

BÁO CÁO ĐỒ ÁN HỆ ĐIỀU HÀNH



Tên đồ án: SYSTEMCALL AND HOOK IN UBUNTU

I. Giới thiệu sơ lược đồ án

• Danh sách thành viên:

1. Nguyễn Tất Hưng – 18127104
2. Nguyễn Minh Đức – 18127081
3. Nguyễn Hoàng Anh Tú - 18127243

• Mô tả sơ lược:

✓ Cài đặt các syscall dưới đây:

• **pnametoid (char *name)**

- Syscall này sẽ nhận vào name và trả về pid nếu tìm thấy và trả về -1 nếu không tìm thấy

• **pidtoname (int pid, char* buf, int len)**

- Syscall này sẽ nhận vào pid, ghi process name vào trong biến buff với max len là len – 1 phần tử cuối cùng sẽ tự động thêm NULL Giá trị trả về là -1 nếu lỗi, 0 nếu len buffer truyền vào lớn hơn len của process name, và n với n là độ dài thật sự của process name, trong trường hợp len buffer chuyển vào nhỏ hơn len của process name.

✓ Hook vào 2 syscall có sẵn dưới đây:

- **syscall open** => ghi vào dmesg tên tiến trình mở file và tên file được mở
- **syscall write** => ghi vào dmesg tên tiến trình, tên file bị ghi và số byte được ghi

II. Các bước thực hiện của từng yêu cầu và giải thích source code và cách thức thực hiện:

1. Các cài đặt cơ bản cho Ubuntu trước khi viết syscall:

- Phiên bản Ubuntu hiện tại đang sử dụng là bản **3.13.0-24-generic**

```
nguyentathung943@ubuntu: ~/Desktop
nguyentathung943@ubuntu:~/Desktop$ uname -r
3.13.0-24-generic
```

- Ta có thể kiểm tra phiên bản bằng cách gõ lệnh **uname -r** trong Terminal
- Vì phiên bản hiện tại đang sử dụng là **3.13.0-24-generic** nên ta sẽ sử dụng bản kernel **3.13.11** được tải theo đường dẫn:

<https://mirrors.edge.kernel.org/pub/linux/kernel/v3.x/linux-3.13.11.tar.xz>

Lưu ý: Tải các phiên bản mới nhất sẽ bị lỗi khi gọi hàm syscall trong Userspace nếu không cài đặt Makefile đúng cách, team đã sử dụng nhiều phiên bản khác nhau và khuyến khích xài những bản Ubuntu có đính kèm kernel version 3.x or 4.x để dễ thực hiện nhất

- Gõ lệnh sau để tải gói tin về

wget <https://mirrors.edge.kernel.org/pub/linux/kernel/v3.x/linux-3.13.11.tar.xz>

- Sau khi tải xong gói tin, ta giải nén gói tin vào thư mục **/usr/src** bằng câu lệnh:

```
sudo tar -xvf linux-3.13.11.tar.xz -C /usr/src/
```

- Sau khi giải nén thành công, ta mở Terminal, thực hiện cài các gói biên dịch cần thiết cho quá trình biên dịch syscall, ta cài các gói sau:

```
sudo apt-get install libncurses5-dev libncursesw5-dev
sudo apt-get install libssl-dev
sudo apt-get install libelf-dev
sudo apt-get install bison
sudo apt-get install flex
sudo apt-get install bc
sudo apt-get install perl
sudo apt-get install gcc
sudo apt-get install build-essential
sudo apt-get update
sudo apt-get upgrade
```

Lưu ý: Phải chắc chắn rằng các gói này được cài đặt thành công, vì khi biên dịch bị lỗi sẽ tốn rất nhiều thời gian để sửa và tìm ra lỗi

- ➔ Sau khi hoàn thành các thao tác trên, ta di chuyển vào thư mục vừa giải nén bằng lệnh:

```
cd /usr/src/linux-3.13.11
```

- ➔ Các syscall sẽ được định nghĩa và biên dịch trong thư mục **linux-3.13.11**

2. Định nghĩa các syscall và biên dịch:

a) `pnametoid(char *name):`

- Sau khi vào thư mục **linux-3.13.11**, ta tạo thư mục tên là **pnametoid** bằng lệnh

`sudo mkdir pnametoid`

- Ta tạo file **pnametoid.c** bằng lệnh **`sudo gedit pnametoid.c`** và định nghĩa như sau:

```
pnametoid.c x
#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/init.h>
#include <linux/tty.h>
#include <linux/string.h>
#define MAX 32
asmlinkage long sys_pnametoid(char* name){
    printk("WELCOME TO PNAMETOID \n"); /// DEBUG ONLY
    struct task_struct *task;
    char charname[MAX]; //Placeholder
    copy_from_user(charname, name, MAX); ///Get data from user
    printk("Process name: = %s \n",charname); ///DEBUG ONLY
    for_each_process(task){
        /*compares the current process name (defined in task->comm) to the passed in name*/
        if(strcmp(task->comm,charname) == 0){
            return task_pid_nr(task); /// return the PID
        }
    }
    return -1; /// Not found anything
}
```

- **task_struct** là struct có sẵn trong linux giúp ta thao tác với các **process** đang chạy trong linux.
- **copy_from_user(charname,name,MAX):** lấy data nhập từ **Userspace** sao chép vào bộ nhớ của Kernel và pass vào syscall, copy đúng theo số byte khai báo (**MAX = 32 byte**).
- Tên được nhập từ **Userspace** sẽ được so sánh với tên của từng task đang chạy (**task→comm**) và kết quả trả về sẽ là **PID** nếu tìm thấy bằng hàm **task_pid_nr(task)**.

➔ Kết quả trả về **PID** nếu tìm thấy và **-1** nếu không có tiến trình đó đang chạy

- Tương tự với **Makefile** ➔ **`sudo gedit Makefile`**

```
Makefile x
obj-y := pnametoid.o
```

- Sau khi hoàn tất, ta **`cd`** về lại thư mục **linux-3.13.11** và chỉnh sửa file **Makefile** tại đây bằng lệnh **`sudo gedit Makefile`**

→ Trong **Makefile**, ta tìm dòng lệnh như bên dưới và thêm **pnametoid/** ở cuối, cách nhau 1 khoảng trắng so với chữ trước đó

```
ifeq ($(KBUILD_EXTMOD),)
core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ pnametoid/
```

- Chỉnh sửa file **syscall_64.tbl** bằng cách **cd** vào thư mục **syscalls**
cd /usr/src/linux-3.13.11/arch/x86/syscalls và **sudo gedit syscall_64.tbl**

```
313      common   finit_module           sys_finit_module
314      64       pnametoid             sys_pnametoid

#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
512      x32     rt_sigaction          compat_sys_rt_sigaction
```

→ Ta thêm dòng số **314** (Mã số syscall sẽ được gọi trong userspace để test)

pnametoid: tên syscall

sys_pnametoid: tên hàm để thực hiện syscall

- Ta **cd** ngược về thư mục **linux-3.13.11** và **cd** tiếp theo đường dẫn sau:

/usr/src/linux-3.13.11/include/linux

→ Tại đây, ta chỉnh sửa file **syscalls.h** bằng lệnh: **sudo gedit syscalls.h**

```
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_pnametoid(char*);
#endif
```

- Thêm dòng **asmlinkage long sys_pnametoid(char*)** vào cuối file sau **#endif**

b) pidtoname(int pid, char* buf, int len):

- Thư mục **pidtoname** chứa file **pidtoname.c** được đặt cùng chỗ với thư mục **pnametoid** lúc đầu, cách tạo tương tự **pnametoid**



- Ta tạo file **pidtoname.c** bằng lệnh **sudo gedit pidtoname.c** và định nghĩa như sau:

```
pidtoname.c x
#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/init.h>
#include <linux/tty.h>
#include <linux/string.h>
#include <linux/slab.h>
#define LENKERNEL 60

asmlinkage long sys_pidtoname(int pid, char* buf, int len){
    printk("WELCOME TO PIDTONAME\n");
    struct task_struct *task;
    char *process_name = kmalloc(LENKERNEL, GFP_KERNEL);

    for_each_process(task){ //compare to each task is running
        if (task_pid_nr(task) == pid){
            strcpy(process_name, task->comm); //compare name
            if(strlen(task->comm) <= LENKERNEL)
            {
                process_name[strlen(task->comm)] = 0;

                printk("Process Name :=%s\n", process_name);
                copy_to_user(buf, process_name, len - 1);
                if (strlen(process_name) > len - 1)
                {
                    return strlen(process_name); //// Length of string if found
                }
                else return 0;
            }
        }
    }
    return -1; ///Error
}
```

- Ta so sánh id nhập vào từ user space và so sánh với id của các process đang chạy, copy tên của **process** vào biến **process_name**. Sau đó, ta thêm **NULL** vào cuối biến **process_name**. Sử dụng hàm **copy_to_user** để chuyển tên vào biến **buf** với độ dài len - 1. Giá trị trả về là -1 nếu lỗi, 0 nếu **len buffer** truyền vào lớn hơn len của **process_name**, và n với n là độ dài thật sự của process name, trong trường hợp **len buffer** chuyển vào nhỏ hơn len của **process_name**.
- Sau khi hoàn tất, ta **cd** về lại thư mục **linux-3.13.11** và chỉnh sửa file **Makefile** tại đây bằng lệnh **sudo gedit Makefile**. Ta thêm tên **pidtoname** vào cuối dòng.
- Thêm **pidtoname/** vào sau dòng **pnametoid/** trong file **Makefile** trong thư mục **linux-3.13.11**

```
ifeq ($(KBUILD_EXTMOD),)
core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ pnametoid/ pidtoname/

vmlinux-dirs := $(patsubst %/,%, $(filter %/, $(init-y) $(init-m) \
```

- Chỉnh sửa file **syscall_64.tbl** bằng cách **cd** vào thư mục **syscalls** **cd /usr/src/linux-3.13.11/arch/x86/syscalls** và **sudo gedit syscall_64.tbl**. Ta thêm dòng số 315 (Mã số syscall sẽ được gọi trong userspace để test).

```

313      common  finit_module          sys_finit_module
314      64      pnametoid             sys_pnametoid
315      64      pidtoname             sys_idtoname

#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
512      x32      rt_sigaction        compat_sys_rt_sigaction
513      x32      rt_sigreturn        stub_x32_rt_sigreturn

```

- Tiếp theo chúng ta sử dụng câu lệnh:

cd /usr/src/linux-3.13.11/include/linux sau đó chỉnh sửa file **syscalls.h** bằng lệnh: **sudo gedit syscalls.h**.

Tại đây chúng ta thêm dòng

asmlinkage long sys_pidtoname(int, char*, int) vào cuối file sau **#endif**

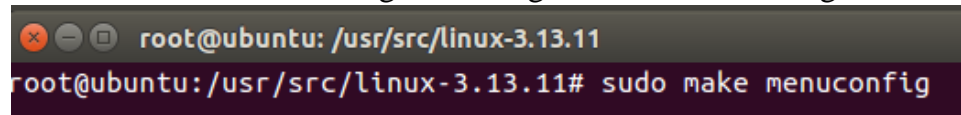
```

asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_pnametoid(char*);
asmlinkage long sys_pidtoname(int, char*, int);
#endif

```

c) Biên dịch syscall:

- Trước khi biên dịch, trong thư mục gốc là **linux-3.13.11** gõ **sudo make menuconfig**

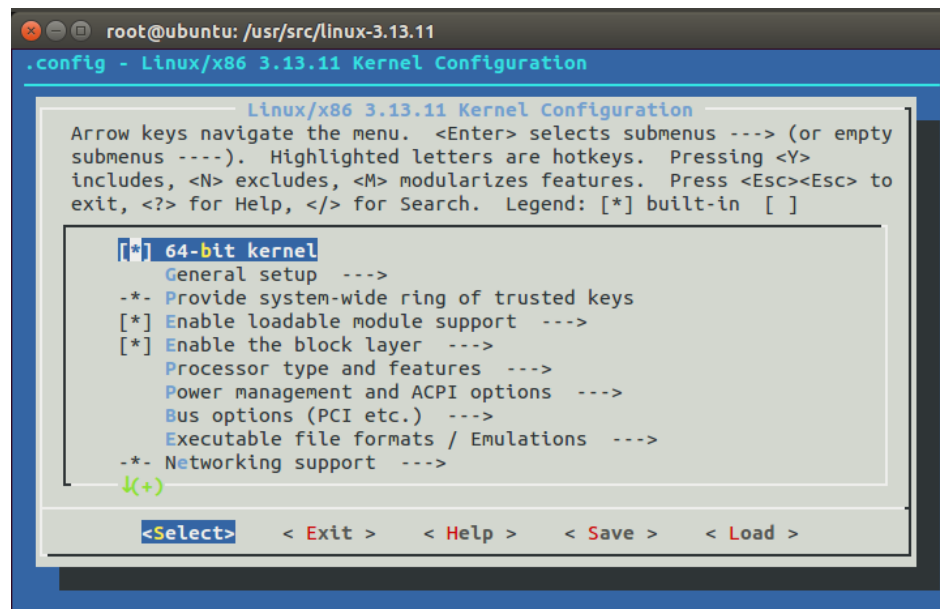


```

root@ubuntu: /usr/src/linux-3.13.11
root@ubuntu: /usr/src/linux-3.13.11# sudo make menuconfig

```

- Tiếp đến chuyển qua tab **save** do không cần cài đặt chuyên sâu, sử dụng các cài đặt mặc định, sau đó **exit**



```

Linux/x86 3.13.11 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
[*] 64-bit kernel
  General setup --->
  *- Provide system-wide ring of trusted keys
  [*] Enable loadable module support --->
  [*] Enable the block layer --->
    Processor type and features --->
    Power management and ACPI options --->
    Bus options (PCI etc.) --->
    Executable file formats / Emulations --->
  *- Networking support --->
    (+)
<Select> < Exit > < Help > < Save > < Load >

```

Lưu ý: Không cài đặt menuconfig thì sẽ không biên dịch được kernel

- Bước cuối cùng là biên dịch, ta gõ lệnh sau trong **Terminal**

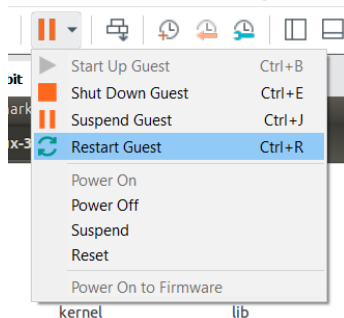
sudo make -j 8 && sudo make modules_install -j 8 && sudo make install -j 8

```
root@ubuntu:/usr/src/linux-3.13.11# sudo make -j 8 && sudo make modules_install
-j 8 && sudo make install -j 8
```

→ Số 8 tượng trưng cho 8 cores (Tuỳ vào số core mà ta cài đặt khi setting VMWARE), càng nhiều core thì thời gian biên dịch càng thấp. Vì sử dụng 8 core nên thời gian biên dịch chỉ tốn 45 phút.

```
Found initrd image: /boot/initrd.img-3.13.11
Found linux image: /boot/vmlinuz-3.13.11.old
Found initrd image: /boot/initrd.img-3.13.11
Found linux image: /boot/vmlinuz-3.13.0-24-generic
Found initrd image: /boot/initrd.img-3.13.0-24-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
root@ubuntu:/usr/src/linux-3.13.11#
```

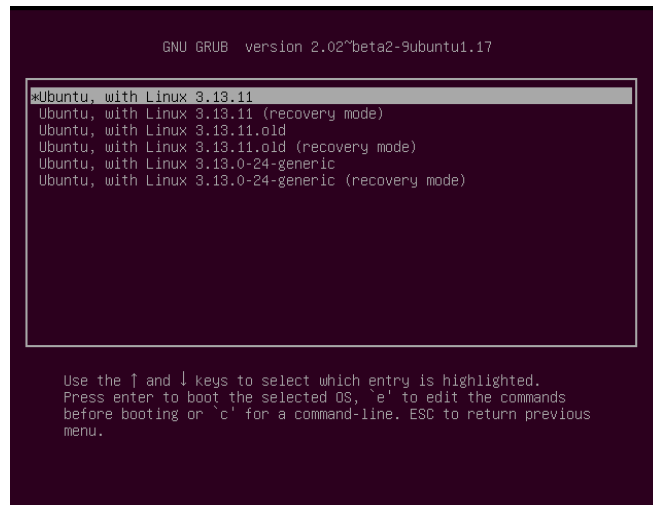
→ **Đã biên dịch xong**



- Sau khi biên dịch xong, ta restart Ubuntu ở VMWARE
- Khi màn hình bắt đầu, ta bấm **SHIFT** để vào **boot menu** của Ubuntu



- Chọn **Advanced options for Ubuntu**



- Chọn **3.13.11 (Bản kernel ta cài đặt lúc đầu)**, trong này chứa các syscall ta đã biên dịch và đã được ráp vào **Kernel**

Lưu ý: Phải chắc chắn rằng chọn đúng phiên bản, nếu không Userspace gọi syscall sẽ bị lỗi

d) Userspace test:

- Ta tạo thư mục **test** ở Desktop bằng lệnh **mkdir test**, trong đây ta định nghĩa 2 file **pnameUser.c** và **Makefile** như sau:
+ **pnameUser.c**

```
*pnameUser.c x
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <string.h>
#define MAX 32
//////// 314 -> PNAMETOID
//////// 315 -> PIDTONAME
int main(){
//----- PNAMETOID TEST
char process_name[MAX]; ///PNAMETOID
printf("Enter process' name to find: ");
scanf("%s",process_name);
strtok(process_name, "\n");
printf("Your Input: %s \n",process_name);
long int id = syscall(314,process_name); //syscall number 314 and passing in the string.
printf("System call return: %ld \n",id);
if(id== -1){
printf("Status: Process' name not found!\n");
}
else{
printf("Status: success!\n");
printf("Name = %s \n",process_name);
printf("PID = %ld \n",id);
}
//----- PIDTONAME TEST
char process_name_out[MAX]; ///PIDTONAME -> NAME OUT
int ID; //----> Input ID
printf("Input ID of process to find: \n");
scanf("%d", &ID);
int ret = syscall(315, ID, process_name_out, MAX); //syscall number 314 and passing in the ID
if(ret!= -1){
printf("Status: Success!\n");
printf("Process' name: = %s\n",process_name_out);
printf("PID = %d \n",ID);
}
return 0;
}
```

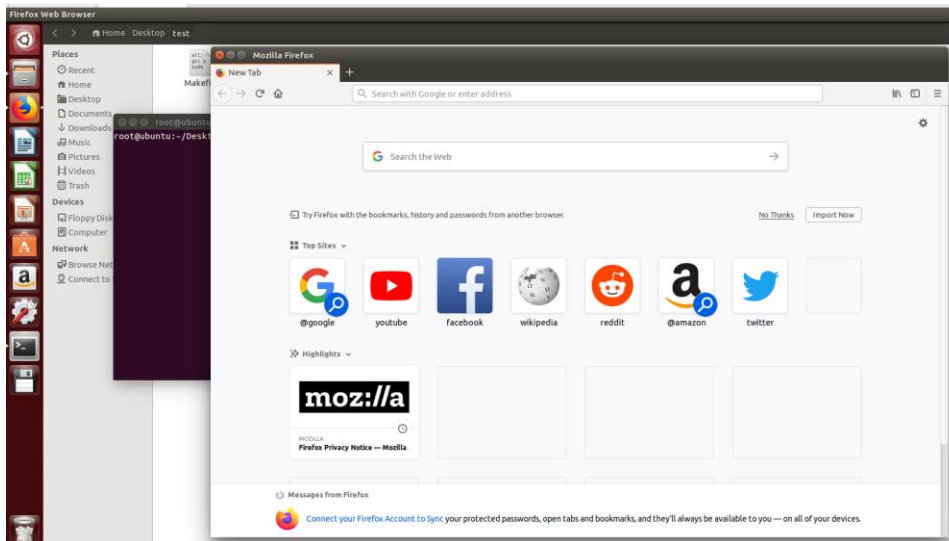

+ Makefile

```
Makefile x
all:
    gcc pnameUser.c -o testPname
    sudo ./testPname
```

→ Ta sẽ test cả hai syscall cùng một lúc

- **pnametoid(char *name):**

→ Hàm **sys_pnametoid** sẽ trả về **Process ID (PID)** của một tiến trình đang chạy. Ví dụ đơn giản nhất đó là **Firefox** (Có Process name là **firefox**) → Ta mở **Firefox** lên



→ Ta **cd** vào thư mục **test** ở Desktop và mở Terminal gõ **make all** → nhập Process name là **firefox**, chuỗi tên sẽ được truyền vào trong **kernel** thông qua mã số **system call** của **pnametoid** ở đây là 314

```
root@ubuntu: ~/Desktop/test
root@ubuntu:~/Desktop/test# make all
gcc pnameUser.c -o testPname
sudo ./testPname
Enter process' name to find: firefox
Your Input: firefox
System call return: 2652
Status: success!
Name = firefox
PID = 2652
root@ubuntu:~/Desktop/test#
```

➔ Kết quả trả về là 2652 tương ứng với PID của firefox

- Ta có thể kiểm chứng bằng lệnh **pgrep + [Tên Process]**

```
root@ubuntu:~/Desktop/test# pgrep firefox
2652
```

- Kết quả hiện trong **dmesg**

```
[ 185.086501] WELCOME TO PNAMETOID
[ 185.086504] Process name: = firefox
root@ubuntu:~/Desktop/test#
```

- **pidtoname (int pid, char* buf, int len)**

- Để kiểm tra hàm **pidtoname**, chúng ta sẽ tìm kiếm bằng **id** của tiến trình **firefox**. Các bước chạy tương tự như hàm **pnametoid**.

```
nguyentathung943@ubuntu:~/Desktop/test$ sudo make all
gcc pnameUser.c -o testPname
sudo ./testPname
Enter process' name to find: firefox
Your Input: firefox
System call return: 2752
Status: success!
Name = firefox
PID = 2752
Input ID of process to find:
2752
Status: Success!
Process' name: = firefox
PID = 2752
```

➔ Ta có thể thấy rằng tiến trình trả về name là **firefox** tức là đã tìm thấy **process_name** đang chạy trong tiến trình

➔ Ngoài ra, ta có thể kiểm tra với nhiều tiến trình khác nhau trong linux, để kiểm tra bảng tiến trình chứa **name** và **PID** của tất cả các tiến trình đang chạy, ta gõ lệnh:

ps -aux

3. Hook syscall open và write

a) Cài đặt hook

- Để hook vào syscall của máy ta cần tìm địa chỉ syscall table bằng lệnh:

cat /boot/System.map-3.16.0-23-generic | grep sys_call_table

```
aa@ubuntu:~$ sudo cat /boot/System.map-3.16.0-23-generic | grep sys_call_table
ffffffff81801680 R sys_call_table
ffffffff8180cd40 R ia32_sys_call_table
```

- Copy địa chỉ **syscall_call_table** vào hàm init của code hook như sau:

```
static int __init entry_point(void){
    printk(KERN_INFO "Hook loaded successfully..\n");
    /*MY sys_call_table address*/
    system_call_table_addr = (void*)0xffffffff81801680;
    /* Replace custom syscall with the correct system call name
    (write,open,etc) to hook*/
    temp_open = system_call_table_addr[__NR_open];
    temp_write = system_call_table_addr[__NR_write];
    /*Disable page protection*/
    make_rw((unsigned long)system_call_table_addr);
    /*Change syscall to our syscall function*/
    system_call_table_addr[__NR_open] = hook_open;
    system_call_table_addr[__NR_write] = hook_write;
    return 0;
}
```

- Đây là hàm đọc thông tin **open** và **write** từ hệ thống rồi sau đó in thông tin ta muốn in vào **dmesg**

```
asmlinkage long hook_open(const char* filename, int flags, umode_t mode)
{
    char buff[100];
    copy_from_user(buff, filename, 100);
    printk(KERN_INFO "process name opens file: %s | hooked open: filename = %s\n", current->comm, buff);
    return temp_open(filename, flags, mode);
}
asmlinkage long hook_write(unsigned int fd, const char* buf, size_t len)
{
    char *tmp;
    char *pathname;
    struct file *file;
    struct path *path;

    spin_lock(&current->files->file_lock);
    file = fcheck_files(current->files, fd);
    if (!file) {
        spin_unlock(&current->files->file_lock);
        return -ENOENT;
    }

    path = &file->f_path;
    path_get(path);
    spin_unlock(&current->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);
    }

    ssize_t bytes;
    bytes = (*temp_write)(fd, buf, len);
    if (fd > 5) {
        printk(KERN_INFO "process name writes file: %s | hooked write: filename = %s, len = %d\n", current->comm, pathname, bytes);
        free_page((unsigned long)tmp);
        return bytes;
    }
}
```

➔ Đặc biệt là hàm **hook_write**, vì không trả về **filename**, chỉ trả về file descriptor – “mô tả file” nên phải gõ thêm câu lệnh để lấy filename từ file descriptor trong tiến trình, các dòng if để tránh trường hợp filename bị lỗi

- Hàm **make_rw** tắt lớp bảo vệ của syscall, còn hàm **make_ro** bật lại lớp bảo vệ của syscall.

```
/*Make page writeable*/
int make_rw(unsigned long address){
    unsigned int level;
    pte_t *pte = lookup_address(address, &level);
    if(pte->pte & ~_PAGE_RW){
        pte->pte |= _PAGE_RW;
    }
    return 0;
}
/* Make the page write protected */
int make_ro(unsigned long address){
    unsigned int level;
    pte_t *pte = lookup_address(address, &level);
    pte->pte = pte->pte & ~_PAGE_RW;
    return 0;
}
```

- Đây là 2 hàm sẽ được thực thi khi chạy **insmod** để ráp module hook vào kernel và **rmmod** để gỡ module ra:

```
static int __init entry_point(void){
    printk(KERN_INFO "Hook loaded successfully..\n");
    /*MY sys_call_table address*/
    system_call_table_addr = (void*)0xffffffff81801680;
    /* Replace custom syscall with the correct system call name
    (write,open,etc) to hook*/
    open = system_call_table_addr[__NR_open];
    write = system_call_table_addr[__NR_write];
    /*Disable page protection*/
    make_rw((unsigned long)system_call_table_addr);
    /*Change syscall to our syscall function*/
    system_call_table_addr[__NR_open] = hook_open;
    system_call_table_addr[__NR_write] = hook_write;
    return 0;
}
static int __exit exit_point(void){
    printk(KERN_INFO "Unloaded Captain Hook successfully\n");
    /*Restore original system call */
    system_call_table_addr[__NR_open] = open;
    system_call_table_addr[__NR_write] = write;
    /*Renable page protection*/
    make_ro((unsigned long)system_call_table_addr);
    return 0;
}
```

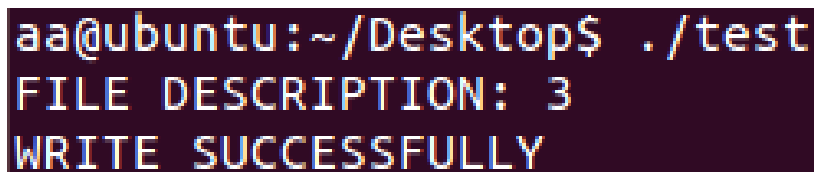
- Hàm init sẽ hook vào syscall bằng địa chỉ của syscall, tắt lớp bảo vệ của syscall đồng thời thực hiện gán hàm open với hàm write để đọc thông tin từ syscall.
- Hàm exit sẽ khôi phục syscall, bật lớp bảo vệ cho syscall

b) Test Hook

- Ta xây dựng file **test** hook ở Desktop như sau:

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <string.h>
int main(){
int fd = open("in.txt",O_WRONLY | O_CREAT | O_APPEND);
printf("FILE DESCRIPTION: %d\n",fd);
if(write(fd, "He Dieu Hanh\n", 13) == 13) {
    printf("WRITE SUCCESSFULLY\n");
}
else{
    printf("WRITE FAILURE\n");
}
return 0;
}
```

- Ta sẽ in ra các thông báo nhằm phục vụ cho quá trình debug và thông báo thành công
- Ta biên dịch file **test.c** và thấy hiện thông báo đã hook thành công

A terminal window with a dark background and light-colored text. The prompt is 'aa@ubuntu:~/Desktop\$'. The user has entered './test'. The output of the program is displayed on the next two lines: 'FILE DESCRIPTION: 3' and 'WRITE SUCCESSFULLY'.

```
aa@ubuntu:~/Desktop$ ./test
FILE DESCRIPTION: 3
WRITE SUCCESSFULLY
```

- Sau khi **insmod** ráp kernel thì ta sẽ gõ lệnh **dmesg -wH** để theo dõi theo thời gian thực:

```
+0.000011] process name opens file: sudo | hooked open: filename = /etc/group
+0.000015] process name opens file: sudo | hooked open: filename = /etc/group
+0.000012] process name opens file: sudo | hooked open: filename = /etc/group
+0.000011] process name opens file: sudo | hooked open: filename = /etc/group
+0.000013] process name opens file: sudo | hooked open: filename = /etc/group
+0.000014] process name opens file: sudo | hooked open: filename = /etc/group
+0.000020] process name opens file: sudo | hooked open: filename = /etc/nsswitch.conf
+0.000028] process name opens file: sudo | hooked open: filename = /etc/sudoers
+0.000073] process name opens file: sudo | hooked open: filename = /etc/sudoers.d/README
+0.000087] process name opens file: sudo | hooked open: filename = /etc/passwd
+0.000034] process name opens file: sudo | hooked open: filename = /etc/host.conf
+0.000019] process name opens file: sudo | hooked open: filename = /etc/resolv.conf
+0.000025] process name opens file: sudo | hooked open: filename = /etc/hosts
+0.000036] process name opens file: sudo | hooked open: filename = /proc/sys/kernel/ngroups_max
+0.000021] process name opens file: sudo | hooked open: filename = /etc/group
+0.000026] process name opens file: sudo | hooked open: filename = /etc/group
+0.000094] process name opens file: sudo | hooked open: filename = /etc/pam.d/sudo
+0.000020] process name opens file: sudo | hooked open: filename = /lib/x86_64-linux-gnu/security/pam_env.so
+0.000059] process name opens file: sudo | hooked open: filename = /etc/pam.d/common-auth
+0.000020] process name opens file: sudo | hooked open: filename = /lib/x86_64-linux-gnu/security/pam_unix.so
+0.000029] process name opens file: sudo | hooked open: filename = /etc/ld.so.cache
+0.000013] process name opens file: sudo | hooked open: filename = /lib/x86_64-linux-gnu/libcrypt.so.1
+0.000107] process name opens file: sudo | hooked open: filename = /lib/x86_64-linux-gnu/security/pam_deny.so
+0.000037] process name opens file: sudo | hooked open: filename = /lib/x86_64-linux-gnu/security/pam_permit.so
+0.000046] process name opens file: sudo | hooked open: filename = /lib/x86_64-linux-gnu/security/pam_cap.so
+0.000075] process name opens file: sudo | hooked open: filename = /etc/ld.so.cache
+0.000023] process name opens file: sudo | hooked open: filename = /lib/x86_64-linux-gnu/libcap.so.2
+0.000101] process name opens file: sudo | hooked open: filename = /etc/pam.d/common-account
+0.000033] process name opens file: sudo | hooked open: filename = /etc/pam.d/common-session-noninteractive
+0.000024] process name opens file: sudo | hooked open: filename = /lib/x86_64-linux-gnu/security/pam_umask.so
+0.000100] process name opens file: sudo | hooked open: filename = /etc/pam.d/other
```

- Vì hệ thống thực hiện đọc ghi liên tục nên cửa sổ của **dmesg** sẽ bị tràn, thông tin của lệnh open và write từ hệ thống sẽ xuất hiện liên tục. Nên ta sẽ dùng file test và gõ thêm câu lệnh **if** với **strcmp(current->comm,"test")** để khi thực thi file test thì ta có thể thấy được kết quả mà không bị tràn

```
if(strcmp(current->comm,"test") == 0){
    printk(KERN_INFO "process name writes file: %s | hooked write: filename = %s, len = %d\n", current->comm, pathname, bytes);}
from proc/(un)read len+time);
```

```
[ +1.724241] Unloaded Captain Hook successfully
[ +5.193033] Hook loaded successfully..
[ +33.225061] process name opens file: test | hooked open: filename = /etc/ld.so.cache
[ +0.000016] process name opens file: test | hooked open: filename = /lib/x86_64-linux-gnu/libc.so.6
[ +0.000130] process name opens file: test | hooked open: filename = in.txt
[ +0.000040] process name writes file: test | hooked write: filename = /dev/pts/0, len = 20
[ +0.000423] process name writes file: test | hooked write: filename = /home/aa/in.txt, len = 13
[ +0.000034] process name writes file: test | hooked write: filename = /dev/pts/0, len = 19
```

- Và đây là kết quả sau khi chạy file **test** của hook (hiện trong **dmesg**):
➔ Ta thấy thông báo hook thành công sau khi thực hiện ráp module và gỡ module ra khỏi kernel, biên dịch file hook để thấy kết quả in trong dmesg

III. Tài liệu tham khảo:

1. Syscall:

[Adding a System Call Which Can Pass a Userspace String](#)

[Adding a Hello World System Call to Linux Kernel - Anubhav Shrimal - Medium](#)

[Basics of Making a Rootkit: From syscall to hook! | University of South Wales: Cyber University of the Year: 2019](#)

https://github.com/htn274/OperatingSystem/tree/master/Project_2_Systemcall?fbclid=IwAR0L7GJQtnJpmPffZvS4RzgluTzQBqa0Ji9cpQd46eTBsex2L-aiXiPcfs4

<https://stackoverflow.com/questions/11035119/passing-parameter-to-a-kernel-module>

https://www.quora.com/Linux-Kernel-How-does-copy-to-user-work?fbclid=IwAR0S_uVc3AnFzHrKDgo6iQhUr8NIVYTCcplGXafHjhbSdJYOLd_NI4ysiUQ

<https://stackoverflow.com/questions/45653760/adding-char-with-null-terminator-to-string-in-c?fbclid=IwAR1Zlnavu1uB7OmD6JkFVqmqsqdtjC3EIGM6DmY8CTsCYFSEGXstYNq0>

2. Hook:

<https://github.com/1612198/My-University/tree/master/System%20Call/Sources/Hook>

https://github.com/hoangknguyen1807/OS-Project02?fbclid=IwAR1GQSAHrcLkrj1iAl_cS1CfVf6D2Zw8GpcAUxhYCyvfo2J_T1ls0HAa7so#ph%E1%BA%A7n-1-kernelmodule--m%E1%BB%A5c-ti%C3%AAu-hi%E1%BB%83u-v%E1%BB%81-linux-kernel-module-v%C3%A0-h%E1%BB%87-th%E1%BB%91ng-qu%E1%BA%A3n-l%C3%BD-file-v%C3%A0-device-trong-linux-giao-ti%E1%BA%BFp-gi%E1%BB%AFa-ti%E1%BA%BFn-tr%C3%ACnh-%E1%BB%9F-user-space-v%C3%A0-kernel-space