TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM

KHOA CÔNG NGHỆ THÔNG TIN

MÔN: **HỆ ĐIỀU HÀNH**

BÁO CÁO

Tìm hiểu về Linux Kernel Module

GVHD: **Dũng Trần Trung**

Thực hiện: **Huỳnh Kim Ninh**

1612484

TP.HCM, 29/09/2018 12:39

Mục lục

1. Các chức năng của module 3
1.1 Phát sinh một số ngẫu nhiên 3
1.2 Tạo một Character Device 4
1.3 Đọc số ngẫu nhiên thông qua Character Device 5
1.4 Test thử chương trình từ user-space 6
2. Tài liệu tham khảo 7

1. Các chức năng của module

1.1 Phát sinh một số ngẫu nhiên

Để phát sinh một số ngẫu nhiên từ **kernel-space**, ta không thể dùng hàm đã được viết từ thư viện **user-space** như: rand(), srand(),... Thay vào đó, ta phải dùng các hàm được hỗ trợ từ kernel-space.

Ở đây, ta sẽ dùng hàm get_random_bytes() đã được định nghĩa ở <linux/random.h>

```
get_random_bytes(&randomNumber, sizeof(randomNumber));
```

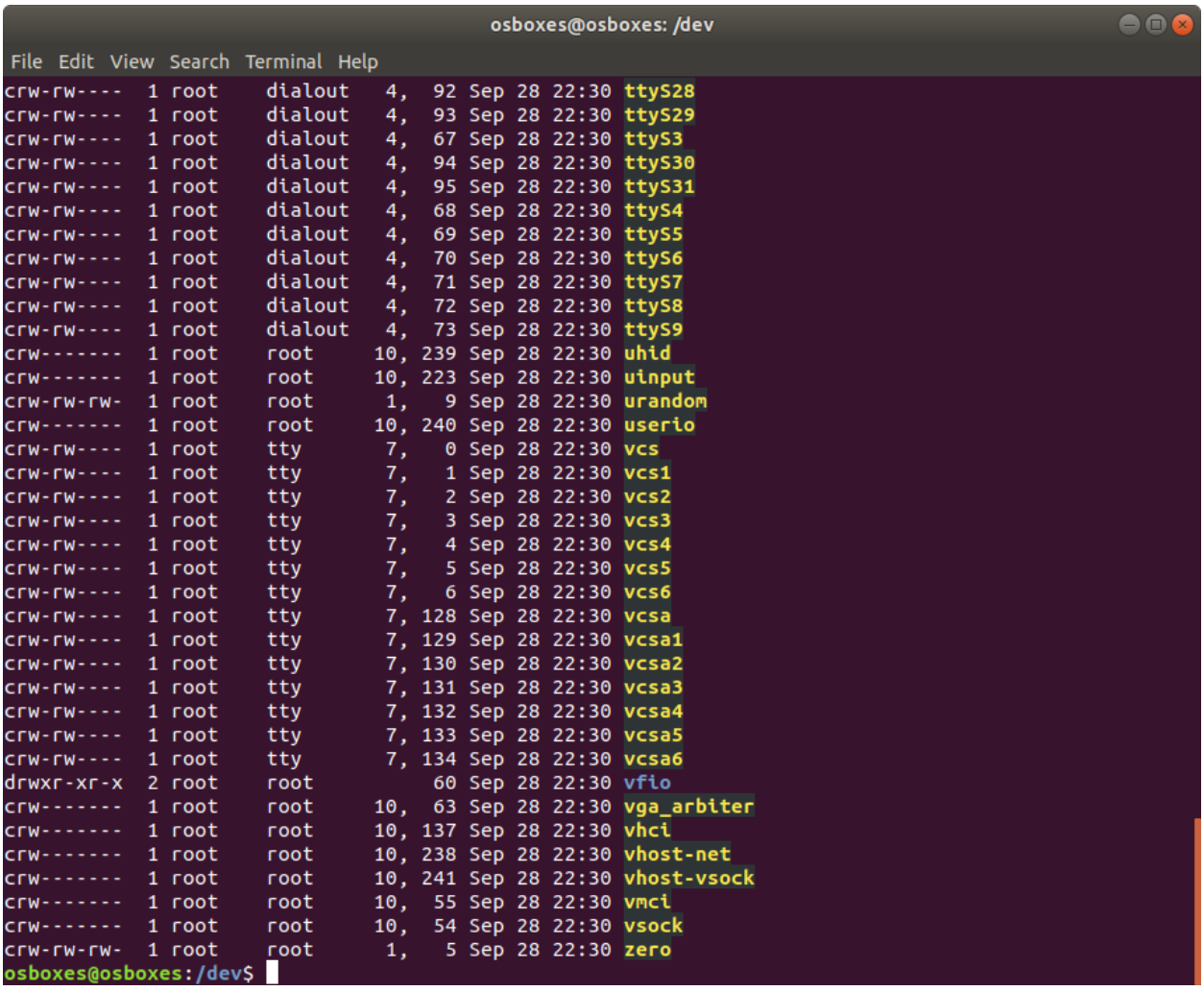
randomNumber chính là biến toàn cục. Lưu ý, nếu ta khai báo randomNumber là biến kiểu int thì số ngẫu nhiên nhận được có thể là số âm với cách truyền tham số như trên. Vì thế cho nên nếu muốn số nguyên được tạo ra chỉ toàn số nguyên dương thì phải khai báo kiểu usigned int.

Ta sẽ gọi hàm get_random_bytes() ở trong hàm random_init(void), đây là 1 hàm đặc biệt được hiểu tương tự như hàm main trong chương trình ở user-space. Nó là hàm đầu tiên được gọi ở trong module khi cài đặt. Ngoài ra, ta còn phải khai báo thêm 1 hàm khi module được "gỡ bỏ", có tên là random_exit(void)

```
#include <linux/random.h>       // get_random_bytes()

//...

static int randomNumber = 0;
static int __init random_init(void){
    printk(KERN_INFO "GRNChar: Initializing the GRNChar LKM\n");

    //Phat sinh 1 so nguyen 4 byte
    get_random_bytes(&randomNumber, sizeof(randomNumber));
    printk(KERN_INFO "GRNChar: The random number is: %d", randomNumber);
    // ...
}

static void __exit random_exit(void){
    // ...
    printk(KERN_INFO "GRNChar: Goodbye from the LKM\n");
}

module_init(random_init);
module_exit(random_exit);
```

1.2 Tạo một Character Device

Character Device có thể được hiểu là nó dùng để chuyển dữ liệu (lấy dữ liệu, ghi dữ liệu) từ ứng dụng ở user-space. Ngoài ra còn có thêm Block Device, chức năng cũng tương đồng với Character Device. Cả hai loại hiện diện như là 1 file có thể truy cập thông qua đường dẫn "/dev/"



Mỗi một Device đều phải có 2 con số Major Number và Minor Number được dùng để định danh.

Ta hoàn ta có thể tạo một Character Device từ user-space (thông qua terminal) hoặc từ kernel-space (thông qua module được viết).

Các Device đều được thể hiện là cấu trúc file ở trong kernel. Cấu trúc kiểu file_operations được định nghĩa tại `/linux/fs.h` giữ các con trỏ đến các hàm thao tác với file. Chúng ta chỉ cần cài đặt các hàm mà ta sẽ cần sử dụng và trỏ các con trỏ đến hàm cài đặt tương ứng (trong phạm vi đồ án chính là 3 hàm open(), read(), realese()).

1.3 Đọc số ngẫu nhiên thông qua Character Device

Sau khi đã khai báo cấu trúc file_operations và gán các hàm dev_open(), dev_read(), dev_release(). Ta tiến hành cài đặt cụ thể. Ta sẽ dùng hàm copy_to_user() để sao chép nội dung vùng nhớ của số ngẫu nhiên từ kernel-space sang 1 biến ở user-space để lưu kết quả đó.

```
 static ssize_t dev_read(struct file *filep, char *buffer, size_t len, loff_t
*offset){

    int error_count = 0;

    error_count = copy_to_user(buffer,(char*) &randomNumber,
sizeof(randomNumber));

    if (error_count == 0){
        printk(KERN_INFO "GRNChar: Sent number %d to the user\n", randomNumber);
        return 0;
    }

    printk(KERN_INFO "GRNChar: Failed to sent number %d to the user\n",
randomNumber);
    return -EFAULT;
}
```

1.4 Test thử chương trình từ user-space

Bây giờ sau khi đã chuẩn bị hết mọi thứ ở kernel-space rồi, ta sẽ bắt đầu sử dụng ở user-space.

Gọi hàm mở Character Device đã tạo trong module và bật cờ chỉ cho phép đọc:

```
    int ret, fd;

    fd = open("/dev/grnchar", O_RDONLY);
```

Sau đó tiến hành "lấy" con số ngẫu nhiên từ kernel-space thông qua "grnchar"

```
    int number;
    ret = read(fd, (char*)&number, sizeof(number));
```

Sau khi build module và file source code dùng để test. Ta tiến hành cài module và chạy chương trình test.

```
root@osboxes:/home/osboxes/Documents/New_Folder# insmod generateRandom.ko
root@osboxes:/home/osboxes/Documents/New_Folder# ./test
The received number is 90396285
root@osboxes:/home/osboxes/Documents/New_Folder#
```

Chạy chương trình từ test từ user-space

```
Sep 29 01:12:59 osboxes kernel: [ 9764.763170] GRNChar: Initializing the GRNChar LKM
Sep 29 01:12:59 osboxes kernel: [ 9764.763177] GRNChar: The random number is: 90396285
Sep 29 01:12:59 osboxes kernel: [ 9764.764322] GRNChar: registered correctly with major number 243
Sep 29 01:12:59 osboxes kernel: [ 9764.779362] GRNChar: device class registered correctly
Sep 29 01:12:59 osboxes kernel: [ 9764.814656] GRNChar: device class created correctly
Sep 29 01:13:46 osboxes kernel: [ 9811.249426] GRNChar: Device has been opened
Sep 29 01:13:46 osboxes kernel: [ 9811.249430] GRNChar: Sent number 90396285 to the user
Sep 29 01:13:46 osboxes kernel: [ 9811.249504] GRNChar: Device successfully closed
```

Kiểm tra số ngẫu nhiên từ kernel log

2. Tài liệu tham khảo

- Writing a Linux Kernel Module – Introduction
- Writing a Linux Kernel Module – A Character Device
- Learn Enough Command Line to Be Dangerous