

VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE & ENGINEERING



HỆ ĐIỀU HÀNH

ASSIGNMENT 1 SYSTEM CALL

Teacher: Nguyễn Quang Hùng
Teacher Assistant: Hoàng Lê Hải Thanh
Class: L04, Group: 04
Members: Võ Hùng - 2013375
Trần Quốc Thái - 2010616
Đào Đức Thiện - 1713287
Bùi Hoàng Minh - 2010410

Ho Chi Minh City, 10/2021



Mục lục

1	Biên dịch Linux Kernel	2
1.1	Chuẩn bị	2
1.2	Cấu hình	2
1.3	Xây kernel đã cấu hình	2
1.4	Cài đặt kernel mới	3
2	Trim the kernel	4
3	System Call	4
3.1	Thêm system call mới	4
3.2	Hiện thực system call	5
3.3	Quá trình biên dịch và cài đặt	6
3.4	Tạo API cho system call	7

1 Biên dịch Linux Kernel

1.1 Chuẩn bị

Cài Ubuntu's Toolchain bằng cách tải về build-essential metapackage:

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

Cài đặt kernel-package:

```
1 $ sudo apt-get install kernel-package
```

QUESTION: Why we need to install kernel-package?

Trả lời: Gói kernel-package giúp chúng ta tự động hóa các bước thông thường cần để biên dịch và cài đặt một custom kernel.

Tạo thư mục chứa kernel biên dịch:

```
1 $ mkdir kernelbuild
```

Di chuyển đến thư mục vừa tạo và tải kernel source:

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

QUESTION: 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?

Trả lời: Chúng ta có thể biên dịch kernel gốc của hệ điều hành. Tuy nhiên việc sử dụng kernel source tải từ các server sẽ đảm bảo kernel cùng phiên bản so với hướng dẫn, tránh các vấn đề về tương thích nếu chúng ta sử dụng kernel mới hoặc cũ hơn.

Giải nén file kernel:

```
1 $ tar -xvJf linux-5.0.5.tar.xz
```

1.2 Cấu hình

Copy file cấu hình của hệ điều hành sang thư mục linux-5.0.5:

```
1 $ cp /boot/config-$(uname -r) ~/kernelbuild/linux-5.0.5/.config
```

Để chỉnh sửa file cấu hình thông qua giao diện terminal, ta cần cài đặt các gói cần thiết:

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

Mở Kernel Configuration:

```
1 $ make nconfig
```

Đổi tên của phiên bản kernel trong General setup -> Local version. Trong khi biên dịch có thể gặp lỗi missing openssl packages, ta cần cài đặt các packages sau:

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

1.3 Xây kernel đã cấu hình

Biên dịch kernel và tạo vmlinux:

```
1 $ make
```

Xây dựng loadable kernel modules:

```
1 $ make modules
```

QUESTION: What is the meaning of these two stages, namely “make” and “make modules”? What are created and what for?

Trả lời: make biên dịch kernel và các modules theo cấu hình, tạo ra file vmlinuz . Còn make modules chỉ biên dịch các modules được chọn trong cấu hình.

1.4 Cài đặt kernel mới

Cài đặt modules:

```
1 $ sudo make modules_install
```

Cài đặt kernel mới:

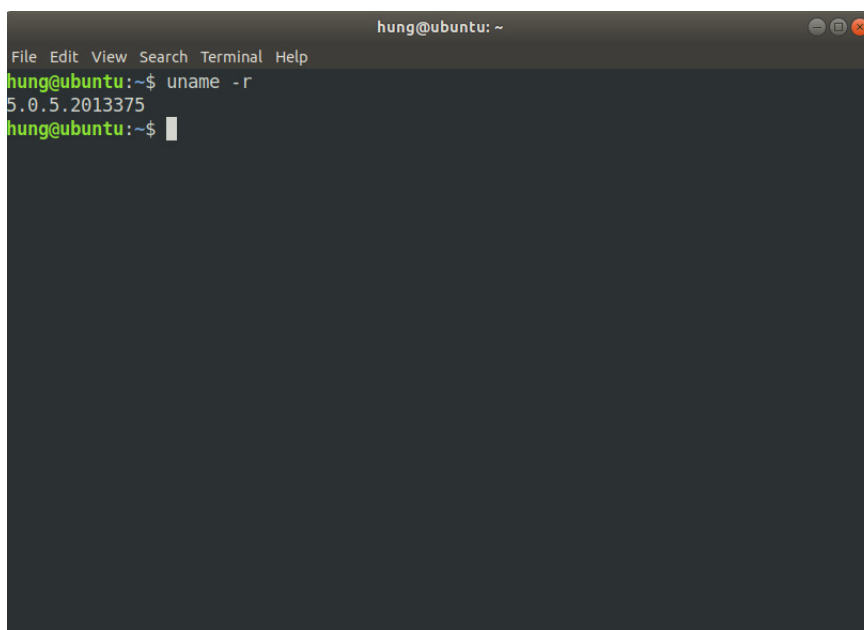
```
1 $ sudo make install
```

Khởi động lại máy ảo:

```
1 $ sudo reboot
```

Kiểm tra kernel mới đã được biên dịch và cài đặt thành công:

```
1 $ uname -r
```



Hình 1: Kernel mới có đuôi .MSSV

Lưu ý: Trong khi chạy có thể gặp lỗi:

```
1 make[2]: *** No rule to make target 'debian/certs/benh@debian.org.cert.pem', needed by 'certs/x509_certificate_list'. Stop.
2 Makefile:951: recipe for target 'certs' failed
3 make[1]: *** [certs] Error 2
```

Hãy sử dụng file .config dòng:

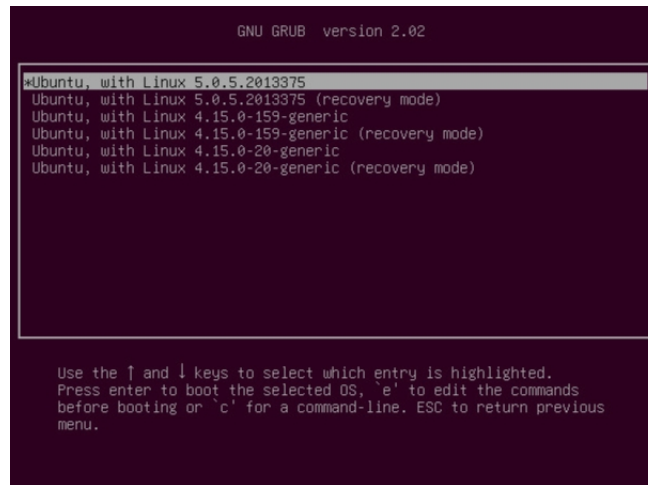
```
1 CONFIG_SYSTEM_TRUSTED_KEYS="debian/canonical-certs.pem"
```

thành:

```
1 CONFIG_SYSTEM_TRUSTED_KEYS=""
```

2 Trim the kernel

Mở make nconfig và chọn các options configuration phù hợp sau đó xây lại kernel và khởi động lại máy ảo.



Hình 2: Màn hình Grub

3 System Call

3.1 Thêm system call mới

Trong thư mục linux-5.0.5 tạo thư mục `get_proc_info`. Trong `get_proc_info` tạo `sys_get_proc_info.c` và Makefile:

```
1 $ mkdir get_proc_info
2 $ cd get_proc_info
3 $ touch sys_get_proc_info.c
```

Thêm code cần thiết, sau đó:

```
1 $ touch Makefile
2 $ echo "obj-y := sys_get_proc_info.o"
```

Thêm `get_proc_info/` vào Makefile của kernel bằng cách sửa dòng sau:

```
1 core -y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ get\_proc\_info/
```

Thêm system call mới vào bảng system call:

```
1 $ cd arch/x86/entry/syscalls/
2 $ echo "548 64 get_proc_info sys_get_proc_info" >> syscall_64.tbl
```

QUESTION: What is the meaning of fields of the line that just add to the system call table (548, 64, get proc info, i.e.)

Trả lời:

- 548: Mỗi system call được xác định bởi một số và để gọi một system call ta gọi theo số của nó. Trong bảng system call đã có số 547 nên system call ta thêm vào là số 548
- 64: Application Binary Interface, là interface giữa hai chương trình modules, ở mức mã máy phổ biến là 64, x32, i386
- `get_proc_info`: tên của system call

- `sys_get_proc_info`: tên hàm gọi để xử lý system call. Theo quy tắc đặt tên hàm là tiền tố sys cộng với tên của system call

Thêm system call mới vào system call header file bằng cách mở file `syscalls.h` và thêm vào trước lệnh `#endif` :

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

QUESTION: What is the meaning of each line above?

Trả lời:

`struct proc_info` và `struct procinfo`: thêm hai cấu trúc dữ liệu

`asmlinkage long sys_get_proc_info(pid_t pid, struct procinfo * info)` là hàm xử lý system call kiểu trả về long (`asmlinkage` là một thẻ định nghĩa cho gcc nói với trình biên dịch rằng hàm không mong đợi tìm thấy bất kỳ đối số nào trong thanh ghi mà chỉ trên CPU stack)

3.2 Hiện thực system call

Nội dung file `sys_get_proc_info.c`:

```
1 #include <linux/kernel.h>  
2 #include <asm/unistd.h>  
3 #include <linux/linkage.h>  
4 #include <linux/sched.h>  
5 #include <asm/uaccess.h>  
6 #include <asm/current.h>  
7 #include <asm/errno.h>  
8 #include <linux/syscalls.h>  
9 #include <linux/string.h>  
10  
11 struct proc_info {  
12     pid_t pid;  
13     char name[16];  
14 };  
15  
16 struct procinfo {  
17     long studentID;  
18     struct proc_info proc;  
19     struct proc_info parent_proc;  
20     struct proc_info oldest_child_proc;  
21 };  
22  
23 SYSCALL_DEFINE2 (get_proc_info, pid_t, pid, struct procinfo *,info) {  
24     struct task_struct *process;  
25     if (pid == -1) {  
26         process = current;  
27     }  
28     else if (pid >= 0) {  
29         process = find_task_by_vpid(pid);  
30         if (process == NULL)  
31             return EINVAL;  
32     }  
33     else return EINVAL;  
34     struct task_struct *parent_process;  
35     struct task_struct *oldest_child_process;  
36     struct list_head *children_list;  
37     struct procinfo fieldproc;  
38     memset(&fieldproc, 0, sizeof(struct procinfo));  
39     fieldproc.studentID = 2013375;  
40     (fieldproc.proc).pid = process->pid;  
41     get_task_comm((fieldproc.proc).name, process);  
42     parent_process = process->parent;  
43     if (parent_process) {  
44         (fieldproc.parent_proc).pid = parent_process->pid;  
45         get_task_comm((fieldproc.parent_proc).name, parent_process);  
46     }  
47     children_list = &process->children;  
48     if (list_empty(children_list)) {
```

```

49     oldest_child_process = NULL;
50 }
51 else {
52     oldest_child_process = list_first_entry(children_list, struct task_struct, sibling);
53 }
54 if (oldest_child_process) {
55     (fieldproc.oldest_child_proc).pid = oldest_child_process->pid;
56     get_task_comm((fieldproc.oldest_child_proc).name, oldest_child_process);
57 }
58 else (fieldproc.oldest_child_proc).pid = 0;
59 copy_to_user(info, &fieldproc, sizeof(struct procinfos));
60 return 0;
61 }

```

3.3 Quá trình biên dịch và cài đặt

Biên dịch và cài đặt lại kernel:

```

1 $ make
2 $ make modules
3 $ sudo make modules_install
4 $ sudo make install

```

Sau khi cài đặt kernel mới, chạy một chương trình C để kiểm tra system call đã được tích hợp chưa

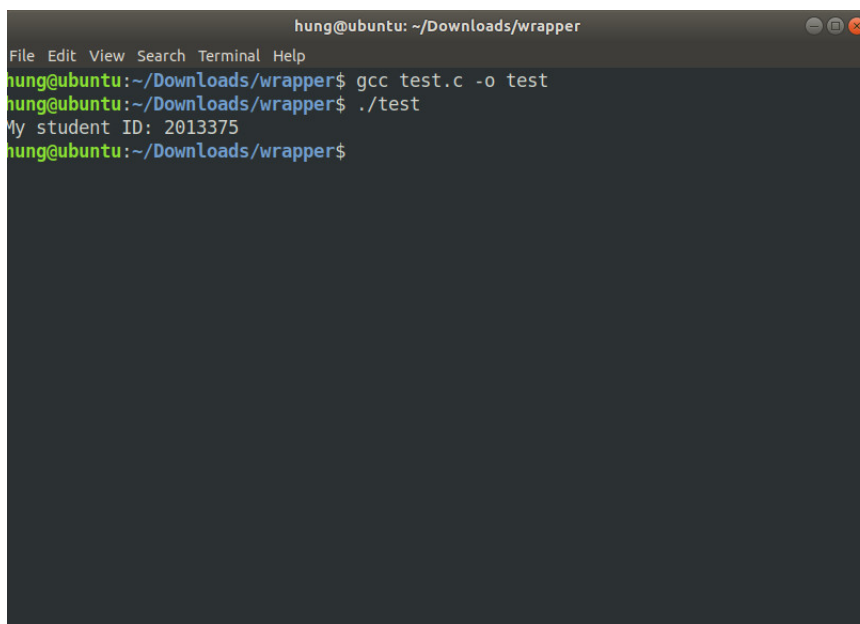
```

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 }

```

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

Trả lời: Vì chương trình gọi system call với PID=-1, nếu system call đúng thì sẽ in ra MSSV còn không thì system call không chính xác



```

hung@ubuntu: ~/Downloads/wrapper
File Edit View Search Terminal Help
hung@ubuntu:~/Downloads/wrapper$ gcc test.c -o test
hung@ubuntu:~/Downloads/wrapper$ ./test
My student ID: 2013375
hung@ubuntu:~/Downloads/wrapper$

```

Hình 3: Test system call

3.4 Tạo API cho system call

Mặc dù system call `get_proc_info` hoạt động tốt nhưng vẫn phải gọi qua number, gây bất tiện cho lập trình viên khác vì vậy ta cần hiện thực một chương trình C wrapper để dễ sử dụng. Tạo header file `get_proc_info.h`:

```
1 #ifndef _GET_PROC_INFO_H_
2 #define _GET_PROC_INFO_H_
3
4 #include <unistd.h>
5 #include <unistd.h>
6
7 struct proc_info {
8     pid_t pid;
9     char name[16];
10 };
11
12 struct procinfos {
13     long studentID;
14     struct proc_info proc;
15     struct proc_info parent_proc;
16     struct proc_info oldest_child_proc;
17 };
18
19 long get_proc_info(pid_t pid, struct procinfos * info);
20 #endif // _GET_PROC_INFO_H_
```

QUESTION: Why we have to redefine `procinfos` and `proc_info` struct while we have already defined it inside the kernel?

Trả lời: Vì tạo API để sử dụng thuận tiện hơn thay vì phải gọi syscall thông qua đối số number, ta cần một hàm trung gian để nhận giá trị từ lời gọi hàm `syscall(548,pid,info)` trả về. Info là yếu tố cần thiết cho việc tạo API(thuộc kiểu con trỏ của struct `procinfo`). Do đó cần định nghĩa lại struct `procinfo`, và struct `proc_info` để truyền tham số cho lời gọi syscall dù đã được định nghĩa sẵn trong kernel.

Tạo file `get_proc_info.c` chứa source code của wrapper:

```
1 #include "get_proc_info.h"
2 #include <linux/kernel.h>
3 #include <sys/syscall.h>
4 #include <unistd.h>
5
6 long get_proc_info(pid_t pid, struct procinfos * info) {
7     long sysvalue;
8     sysvalue = syscall(548, pid, info);
9     return sysvalue;
10 }
```

Cài đặt wrapper vào máy ảo:

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

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

Trả lời: Vì `/usr/` thuộc sở hữu của phân quyền root, ta cần phải chạy dưới người dùng root để ghi vào tệp đó bằng cách thêm sudo vào trước câu lệnh

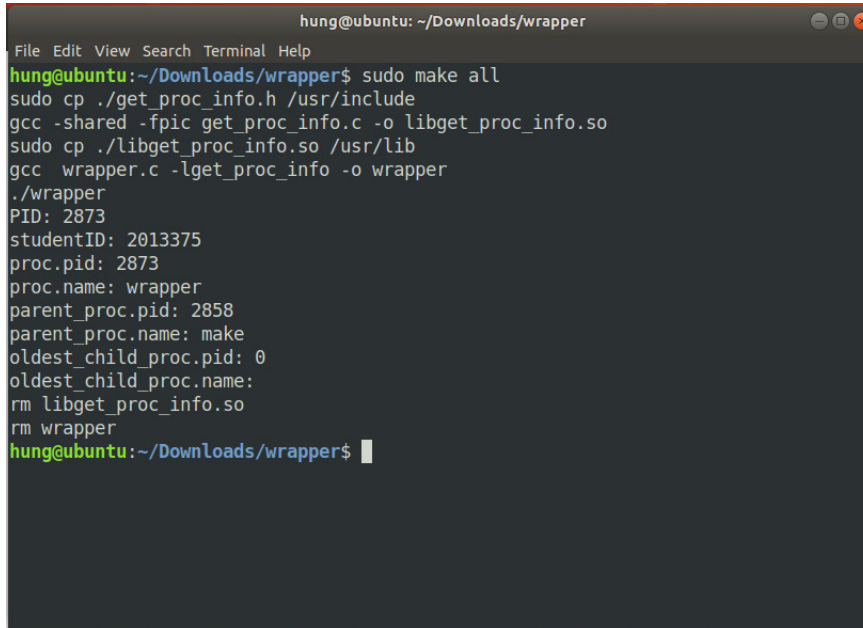
Sau đó biên dịch source code dưới dạng shared object để người dùng có thể tích hợp vào các chương trình:

```
1 $ gcc -share -fpic get_proc_info.c -o libget_proc_info.so
```

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

Trả lời: Bởi vì `-share` và `-fpic` giúp liên kết đến chức năng trong thư viện động được sử dụng khi thư viện được tải hoặc tại thời điểm chạy. Do đó, cả tệp thực thi và thư viện động đều được tải vào bộ nhớ khi chương trình được chạy. Trong đó `.so` là thư viện liên kết động, nó sẽ được tham chiếu, và load lên trong thời gian chạy.

Chạy sudo make all trong folder wrapper để kiểm thử:



```
hung@ubuntu: ~/Downloads/wrapper
File Edit View Search Terminal Help
hung@ubuntu:~/Downloads/wrapper$ sudo make all
sudo cp ./get_proc_info.h /usr/include
gcc -shared -fpic get_proc_info.c -o libget_proc_info.so
sudo cp ./libget_proc_info.so /usr/lib
gcc wrapper.c -lget_proc_info -o wrapper
./wrapper
PID: 2873
studentID: 2013375
proc.pid: 2873
proc.name: wrapper
parent_proc.pid: 2858
parent_proc.name: make
oldest_child_proc.pid: 0
oldest_child_proc.name:
rm libget_proc_info.so
rm wrapper
hung@ubuntu:~/Downloads/wrapper$
```

Hình 4: API chạy thành công