

VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH UNIVERSITY OF TECHNOLOGY
FACULTY OF APPLIED SCIENCE



HỆ ĐIỀU HÀNH (CO2017)

SYSTEM_CALL

Giảng viên hướng dẫn: La Hoàng Lộc
Sinh viên thực hiện: Nguyễn Đắc Hoàng Phú - 2010514
Du Thành Đạt - 2010206
Phạm Lê Bảo - 2010153

Thành phố Hồ Chí Minh, 4/2022



Mục lục

1	GIỚI THIỆU	2
1.1	Nhóm	2
1.2	Bằng chứng trimming kernel của các thành viên	2
2	Adding new system call	3
2.1	Preparation	3
2.2	Configuration	3
2.3	Build the configured kernel	4
2.4	Installing the new kernel	5
2.5	Trim the kernel	5
3	System call Implementation	7
3.1	Implementation	7
4	Compilation and Installation folder	12
5	Making API for system call	14
5.1	Testing	14
5.2	Wrapper	15
5.3	Validation	16
6	References	19

1 GIỚI THIỆU

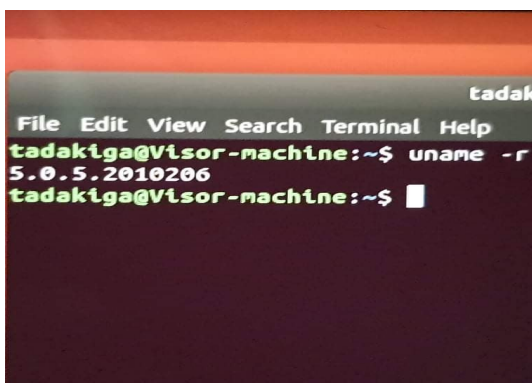
1.1 Nhóm

STT	Họ và Tên	MSSV	Vai Trò	Đóng góp
1	Nguyễn Đắc Hoàng Phú	2010514	Leader	100%
2	Du Thành Đạt	2010206	Member	100%
3	Phạm Lê Bảo	2010153	Member	100%

1.2 Bằng chứng trimming kernel của các thành viên

```
hoangphu7122002@hoangphu7122002-VirtualBox:~$ uname -r  
5.0.5.2010514
```

Hình 1: Hoàng Phú



```
tadakiga@Visor-machine:~$ uname -r  
5.0.5.2010206  
tadakiga@Visor-machine:~$
```

Hình 2: Du Thành Đạt

```
phambao@PhamBao:~$ uname -r  
5.0.5.2010153
```

Hình 3: Phạm Lê Bảo

2 Adding new system call

2.1 Preparation

Đầu tiên, chúng ta cài đặt Ubuntu's Toolchain:

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

Sau đó, chúng ta cài đặt thêm kernel-package:

```
$ sudo apt-get install kernel-package
```

QUESTION 1: Why we need to install kernel-package ?

ANSWER: Gói **kernel-package** giúp chúng ta có thể tự động hóa các bước cần thiết để cài đặt và biên dịch kernel tùy chỉnh. Đặc biệt là đối với người dùng mới, giúp việc xây dựng và cài đặt trở nên dễ dàng hơn rất nhiều so với việc tự cài đặt thủ công. Bên cạnh đó, gói **kernel-package** còn có một số lợi ích như:

- Sự tiện dụng: chỉ cần một câu lệnh **make-kpkg** để thực thi toàn bộ các bước cần thiết cho việc biên dịch kernel.
- Multiple Image Support: cho phép giữ nhiều bản kernel image mà không gây rối.
- Multiple flavour of the same kernel: cho phép quản lý dễ dàng nhiều phiên bản của một kernel mà không cần lo lắng về việc chồng chéo.

Tiếp theo, ta sẽ phải tạo thư mục "kernelbuild" và tải kernel source bằng cách sử dụng command.

```
$ mkdir ~/kernelbuild  
$ cd ~/kernelbuild  
$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.0.5.tar.xz
```

QUESTION 2: Why we have to use another kernel source from the server such as <http://www.kernel.org>, can we compile the original kernel (the local kernel on the running OS) directly?

ANSWER: Chúng ta hoàn toàn có thể biên dịch được kernel gốc trong hệ điều hành đang chạy nếu trên hệ điều hành có file *.c chưa biên dịch. Vì kernel được viết từ ngôn ngữ C, sau khi chúng ta cài compiler gcc thì có thể biên dịch được. Tuy nhiên ở hiện tại thì các bản phân phối tới người dùng bằng file thực thi chứ không chứa mã nguồn khiến việc chỉnh sửa cực kì khó khăn và ảnh hưởng trực tiếp đến máy.

Thay vào đó chúng ta có thể custom bản kernel giống của OS đang chạy bằng cách xem phiên bản và download về từ <https://www.kernel.org/>.

Tiến hành giải nén tệp đã tải về trong directory "kernelbuild" bằng command:

```
$ tar -xvJf linux-5.0.5.tar.xz
```

2.2 Configuration

Bởi việc viết file .config từ đầu sẽ rất phức tạp, ta tiến hành copy file .config sang thư mục linux5.0.5 vừa mới giải nén và cài đặt các package cần thiết. Việc này sẽ giúp dự phòng nếu lỗi có thể xảy ra và tiết kiệm thời gian viết file .config từ đầu. Tiến hành copy sau đó chỉnh sửa trên file .config copy đó sẽ được thực hiện sau:

```
$ cp /boot/config-$(uname -r) ~/kernelbuild/.config
```

Chú ý: Hãy ghi nhớ việc đổi tên phiên bản kernel của bạn. Nếu bỏ qua bước này, chương trình có thể gặp tình trạng ghi đè lên các kernel tồn tại khác. Để chỉnh sửa tệp cấu hình thông qua giao diện terminal, chúng ta cần phải cài đặt một số package cần thiết trước tiên:

```
$ sudo apt-get install fakeroot ncurses-dev xz-utils bc flex libelf-dev bison
```

Sau đó, mở Kernel Configuration.

```
$ make nconfig hoặc $ make menuconfig
```

Để thay đổi phiên bản Kernel, chuyển đến General setup, truy cập dòng "(-ARCH) Local version -append to kernel release". Sau đó, gõ mã số sinh viên của bạn sau dấu ".". ví dụ như sau:

```
.2010514
```

Cuối cùng, nhấn F6 để lưu thay đổi và F9 để thoát.

Lưu ý: Trong quá trình biên dịch (**compile**), bạn có thể gặp lỗi vì không tìm thấy gói (**package**) **openssl**. Lúc này, bạn cần phải cài đặt (**install**) những gói (**package**) đó bằng dòng lệnh (**command**) sau:

```
$ sudo apt-get install openssl libssl-dev
```

2.3 Build the configured kernel

Đầu tiên, chạy lệnh (**command**) "make" để biên dịch (**compile**) nhân (**kernel**) và khởi tạo **vmlinuz**.

```
$ make  
hoặc  
$ make -j 4
```

vmlinuz chính là "**the kernel**". Đặc biệt, đây là **kernel image** sẽ được giải nén và tải vào bộ nhớ bởi GRUB hoặc bất kì bộ tải nào mà bạn sử dụng.

Cuối cùng, xây dựng mô-đun nhân (**loadable kernel modules**).

```
$ make modules  
hoặc  
$ make -j 4 modules
```

QUESTION 3: What is the meaning of these two stages, namely "make" and "make modules"?

ANSWER:

- **make:** Lệnh dùng để biên dịch và liên kết **kernel image**. Kết quả file **vmlinuz** sẽ được khởi tạo.
- **make modules:** Lệnh **make modules** chỉ compile các modules đã được chỉ định trong Makefile, còn các file nhị phân đã được biên dịch thì để lại trong thư mục build.

Khi thực hiện các bước trên xong, tiến hành "sudo reboot" để kiểm tra xem có vô đúng nhân mình đang cài đặt hay không, nếu khi gõ trên cmd: "uname -r" hiển thị ra dòng lệnh: 5.0.5.2010514, tức là đã vô đúng nhân mình đang trim, nếu không ta phải tiến hành một số thay đổi sau:

- Thực hiện cd .. để đưa về thư mục root, sau đó "cd etc/default", sau đó "sudo vim grub" để tiến hành chỉnh sửa một số thông tin trong grub

- Ta tiến hành thay đổi grub để việc tự động hóa quá trình chọn kernel và đăng nhập vào.

```
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT="1>4"
GRUB_TIMEOUT_STYLE="menu"
GRUB_TIMEOUT="2"
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""
```

Hình 4: Thông tin edit lại file grub

- Sau đó "sudo apt-get update grub" và "sudo reboot"
- Trong khi reboot nhấn giữ phím shift và esc liên tục để hiển thị ra bảng chọn nhân kernel, chọn vào nhân chứa MSSV của mình, lưu ý không phải phiên bản có "recovery mode".
- Trong những lần đăng nhập sau, khi tiến hành kiểm tra phiên bản nhân kernel, sẽ tìm được đúng nhân mình đã cài đặt

2.4 Installing the new kernel

Đầu tiên, cài đặt mô-đun (modules), sau đó là nhân (kernel).

```
$ sudo make modules_install
hoặc
$ sudo make -j 4 modules_install
```

```
$ sudo make install
hoặc
$ sudo make -j 4 install
```

Sau khi hoàn tất quá trình trên, khởi động lại máy ảo (**virtual machine**) của bạn.

```
$ sudo reboot
```

Kiểm tra bằng lệnh (command) sau.

```
$ uname -r
```

2.5 Trim the kernel

Với cấu hình đã cài đặt, bạn sẽ cần rất nhiều thời gian để biên dịch (**compile**) và tải (**load**). Trong bài tập lớn này, bạn cần một cấu hình nhân (kernel configuration) khác. Bạn cần phải mở lại "make nconfig" để cấu hình lại máy ảo của mình. Sau đó, chúng ta đã hoàn toàn có thể xây dựng lại nhân (rebuild the kernel) như phần 2.3.

YÊU CẦU:



- Khởi động hệ thống với nhân (**kernel**) mới thành công.
- Nhân (**kernel**) mới phải nhẹ hơn nhân (**kernel**) cũ (boot image nhỏ hơn, ít mô-đun (**module**) phải biên dịch (**compile**) hơn,...).

3 System call Implementation

3.1 Implementation

Đầu tiên, tạo directory `get_folder_stat` để lưu trữ hiện thực của system call. Đồng thời, tạo và hiện thực file `sys_get_folder_stat.c` trong directory đó. Quá trình được thực hiện bằng chuỗi command sau:

```
$ pwd
~/kernelbuild
$ mkdir get_proc_info
$ cd get_proc_info
$ touch sys_get_proc_info.c
```

Dưới đây là file `sys_get_folder_stat.c` đã được hiện thực:

Một số "helper function" sử dụng cho việc đọc ghi file, ánh xạ sang kernel space, và trả thông tin từ kernel_space cho user_space xài cho sys_call

```
1 //open file with path in kernel_space
2 static struct file* get_file_utils(char* kernel_path){
3     struct file* current_file = kmalloc(sizeof(struct file),GFP_KERNEL);
4     current_file = filp_open(kernel_path,0_RDONLY,0);
5
6     return current_file;
7 }
8
9 //map path in user_space to kernel_space
10 static char* map_path_in_kernel(char* path){
11     //PATH_MAX = 4096bytes
12     char* kernel_path = kmalloc(PATH_MAX, GFP_KERNEL); //GFP_KERNEL is flag for KERNEL
13     allocation smaller than page
14
15     long flag = strncpy_from_user(kernel_path,path,PATH_MAX);
16
17     if (path == NULL || flag == -EFAULT){
18         return NULL;
19     }
20     return kernel_path;
21 }
22
23 static void state_copy_user_space(struct folder_stat* info, struct folder_stat* stat){
24     copy_to_user(stat, info, sizeof(struct folder_stat));
25     copy_to_user(&stat->parent_folder, &info->parent_folder,sizeof(struct folder_info));
26     copy_to_user(&stat->folder, &info->folder, sizeof(struct folder_info));
27     copy_to_user(&stat->last_access_child_folder,&info->last_access_child_folder,sizeof(struct
28     folder_info));
29     copy_to_user(&stat->studentID,&info->studentID,sizeof(long));
30 }
```

Còn đây là các Helper Function sử dụng cho việc gán thông tin cho object và lấy thông tin từ các object và in ra các thông tin quan trọng của `folder_info` object để sử dụng cho việc debug.

```
1 static void print_information(const char* text,struct folder_info folder){
2     printk("\n=====n");
3     printk("%s information\n",text);
4     printk("%s name: %s\n",text,folder.name);
5     printk("%s size: %lld\n",text,folder.size);
6     printk("%s atime: %lld\n",text,folder.atime);
7     printk("%s permission: %d\n",text,folder.permission);
8     printk("%s gid: %u\n",text,folder.gid);
9     printk("%s uid: %u\n",text,folder.uid);
10    printk("\n=====n");
11 }
```



```
11 }
12
13 static struct folder_info assign_information(struct dentry* assign_dentry){
14     struct folder_info folder_utils;
15
16     folder_utils.atime = assign_dentry->d_inode->i_atime.tv_sec;
17     folder_utils.gid = assign_dentry->d_inode->i_gid.val;
18     folder_utils.uid = assign_dentry->d_inode->i_uid.val;
19     strcpy(folder_utils.name,assign_dentry->d_name.name,128);
20     folder_utils.permission = assign_dentry->d_inode->i_mode;
21     folder_utils.size = loop_entry_utils(0,assign_dentry);
22
23     return folder_utils;
24 }
25
26 static void state_copy_user_space(struct folder_stat* info, struct folder_stat* stat){
27     copy_to_user(stat, info, sizeof(struct folder_stat));
28     copy_to_user(&stat->parent_folder, &info->parent_folder,sizeof(struct folder_info));
29     copy_to_user(&stat->folder, &info->folder, sizeof(struct folder_info));
30     copy_to_user(&stat->last_access_child_folder,&info->last_access_child_folder,sizeof(struct
31     folder_info));
32     copy_to_user(&stat->studentID,&info->studentID,sizeof(long));
33 }
34
35 static struct folder_info initialize_current_folder(struct inode* current_file_node){
36     struct folder_info current_folder;
37
38     //Assign value for current_folder;
39     current_folder.size = 0;
40     current_folder.permission = (mode_t)current_file_node->i_mode;
41     current_folder.gid = (gid_t)current_file_node->i_gid.val;
42     current_folder.uid = (uid_t)current_file_node->i_uid.val;
43
44     //
45     return current_folder;
46 }
47
48 static struct inode* get_inode_from_file(struct file* current_file){
49     struct inode* current_file_node = kmalloc(sizeof(struct inode), GFP_KERNEL); //state with
50     current_file
51     current_file_node = current_file->f_inode;
52
53     return current_file_node;
54 }
```

Cuối cùng là Helper function và các biến cục bộ sử dụng cho việc tìm size của một folder (với mode = 0 trong tham số) cũng như tìm thư mục con được truy cập cuối cùng từ thư mục hiện tại (mode = 1 trong tham số)

```
1 static struct dentry* save_dir = NULL;
2 static time_t late_access = 0;
3 static long long loop_entry_utils(long mode,struct dentry* dir){
4     struct dentry* current_dentry = NULL;
5     long long size = 0;
6     if (dir == NULL){
7         printk("NULL direction\n");
8         return 0;
9     }
10    list_for_each_entry(current_dentry, &dir->d_subdirs,d_child){
11        if (current_dentry == NULL){
12            break;
13        }
14    }
```

```
13     }
14     if (strlen(current_dentry->d_iname) == 0){
15         continue;
16     }
17     if (current_dentry->d_inode == NULL){
18         continue;
19     }
20     if (mode == 0){
21         if (S_ISDIR(current_dentry->d_inode->i_mode)){
22             size += loop_entry_utils(0,current_dentry);
23         }
24         else {
25             size += (long long)current_dentry->d_inode->i_size;
26         }
27     }
28     else {
29         if (S_ISDIR(current_dentry->d_inode->i_mode)){
30             if (current_dentry->d_inode->i_atime.tv_sec > late_access){
31                 save_dir = current_dentry;
32                 late_access = current_dentry->d_inode->i_atime.tv_sec;
33             }
34         }
35     }
36 }
37 return size;
38 }
```

Cuối cùng là định nghĩa cho System_Call

```
1 SYSCALL_DEFINE2(get_folder_stat, char*, path, struct folder_stat*, stat) {
2     //inode* instance variable
3     struct inode* current_file_node = NULL;
4
5     //dentry* instance
6     struct dentry* current_file_dentry = NULL;
7     struct dentry* subdir_dentry = NULL;
8     struct dentry* parentdir_dentry = NULL;
9
10    //file* instance
11    struct file* current_file = NULL;
12
13    //stat* instance
14    struct folder_stat* info = NULL;
15
16    //folder instance variable
17    struct folder_info current_folder;
18    struct folder_info parent_folder;
19    struct folder_info latest_access_child_folder;
20
21    char* kernel_path = map_path_in_kernel(path);
22    if (kernel_path == NULL){
23        printk("path is NULL\n");
24        return EINVAL; //Null Address
25    }
26
27    current_file = get_file_utils(kernel_path);
28    //Macro for check error_value
29    if (IS_ERR(current_file)){
30        printk("file is error when opening\n");
31        return ENOENT; //ERROR_NO_ENTITY
32    }
33    current_file_node = get_inode_from_file(current_file);
```

```
34     current_folder = initialize_current_folder(current_file_node);
35
36     current_file_dentry = current_file->f_path.dentry;
37     strncpy(current_folder.name, current_file_dentry->d_name.name,128);
38     current_folder.size = loop_entry_utils(0,current_file_dentry);
39
40     loop_entry_utils(1,current_file_dentry);
41     subdir_dentry = save_dir;
42     if (subdir_dentry != NULL){
43         latest_access_child_folder = assign_information(subdir_dentry);
44     }
45     if (subdir_dentry == NULL){
46         printk("====NO SUBDIR FOLDER====\n");
47     }
48     else {
49         print_information("last access child folder",latest_access_child_folder);
50     }
51
52     parentdir_dentry = current_file_dentry->d_parent;
53     if (parentdir_dentry != NULL){
54         parent_folder = assign_information(parentdir_dentry);
55     }
56
57     print_information("current_folder",current_folder);
58     if (parentdir_dentry == NULL){
59         printk("====NO PARENT FOLDER====\n");
60     }
61     else {
62         print_information("parent_folder",parent_folder);
63     }
64
65
66     filp_close(current_file,0);
67     info = kmalloc(sizeof(struct folder_stat), GFP_KERNEL);
68     copy_from_user(info, (struct folder_stat *)stat,sizeof(struct folder_stat));
69
70     info->studentID = 2010514;
71     info->folder = current_folder;
72     info->last_access_child_folder = latest_access_child_folder;
73     info->parent_folder = parent_folder;
74
75     state_copy_user_space(info,stat);
76     //free
77     //=====
78     kfree(kernel_path);
79     kfree(info);
80     kfree(current_file_node);
81     //=====
82
83     return 0;
84 }
```

Ý tưởng:

- + Đầu tiên System_call này sẽ kiểm tra xem path từ người dùng nhập có hợp lệ hay không và tồn tại hay không, nếu không thỏa sẽ trả về "EINVAL" (Lỗi NULL ADDRESS), nếu thỏa trả về path được ánh xạ sang kernel_space
- + Tiếp đến ta tiến hành mở file từ kernel_path đã có và khởi tạo thông tin cơ bản cho folder nhờ thông tin có được từ path, ta sẽ quan tâm thông tin được cung cấp từ các dentry object (chứa link đến thư mục cha và thư mục con), inode object (chứa thông tin hiện tại của folder đang được truy cập).

- + Sau đó ta sử dụng hàm `loop_entry_utils` với `model` để tìm size cho thư mục được cung cấp thông tin từ `path` (`current_folder`) với `mode = 0`, tiếp tục ta sử dụng `mode = 1` để tìm `subdir` gần nhất của nó. Nếu `subdir` khác `NULL` ta tiến hành in các thông tin từ trong thư mục đó, tương tự với thư mục cha của `current_folder` (`parent_folder`)
- + Sau đó thực hiện việc khởi tạo một biến `folder_stat` info để lấy thông tin từ `kernel_space` chuyển sang `user_space`
- + Giải biến các biến đã được cấp phát động để tránh hiện tượng `memory_leak`

Tiếp theo, ta tạo Makefile cho source code:

```
$ pwd
~/kernelbuild/get_folder_stat
$ touch Makefile $ echo "obj-y : += sys_get_folder_stat.o"
```

Ta thêm 4 dòng sau để tăng tốc quá trình compile bằng cách thay vì kiểm xem trong chương trình có file `sys_get_folder_stat.c` thì đánh dấu là đã có chương trình đó rồi

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

4 Compilation and Installation folder

Ta add `get_folder_stat` trong **kernel Makefile**.

```
$ pwd
~/kernelbuild/get_folder_stat
$ cd ..
$ pwd
~/kernelbuild
$ vim Makefile
```

Tìm và add `get_folder_stat` vào cuối line

```
core -y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ get_folder_stat/
```

Tiếp theo, ta add system call mới vào system call table:

```
$ pwd
~/kernelbuild
$ cd arch/x86/entry/syscalls/
$ echo "548 common get_folder_stat __x64_sys_get_folder_stat" » syscall_64.tbl
```

QUESTION 4: What is the meaning of each fields in the line which we just add to the system call table (548, 64, `get_folder_stat`, i.e.)? Can we use other options for this line?

ANSWER: Trong kernel sẽ có một bảng system call (**system call table**), bao gồm tất cả các system call và các implementing function của chúng. Mỗi dòng của bảng này sẽ là cấu trúc bao gồm các thông tin: **<number>** **<abi>** **<name>** **<entry point>**. Trong đó:

- **<number>**: Là **548** trong chuỗi. Đây là mã gọi, cũng như số thứ tự của system call, mỗi số ứng với mỗi syscall là độc nhất.
- **<abi>**: application binary interface, tương ứng với **64** trong chuỗi. Ngoài ra còn có thể mang giá trị **32** ứng với kiến trúc x32.
- **<name>**: Là **get_folder_stat** trong chuỗi. Đó là tên của system call mới, được dùng trong user-space.
- **<entry point>**: Là **sys_get_folder_stat** trong chuỗi. Đây cũng là tên của system call, nhưng được dùng bởi kernel space.

Như đã nói, mỗi dòng của system call table đều mang cấu trúc này. Vì vậy khi khai báo một system call mới cho kernel, phải dùng đúng cấu trúc nêu trên. Tuy nhiên, các giá trị và thông số có thể thay đổi sao cho phù hợp. Ví dụ **<abi>** có thể mang các giá trị như **32**, **64** hay **"common"**. Khi thực hiện add system call vào system call table, quy trình thực hiện thêm như sau:

```
1 echo "548 common get_folder_stat __x64_sys_get_folder_stat" >> syscall_64.tbl
```

Mở `syscall.h` và thêm những dòng sau trước `#endif` để thêm system call mới.

```
struct folder_info;
struct folder_stat;
asmlinkage long sys_get_folder_stat (char* path, struct folder_stat * stat);
```

QUESTION 5: What is the meaning of each line above ?

ANSWER:

- **folder_info**: Khai báo sự xuất hiện của cấu trúc `folder_info` dùng để lưu trữ thông tin thống kê của folder.

- **folder_stat**: Khai báo sự xuất hiện của cấu trúc `folder_stat` dùng để lưu trữ thông tin của từng folder.
- `asmlinkage long sys_get_folder_stat (char* path, struct folder_stat * stat)`: Prototype hiện thị thông tin về tham số, kiểu trả về. **asmlinkage** thông báo cho compiler biết tham số được lưu trong stack.

Sau khi hoàn thành các bước trên, ta tiến hành biên dịch và cài đặt lại kernel mới:

```
$ make -j 4
$ make modules -j 4
$ sudo make modules_install
$ sudo make install
$ sudo reboot
```

5 Making API for system call

5.1 Testing

Tạo một chương trình nhỏ mới nhằm kiểm tra xem system call mới đã được thêm vào kernel chưa. Ta định nghĩa file test.c với nội dung sau đây. Ta sẽ sử dụng path = "/home/hoangphu7122002" để test.

```
1 #include <sys/syscall.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #define SIZE 200
5 int main(){
6     long sys_return_value;
7     unsigned long stat[SIZE];
8     sys_return_value = syscall(548, NULL, &stat);
9     printf("MSSV %lu\n", stat[0]);
10    return 0;
11 }
```

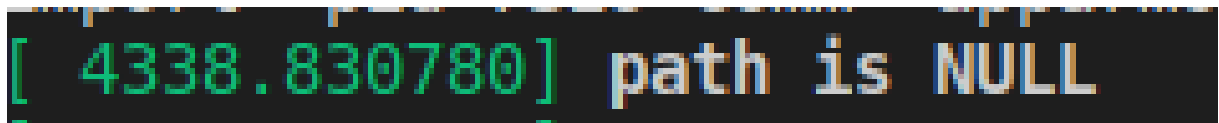
QUESTION 6: Why this program could indicate whether our system call works or not ?

ANSWER:

- + Khi compile và chạy chương trình, do chúng ta chưa có cách nào để truyền path vào test, và trong tham số thứ hai của syscall do là NULL nên sẽ hiểu path ở đây là NULL, chương trình sẽ không thực hiện đúng như kì vọng.
- + Đây là kết quả của đoạn chương trình trên, nó sẽ in ra một MSSV bất kì chứ không phải 2010514 như đã cài đặt ở trên. Ta sẽ in ra file log quá trình trong kernel_space bằng lệnh sudo dmesg

```
[Running] cd "/home/hoangphu7122002/test/" && gcc test.c -o test && "/home/hoangphu7122002/t
MSSV 1825868
```

Hình 5: MSSV không phải là 2010514



Hình 6: log path hiển thị đường dẫn là NULL nên chưa hoạt động đúng như kì vọng

Ta tiến hành tinh chỉnh lại chương trình để có thể truyền được biến path vào trong chương trình

```
1 #include <sys/syscall.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #define SIZE 200
5 int main(int argc, char* argv[]){
6     long sys_return_value;
7     unsigned long stat[SIZE];
8     sys_return_value = syscall(548, argv[1], &stat);
9     printf("MSSV %lu\n", stat[0]);
10    return 0;
11 }
```

- + Ta tiến hành compile ra file test.exe, sau đó truyền path bằng cách: `./test /home/hoangphu7122002`. Lúc này chương trình hoạt động và kết quả như sau:

```
hoangphu7122002@hoangphu7122002-VirtualBox:~/test$ ./test /home/hoangphu7122002
MSSV 2010514
```

Hình 7: MSSV là 2010514

```
[ 4409.814192] last access child folder information
[ 4409.814192] last access child folder name: test
[ 4409.814193] last access child folder size: 31646
[ 4409.814193] last access child folder atime: 1648977468
[ 4409.814194] last access child folder permission: 16893
[ 4409.814194] last access child folder gid: 1000
[ 4409.814194] last access child folder uid: 1000
[ 4409.814195]
=====
[ 4409.814685]
=====
[ 4409.814686] current_folder information
[ 4409.814687] current_folder name: hoangphu7122002
[ 4409.814687] current_folder size: 2799911407
[ 4409.814688] current_folder atime: 57
[ 4409.814688] current_folder permission: 16877
[ 4409.814688] current_folder gid: 1000
[ 4409.814689] current_folder uid: 1000
[ 4409.814689]
=====
[ 4409.814689]
=====
[ 4409.814689] parent_folder information
[ 4409.814690] parent_folder name: home
[ 4409.814690] parent_folder size: 2799911407
[ 4409.814690] parent_folder atime: 1648875464
[ 4409.814691] parent_folder permission: 16877
[ 4409.814691] parent_folder gid: 0
[ 4409.814691] parent_folder uid: 0
[ 4409.814692]
=====
```

Hình 8: log path hiển thị thông tin của các thư mục cha, con gần nhất và thư mục hiện tại

5.2 Wrapper

QUESTION 7: Why do we have to re-define the `folder_stat` and `folder_info` structs while we have already defined it inside the kernel?

ANSWER: Vì khi định nghĩa các cấu trúc này trong kernel, ta phải gọi system call qua số định danh của chúng, gây khó khăn khi lập trình. Do đó, ta tái định nghĩa lại `folder_stat` và `folder_info` trong wrapper để lưu như file header. Từ đó, các chương trình khác khi muốn sử dụng chỉ cần thêm file header `get_folder_stat.h` để compiler biết được các cấu trúc này có tồn tại và sử dụng một cách thuận tiện.

Ta hiện thực wrapper lại file như sau:

- + Đầu tiên, định nghĩa lại header file để chương trình khác có thể sử dụng file này.

```
1 #ifndef _GET_FOLDER_STAT_H
2 #define _GET_FOLDER_STAT_H
3
4 #include <unistd.h>
5 #include <sys/stat.h>
6 #include <dirent.h>
7 #include <stdlib.h>
8 #include <string.h>
```



```
9
10 struct folder_info {
11     char name[128];
12     mode_t permission;
13     uid_t uid;
14     gid_t gid;
15     double size;
16     time_t atime;
17 };
18
19 struct folder_stat {
20     long studentID;
21     struct folder_info folder;
22     struct folder_info parent_folder;
23     struct folder_info last_access_child_folder;
24 };
25
26 long get_folder_stat(char* path, struct folder_stat * info);
27 #endif
```

+ Sau đó ta định nghĩa lại hàm `get_folder_stat` được ghi trong `header_file`, ta sử dụng lại lệnh thứ 548 trong `syscalls.h`

```
1 #include "get_folder_stat.h"
2 #include <linux/kernel.h>
3 #include <sys/syscall.h>
4 #include <unistd.h>
5
6 long get_folder_stat(char* path, struct folder_stat * info){
7     long sys_return_value = syscall(548, path, info);
8     return sys_return_value;
9 }
```

5.3 Validation

Tạo ra file `.h`, ta lưu vào thư viện headerfile bằng câu lệnh sau đây với `path` là: `/home/hoangphu7122002/kernelbuild/linux-5.0.5/get_folder_stat`

```
1 $ sudo cp <path to get_folder_stat.h> /usr/include
```

Bạn cần phải kiểm tra mọi thứ bằng một mô-đun kiểm tra bổ sung (additional test module) để gọi hàm (function) này nhưng không bao gồm phần kiểm tra file gốc.

QUESTION 8: Why root privilege (e.g. adding `sudo` before the `cp` command) is required to copy the header file to `/usr/include`?

ANSWER: Lệnh **sudo** (super-user do) cho phép thực hiện thay đổi sâu trên hệ thống (chẳng hạn quản lý gói, cài đặt phần mềm, ...). **/usr/include** là folder chứa các tệp header, là nơi mà trình biên dịch sẽ truy cập khi người dùng `include` trong ngôn ngữ C/C++, và cần có quyền cao hơn user tiêu chuẩn (standard user). Do đó, cần thêm **sudo** vào trước lệnh **cp** để cấp quyền có thể thay đổi đối với folder. Câu lệnh để tạo ra file `.so`

```
1 $ gcc -shared -fpic get_folder_stat.c -o libget_folder_stat.so
```

QUESTION 9: Why we must put `-shared` and `-fpic` option into `gcc` command?

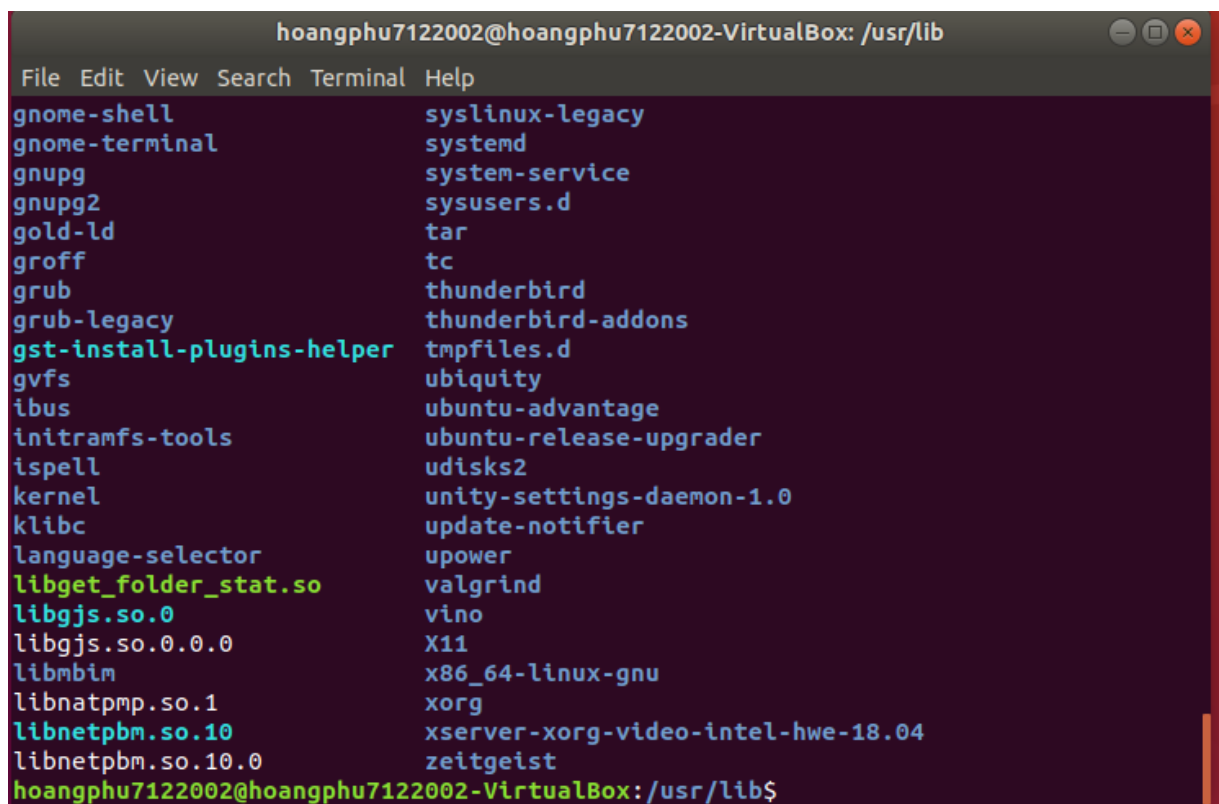
- **-shared:** tạo ra một file dùng chung `.so` (shared object). File object sẽ được dùng chung giữa các file thực thi, giúp giảm kích thước của file thực thi. Điều này cũng đảm bảo sự linh động, ví dụ khi sửa lỗi chỉ cần sửa trên file object này.

- **-fpic**: là biên dịch theo kiểu PIC (Position Independent Code). Điều này có nghĩa là mã máy được tạo ra không phụ thuộc vào việc được đặt tại một địa chỉ cụ thể để hoạt động. Điều này là cần thiết đối với 1 file trong thư viện, bởi nó có thể được gọi ở một thư mục khác với thư mục chứa nó.

Sau đây là demo lưu file .so vào trong thư mục /usr/lib. Lúc này khi muốn sử dụng system_call mới, ta

```
hoangphu7122002@hoangphu7122002-VirtualBox: ~/kernelbuild/linux-5.0.5/get_folder_stat$ ls
a                               libget_folder_stat.so  sys_get_folder_stat.c
built-in.a                     Makefile               sys_get_folder_stat.o
get_folder_stat.c              modules.order
get_folder_stat.h              Module.symvers
```

Hình 9: file .so được lưu trong thư mục get_folder_stat



```
hoangphu7122002@hoangphu7122002-VirtualBox: /usr/lib
File Edit View Search Terminal Help
gnome-shell          syslinux-legacy
gnome-terminal       systemd
gnupg                system-service
gnupg2               sysusers.d
gold-ld              tar
groff                tc
grub                 thunderbird
grub-legacy          thunderbird-addons
gst-install-plugins-helper tmpfiles.d
gvfs                 ubiquity
ibus                 ubuntu-advantage
initramfs-tools      ubuntu-release-upgrader
ispell               udisks2
kernel               unity-settings-daemon-1.0
klibc                update-notifier
language-selector    upower
libget_folder_stat.so valgrind
libgjs.so.0           vino
libgjs.so.0.0.0       X11
libmbim              x86_64-linux-gnu
libnatpmp.so.1        xorg
libnetpbm.so.10       xserver-xorg-video-intel-hwe-18.04
libnetpbm.so.10.0     zeitgeist
hoangphu7122002@hoangphu7122002-VirtualBox: /usr/lib$
```

Hình 10: copy file .so được lưu vào /usr/lib

chỉ cần add thư viện <get_folder_stat.h> , sau đây là demo sử dụng, ta đặt tên file .c là new

```
1 #include <get_folder_stat.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4 #include <stdio.h>
5 #include <stdint.h>
6
7 int main(int argc, char* argv[]){
8     char* path = argv[1];
9     struct folder_stat stat;
10    if(get_folder_stat(path,&stat) == 0){
11        printf("MSSV: %lu",stat.studentID);
```

```

12 }
13 else {
14     printf("path is error\n");
15 }
16 return 0;
17 }

```

+ Compile file và để sử dụng get_folder_stat.h ta thêm đằng sau flag: -l get_folder_stat và sau đó truyền path vào bằng cách dùng: ./test_a /home/hoangphu7122002

```

hoangphu7122002@hoangphu7122002-VirtualBox:~/test$ vim test
hoangphu7122002@hoangphu7122002-VirtualBox:~/test$ touch test_new_function.c
hoangphu7122002@hoangphu7122002-VirtualBox:~/test$ vim test_new_function.c
hoangphu7122002@hoangphu7122002-VirtualBox:~/test$ gcc -o test_a test_new_function.c -l get_folder_stat
hoangphu7122002@hoangphu7122002-VirtualBox:~/test$ ./test_a /home/hoangphu7122002
MSSV 2010514:hoangphu7122002@hoangphu7122002-VirtualBox:~/test$ sudo dmesg

```

Hình 11: compile demo với system_call new folder_stat

+ sau đó xem log file để xem các thông tin phân tích trong thư mục.

```

56.718129] =====
56.718131] last access child folder information
56.718131] last access child folder name: test
56.718132] last access child folder size: 17607
56.718133] last access child folder atime: 1648985427
56.718133] last access child folder permission: 16893
56.718133] last access child folder gid: 1000
56.718134] last access child folder uid: 1000
56.718134] =====
56.718137] =====
56.718137] current_folder information
56.718137] current_folder name: hoangphu7122002
56.718138] current_folder size: 915201
56.718138] current_folder atime: 51
56.718138] current_folder permission: 16877
56.718138] current_folder gid: 1000
56.718139] current_folder uid: 1000
56.718139] =====
56.718139] =====
56.718139] parent_folder information
56.718139] parent_folder name: home
56.718140] parent_folder size: 915201
56.718140] parent_folder atime: 1648875464
56.718140] parent_folder permission: 16877
56.718141] parent_folder gid: 0
56.718141] parent_folder uid: 0
56.718141] =====
hoangphu7122002@hoangphu7122002-VirtualBox:~/test$

```

Hình 12: log path hiển thị thông tin của các thư mục cha, con gần nhất và thư mục hiện tại

6 References

- [Tham khảo cho loop_entry_utils](#)
- [Tham khảo cách cop từ user_space qua kernel_space](#)
- [Tham khảo lấy thông tin từ kernel_space qua user_space](#)
- [dentry_object](#)
- [inode_object](#)
- [Map in kernel_space](#)