TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP HCM

KHOA CÔNG NGHỆ THÔNG TIN

**MÔN: HỆ ĐIỀU HÀNH**

BÁO CÁO ĐỒ ÁN MÔN HỌC

ĐỒ ÁN 2

-Linux kernel module -

Lớp: Hệ điều hành – CQ2017/32

Sinh viên thực hiện:

Hoàng Quốc Thịnh – 1712790

Lê Phúc Thịnh – 1712792

## Random number generator

### File number.c

File này là chương trình Linux Kernel Module dùng để sinh ra một số ngẫu nhiên, cho phép các tiến trình ở user space có thể open và read các số ngẫu nhiên.

## **static int\_\_init \_init\_numGenerator(void)**

Nhiệm vụ: Hàm này dùng để khởi tạo khi xây dựng một driver. Hàm được gọi khi module được load lên.  
Quy trình:

* Đầu tiên, đăng kí với hệ thống để hệ thống cấp cho một majorNumber. Nếu thành công (majorNumber < 0) thì đăng kí thất bại.
* Sau khi đăng kí thành công, tiến hành khởi tạo một device class với câu lện generClass = class\_create(THIS\_MODULE, CLASS\_NAME); Hàm sẽ trả về một con trỏ hàm đến class device được khởi tạo nếu thành công. Ngược lại chúng ta phải hủy bỏ đi số majorNumber đã đăng kí. Sau đó thông báo ra màn hình: “Failed to register device class”.
* Cuối cùng, đăng kí một driver qua lệnh: generDev = device\_create(generClass, NULL, MKDEV(majorNumber, 0), NULL, DEVICE\_NAME); Hàm này sẽ trả về con trỏ kiểu device-driver device struct được định nghĩa ở <linux/device.h> nếu thành công. Ngược lại chúng ta sẽ hủy bỏ generClass và majorNumber vừa mới đăng kí.

## **static void \_\_exit\_exit\_numGenerator (void)**

Nhiệm vụ: Hàm này sẽ dọn những gì được làm trước khi gỡ module. Hàm được gọi trong quá trình gỡ module.

Quy trình:

* Hàm lần lượt hủy đi những con trỏ được cấp phát và hủy đăng kí majorNumber.

## **static int device\_Open(struct inode\* inodep, struct file\* filep)**

Nhiệm vụ: Hàm sẽ được gọi khi nhận yêu cầu mở từ user space.

Quy trình:

* Hàm sẽ có một biến toàn cục count để đếm số lần module này được yêu cầu mở và xuất ra thông tin này khi nhận được yêu cầu mở.

## **static ssize\_t device\_Read(struct file\* filep, char\* buffer,**

## **size\_t len, loff\_t\* offset)**

Nhiệm vụ: Hàm này được gọi khi user space muốn truy cập dữ liệu từ device.

Quy trình:

* Ta dùng lệnh **get\_random\_bytes** để random một số ngẫu nhiên.
* Sau đó, ta gửi dữ liệu này đến user space thông qua lệnh: copy\_to\_user(buffer, numberGen, sizeof(int)); Nếu thành công, hàm copy\_to\_user sẽ trả về 0.

## **static int device\_Close(struct inode\* inodep, struct file\* filep)**

Nhiệm vụ: Hàm này được gọi khi đóng device.

Quy trình:

* Hàm này xuất thông báo ra màn hình thiết bị đã đóng bằng printk.

### File generator.c

Đây là chương trình thuộc user space. Chương trình này sẽ gọi lệnh **open("/dev/numGenerator", O\_RDWR)** để yêu cầu mở device được tạo từ number.c trên để tạo ra 1 số ngẫu nhiên.

Khi mở thành công, tiến hành đọc số ngẫu nhiên mà device gửi đến.

Khi đọc thành công, xuất ra số ngẫu nhiên đó.

## Random number generator

### Tạo hook

Để hook vào một syscall sẵn có. Ta cần phải tìm địa chỉ syscall table. Ta có lệnh:

cat/boot/System.map-XXX |grepsys\_call\_table

XXX là phiên bản kernel đang xử dụng.

Sau đó ta sẽ copy địa chỉ syscall table vào code hook bên dưới:

***#include* <asm/unistd.h>**

***#include* <asm/cacheflush.h>**

***#include* <asm/pgtable\_types.h>**

***#include* <linux/init.h>**

***#include* <linux/module.h>**

***#include* <linux/kernel.h>**

***#include* <linux/syscalls.h>**

***#include* <linux/highmem.h>**

***#include* <linux/fs.h>**

***#include* <linux/sched.h>**

***#include* <linux/moduleparam.h>**

***#include* <linux/unistd.h>**

**MODULE\_LICENSE("GPL");**

***/\*MY sys\_call\_table address\*/***

***//ffffffff82000260***

**void \*\*system\_call\_table\_addr;**

**asmlinkage long (\*temp\_open) (const char\*, int, umode\_t);**

**asmlinkage long (\*temp\_write) (unsigned int, const char\*, size\_t);**

**asmlinkage long hook\_open(const char\* filename, int flags, umode\_t mode)**

**{**

**char buff[100];**

**copy\_from\_user(buff, filename, 100);**

**printk(KERN\_INFO "Process name opens file: %s", current->comm);**

**printk(KERN\_INFO "Hooked open: filename = %s\n", buff);**

***return* temp\_open(filename, flags, mode);**

**}**

**asmlinkage long hook\_write(unsigned int fd, const char\* buf, size\_t len)**

**{**

**printk(KERN\_INFO "Process name writes file: %s", current->comm);**

**printk(KERN\_INFO "Hooked write: fd = %u, len = %d\n", fd, (int)len);**

***return* temp\_write(fd, buf, len);**

**}**

**int make\_rw(unsigned long address)**

**{**

**unsigned int level;**

**pte\_t \*pte = lookup\_address(address, &level);**

***if*(pte->pte &~\_PAGE\_RW)**

**{**

**pte->pte |=\_PAGE\_RW;**

**}**

***return* 0;**

**}**

**int make\_ro(unsigned long address)**

**{**

**unsigned int level;**

**pte\_t \*pte = lookup\_address(address, &level);**

**pte->pte = pte->pte & ~\_PAGE\_RW;**

***return* 0;**

**}**

**static int \_\_init entry\_point(void)**

**{**

**printk(KERN\_INFO "Hook loaded successfully..\n");**

***/\*MY sys\_call\_table address\*/***

**system\_call\_table\_addr = (void\*)0xffffffff82000260;**

***/\* Replace custom syscall with the correct system call name (write,open,etc) to hook\*/***

**temp\_open = system\_call\_table\_addr[\_\_NR\_open];**

**temp\_write = system\_call\_table\_addr[\_\_NR\_write];**

***/\*Disable page protection\*/***

**make\_rw((unsigned long)system\_call\_table\_addr);**

***/\*Change syscall to our syscall function\*/***

**system\_call\_table\_addr[\_\_NR\_open] = hook\_open;**

**system\_call\_table\_addr[\_\_NR\_write] = hook\_write;**

***/\*Renable page protection\*/***

**make\_ro((unsigned long)system\_call\_table\_addr);**

***return* 0;**

**}**

**static void \_\_exit exit\_point(void)**

**{**

**printk(KERN\_INFO "Unloaded Hook successfully\n");**

***/\*Disable page protection\*/***

**make\_rw((unsigned long)system\_call\_table\_addr);**

***/\*Restore original system call \*/***

**system\_call\_table\_addr[\_\_NR\_open] = temp\_open;**

**system\_call\_table\_addr[\_\_NR\_write] = temp\_write;**

***/\*Renable page protection\*/***

**make\_ro((unsigned long)system\_call\_table\_addr);**

**}**

**module\_init(entry\_point);**

**module\_exit(exit\_point);**

Giải thích:

* + - 2 hàm quan trọng ở đây là entry\_point và exit\_point. Đây là 2 hàm được gọi khi hook đươc kích hoạt khi chạy lệnh insmod và rmmod.
    - Hàm entry\_point:
      * Dòng 57, có biến system\_call\_table\_addr, cần phải chạy lệnh để lấy địa chỉ của systemcall table để dán vào đây.
      * Dòng 59, 60: temp\_open và temp\_write là biến để lưu lại địa chỉ của các syscall open và write của hệ thống.
      * Dòng 62: Gọi hàm make\_rw để xóa lớp bảo vệ của syscall.
      * Sau đó ta tiến hành thay syscall của hệ thống bằng syscall của mình bằng 2 dòng lệnh ở 64, 65.
      * Dòng 67: Mở lại lớp bảo vệ của syscall.
    - Hàm exit\_point:
      * Dòng 74: Gọi hàm make\_rw để xóa lớp bảo vệ của syscall.
      * Dòng 76, 77: Trả syscall cho hệ thống.
      * Dòng 79: Mở lại lớp bảo vệ của syscall.