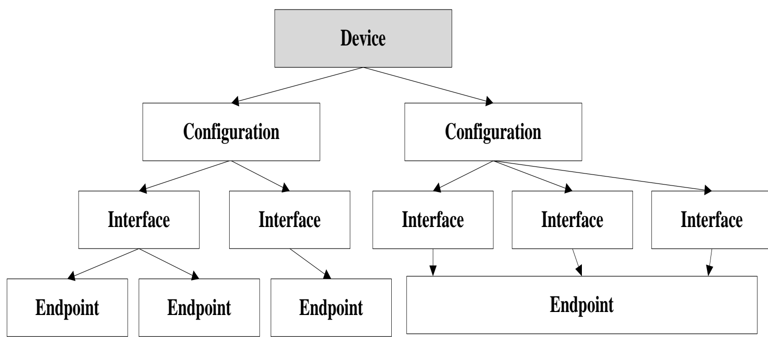


**Figure 6. USB device descriptors.**

In summary, the USB devices are complex and the devices are composed of different logic units. The devices can have one or multiple configurations and configurations usually include one or multiple interfaces. An interface often has one or more settings and an interface can include no endpoints or multiple endpoints.

## 3. Supporting USB Devices in LyraOS

Section 3 mainly describes how to integrate LyraOS with Linux's USB device drivers. Section 3.1 describes the modified LyraOS device driver model. Section 3.2 introduces Linux USB device driver architecture. Section 3.3 describes what kinds of USB devices drivers are integrated in the study. Finally, Section 3.4 focuses on the modification of LyraOS wrappers, and adding Linux 2.6 kernel wrapper to let the USB device driver work smoothly in LyraOS.

### 3.1 Modified LyraOS Device Driver Model

To integrate Linux's USB device drivers into LyraOS, we adopt the same design concept of LyraDD and add a USB driver wrapper to LyraDD. Figure 7 shows the modified LyraOS device driver model. In this study, we focus on integrating Linux's USB keyboard driver and mouse driver into LyraDD, so that USB keyboard and mouse can work under LyraOS. Since OS service wrapper is originally designed to work with Linux 2.4 device drivers, however, in this study we use Linux 2.6 USB device drivers. Therefore, OS service wrapper also gets updated.

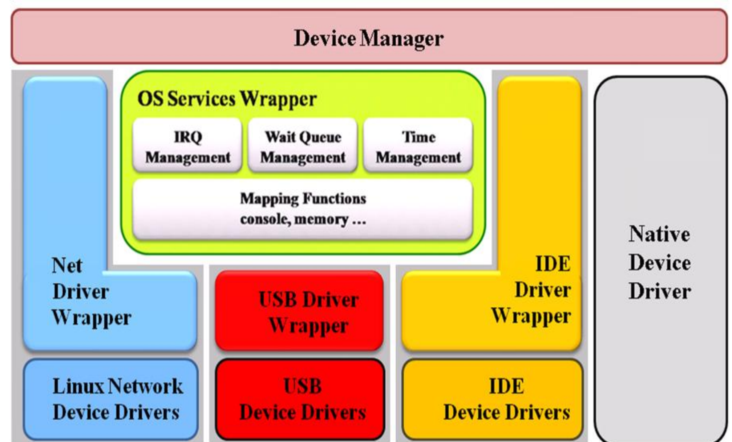


**Figure 7. Modified LyraOS device driver model.**

### 3.2 Linux USB Driver Architecture

As shown in Figure 8 [17], the overall structure of USB device drivers can be classified into several layers such as hardware USB host controller, USB host controller driver, USB core, and USB device drivers for flash drive, mouse, keyboard, hub, wireless network interface, etc.

Figure 8 depicts that there are two kinds of USB drivers, USB controller driver and USB device driver. The former controls the inserted USB device and the latter controls the USB device to commu-

nicate with operating system. USB Core of Linux kernel manages the USB device driver and handles USB protocol and USB data transmission. It plays an important role between the controller driver and the USB device driver.

The USB controller can be classified to OHCI, EHCI, UHCI. Each kind of controller has a corresponding driver. USB core includes Endpoint, Interface, Configurations structure, and URB (USB request block) structure. Programs in Linux use the URB to communicate with the USB devices. The keyboard and mouse drivers belong to human interface device (HID) category of Linux USB device, so they follow HID specification.

Figure 9 shows the hot-plugging flow of the USB device module [18], including device attachment and detachment.

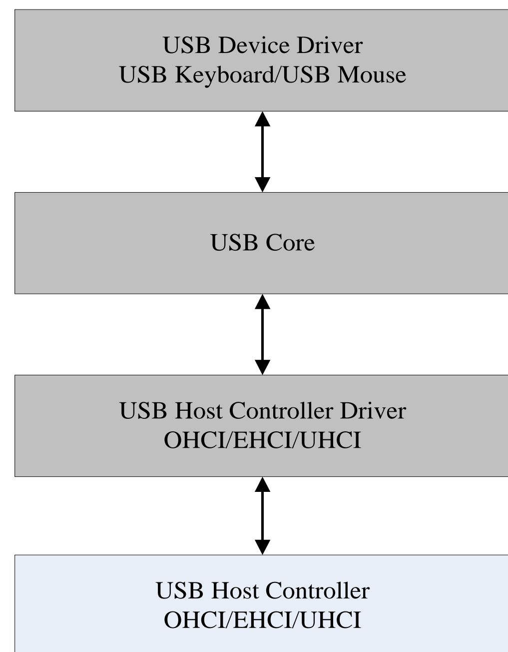


**Figure 8. Linux USB driver architecture**

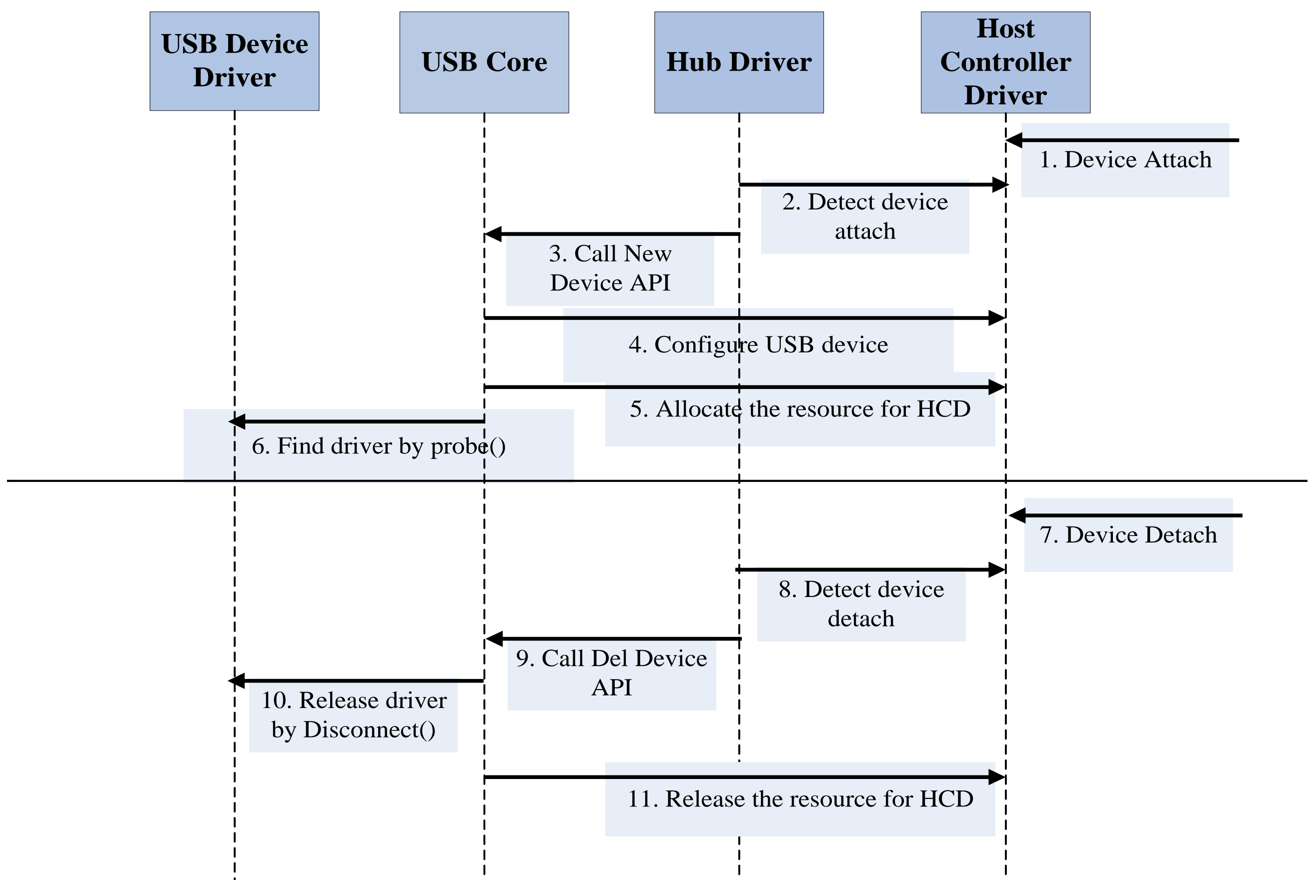


**Figure 9. Hot-plugging flow of USB device**

1. Device Attachment: The interrupt is invoked and hub driver obtains the information of connection when the computer senses the attachment of the USB device. USB Core provides the related functions to accomplish the configurations of USB device. Then, USB Core and Hub driver allocate the required resources for the new devices in USB Core, Hub driver, and host controller driver. USB Core calls the function *probe()* of USB device driver to seek the proper driver to the device.
2. Device detachment: The interrupt is invoked and hub driver obtains the information of disconnection when the computer senses the detachment. Then hub driver calls the related functions provided by USB Core to release the allocated resource in USB Core, hub driver and host controller. At the same time, USB Core calls the function *disconnect()* of USB device driver to release the driver.

### 3.3 Integrating Linux USB Device Drivers into LyraOS

Currently, we have transplanted USB host controller drivers, USB core, USB keyboard driver, and USB mouse driver from Linux 2.6 kernel. To integrate Linux USB device drivers into LyraOS, we wrap the transplanted device drivers with wrapper. There are two types of wrappers, which are device specific wrapper and OS service wrapper. The device specific wrapper provides device drivers the required data structures and specific functions which are needed by Linux device drivers but are lacked in LyraOS. Originally, there are only Net device wrapper and IDE driver wrapper in LyraOS. We add USB driver wrapper to support USB device drivers. The OS service wrapper provides wrapper functions to map Linux kernel functions to OS service functions in LyraOS and implements some certain functions lacked in LyraOS.

To integrate USB Host Controller Driver into LyraDD, some data structures are needed. In Linux kernel, *usb_hcd* (Host Controller Driver) data structure is used to describe USB host controller drivers. It includes information of USB host controller, hardware resource, status, and pointers to driver functions which are used to operate the controller. Besides, Linux *usb_driver* data structure is used to identify USB driver to USB Core.

In Linux kernel, the USB Core is a subsystem with a specific application programming interface (API) to support USB devices and host controllers. It aims to abstract all hardware or device dependent parts by defining a set of data structures, macros, and functions [17].

Every USB device must have the following four descriptors. The first one is endpoint descriptor. There are four types of USB endpoint, which are control endpoint, interrupt endpoint, bulk endpoint, and isochronous endpoint. Data structure of usb endpoint descriptor includes fields about length, endpoint descriptor type, endpoint address, attributes, max packets size, etc. The second one is interface descriptor. Fields of interface descriptor include descriptor types, interface number, settings, endpoint number, interface class and subclass, and interface protocol. The third one, configurations descriptor, includes length, descriptor type, total length, interface number, configuration value, max electricity, etc. Device descriptor includes length, descriptor type, USB version, device class and subclass, device protocol, endpoint max packet size, vendor identity, product identity, etc.

All the USB codes in Linux kernel use urb (USB request block) data structure to communicate with USB devices. This request block is described with the struct urb structure. A urb is used in sending or receiving data to or from a specific USB endpoint on a specific USB device in an asynchronous manner. A USB device driver may allocate many urbs for a single endpoint or may reuse a single urb for many different endpoints, depending on the need of the driver. Every endpoint in a device can handle a queue of urbs, so that multiple urbs can be sent to the same endpoint before the queue is empty [17].

The typical lifecycle of an urb is as follows [17]:

1. Created by a USB device driver. Firstly, USB device driver uses the function *usb_alloc_urb()* to create a urb data strucuture.
2. Assigned to a specific endpoint of a specific USB device. There are four types of endpoints, which are control, interrupt, bulk, and isochronous. Each endpoint has distinct initialization function and the mappings are as follows. The function *usb_fill_int_urb()* is a helper function to properly initialize a urb to be sent to a interrupt endpoint of a USB device. Bulk urbs are initialized much like interrupt urbs. The function that does this is *usb_fill_bulk_urb()*, the control urbs are initialized almost the same way as bulk urbs, with a call to the function *usb_fill_control_urb()*, the isochronous urbs unfortunately do not have an initializer function like the interrupt, control, and bulk urbs do. So they must be initialized "by hand" in the driver before they can be submitted to the USB Core.
3. Submitted to the USB core, by the USB device driver. USB device driver uses the function *usb_submit_urb()* to submit urb data strucuture to USB core.
4. Submitted to the specific USB host controller driver for the specified device by the USB core.
5. Processed by the USB host controller driver that makes a USB transfer to the device.
6. When the urb is completed, the USB host controller driver notifies the USB device driver. If USB device is disconnected by some must-interrupted reasons, USB Core calls either the function *usb_kill_urb()* or *usb_unlink_urb()* and destroy the urb data structure. Figure 10 shows the urb handling procedure [17].

USB keyboard is a kind of Human Interface Devices (HID) which can be devices with either low or full speed rate. In addition to control endpoint, the IN/OUT of data transmission uses interrupt endpoint. IN endpoint is in charge of every key pressing value, whereas OUT endpoint manages the LED of keyboard driver.

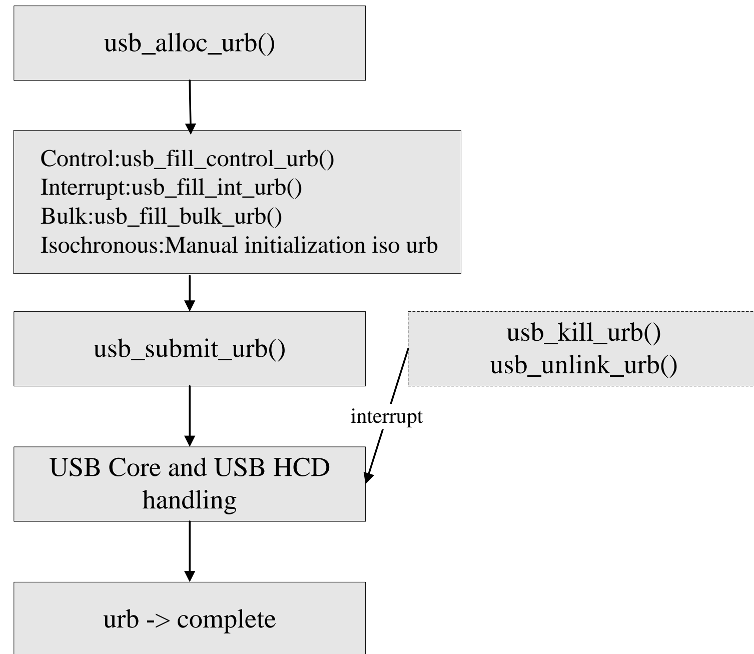


**Figure 10. urb handling procedure**

The USB keyboard driver of Linux must implement the following items. They are the functions of *usb_driver* data structure field (e.g. *usb_kbd_prob()*, *usb_kbd_disconnect()*), the initialization of keyboard (e.g. *usb_kbd_init()*), the cancellation of keyboard (e.g. *usb_kbd_exit()*), keyboard open (e.g. *usb_kbd_open()*), keyboard close (e.g. *usb_kbd_close()*) and the functions of interrupt handler (e.g. *usb_kbd_irq()*).

When the USB keyboard driver is initialized, it registers the function *usb_kbd_irq()* of USB keyboard driver to the *irq_handle* of USB device data structure. In the meantime, USB keyboard driver applies the endpoint of IN/OUT through USB Core and is mounted on periodic list. Host controller acquires the data from USB keyboard buffer according to the period. Finally, Input subsystem outputs the data from keyboard buffer, and then USB keyboard driver and USB keyboard function properly.

USB Mouse is one of Human Interface Devices (HID), like USB keyboard. In addition to control endpoint, the IN/OUT of data transmission uses interrupt endpoint. IN endpoint manages the signaling of every mouse clicking.

The USB mouse driver of Linux must implement the following items. They are the functions of *usb_driver* data structure field (e.g. *usb_mouse_prob()*, *usb_mouse_disconnect()*), the initialization of keyboard (e.g. *usb_mouse_init()*), the cancellation of keyboard (e.g. *usb_mouse_exit()*), keyboard open (e.g. *usb_mouse_open()*), keyboard close (e.g. *usb_mouse_close()*) and the functions of interrupt handler (e.g. *usb_mouse_irq()*).

When the USB mouse driver is initialized, it registers the function *usb_kbd_irq()* of USB mouse driver to the *irq_handle* of USB device data structure. In the meantime, USB mouse driver applies the endpoint of IN/OUT through USB Core and is mounted on periodic list. Host controller acquires the data from USB mouse buffer according to the period. Finally, Input subsystem outputs the data from mouse buffer, and then USB mouse driver and USB mouse function properly.

### 3.4 Modification of LyraOS Wrappers

The modification of LyraOS wrappers includes the modification of OS service wrapper and addition of USB device specific wrapper.

Slab allocator is implemented in Linux kernel, however, not in LyraOS. The allocated memory in slab allocator is divided into the planned-sized slabs in order to be accessed efficiently. Owing to the complex implementation of slab allocator, we just make modifications over some functions to avoid the complicated slab allocator procedure while the driver is still capable of accessing memory space properly.

At first, we implement the function *kmem_cache_create()* which is the function of Linux slab allocator to create an object cache for a specific object to create a *kmem_cache* data structure. The *obj_size* member of the *kmem_cache* data structure records the allocated size of memory. The function *kmem_cache_zalloc()*, which is the function of Linux slab allocator to allocate memory space for an object from the object cache, is implemented to use LyraOS's memory allocation function named *malloc()* to memory space using the stored *obj_size* in the *kmem_cache* data structure. Another slab allocators's function *kmem_cache_free()* which is used to release memory space is implemented by invoking *free()* function in LyraOS, and the function *kmem_cache_destroy()* is implemented to releases the *kmem_cache* data structure.

The input subsystem is the part of the Linux kernel that manages the various input devices (such as keyboards, mice, joysticks, tablets, and a wide range of other devices) that a user uses to interact with the system. This subsystem is included in the kernel because these input devices usually are accessed through special hardware interfaces such as serial ports, PS/2 ports, and the Universal Serial Bus, which are protected and managed by the kernel. The kernel then exposes the user input in a consistent, device-independent way to user space through a range of defined APIs [18].

The transplanted USB device driver requires Linux kernel's input subsystem which is what LyraOS lacks. Therefore, we transplant the input subsystem in Linux kernel 2.6 to LyraOS.

Figure 11 depicts the structure of input subsystem. Input subsystem is composed of driver, input core, and event handler. Input events, such as USB keyboard key pressing and USB mouse movement, would trigger event handler to display the values of key pressing on the screen after being processed by device driver and input core.

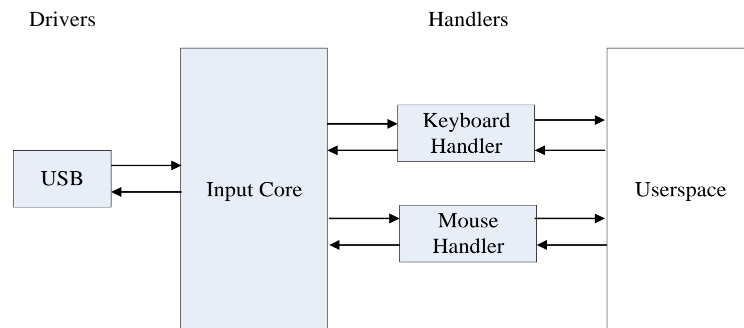


**Figure 11. Structure of input subsystem**

USB wrapper supports the required data structures and kernel functions. When *usb_init()* function to initiate, it needs to use the following func-

tions: *devices_init()*, *device_register()*, *device_unregister()*, *buses_init()*, *bus_register()*, *bus_unregister()*, *classes_init()*, *class_register()*, *class_unregister()*, *kset_init()*, *kset_register()*, *kset_unregister()*, *kset_add()*, *kobject_init()*, *kobject_set_name()*, *kobject_register()*, *my_object_release()*, *kobject_add()*, *kobject_del()*, *subsystem_init()*, *subsystem_register()*, etc.

The functions that must be used in USB host driver and USB Core are *dma_pool_create()*, *dma_pool_destory()*, *register_chrdev()*, *schedule_timout()*, *prepare_to_wait()*, *snprintf()*, *scnprintf()*, *kref_get()*, *kref_pur()*, *kobj_map()*, *kobj_unmap()*, *cdev_get()*, *register_chrdev()*, *__unegister_chrdev_region()*, *cdev_dynamic_release()*, *exact_lock()*, *cdev_alloc()*, *cdev_add()*, *cdev_purge()*, *delay_loop()*, *__underlay()*, *find_next_zero_bit()*, *msleep()*, *up_write()*, *pte_alloc_one()*, *remap_pmd_range()*, *pud_alloc()*, *pmd_alloc()*, *remap_pud_range()*, *remap_pfn_range()*, *bitmap_scnprintf()*, *cap_capable()*, *param_set_bool()*, *param_get_bool()*, *register_chrdeb_region()*, etc.

Mapping functions deal with the mapping of functions whose tasks are identical between Linux and LyraOS, however, the naming of the functions is different. The typical example is that the function *printk()* in Linux and the function *printf()* in LyraOS have the same functionality, but they have different function names. Other examples are like *jiffies_64()*, *kmalloc()*, *__get_free_pages()*, *spin_lock()*, *spin_unlock()*, *schedule()*, *kthread_create()*, *yield()*, *kthread_stop()*, *request_irq_register_isr()*, *free_irq()*, *thread_sleep()*, *thread_wakeup()*, etc.

## 4. Experimental Results

In this section, we make some analysis and statistics of codes of transplanted USB device drivers in LyraOS. Table 2 shows our experimental platform and USB devices.

Table 3 lists code lines of LyraOS wrapper and Device Manager component, where the statistic of the lines contains programming comments. Table 4 lists code lines of Linux device drivers.

| CPU | AMD Athlon 64X2 Dual Core Processor 4200+ |
|---|---|
| System Memory | DDR2 1G |
| Operating System | LyraOS 2.1.17 |
| USB Keyboard | ViewSonic USB 1.1 |
| USB Mouse | Acer USB1 .1 |

**Table 2. Experimental platform**

| Description | Original Codes | Added Codes |
|---|---|---|
| Device Manager | 776 | 0 |
| IRQ Management | 876 | 0 |
| Timer Management | 59 | 392 |
| Wait Queue Management | 174 | 0 |
| Slab Allocator | 0 | 100 |
| Input Core | 0 | 1162 |
| Mapping Functions | 124 | 18 |
| USB Driver Wrapper | 0 | 5388 |
| Total | 2009 | 7060 |

**Table 3. The line of codes of Device Manager and LyraOS wrapper**

| Description | Line of Codes |
|---|---|
| USB Host Controller Drivers- EHCI | 6631 |
| USB Host Controller Drivers- OHCI | 4323 |
| USB Host Controller Drivers- UHCI | 3418 |
| USB Core | 12282 |
| Human Interface Devices | 3070 |
| USB keyboard | 366 |
| USB mouse | 248 |
| Total | 30338 |

**Table 4. The line of codes of device drivers**

Table 5 lists object code sizes of the LyraOS wrapper and Device Manager component. Table 6 lists the object code sizes of the USB driver wrapper, UHCI host controller driver, USB Core, HID, USB keyboard driver, and USB mouse driver.

| Description | Object Code Sizes (bytes) | Percentage |
|---|---|---|
| Device Manager | 10552 | 10.7% |
| IRQ Management | 11912 | 12% |
| Timer Management | 4176 | 4.2% |
| Wait Queue Management | 19 | 0% |
| Slab Allocator | 11 | 0% |
| Input Core | 14716 | 15% |
| Mapping Functions | 4724 | 4.8% |
| USB Driver Wrapper | 52540 | 53.3% |
| Total | 98650 | 100% |

**Table 5. The object codes size of Device Manager component and LyraOS wrapper**

| Description | Object Code Sizes(bytes) | Percentage |
|---|---|---|
| USB Host Controller Drivers- EHCI | 26432 | 13% |
| USB Host Controller Drivers- OHCI | 5540 | 3% |
| USB Host Controller Drivers- UHCI | 19272 | 9.3% |
| USB Core | 103724 | 50% |
| Human Interface Devices | 33916 | 16.5% |
| USB keyboard driver | 10228 | 5% |
| USB mouse driver | 6674 | 3.2% |
| Total | 205786 | 100% |

**Table 6. The object code size of device drivers**

## 5. Conclusions and Future Work

The development of device drivers in Linux is very mature and the source codes are available to be modified, added, and investigated in public due to the feature of open source. Therefore, we aim at transplanting the source codes of device drivers from Linux into LyraOS without major modifications. In this paper, our major tasks include (1) transplanting USB host controller drivers, USB Core, USB keyboard and mouse drivers into LyraOS, (2) implementing the required data structures and functions for supporting these transplanted device drivers, and (3) integrating the transplanted USB device drivers into LyraOS's Driver Manager component LyraDD.

The development of device drivers is a quite time-consuming task. In order to shorten the time of development of device drivers and make the USB drivers properly operate on our LyraOS, the component-based embedded operating system, without significant modifications, we transplant source codes of device drivers from Linux and wrap them with wrappers. Moreover, this concept can be also used in transplanting other device drivers into operating systems.

However, the data structures, functions, and the process of USB device drivers are quite complicated, which significantly increases the difficulties of transplanting device drivers from Linux. Therefore, the thorough understanding over Linux kernel and USB device drivers is required and helps us to successfully transplant the device drivers.

We have transplanted some USB device drivers, including USB keyboard and mouse drivers into our embedded operating system - LyraOS. Hence, any future appendant of other USB device drivers is helpful for LyraOS to support more USB devices. In the future, we hope that LyraOS is capable of supporting more diverse USB devices, like USB drives, USB disks, etc.

## References


[1] LyraOS homepage, http at http://163.22.32.199/joannaResearch/LyraOS/index.htm.

[2] Z. Y. Cheng, M. L. Chiang, and R. C. Chang, "A Component Based Operating System for Resource Limited Embedded Devices," IEEE International Symposium on Consumer Electronics (ISCE'2000), Hong Kong, Dec. 5-7, 2000.

[3] Chi-Wei Yang, C. H. Lee, and R. C. Chang, "Lyra: A System Framework in Supporting Multimedia Applications," IEEE International Conference on Multimedia Computing and Systems'99, Florence, Italy, June 1999.

[4] Zan-Yu Chen, "A Component Based Embedded Operating System," Master Thesis, Department of Information and Computer Science, National Chiao-Tung University, June 2000.

[5] eCos, http://sources.redhat.com/ecos/.

[6] MicroC/OS-II, at http://www.ucos-ii.com/.

[7] Mei-Ling Chiang and Ching-Ru Lo, "LyraFILE: A Component-Based VFAT FileSystem for Embedded Systems," International Journal of Embedded Systems, Vol. 2, Nos. 3/4, pp. 248-259, Aug 2007.

[8] H. K. Ting, C. R. Lo, M. L. Chiang, and R. C. Chang, "Adapting LINUX VFAT Filesystem To Embedded Operating Systems," International Computer Symposium (ICS'2002), HwaLian, Taiwan, R.O.C., 2002.

[9] Chun-Hui Chen, "LyraDD: Design and Implementation of the Device Driver Model for Embedded Systems," Master Thesis, Department of Information Management, National Chi-Nan University, June 2004.

[10] C. W. Yang, "An Integrated Core-Work for Fast Information-Appliance Buildup," Master Thesis, Department of Information and Computer Science, National Chiao Tung University, June 1998.

[11] Chi-Wei Yang, Paul C. H. Lee, and R. C. Chang, "Reuse Linux Device Drivers in Embedded Systems," Proceeding of the 1998 International Computer Symposium (ICS'98), Taiwan, 1998.

[12] Jer-Wei Chuang, Kim-Seng Sew, Mei-Ling Chiang, and Ruei-Chuan Chang, "Integration of Linux Communication Stacks into Embedded Operating Systems," International Computer Symposium (ICS'2000), Taiwan, December 6-8, 2000.

[13] Daniel P. Bovet and Marco Cesati, Understanding the Linux kernel, 3rd edition, O'REILLY, November 2005.

[14] Mei-Ling Chiang and Yun-Chen Lee, "LyraNET: A Zero-Copy TCP/IP Protocol Stack for Embedded Systems," Journal of Real-Time Systems, Vol. 34, No. 1, pp. 5-18, Sep. 2006.

[15] Bor-Yeh Shen and Mei-Ling Chiang, "A Server-side Pre-linking Mechanism for Updating Em-bedded Operating System Dynamically," Journal of Information Science and Engineering, Vol. 26, No. 1, Jan. 2010.

[16] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, "Linux Device Drivers," 3rd edition, O'REILLY, February 2005.

[17] Jing Chen and Jun-Lin Huang, "The Design and Implementation of Universal Serial Bus Driver in a Microkernel Operating System," Master Thesis, Department of Electrical Engineering, National Cheng Kung University, July 2009.

[18] Brad Hards, The Linux USB Input Subsystem, Part I, Linux Journal, at http://www.linuxjournal.com/article/6396, February 1st, 2003.