

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
Khoa học - Kỹ thuật Máy tính



HỆ ĐIỀU HÀNH

Group: NNHTV

ASSIGNMENT #1: SYSTEM CALL

GVHD: Hoàng Lê Hải Thanh

SV thực hiện: Huỳnh Quang Huy – 1812355
Trương Hoài Nam – 1813175
Nguyễn Hồng Nhân – 1813330
Trần Văn Tâm – 1713057
Hoàng Vương – 1811345

TP. HỒ CHÍ MINH, THÁNG 05/2020



Mục lục

1	Description of member's Work	2
2	Adding new System Call	2
2.1	Preparation	2
2.2	Configuration	3
2.3	Build the configured kernel	3
2.4	Installing the new kernel	4
3	System call Implementation	5
4	Compilation and Installation process	6
5	Making API for system call	6
5.1	Testing	6
5.2	Wrapper	8
5.3	Validation	8

1 Description of member's Work

Member	Description work
Huỳnh Quang Huy	Build kernel, Implement wrapper
Trương Hoài Nam	Build kernel, Report
Nguyễn Hồng Nhân	Implement get_proc_info, Answer questions: 456
Trần Văn Tâm	Answer questions: 123
Hoàng Vương	Build kernel, Implement get_proc_info, Answer questions: 789

2 Adding new System Call

2.1 Preparation

- Cài đặt máy ảo Ubuntu-18.04-LTS và phần mềm VMware Workstation 15/ Oracle VM VirtualBox.
- Cài đặt build-essential package và kernel-package:

```
1 $ sudo apt-get update
2 $ sudo apt-get install build-essential
3 $ sudo apt-get install kernel-package
```

- Tạo thư mục build kernel, tải kernel 5.0.5 và giải nén; sau đó sao chép toàn bộ file trong thư mục *kernelbuild/linux-5.0.5* đến thư mục *~/kernelbuild*:

```
1 $ mkdir ~/kernelbuild
2 $ cd ~/kernelbuild
3 $ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.0.5.tar.xz
4 $ tar -xvJf linux-5.0.5.tar.xz
```

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

Answer:

- *Kernel - package* là gói các tiện ích, tool (bên cạnh build-essential) cho quá trình compile và build kernel. Có các loại kernel-package như sau:

- + *linux - generic* (for Desktop).
- + *linux - headers - generic* (cung cấp files để build out-of-tree kernel modules và chương trình từ source).
- + *linux - image - generic* (tải lên kernel image và các modules liên quan khi OS khởi động).
- + *linux - server* (for Server).
- + *linux - virtual* (chứa các drivers để chạy trong công nghệ ảo hóa như VMware, Virtual Box,...).

- Dùng *kernel - package* sẽ hỗ trợ user dùng nhiều version của kernel trên 1 máy vì các package trên sẽ có nhiều version khác nhau tương ứng với các version của kernel; OS sẽ giữ lại các version cũ của package phòng khi cài đặt kernel mới nhưng gây error, crash hệ thống thì còn có package cũ của kernel cũ lấy ra dùng.

- Ngoài ra, để thời gian compile nhanh hơn thì ta cần cài đặt các kernel-package, nó sẽ tự động script các step để compile thay vì compile kernel 1 cách thủ công, ta phải làm theo trình tự từng step-by-step.

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:

- Theo mục đích của bài assignment, sinh viên phải tự modify, config, customize kernel theo ý mình nên ta cần dùng 1 kernel source, nếu dùng kernel hiện tại trên OS thì ta có thể làm hỏng các chức năng của kernel, khiến cho

OS không còn hoạt động trơn tru. Bên cạnh đó, ta còn thêm 1 system call mới (know the parent and a child of a given process) vào trong kernel do ta tự implement nghĩa là làm thay đổi hoạt động của kernel trên hệ thống, nên không được dùng local kernel để compile được.

- Kernel được cấp trên OS hiện tại là dạng pre-compiled, là các file trên kernel đã được compiled, nên ta không thể dùng kernel này để compile lại như source code được nên càng phải tải kernel source từ server về, nếu như không muốn tải kernel source về thì phải backup các file trong kernel hiện hành để tránh các bug xảy ra khi compile trên kernel hệ thống.

2.2 Configuration

• Do cấu hình kernel được cài trong trong tệp .config của nó, chứa tác dụng của các module kernel. Để tự tạo quá trình cấu hình, ta sao chép tệp trong thư mục /boot/ vào thư mục kernel build:

```
1 $ cp /boot/config-$(uname -r) ~/kernelbuild/.config
```

• Cài đặt gói thiết yếu để cấu hình lại tệp trên giao diện của terminal:

```
1 $ sudo apt-get install fakeroot ncurses-dev xz-utils bc flex libelf-dev bison
```

• Mở Kernel Configuration và thay đổi version kernel bằng:

```
1 $ made nconfig
2 Access to theline General setup -> (-ARCH) Local version - append to kernel release -> Enter ".1811345" -> Save and Exit
```

• Cài đặt gói openssl tránh lỗi trong quá trình biên dịch:

```
1 $ sudo apt-get install openssl libssl-dev
```

2.3 Build the configured kernel

• Trim kernel sau đó biên dịch kernel bằng lệnh "make" nhằm tạo vmlinuz, cho máy ảo 4 process và build các module:

```
1 $ make localmodconfig
2 $ make -j 4
3 $ make -j 4 modules
```

Question 3: What is the meaning of these two stages, namely "make" and "make modules"? What are created and what for?

Answer:

- Lệnh "make" sẽ tạo ra file vmlinuz, là binary file, là 1 kernel image, là file nén và nó sẽ được giải nén và load vào trong bộ nhớ bởi GRUB hay boot loader khác, được dùng để compile và liên kết đến kernel image.

- Lệnh "make modules": kernel modules là các phần chức năng của kernel được chia ra, nó sẽ tạo ra các individual files, thích hợp cho các developer muốn apply changes lên nó mà không phải dùng toàn bộ kernel, các modules này có thể là 1 phần code được load hoặc chưa load vào kernel lệnh. Chúng có thể mở rộng chức năng của kernel mà không cần phải reboot hệ thống, dùng cho việc để build, compile các kernel modules.

2.4 Installing the new kernel

- Cài đặt các module và kernel:

```
1 $ sudo make -j 4 modules_install
2 $ sudo make -j 4 install
```

- Xem phiên bản hiện tại của kernel bằng lệnh:

```
1 $ sudo reboot
2 $ uname -r
```

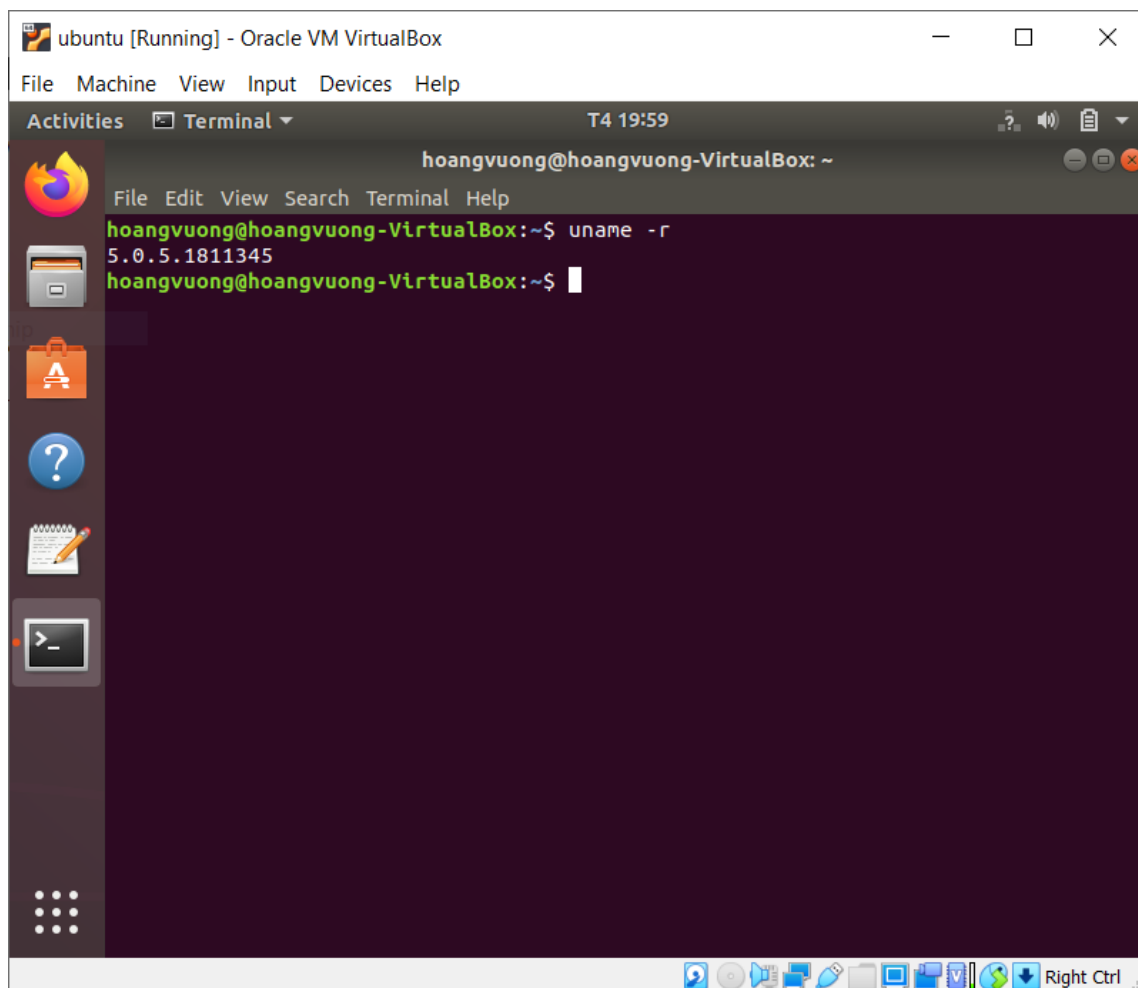
- Trường hợp không hiện phiên bản kernel, khắc phục như sau:

```
1 $ sudo gedit /etc/default/grub
```

- Thay đổi GRUB_TIMEOUT thành -1 và comment GRUB_TIMEOUT = hidden.

```
1 $ sudo update-grub
2 $ sudo reboot
3 $ uname -r
```

- Kết quả thu được:



3 System call Implementation

- Tạo file chương trình `sys_get_proc_info.c` và hiện thực:

```
1 $ cd ~/kernelbuild/  
2 $ mkdir get_proc_info  
3 $ cd get_proc_info  
4 $ touch sys_get_proc_info.c
```

- Tạo Makefile sau khi đã implement xong `sys_get_proc_info.c` bằng cách thực hiện các lệnh:

```
1 $ pwd ~/kernelbuild/get_proc_info  
2 $ touch Makefile  
3 $ echo "obj-y := sys_get_proc_info.o"
```

- Thêm `get_proc_info` vào Makefile của kernel bằng cách vào kernel Makefile, tìm đến dòng:

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

và thêm `get_proc_info/` vào cuối dòng trên. Sau đó truy cập đến file [syscall_64.tbl](#) trong thư mục `syscalls` để thêm dòng:

```
1 548 64 get_proc_info _x64_sys_get_proc_info
```

Question 4: What is the meaning of fields of the line that just add to the system call table (548, 64, `get_proc_info`, i.e.)

Answer:

Mỗi system call được định nghĩa với mỗi hàng, trong đó theo thứ tự là các đối số `<number>` `<abi>` `<name>` `<entry point>` với ý nghĩa:

- `<number>`: Định danh riêng cho từng syscall, cần cung cấp đối số này khi thực hiện lời gọi syscall để hệ thống tìm đúng lời gọi tương ứng.

- `<abi>`: (Application Binary Interface) mang giá trị:

+ Hệ máy 32-bit (file `syscall_32.tbl`): "i386"

+ Hệ máy 64-bit (file `syscall_64.tbl`): "common", "64" hoặc "x32"

- `<name>`: Tên gọi của system call, ví dụ: read, write, open, close, fork, ...

- `<entry point>`: điểm bắt đầu của lời gọi hàm thực thi tác vụ.

- Tiếp tục mở file `"syscalls.h"` và thêm các dòng sau vào sau `#endif`.

```
1 struct proc_info;  
2 struct procinfo;  
3 asmlinkage long sys_get_proc_info(pid_t pid, struct procinfo * info);
```

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

Answer:

```
1 struct proc_info;  
2 struct procinfos;
```

- Cả 2 dòng đều là khai báo thông thường struct trong file header.

```
1 asmlinkage long sys_get_proc_info(pid_t pid, struct procinfos * info);
```

- Khai báo 1 hàm asmlinkage trong file header, dùng từ khóa asmlinkage để trình biên dịch tìm các đối số hàm trên stack của CPU thay vì trên thanh ghi như bình thường.

- System call giúp userspace gọi request tới kernel để thực hiện công việc (nằm trong kernel space). Công việc này đòi hỏi các hàm không như bình thường, các đối số truyền qua program stack, không phải thanh ghi như thông thường.

4 Compilation and Installation process

- Truy cập vào thư mục `~/kernelbuild`, chạy các lệnh sau:

```
1 $ make -j 4  
2 $ make modules -j 4
```

- Cài đặt modules, kernel và reboot:

```
1 $ sudo make modules_install  
2 $ sudo make install  
3 $ sudo reboot
```

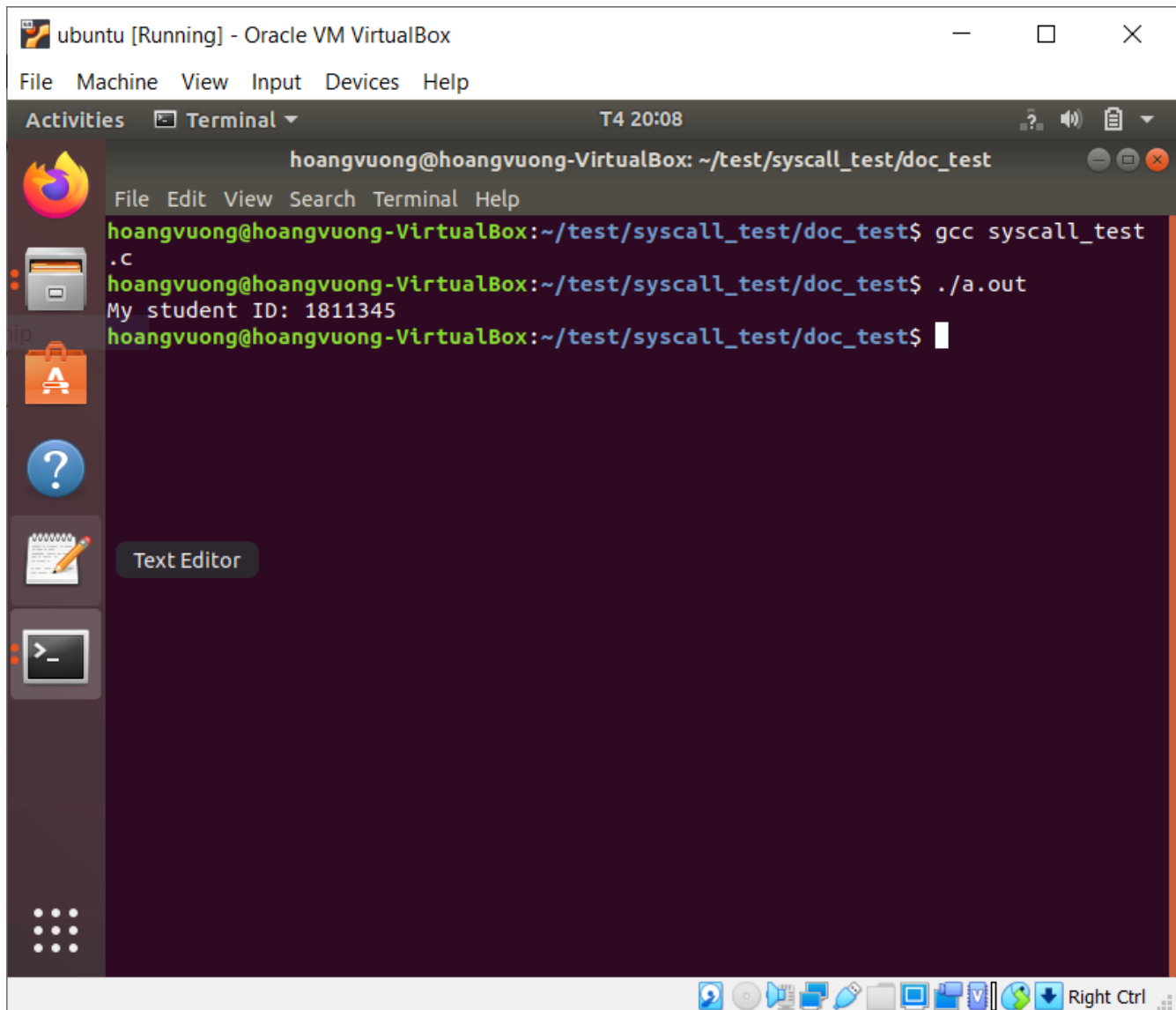
5 Making API for system call

5.1 Testing

- Tạo file `syscall_test.c` có nội dung sau:

```
1 #include <sys/syscall.h>  
2 #include <stdio.h>  
3 #include <unistd.h>  
4  
5 #define SIZE 200  
6  
7 int main(){  
8     long sys_return_value;  
9     unsigned long info[SIZE];  
10    sys_return_value = syscall(548, -1, &info);  
11    printf("My student ID: %lu\n", info[0]);  
12    return 0;  
13 }
```

- Biên dịch `syscall_test.c` và thu được kết quả đúng như hình sau:



```
hoangvuong@hoangvuong-VirtualBox: ~/test/syscall_test/doc_test
hoangvuong@hoangvuong-VirtualBox:~/test/syscall_test/doc_test$ gcc syscall_test.c
hoangvuong@hoangvuong-VirtualBox:~/test/syscall_test/doc_test$ ./a.out
My student ID: 1811345
hoangvuong@hoangvuong-VirtualBox:~/test/syscall_test/doc_test$
```

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

Answer:

Thông qua thư viện `<sys/syscall.h>` sử dụng lệnh `syscall(<number>,<pid>,&info)`. Với `number` là định danh riêng của system call ta đã khai báo là 548. Nếu system call của chúng ta không hoạt động, chương trình sẽ không hoạt động. Ngược lại, `syscall` sẽ trả về giá trị trong con trỏ `info` và xuất ra bởi lệnh `printf` bên dưới.

Vì vậy, chương trình này có thể xác định system call của chúng ta có hoạt động hay không.

5.2 Wrapper

- Tạo file header "*get_proc_info.h*" chứa các prototype của wrapper và khai báo hai cấu trúc *procinfos* và *proc_info*.

Question 7: Why we have to redefine *procinfos* and *proc_info* struct while we have already defined it inside the kernel?

Answer:

Mặc dù struct *procinfos* đã được định nghĩa, nhưng nó chỉ được sử dụng trong kernel space, không phải user space để người dùng có thể sử dụng được. Vì vậy ta cần phải định nghĩa lại struct *procinfos* để người dùng thuận lợi trong việc sử dụng sau này.

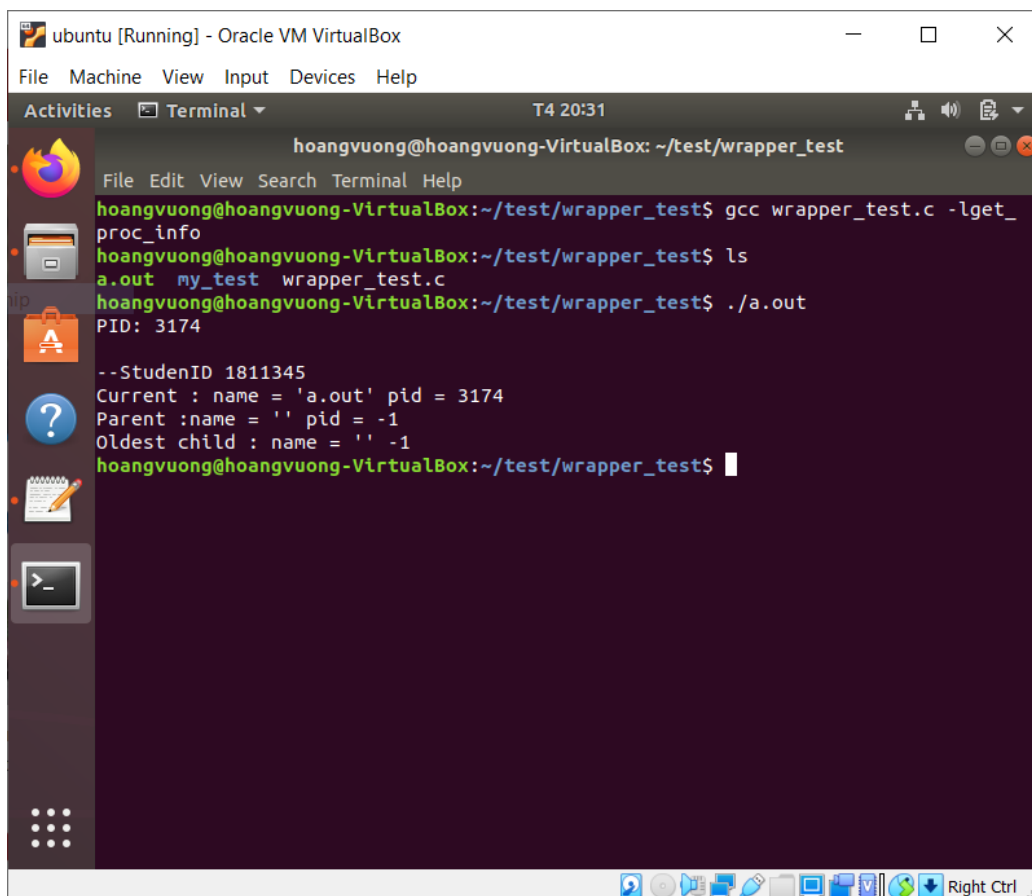
5.3 Validation

- Thêm thư viện (file *get_proc_info.h*) vào kernel và đảm bảo chúng đã sẵn sàng có thể truy cập.

```
1 $ sudo cp <path to get_pro_info.h> /usr/include
2 $ gcc -shared -fpic get_proc_info.c -o libget_proc_info.so
3 $ sudo cp <path to libget_proc_info.so> /usr/lib
```

- Kiểm tra lại tính đúng đắn của chương trình đã hiện thực, bằng việc test với file *wrapper_test.c*:

```
1 $ gcc wrapper_test.c -lget_proc_info
2 $ ./a.out
```



```
ubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal T4 20:31
hoangvuong@hoangvuong-VirtualBox: ~/test/wrapper_test
File Edit View Search Terminal Help
hoangvuong@hoangvuong-VirtualBox:~/test/wrapper_test$ gcc wrapper_test.c -lget_
proc_info
hoangvuong@hoangvuong-VirtualBox:~/test/wrapper_test$ ls
a.out my_test wrapper_test.c
hoangvuong@hoangvuong-VirtualBox:~/test/wrapper_test$ ./a.out
PID: 3174
--StudenID 1811345
Current : name = 'a.out' pid = 3174
Parent :name = '' pid = -1
Oldest child : name = '' -1
hoangvuong@hoangvuong-VirtualBox:~/test/wrapper_test$
```



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

Answer:

Thư mục /usr thuộc sở hữu của user root, người dùng bình thường không thể sao chép vào thư mục này. Vì vậy, ta buộc phải dùng lệnh sudo (super user do) cho phép chạy lệnh như user root.

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

Answer:

Khi biên dịch -shared và -fPIC đều quan trọng để tạo ra filename.so (share object). File này được liên kết tới các chương trình có sử dụng thư viện đã viết trên. Trong khi -fPIC sẽ tạo ra “position-independent code” (PIC) phù hợp để sử dụng trong share library (nếu máy tính có hỗ trợ) thì -shared sẽ tạo ra một shared library.



Tài liệu

1. Robert Love, "*Linux Kernel Development: Linux Kernel Development _p3 3rd Edition*".
2. Unable to add a custom system call on x86 ubuntu linux, <https://stackoverflow.com/questions/54840772/unable-to-add-a-custom-system-call-on-x86-ubuntu-linux>
3. Adding a New System - The Linux Kernel documentation, <https://www.kernel.org/doc/html/v4.12/process/adding-syscalls.html>
4. sched.h - include/linux/sched.h - Linux source code (v5.6.13) - Bootlin , <https://elixir.bootlin.com/linux/latest/source/include/linux/sched.h>