BÁO CÁO ĐỒ ÁN 1 - MÔN HỆ ĐIỀU HÀNH

Trần Quốc Cường - 1612843

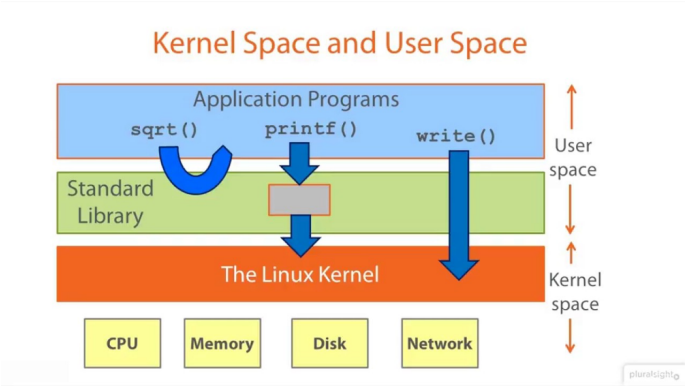
September 2018

1 Kernel Module là gì?

Một Loadable Kernel Module (LKM) là một cơ chế cho phép thêm và sửa code vào Linux kernel (nhân Linux) trong thời gian thực chạy. LKM vô cùng phù hợp cho driver của các thiết bị, cho phép kernel giao tiếp với phần cứng mà không cần phải biết phần cứng làm việc thế nào.

Nhờ có khả năng chỉ lắp ghép các module cần thiết mà Linux Kernel tránh việc trở nên rất lớn khi phải cài sẵn tất cả driver của các thiết bị. Chúng ta cũng hạn chế được việc phải "rebuild" kernel mỗi khi phải thêm hay cập nhật driver mới. Kernel module được thực thi như là một phần của Kerel, không phải trong user space (không gian người dùng).

Kernel module và ứng dụng người dùng chạy trong 2 không gian khác nhau (kernel space và user space). Chúng cũng sử dụng 2 địa chỉ ô nhớ khác nhau. Điều này giúp cho ứng dụng người dùng chạy ổn định hơn và tránh sự giao tranh tài nguyên của các ứng dụng người dùng.



Hình 1: Kernel space và User space

2 Cách viết code và gắn Kernel Module vào hệ thống

2.1 Kernel Module code

Kernel module được viết bằng bằng C và không phải là một ứng dụng do đó code của nó không có hàm main() và có những đặc tính đặc biệt sau:

- Không thực hiện tuần tự: Những kernel module đăng ký nó bằng hàm khởi tạo, chúng xử lý bằng những module đã được chỉ sẵn trong source code.
- Không được dọn rác tự động: Những tài nguyên hay bộ nhớ mà kernel module sử dụng phải được dọn dẹp thủ công hoặc là sẽ không thể sử dụng cho đến khi khởi động lại kernel
- Không có hàm printf(): Kernel module không thể sử dụng các thư viện của không gian người dùng, có thể dùng hàm printk() để thay thế
- Có thể bị gián đoạn: Kernel module có thể được sử dụng bởi nhiều ứng dụng cùng một lúc. Chúng ta phải xử lý trường hợp này để tránh gây ra lỗi không mong muốn
- Có quyền thực thi ở mức độ cao: do nằm ở kernel space nên kernel module có quyền truy cập cao hơn ứng dụng bình thường
- Không hỗ trợ số thực dấu phẩy động: code của nhân sử dụng traps để chuyển từ số nguyên sang số chấm phẩy động điều này không được khuyên dùng và nên được thực hiện tại không giao người dùng

Source code của một kernel module gồm nhiều phần (tham khảo *kernel_module.c*)
- Các thư viện cần sử dụng
- Khai báo các thông tin

```
///< The device will appear at /dev/randomness... using this value
#define  DEVICE_NAME "randomness_generator"
///< The device class -- this is a character device driver
#define  CLASS_NAME  "first_project"

///< The license type -- this affects available functionality
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Quoc-Cuong TRAN");     // modinfo
MODULE_DESCRIPTION("First Project in Operating system course");
MODULE_VERSION("1.0");
```

- Các thao tác mà chúng ta sẽ xử lý
- Khai báo hàm nào dùng để khởi tạo và thoát ở cuối file code của kernel module

```
module_init(RNG_init);
module_exit(RNG_exit);
```

2.2 Cách chèn và xóa module vào Linux kernel

Một Makefile tiêu chuẩn để build một module từ file code tên hello.c

```
obj-m+=hello.o
all:
 make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules
clean:
 make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean
```

- Khi ta gọi lệnh **make** từ Terminal thì chương trình sẽ tạo ra những file cần thiết để tạo module rồi "thêm" *hello.o* vào *obj-m*. *obj-m* chỉ ra rằng file *hello.o* vừa tạo là một LKM. Dùng lệnh **ls** ta sẽ thấy một file module mới được tạo ra có tên hello.ko

- Tiến hành chèn vào kernel bằng lệnh: **sudo insmod hello.ko**. Kiểm tra module đã được gắn vào thành công chưa bằng lệnh: **lsmod | grep "hello"**

- Khi chương trình ở user space muốn gọi tới kernel module thì phải được cấp quyền bằng lệnh **sudo**

- Xóa module khỏi kernel bằng lệnh: **sudo rmmod hello**

- Dọn dẹp những file đã tạo bằng lệnh **make clean**

Những thông tin mà ta đã in ra dùng câu lệnh *printk()* có thể được xem ở file textit/var/log/kern.log.

3 Character Device Driver

3.1 Thế nào là Chrater Device Driver

Các Character Device thường được dùng để truyền và nhận dữ liệu từ các ứng dụng user space được truyền bằng các stream các character. Một thiết bị khác là Block Device cho phép truyền dữ liệu dưới dạng các buffer. Các loại thiết bị trên đều có thể truy cập qua những file thiết bị được gắn vào đường dẫn hệ thống (ví dụ: device ABC có thể truy cập qua đường dẫn /dev/ABC)

Device có một số chính và số phụ ứng với nó để định danh và gọi ở ngoài hệ thống hoặc ở trọng nội bộ driver.

3.2 Cấu trúc dữ liệu phục vụ quản lý file

file_operations được định nghĩa trong thư viện linux/fs.h giữ một con trỏ hàm trong driver mà cho phép ta định nghĩa một số thao tác xác định. Nếu ta không định nghĩa hàm nào thì hàm đó mặc định trả về **NULL** tương ứng với việc không thể sử dụng hàm đó.

3.3 Cách viết code cho Device Driver

Code ví dụ ta thực hiện khai báo thay đổi các thao tác open, read, write và release

```
static int     dev_open(struct inode *, struct file *);
static int     dev_release(struct inode *, struct file *);
static ssize_t dev_read(struct file *, char *, size_t, loff_t *);
static ssize_t dev_write(struct file *, const char *, size_t, loff_t *);

static struct file_operations fops = {
   .open = dev_open,
   .read = dev_read,
   .write = dev_write,
   .release = dev_release,
};
```

Tiến hành cài đặt các hàm đã khai báo theo đúng chuẩn.

Ta cũng nên tiến hành cài đặt thêm mutex để tránh tình trạng LKM bị lỗi đồng bộ khi được nhiều ứng dụng ở user space gọi.

4 Nguồn tham khảo

- Writing a Linux Kernel Module — Part 1: Introduction
- Writing a Linux Kernel Module — Part 2: A Character Device
- The Linux Kernel Module Programming Guide