QUY TRÌNH VIẾT DRIVER CHO CÁC THIẾT BỊ THEO CHUẨN USB TRONG HỆ THỐNG NHÚNG LINUX

Ngô Thị Vinh, Đoàn Ngọc Phương*, Ngô Hữu Huy
Trường Đại học Công nghệ Thông tin và Truyền thông – ĐH Thái Nguyên

TÓM TẮT

Để điều khiển được các thiết bị ngoại vi, các module mở rộng của các hệ thống nhúng, người lập trình thường phải tự viết các chương trình điều khiển – Driver cho các cổng giao tiếp thay vì sử dụng các Driver có sẵn của nhà cung cấp thiết bị vì tính riêng biệt của các hệ nhúng. Việc viết Driver cho các cổng giao tiếp luôn được coi là một công việc hết sức quan trọng và tốn nhiều công sức của người lập trình, do đó cần có các quy trình rõ ràng để cụ thể hóa và đơn giản hóa công việc này. Bài báo trình bày về quy trình viết Driver cho một thiết bị USB – một chuẩn giao tiếp hết sức phổ biến hiện nay, Driver được xây dựng cho các hệ thống sử dụng hệ điều hành Linux và nhân ARM là những nền tảng phổ biến trên các hệ thống nhúng hiện đại.

Từ khóa: *Linux, Tiny 6410, vi điều khiển ARM11, hệ điều hành nhúng, USB Driver*

GIỚI THIỆU

Khi xây dựng ứng dụng cho các hệ nhúng thì một trong những công việc quan trọng nhất của người lập trình đó chính là lập trình ghép nối để điều khiển các module, các thiết bị ngoại vi ghép nối với hệ vi điều khiển trung tâm. Nếu các module và các thiết bị được ghép nối với các hệ thống tương thích với nhau về chuẩn giao tiếp và giao thức truyền thông thì chúng có thể hoạt động và trao đổi dữ liệu cho nhau. Các hệ nhúng thường sử dụng các giao thức truyền thông phổ biến như Ethernet, RS232, SPI, I2C và đặc biệt ngày nay là giao thức USB. Sử dụng giao thức USB có nhiều ưu điểm [1] như dễ sử dụng, tốc độ truyền cao chỉ sau chuẩn Ethernet [1], độ tin cậy cao, chi phí thấp, yêu cầu điện áp nguồn nuôi nhỏ (+5V). Có được những hiểu biết sâu sắc về chuẩn USB sẽ giúp người lập trình có thể thực hiện được rất nhiều công việc như: thiết kế, chế tạo thiết bị hoạt động theo chuẩn USB, viết driver cho thiết bị giao tiếp theo chuẩn USB, lập trình ghép nối với các thiết bị hoạt động theo chuẩn USB.

Bài báo sẽ trình bày các bước viết một driver cho một thiết bị USB trên hệ điều hành nhúng Linux được cài đặt trên kít ARM Tiny6410 [8]. Linux là một trong những hệ điều hành được phát triển rộng rãi cho các hệ nhúng với phần nhân có kích thước rất nhỏ gọn và miễn phí [4]. Ngoài ra, Linux hỗ trợ trình biên dịch cho các ứng dụng được viết bằng C/C++ và java [4]. Đây là hai trong ba ngôn ngữ lập trình được sử dụng nhiều nhất thế giới bởi tốc độ chạy nhanh và không phụ thuộc vào nền phần cứng của chúng.

CHUẨN USB

Tín hiệu: Chuẩn USB sử dụng 4 đường tín hiệu trong đó có 2 đường cấp nguồn DC là VBUS-5V và đường GND. 2 đường còn lại là một cặp tín hiệu vi sai D+ và D- cho phép truyền dữ liệu [1].

Thiết bị USB: Các thiết bị USB có thể được chia làm 3 loại chính [1] dựa theo vài trò của chúng: (1) USB Host là thiết bị đóng vai trò điều khiển toàn bộ mạng USB. Để giao tiếp và điều khiển các USB device, Bộ điều khiển USB Host controller cần được thiết kế tích hợp với một USB RootHub. USB Host có chức năng trao đổi dữ liệu với các USB Device, điều khiển bus USB, quản lý các thiết bị cắm vào hay rút ra khỏi Bus USB qua quá trình định danh, phân xử, quản lý luồng dữ liệu trên Bus, đảm bảo các thiết bị đều có cơ hội trao đổi dữ liệu tùy thuộc vào cấu hình của mỗi thiết bị. (2) **USB Device** là các thiết bị đóng vai trò như các slave giao tiếp với USB Host. Các USB Device là các thiết bị bị

* *Tel: 0979 479940*

động, quá trình trao đổi dữ liệu của chúng đều phải thông qua quá trình điều phối của USB Host. (3) **USB Hub** đóng vai trò như các Hub trong mạng Ethernet để cấp nguồn cho các thiết bị USB.

Hoạt động của chuẩn USB: được chia làm hai giai đoạn chính gồm quá trình định danh và quá trình truyền dữ liệu. Quá trình định danh là quá trình USB Host phát hiện các thiết bị cắm vào và rút ra khỏi đường USB Bus. Mỗi khi một thiết bị tham gia vào Bus USB, USB Host sẽ tiến hành đọc các thông tin mô tả của USB Device rồi thiết lập địa chỉ NodeID và chế độ hoạt động tương ứng cho thiết bị USB Device. Các địa chỉ sẽ được đánh một số nguyên từ 1 đến 126. Khi thiết bị rút ra khỏi đường Bus, địa chỉ này sẽ được thu hồi. **Quá trình truyền dữ liệu** liên quan đến hai khái niệm là **Interface** và **Endpoint**. Một thiết bị USB có thể có nhiều Interface và một Interface có thể sử dụng nhiều Endpoint khác nhau. Các Endpoint đóng vai trò như các bộ đệm truyền/nhận dữ liệu. Nhờ việc sử dụng nhiều bộ đệm mà các quá trình truyền thông được tiến hành song song và cho tốc độ cao hơn, đồng thời giúp cho việc phân tách các dịch vụ khác nhau dễ dàng. Với chuẩn USB, các thiết bị được thiết kế với tối đa là 16 Endpoint. Các Endpoint được phân loại theo hướng truyền dữ liệu nhìn từ phía USB Host theo cách: Các Endpoint truyền dữ liệu từ USB Device tới USB Host gọi là endpoint IN, còn các Endoint truyền dữ liệu từ USB Host tới USB Device là endpoint OUT [1].

Chuẩn USB cung cấp bốn chế độ truyền [1] dùng cho các mục đích khác nhau. Chế độ truyền điều khiển là chế độ truyền được tất cả các thiết bị USB hỗ trợ để truyền các thông tin điều khiển với tốc độ tương đối chậm. Chế độ truyền theo ngắt sử dụng cho các thiết bị cần truyền một lượng dữ liệu nhỏ, tuần hoàn theo thời gian như chuột, bàn phím. Chế độ truyền theo khối sử dụng cho các thiết bị cần truyền một lượng dữ liệu lớn, yêu cầu độ chính xác tuyệt đối và không có ràng buộc quá chặt chẽ về thời gian thực như các thẻ nhớ USB, máy in. Chế độ truyền đẳng thời sử dụng cho các thiết bị cần truyền một lượng dữ liệu lớn với tốc độ rất nhanh, đảm bảo ràng buộc về thời gian thực nhưng chấp nhận với một độ chính xác ở một mức nhất định như các thiết bị nghe nhạc, xem phim kết nối theo chuẩn USB.

QUY TRÌNH VIẾT DRIVER CHO THIẾT BỊ USB TRÊN HỆ ĐIỀU HÀNH LINUX

Hệ thống USB trong Linux được phân làm nhiều tầng [2], [5]. Tầng Driver của các USB nằm giữa hai tầng là tầng lõi và hệ thống các tầng con khác nhau.

Tầng lõi của USB do hệ điều hành Linux cung cấp, nó là một tập các hàm API [5] nhằm trừu tượng hóa các nền phần cứng khác nhau và các thành phần phụ thuộc. Tầng Driver là tập các API dẫn xuất cho các tầng con ở lớp trên. Tùy thuộc vào mục đích sử dụng mà người lập trình cần sử dụng các API phù hợp với các thiết bị.

Để viết Driver cho một thiết bị USB trên Linux người lập trình cần thực hiện các bước như sau:

Bước 1: Tìm hiểu thông tin về thiết bị USB. Người lập trình cần biết thông tin cần thiết về Firmware của thiết bị bao gồm [2]: định danh nhà sản xuất (idvendor), định danh về sản phẩm (idproduct), số lượng các cấu hình Configuration, số lượng các Interface trong từng cấu hình, số lượng và loại Endpoint trong từng Interface. Trên hệ điều hành Linux ta chỉ việc kết nối thiết bị tới máy tính và gõ lệnh lsusb [5] trên terminal để xem danh sách các thiết bị USB và cấu hình của chúng.

Bước 2: Khai báo danh sách các thiết bị có thể được điều khiển bởi Driver. Người lập trình cần phải chỉ định Driver sẽ được sử dụng cho các thiết bị hoặc các lớp thiết bị nào. Cấu trúc usb_device_id cung cấp một kiểu thiết bị, nó có thể là một thiết bị cụ thể cũng có thể là một lớp thiết bị. Một số macro có thể được sử dụng để khởi tạo cấu trúc này [2], [5] như sử dụng macro USB_DEVICE

(vendor, product) để tạo ra một cấu trúc usb_device_id với IDvendor và IDproduct.

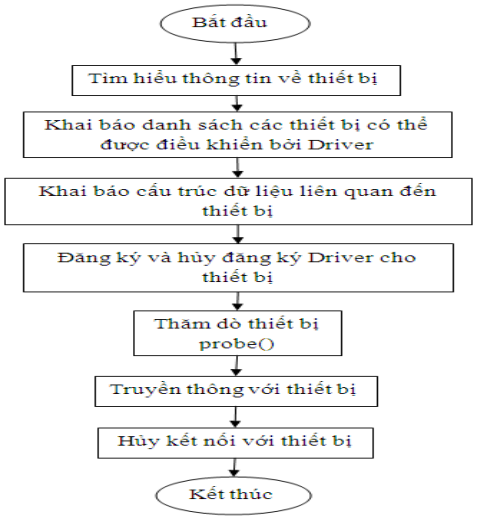
Sau khi đã tạo ra một cấu trúc usb_device_id người lập trình cần phải khai báo cấu trúc này với tầng lõi sử dụng macro MODULE_DEVICE_TABLE [5].

Bước 3: Khai báo cấu trúc dữ liệu liên quan tới thiết bị. Trong quá trình thăm dò thiết bị người lập trình cần lưu lại các thông tin cần thiết như: định danh nhà sản xuất, định danh thiết bị, thông tin các Configuration, Interface, Endpoint của thiết bị. Tùy theo đặc điểm của từng phần cứng cụ thể để ta định nghĩa ra một cấu trúc dữ liệu phù hợp.

Cấu trúc dữ liệu có dạng:

```
struct usb_mydevice {
    struct sb_device *udev;(1)
    struct usb_inter *inter;(2)
    unsigned char *in_buffer;(3)
    size_t in_size;(4)
    __u8 in_endpointAddr;(5)
    __u8 out_endpointAddr;(6) };
```

Trong đó: (1) và (2) là con trỏ cấu trúc mô tả thiết bị, (3) là bộ đệm dữ liệu vào, (4) là kích thước bộ đệm dữ liệu vào, (5) (6) là địa chỉ của hai Endpoint IN và OUT.



Hình 1. *Quy trình viết USB Driver*

Bước 4. Đăng kí và hủy đăng kí Driver cho thiết bị USB Device. Để tầng USB Core có thể nhận ra Driver, người lập trình cần phải đăng kí Driver sử dụng hàm sau:

usb_register(struct usb_driver &);

Hàm này thường được gọi trong hàm khởi tạo Driver. Tham số cần truyền cho hàm này là một con trỏ tới cấu trúc usb_driver [2]. Cấu trúc này chứa các thông tin về Driver đang viết bao gồm: (1) tên của Driver, (2) con trỏ tới bảng chứa các thiết bị sẽ được điều khiển bởi Driver đã được khai báo MODULE_DEVICE_TABLE(), (3) một con trỏ tới một hàm thăm dò, hàm này sẽ được gọi khi thiết bị được kết nối tới hệ thống để xác định các Endpoint, cấp phát bộ nhớ…(4) Một con trỏ tới một hàm ngắt kết nối, hàm này sẽ được gọi khi thiết bị được gỡ bỏ ra khỏi hệ thống.

Bước 5. Thực hiện thăm dò thiết bị với hàm probe(). Khi thiết bị được kết nối tới hệ thống, hàm thăm dò [2]của Driver sẽ được gọi với một con trỏ trỏ tới cấu trúc usb_interface mô tả Interface được chọn trên thiết bị do tầng lõi USB truyền tới. Hàm thăm dò sẽ thực hiện các công việc như lấy ra địa chỉ các Endpoint cần dùng và kích thước các bộ đệm cho thiết bị, cấp phát bộ đệm, lưu lại các thông tin như địa chỉ Endpoint, kích thước bộ đệm, địa chỉ bộ đệm…, đăng kí lớp thiết bị cho Driver.

Bước 6: Thực hiện đọc/ghi dữ liệu từ thiết bị. Để đọc ghi dữ liệu từ thiết bị USB ta phải lấy các thông tin từ thiết bị sử dụng hàm usb_get_intfdata() và thiết lập dữ liệu cho cấu trúc file như sau:

dev = usb_get_intfdata(interface);

file->private_data = dev;

Cấu trúc file được định nghĩa trong <linux/fs.h>[5] là một cấu trúc quan trọng trong không gian nhân của Linux rất cần thiết cho việc viết Driver của USB. Sau đó, người lập trình sẽ thực hiện đọc, ghi dữ liệu từ thiết bị sử dụng các hàm read() và write() [5].

Bước 7: Hủy kết nối tới thiết bị bằng hàm disconnect(). Khi thiết bị được gỡ bỏ ra khỏi hệ thống, hàm ngắt kết nối được gọi. Hàm disconnect sẽ thực hiện hai công việc chính là hủy các dữ liệu về thiết bị đã lưu trữ từ hàm thăm dò bằng cách thiết lập giá trị NULL cho

interface và hủy đăng kí lớp thiết bị qua hàm usb_deregister_dev().

KẾT QUẢ THỰC NGHIỆM

Chúng tôi đã tiến hành viết Driver cho một Camera USB được kết nối tới kít Tiny6410 cài hệ điều hành nhúng Linux. Kết quả cho thấy hệ điều hành cài trên kít đã nhận được USB Camera của hãng Tako, với 100 lần thử kết nối tỉ lệ thành công là 100%, thời gian nhận thiết bị trung bình là 25 giây, chương trình Driver không gây tranh chấp về cổng khi cắm đồng thời 2 thiết bị trên 2 cổng USB của kit. Ảnh thu được từ Camera được hiển thị trên màn hình của kít.

Hình 2. *Kết quả thu ảnh từ camera USB của hãng Tako trên kít Tiny6410 cài Linux*

KẾT LUẬN

Bài báo đã trình bày quy trình viết một Driver cho một thiết bị USB trên hệ điều hành nhúng Linux. Đây cũng là các bước chung mà người lập trình có thể tham khảo khi viết Driver cho một thiết bị USB trên nền của hầu hết các hệ điều hành nhúng khác. Bài báo đã thực hiện viết Driver cho một USB Camera của hãng Tako và kiểm chứng kết quả trên kít ARM11 Tiny6410.

TÀI LIỆU THAM KHẢO

1. Jan Axelson (2009), USB Complete: The Developer's Guide, Fourth Edition, Lakeview Research LLC, 5310 Chinook Ln., Madison WI 53704.

2. Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman (2013), Linux Device Drivers, O'Reilly.

3. Greg Kroah-Hartman (2001), How to Write a Linux USB Device Driver, http://www.linuxjournal.com/article/4786.

4. Graham Glass, King Ables, (2006), Linux for Programmers and Users, Prentice Hall,

5. Michael Opdenacker (2012) **,** Linux USB device drivers, Free Electrons.

6. Rajaram Regupathy, Bootstrap Yourself with Linux-USB Stack: Design, Develop, Debug, and Validate Embedded USB, Course Technology PTR, March 16, 2011

7.http://www.linux-drivers.org/usb_webcams.html

8. http://api.haiku-os.org/usb_modules.html.

9. www.minidevs.com, Tiny6410 Development Kit, 2013

SUMMARY

THE PROCESS OF WRITING DRIVER FOR USB DEVICES ON LINUX EMBEDDED SYSTEM

Ngo Thi Vinh, Doan Ngoc Phuong[*], Ngo Huu Huy

College of Information and Communication Technology - TNU

To control the peripherals, expansion modules of embedded systems, programmers often have to write the driver - Driver for the interface instead of using the available Driver suppliers equipment because of the peculiarities of embedded systems. Writing Driver for the interface are always considered to be a very important job and spend a lot of programming effort, so there should be clear procedures to concretize and simplify this task. This paper presents the process of writing a device driver for USB - a communication standard that is extremely popular today, Driver was developed for systems using the Linux operating system and ARM is the common platform on modern embedded systems.

Key words: *Linux Embedded, Tiny 6410, ARM11 microprocessor, Embedded Operating System, USB Driver*

Ngày nhận bài:25/01/2014; Ngày phản biện:10/02/2014; Ngày duyệt đăng: 26/02/2014

Phản biện khoa học: *TS. Phùng Trung Nghĩa – Trường ĐH Công nghệ Thông tin & Truyền thông - ĐHTN*

[*] *Tel: 0979 479940*