

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐẠI HỌC QUỐC GIA TP HCM
KHOA CÔNG NGHỆ THÔNG TIN
MÔN HỆ ĐIỀU HÀNH



ĐỒ ÁN

TÌM HIỂU VÀ LẬP TRÌNH LINUX KERNEL MODULE

CÁC THÀNH VIÊN TRONG NHÓM

Trần Đức Lộc – 18120439

Cam Quốc Bảo Long – 18120442

Dương Thành Long – 18120444

GVHD: Thầy Lê Giang Thanh

Thầy Trần Trung Dũng

TP. HỒ CHÍ MINH – NĂM 2020

Đồ án

BÁO CÁO NHÓM	3
1. Thông tin nhóm:	3
2. Nguyên tắc hoạt động nhóm:.....	3
3. Kế hoạch làm việc:	3
4. Phân công :	4
5. Môi trường lập trình: Ubuntu và Visual Studio Code.....	4
6. Phần tìm hiểu của nhóm em:	4
a. Giới thiệu linux kernel	4
b. Giới thiệu Linux Kernel Module	5
c. Cách viết Linux Kernel Module	5
d. Character device driver	7
e. Cấu trúc và cách tạo file thiết bị	7
7. Một số ảnh minh họa đồ án	8
a. Source code module	8
b. Source code user read	11
c. Compile và run	12
8. Mức độ hoàn thành: 100%	12
9. Tài liệu tham khảo:	12

BÁO CÁO NHÓM

1. Thông tin nhóm:

STT	MSSV	Họ và Tên	Email
1	18120439	Trần Đức Lộc	ducloc20810@gmail.com
2	18120442	Cam Quốc Bảo Long	18120442@student.hcmus.edu.vn
3	18120444	Dương Thành Long	18120444@student.hcmus.edu.vn

2. Nguyên tắc hoạt động nhóm:

- Nghiêm túc trong lúc hoạt động nhóm.
- Có tinh thần trách nhiệm với nhóm, với tập thể.
- Biết lắng nghe và cho ý kiến.

3. Kế hoạch làm việc:

Giao tiếp và phân công thông qua facebook.

Các thành viên sau khi nhận tin nhắn hay thông báo phải hồi đáp lại để chứng tỏ đã nhận và đã đọc tin nhắn.

Nếu thành viên không hồi đáp tin nhắn hay thông báo nhóm trưởng sẽ gọi cho thành viên đó.

4. Phân công :

STT	MSSV	Họ và Tên	Nhiệm vụ	Hoàn thành/ Chưa hoàn thành
1	18120439	Trần Đức Lộc	+ Viết một module dùng để tạo ra số ngẫu nhiên. + Viết hàm tạo character device + Viết báo cáo	Hoàn thành
2	18120442	Cam Quốc Bảo Long	+ Viết hàm tạo character device + Tóm tắt tài liệu hướng dẫn + Chạy thử các hàm trong tài liệu hướng dẫn và giải thích	Hoàn thành
3	18120444	Dương Thành Long	+ Chạy thử các hàm trong tài liệu hướng dẫn + Viết tài liệu giải thích tài liệu hướng dẫn + Tóm tắt tài liệu hướng dẫn	Hoàn thành

5. Môi trường lập trình: Ubuntu và Visual Studio Code.

6. Phần tìm hiểu của nhóm em:

a. Giới thiệu linux kernel

i. Lịch sử

- Linux kernel được tạo ra năm 1991 bởi một học sinh, **Linus Torvalds**, như một thú vui của ông.
- Linus Torvalds đã tạo ra một cộng đồng rộng lớn cũng như mạnh mẽ các developer và user xung quanh Linux.
- Ngày nay, hơn hàng ngàn người đóng góp cho mỗi bản release của Linux kernel, bao gồm các cá nhân, công ty, tổ chức lớn nhỏ

ii. Chức năng

- Sự linh động và hỗ trợ hardware mạnh mẽ. Có thể chạy trên hầu hết các kiến trúc.
- Khả năng mở rộng. Có thể chạy trên các siêu máy tính cũng như các thiết bị tí hon (4Mb RAM là đủ).
- Tuân thủ các chuẩn cũng như tính tương thích.
- Bảo mật. Code của linux được review bởi rất nhiều chuyên gia, do đó có thể rà soát được hầu hết những lỗi.
- Ổn định và tin cậy.
- Tính module hóa. Có thể chỉ include những thứ mà hệ thống cần ở thời điểm chạy.
- Dễ dàng lập trình. Có thể học từ những source code có sẵn.

iii. Tổ chức

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.

- **Kernel space** là vùng không gian chứa các lệnh và dữ liệu của kernel.
- **User space** là vùng không gian chứa các lệnh và dữ liệu của các tiến trình.

b. Giới thiệu 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

c. Cách viết Linux Kernel Module

- Khai báo các thư viện cần thiết

```
#include <linux/module.h>
#include <linux/init.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/uaccess.h>
#include <linux/random.h>
```

- Viết hàm khởi tạo

```
static int __init ofcd_init(void) /* Constructor */
```

- Viết hàm hủy

```
static void __exit ofcd_exit(void) /* Destructor */
```

- Tạo module

```
module_init(ofcd_init);
module_exit(ofcd_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Tran Duc Loc");
MODULE_DESCRIPTION("Random number character device driver");
```

- Để có thể compile module và sử dụng ta cần tạo 2 file là Makefile và Kbuild có cấu trúc lần lượt như sau:

```
KDIR = /lib/modules/`uname -r`/build
all:
    make -C $(KDIR) M=`pwd`
clean:
    make -C $(KDIR) M=`pwd` clean
```

```
EXTRA_CFLAGS = -Wall
obj-m = randModule.o
```

Makefile

Kbuild

- Thẻ all chứa câu lệnh để biên dịch các module trong thư mục hiện tại.
- Thẻ clean chứa lệnh xóa tất cả các object file có trong thư mục hiện tại.
- (obj-m) sẽ là đối tượng sẽ được biên dịch theo kiểu kernel module

- Cờ -Wall cho phép trình biên dịch hiển thị tất cả các bản tin cảnh báo trong quá trình biên dịch
- Ta dùng lệnh **make** để biên dịch. Hệ thống sau khi biên dịch sẽ tạo ra một file có đuôi .ko. Sau đó ta dùng lệnh **sudo insmod "filename".ko** để gắn module. Dùng lệnh **lsmod** để kiểm tra module mới được gắn vào. Để kiểm tra các chức năng chạy của module hay lịch sử chạy của module ta dùng lệnh **sudo dmesg -c**
- Để tháo module ta dùng lệnh **sudo rmmod "filename".ko**
- Dọn những file được tạo ra trong quá trình biên dịch bằng lệnh **make clean**
- Chú ý: Trong module ta phải dùng lệnh **printk** thay cho **printf**

d. Character device driver

- Character device driver là driver của các thiết bị thuộc kiểu hướng byte (byte - oriented)
- Việc kết nối từ ứng dụng đến thiết bị được thực hiện hoàn chỉnh thông qua 4 thực thể chính liên quan gồm:
 - o Application (ứng dụng)
 - o Character device file (File thiết bị)
 - o Character device driver (Driver thiết bị)
 - o Character device (Thiết bị)
- Một character device file phải có số hiệu thiết bị gọi là major và minor number để kết nối với driver.
- Việc tạo file thiết bị có thể được tạo thủ công hoặc tự động.
- Việc kết nối giữa file thiết bị và driver được thực hiện thông qua 2 bước sau:
 - o Đăng ký số hiệu cho file thiết bị
 - o Kết nối các thao tác file thiết bị với các hàm tương ứng trong driver
- Các thao tác trên file thiết bị cũng tương tự như các thao tác trên một file thông thường.

e. Cấu trúc và cách tạo file thiết bị

- Viết hàm tạo các thao tác trên file

```
static int my_open(struct inode *i, struct file *f)
static int my_close(struct inode *i, struct file *f)
static ssize_t my_read(struct file *f, char __user *buf, size_t len, loff_t *off)
static ssize_t my_write(struct file *f, const char __user *buf, size_t len, loff_t *off)
```

- Kết nối các thao tác trên với file

```
static struct file_operations pugs_fops =
{
    .owner = THIS_MODULE,
    .open = my_open,
    .release = my_close,
    .read = my_read,
    .write = my_write};
```

- Viết hàm khởi tạo module driver và đăng kí số hiệu

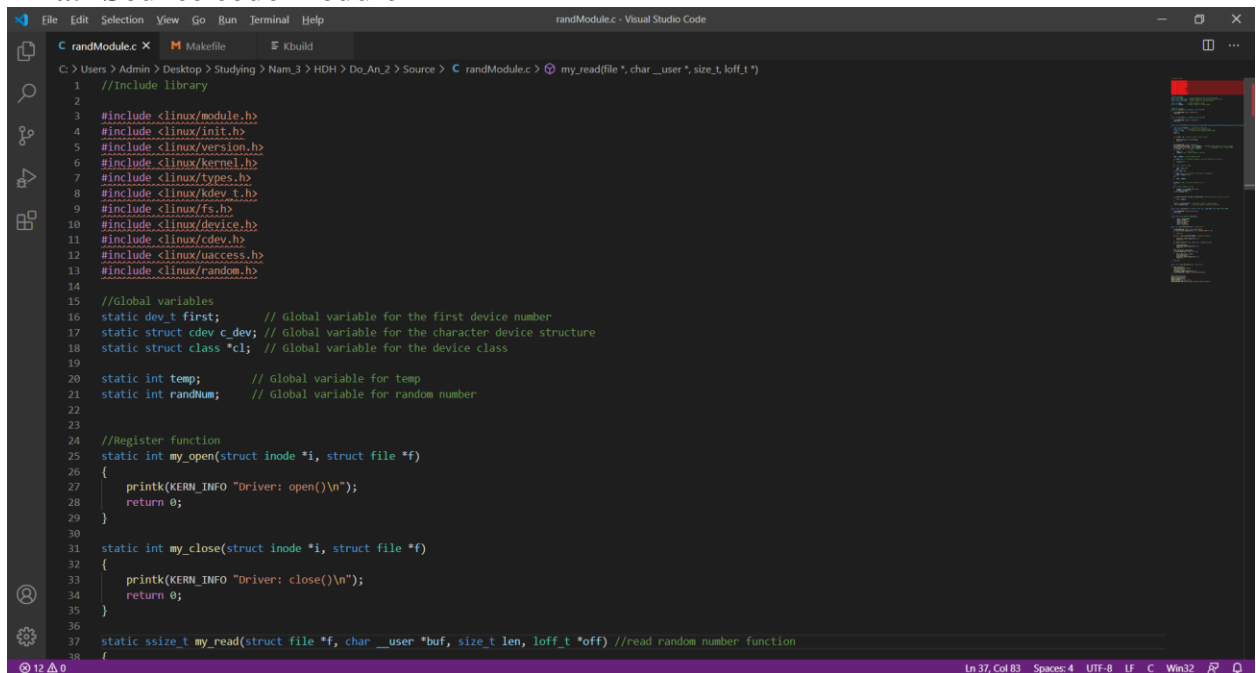
```
static int __init ofcd_init(void) /* Constructor */
```

- Viết hàm hủy module driver và hủy đăng kí số hiệu

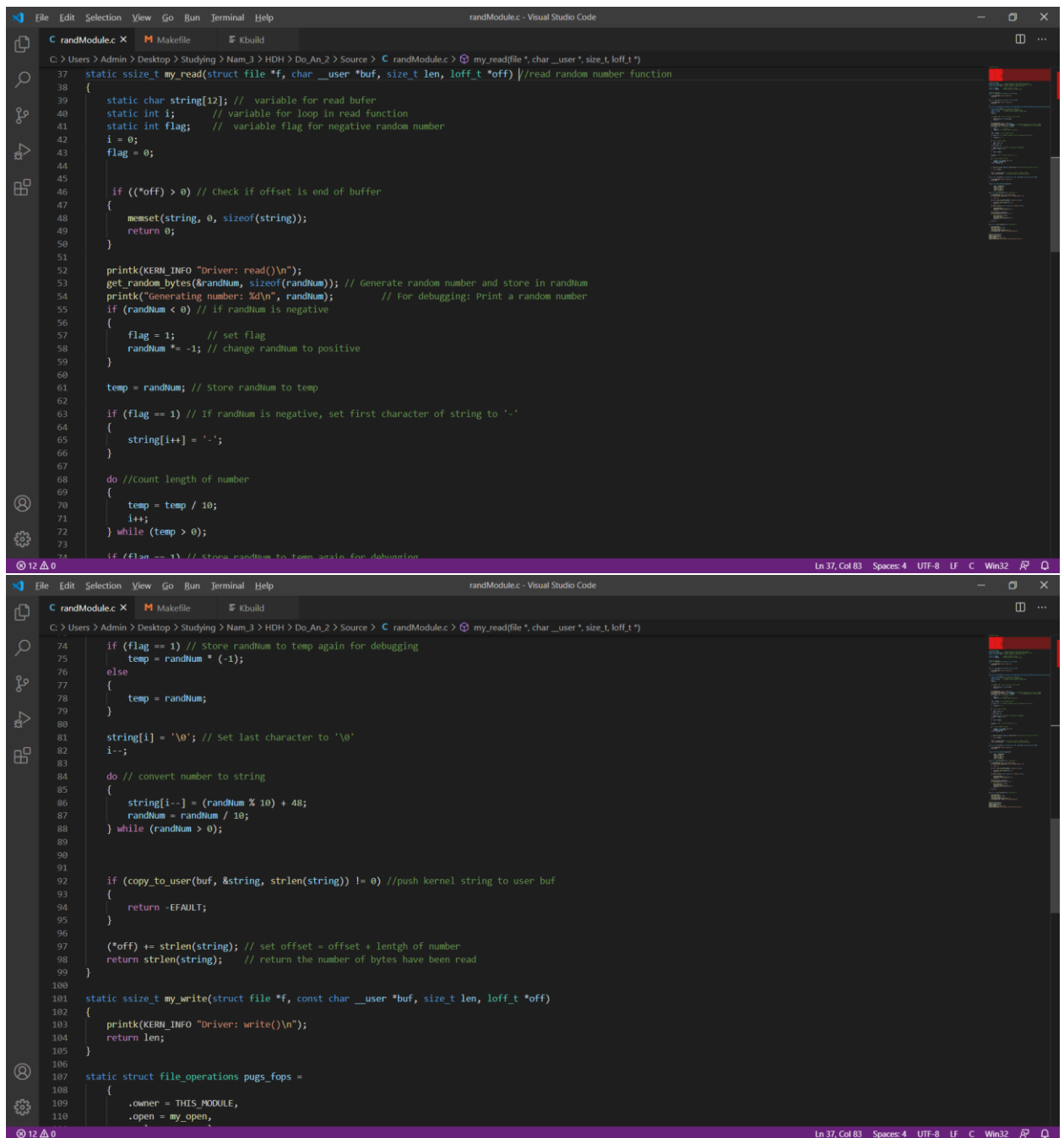
```
static void __exit ofcd_exit(void) /* Destructor */
```

7. Một số ảnh minh họa đồ án

a. Source code module



```
randModule.c - Visual Studio Code
C randModule.c x Makefile Kbuild
C: > Users > Admin > Desktop > Studying > Nam_3 > HDH > Do_An_2 > Source > C randModule.c > my_read(file, char __user *, size_t, loff_t *)
1 //Include library
2
3 #include <linux/module.h>
4 #include <linux/init.h>
5 #include <linux/version.h>
6 #include <linux/kernel.h>
7 #include <linux/types.h>
8 #include <linux/kdev_t.h>
9 #include <linux/fs.h>
10 #include <linux/device.h>
11 #include <linux/cdev.h>
12 #include <linux/uaccess.h>
13 #include <linux/random.h>
14
15 //Global variables
16 static dev_t first; // Global variable for the first device number
17 static struct cdev c_dev; // Global variable for the character device structure
18 static struct class *cl; // Global variable for the device class
19
20 static int temp; // Global variable for temp
21 static int randNum; // Global variable for random number
22
23
24 //Register function
25 static int my_open(struct inode *i, struct file *f)
26 {
27     printk(KERN_INFO "Driver: open()\n");
28     return 0;
29 }
30
31 static int my_close(struct inode *i, struct file *f)
32 {
33     printk(KERN_INFO "Driver: close()\n");
34     return 0;
35 }
36
37 static ssize_t my_read(struct file *f, char __user *buf, size_t len, loff_t *off) //read random number function
38 {
```

```
1  File Edit Selection View Go Run Terminal Help
2  randModule.c - Visual Studio Code
3  C:\Users> Admin> Desktop> Studying> Nam_3> HDH> Do_An_2> Source> C:\randModule.c> my_read(file *, char __user *, size_t, loff_t *)
4
5  37 static ssize_t my_read(struct file *f, char __user *buf, size_t len, loff_t *off) //read random number function
6  {
7
8      static char string[12]; // variable for read bufer
9      static int i; // variable for loop in read function
10     static int flag; // variable flag for negative random number
11     i = 0;
12     flag = 0;
13
14     if ((*off) > 0) // Check if offset is end of buffer
15     {
16         memset(string, 0, sizeof(string));
17         return 0;
18     }
19
20     printk(KERN_INFO "Driver: read()\n");
21     get_random_bytes(&randNum, sizeof(randNum)); // Generate random number and store in randNum
22     printk("Generating number: %d\n", randNum); // For debugging: Print a random number
23     if (randNum < 0) // if randNum is negative
24     {
25         flag = 1; // set flag
26         randNum *= -1; // change randNum to positive
27     }
28
29     temp = randNum; // Store randNum to temp
30
31     if (flag == 1) // If randNum is negative, set first character of string to '-'
32     {
33         string[i++] = '-';
34     }
35
36     do //Count length of number
37     {
38         temp = temp / 10;
39         i++;
40     } while (temp > 0);
41
42     if (flag == 1) // Store randNum to temp again for debugging
43     {
44         temp = randNum * (-1);
45     }
46     else
47     {
48         temp = randNum;
49     }
50
51     string[i] = '\0'; // Set last character to '\0'
52     i--;
53
54     do // convert number to string
55     {
56         string[i--] = (randNum % 10) + 48;
57         randNum = randNum / 10;
58     } while (randNum > 0);
59
60     if (copy_to_user(buf, &string, strlen(string)) != 0) //push kernel string to user buf
61     {
62         return -EFAULT;
63     }
64
65     (*off) += strlen(string); // set offset = offset + length of number
66     return strlen(string); // return the number of bytes have been read
67 }
68
69 static ssize_t my_write(struct file *f, const char __user *buf, size_t len, loff_t *off)
70 {
71     printk(KERN_INFO "Driver: write()\n");
72     return len;
73 }
74
75 static struct file_operations pugs_fops =
76 {
77     .owner = THIS_MODULE,
78     .open = my_open,
```

```
File Edit Selection View Go Run Terminal Help
randModule.c - Visual Studio Code

C:\Users\Admin\Desktop> Studying > Nam_3 > HDH > Do_An_2 > Source > C:\randModule.c > my_read(file *, char __user *, size_t, loff_t *)

106
107 static struct file_operations pugs_fops =
108 {
109     .owner = THIS_MODULE,
110     .open = my_open,
111     .release = my_close,
112     .read = my_read,
113     .write = my_write;
114 }
115 static int __init ofcd_init(void) /* Constructor */
116 {
117     printk(KERN_INFO "Hello: ofcd registered\n");
118     if (alloc_chrdev_region(&first, 0, 1, "Random number") < 0)
119     {
120         return -1;
121     }
122     if ((cl = class_create(THIS_MODULE, "chardrv")) == NULL)
123     {
124         unregister_chrdev_region(first, 1);
125         return -1;
126     }
127     if (device_create(cl, NULL, first, NULL, "myrand") == NULL)
128     {
129         class_destroy(cl);
130         unregister_chrdev_region(first, 1);
131         return -1;
132     }
133     cdev_init(&c_dev, &pugs_fops);
134     if (cdev_add(&c_dev, first, 1) == -1)
135     {
136         device_destroy(cl, first);
137         class_destroy(cl);
138         unregister_chrdev_region(first, 1);
139         return -1;
140     }
141     return 0;
142 }
143
```

```
static void __exit ofcd_exit(void) /* Destructor */
{
    cdev_del(&c_dev);
    device_destroy(cl, first);
    class_destroy(cl);
    unregister_chrdev_region(first, 1);
    printk(KERN_INFO "Goodbye: ofcd unregistered\n");
}

module_init(ofcd_init);
module_exit(ofcd_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Tran Duc Loc");
MODULE_DESCRIPTION("Random number character device driver");
```

b. Source code user read

```
#include <stdio.h>
#include<fcntl.h>
#include<string.h>
#include<stdlib.h>
#include <unistd.h>
#define buffLength 6
static char readResult[buffLength];

int main()
{
    int r,fd;
    fd= open("/dev/myrand",O_RDWR);
    if(fd<0)
    {
        perror("Open failed\n");
        return -1;
    }
    printf("Device opened\n");
    printf("Device is running\n");
    r=read(fd,readResult,buffLength);
    if(r<0)
    {
        perror("Read failed\n");
    }
    printf("Your random number: %s\n",readResult);
    printf("Device closed\n");
    close(fd);
    return 0;
}
```

c. Compile và run

```
loc@loc-VirtualBox: ~/a/Source
loc@loc-VirtualBox:~/a/Source$ make
make -C /lib/modules/`uname -r`/build M=`pwd`
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-52-generic'
  AR      /home/loc/a/Source/built-in.a
  CC [M]  /home/loc/a/Source/randModule.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/loc/a/Source/randModule.mod.o
  LD [M]  /home/loc/a/Source/randModule.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-52-generic'
loc@loc-VirtualBox:~/a/Source$ sudo insmod randModule.ko
loc@loc-VirtualBox:~/a/Source$ sudo ./test
Device opened
Device is running
Your random number: 162071158
Device closed
loc@loc-VirtualBox:~/a/Source$ sudo rmmod randModule.ko
loc@loc-VirtualBox:~/a/Source$
```

8. Mức độ hoàn thành: 100%

9. Tài liệu tham khảo:

- Tài liệu hướng dẫn lập trình kernel linux module trên moodle
- <https://www.linuxquestions.org/questions/programming-9/random-numbers-kernel-642087/>