

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**  
**KHOA CÔNG NGHỆ THÔNG TIN**



# **HỆ ĐIỀU HÀNH**

## **ĐỒ ÁN**

### **SYSTEM CALL HOOK**

Thành viên nhóm:

Nguyễn Trung Kiên – 18127123

Nguyễn Quang Pháp – 18127174

Nguyễn Hà Thành – 18127214

Thành phố Hồ Chí Minh - 2020



---

## CHUẨN BỊ

---

Ở đồ án này, nhóm sử dụng:

- Ubuntu 14.04.2 LTS
- Kernel version: 3.16.0-77-generic

Tải các gói sau (nhập trong terminal):

```
sudo apt-get install build-essential  
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
```

Kiểm tra phiên bản kernel: **uname -r**

```
kien@ubuntu:~$ uname -r  
3.16.0-77-generic
```

Tải linux kernel phù hợp (ở đây dùng gói **linux-3.16.tar.xz**)

<https://mirrors.edge.kernel.org/pub/linux/kernel/>

Tải về: **wget <đường link của gói>**

Giải nén vào thư mục **/usr/src/** bằng lệnh **sudo tar -xvf linux-3.16.tar.xz -C /usr/src/**

Kiểm tra lại, vào thư mục **/usr/src/** và gõ **ls**

```
kien@ubuntu:~$ cd /usr/src  
kien@ubuntu:/usr/src$ ls  
linux-3.16 linux-headers-3.16.0-77  
linux-headers-3.16.0-30 linux-headers-3.16.0-77-generic  
linux-headers-3.16.0-30-generic
```

Đến đây đã xong bước chuẩn bị, ở các phần sau sẽ hướng dẫn chi tiết từng bước viết system call cũng như hook một system call có sẵn.

## VIẾT SYSTEM CALL CHO HỆ THỐNG

### Lưu ý:

- Nếu dùng Ubuntu phiên bản khác thì ở một số bước làm sẽ có sự thay đổi nhỏ. Tuy nhiên nhìn chung hướng đi sẽ như nhau.
- Sau khi giải nén vào /usr/src/ phải biên dịch kernel trước (chi tiết xem ở B7)

### 1. int pnametoid (char \*name);

- Chức năng: Nhận vào tên của một tiến trình và trả về PID của tiến trình đó nếu tìm thấy và trả về -1 nếu không tìm được.

- Các bước thực hiện:

**B1:** Vào /usr/src/linux-3.16 và tạo thư mục **pnametoid**

```
kien@ubuntu:/usr/src/linux-3.16$ sudo mkdir pnametoid
```

**B2:** Tạo **pnametoid.c** chứa mã nguồn xử lý cho system call này

```
kien@ubuntu:/usr/src/linux-3.16$ cd pnametoid  
kien@ubuntu:/usr/src/linux-3.16/pnametoid$ sudo gedit pnametoid.c
```

Trong file pnametoid.c

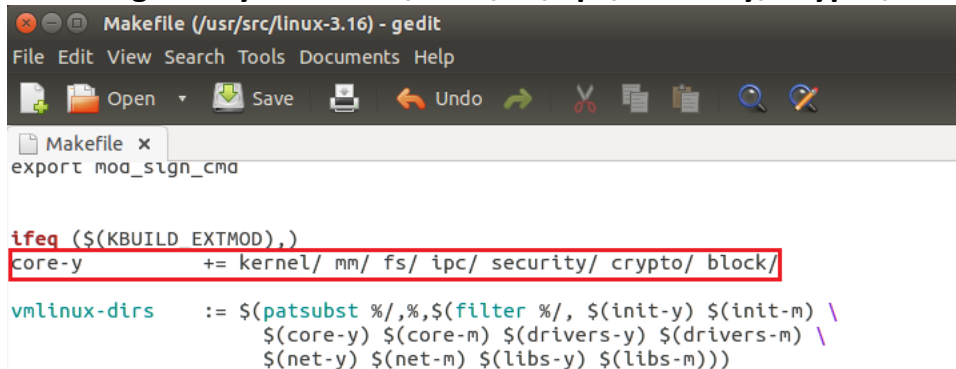
```
#include <linux/syscalls.h>  
#include <linux/kernel.h>  
#include <linux/sched.h>  
#include <linux/init.h>  
#include <linux/string.h>  
#include <linux/uaccess.h>  
  
asmlinkage long sys_pnametoid(char __user *data)  
{  
    /* task_struct là một cấu trúc dữ liệu, chứa các thông tin của process */  
    struct task_struct *task;  
  
    /* copy data từ user space xuống kernel space */  
    char buf[16];  
    if(copy_from_user(buf, data, 16))  
        return -1;  
  
    /* duyệt qua tất cả các process của hệ điều hành */  
    for_each_process(task)  
    {  
        /* task->comm là một mảng char lưu tên của process */  
        /* so sánh tên process nhập vào (data) với task->comm */  
        if(strcmp(task->comm, buf) == 0)  
        {  
            /* task_pid_nr(task) trả về "global" PIDs của process hiện tại và con số này là duy nhất trong toàn bộ hệ điều hành */  
            return (long)task_pid_nr(task);  
        }  
    }  
    return -1;  
}
```

Tạo Makefile trong thư mục **pnametoid** để biên dịch

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

**B3:** Chỉnh sửa lại Makefile của kernel:

- cd /usr/src/linux-3.16
- sudo gedit Makefile
- Tìm dòng **core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block**

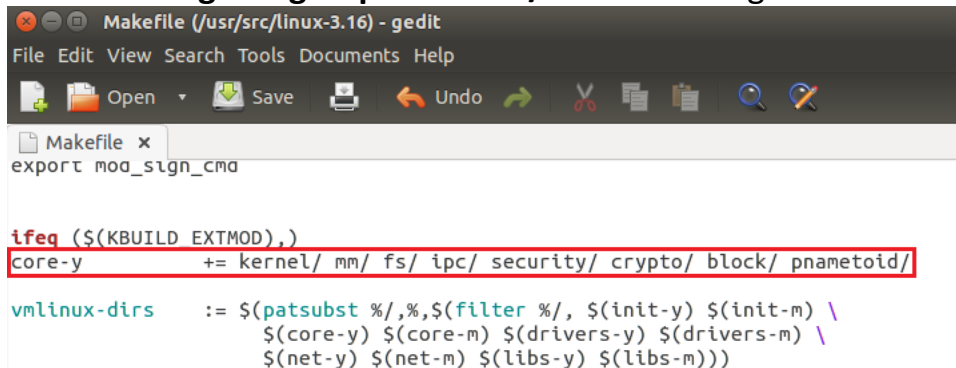


```
Makefile (/usr/src/linux-3.16) - gedit
File Edit View Search Tools Documents Help
Open Save Undo Redo
Makefile x
export mod_sign_cmd

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

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

- Thêm **khoảng trắng** và **pnametoid/** vào cuối dòng



```
Makefile (/usr/src/linux-3.16) - gedit
File Edit View Search Tools Documents Help
Open Save Undo Redo
Makefile x
export mod_sign_cmd

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

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

**B4:** Trong terminal, gõ lệnh:

```
cd /usr/src/linux-3.16/arch/x86/syscalls/
```

Do máy 64-bit nên ta sẽ sửa tập tin *syscall\_64.tbl*  
(đối với máy 32-bit thì là *syscall\_32.tbl*)

```
kien@ubuntu:/usr/src/linux-3.16/arch/x86/syscalls$ ls
Makefile      syscall_64.tbl  syscallhdr.sh
syscall_32.tbl syscall_64.tbl~ syscalltbl.sh
```

**B5:** Đến đây, ta sẽ thêm các thông tin của syscall này

*317 common pnametoid sys\_pnametoid*

syscall_64.tbl x			
306	common	syncfs	sys_syncfs
307	64	sendmmsg	sys_sendmmsg
308	common	setns	sys_setns
309	common	getcpu	sys_getcpu
310	64	process_vm_readv	sys_process_vm_readv
311	64	process_vm_writev	sys_process_vm_writev
312	common	kcmp	sys_kcmp
313	common	finit_module	sys_finit_module
314	common	sched_setattr	sys_sched_setattr
315	common	sched_getattr	sys_sched_getattr
316	common	renameat2	sys_renameat2
317	common	pnametoid	sys_pnametoid
#			
# x32-specific system call numbers start at 512 to avoid cache impact			
# for native 64-bit operation.			
#			

Mỗi một system call sẽ có một id riêng, ta chọn **317**  
(có thể chọn số khác, miễn là số đó chưa được dùng)

**B6:** Dẫn tới thư mục */usr/src/linux-3.16/include/linux* và mở tập tin **syscalls.h**.

*kien@ubuntu:/usr/src/linux-3.16/include/linux\$ sudo gedit syscalls.h*

Thêm prototype của syscall này vào cuối file, trước **#endif**

```

asmlinkage long sys_process_vm_writev(pid_t pid,
                                     const struct iovec __user *lvec,
                                     unsigned long liovcnt,
                                     const struct iovec __user *rvec,
                                     unsigned long riovcnt,
                                     unsigned long flags);

asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                        unsigned long idx1, unsigned long idx2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_pnametoid(char __user *data);
#endif

```

**B7:** Biên dịch kernel:

- **cd /usr/src/linux-3.16**
- Ở lần biên dịch đầu tiên, gõ lệnh **sudo make menuconfig**
  - + Chọn **Save -> OK -> Exit -> Exit**
  - + Biên dịch và cài các module cần thiết
    - sudo make -j 4
    - sudo make modules\_install -j 4
    - sudo make install -j 4

- Trong các lần sau, chỉ cần gõ:  
    sudo make -j 4  
    sudo make install -j 4  
(số ở cuối mỗi lệnh là số cores mà máy tính có)

**B8:** Reboot lại hệ điều hành:

- Lúc khởi động, giữ **Shift** để vô menu boot
- Chọn **Advanced options for Ubuntu -> Ubuntu, with Linux 3.16.0**

**B9:** Viết file để test system call vừa tạo, file này tên **run\_pnametoid.c**

```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char data[16];
    long int pid;

    printf("Enter process's name: ");
    scanf("%s", data);
    while((getchar()) != '\n');

    pid = syscall(317, data);
    printf("PID = %ld\n", pid);
    return 0;
}
```

**B10:** Biên dịch và chạy thử file test:

```
kien@ubuntu:~/Desktop/linux$ gcc run_pnametoid.c -o run1
```

```
kien@ubuntu:~/Desktop/linux$ ./run1
Enter process's name: compiz
PID = 2396
```

## 2. int pidtoname(int pid, char\* buf, int len);

- Chức năng: Nhận vào pid của một tiến trình, ghi process name vào biến buff với max len là len - 1, phần tử cuối tự thêm NULL.
- Các bước thực hiện: Tương tự như system call trên

- **pidtoname.c** chứa mã xử lí

```
#include <linux/syscalls.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/init.h>
#include <linux/string.h>
#include <linux/uaccess.h>
#include <linux/slab.h>

asmlinkage long sys_pidtoname(int pid, char __user *buf, int len)
{
    struct task_struct *task;
    int len_process_name = 0;

    /* Kernel buffer de chua ten process */
    char* kernel_buf = kmalloc(16, GFP_KERNEL);

    for_each_process(task)
    {
        if((int)task_pid_nr(task) == pid)
        {
            /* Copy ten cua process vao kernel buffer */
            strcpy(kernel_buf, task->comm);

            /* Tinh chieu dai cua process name */
            len_process_name = strlen(task->comm);

            /* Gan ki tu cuoi bang NULL */
            kernel_buf[len_process_name] = '\0';

            /* Copy du lieu tu kernel space qua user space */
            copy_to_user(buf, kernel_buf, len_process_name + 1);

            if(len > len_process_name)
                return 0;

            return len_process_name;
        }
    }
    return -1;
}
```

- Sửa Makefile của kernel

```
Makefile x
else
mod_sign_cmd = true
endif
export mod_sign_cmd

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

- Sửa tập tin `syscall_64.tbl`

```
syscall_64.tbl x
314 common sched_setattr sys_sched_setattr
315 common sched_getattr sys_sched_getattr
316 common renameat2 sys_renameat2
317 common pnametoid sys_pnametoid
318 common pidtoname sys_pidtoname
#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
```

- Thêm prototype vào tập tin `syscalls.h`

```
asm linkage long sys_pnametoid(char __user *data);
asm linkage long sys_pidtoname(int pid, char __user *buf, int len);
#endif
```

- Test chương trình `run_pidtoname.c`

```
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    char data[16];
    int pid;
    long int num_bytes;

    printf("Enter PID = ");
    scanf("%d", &pid);

    num_bytes = syscall(318, pid, data, 16);

    if(num_bytes < 0)
        printf("Process does not exist\n");
    else
        printf("Process's name: %s\n", data);

    return 0;
}
```

- Biên dịch và chạy thử file test

```
kien@ubuntu:~/Desktop/linux$ gcc run_pidtoname -o run2
```

```
kien@ubuntu:~/Desktop/linux$ ./run2
Enter PID = 2396
Process's name: compiz
```

- Để xem thông tin các process đang chạy, gõ **`ps -aux`** trong terminal



## HOOK VÀO MỘT SYSTEM CALL CÓ SẴN

**Ý tưởng:** Quá trình Hook được thực hiện bằng cách tạo ra module. Module này chứa các hàm truy cập vào syscall table của hệ thống và thay đổi hàm open và write.

### B1: Lấy địa chỉ syscall table của hệ thống

```
phap@phap-virtual-machine:~$ sudo cat /boot/System.map-$(uname -r) | grep sys_call_table
[sudo] password for phap:
ffffffff81801460 R sys_call_table
ffffffff8180c5a0 R ia32_sys_call_table
phap@phap-virtual-machine:~$
```

### B2: Truyền vào các thư viện

```
#include <asm/unistd.h>
#include <asm/cacheflush.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <asm/pgtable_types.h>
#include <linux/highmem.h>
#include <linux/fs.h>
#include <linux/sched.h>
#include <linux/moduleparam.h>
#include <linux/unistd.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/fdtable.h>
```

```
MODULE_LICENSE("GPL");
```

```
MODULE_AUTHOR("Nguyen Trung Kien: 18127123 - Nguyen Quang Phap: 18127174 - Nguyen Ha Thanh: 18127214");
```

### B3: Xây dựng hàm để phá gỡ và khôi phục lớp bảo vệ system call table

```
/* Pha lop bao ve sys_call_table */
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;
}

/* Khoi phuc lai lop bao ve sys_call_table */
int make_ro(unsigned long address){
    unsigned int level;
    pte_t *pte = lookup_address(address, &level);
    pte->pte = pte->pte &~ _PAGE_RW;
    return 0;
}
```

## B4: Thiết lập module

```
static int __init entry_point(void){
    printk(KERN_INFO "Captain Hook loaded successfully..\n");

    /* Địa chỉ của sys_call_table */
    system_call_table_addr = (void *)0xffffffff81801460;

    /* Khởi tạo các biến cần theo dõi */
    open_syscall = (void *)system_call_table_addr[__NR_open];
    write_syscall = (void *)system_call_table_addr[__NR_write];

    /* Pha lớp bảo vệ cho sys_call_table */
    make_rw((unsigned long)system_call_table_addr);

    /* Change syscall to our syscall function */
    system_call_table_addr[__NR_open] = new_open_syscall;
    system_call_table_addr[__NR_write] = new_write_syscall;
    return 0;
}

static void __exit exit_point(void){
    printk(KERN_INFO "Unloaded Captain Hook successfully\n");

    /* Restore original system call */
    system_call_table_addr[__NR_open] = open_syscall;
    system_call_table_addr[__NR_write] = write_syscall;

    /* Phục hồi lớp bảo vệ cho sys_call_table */
    make_ro((unsigned long)system_call_table_addr);
}

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

## B5: Thay đổi hàm open

- Cấu trúc của hàm open trong hệ thống

```
int open (const char *pathname, int flags, mode_t mode);
```

- Cập nhật hàm open

```
void **system_call_table_addr;
char buff[100];

/* My custom syscall that takes process name */
asmlinkage int (*open_syscall) (const char *, int, mode_t);
asmlinkage int (*write_syscall) (int, void *, size_t);

/* Hook */
asmlinkage int new_open_syscall(const char * filename, int flags, mode_t mode) {
    copy_from_user(buff, filename, 100);
    printk(KERN_INFO "Process Name: %s Filename is open: %s\n", current->comm, buff);
    return open_syscall(filename, flags, mode);
}
```

## B6: Thay đổi hàm write

- Cấu trúc của hàm write trong hệ thống

```
ssize_t write(int fd, void *buffer, size_t count);
```



- Cập nhật hàm write

```
asmlinkage int new_write_syscall(int fd, void *buf, size_t len) {
    /* Hàm lấy đường dẫn tên file từ giá trị fd */
    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);
    }

    free_page((unsigned long)tmp);
    if (strcmp(current->comm, "user_test") == 0)
        printk(KERN_INFO "Process Name: %s Filename is write: %s fd: %d nByte: %d\n", current->comm, pathname, fd, (int)len);

    return write_syscall(fd, buf, len);
}
```

## B7: Chạy chương trình

- Terminal thứ 1:

```
phap@phap-virtual-machine:~$ cd Desktop/Hook
phap@phap-virtual-machine:~/Desktop/Hook$ sudo insmod hook.ko
[sudo] password for phap:
phap@phap-virtual-machine:~/Desktop/Hook$ ./user_test
Open File: Success Write File: Success
```

- Terminal thứ 2:

```
phap@phap-virtual-machine:~$ sudo dmesg -C
[sudo] password for phap:
phap@phap-virtual-machine:~$ dmesg
```

- Kết quả:

```
[ 3669.390034] Process Name: user_test Filename is open: /etc/ld.so.cache
[ 3669.390051] Process Name: user_test Filename is open: /lib/x86_64-linux-gnu/libc.so.6
[ 3669.390200] Process Name: user_test Filename is open: file_test.txt
[ 3669.390220] Process Name: user_test Filename is write: /home/phap/Desktop/Hook/file_test.txt fd: 3 nByte: 16
[ 3669.390242] Process Name: user_test Filename is write: /dev/pts/0 fd: 1 nByte: 39
```

---

## ĐÁNH GIÁ BÀI LÀM

---

- Mức độ hoàn thành: 95%
- Đánh giá:
  - Hai system call pnametoid và pidtoname đều chạy đúng.
  - System call open và write đều hook thành công (ghi tên process mở file, tên file được mở; tên process, tên file và số byte được ghi).
  - Để chạy được phần hook thì kernel *không* cài hai system call pnametoid và pidtoname ở trên. Do phần hook và phần syscall bị xung đột khiến máy bị treo màn hình đen.

---

## NGUỒN THAM KHẢO

---

1. <https://medium.com/anubhav-shrimal/adding-a-hello-world-system-call-to-linux-kernel-dad32875872>
2. <https://stackoverflow.com/questions/53735886/how-to-pass-parameters-to-linux-system-call>
3. <https://github.com/1612198/My-University/tree/master/System%20Call/Sources>
4. <https://github.com/mainguyenanhvu/Syscall Hook Linuxkernel HDH DoAn2>
5. <https://stackoverflow.com/questions/40433827/passing-arguments-to-custom-system-call>
6. <https://medium.com/@lee1003094395/adding-a-system-call-which-can-pass-a-userspace-string-b245105bed38>
7. <https://uwnthesis.wordpress.com/2016/12/26/basics-of-making-a-rootkit-from-syscall-to-hook/>
8. <https://vimentor.com/vi/lesson/thao-tac-file-trong-linux-1>
9. <https://stackoverflow.com/questions/8250078/how-can-i-get-a-filename-from-a-file-descriptor-inside-a-kernel-module>