**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

**ĐỒ ÁN 2 HỆ ĐIỀU HÀNH**

**LINUX KERNEL MODULE**

**& SYSTEM CALL HOOKING**

C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.png

Bộ môn Công nghệ phần mềm

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP HCM

**MỤC LỤC**

[**1** **Bảng đánh giá thành viên** 3](#_Toc26109858)

[**2** **Mô tả tổ chức, thiết kế của Linux Kernel phát sinh số ngẫu nhiên** 4](#_Toc26109859)

[**3** **Test case linux kernel** 9](#_Toc26109860)

[**4** **Mô tả tổ chức,thiết kế của system call hooking** 10](#_Toc26109861)

[**5 Test case syscall hook** 12](#_Toc26109862)

[**6 Tài liệu tham khảo** 14](#_Toc26109863)

# **Bảng đánh giá thành viên**

|  |  |  |  |
| --- | --- | --- | --- |
| **MSSV** | **Họ Tên** | **Chức năng** | **% đóng góp (tối đa 100%)** |
| 1712126 | Võ Trọng Phúc | Viết báo cáo, viết các chức năng: 1,2 | 100% |
| 1712147 | Trịnh Tấn Tài | Viết báo cáo, viết các chức năng: 1,2 | 100% |
| 1712148 | Hà Duy Tân | Test chương trình, viết các chức năng: 1,2 | 100% |

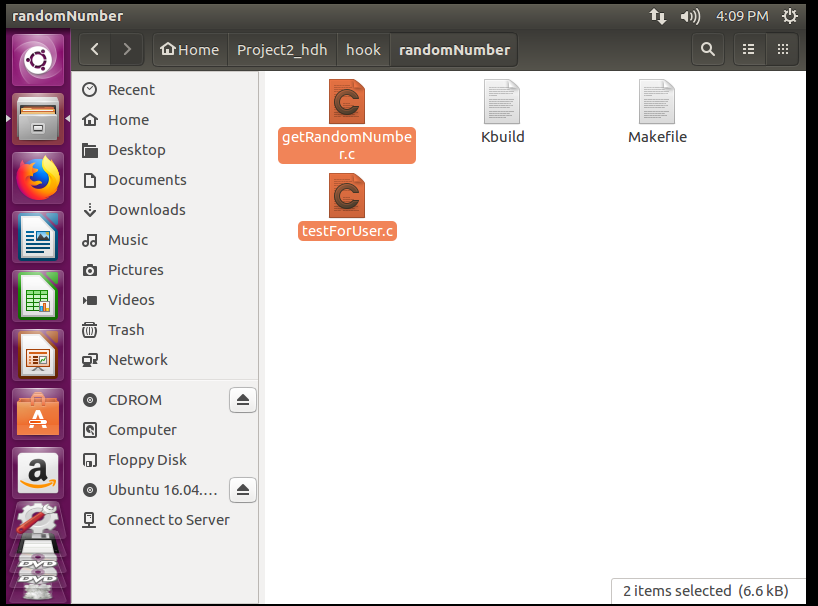
# **Mô tả tổ chức, thiết kế của Linux Kernel phát sinh số ngẫu nhiên**

1) Cấu trúc:

- Chương trình sẽ gồm file **testForUser.c** để gọi xuống kernel module được tạo để phát sinh số ngẫu nhiên

- Kernel module được tạo sẽ nằm bên dưới kernel space trong file **getRandomNumber.c**, sẽ tiến hành tạo ra số ngẫu nhiên và gởi lại cho user khi được yêu cầu

-Ngoài ra ta sẽ tiến hành tạo thêm **Makefile** và **Kbuild** để biên dịch chương trình



2) Các tham số, biến môi trường có trong kernel module phát sinh số ngẫu nhiên

**a) Struct file\_operators:**

struct file\_operations fops=

{

.open=dev\_open;

.read=dev\_read;

.release=dev\_release;

}

- Mục đích: Khai báo biến một struct file\_operators. Mục địch của struct này là để lưu các chức năng thực thi của kernel module yêu cầu khi open(mỗi khi mở kernel ra ), read(mỗi khi phát sinh random number và gởi đến user),release(mỗi khi phát sinh xong số và đóng thiết bị)

**b) Struct \_randomNum\_drv:**

struct \_randomNum\_drv

{

Struct class\* dev\_class\_randomNum;// Lưu kết quả trả về của hàm class\_create

Struct device\* dev\_randomNum;//Lưu kết quả trả về của ham device\_create

}randomNum\_drv;

- Mục đích: Khai báo struct randomNum\_drv để lưu các trường device class và device khi ta tiến hành khởi tạo device.

**c) static int majorNumber:**

-Mục đích: biến này để lưu giá trị nhận được khi khởi tạo ngẫu nhiên major number cho device

**d) static int numberOpens:**

- Mục đích: biến này để lưu giá trị của số lần ta gởi yêu cầu và nhận số random number tới kernel module(mỗi lần được yêu cầu và kernel module gởi một số random number thì biến này sẽ tăng lên một đơn vị)

3) Các hàm quan trọng có trong kernel module phát sinh số ngẫu nhiên

* static int \_\_init randomNumber\_driver\_init(void): hàm khởi tạo device phát sinh số ngẫu nhiên
* static void \_\_exit randomNumber\_driver\_exit(void): hàm giải phóng device class, device driver,hủy và xóa character device
* static int dev\_open(struct inode \*inodep, struct file \*filep): hàm tính số lần mà user gởi yêu cầu vào kernel để nhận số random
* static ssize\_t dev\_read(struct file \*filep, char \*buffer, size\_t len, loff\_t \*offset): hàm phát sinh số ngẫu nhiên và gởi đến user
* static int dev\_release(struct inode \*inodep, struct file \*filep) : hàm đóng device
* int register\_chrdev (unsigned int major, const char \* name,const struct file\_operations \* fops): hàm này nhận vào **major** là một số khác 0(nếu muốn cấp phát tĩnh) hoặc bằng 0 nếu muốn cấp phát động major number**, name** là tên của character device và một **file\_operators** là struct chứa các thao tác trên device để đăng kí major number cho character device. Thành công nếu kếu quả trả về lớn hơn 0
* struct class \* class\_create (struct module \* owner,const char \* name): hàm này để tạo device class, nhận vào tham số là **struct module \* owner** trỏ đến module đang khởi tạo và **const char \* name** là tên của device class. Thành công nếu kếu quả trả về khác NULL
* struct device \* device\_create (struct class \* class, struct device \* parent,dev\_t devt,const char \* fmt,...): hàm này để khởi tạo character device , tham số nhận vào là **struct class \* class** chứa device class vừa khởi tạo, **struct device \* parent** chứa parent struct của device(nếu không có thì truyền vào NULL), **dev\_t devt** chứa character device với major number vừa khởi tạo bên trên và minor number **MKDEV(majorNumber, 0),** **const char \* fmt** chứa tên của device.
* void device\_destroy (struct class \*  class, dev\_t  devt): hàm hủy device được tạo trước đó bởi device\_create, nhận vào tham số **struct class \*  class** là con trỏ trỏ đến device class đã đăng kí và **dev\_t  devt** là character device với major number đã tạo trước đó
* void **class\_destroy**(struct class \*  cls): hàm hủy device class đã khởi tạo, tham số nhận vào là **struct class \*  cls** là class đã đăng kí trước đó qua hàm class\_create
* void get\_random\_bytes(void \*buf, int nbytes ): hàm này sẽ tạo phát sinh ra một số ngẫu nhiên có kích thước **nbytes** và lưu vào trong **buf**
* unsigned long copy\_to\_user ( void \_\_user \* to,const void \* from,unsigned long n): hàm này sẽ truyền giá trị **from** là gía trị của random number đã phát sinh ở trên qua hàm get\_random\_bytes vào **to** với kích thước **n** là kích thước của random number và gởi đến user. Nếu kết quả trả về bằng 0 thì gởi thành công và ngược lại

**Thiết kế kernel module**

Hàm static int \_\_init randomNumber\_driver\_init(void):

* Hàm này sẽ được thực thi khi gọi **module\_init(randomNumber\_driver\_init)**
* Đầu tiên sẽ tiến hành khởi tạo major number động cho device file thông qua việc gọi hàm **register\_chrdev(0, DEVICE\_NUM\_NAME, &fops)**
* Nếu kết quả trả về của major number < 0 sẽ thất bại và kết thúc hàm
* Ngược lại nếu thành công sẽ tiếp tục khởi tạo device class qua hàm **randomNum\_drv.dev\_class\_randomNum=class\_create(THIS\_MODULE,DEVICE\_CLASS\_NAME).** Nếu kết quả trả về là NULL thì khởi tạo thất bại và gọi hàm **unregister\_chrdev(majorNumber, DEVICE\_NAME)** để hủy device number đã cấp trước đó
* Nếu khởi tạo **device class** thành công, ta tiếp tục khởi tạo **character** **device** qua hàm **device\_create(randomNum\_drv.dev\_class\_randomNum,NULL,MKDEV(majorNumber, 0),NULL,DEVICE\_NAME)**.
* Nếu thất bại gọi hàm **class\_destroy(randomNum\_drv.dev\_class\_randomNum)** để hủy bỏ device class đã tạo trước đó. Nếu thành công sẽ thông báo khởi tạo thành công driver.

Hàm static void \_\_exit randomNumber\_driver\_exit(void):

* Hàm này sẽ được thực thi khi gọi **module\_exit(randomNumber\_driver\_exit),** tiến hành hủy device class,device driver và character device trước khi thoát
* Tiến hành hủy bỏ device driver đã đăng kí qua hàm **device\_destroy(randomNum\_drv.dev\_class\_randomNum,MKDEV(majorNumber, 0))**
* Sau đó, hủy bỏ class device bằng **class\_destroy(randomNum\_drv.dev\_class\_randomNum)**
* Sau cùng hủy đăng kí và xóa character devicequa **unregister\_chrdev(majorNumber,DEVICE\_NAME)**
* Hiển thị thông báo giải phóng thành công

Hàm static int **dev\_open**(struct inode \*inodep, struct file \*filep):

* Hàm nhận vào tham số là một con trỏ struct idode\*(để chỉ thao tác thuộc về hệ thống) và struct file \*(để chỉ thac tác thuộc về phía user)
* Mỗi lần có theo tác diễn ra(mỗi lần user yêu cầu thực hiện việc gởi và nhận random number) thì biến môi trường numberOpens sẽ tăng lên 1 (tức là sẽ tính số lần mà user đã yêu cầu truy cập vào device)
* Sau cùng hiện thị thông báo

Hàm static ssize\_t dev\_read(struct file \*filep, char \*buffer, size\_t len, loff\_t \*offset):

* Hàm nhận vào 4 tham số là struct file\* filep(con trỏ trỏ đến file object ,thao tác phía user),char \*buffer (con trỏ trỏ đến buffer chứa data), size\_t len(chứa kích thước buffer), và con trỏ loff\_t\* offset(chứa offset nếu yêu cầu)
* Hàm sẽ tạo ra một số ngẫu nhiên thông qua hàm có sẵn là **get\_random\_bytes(&number, sizeof(number))**
* Sau đó tiến hành copy giá trị number ngẫu nhiên vừa tạo vào buffer và gởi đến user qua hàm **copy\_to\_user(buffer, &number, sizeof(number))**
* Nếu kết quả trả về của hàm trên là 0(tức thành công) sẻ hiển thị thông báo thành công và giá trị số đã gởi đến user
* Ngược lại,hiển thị thông báo thất bại nếu có lỗi xảy ra

Hàm static int dev\_release(struct inode \*inodep, struct file \*filep):

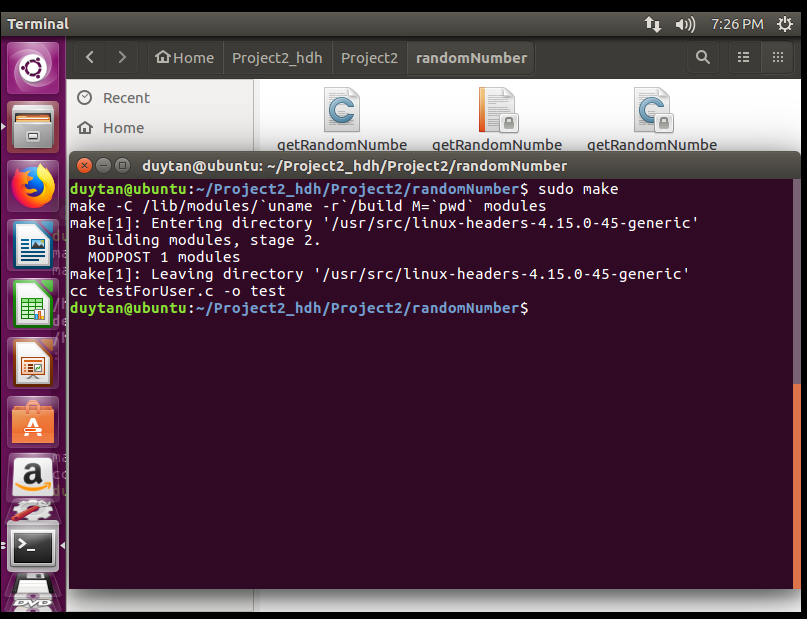
* Tham số nhận vào vẫn là **struct inode**\* **inodep** (thao tác với phía hệ thống),**struct file\*filep(**file thao tác với phía user)
* Hàm này chỉ đơn giản là thực hiện việc hiển thị thông báo đóng device sau khi thực hiện xong các thao tác trên device

**Thiết kế file thực thi phía user**

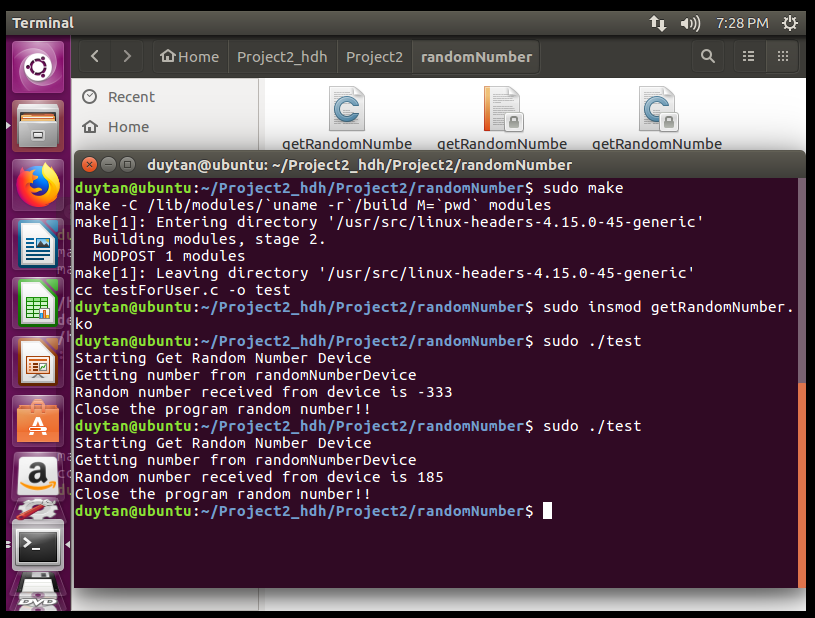
* Thiết bị device tạo ra có tên là **randomNumberDevice** và được chứa trong thư mục **/dev/randomNumberDevice**
* Ta tiến hành mở file chứa device với mục đích read thông qua hàm open("/dev/randomNumberDevice", O\_RDONLY).Kết quả trả về không âm sẽ thành công và ngược lại
* Sau đó đọc giá trị random number qua **hàm ssize\_t read(int fd, void \*buf, size\_t count)** và hiển thị giá trị number nhận được

# **Test case linux kernel**

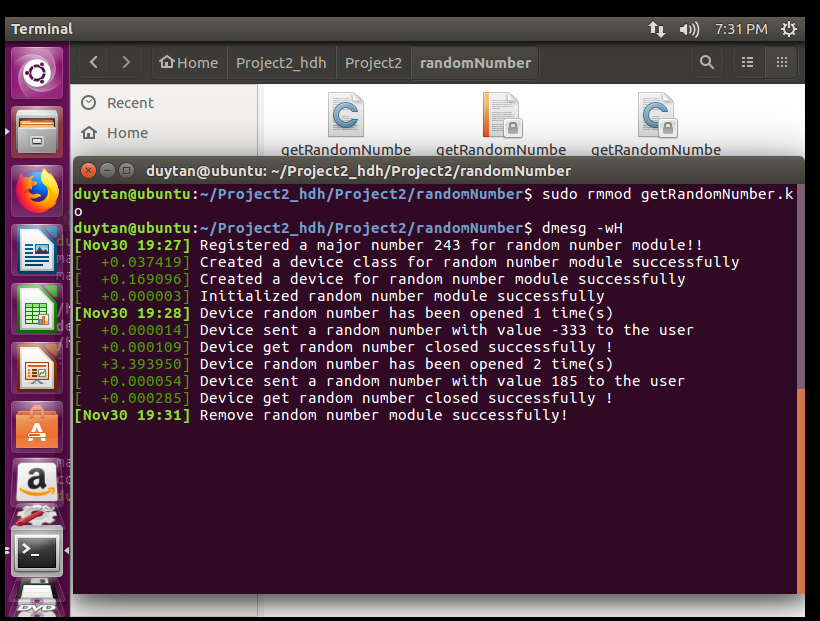
- Biên dịch chương trình:



-Thực thi chương trình phía user:



- Đóng chương trình và hiển thị các thông báo khi đã thực hiện từ đầu lúc khởi tạo đến hiện giờ phía system:



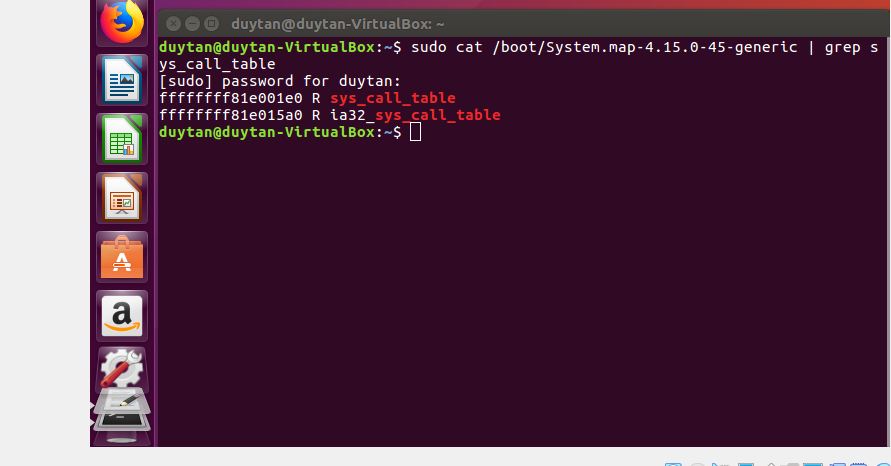
# **Mô tả tổ chức,thiết kế của system call hooking**

1) Cấu trúc:

- Chương trình sẽ gồm file **testForUser.c** để test hook syscall open và hook syscall write

- File **hook\_into\_sys.c** để xây dựng một Loadable Kenel Module để nạp và tháo ra khỏi kernel nhằm để chiếm quyền thực thi kernel mode

-Theo như tài liệu tham khảo, ta có thể hook thủ công bằng cách dụng lệnh **cat/boot/System.map-<version\_kernel> | grep sys\_call\_table** như hình



- Tuy nhiên cách làm trên sẽ không linh hoạt vì khi mang sang xây dựng một kernel khác thì địa chỉ đó sẽ thay đổi.

- Do vậy ta sẽ tiến hành tìm địa tự động tìm địa chỉ tự động cho bảng system call và thực hiện hook đối với 2 syscall open và write theo các bước sau:

+ Tìm tự động địa chỉ của bảng system call và lưu lại

+ Xây dựng 2 hàm hook open và hook write

+ Thay đổi kernel function pointer của system call cần hook thành địa chỉ của hàm hook khi gởi tạo module. Cần có hàm hỗ trợ tắt memory protection

+ Phục hồi lại địa chỉ gốc ban đầu của system call được hook khi tháo module ra, bật lại memory protection.

2) Các tham số,biến môi trường quan trọng:

* unsinged long \*\*sys\_call\_table: chứa địa chỉ của system call table

3) Các hàm quan trọng được sử dụng:

* asmlinkage int ( \*original\_write ) ( unsigned int, const char \_\_user \*, size\_t ) : chứa địa chỉ syscall write và thực thi hàm write tự tạo
* asmlinkage long ( \*original\_open ) (const char \_\_user \*, int, mode\_t ): chứa địa chỉ syscall open và thực thi việc đọc dữ liệu
* asmlinkage int write(unsigned int fd,const char \_\_user \*buf,size\_t count ): hàm thực hiện việc hook vào syscall write
* static void allow\_write(void): tắt memmory protection
* static void disallow\_write(void): bật memory protection trở lại
* static int init\_mod(void): hàm bắt đầu việc hook
* static void exit\_mod(void): hàm kết thúc quá trình hook

**Thiết kế hook open và hook write**

Hàm static int init\_mod(void): Đầu tiên sẽ tiến hành tìm địa chỉ bảng syscall table và lưu lại vào biến môi trường **sys\_call\_table** qua hàm **kallsyms\_lookup\_name("sys\_call\_table") .** Sau đó, tiến hành lưu địa chỉ của syscall open và syscall write lại, lưu vào **original open** và **original write** đã trình bày bên trên. Kế tiếp để tiến hành hook open và hook write ta sẽ tiến hành tắt memory protection thông qua hàm **allow\_write** và **disallow\_write** sau khi hook xong

Hàm static void exit\_mod(void): Sau khi hook xong,trước khi kết thúc chương trình sẽ trả lại địa chỉ cho syscall open và syscall write như ban đầu, bật lại memory protection

Hàm asmlinkage int write(unsigned int fd,const char \_\_user \*buf,size\_t count ) : Hàm này sẽ tiến hành hook vào syscall write nhưng đã được thiết lập để chỉ hiển thị ra những thông tin liên quan đến file user thực thi, nhận vào file thực thi, chuỗi và kích thước chuỗi cần ghi. Kết thúc hàm sẽ hiển thị ra thông tin chuỗi ghi và kích thước chuỗi ghi

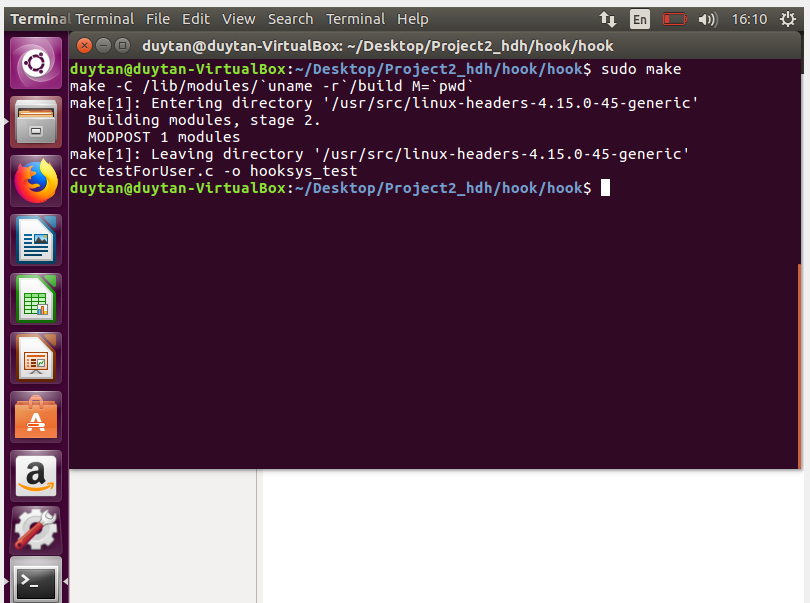
Hàm asmlinkage long open(const char \_\_user \* pathname, int flags, mode\_t mode ) : Hàm này sẽ tiến hành hook vào syscall open nhưng cũng đã được thiết lập để hiển thị ra nhưng thông tin liên quan đến file user thực thi, nhận vào đường dẫn, chế độ thực thi trên file(O\_RDONLY, O\_WRONLY | O\_CREAT | O\_TRUNC). Kết thúc hàm sẽ hiển thị thông báo cho ta biết được tên file đọc, cũng như tên của tiến trình đang thực thi việc đọc file đó.

**Thiết kế file thực thi phía user**

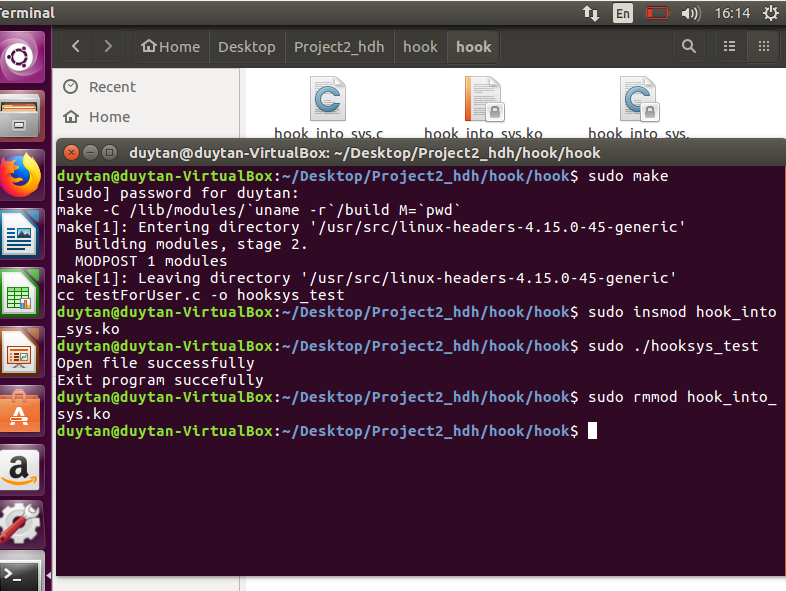
* Tiến hành mở file **test** ra bằng lệnh **open**
* Sau đó ghi dùng hàn **write** để ghi dữ liệu xuống
* Các quá trình mở file và ghi dữ liệu sẽ được xử lí và hiển thị khi gõ dmesg

# **5 Test case syscall hook**

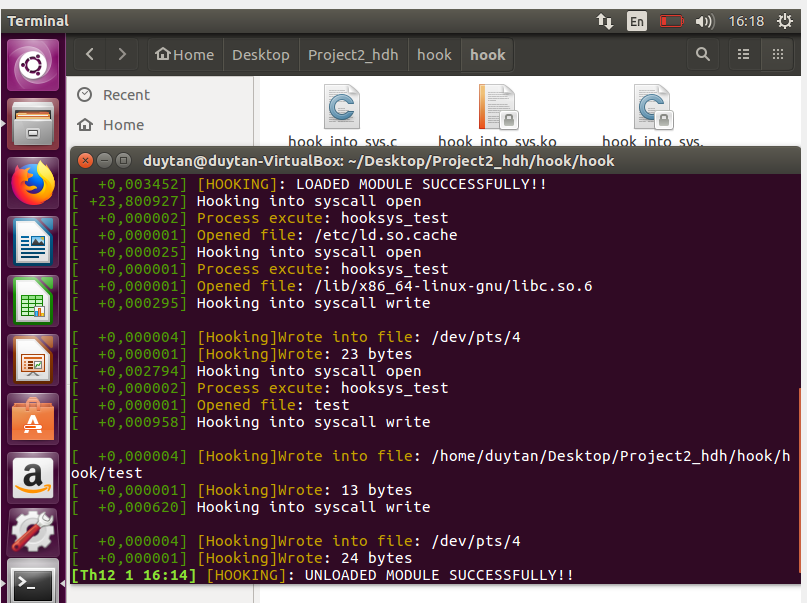
- Biên dịch chương trình



- Cài đặt kernel, thực thi chương trình và gỡ bỏ kernel:



- Hiển thị các thông báo trong dmesg



# **6 Tài liệu tham khảo**

1) <http://derekmolloy.ie/writing-a-linux-kernel-module-part-2-a-character-device/>

2) Tài liệu hường dẫn thực hành HĐH CQ\_2017/33

3) <https://uwnthesis.wordpress.com/2016/12/26/basics-of-making-a-rootkit-from-syscall-to-hook/>

4) <https://stackoverflow.com/>