



BÁO CÁO ĐỒ ÁN

MÔN HỌC: HỆ ĐIỀU HÀNH

CHỦ ĐỀ: MODULE KERNEL LINUX VÀ HOOK SYSTEM CALL

Danh sách thành viên

1. Nguyễn Huỳnh Thảo Nhi - 1712117
2. Nguyễn Anh Thư - 1712177

GVHD:Thầy Lê Giang Thanh
..... Cô Nguyễn Thị Thanh Huyền
Lớp: Hệ điều hành CQ2017/21

Năm học: 2019 – 2020

Mục lục

I.	TỔNG QUAN ĐỒ ÁN:	3
1.	Yêu cầu 1:.....	3
2.	Yêu cầu 2:.....	3
II.	PHÂN CÔNG CÔNG VIỆC:	3
III.	MÔI TRƯỜNG CÀI ĐẶT:	3
IV.	TÌM HIỂU CÁC KHÁI NIỆM:	3
1.	Linux Kernel Module:	3
2.	Pseudo Random Number Generator – PRNG:.....	4
V.	THIẾT KẾ VÀ TỐ CHỨC RANDOM KERNEL MODULE:	4
1.	Khởi tạo macro init() và macro exit() cho module:.....	5
2.	Bổ sung thông tin của module:.....	5
3.	Khởi tạo thông tin lớp thiết bị:	5
4.	Thông tin (<major, minor>) của thiết bị trong lớp này sẽ được khởi tạo:	6
5.	Bổ sung các thao tác với file device:	7
6.	Code các hàm hỗ trợ sinh số ngẫu nhiên:.....	8
VI.	HIỆN THỰC RANDOM KERNEL MODULE:	9
1.	Viết Makefile và chạy make để biên dịch driver tạo ra file .ko:	9
2.	Nạp driver vào nhân hệ thống sử dụng lệnh insmod:	9
3.	Xem thông tin số hiệu major và module đã nạp:.....	9
4.	Xem thông tin về driver:.....	10
5.	Tạo số ngẫu nhiên:.....	10
6.	Gỡ bỏ driver:	11
VII.	THIẾT KẾ VÀ TỐ CHỨC KERNEL MODULE ĐỂ HOOK SYSCALL OPENAT VÀ SYSCALL WRITE:	12
1.	Cài đặt hàm khởi tạo và hàm tháo gỡ module:	12
2.	Cài đặt hàm bật tắt chế độ bảo vệ của “sys_call_table” để có thể thay đổi giá trị trong bảng.....	12
3.	Cài đặt hàm hook syscall openat và hook syscall write	13
VII.	HIỆN THỰC HÓA KERNEL MODULE ĐỂ HOOK SYSCALL OPENAT VÀ SYSCALL WRITE:	13
1.	Viết Makefile và chạy make để biên dịch thành hook.ko.....	14
2.	Gắn module vào bằng câu lệnh sudo insmod hook.ko và gỡ ra bằng sudo rmmod hook.ko	14
3.	Chạy dmesg để xem kết quả.....	15
IX.	ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH:	15

I. TỔNG QUAN ĐO ÁN:

1. Yêu cầu 1:

- 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 kernel space.

- o Viết một module dùng để tạo ra số ngẫu nhiên.
- o Module này sẽ tạo một character device để cho phép các tiến trình ở user space có thể open và read các số ngẫu nhiên.

2. Yêu cầu 2:

- Chương trình hook vào một system call:

- o syscall open \Rightarrow ghi vào dmesg tên tiến trình mở file và tên file được mở.
- o syscall write \Rightarrow ghi vào dmesg tên tiến trình, tên file bị ghi và số byte được ghi.

II. PHÂN CÔNG CÔNG VIỆC:

STT	Công việc	Thành viên thực hiện
1	Thực hiện yêu cầu 1	Nguyễn Huỳnh Thảo Nhi
2	Thực hiện yêu cầu 2	Nguyễn Anh Thư
3	Hoàn thành báo cáo và tổng hợp	Nguyễn Huỳnh Thảo Nhi Nguyễn Anh Thư

III. MÔI TRƯỜNG CÀI ĐẶT:

- Hệ điều hành: Ubuntu 19.04 LTS 64 bit.

IV. TÌM HIỂU CÁC KHÁI NIỆM:

1. 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. Việc thiết kế driver theo kiểu loadable module mang lại 3 lợi ích:

- o Giúp giám sát kernel. Do đó, giám sự lãng phí bộ nhớ và giảm thời gian khởi động hệ thống.
- o Không phải biên dịch lại kernel khi thêm mới driver hoặc khi thay đổi driver.
- o 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.

2. Pseudo Random Number Generator – PRNG:

- Là bộ sinh bit ngẫu nhiên tất định (DRBG), là thuật toán sinh ra chuỗi các số có các thuộc tính gần như thuộc tính của chuỗi số ngẫu nhiên. Chuỗi sinh ra từ bộ sinh số giả ngẫu nhiên không thực sự là ngẫu nhiên, do nó hoàn toàn được xác định từ giá trị khởi đầu, được gọi là nguồn (seed) của nó (mà giá trị này có thể hoàn toàn là ngẫu nhiên). Mặc dù chuỗi gần ngẫu nhiên này gần giống với chuỗi được sinh ra bằng bộ sinh số ngẫu nhiên phần cứng, bộ sinh số giả ngẫu nhiên có vai trò rất quan trọng trong thực tế vì tốc độ trong quá trình tạo số và khả năng tái sử dụng của nó.

3. System call:

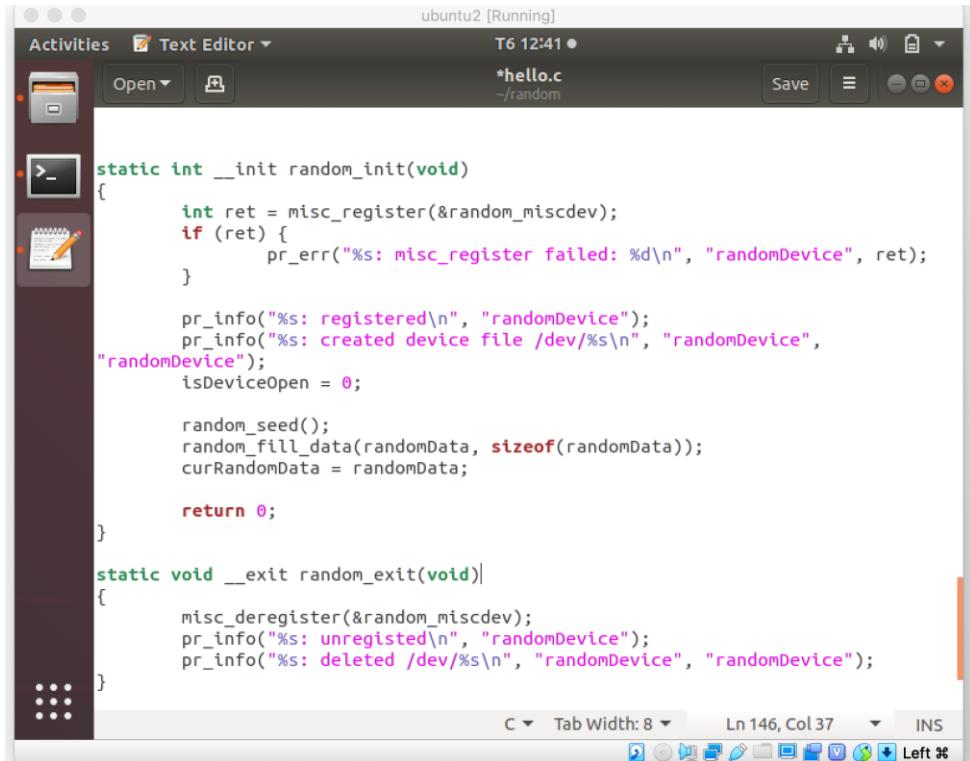
- Là một phương thức mà qua đó một chương trình máy tính (ở userspace) có thể yêu cầu dịch vụ từ kernel của hệ điều hành. Để có thể lấy và sử dụng tài nguyên của hệ thống, chương trình cần phải gọi system call của hệ thống. Tùy thuộc vào hệ điều hành và phiên bản mà chương trình ở user space phải gọi system call tương ứng.

4. Hook:

- Là cách để chặn (hay chuyển hướng) một số lời gọi đến hệ điều hành và đưa chúng đến một đoạn chương trình khác. Ví dụ, hook để nhận sự kiện án một phím trên bàn phím (F1) và đưa đến một hàm xử lý nào đó (hiện lên popup, tắt máy) trước khi sự kiện này được đưa đến CPU và thực hiện thao tác chức năng của nó. Trong bài làm này sẽ trình bày mã nguồn và cách cài đặt/gỡ bỏ một kernel module để hook vào syscall openat và write trong Linux. Từ đó theo dõi được các lời gọi đến những system call này. Do cơ chế tối ưu trong kernel của phiên bản Linux người viết dùng, các thao tác theo dõi (hiển thị tên tiến trình, số byte ghi, ...) phải được thực hiện sau khi lời gọi syscall "thật" được thi hành.

V. THIẾT KẾ VÀ TỐ CHỨC RANDOM KERNEL MODULE:

1. Khởi tạo macro init() và macro exit() cho module:



```
static int __init random_init(void)
{
    int ret = misc_register(&random_miscdev);
    if (ret) {
        pr_err("%s: misc_register failed: %d\n", "randomDevice", ret);
    }

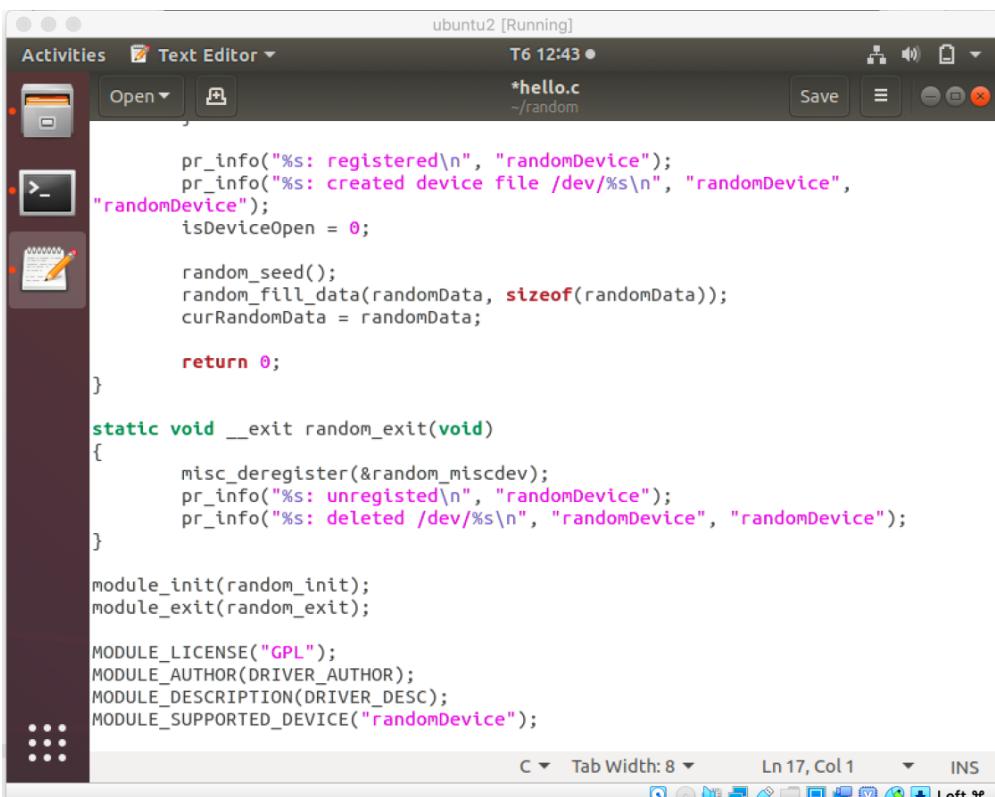
    pr_info("%s: registered\n", "randomDevice");
    pr_info("%s: created device file /dev/%s\n", "randomDevice",
"randomDevice");
    isDeviceOpen = 0;

    random_seed();
    random_fill_data(randomData, sizeof(randomData));
    curRandomData = randomData;

    return 0;
}

static void __exit random_exit(void)
{
    misc_deregister(&random_miscdev);
    pr_info("%s: unregistered\n", "randomDevice");
    pr_info("%s: deleted /dev/%s\n", "randomDevice", "randomDevice");
}
```

2. Bổ sung thông tin của module:



```
pr_info("%s: registered\n", "randomDevice");
pr_info("%s: created device file /dev/%s\n", "randomDevice",
"randomDevice");
isDeviceOpen = 0;

random_seed();
random_fill_data(randomData, sizeof(randomData));
curRandomData = randomData;

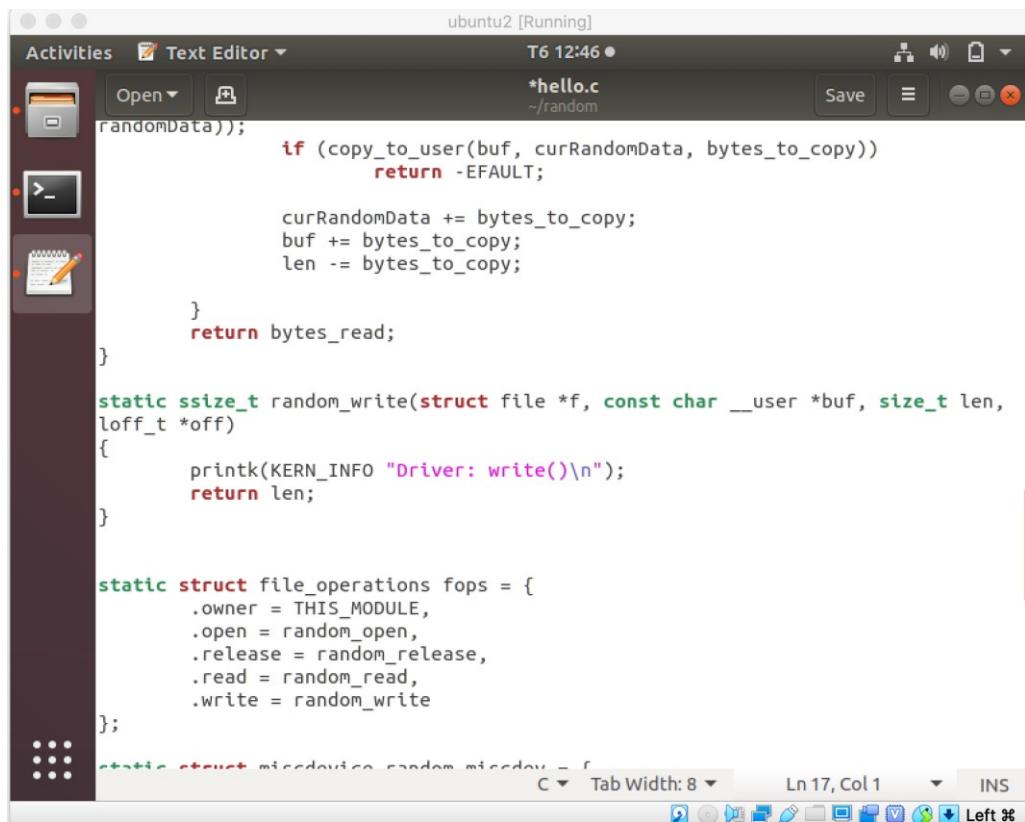
return 0;
}

static void __exit random_exit(void)
{
    misc_deregister(&random_miscdev);
    pr_info("%s: unregistered\n", "randomDevice");
    pr_info("%s: deleted /dev/%s\n", "randomDevice", "randomDevice");

module_init(random_init);
module_exit(random_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(DRIVER_DESC);
MODULE_SUPPORTED_DEVICE("randomDevice");
```

3. Khởi tạo thông tin lớp thiết bị:



```
ubuntu2 [Running]
Activities Text Editor T6 12:46 •
Open ↘ Save ↗
*hello.c
~/random
randomData));
    if (copy_to_user(buf, curRandomData, bytes_to_copy))
        return -EFAULT;

    curRandomData += bytes_to_copy;
    buf += bytes_to_copy;
    len -= bytes_to_copy;

}
return bytes_read;
}

static ssize_t random_write(struct file *f, const char __user *buf, size_t len,
loff_t *off)
{
    printk(KERN_INFO "Driver: write()\n");
    return len;
}

static struct file_operations fops = {
    .owner = THIS_MODULE,
    .open = random_open,
    .release = random_release,
    .read = random_read,
    .write = random_write
};
```

4. Thông tin (<major, minor>) của thiết bị trong lớp này sẽ được khởi tạo:

```

Activities Text Editor
Open Save
*hello.c
~/random
T6 12:50 •

        .write = random_write
};

static struct miscdevice random_miscdev = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = "randomDevice",
    .fops = &fops,
    .mode = S_IRUGO,
};

static int __init random_init(void)
{
    int ret = misc_register(&random_miscdev);
    if (ret) {
        pr_err("%s: misc_register failed: %d\n", "randomDevice", ret);
    }

    pr_info("%s: registered\n", "randomDevice");
    pr_info("%s: created device file /dev/%s\n", "randomDevice",
"randomDevice");
    isDeviceOpen = 0;

    random_seed();
    random_fill_data(randomData, sizeof(randomData));
    curRandomData = randomData;
}

C Tab Width: 8 Ln 127, Col 1 INS

```

5. Bổ sung các thao tác với file device:

```

static int random_open(struct inode *i, struct file *f)
{
    printk(KERN_INFO "Drive: open()\n");
    if (isDeviceOpen) {
        return -EBUSY;
    }
    isDeviceOpen++;
    try_module_get(THIS_MODULE);
    return 0;
}

static int random_release(struct inode *i, struct file *f)
{
    isDeviceOpen--;
    module_put(THIS_MODULE);
    printk(KERN_INFO "Driver: close()\n");
    return 0;
}

```

```

static ssize_t random_write(struct file *f, const char __user *buf, size_t len,
loff_t *off)
{
    printk(KERN_INFO "Driver: write()\n");
    return len;
}

```

```

static ssize_t random_read(struct file *f, char __user *buf, size_t len, loff_t
*off)
{
    printk(KERN_INFO "Driver: read()\n");
    ssize_t bytes_read = len;
    ssize_t bytes_to_copy = 0;

    while (len) {
        if (currRandomData == randomData + sizeof(randomData)) {
            random_fill_data(randomData, sizeof(randomData));
            currRandomData = randomData;
        }

        bytes_to_copy = min(len, sizeof(randomData) - (currRandomData -
randomData));
        if (copy_to_user(buf, currRandomData, bytes_to_copy))
            return -EFAULT;

        currRandomData += bytes_to_copy;
        buf += bytes_to_copy;
        len -= bytes_to_copy;
    }
    return bytes_read;
}

```

6. Code các hàm hỗ trợ sinh số ngẫu nhiên:

```

static void random_seed(void) {
    uint64_t i;
    for (i = 0; i < STATE_BUFFER_LEN; i++) {
        rand_state[i] = i+1;
    }

    rand_state_p = i+1;
}

// random pseudo random number generators
static uint64_t random_prng(void)
{
    uint64_t s0 = rand_state[rand_state_p];
    uint64_t s1 = rand_state[rand_state_p] = (rand_state_p+1)&15;

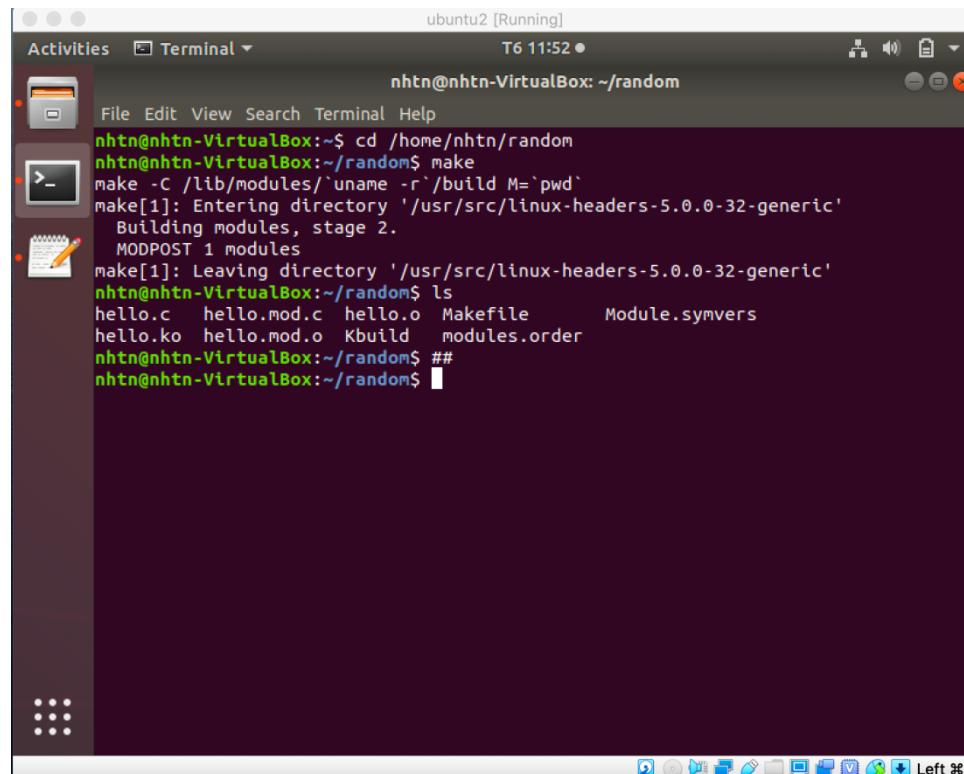
    s1 ^= s1 << 31;
    s1 ^= s1 >> 11;
    s0 ^= s0 >> 30;
    return (rand_state[rand_state_p] = s0^s1) * 1181783497276652981ULL;
}

static void random_fill_data(char *currRandomData, size_t len)
{
    uint64_t randomNumber;
    while (len) {
        randomNumber = random_prng();
        *((uint64_t *) currRandomData) = randomNumber;
        currRandomData += sizeof(uint64_t);
        len -= sizeof(uint64_t);
    }
}

```

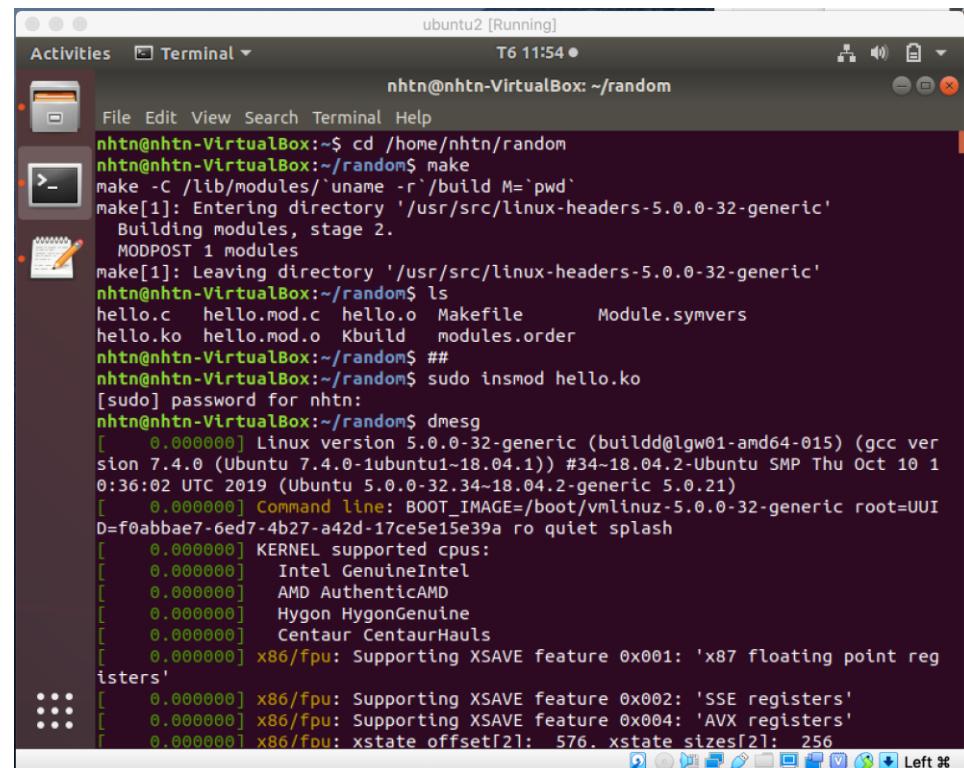
VI. HIỆN THỰC RANDOM KERNEL MODULE:

1. Viết Makefile và chạy make để biên dịch driver tạo ra file .ko:



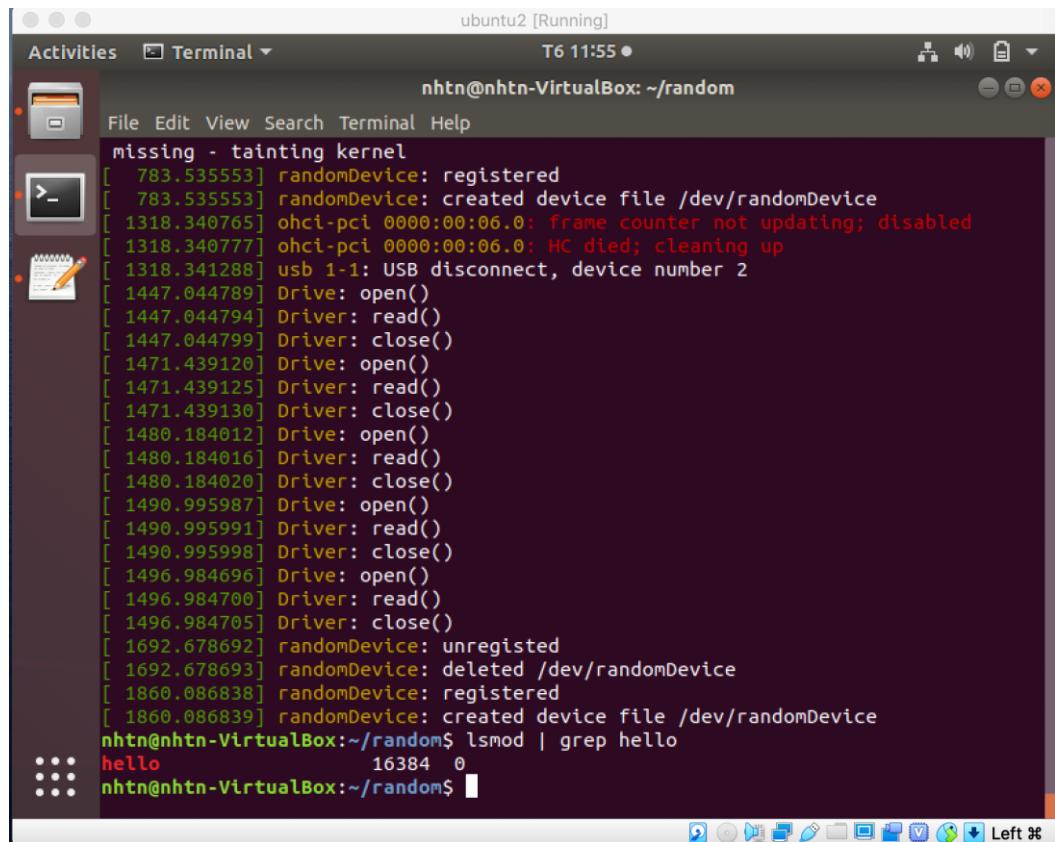
```
ubuntu2 [Running]
Activities Terminal T6 11:52 •
nhtn@nhtn-VirtualBox: ~/random
File Edit View Search Terminal Help
nhtn@nhtn-VirtualBox:~/random$ cd /home/nhtn/random
nhtn@nhtn-VirtualBox:~/random$ make
make -C /lib/modules/`uname -r`/build M=`pwd`
make[1]: Entering directory '/usr/src/linux-headers-5.0.0-32-generic'
  Building modules, stage 2.
    MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-5.0.0-32-generic'
nhtn@nhtn-VirtualBox:~/random$ ls
hello.c hello.mod.c hello.o Makefile      Module.symvers
hello.ko hello.mod.o Kbuild     modules.order
nhtn@nhtn-VirtualBox:~/random$ ##
nhtn@nhtn-VirtualBox:~/random$
```

2. Nạp driver vào nhân hệ thống sử dụng lệnh insmod:



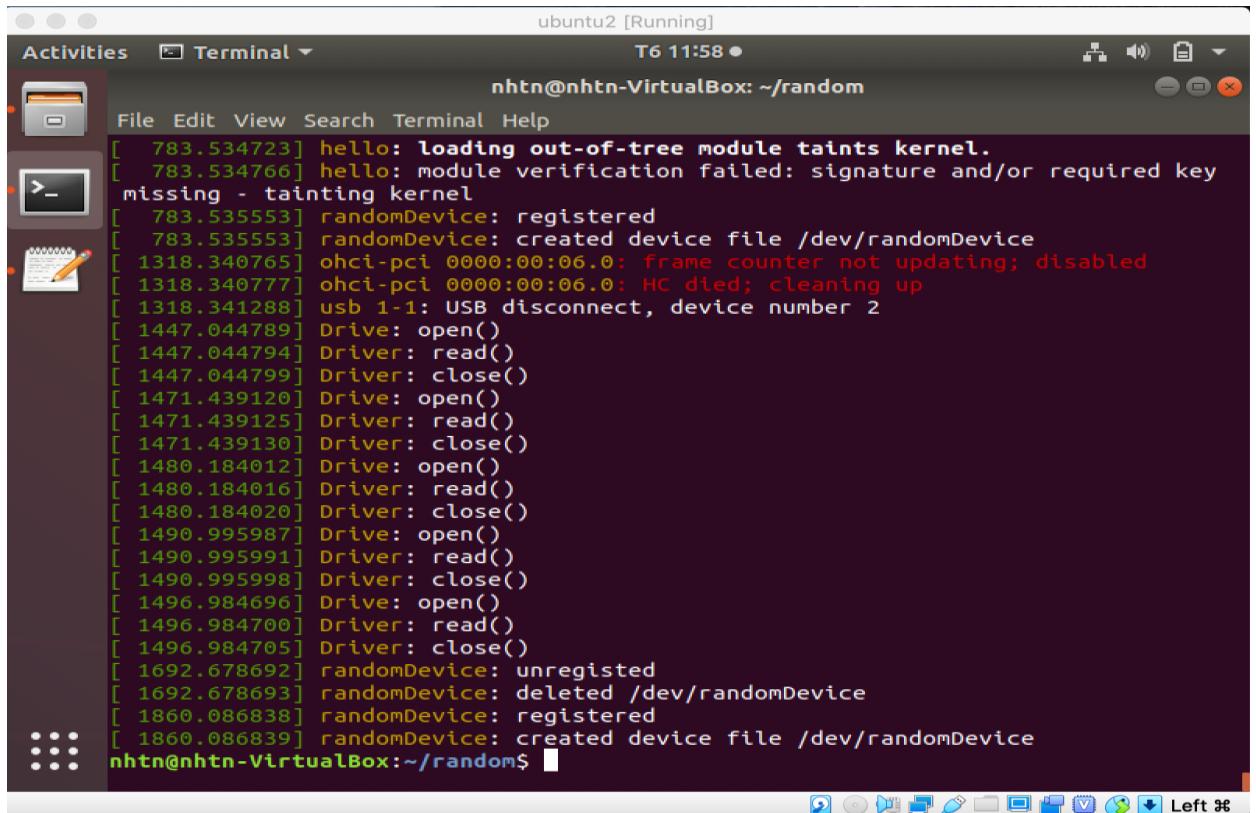
```
ubuntu2 [Running]
Activities Terminal T6 11:54 •
nhtn@nhtn-VirtualBox: ~/random
File Edit View Search Terminal Help
nhtn@nhtn-VirtualBox:~/random$ cd /home/nhtn/random
nhtn@nhtn-VirtualBox:~/random$ make
make -C /lib/modules/`uname -r`/build M=`pwd`
make[1]: Entering directory '/usr/src/linux-headers-5.0.0-32-generic'
  Building modules, stage 2.
    MODPOST 1 modules
make[1]: Leaving directory '/usr/src/linux-headers-5.0.0-32-generic'
nhtn@nhtn-VirtualBox:~/random$ ls
hello.c hello.mod.c hello.o Makefile      Module.symvers
hello.ko hello.mod.o Kbuild     modules.order
nhtn@nhtn-VirtualBox:~/random$ ##
nhtn@nhtn-VirtualBox:~/random$ sudo insmod hello.ko
[sudo] password for nhtn:
nhtn@nhtn-VirtualBox:~/random$ dmesg
[ 0.000000] Linux version 5.0.0-32-generic (buildd@lgw01-amd64-015) (gcc version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #34-18.04.2-Ubuntu SMP Thu Oct 10 1
0:36:02 UTC 2019 (Ubuntu 5.0.0-32.34-18.04.2-generic 5.0.21)
[ 0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.0.0-32-generic root=UUID
D=f0abbae7-6ed7-4b27-a42d-17ce5e15e39a ro quiet splash
[ 0.000000] KERNEL supported cpus:
[ 0.000000]   Intel GenuineIntel
[ 0.000000]   AMD AuthenticAMD
[ 0.000000]   Hygon HygonGenuine
[ 0.000000]   Centaur CentaurHauls
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point reg
isters'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[ 0.000001] x86/fpu: xstate offset[2]: 576. xstate sizes[2]: 256
```

3. Xem thông tin số hiệu major và module đã nạp:



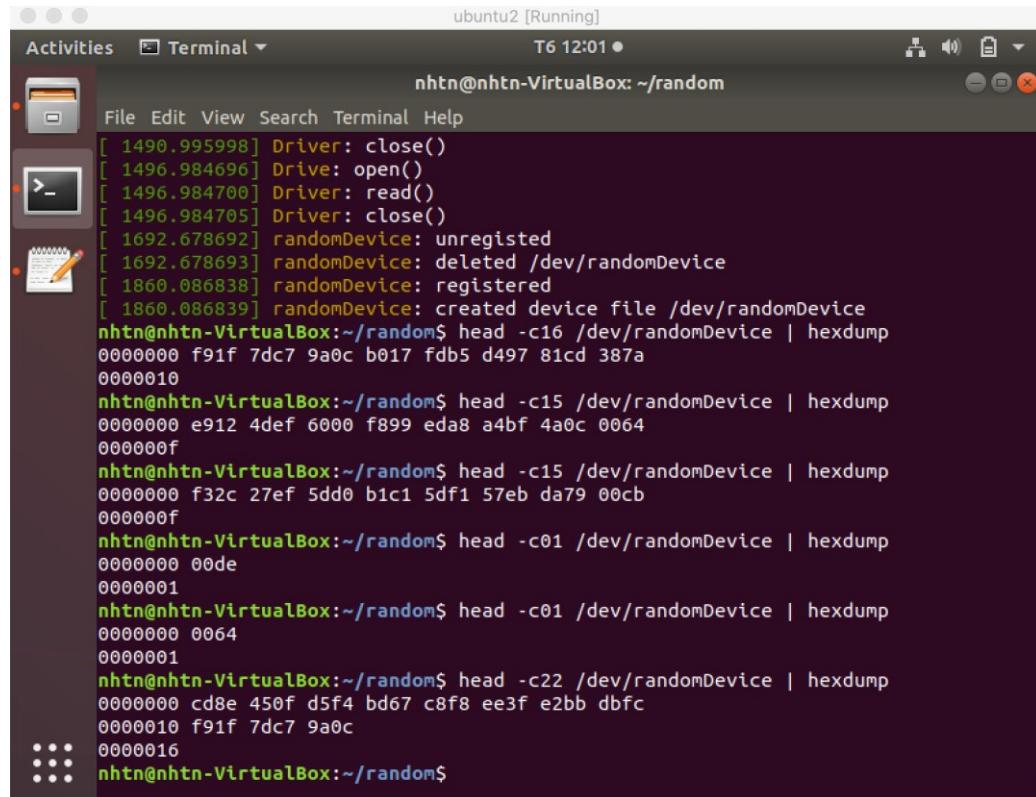
ubuntu2 [Running]
Activities Terminal T6 11:55 ●
nhtn@nhtn-VirtualBox: ~/random
File Edit View Search Terminal Help
missing - tainting kernel
[783.535553] randomDevice: registered
[783.535553] randomDevice: created device file /dev/randomDevice
[1318.340765] ohci-pci 0000:00:06.0: frame counter not updating; disabled
[1318.340777] ohci-pci 0000:00:06.0: HC died; cleaning up
[1318.341288] usb 1-1: USB disconnect, device number 2
[1447.044789] Drive: open()
[1447.044794] Driver: read()
[1447.044799] Driver: close()
[1471.439120] Drive: open()
[1471.439125] Driver: read()
[1471.439130] Driver: close()
[1480.184012] Drive: open()
[1480.184016] Driver: read()
[1480.184020] Driver: close()
[1490.995987] Drive: open()
[1490.995991] Driver: read()
[1490.995998] Driver: close()
[1496.984696] Drive: open()
[1496.984700] Driver: read()
[1496.984705] Driver: close()
[1692.678692] randomDevice: unregistered
[1692.678693] randomDevice: deleted /dev/randomDevice
[1860.086838] randomDevice: registered
[1860.086839] randomDevice: created device file /dev/randomDevice
nhtn@nhtn-VirtualBox:~/random\$ lsmod | grep hello
hello 16384 0
nhtn@nhtn-VirtualBox:~/random\$

4. Xem thông tin về driver:



ubuntu2 [Running]
Activities Terminal T6 11:58 ●
nhtn@nhtn-VirtualBox: ~/random
File Edit View Search Terminal Help
missing - tainting kernel
[783.534723] hello: loading out-of-tree module taints kernel.
[783.534766] hello: module verification failed: signature and/or required key
missing - tainting kernel
[783.535553] randomDevice: registered
[783.535553] randomDevice: created device file /dev/randomDevice
[1318.340765] ohci-pci 0000:00:06.0: frame counter not updating; disabled
[1318.340777] ohci-pci 0000:00:06.0: HC died; cleaning up
[1318.341288] usb 1-1: USB disconnect, device number 2
[1447.044789] Drive: open()
[1447.044794] Driver: read()
[1447.044799] Driver: close()
[1471.439120] Drive: open()
[1471.439125] Driver: read()
[1471.439130] Driver: close()
[1480.184012] Drive: open()
[1480.184016] Driver: read()
[1480.184020] Driver: close()
[1490.995987] Drive: open()
[1490.995991] Driver: read()
[1490.995998] Driver: close()
[1496.984696] Drive: open()
[1496.984700] Driver: read()
[1496.984705] Driver: close()
[1692.678692] randomDevice: unregistered
[1692.678693] randomDevice: deleted /dev/randomDevice
[1860.086838] randomDevice: registered
[1860.086839] randomDevice: created device file /dev/randomDevice
nhtn@nhtn-VirtualBox:~/random\$

5. Tạo số ngẫu nhiên:

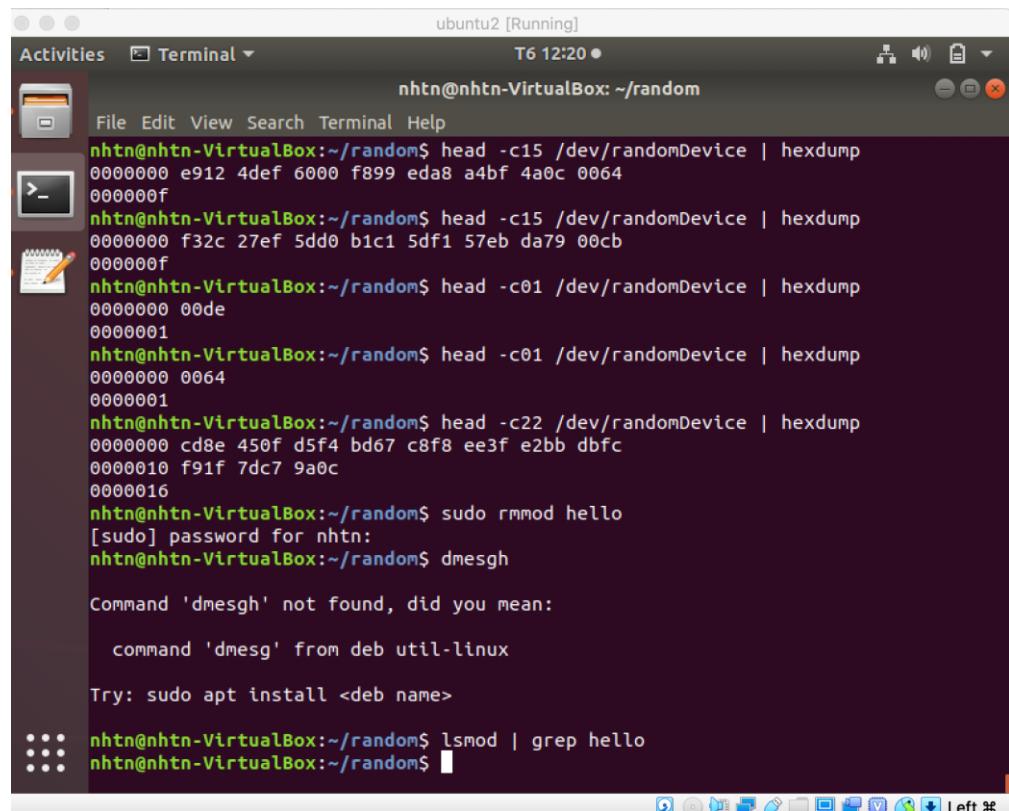


A screenshot of an Ubuntu desktop environment. A terminal window titled "ubuntu2 [Running]" is open, showing the command "head -c16 /dev/randomDevice | hexdump" being run. The terminal output shows several lines of random bytes. The desktop background is dark, and the Unity interface is visible.

```
[ 1490.995998] Driver: close()
[ 1496.984696] Drive: open()
[ 1496.984700] Driver: read()
[ 1496.984705] Driver: close()
[ 1692.678692] randomDevice: unregistered
[ 1692.678693] randomDevice: deleted /dev/randomDevice
[ 1860.086838] randomDevice: registered
[ 1860.086839] randomDevice: created device file /dev/randomDevice
nhtn@nhtn-VirtualBox:~/random$ head -c16 /dev/randomDevice | hexdump
00000000 f91f 7dc7 9a0c b017 fdb5 d497 81cd 387a
00000010
nhtn@nhtn-VirtualBox:~/random$ head -c15 /dev/randomDevice | hexdump
00000000 e912 4def 6000 f899 eda8 a4bf 4a0c 0064
000000f
nhtn@nhtn-VirtualBox:~/random$ head -c15 /dev/randomDevice | hexdump
00000000 f32c 27ef 5dd0 b1c1 5df1 57eb da79 00cb
000000f
nhtn@nhtn-VirtualBox:~/random$ head -c01 /dev/randomDevice | hexdump
00000000 00de
0000001
nhtn@nhtn-VirtualBox:~/random$ head -c01 /dev/randomDevice | hexdump
00000000 0064
0000001
nhtn@nhtn-VirtualBox:~/random$ head -c22 /dev/randomDevice | hexdump
00000000 cd8e 450f d5f4 bd67 c8f8 ee3f e2bb dbfc
00000010 f91f 7dc7 9a0c
0000016
nhtn@nhtn-VirtualBox:~/random$
```

Module sẽ tạo ra character device /dev/randomDevice để khi đọc thiết bị sẽ sinh ra chuỗi bytes random bằng Pseudo Random Number Generator – PRNG.

6. Gỡ bỏ driver:



A screenshot of an Ubuntu desktop environment. A terminal window titled "ubuntu2 [Running]" is open, showing the command "head -c16 /dev/randomDevice | hexdump" being run. The terminal output shows several lines of random bytes. The desktop background is dark, and the Unity interface is visible.

```
nhtn@nhtn-VirtualBox:~/random$ head -c15 /dev/randomDevice | hexdump
00000000 e912 4def 6000 f899 eda8 a4bf 4a0c 0064
000000f
nhtn@nhtn-VirtualBox:~/random$ head -c15 /dev/randomDevice | hexdump
00000000 f32c 27ef 5dd0 b1c1 5df1 57eb da79 00cb
000000f
nhtn@nhtn-VirtualBox:~/random$ head -c01 /dev/randomDevice | hexdump
00000000 00de
0000001
nhtn@nhtn-VirtualBox:~/random$ head -c01 /dev/randomDevice | hexdump
00000000 0064
0000001
nhtn@nhtn-VirtualBox:~/random$ head -c22 /dev/randomDevice | hexdump
00000000 cd8e 450f d5f4 bd67 c8f8 ee3f e2bb dbfc
00000010 f91f 7dc7 9a0c
0000016
nhtn@nhtn-VirtualBox:~/random$ sudo rmmod hello
[sudo] password for nhtn:
nhtn@nhtn-VirtualBox:~/random$ dmesgh
Command 'dmesgh' not found, did you mean:
      command 'dmesg' from deb util-linux
Try: sudo apt install <deb name>
nhtn@nhtn-VirtualBox:~/random$ lsmod | grep hello
nhtn@nhtn-VirtualBox:~/random$
```

VII. THIẾT KẾ VÀ TỔ CHỨC KERNEL MODULE ĐỂ HOOK SYSCALL OPENAT VÀ SYSCALL WRITE:

1. Cài đặt hàm khởi tạo và hàm tháo gỡ module:

```
// FUNCTION WILL EXECUTED WHEN INSERTING KERNEL MODULE
static int __init entry_point(void)
{
    printk(KERN_INFO "MODULE LOADED SUCCESSFULLY\n");
    system_call_table_addr = (void*)kallsyms_lookup_name("sys_call_table");

    original_openat = system_call_table_addr[__NR_openat];
    original_write = system_call_table_addr[__NR_write];

    __flush_tlb();

    make_rw();
    system_call_table_addr[__NR_openat] = hook_openat;
    system_call_table_addr[__NR_write] = hook_write;
    make_ro();

    return 0;
}

//FUNCTION WILL EXECUTED WHEN REMOVING KERNEL MODULE
static void __exit exit_point(void){

    __flush_tlb();

    make_rw();
    system_call_table_addr[__NR_openat] = original_openat;
    system_call_table_addr[__NR_write] = original_write;
    make_ro();

    printk(KERN_INFO "MODULE UNLOADED SUCCESSFULLY\n");
}

module_init(entry_point);
module_exit(exit_point);
```

2. Cài đặt hàm bật tắt chế độ bảo vệ của “sys_call_table” để có thể thay đổi giá trị trong bảng

```
// FUNCTION MAKE PAGE WRITABLE
void make_rw(void)
{
    write_cr0 (read_cr0() & (~ 0x10000));
}

// FUNCTION MAKE PAGE PROTECTED
void make_ro(void)
{
    write_cr0 (read_cr0() | 0x10000);
}
```

3. Cài đặt hàm hook syscall openat và hook syscall write

```
static void **system_call_table_addr;

asmlinkage long (*original_openat) (int dirfd, const char* name, int flags);
asmlinkage long (*original_write) (unsigned int fd, const char __user *buf, size_t count);

// HFUNCTION HOOK SYSCALL OPENAT
asmlinkage long hook_openat (int dirfd, const char* name, int flags)
{
    long result = original_openat(dirfd, name, flags);

    printk(KERN_INFO "HOOK SYSCALL OPEN\n");
    printk(KERN_INFO "CALLING PROCESS (OPEN): %s\n", current->comm);

    int len = strlen(name);
    char* kname = (char*)kmalloc(len + 1, GFP_KERNEL);
    kname[len] = '\0';
    copy_from_user(kname, name, len);
    printk(KERN_INFO "OPENNING FILE: %s\n", name);
    kfree(kname);

    return result;
}

// FUNCTION HOOK SYSCALL WRITE
asmlinkage long hook_write(unsigned int fd, const char __user *buf, size_t count)
{
    long byte_count = original_write(fd, buf, count);

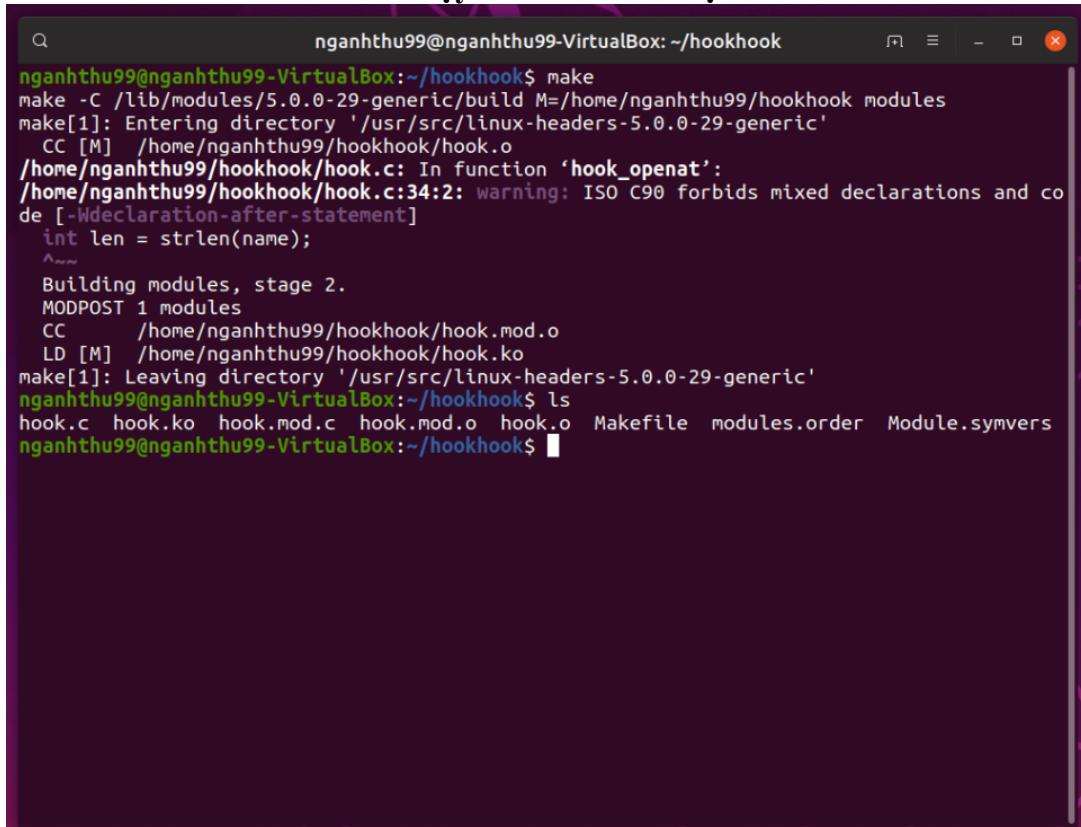
    printk(KERN_INFO "HOOK SYSCALL WRITE\n");
    printk(KERN_INFO "CALLING PROCESS (WRITE): %s\n", current->comm);
    printk(KERN_INFO "BYTE WRITTEN: %ld\n", byte_count);

    return byte_count;
}
```

*Chú thích: khi gọi syscall open, hệ điều hành sẽ direct đến syscall openat. Do đó hook syscall open sẽ không cho ra kết quả, nên thay vào đó, chúng em đã hook syscall openat

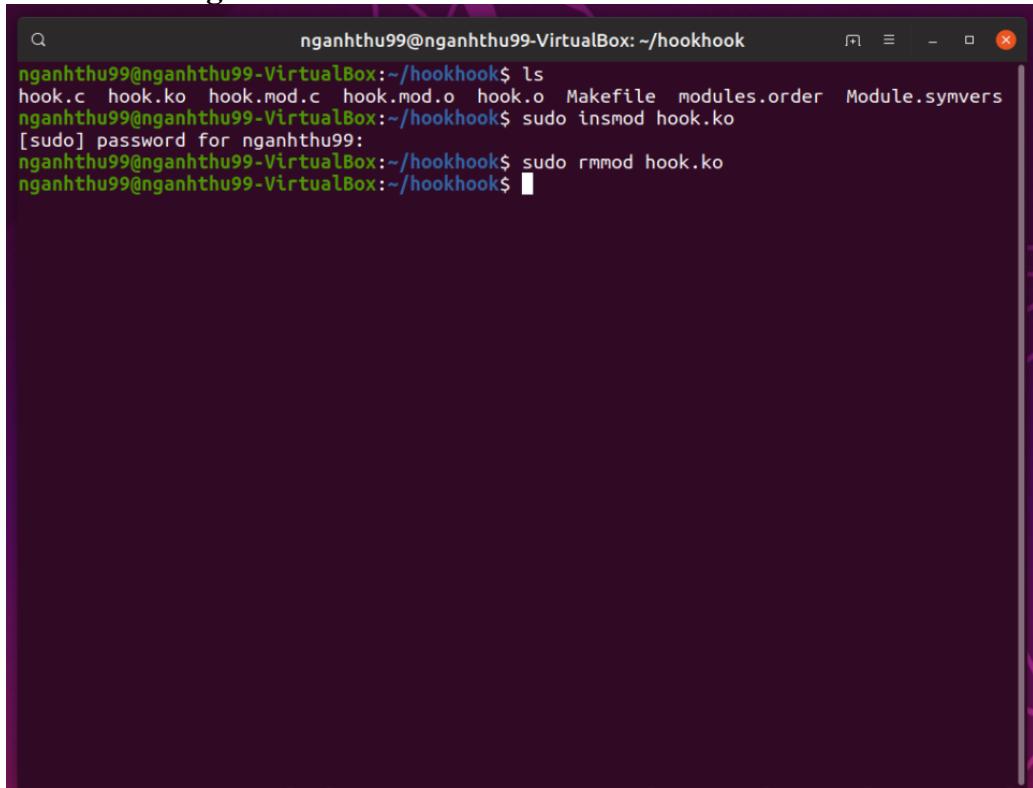
VII. HIỆN THỰC HÓA KERNEL MODULE ĐỂ HOOK SYSCALL OPENAT VÀ SYSCALL WRITE

1. Viết Makefile và chạy make để biên dịch thành hook.ko



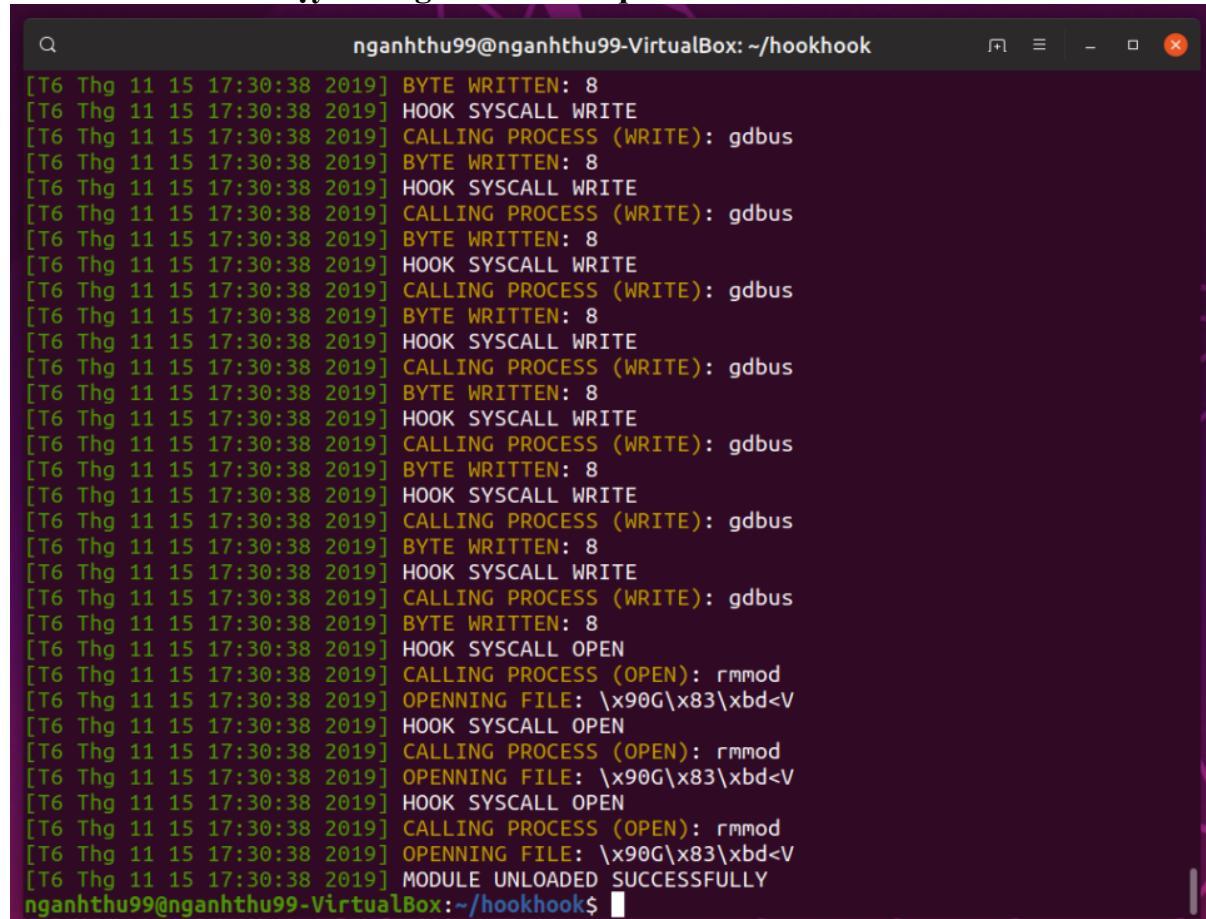
```
nganhthu99@nganhthu99-VirtualBox:~/hookhook$ make
make -C /lib/modules/5.0.0-29-generic/build M=/home/nganhthu99/hookhook modules
make[1]: Entering directory '/usr/src/linux-headers-5.0.0-29-generic'
  CC [M]  /home/nganhthu99/hookhook/hook.o
/home/nganhthu99/hookhook/hook.c: In function 'hook_openat':
/home/nganhthu99/hookhook/hook.c:34:2: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
  int len = strlen(name);
  ^
Building modules, stage 2.
MODPOST 1 modules
CC      /home/nganhthu99/hookhook/hook.mod.o
LD [M]  /home/nganhthu99/hookhook/hook.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.0.0-29-generic'
nganhthu99@nganhthu99-VirtualBox:~/hookhook$ ls
hook.c  hook.ko  hook.mod.c  hook.mod.o  hook.o  Makefile  modules.order  Module.symvers
nganhthu99@nganhthu99-VirtualBox:~/hookhook$
```

2. Gắn module vào bằng câu lệnh sudo insmod hook.ko và gỡ ra bằng sudo rmmod hook.ko



```
nganhthu99@nganhthu99-VirtualBox:~/hookhook$ ls
hook.c  hook.ko  hook.mod.c  hook.mod.o  hook.o  Makefile  modules.order  Module.symvers
nganhthu99@nganhthu99-VirtualBox:~/hookhook$ sudo insmod hook.ko
[sudo] password for nganhthu99:
nganhthu99@nganhthu99-VirtualBox:~/hookhook$ sudo rmmod hook.ko
nganhthu99@nganhthu99-VirtualBox:~/hookhook$
```

3. Chạy dmesg để xem kết quả



```
[T6 Thg 11 15 17:30:38 2019] BYTE WRITTEN: 8
[T6 Thg 11 15 17:30:38 2019] HOOK SYSCALL WRITE
[T6 Thg 11 15 17:30:38 2019] CALLING PROCESS (WRITE): dbus
[T6 Thg 11 15 17:30:38 2019] BYTE WRITTEN: 8
[T6 Thg 11 15 17:30:38 2019] HOOK SYSCALL WRITE
[T6 Thg 11 15 17:30:38 2019] CALLING PROCESS (WRITE): dbus
[T6 Thg 11 15 17:30:38 2019] BYTE WRITTEN: 8
[T6 Thg 11 15 17:30:38 2019] HOOK SYSCALL WRITE
[T6 Thg 11 15 17:30:38 2019] CALLING PROCESS (WRITE): dbus
[T6 Thg 11 15 17:30:38 2019] BYTE WRITTEN: 8
[T6 Thg 11 15 17:30:38 2019] HOOK SYSCALL WRITE
[T6 Thg 11 15 17:30:38 2019] CALLING PROCESS (WRITE): dbus
[T6 Thg 11 15 17:30:38 2019] BYTE WRITTEN: 8
[T6 Thg 11 15 17:30:38 2019] HOOK SYSCALL WRITE
[T6 Thg 11 15 17:30:38 2019] CALLING PROCESS (WRITE): dbus
[T6 Thg 11 15 17:30:38 2019] BYTE WRITTEN: 8
[T6 Thg 11 15 17:30:38 2019] HOOK SYSCALL WRITE
[T6 Thg 11 15 17:30:38 2019] CALLING PROCESS (WRITE): dbus
[T6 Thg 11 15 17:30:38 2019] BYTE WRITTEN: 8
[T6 Thg 11 15 17:30:38 2019] HOOK SYSCALL WRITE
[T6 Thg 11 15 17:30:38 2019] CALLING PROCESS (WRITE): dbus
[T6 Thg 11 15 17:30:38 2019] BYTE WRITTEN: 8
[T6 Thg 11 15 17:30:38 2019] HOOK SYSCALL WRITE
[T6 Thg 11 15 17:30:38 2019] CALLING PROCESS (WRITE): dbus
[T6 Thg 11 15 17:30:38 2019] BYTE WRITTEN: 8
[T6 Thg 11 15 17:30:38 2019] HOOK SYSCALL WRITE
[T6 Thg 11 15 17:30:38 2019] CALLING PROCESS (WRITE): dbus
[T6 Thg 11 15 17:30:38 2019] BYTE WRITTEN: 8
[T6 Thg 11 15 17:30:38 2019] HOOK SYSCALL WRITE
[T6 Thg 11 15 17:30:38 2019] CALLING PROCESS (OPEN): rmmod
[T6 Thg 11 15 17:30:38 2019] OPENNING FILE: \x90G\x83\xbd<V
[T6 Thg 11 15 17:30:38 2019] HOOK SYSCALL OPEN
[T6 Thg 11 15 17:30:38 2019] CALLING PROCESS (OPEN): rmmod
[T6 Thg 11 15 17:30:38 2019] OPENNING FILE: \x90G\x83\xbd<V
[T6 Thg 11 15 17:30:38 2019] HOOK SYSCALL OPEN
[T6 Thg 11 15 17:30:38 2019] CALLING PROCESS (OPEN): rmmod
[T6 Thg 11 15 17:30:38 2019] OPENNING FILE: \x90G\x83\xbd<V
[T6 Thg 11 15 17:30:38 2019] MODULE UNLOADED SUCCESSFULLY
nganhthu99@nganhthu99-VirtualBox:~/hookhook$
```

IX. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH:

CHỨC NĂNG	MỨC ĐỘ HOÀN THÀNH
PHẦN 1: Viết một module dùng để tạo ra số ngẫu nhiên.	100%
Module này sẽ tạo một character device để cho phép các tiến trình ở user space có thể open và read các số ngẫu nhiên.	100%
PHẦN 2: Chương trình hook vào syscall open → ghi vào dmesg tên tiến trình mở file và tên file được mở	100%
Chương trình hook vào syscall write → ghi vào dmesg tên tiến trình, tên file bị ghi và số byte được ghi	95% (Chương trình chưa ghi được tên file bị ghi)
TỔNG KẾT	97%

----- *The End* -----