

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP HCM

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN 3 HỆ ĐIỀU HÀNH TÌM HIỂU VÀ LẬP TRÌNH LINUX KERNEL MODULE

Giảng viên:

- *Phạm Tuấn Sơn*
- *Lê Giang Thanh*

Nhóm sinh viên thực hiện:

1. *Phan Gia Hân - 18120026*
2. *Đặng Văn Hiển - 18120363*
3. *Lê Thanh Viễn - 18120647*

1. Thông tin nhóm:

1.1: Thông tin thành viên:

MSSV	Họ Tên	Email
18120026	Phan Gia Hân	18120026@student.hcmus.edu.vn
18120363	Đặng Văn Hiến	18120363@student.hcmus.edu.vn
18120647	Lê Thanh Viễn	18120647@student.hcmus.edu.vn

1.2: Mức độ hoàn thành công việc:

Người thực hiện	Công việc	Mức độ hoàn thành
Phan Gia Hân	Viết hàm khởi tạo và hủy device	100%
Đặng Văn Hiến	Tổng hợp các file và viết báo cáo	100%
Lê Thanh Viễn	Viết hàm thao tác với device (đọc,...) , hàm tạo số ngẫu nhiên	100%

2. Tìm hiểu linux kernel :

a. Sơ lược Linux:

Năm 1991, dựa trên UNIX kernel, Linus Torvalds đã tạo ra Linux kernel chạy trên máy tính của ông ấy. Dựa vào chức năng của hệ điều hành, Linux kernel được chia làm 6 thành phần:

- **Process management:** có nhiệm vụ quản lý các tiến trình:
 - o Tạo/hủy các tiến trình.
 - o Lập lịch cho các tiến trình.
 - o Hỗ trợ các tiến trình giao tiếp với nhau.
 - o Đồng bộ hoạt động của các tiến trình tránh tranh chấp tài nguyên.
- **Memory management:** có nhiệm vụ quản lý bộ nhớ, bao gồm các công việc:
 - o Cấp phát/ thu hồi bộ nhớ chương trình .
 - o Đảm bảo chương trình nào cũng có cơ hội được đưa vào bộ nhớ.
 - o Bảo vệ vùng nhớ của mỗi tiến trình.
- **Device management:** có nhiệm vụ quản lý thiết bị, bao gồm các công việc:
 - o Điều khiển hoạt động của các thiết bị.
 - o Giám sát trạng thái của các thiết bị.
 - o Trao đổi dữ liệu với các thiết bị.
 - o Lập lịch sử dụng các thiết bị, đặc biệt là thiết bị lưu trữ (ví dụ ổ cứng).
- **File system management:** quản lý dữ liệu trên thiết bị lưu trữ (như ổ cứng, thẻ nhớ). Quản lý dữ liệu gồm các công việc: thêm, tìm kiếm, sửa, xóa dữ liệu.
- **Networking management:** có nhiệm vụ quản lý các gói tin (packet) theo mô hình TCP/IP.
- **System call Interface:** có nhiệm vụ cung cấp các dịch vụ sử dụng phần cứng cho các tiến trình. Mỗi dịch vụ được gọi là một system call.

Khi triển khai thực tế, mã nguồn của Linux kernel gồm các thư mục sau:

Thư mục	Vai trò
/arch	Chứa mã nguồn giúp Linux kernel có thể thực thi được trên nhiều kiến trúc CPU khác nhau như x86, alpha, arm, mips, mk68, powerpc, sparc,...
/block	Chứa mã nguồn triển khai nhiệm vụ lập lịch cho các thiết bị lưu trữ.
/drivers	Chứa mã nguồn để triển khai nhiệm vụ điều khiển, giám sát, trao đổi dữ liệu với các thiết bị.
/fs	Chứa mã nguồn triển khai nhiệm vụ quản lý dữ liệu trên các thiết bị lưu trữ.
/ipc	Chứa mã nguồn triển khai nhiệm vụ giao tiếp giữa các tiến trình
/kernel	Chứa mã nguồn triển khai nhiệm vụ lập lịch và đồng bộ hoạt động của các tiến trình.
/mm	Chứa mã nguồn triển khai nhiệm vụ quản lý bộ nhớ
/net	Chứa mã nguồn triển khai nhiệm vụ xử lý các gói tin theo mô hình TCP/IP.

b. Sơ lược về module Linux:

Linux kernel module là một file với tên mở rộng là (.ko). Nó sẽ được lắp vào hoặc tháo ra khỏi kernel khi cần thiết. Chính vì vậy, nó còn có một tên gọi khác là loadable kernel module. Một trong những kiểu loadable kernel module phổ biến đó là driver. Việc thiết kế driver theo kiểu loadable module mang lại 3 lợi ích:

- Giúp giảm kích thước kernel. Do đó, giảm sự lãng phí bộ nhớ và giảm thời gian khởi động hệ thống.
- Không phải biên dịch lại kernel khi thêm mới driver hoặc khi thay đổi driver.
- Không cần phải khởi động lại hệ thống khi thêm mới driver. Trong khi đối với Windows, mỗi khi cài thêm driver, ta phải khởi động lại hệ thống, điều này không thích hợp với các máy server.

Phần lớn các driver đều là các loadable kernel module, nhưng không phải là tất cả. Vẫn có một số driver được tích hợp luôn vào trong kernel, đặc biệt là các bus driver.

Chúng được gọi là built-in driver. Các device driver thường sẽ là các loadable kernel module.

Ngược lại, không phải loadable kernel module nào cũng là driver, ví dụ kvm.ko là loadable kernel module nhưng không phải là driver. Trên thực tế, loadable kernel module được chia làm 3 loại chính: device driver, system call và file system.

4. Chi tiết mã nguồn trong linux kernel module:

a. Make file :

```
KDIR = /lib/modules/`uname -r`/build

all:
    make -C $(KDIR) M=`pwd`
clean:
    make -C $(KDIR) M=`pwd` clean
```

b. Kbuild:

```
1 EXTRA_CFLAGS = -Wall
2 obj-m = mymodule.o
3
```

c. Mã nguồn module:

-Include các thư viện linux kernel

```
#include <linux/module.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/types.h>
#include <linux/kdev_t.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/cdev.h>
#include <linux/random.h>
```

-Thêm thông tin cho module

```
// Thông tin module
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Hien Vien Han");
MODULE_DESCRIPTION("Module tạo số ngẫu nhiên");
MODULE_SUPPORTED_DEVICE("character device");
```

• Hàm khởi tạo và đăng kí character device driver:

- Ta thực hiện cấp phát động với hàm `alloc_chrdev_region`. Nhiệm vụ của hàm này là tìm ra một giá trị có thể dùng làm device number.

- Major number cho biết device driver.
- Minor number dùng để biểu diễn cho các chức năng con drive.

Để lưu giá trị các device number trong mảng cdev.

- Tạo lớp characterdevice bằng hàm `class_create`.
- Tạo các device file bằng các hàm `cdev_init` – tạo cdev thích hợp , `cdev_add` -đăng kí cdev, `device_create` – tạo device file.

• Hàm gỡ character device driver:

* Ngược với hàm tạo ta tạo hàm hủy cho module.

- Hủy các device file bằng hàm `device_destroy`.
- Hủy lớp device bằng hàm `class_destroy`.
- Hủy device major đã đăng kí bằng hàm `unregister_chrdev_region`.

• Các hàm thao tác với character device file:

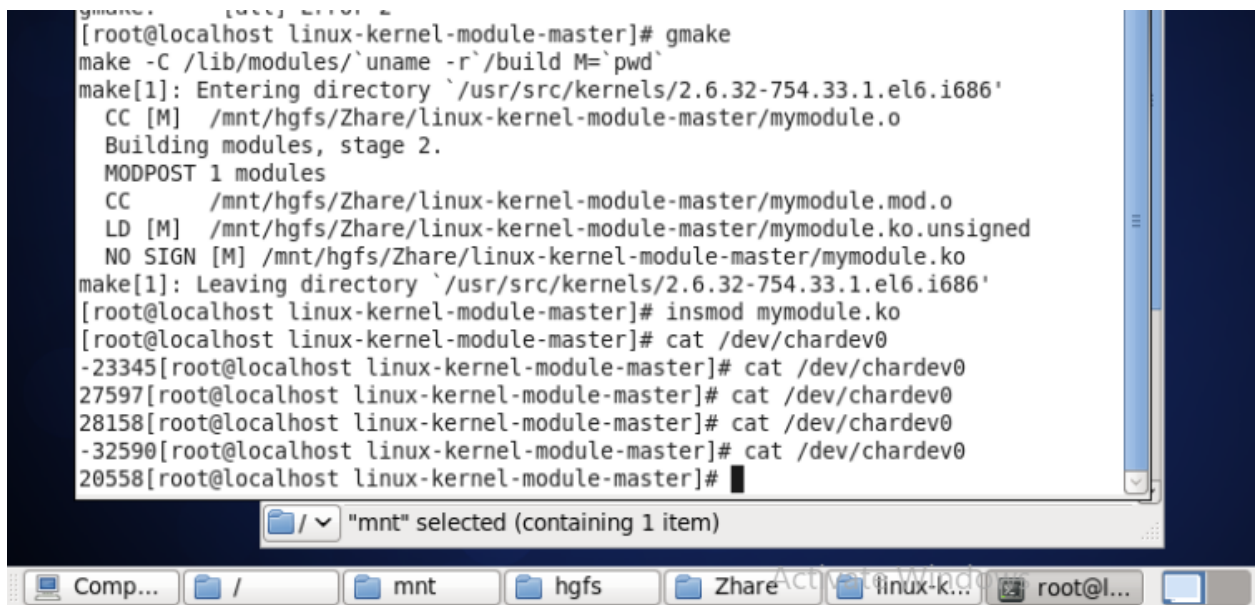
- Tạo đối tượng struct `file_operations` để thao tác với device file:

```
// Struct thao tác với device file
static const struct file_operations pugs_fops = {
    .owner      = THIS_MODULE,
    .open       = myOpen,
    .release    = myRelease,
    .read       = myRead
};
```

- * Viết các hàm tương ứng : open, release, ..
- * Với hàm đọc ta chú thao tác gián tiếp bằng cách copy dữ liệu giữa các vùng nhớ bằng hàm `copy_to_user` , cập nhập offset phù hợp với mỗi lượt đọc xong.
- * Hàm tạo số ngẫu nhiên được cung cấp trong thư viện `linux/random.h`

5. Chạy thử:

- Build mã nguồn \$ `gmake`
- Cài đặt module \$ `insmod mymodule.ko`
- Đọc device file \$ `cat /dev/chardev0`



```
[root@localhost linux-kernel-module-master]# gmake
make -C /lib/modules/`uname -r`/build M=`pwd`
make[1]: Entering directory `/usr/src/kernels/2.6.32-754.33.1.el6.i686'
CC [M] /mnt/hgfs/Zhare/linux-kernel-module-master/mymodule.o
Building modules, stage 2.
MODPOST 1 modules
CC      /mnt/hgfs/Zhare/linux-kernel-module-master/mymodule.mod.o
LD [M] /mnt/hgfs/Zhare/linux-kernel-module-master/mymodule.ko.unsigned
NO SIGN [M] /mnt/hgfs/Zhare/linux-kernel-module-master/mymodule.ko
make[1]: Leaving directory `/usr/src/kernels/2.6.32-754.33.1.el6.i686'
[root@localhost linux-kernel-module-master]# insmod mymodule.ko
[root@localhost linux-kernel-module-master]# cat /dev/chardev0
-23345[root@localhost linux-kernel-module-master]# cat /dev/chardev0
27597[root@localhost linux-kernel-module-master]# cat /dev/chardev0
28158[root@localhost linux-kernel-module-master]# cat /dev/chardev0
-32590[root@localhost linux-kernel-module-master]# cat /dev/chardev0
20558[root@localhost linux-kernel-module-master]#
```

*Tham khảo file và video hướng dẫn của thầy