CSC10007 - HỆ ĐIỀU HÀNH - CQ2018/21

ĐỒ ÁN 2

TÌM HIỂU VÀ LẬP TRÌNH LINUX KERNEL MODULE

Sinh viên thực hiện:

Nguyễn Huy Tú - 18120254

Nguyễn Duy Vũ - 18120264

Giáo viên hướng dẫn:

TS. Trần Trung Dũng

Ths. Lê Giang Thanh

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

MỤC LỤC

MỤC LỤC 2
BÁO CÁO ĐỒ ÁN 3
Thông tin nhóm. 3
Nội dung đồ án. 3
Tiến độ đồ án. 3
Mức độ hoàn thành. 3
Phân công thành viên. 3
NỘI DUNG TÌM HIỂU 5
Tổng quan về Linux. 5
Linux Kernel Module. 5
Driver. 5
Character Device Driver. 6
Tương tác với Kernel Module. 7
Các thư mục liên quan. 7
Các lệnh biên dịch. 8
Các kiểu dữ liệu. 8
Các hàm phát triển. 8
Các marco sử dụng. 9
BÁO CÁO KỸ THUẬT 10
Random Number Generator. 10
Tạo loadable kernel module. 10
Tạo character device. 10
Chương trình ở user space. 11
HƯỚNG DẪN SỬ DỤNG 12
Thực thi module ở kernel space. 12
Thực thi chương trình ở user space. 13
TÀI LIỆU THAM KHẢO 14

BÁO CÁO ĐỒ ÁN

1. Thông tin nhóm.

MSSV	Họ và tên	Email	Vai trò
18120254	Nguyễn Huy Tú	18120254@student.hcmus.edu.vn	Developer
18120264	Nguyễn Duy Vũ	vu38988@gmail.com	Developer

2. Nội dung đồ án.

Mục tiêu của đồ án là tìm hiểu về Linux kernel module và hệ thống quản lý tập tin và thiết bị trong Linux, giao tiếp giữa tiến trình ở user space và code kernel space.

- Viết một module dùng để tạo ra số ngẫu nhiên *(RNG - Random Number Generator)*.
- Module này sẽ tạo một character device để cho phép các tiến trình ở user space có thể open và read các số ngẫu nhiên.

3. Tiến độ đồ án.

3.1. Mức độ hoàn thành.

✓ Chức năng đã làm được: 100%.

✓ Phần chưa làm được: 0%.

3.2. Phân công thành viên.

Công việc	Thực hiện	% Hoàn thành
Viết báo cáo.	Nguyễn Huy Tú	100%
Viết module tạo số ngẫu nhiên.	Nguyễn Huy Tú	100%
Tạo character device cho phép tiến trình ở user space có thể open và read các RNG.	Nguyễn Huy Tú	100%
Tìm hiểu nội dung.	Nguyễn Duy Vũ	100%

Viết chương trình thao tác đọc ghi với file device ở user space.	Nguyễn Duy Vũ	100%

NỘI DUNG TÌM HIỂU

1. Tổng quan về Linux.

1.1. Linux Kernel Module.

Linux Kernel Module *(LKM - Loadable Kernel Module)* là một file với đuôi mở rộng là .ko *(kernel object)*. Mỗi hệ thống Linux qui định một nơi để chứa các modules đã được biên dịch có sẵn, thường được tổ chức trong thư mục:

```
/lib/modules/<kernel_version>/kernel
```

(với `<kernel_version>` *là phiên bản kernel đang sử dụng, có thể lấy bằng lệnh* `uname -r`*).*

Để giúp kích thước kernel được nhỏ gọn, trong quá trình hoạt động, module nào cần thiết sẽ được lắp vào kernel, còn module nào không cần thiết sẽ bị tháo ra khỏi kernel, công việc này gọi là **nạp động** *(dynamic loading)*, kernel không cần phải biên dịch lại khi có sự thay đổi.

1.2. Driver.

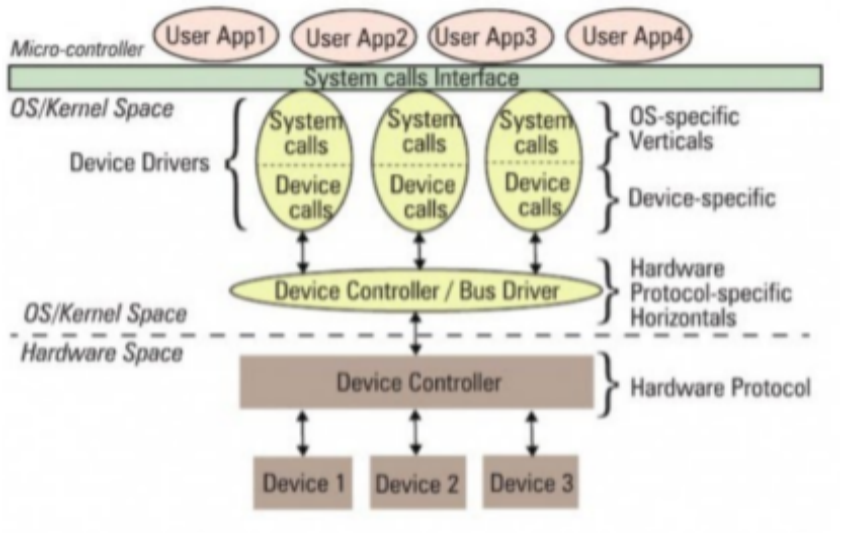
Một trong những kiểu LKM phổ biến nhất là **Driver** - một trình điều khiển có vai trò điều khiển, quản lý, giám sát một thực thể nào đó dưới quyền của nó.

Người ta sẽ thiết kế các **driver** dưới dạng các module tách rời với kernel. **Driver** được viết bằng C, nhưng không có hàm main() và được biên dịch giống cách biên dịch kernel (được lập trình theo kiểu hướng đối tượng trong C).

Mục đích quan trọng của các **driver** thiết bị là cung cấp một giao diện trừu tượng hóa cho người sử dụng, tức là cung cấp một giao diện lên tầng trên của hệ điều hành. Một cách tổng quan, một **driver** sẽ bao gồm 2 phần quan trọng:

- giao tiếp với thiết bị *(Device-specific)*
- giao tiếp với hệ điều hành *(OS-specific)*

Trên Linux, device driver cung cấp một giao diện **system call** đến tầng ứng dụng cho user, là một ranh giới giữa **kernel space** và **user space**. **System call** và **file system** cũng được thiết kế theo kiểu LKM. Phần cứng được quản lý bởi một **driver** mà có vai trò điều khiển một thành phần phần cứng khác gọi là **Device Controller**, kết nối với CPU thông qua bus.



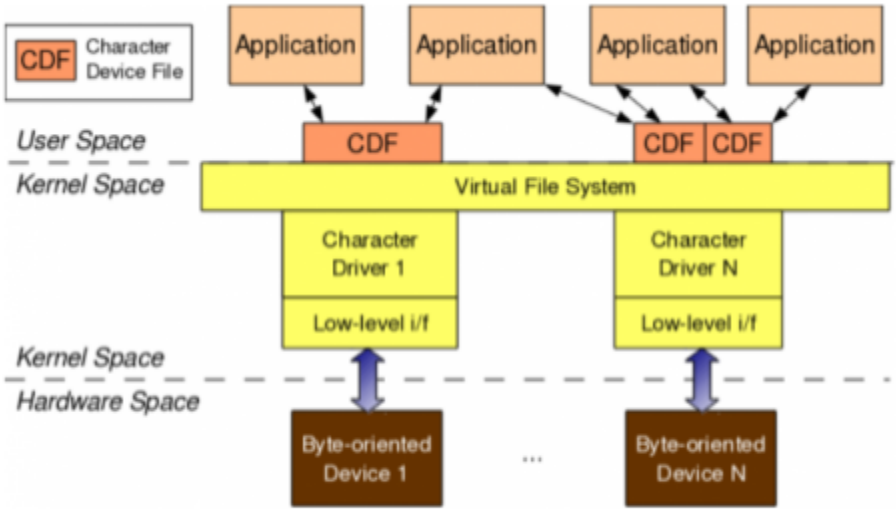
Hình 1. Các thành phần của Driver

1.3. Character Device Driver.

Driver trên Linux được phân chia thành 3 loại theo phân cấp chiều dọc. Ở đồ án này, ta chỉ quan tâm đến **driver** cho thiết bị kiểu có đặc tính trao đổi dữ liệu **hướng byte** *(byte-oriented)*, gọi là **Character Device Driver.**

Việc kết nối từ ứng dụng đến thiết bị được thực hiện hoàn chỉnh thông qua 4 thực thể chính liên quan gồm:

- Application *(ứng dụng)*.
- Character device file *(file thiết bị, được tạo tự động bởi udev)*.
- Character device driver *(driver thiết bị)*.
- Character device *(thiết bị)*.



Hình 2. Giao tiếp với Character Divice

2. Tương tác với Kernel Module.

2.1. Các thư mục liên quan.

Thư mục	Vai trò
/dev	Thư mục hệ thống chứa một số tệp đặc biệt đại diện cho thiết bị gắn kèm vào hệ thống , cũng chứa các thiết bị giả, là các thiết bị ảo không thực sự tương ứng với phần cứng.
/home	Chứa một thư mục trang chủ cho mỗi người dùng.
/proc	Thông tin về các tiến trình lưu dưới dạng một hệ thống file thư mục mô phỏng, các tài nguyên đang sử dụng của hệ thống. Xem thông tin về major number và tên module tại /proc/devices.
/sys	Thư mục cho Virtual File System *(có trong các bản distro Linux hiện đại)*. Cho phép lưu trữ và sửa đổi các lớp thiết bị (device class) được kết nối với hệ thống.
/var	Lưu file về biến của chương trình, hệ thống. Ta sử dụng thư mục này để xem log file tại /var/log.
/kernel	Chứa mã nguồn triển khai nhiệm vụ lập lịch và đồng bộ hoạt động của các tiến trình.

2.2. Các lệnh biên dịch.

Cách lệnh sau cần được thực thi dưới quyền root.

- `lsmod`: liệt kê các modules đã được nạp.
- `insmod <module_file>`: nạp `<module_file>` vào hệ thống (yêu cầu modules phụ thuộc nếu có phải được nạp trước).
- `modprobe <module>`: nạp `<module>` vào hệ thống (tự động tìm các module phụ thuộc nếu có để nạp).
- `rmmod <module>`: gỡ `<module>` khỏi hệ thống.
- `modinfo`: xem thông tin của module.
- `dmesg`: để xem thông tin ghi log từ kernel.
- `make`: tạo kernel module với đối số dòng lệnh và hành động liên kết tương ứng được chỉ định trong file `Makefile`.

2.3. Các kiểu dữ liệu.

- `dev_t`: chứa cả số hiệu major và minor (số hiệu device file).
- `cdev`: struct lưu trữ thông tin của character device driver.
- `class`: struct đại diện cho device class.
- `file_operations`: lưu trữ các thao tác xử lý với file thiết bị đang viết driver (vd: `my_open, my_close, my_read, my_write`, v.v).

2.4. Các hàm phát triển.

Các hàm cần được `#include` thư viện tương ứng trước khi sử dụng.

- `printk()`: ghi log ra bộ đệm ở kernel (dùng lệnh `dmesg` để xem).
- `register_chrdev_region(dev_t, unsigned int, char*)`: định danh tĩnh major number.
  Truyền vào: biến lưu trữ định danh, số lượng định danh, tên driver.
- `alloc_chrdev_region(dev_t*, unsigned int, unsigned int, char*)`: cấp động major number một cách ngẫu nhiên bởi kernel.
  Truyền vào: con trỏ lưu trữ định danh, số minor đầu tiên, số lượng định danh, tên driver.
- `unregister_chrdev_region(dev_t, unsigned int)` : giải phóng major number đã đăng ký.
  Truyền vào: biến lưu trữ định danh, số lượng định danh.
- `class_create(struct module*, char*):` tạo ra class device.

Truyền vào: `THIS_MODULE`, tên lớp thiết bị.

- `device_create(struct class*, NULL, dev_t, NULL, char*)`: tạo số định danh thiết bị.
  Truyền vào: lớp thiết bị, biến định danh, định dạng thiết bị.
- `device_destroy(struct class*, dev_t)`: huỷ số định danh cho class device.
- `class_destroy(struct class*)`: huỷ class device.
- `cdev_init(cdev*, file_operations)`: dành ra một vùng nhớ riêng lưu trữ `cdev` mới vừa tạo ra.
  Truyền vào: con trỏ lưu thông tin driver đã được khai báo, cấu trúc tập lệnh của driver.
- `cdev_add(cdev*, dev_t, int)`: cài đặt cấu trúc driver này vào kernel.
  Truyền vào: con trỏ cấu trúc driver cần cài đặt, số định danh thiết bị , số thiết bị muốn cài đặt vào kernel.
- `cdev_del(cdev)`: giải phóng driver.
- `copy_to_user(*to,*from, size)`: đọc dữ liệu từ bộ đệm giữa tầng ứng dụng và driver một cách an toàn.

2.5. Các marco sử dụng.

- `module_init(initfn)` : xác định `initfn` là hàm khởi tạo module.
- `module_exit(exitfn)` : xác định `exitfn` là hàm kết thúc module.
- `__init` : cho biết hàm chỉ thực thi lúc khởi tạo (đặt trước `initfn` để tiết kiệm bộ nhớ).
- `__exit` : cho biết hàm thực thi khi chuẩn bị tháo module khỏi kernel (đặt trước `exitfn` để tiết kiệm bộ nhớ).
- `MODULE_AUTHOR(_author <email>)` : thông tin người tạo ra module.
- `MODULE_DESCRIPTION(_description)` : thông tin chức năng của module.
- `MODULE_SUPPORTED_DEVICE(name)` : thông tin các thiết bị hỗ trợ module.
- `MODULE_LICENSE(_license)` : thông tin loại giấy phép sử dụng module.
- `KERN_ALERT` : phân loại thông điệp cảnh báo lập tức trong log file.
- `KERN_INFO` : phân loại thông điệp thông tin trong log file.
- `MAJOR(dev_t)` : lấy số hiệu major từ tham số truyền vào `dev_t`.
- `MINOR(dev_t)` : lấy số hiệu minor từ tham số truyền vào `dev_t`.
- `MKDEV(int, int)` : tạo ra dữ liệu `dev_t` từ cặp số hiệu major và minor truyền vào.

BÁO CÁO KỸ THUẬT

1. Random Number Generator.

Bao gồm file Makefile và Kbuild để biên dịch kernel module. File randomModule.c chứa mã nguồn tạo module phát sinh số ngẫu nhiên.

1.1. Tạo loadable kernel module.

Ta khai báo các thông tin cho module thông qua các marco `MODULE_LICENSE`, `MODULE_AUTHOR`, `MODULE_DESCRIPTION`.

Module được lập trình theo kiểu hướng đối tượng nên cần có hàm khởi tạo `random_init` đi cùng marco `__init` và hàm huỷ `random_exit` đi cùng marco `__exit`. Hàm khởi tạo là tham số truyền vào `module_init` để lắp module vào kernel. Tương tự, hàm huỷ là tham số truyền vào `module_exit` để tháo module ra khỏi kernel.

Sử dụng hàm `get_random_bytes` trong thư viện Linux để tạo số ngẫu nhiên trong giới hạn là `MAX`.

1.2. Tạo character device.

Đầu tiên, ta cần định danh các số hiệu major và minor cho character device. Đồ án này chúng em sử dụng định danh động bằng hàm `alloc_chrdev_region` và lưu trữ thông tin đó trong biến `static dev_t first`.

Để tạo file thiết bị một cách tự động mà không cần dùng `mknod()`, ta cần tạo ra class device bằng hàm `class_create` và lưu nó vào `static struct class *cl`. Sau đó lưu thông tin định danh major và minor đã đăng ký trước đó vào hàm `device_create`.

Để thao tác với file thiết bị, ta cần khai báo file operations với các thao tác `device_open`, `device_close`, `device_read` và lưu chúng trong biến `static struct file_operations fops`. Sau đó, ta điều khiển cấu trúc này đến hệ thống file ảo VFS bằng cách gọi hàm `cdev_init` và `cdev_add`.

Trước khi tháo module ra khỏi kernel, hàm huỷ được nạp vào và thực thi. Hàm huỷ gọi đến các hàm dọn dẹp cho các đối tượng tương ứng: `cdev_del`, `device_destroy`, `class_destroy` và `unregister_chrdev_region`.

2. Chương trình ở user space.

Bao gồm file userTest.c chứa mã nguồn chương trình gọi từ user space xuống kernel để chạy thử.

Chương trình thực hiện mở file `/dev/RANDOM_NUMBER` ở dạng readonly. Sau đó đọc số ngẫu nhiên trong file device drive được gửi lên từ kernel đến user space rồi in ra màn hình.

HƯỚNG DẪN SỬ DỤNG

1. Thực thi module ở kernel space.

Mở Terminal rồi gõ lệnh cd tới folder RNG_LKM chứa mã nguồn. Hoặc mở folder, click chuột phải chọn 'Open in Terminal' rồi lần lượt thực thi các lệnh sau.

1. Build kernel module theo phương pháp Kbuild bằng lệnh:

```
make
```

2. Cài đặt module vào kernel.

```
sudo insmod randomModule.ko
```

3. Đọc dữ liệu số ngẫu nhiên được tạo từ file device ra Terminal.

```
sudo cat /dev/RANDOM_NUMBER
```

4. Tháo module ra khỏi kernel.

```
sudo rmmod randomModule
```

5. Dọn dẹp file build dư thừa trong folder.

```
make clean
```

Một số lệnh kiểm tra và xem log trong quá trình thực thi:

Sau bước 1, có thể xem thông tin của module.

```
modinfo randomModule.ko
```

Sau bước 2, có thể kiểm tra module đã cài đặt thành công hay chưa bằng cách:

```
cd /dev
ls
```

nếu thấy tồn tại file `RANDOM_NUMBER` tức là cài đặt thành công.

Trước khi thực hiện bước 3, mở một cửa sổ Terminal mới để xem log file (Ctrl+C để dừng việc xem log).

```
tail -f /var/log/kern.log
```

2. Thực thi chương trình ở user space.

Thực hiện từ bước 1 đến bước 2 như phần trên.

Thay thế bước 3 bằng câu lệnh:

```
sudo ./userTest
```

Bước 4 và 5 thực thi như phần trên.

TÀI LIỆU THAM KHẢO

Hướng dẫn lập trình device driver cơ bản

LINUX DEVICE DRIVER

Lập trình Device Driver trên Linux

Linux Kernel HTML Documentation

Viết một driver đơn giản theo cơ chế kernel module