

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



**BÁO CÁO THỰC HÀNH ĐỒ ÁN 2 – HỆ ĐIỀU HÀNH**  
**LẬP TRÌNH LINUX KERNEL MODULE**

<b>MSSV</b>	<b>Họ tên</b>
18120287	Phan Xuân Bảo
18120631	Lê Nguyên Tuấn
18120658	Phạm Viết Xuân

Tp. Hồ Chí Minh, tháng 12 năm 2020

## MỤC LỤC

MỤC LỤC .....	2
Phân công đồ án.....	2
PHẦN 1: NỘI DUNG YÊU CẦU ĐỒ ÁN .....	3
PHẦN 2: TÌM HIỂU VỀ LINUX KERNEL MODULE .....	3
1. Một định nghĩa về các thành phần trong linux kernel .....	3
1.1. Sơ lược về linux kernel.....	3
1.2. Các lệnh trong linux kernel .....	5
2. Linux kernel module .....	6
PHẦN 3: TỔ CHỨC MODULE VÀ GIẢI THÍCH SOURCE CODE.....	6
1. Đánh giá mức độ hoàn thành .....	6
2. Tổ chức file .....	6
3. Tạo và gắn module phát sinh số ngẫu nhiên.....	7
4. Tạo character device phát sinh số ngẫu nhiên .....	7
5. Một số thông tin về module đã tạo .....	10
6. Makefile và cách chạy chương trình.....	10
6.1. Makefile.....	10
6.2. Tạo, gắn module vào kernel và chạy chương trình người dùng.....	11

## Phân công đồ án

MSSV	Họ tên	Phân chia	Mức độ hoàn thành
18120287	Phan Xuân Bảo	Report, Readme.md	100%
18120631	Lê Nguyên Tuấn	User test, random number	100%
18120658	Phạm Viết Xuân	Character Device	100%

## PHẦN 1: NỘI DUNG YÊU CẦU ĐỒ ÁN

Mục tiêu hiểu về Linux kernel module và hệ thống quản lý file và device trong linux, giao tiếp giữa tiến trình ở user space và code kernel space

+ Viết một module dùng để tạo ra số ngẫu nhiên.

+ Module này sẽ tạo một character device để cho phép các tiến trình ở userspace có thể open và read các số ngẫu nhiên.

## PHẦN 2: TÌM HIỂU VỀ LINUX KERNEL MODULE

### 1. Một định nghĩa về các thành phần trong linux kernel

#### 1.1. Sơ lược về linux kernel

- **Process management** có nhiệm vụ quản lý các tiến trình, bao gồm các công việc:
  - Tạo/hủy các tiến trình.
  - Lập lịch cho các tiến trình. Đây thực chất là lên kế hoạch: CPU sẽ thực thi chương trình khi nào, thực thi trong bao lâu, tiếp theo là chương trình nào.
  - Hỗ trợ các tiến trình giao tiếp với nhau.
  - Đồng bộ hoạt động của các tiến trình để tránh xảy ra tranh chấp tài nguyên.
- **Memory management** có nhiệm vụ quản lý bộ nhớ, bao gồm các công việc:
  - Cấp phát bộ nhớ trước khi đưa chương trình vào, thu hồi bộ nhớ khi tiến trình kết thúc.
  - Đảm bảo chương trình nào cũng có cơ hội được đưa vào bộ nhớ.
  - Bảo vệ vùng nhớ của mỗi tiến trình.
- **Device management** có nhiệm vụ quản lý thiết bị, bao gồm các công việc:
  - Điều khiển hoạt động của các thiết bị.
  - Giám sát trạng thái của các thiết bị
  - Trao đổi dữ liệu với các thiết bị.
  - Lập lịch sử dụng các thiết bị, đặc biệt là thiết bị lưu trữ (ví dụ ổ cứng).
- **File system management** có nhiệm vụ 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.
- Arch/: Kernel linux có thể được cài đặt bằng công cụ cho các server lớn. Nó hỗ trợ intel, alpha, mips, arm, cấu trúc bộ xử lý sparc. Danh mục 'arch' có thể chứa các thư mục nhỏ cho một bộ xử lý cụ thể. Mỗi thư mục nhỏ chứa 1 mã cấu trúc độc lập. Ví dụ, đối với một PC, mã sẽ là thư mục arch/i386, đối với bộ xử lý arm, mã sẽ là thư mục arch/arm/arm64.
- fs/: Linux được hỗ trợ từ nhiều hệ thống file như ext2, ext3, fat, vfat, ntfs, nfs, jffs và nhiều hơn. Tất cả mã nguồn cho những file khác nhau này được hỗ trợ trong thư mục này theo những danh mục con như fs/ext2, fs/ext3 etc. Và linux cung cấp hệ thống file ảo giống như lớp bọc cho những file khác. Hệ thống file ảo tương tác có thể giúp người dùng sử dụng hệ thống file khác với gốc ( '/'). Mã này cho vfs cũng nằm ở đây. Cơ cấu dữ liệu liên quan vfs được xác định trong include/linux/fs.h. Điều này rất quan trọng để phát triển file cho kernel.
- mm/: Thư mục này rất quan trọng là thư mục cho phát triển kernel. Nó chứa mã chung quản lý bộ nhớ và hệ thống bộ nhớ ảo. Mã kiến trúc cụ thể là trong kiến trúc thư mục / \* / mm /. Điều này một phần của mã kernel có trách nhiệm yêu cầu / giải phóng bộ nhớ, tin nhắn, xử lý lỗi trang, lập bản đồ bộ nhớ, lưu trữ khác nhau.
- Giao diện mạng (Network Interface - NET): Linux dựng sẵn TCP/IP trong kernel. Thành phần này của Linux Kernel cung cấp truy cập và kiểm soát các thiết bị mạng khác nhau.
- Bộ truyền thông nội bộ (Inter-process communication IPC): Cung cấp các phương tiện truyền thông giữa các tiến trình trong cùng hệ thống. Hệ thống phụ IPC cho phép các tiến trình khác nhau có thể chia sẻ dữ liệu với nhau.

<b>Bảng tóm tắt</b>	
<b>Thư mục</b>	<b>Vai trò</b>
/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

## 1.2. Các lệnh trong linux kernel

### ➤ User space và kernel space:

- **Kernel space:** Mã thực thi có quyền truy cập không hạn chế vào bất kỳ không gian địa chỉ nào của memory và tới bất kỳ phần cứng nào. Nó được dành riêng cho các chức năng có độ tin cậy cao nhất bên trong hệ thống. Kernel mode thường được dành riêng cho các chức năng hoạt động ở cấp độ thấp nhất, đáng tin cậy nhất của hệ điều hành. Do số lượng truy cập mà kernel có, bất kỳ sự không ổn định nào bên trong mã thực thi kernel cũng có thể dẫn đến lỗi hệ thống hoàn toàn.

- **User space:** Mã thực thi bị giới hạn truy cập. Nó là không gian địa chỉ mà các process user thông thường chạy. Những processes này không thể truy cập trực tiếp tới kernel space được. Khi đó lời gọi API được sử dụng tới kernel để truy vấn memory và truy cập thiết bị phần cứng. Bởi truy cập bị hạn chế, các trục trặc hay vấn đề gì xảy ra trong user mode chỉ bị giới hạn trong không gian hệ thống mà chúng đang hoạt động và luôn có thể khôi phục được. Hầu hết các đoạn mã đang chạy trên máy tính của bạn sẽ thực thi trong user mode.

### ➤ User mode và kernel mode:

- Khi CPU thực thi các lệnh của kernel, thì nó hoạt động ở chế độ **kernel mode**. Khi ở chế độ này, CPU sẽ thực hiện bất cứ lệnh nào trong tập lệnh của nó, và CPU có thể truy cập bất cứ địa chỉ nào trong không gian địa chỉ.

- Khi CPU thực thi các lệnh của tiến trình, thì nó hoạt động ở chế độ **user mode**. Khi ở chế độ này, CPU chỉ thực hiện một phần tập lệnh của nó, và CPU cũng chỉ được phép truy cập một phần không gian địa chỉ.

### ➤ System call và ngắt:

- **System call** là một cửa ngõ vào kernel, cho phép tiến trình trên tầng user yêu cầu kernel thực thi một vài tác vụ cho mình. Những dịch vụ này có thể là tạo một tiến trình mới (fork), thực thi I/O (read, write), hoặc tạo ra một pipe cho giao tiếp liên tiến trình (IPC).

- **Ngắt** là một sự kiện làm gián đoạn hoạt động bình thường của CPU, buộc CPU phải chuyển sang thực thi một đoạn mã lệnh đặc biệt để xử lý sự kiện đó. Thực

chất, sự kiện này là một tín hiệu điện, do một mạch điện tử nằm bên trong hoặc bên ngoài CPU phát ra.

## 2. Linux kernel module

- 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.
- Khi cần một module nhưng nó lại chưa có trong kernel space, kernel sẽ đưa module ấy vào. Quá trình này có thể diễn ra một cách tự động, với trình tự sau:
  - Bước 1: Kernel kích hoạt tiến trình **modprobe** cùng với tham số truyền vào là tên của module (ví dụ xxx.ko).
  - Bước 2: Tiến trình modprobe kiểm tra file `/lib/modules/<kernel-version>/modules.dep` xem xxx.ko có phụ thuộc vào module nào khác không. Giả sử xxx.ko phụ thuộc vào module yyy.ko.
  - Bước 3: Tiến trình modprobe sẽ kích hoạt tiến trình **insmod** để đưa các module phụ thuộc vào trước (yyy.ko), rồi mới tới module cần thiết (xxx.ko)

## PHẦN 3: TỔ CHỨC MODULE VÀ GIẢI THÍCH SOURCE CODE

### 1. Đánh giá mức độ hoàn thành

- Đánh giá tổng thể: 100%
- Đánh giá chi tiết:
  - + Tạo module sinh số ngẫu nhiên : 100%
  - + Tạo character device cho phép tiến trình ở user space có thể open và read các số tự nhiên được tạo : 100%

### 2. Tổ chức file

Đồ án gồm có 5 file trong đó có 1 file Readme.md là file hướng dẫn chạy chương trình

```
pxbao0234@ubuntu: ~/Project2 operator system
pxbao0234@ubuntu:~/Project2 operator system$ ls
Kbuild Makefile RandNum_user_test.c randomNumber.c Readme.md
pxbao0234@ubuntu:~/Project2 operator system$
```

### 3. Tạo và gắn module phát sinh số ngẫu nhiên

Để lập trình linux kernel module bằng ngôn ngữ c cần sử dụng các thư viện của linux

```
1 #include <linux/module.h>
2 #include <linux/random.h>
3 #include <linux/device.h>
4 #include <linux/init.h>
5 #include <linux/kernel.h>
6 #include <linux/fs.h>
7 #include <linux/types.h>
8 #include <linux/uaccess.h>
```

Các thư viện do linux hỗ trợ sẽ có các hàm và các marco để ta tạo ra các module. Kernel module không có hàm main nhưng có 2 marco cơ bản là `__init` và `__exit` được gắn vào `module_init`, `module_exit`.

```
58 static int __init init_random(void) //ham tao
59 {

95
96 static void __exit exit_random(void) //ham huy
97 {

111 module_init(init_random);
112 module_exit(exit_random);
...
```

Để thực hiện phát sinh số ngẫu nhiên ta dùng hàm `get_random_bytes()` của thư viện `<linux/random.h>`

```
33     int randNum;
34     int error;
35     get_random_bytes(&randNum, sizeof(randNum));
36     randNum %= MAX;
37

10 #define MAX 100000000
11 #define AUTHOR "Phan Xuan Bao<18120287>, Le Nguyen Tuan<18120631>, Pham Viet Xuan<18120658>"
12 #define DEVICE_NAME "RandomNumber"
13 #define MOD_DESC "Generative a random number. "
14 #define CLASS_NAME "RandNum"
15
```

### 4. Tạo character device phát sinh số ngẫu nhiên

## Các bước để tạo 1 character device để phát sinh số ngẫu nhiên

- Đăng ký số hiệu file thiết bị ta sử dụng hàm *register\_chrdev*

```

58 static int __init init_random(void) //ham tao
59 {
60     //dang ky mot major cho file thiet bi
61     majorNum = register_chrdev(0, DEVICE_NAME, &f_ops);
62     //neu that bai
63     if (majorNum < 0)
64     {
65         //Xuat thong bao loi ra man hinh, return
66         printk(KERN_INFO "RandNum: Register a major number failed");
67         return majorNum;
68     }
69     //neu thanh cong
70     printk(KERN_INFO "RandNum: Register a major successfully. Major number is: %d", majorNum);
71 }

```

- Đăng ký toán tử file sẽ sử dụng: trong đồ án này ta sẽ sử dụng 3 toán tử là *open*, *release* và *read* đi kèm với đó là 3 hàm thực hiện các toán tử này

```

50 static struct file_operations f_ops = //file operand
51 {
52     .open = f_open,
53     .release = f_release,
54     .read = f_read,
55 };
56
22 static int f_open(struct inode *inodep, struct file *filep){
23     timesOfOpens++;
24     printk(KERN_INFO "RandNum: Device has been opened %d time(s)\n", timesOfOpens);
25     return 0;
26 }
27 static int f_release(struct inode *inodep, struct file *filep){
28     printk(KERN_INFO "RandNum: Device successfully closed\n");
29     return 0;
30 }
31 static ssize_t f_read(struct file *filep, char* usr_space, size_t len, loff_t* offset)
32 {
33     int randNum;
34     int error;
35     get_random_bytes(&randNum, sizeof(randNum));
36     randNum %= MAX;
37
38     error = copy_to_user(usr_space, &randNum, sizeof(randNum));
39
40     if (error == 0){
41         printk(KERN_INFO "RandNum: Sent random number to the user\n");
42         return 0;
43     }
44     else{
45         printk(KERN_INFO "RandNum: Failed to send random number to the user\n");
46         return -EFAULT;
47     }
48 }

```

- Đăng ký lớp thiết bị ảo và khởi tạo thiết bị tự động với số hiệu đã đăng ký với trường hợp đã đăng ký thành công số hiệu cho thiết bị ở bước 1.

Đăng ký lớp thiết bị ảo ta dùng hàm *class\_create()*. Nếu xảy ra lỗi không đăng ký được lớp thiết bị ảo thì sẽ hủy đăng ký số hiệu và return ta dùng hàm *unregister\_chrdev()*.

```

71
72 //Tao mot device class co ten la DEVICE_NAME
73 RandNum_class = class_create(THIS_MODULE, CLASS_NAME);
74

```



```

75     if (IS_ERR(RandNum_class))
76     {
77         //Neu xay ra loi, unregister majorNum va return
78         unregister_chrdev(majorNum,DEVICE_NAME);
79         printk(KERN_ALERT"register device class failed!");
80         return PTR_ERR(RandNum_class);
81     }
82     printk(KERN_INFO "RandNum: register device class successfulty.\n");

```

Đăng ký và khởi tạo thiết bị với số hiệu đã đăng ký ta dùng hàm *device\_create()*. Nếu đăng ký không thành công thì hủy đăng ký số hiệu đã đăng ký trước đó ta dùng hàm *unregister\_chrdev()*.

```

83     //Dang ky mot device driver
84     RandDevice = device_create(RandNum_class,NULL, MKDEV(majorNum,0), NULL, DEVICE_NAME);
85     if (IS_ERR(RandDevice))
86     {
87         unregister_chrdev(majorNum,DEVICE_NAME);
88         printk(KERN_ALERT "Register device driver failed!\n");
89         return PTR_ERR(RandDevice);
90     }
91     printk(KERN_INFO "RandNum: Create device class correctly\n");
92     return 0;

```

- Gửi kết quả đến chương trình người dùng.

Thực hiện trong toán tử *read* , phát sinh số ngẫu nhiên và chép về vùng nhớ của user space bằng hàm *copy\_to\_user()*

```

37
38     error = copy_to_user(usr_space, &randNum, sizeof(randNum));
39
40     if (error == 0){
41         printk(KERN_INFO "RandNum: Sent random number to the user\n");
42         return 0;
43     }
44     else{
45         printk(KERN_INFO "RandNum: Failed to send random number to the user\n");
46         return -EFAULT;
47     }

```

Chương trình người dùng đọc lên được thực hiện trong file *RanNum\_user\_test.c*

```

18     // Mo file voi che cho Read Only
19     f = open("/dev/RandomNumber", O_RDONLY);
20     if (f < 0){
21         perror("Open the device faile. Error: ");
22         return errno;
23     }
24
25     // doc randNum tu file fd, tra ve gia tri random
26     read(f, &randNum, sizeof(randNum));

```

- Hủy các số hiệu, lớp ảo và thiết bị ảo đã đăng ký bằng các hàm
  - + *device\_destroy()* → Xóa device đã đăng ký
  - + *class\_unregister()* → Hủy đăng ký lớp thiết bị ảo
  - + *class\_destroy()* → Xóa lớp thiết bị ảo
  - + *unregister\_chrdev()* → Xóa số hiệu đã đăng ký

```

96 static void __exit exit_random(void) //ham huy
97 {
98     // Xoa device
99     device_destroy(RandNum_class, MKDEV(majorNum, 0));
100    // huy register device class
101    class_unregister(RandNum_class);
102    // xoa device class
103    class_destroy(RandNum_class);
104    // unregister major number
105    unregister_chrdev(majorNum, DEVICE_NAME);
106
107    printk(KERN_INFO "RandNum: Unregistered random number.");
108    return;
109 }
110

```

## 5. Một số thông tin về module đã tạo

Gồm có:

- AUTHOR → Tên người tạo ra module
- MOD\_DESC → Miêu tả về module đã tạo
- DEVICE\_NAME → Tên thiết bị đăng ký với số hiệu đã đăng ký từ trước

```

10 #define MAX 1000000000
11 #define AUTHOR "Phan Xuan Bao<18120287>, Le Nguyen Tuan<18120631>, Pham Viet Xuan<18120658>"
12 #define DEVICE_NAME "RandomNumber"
13 #define MOD_DESC "Generative a random number. "
14 #define CLASS_NAME "RandNum"
15
114 MODULE_LICENSE("GPL");
115 MODULE_AUTHOR(AUTHOR);
116 MODULE_DESCRIPTION(MOD_DESC);
117 MODULE_VERSION("1");

```

## 6. Makefile và cách chạy chương trình

### 6.1. Makefile

File này chứa các thông tin để tạo ra các file trong quá trình chạy lệnh với *make* trong linux ( ở đây ta dùng ubuntu)

Để thực hiện chạy và cài đặt module đơn giản và nhanh chóng với make trong linux thì ta sẽ dùng cách viết gọn cho lệnh như sau:

- make insert: `sudo insmod randNumber.ko` --> Lệnh này dùng để gắn module đã tạo vào kernel
- make remove: `sudo rmmod randNumber.ko` --> Lệnh này dùng để gỡ module đã gắn vào kernel
- make test: `cc -o RandNum_user_test RandomNumber_user_test.c` --> Lệnh này để tạo chương trình người dùng từ file `RandNum_user_test.c`

- make run\_user: sudo ./RandNum\_user\_test --> Lệnh này dùng để chạy chương trình user từ file RandNum\_user\_test.o đã tạo từ lệnh make test

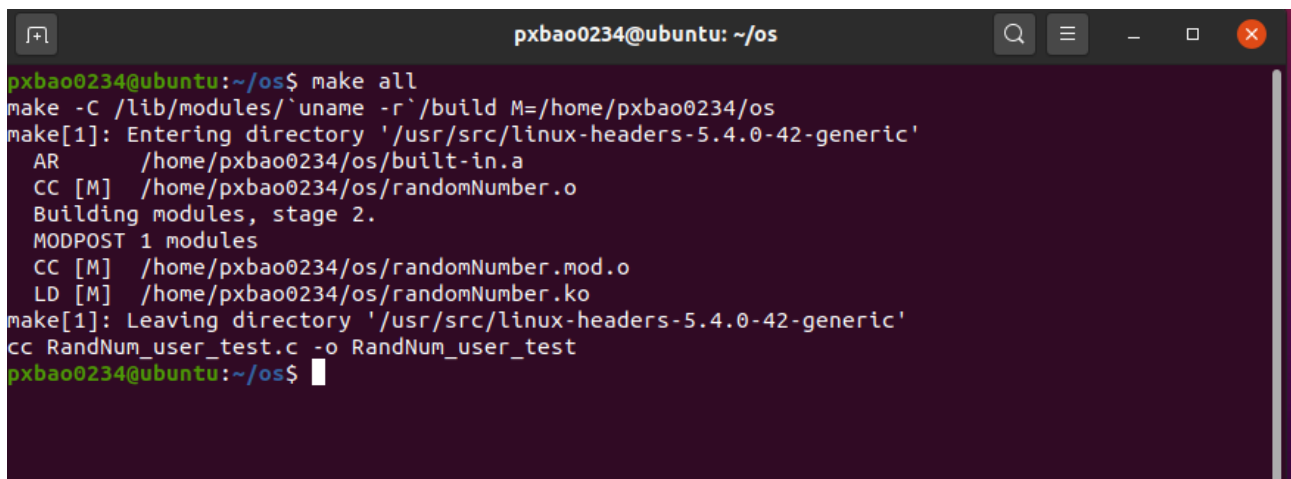
```

1 KDIR = /lib/modules/`uname -r`/build
2 all:
3     make -C $(KDIR) M=$(PWD)
4     $(CC) RandNum_user_test.c -o RandNum_user_test
5 clean:
6     make -C $(KDIR) M=$(PWD) clean
7     rm RandNum_user_test
8 test: RandNum_user_test.c
9     cc -o RandNum_user_test RandNum_user_test.c
10 insert:
11     sudo insmod randomNumber.ko
12 remove:
13     sudo rmmod randomNumber.ko
14 run_user:
15     clear
16     sudo ./RandNum_user_test

```

6.2. Tạo, gắn module vào kernel và chạy chương trình người dùng

- Tạo module : sử dụng lệnh *make all* → tạo ra file module có tên là randomNumber.ko



```

pxbao0234@ubuntu: ~/os
pxbao0234@ubuntu:~/os$ make all
make -C /lib/modules/`uname -r`/build M=/home/pxbao0234/os
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-42-generic'
AR      /home/pxbao0234/os/built-in.a
CC [M]  /home/pxbao0234/os/randomNumber.o
Building modules, stage 2.
MODPOST 1 modules
CC [M]  /home/pxbao0234/os/randomNumber.mod.o
LD [M]  /home/pxbao0234/os/randomNumber.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-42-generic'
cc RandNum_user_test.c -o RandNum_user_test
pxbao0234@ubuntu:~/os$

```

- Kiểm tra thông tin của module đã tạo: sử dụng lệnh *modinfo randomNumber.ko*



```

pxbao0234@ubuntu:~/os$ modinfo randomNumber.ko
filename:       /home/pxbao0234/os/randomNumber.ko
version:        1
description:    Generative a random number.
author:         Phan Xuan Bao<18120287>, Le Nguyen Tuan<18120631>, Pham Viet Xuan<18120658>
license:        GPL
srcversion:     4A3917765E12E2222CF8D0F
depends:
retpoline:      Y
name:           randomNumber
vermagic:       5.4.0-42-generic SMP mod_unload
pxbao0234@ubuntu:~/os$

```

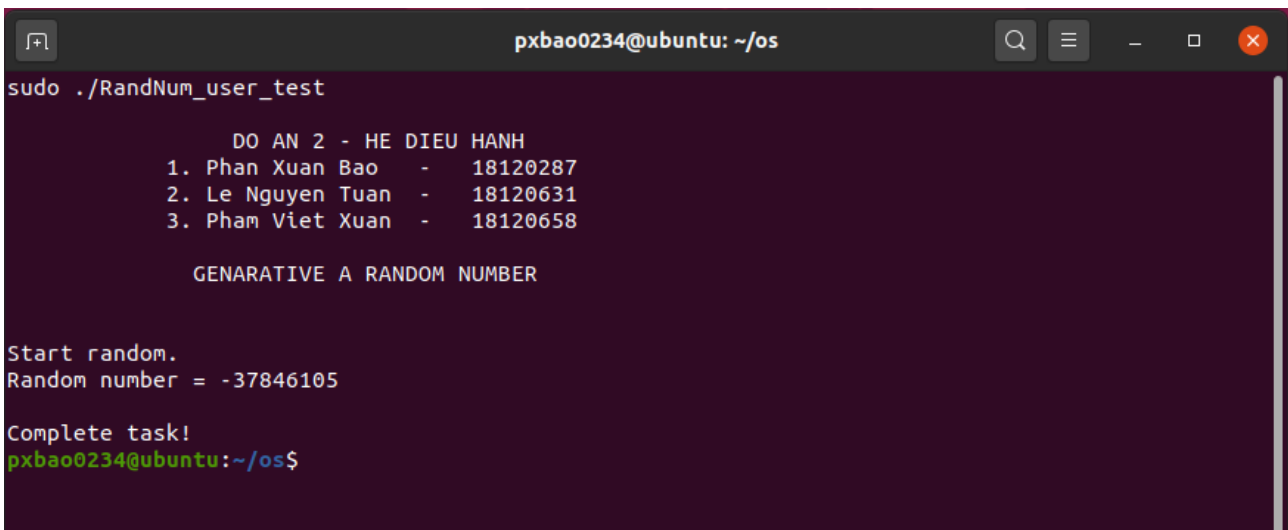
- Gắn module vào kernel bằng lệnh *make insert*

```
pxbao0234@ubuntu:~/os$ make insert
sudo insmod randomNumber.ko
[sudo] password for pxbao0234:
pxbao0234@ubuntu:~/os$
```

- Tạo chương trình người dùng: sử dụng lệnh *make test* → tạo ra file chương trình có tên là *RanNum\_user\_test.o*

```
pxbao0234@ubuntu:~/os$ make test
cc -o RanNum_user_test RanNum_user_test.c
pxbao0234@ubuntu:~/os$
```

- Khởi chạy chương trình người dùng: sử dụng lệnh *make run\_user*



```
pxbao0234@ubuntu: ~/os
sudo ./RanNum_user_test

        ĐỒ AN 2 - HỆ ĐIỀU HÀNH
1. Phan Xuan Bao   -   18120287
2. Le Nguyen Tuan  -   18120631
3. Pham Viet Xuan  -   18120658

        GENERATIVE A RANDOM NUMBER

Start random.
Random number = -37846105

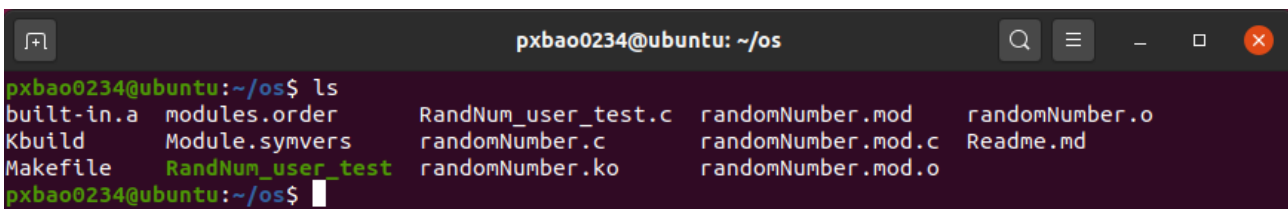
Complete task!
pxbao0234@ubuntu:~/os$
```

⇒ Chương trình người dùng in kết quả ra màn hình.

Có thể kiểm tra cách module hoạt động bằng lệnh *dmesg*

```
[15161.775538] RandNum: Register a major successfully. Major number is: 240
[15161.775827] RandNum: register device class successfully.
[15161.811314] RandNum: Create device class correctly
[15262.922793] RandNum: Device has been opened 1 time(s)
[15262.922805] RandNum: Sent random number to the user
[15262.923061] RandNum: Device successfully closed
pxbao0234@ubuntu:~/os$
```

- Các file sau khi chạy chương trình xong



```
pxbao0234@ubuntu:~/os$ ls
built-in.a  modules.order  RandNum_user_test.c  randomNumber.mod  randomNumber.o
Kbuild      Module.symvers  randomNumber.c        randomNumber.mod.c  Readme.md
Makefile    RandNum_user_test  randomNumber.ko       randomNumber.mod.o
pxbao0234@ubuntu:~/os$
```

- Gỡ module ra khỏi kernel: sử dụng lệnh *make remove*

```
pxbao0234@ubuntu:~/os$ make remove
sudo rmmod randomNumber.ko
pxbao0234@ubuntu:~/os$
```