

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH**



**BÁO CÁO BÀI TẬP LỚN SỐ 1
MÔN HỌC: HỆ ĐIỀU HÀNH
Đề tài: System Calls**

GVHD: Hoàng Lê Hải Thanh

THÀNH VIÊN NHÓM:

1. Lê Nhật Anh – 1927001
2. Đoàn Tấn Lộc – 1812962
3. Nguyễn Văn Khoa – 1812660
4. Hứa Phước Thuận – 1814226

HCMC 23/5/2020

Contents

1. CHUẨN BỊ MÔI TRƯỜNG LÀM VIỆC.....	3
2.THÊM SYSTEM CALL MỚI:.....	6
3. HIỆN THỰC SYSTEM CALL:	6
4. BIÊN DỊCH VÀ CÀI ĐẶT	9
5. TẠO API CHO SYSTEM CALL.....	12

1. CHUẨN BỊ MÔI TRƯỜNG LÀM VIỆC

Cài đặt máy ảo: Do việc biên dịch và cài đặt một kernel nếu có bất kì một sai sót nào có thể dẫn đến lỗi hệ thống, nên để thuận tiện, môi trường làm việc sẽ được thực hiện trên máy ảo với hệ điều hành Ubuntu 18.04 (cho bản Linux Kernel 4.4). Ứng dụng máy ảo được sử dụng ở đây là Virtual Box. Image máy ảo được tải về từ <https://www.osboxes.org/ubuntu/>

Cài đặt core packages: Sau khi đã thiết lập xong máy ảo, một số packages cần được cài đặt để thiết lập môi trường phát triển Kernel

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

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

QUESTION: Why we need to install kernel-package?

ANSWER: We need to install kernel-package to keep multiple kernel-image versions on the device and not cause trouble. The kernel-package provides the tools and configurations which are necessary for kernel development, packaging, and installation. Moreover, it allows the management of installed kernels.

Tạo thư mục để chứa source và biên dịch Kernel:

```
$ mkdir ~/kernelbuild
```

Tải source code của Kernel:

```
$ cd ~/kernelbuild
$ 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?

ANSWER: Kernel supplied by OS is pre-compiled form, meaning that Kernel has been compiled and set up, they exist in computers as machine code.

Giải nén Source Code đã tải về:

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

Cấu hình Kernel:

Do việc tạo file config cho kernel tương đối phức tạp, nên chúng ta sẽ sử dụng lại file config hiện tại đang được sử dụng bởi hệ điều hành. File config này nằm tại thư mục /boot/. Để sử dụng lại file này, chỉ cần sao chép nó vào thư mục source code của kernel.

```
loc@loc-VirtualBox:~/kernelbuild$ pwd
/home/loc/kernelbuild
loc@loc-VirtualBox:~/kernelbuild/linux-5.0.5$ cp /boot/config-$(uname -r) ~/kernelbuild/linux-5.0.5/.config
```

Sau khi đã sao chép file config, cần phải sửa đổi lại version cho kernel. Để thiết lập thông qua giao diện terminal, package libncurses5-dev cần phải được cài đặt:

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

Để truy cập giao diện tùy chỉnh của kernel, chạy lệnh:

```
loc@loc-VirtualBox:~/kernelbuild/linux-5.0.5$ pwd
/home/loc/kernelbuild/linux-5.0.5
loc@loc-VirtualBox:~/kernelbuild/linux-5.0.5$ make nconfig
```

Lưu ý: Trong quá trình biên dịch, bạn có thể gặp phải lỗi do thiếu các gói openssl. Bạn cần phải cài đặt các gói này bằng cách chạy lệnh sau:

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

Tại giao diện tùy chỉnh, chọn *General setup* ^ (-) *Local version - append to kernel release*. Điền vào nội dung `.<MSSV>`



```
loc@loc-VirtualBox: ~/kernelbuild/linux-5.0.5
File Edit View Search Terminal Help

.config - Linux/x86 5.0.5 Kernel Configuration
Linux/x86 5.0.5 Kernel Configuration

*** Compiler: gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0 ***
General setup --->
[*] 64-bit kernel
Processor type and features --->
Power management and ACPI options --->
Bus options (PCI etc.) --->
Binary Emulations --->
Firmware Drivers --->
[*] Virtualization --->
General architecture-dependent options --->
[*] Enable loadable module support --->
[*] Enable the block layer --->
IO Schedulers --->
Executable file formats --->
Memory Management options --->
[*] Networking support --->
Device Drivers --->

F1Help F2SymInfo F3Help 2 F4ShowAll F5Back F6Save F7Load F8SymSearch F9Exit

loc@loc-VirtualBox: ~/kernelbuild/linux-5.0.5
File Edit View Search Terminal Help

.config - Linux/x86 5.0.5 Kernel Configuration
General setup

[ ] Compile also drivers which will not load
(.1812962) Local version - append to kernel release
[ ] Automatically append version information to the version string
() Build ID Salt
Kernel co Local version - append to kernel release
((none))
[*] Support f Please enter a string value.
[*] System V Press <Enter> to apply, <Esc> to cancel.
[*] POSIX Mes
[*] Enable pr .1812962
[*] uselib sy
*- Auditing
IRQ subsystem --->
Timers subsystem --->
Preemption Model (Voluntary Kernel Preemption (Desktop)) --->
CPU/Task time and stats accounting --->
[*] CPU isolation

F1Help F2SymInfo F3Help 2 F4ShowAll F5Back F6Save F7Load F8SymSearch F9Exit
```

2. THÊM SYSTEM CALL MỚI:

Để thêm system call mới, đầu tiên cần phải khai báo nó vào system call table trong kernel. Do kernel hỗ trợ nhiều kiến trúc khác nhau, ở đây ta chỉ quan tâm đến kiến trúc x86, nên sẽ thêm vào system call table cho kiến trúc x86 nằm tại thư mục `arch/x86/entry/syscalls`. Tại đây, system call mới sẽ được khai báo trong file `syscall_64.tbl`.

```
loc@loc-VirtualBox:~/kernelbuild/linux-5.0.5/arch/x86/entry/syscalls$ pwd
/home/loc/kernelbuild/linux-5.0.5/arch/x86/entry/syscalls
```

Mở file `syscall_64.tbl` thêm vào như hình:

```
439      64      sys_get_proc_info      __x64_sys_sys_get_proc_info|
```

QUESTION: What is the meaning of the components 439, 64, `get_proc_info`, i.e.?

ANSWER: **439** is the ID system call, **64** is the id of different process, **`sys_get_proc_info`** is the name of the syscall to be added at the user level, **`__x64_sys_sys_get_proc_info`** is the function name of the kernel that makes the system call.

3. HIỆN THỰC SYSTEM CALL:

Sau khi đã khai báo các system call mới vào system call table, ta tiến hành hiện thực system call.

Khai báo hàm entry point `sys_procsched` và các thành phần cần thiết trong file `include/linux/syscalls.h` bằng cách thêm vào cuối file này:

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

QUESTION: Meaning of the lines added above?

ANSWER: Struct `proc_info` and Struct `procinfos` in order to declare structure used in system call. And, `Asmlinkage long sys_get_proc_info(pid_t pid, struct procinfos *info)` to take information of parent and children process of pid for copying to `info`.

Chỉnh sửa “linux-5.0.5/sched.h”, và chọn macro `SYSCALL_DEFINE` để thực hiện chức năng gọi hệ thống của bạn ở cuối tệp này như sau:

```
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/syscalls.h>

struct proc_info {
    pid_t pid;
    char name[16];
};
struct procinfos {
    long studentID;
    struct proc_info proc;
    struct proc_info parent_proc;
    struct proc_info oldest_child_proc;
};

SYSCALL_DEFINE2(sys_get_proc_info, pid_t, pid, struct procinfos *, info)
{
    struct task_struct *p;
    struct task_struct *OldChild;
    struct list_head *ChildrenList;
    struct procinfos temp;
    bool check=false;
    temp.studentID = 1812962;
    printk(KERN_INFO "My studentID: %ld", temp.studentID);
    if(pid!=-1)
    {
        check=true;
        temp.proc.pid=current->pid;//current process
        strcpy(temp.proc.name,current->comm);
        printk(KERN_INFO "(Current) pid: %d\tname: %s\n", temp.proc.pid, temp.proc.name);
        temp.parent_proc.pid=current->parent->pid;//parent process
        strcpy(temp.parent_proc.name,current->parent->comm);
        printk(KERN_INFO "(Parent) pid: %d\tname: %s\n", temp.parent_proc.pid, temp.parent_proc.name);
        ChildrenList=&current->children;//child process
        if(list_empty(ChildrenList))
        {
            temp.oldest_child_proc.pid=-1;
            strcpy(temp.oldest_child_proc.name, " ");
            printk(KERN_INFO "Process have no childrent!\n");
        }
        else
        {
            OldChild = list_first_entry(ChildrenList, struct task_struct, sibling);
            temp.oldest_child_proc.pid=OldChild->pid;
            strcpy(temp.oldest_child_proc.name,OldChild->comm);
            printk(KERN_INFO "(OldChild) pid: %d\tname: %s\n", temp.oldest_child_proc.pid, temp.oldest_child_proc.name);
        }
    }
    else

```

```

{
    for_each_process(p)
    {
        if(p->pid==pid)
        {
            check=true;
            temp.proc.pid=p->pid; //current process
            strcpy(temp.proc.name,p->comm);
            printk(KERN_INFO "(Current) pid: %d\tname: %s\n", temp.proc.pid, temp.proc.name);
            temp.parent_proc.pid=p->parent->pid; //parent process
            strcpy(temp.parent_proc.name,p->parent->comm);
            printk(KERN_INFO "(Parent) pid: %d\tname: %s\n", temp.parent_proc.pid, temp.parent_proc.name);
            ChildrenList=&p->children; //child process
            if(list_empty(ChildrenList))
            {
                temp.oldest_child_proc.pid=-1;
                strcpy(temp.oldest_child_proc.name, " ");
                printk(KERN_INFO "Process have no childrent!\n");
            }
            else
            {
                OldChild = list_first_entry(ChildrenList, struct task_struct, sibling);
                temp.oldest_child_proc.pid=OldChild->pid;
                strcpy(temp.oldest_child_proc.name, OldChild->comm);
                printk(KERN_INFO "(OldChild) pid: %d\tname: %s\n", temp.oldest_child_proc.pid, temp.oldest_child_proc.name);
            }
        }
    }
}

copy_to_user(info, &temp, sizeof(struct procinfos));
if(check==true) return temp.studentID;
else
{
    printk(KERN_INFO "SYSCALL FAILED!\n");
    return EINVAL;
}
}

```


Mục đích của system call là lấy ra thông tin của 1 process đang chạy nhờ pid của process nên ta cần có 1 kiến trúc để lưu thông tin của process. Đó là **procinfos** và **proc_info**.

Tiếp theo ta tìm thông tin process qua hàm `sys_get_proc_info()`. Cụ thể hơn, mỗi process có 1 `task_struct` riêng, thông qua `task_struct` này, ta có thể truy cập được `sched_info` mỗi process đang lưu giữ (thông tin cần tìm). Để duyệt qua từng process, ta dùng lệnh `for_each_process (struct task_process *)`. Lệnh này có tác dụng duyệt từng process đang chạy, dữ liệu của process được lưu trữ tạm thời trong tham số truyền vào. Do đó ta có thể lấy thông tin của process thông qua cấu trúc **proc_info** vừa định nghĩa. Tiếp đến ta lấy pid của từng process và so sánh với pid được truyền vào. Nếu giống nhau thì sẽ in ra thông tin info.

Sau khi đã định nghĩa các thành phần cần thiết, Makefile cần được sửa đổi để có thể biên dịch được system call vừa được thêm vào. Mở file **kernelbuild/linux-5.0.5/Makefile** và thêm vào cuối file dòng sau:

Tìm dòng :

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

Và thêm vào như sau:

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

Tạo Makefile trong đường dẫn `~/kernelbuild/linux-5.0.5/get_proc_info` và thêm dòng dưới vào.

```
|obj-y := sys_get_proc_info.o
```

4. BIÊN DỊCH VÀ CÀI ĐẶT

Biên dịch kernel:

Đầu tiên , ta chạy lệnh **make** để biên dịch kernel và tạo **vmlinuz**, do việc này mất khá nhiều thời gian nên ta có thể chạy song song bằng cách sử dụng thẻ “-j np”, trong đó np là số process dùng để **make**.

```
$ make
or
$ make -j 4
```

Sau khi tạo vmlinux, ta tiến hành xây dựng các kernel modules. Tương tự như tạo vmlinux, ta cũng có thể dùng thẻ “-j np” để chạy song song

```
$ make modules
or
$ make -j 4 modules
```

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

ANSWER: **make** is used to compile the kernel and create **vmlinuz**, **vmlinuz** is an executable file of kernel, is loaded into memory by GRUB, or whatever other boot loaders you use; **make modules** is used to compile kernel modules.

Tiếp theo, ta cài đặt các kernel modules và kernel đã được biên dịch.

Đầu tiên ta cài đặt kernel modules.

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

Sau đó, cài đặt kernel.

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

Sau khi cài đặt thành công kernel và kernel modules, khởi động lại máy và nhấn shift trong lúc khởi động để hiện boot menu và chọn kernel mới cài đặt để nạp vào.

```
sudo reboot
```

Sau khi khởi động lại, kiểm tra phiên bản của kernel hiện tại bằng cách sử dụng lệnh:

```
uname -r
```

```
loc@loc-VirtualBox:~$ uname -r
5.0.5.1812962
```

Kết quả kiểm tra phiên bản kernel

Tạo một chương trình C nhỏ để kiểm tra xem system call đã được tích hợp vào kernel chưa.

```
#include <sys/syscall.h>
#include <stdio.h>
#include <unistd.h>
int main(){
    long sys_return_value;
    unsigned long info;
    sys_return_value= syscall(439,-1,&info);
    printf("My studentID: %ld\n",sys_return_value);
    return 0;
}
```

```
loc@loc-VirtualBox:~/Desktop/test$ ./main
My studentID: 1812962
```

```
loc@loc-VirtualBox:~/Desktop/test$ dmesg
```

```
[ 3006.224114] My studentID: 1812962
[ 3006.224141] (Current) pid: 2805      name: test
[ 3006.224142] (Parent) pid: 2680      name: bash
[ 3006.224142] Process have no childrent!
```

Kết quả thực thi của đoạn chương trình trên

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

ANSWER: Because this program can call the system call was created, the ordinal number in syscall 64.tbl is 439. If the call succeeds, it means system call add success.

5. TẠO API CHO SYSTEM CALL

Để thuận tiện cho việc sử dụng, nên thay vì dùng số thứ tự của system call để gọi nó, thì ta sẽ tạo một thư viện wrapper cho system call vừa được thêm vào.

Việc này có thể thực hiện bên ngoài kernel. Do đó, để tránh việc phải biên dịch lại kernel một lần nữa, ta tạo một thư mục mới khác với thư mục chứa mã nguồn kernel để chứa mã nguồn cho wrapper.

Sau đó, tạo một file header `get_proc_info.h` chứa prototype của wrapper và khai báo các struct cần thiết.

```
#ifndef _GET_PROC_INFO_H_
#define _GET_PROC_INFO_H_
#include <unistd.h>

#include <unistd.h>

struct procinfos {
    long studentID;
    struct proc_info proc;
    struct proc_info parent_proc;
    struct proc_info oldest_child_proc;
};

struct proc_info {
    pid_t pid;
    char name[16];
};

long sys_get_proc_info(pid_t pid, struct procinfos * info);
#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?

ANSWER: Define of struct in kernel can be only used by component in kernel. To be able to use it outside the kernel, we need to redefine it again.

Sau khi tạo xong file header, tiếp tục tạo file `get_proc_info.c` và tiến hành định nghĩa **function** `get_proc_info`

```
#include "get_proc_info.h"
#include <linux/kernel.h>
#include <sys/syscall.h>
long sys_get_proc_info(pid_t pid, struct procinfo * info) {
    long sys_return_value;
    sys_return_value= syscall(439,pid,info);
    return sys_return_value;
}
```

Sao chép file header **get_proc_info.h** vào thư mục **/usr/include**.

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

QUESTION: Why root privilege is required to copy the header file to **/usr/include**?

ANSWER: Because, in Linux OS, the access permission of directory **/usr/include** is root. So, to be access to this directory, user needs root privilege (adding sudo before cp command).

Kế đó, biên dịch **get_proc_info.c** để tạo thư viện liên kết động (shared object) cho phép người dùng có thể tích hợp system call vào ứng dụng của mình.

```
$ gcc -shared -fpic get_proc_info.c -o libget_proc_info.so
```

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

ANSWER: **-fpic** is used to compile a position-independent code. This is a necessary option while compiling shared object; **-shared** is used to compile a shared object.

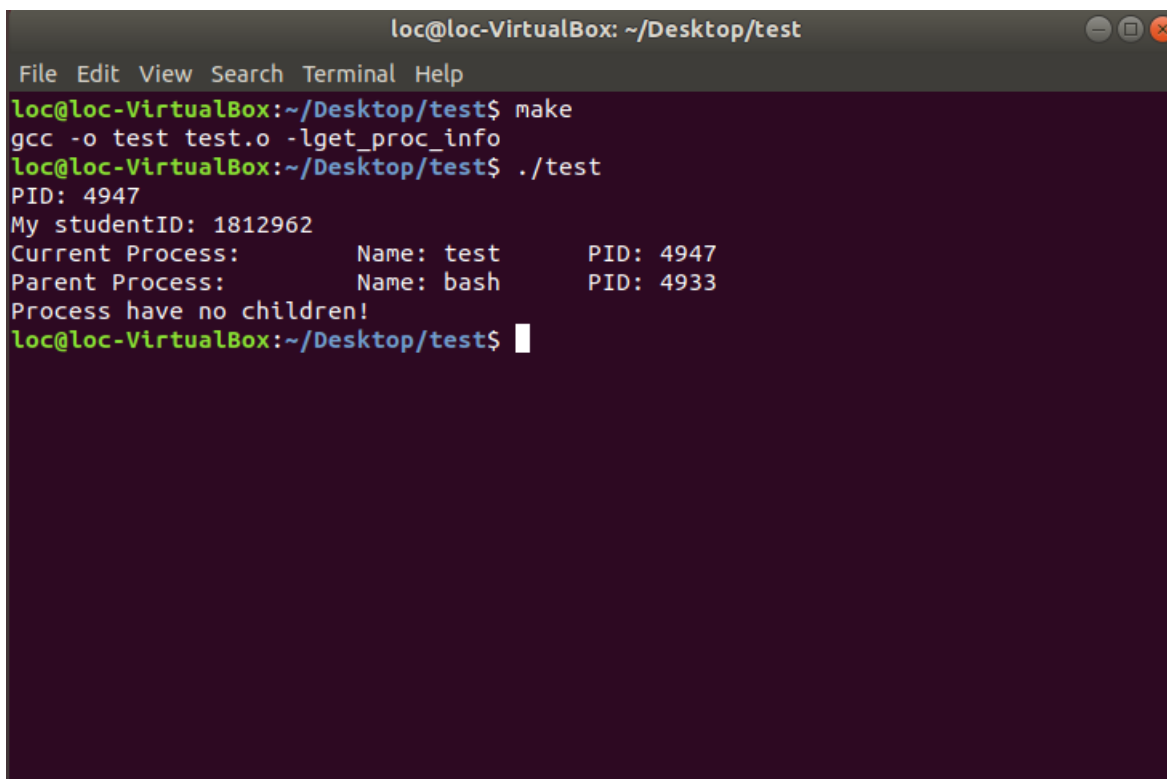
Sau đó, sao chép file **libget_proc_info.so** vào **/usr/lib**.

```
$ sudo cp libget_proc_info.so /usr/lib
```

Tạo file test.c để kiểm tra lại thư viện vừa tạo. Biên dịch và chạy để kiểm tra.

```
#include<sys/syscall.h>
#include<stdio.h>
#include<unistd.h>
#include<get_proc_info.h>
#include<sys/types.h>
#include<stdint.h>
int main(){
    pid_t mypid = getpid();
    printf("PID: %d\n",mypid);
    struct procinfo info;
    if(sys_get_proc_info(mypid,&info) ==1812962)
    {
        printf("My studentID: %ld\n",info.studentID);
        printf("Current Process:\tName: %s\tPID: %d\n",info.proc.name,info.proc.pid);
        printf("Parent Process: \tName: %s\tPID: %d\n",info.parent_proc.name,info.parent_proc.pid);
        if(info.oldest_child_proc.pid!=-1)
        {
            printf("Process have no children!\n");
        }
        else
        {
            printf("OldestChild Process:\tName: %s\tPID: %d\n",info.oldest_child_proc.name,info.oldest_child_proc.pid);
        }
    }
    else
    {
        printf("Cannot get information from the process %d\n",mypid);
    }
    return 0;
}
```

Kết quả thực thi file test.c



```
loc@loc-VirtualBox: ~/Desktop/test
File Edit View Search Terminal Help
loc@loc-VirtualBox:~/Desktop/test$ make
gcc -o test test.o -lget_proc_info
loc@loc-VirtualBox:~/Desktop/test$ ./test
PID: 4947
My studentID: 1812962
Current Process:      Name: test      PID: 4947
Parent Process:       Name: bash      PID: 4933
Process have no children!
loc@loc-VirtualBox:~/Desktop/test$
```

TÀI LIỆU THAM KHẢO

<https://williamthegrey.wordpress.com/2014/05/18/add-your-own-system-calls-to-the-linux-kernel/>

(last accessed 22/05/2020)

<https://www.kernel.org/doc/html/v4.10/process/adding-syscalls.html> (last accessed 22/05/2020)

<https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html> (last accessed 22/05/2020)