 TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM

KHOA CÔNG NGHỆ THÔNG TIN

MÔN: **HỆ ĐIỀU HÀNH**

GVHD: **Trần Trung Dung**

**Lê Giang Thanh**

**Lê Quốc Hòa**

Thực hiện: **Huỳnh Kim Ninh**

**1612484**

BÁO CÁO

**Đồ án 2 Systemcall**

TP.HCM, 06/12/2018 7:06 CH

Mục lục

[1. Môi trường thực hiện 3](#_Toc531888772)

[2. Tải mã nguồn linux-kernel (4.14.81) 4](#_Toc531888773)

[3. Cài đặt syscall 5](#_Toc531888774)

[3.1 Chuẩn bị mã nguồn 5](#_Toc531888775)

[3.2 Biên dịch mã nguồn 8](#_Toc531888776)

[3.3 Kiểm tra kết quả 9](#_Toc531888777)

[4. Hook 2 syscall open và write 11](#_Toc531888778)

[4.1 Chuẩn bị mã nguồn 12](#_Toc531888779)

[4.2 Cài đặt module 15](#_Toc531888780)

[4.3 Kết quả 16](#_Toc531888781)

[5. Tài liệu tham khảo 17](#_Toc531888782)

# Môi trường thực hiện

* Hệ điều hành Ubuntu 16.04.5 Xenial – 64 bit
* Phiên bản kernel: [4.14.81](https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.14.86.tar.xz)
* [Ubuntu16.04.5 Xenial Vmware (VMDK) 64-bit](https://sourceforge.net/projects/osboxes/files/v/vm/55-U--u/16.04/16.04.5/1604564.7z/download)
* VMware Player

# Tải mã nguồn linux-kernel (4.14.81)

Truy cập <https://www.kernel.org/> và chọn phiên bản thích hợp tải về (khuyến khích chọn phiên bản mới)

Hoặc tải trực tiếp từ **Terminal** thông qua lệnh

*wget* [*https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.14.85.tar.xz*](https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.14.85.tar.xz)

Giải nén và copy đến thư mục /usr/src/:

sudo tar -xvf linux-4.14.85.tar.xz -C /usr/src/

Ý nghĩa các thông số:

tar — Tar stores and extracts files from a tape or disk archive.

-x — extract files from an archive

-v — requested using the –verbose option, when extracting archives

-f — file archive; use archive file or device archive

-C — extract to the directory specified after it.(in this case /usr/src/)

# Cài đặt syscall

Trước khi bắt đầu cài đặt systemcall vào hệ thống, ta cần cài sẵn 1 số package hỗ trợ:

* libncurses5-dev
* bison
* flex
* libssl-dev

Ta sẽ tiến hành cài tất cả các package này thông qua lệnh sau:

sudo apt-get install gcc  
sudo apt-get install libncurses5-dev  
sudo apt-get install bison  
sudo apt-get install flex  
sudo apt-get install libssl-dev  
sudo apt-get install libelf-dev  
sudo apt-get update  
sudo apt-get upgrade

## Chuẩn bị mã nguồn

Tạo 1 thư mục có tên pnameid để chứa mã nguồn cho các file **pnameid.c,** **pnameid.h**, và **Makefile**

Nội dung các file mã nguồn như sau:

***File pnameid.h*:**

asmlinkage long sys\_pnametoid(char\* name);

asmlinkage long sys\_pidtoname(int pid, char\* buf, int len);

***File pnameid.c:***

#include <linux/syscalls.h>

#include <linux/kernel.h>

#include <linux/sched.h>

#include <linux/init.h>

#include <linux/string.h>

#include "pnameid.h"

asmlinkage long sys\_pnametoid(char\* name){

char buf[32];

strncpy\_from\_user(buf, name, sizeof(buf));

struct task\_struct \*task;

int pid = -1;

for\_each\_process(task){

if(strcmp(task->comm,buf) == 0){

pid = task->pid;

}

}

return (long)pid;

}

asmlinkage long sys\_pidtoname(int pid, char\* buf, int len){

struct task\_struct \*task;

int n = -1;

for\_each\_process(task){

if(task->pid == pid){

copy\_to\_user(buf, task->comm, len);

n = strlen(task->comm);

}

}

if (n == -1)

return -1;

if (len >= n)

return 0;

if (len < n)

return n;

}

***File Makefile:***

obj-y := pnameid.o

Thêm vào file Makefile của source kernel (**/usr/src/linux-4.14.81/Makefile**) thư mục pnameid/

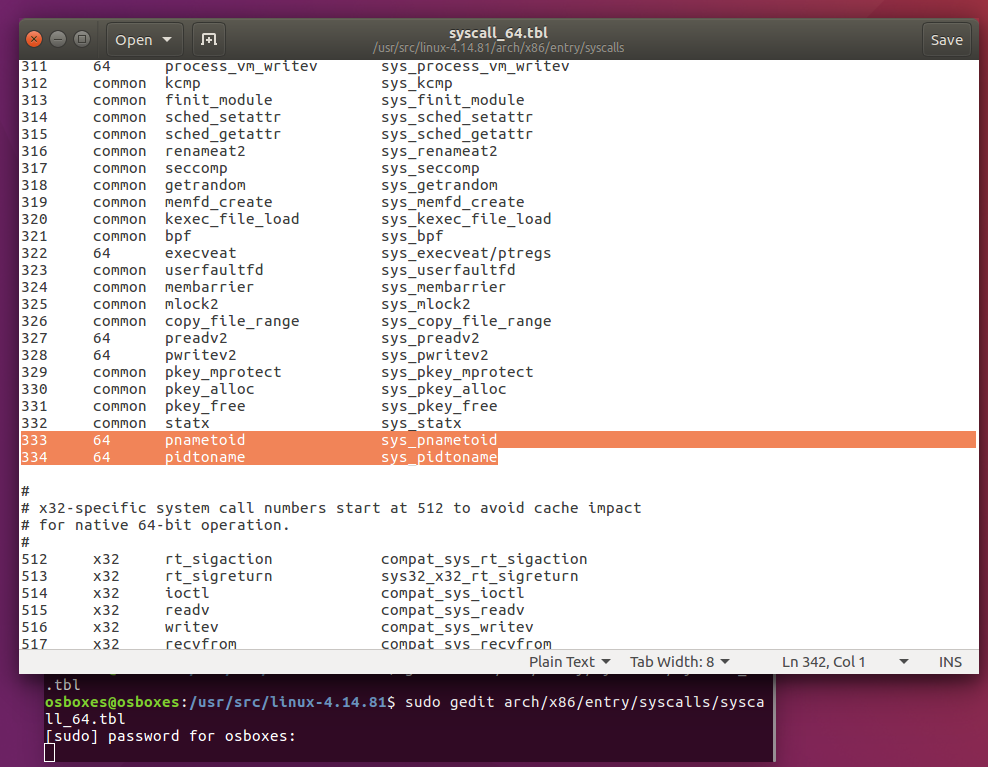
core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ **pnameid/**

Thêm hai system call mới vào bảng system call tại

**/usr/src/linux-4.14.81/arch/x86/entry/syscalls/syscall\_64.tbl**

333 64 pnametoid sys\_pnametoid

334 64 pidtoname sys\_pidtoname



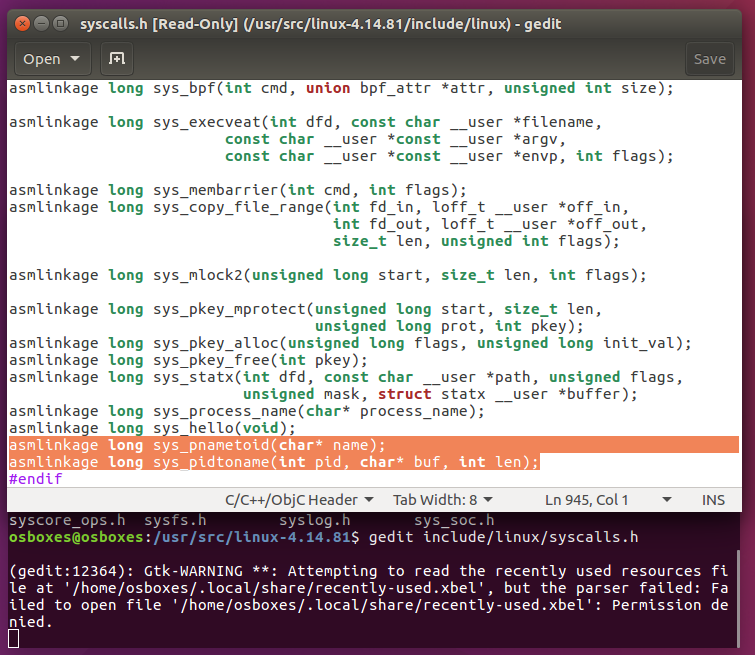
*Thêm hai syscall sys\_pnametoid và sys\_pidtoname vào bảng syscall\_64.tbl*

Tiến hành thêm prototype của 2 syscall vào file syscalls.h tại

**/usr/src/linux-4.14.81/include/linux/syscalls.h**

asmlinkage long sys\_pnametoid(char\* name);

asmlinkage long sys\_pidtoname(int pid, char\* buf, int len);



*Thêm khai báo hàm cài đặt vào syscalls.h*

## Biên dịch mã nguồn

Ngay tại thư mục linux-4.14.81/, ta thực hiện các lệnh sau để build mới kernel, quá trình này có thể kéo dài **2h** và tốn chừng **40GB ổ cứng**

sudo make menuconfig

sudo make

sudo make modules\_install install

Sau khi cài đặt hoàn tất, ta cần restart máy lại để cập nhật bản kernel vừa build

shutdown -r now

uname -r

## Kiểm tra kết quả

Ta sẽ cần file mã nguồn này để kiểm tra hai syscall pnametoid và pidtoname

***File testpnameid.c***

#include <stdio.h>

#include <linux/kernel.h>

#include <sys/syscall.h>

#include <unistd.h>

#include <string.h>

#define MAX\_SIZE 32

int main(){

// Kiểm tra syscall pnametoid

char name[32];

puts("Enter process to find");

scanf("%s",name);

long int status = syscall(333, name);

printf("sys\_pnametoid returned %ld\n", status);

// Kiểm tra syscall pidtoname

int pid;

char buff[MAX\_SIZE];

puts("Enter process ID: ");

scanf("%d", &pid);

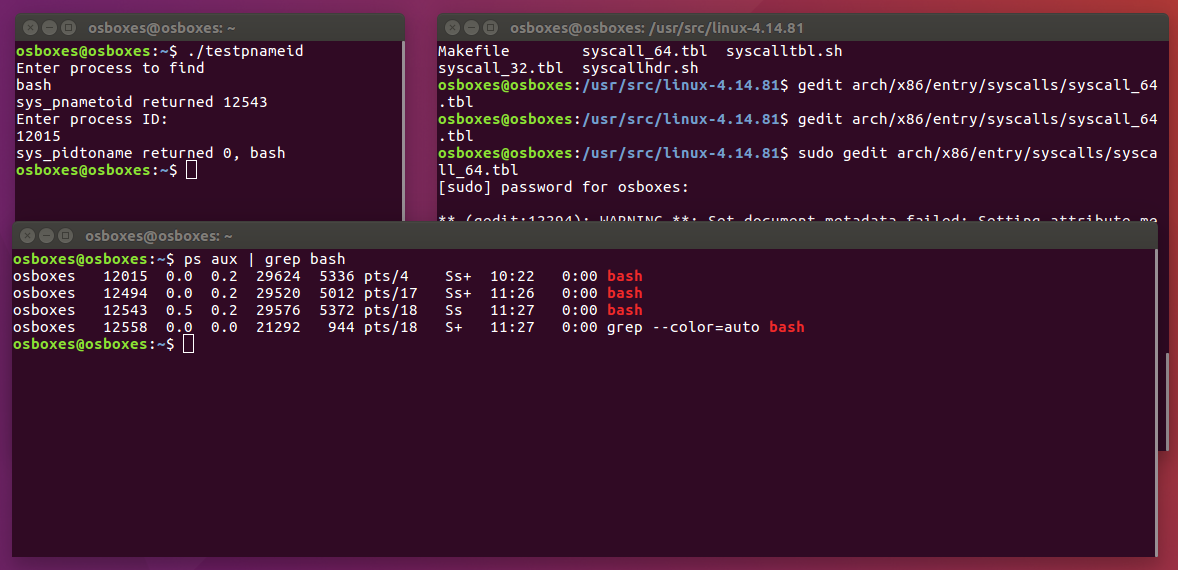
long int status334 = syscall(334, pid, buff, MAX\_SIZE);

printf("sys\_pidtoname returned %ld, %s\n", status334, buff);

return 0;

}

Chạy chương trình lên và nhập vào bash ta sẽ được kết quả như sau:

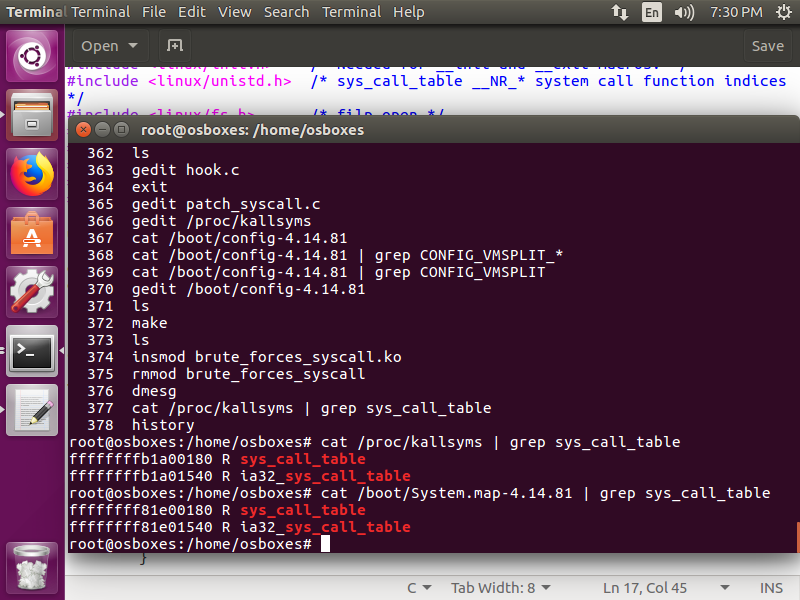


Syscall pnametoid nhận vào bash và trả ra PID cuối cùng được tìm thấy 12543

Ngược lại, syscall pidtoname nhận vào PID 12015 (cũng là 1 cái bash khác) và trả ra bash

# Hook 2 syscall open và write

Đầu tiên ta cần phải lấy được địa chỉ của bảng syscall sys\_call\_table trong hệ điều hành. Có 2 nơi có thể tìm thấy được cái địa chỉ này: **/boot/System.map-4.14.81** và **/proc/kallsyms**. Tùy phiên bản kernel đang sử dụng mà ta sẽ chọn file phù hợp. (Với các bản **4.x trở lên** thì địa chỉ thật sự nằm trong **/proc/kallsyms** )



*Hai địa chỉ khác nhau ở 2 file khác nhau của bảng sys\_call\_table*

Việc lấy địa chỉ của bảng **sys\_call\_table** có thể thông qua code cứng hoặc là brute force vùng nhớ ở kernel để lấy. Tuy nhiên để đơn giản thì chúng ta chỉ code cứng lưu lại **0xffffffffb1a00180** là được. *(Lưu ý: mỗi lần restart lại máy thì địa chỉ trên sẽ* ***thay đổi*** *nên ta cũng sẽ cần* ***cập nhật thủ công*** *lại nếu có restart máy ảo)*

Để có thể lấy được tên process đang chạy khi gọi đến syscall open(), write(), ta sẽ sử dụng đến macro current. Đây là 1 macro đặc biệt trỏ đến process đang thực thi bên trong kernel. Nó chứa đầy đủ các thông tin như ID, tên,…

Đối với syscall write(), ta sẽ cần viết thêm 1 hàm để lấy được tên file đang mở thông qua file descriptor (lấy từ tham số thứ nhất của write) và files\_struct (lấy từ current->files). Hàm này là hàm getPathName() bên dưới.

## Chuẩn bị mã nguồn

Tạo thư mục hooking để chứa mã nguồn cần thiết: **hook\_open.c**, **hook\_write.c**, **Makefile**

Lưu ý là chúng ta hoàn toàn có thể gộp chung thành 1 file hook.c duy nhất. Tuy nhiên để cho đơn giản, tránh việc sau khi hook 2 system call in thông báo lộn xộn thì ta sẽ hook lần lượt 2 system call này.

**File hook\_open.c, hook\_write.c**:

#include <linux/module.h> /\* Needed by all kernel modules \*/

#include <linux/kernel.h> /\* Needed for loglevels (KERN\_WARNING, KERN\_EMERG, KERN\_INFO, etc.) \*/

#include <linux/init.h> /\* Needed for \_\_init and \_\_exit macros. \*/

#include <linux/unistd.h> /\* sys\_call\_table \_\_NR\_\* system call function indices \*/

#include <linux/fs.h> /\* filp\_open \*/

#include <asm/paravirt.h> /\* write\_cr0 \*/

#include <linux/sched.h> /\* need for current\_id \*/

#include <linux/syscalls.h>

#include <linux/string.h>

#include <linux/fdtable.h> /\*need for files\_struct: current\_files\*/

unsigned long \*syscall\_table = (unsigned long \*)0xffffffffb1a00180;

char \*tmp = NULL;

char \*pathname = NULL;

asmlinkage int (\*original\_write)(unsigned int, const char \_\_user \*, size\_t);

asmlinkage int (\*original\_open)(const char \_\_user \*, int);

int getPathName(unsigned int fd,struct files\_struct \*files){

struct file \*file;

struct path \*path;

spin\_lock(&files->file\_lock);

file = fcheck\_files(files, fd);

if (!file) {

spin\_unlock(&files->file\_lock);

return -ENOENT;

}

path = &file->f\_path;

path\_get(path);

spin\_unlock(&files->file\_lock);

tmp = (char \*)\_\_get\_free\_page(GFP\_KERNEL);

if (!tmp) {

path\_put(path);

return -ENOMEM;

}

pathname = d\_path(path, tmp, PAGE\_SIZE);

path\_put(path);

if (IS\_ERR(pathname)) {

free\_page((unsigned long)tmp);

return PTR\_ERR(pathname);

}

/\* do something here with pathname \*/

return 0;

}

asmlinkage int new\_open(const char \_\_user \*pathname, int flags) {

printk(KERN\_INFO "[+] open() hooked.");

char buf[32];

strncpy\_from\_user(buf, pathname, sizeof(buf));

printk(KERN\_INFO "Process name: %s", current->comm);

printk(KERN\_INFO "File open: %s", buf);

printk(KERN\_INFO " ");

return original\_open(pathname, flags);

}

asmlinkage int new\_write (unsigned int x, const char \_\_user \*y, size\_t size) {

printk(KERN\_INFO "[+] write() hooked.");

char buf[32];

int ret = getPathName(x, current->files);

int writtenByte;

strncpy\_from\_user(buf, y, sizeof(buf));

printk(KERN\_INFO "Process name: %s", current->comm);

if (ret == 0)

{

printk(KERN\_INFO "File name written: %s", pathname);

}

writtenByte = original\_write(x, y, size);

printk(KERN\_INFO "Written byte: %d", writtenByte);

printk(KERN\_INFO " ");

return writtenByte;

}

static int \_\_init onload(void) {

printk(KERN\_WARNING "Hello world!\n");

printk(KERN\_INFO "Syscall table address: %p\n", syscall\_table);

if (syscall\_table != NULL) {

write\_cr0 (read\_cr0 () & (~ 0x10000));

//original\_write = (void \*)syscall\_table[\_\_NR\_write];

//syscall\_table[\_\_NR\_write] = &new\_write;

original\_open = (void \*)syscall\_table[\_\_NR\_open];

syscall\_table[\_\_NR\_open] = &new\_open;

write\_cr0 (read\_cr0 () | 0x10000);

printk(KERN\_INFO "[+] onload: sys\_call\_table hooked\n");

printk(KERN\_INFO " ");

} else {

printk(KERN\_INFO "[-] onload: syscall\_table is NULL\n");

}

/\*

\* A non 0 return means init\_module failed; module can't be loaded.

\*/

return 0;

}

static void \_\_exit onunload(void) {

if (syscall\_table != NULL) {

write\_cr0 (read\_cr0 () & (~ 0x10000));

//syscall\_table[\_\_NR\_write] = original\_write;

syscall\_table[\_\_NR\_open] = original\_open;

write\_cr0 (read\_cr0 () | 0x10000);

free\_page((unsigned long)tmp);

printk(KERN\_INFO "[+] onunload: sys\_call\_table unhooked\n");

} else {

printk(KERN\_INFO "[-] onunload: syscall\_table is NULL\n");

}

printk(KERN\_INFO "Goodbye world!\n");

printk(KERN\_INFO " ");

}

module\_init(onload);

module\_exit(onunload);

**Makefile:**

obj-m += hook\_open.o

obj-m += hook\_write.o

all:

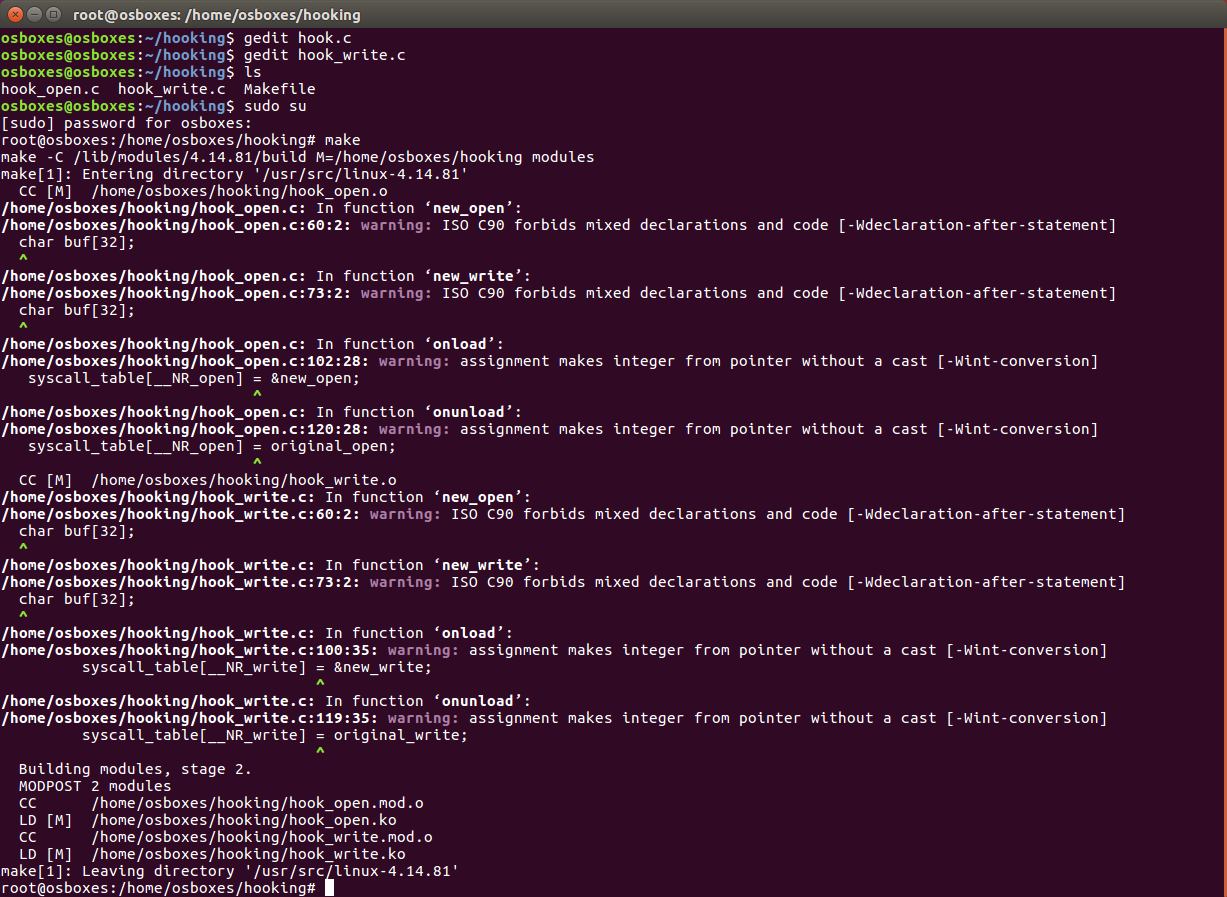
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

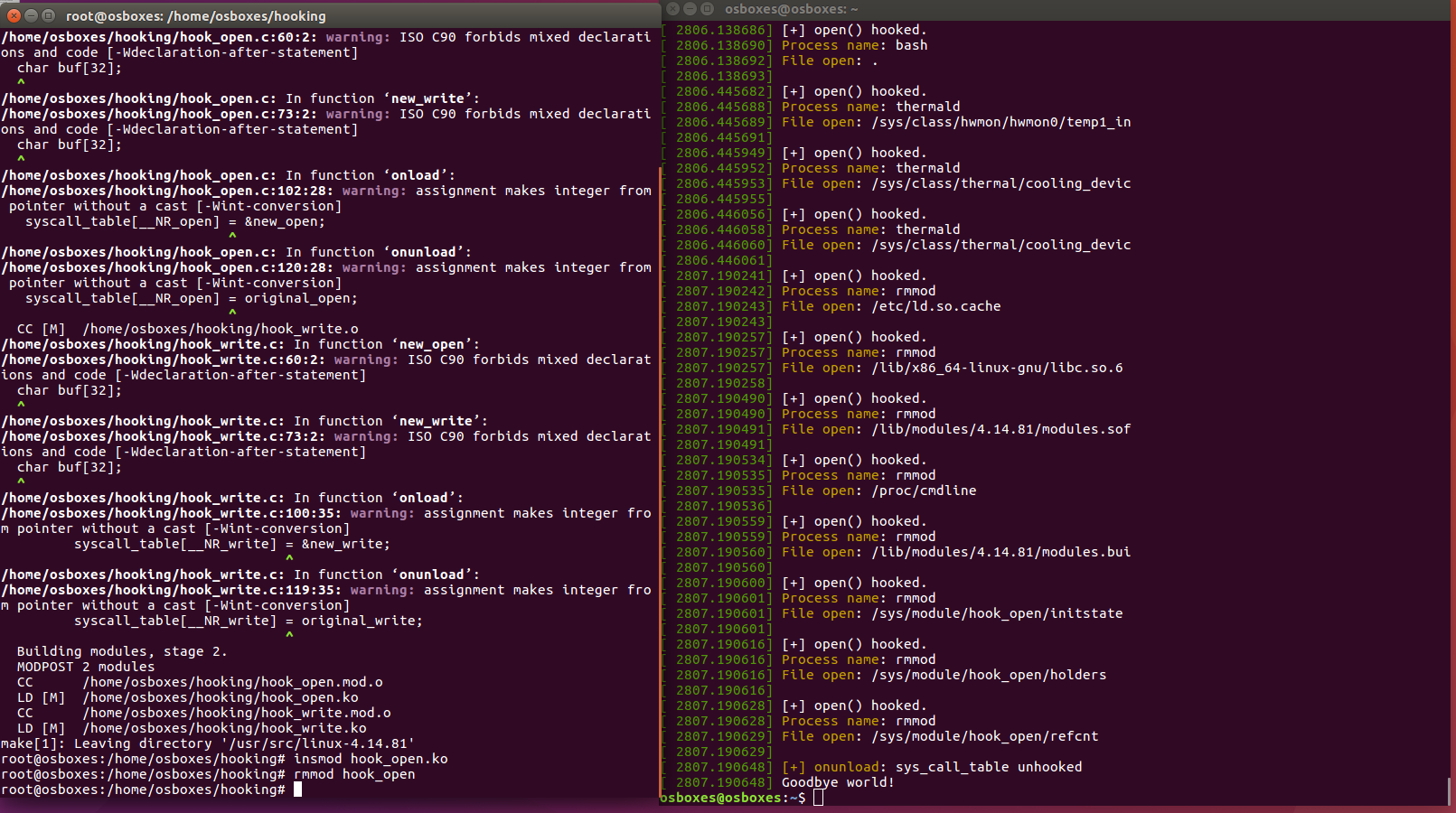
## Cài đặt module

Ta tiến hành cài đặt module như sau:

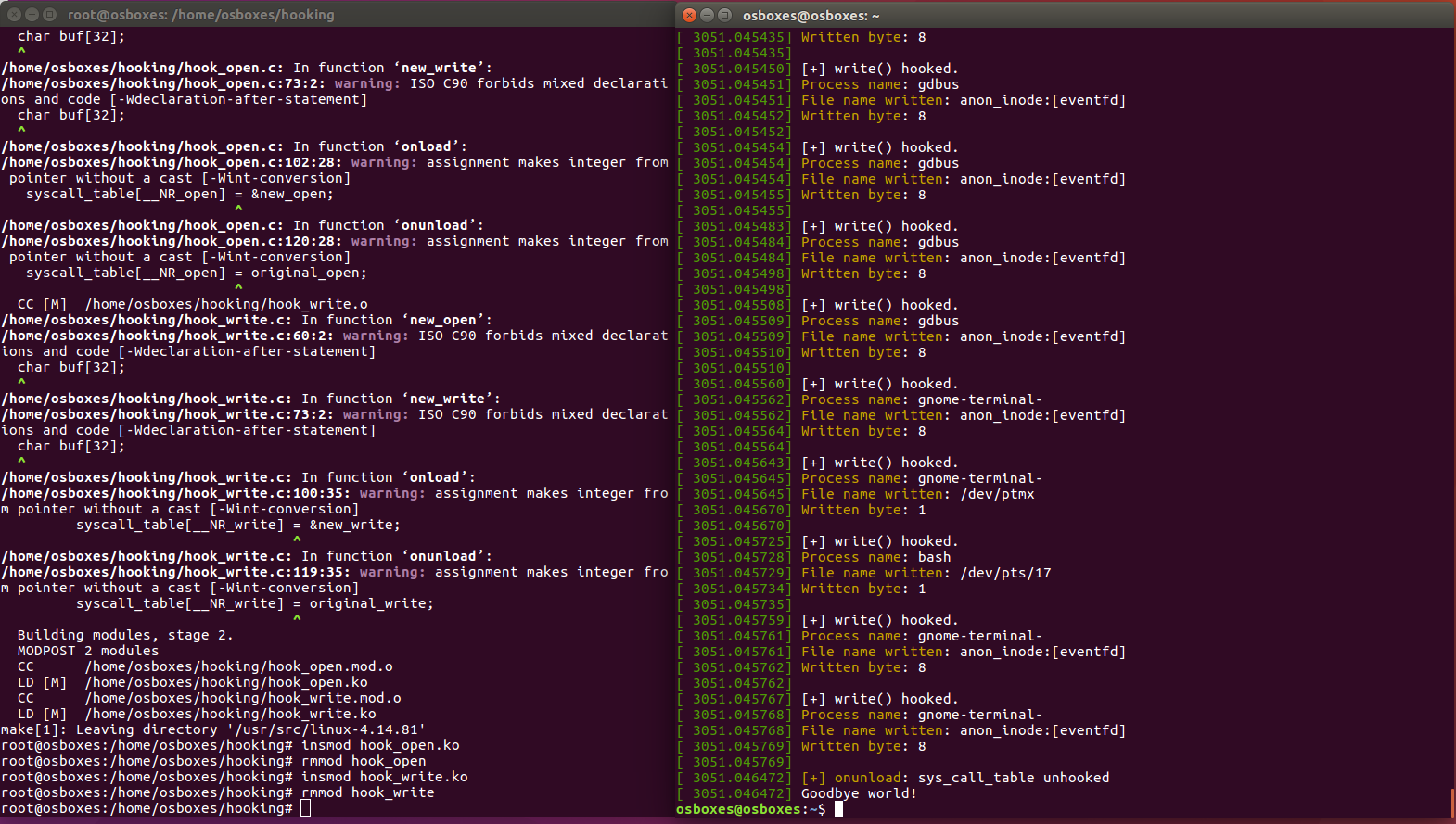


## Kết quả

Cài đặt module hook\_open và xem kết quả



Cài đặt module hook\_write và xem kết quả



# Tài liệu tham khảo

* [Adding a Hello World System Call to Linux Kernel](https://medium.com/anubhav-shrimal/adding-a-hello-world-system-call-to-linux-kernel-dad32875872) (Cài syscall trên ubuntu 16.04)
* [Syscall table hacking toy-example does not work on 64 bit](https://stackoverflow.com/questions/45769966/syscall-table-hacking-toy-example-does-not-work-on-64-bit) (Fix lỗi Paging request)
* [Basics of Making a Rootkit: From syscall to hook!](https://d0hnuts.wordpress.com/2016/12/21/basics-of-making-a-rootkit-from-syscall-to-hook/)
* [How can I get a filename from a file descriptor inside a kernel module?](https://stackoverflow.com/questions/8250078/how-can-i-get-a-filename-from-a-file-descriptor-inside-a-kernel-module) (Lấy đường dẫn file từ file descriptor)
* [how does current->pid work for linux?](https://stackoverflow.com/questions/10838342/how-does-current-pid-work-for-linux) (Cách sử dụng macro current)