

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



**HỆ ĐIỀU HÀNH (CO2017 - CO2018)**

---

**Bài tập lớn 1**

# **System Call**

---

GVHD: Phạm Trung Kiên    Họ và tên  
SV:    Nguyễn Minh Tiến - MSSV 1713484

TP. HỒ CHÍ MINH, THÁNG 05/2019



## Mục lục

<b>1</b>	<b>Adding new system call</b>	<b>2</b>
1.1	Chuẩn bị . . . . .	2
1.1.1	Cài đặt máy ảo . . . . .	2
1.1.2	Cài đặt các package yêu cầu . . . . .	2
1.1.3	Tạo thư mục để biên dịch kernel . . . . .	2
1.1.4	Tải kernel . . . . .	2
1.1.5	Giải nén kernel . . . . .	2
1.2	Thiết lập cấu hình . . . . .	2
1.3	Thêm system call vào kernel . . . . .	3
<b>2</b>	<b>Biên dịch kernel</b>	<b>5</b>
<b>3</b>	<b>Hiện thực system call</b>	<b>6</b>
<b>4</b>	<b>Biên dịch và cài đặt</b>	<b>6</b>
4.1	Cài đặt cấu hình kernel . . . . .	6
4.2	Cài đặt kernel mới . . . . .	6
4.3	Thử nghiệm . . . . .	7
<b>5</b>	<b>Tạo API cho system call</b>	<b>7</b>
5.1	Đóng gói . . . . .	7
5.2	Xác nhận . . . . .	8
5.3	Kết quả . . . . .	9

## 1 Adding new system call

### 1.1 Chuẩn bị

#### 1.1.1 Cài đặt máy ảo

Tải và cài đặt Ubuntu phiên bản 18.04.2 trên phần mềm Virtual Box. Sau đó cập nhật và cài đặt các packages theo hướng dẫn.

#### 1.1.2 Cài đặt các package yêu cầu

```
sudo apt update
sudo apt install build-essential
sudo apt install kernel-package
sudo apt install fakeroot ncurses-dev xz-utils bc flex libelf-dev bison
sudo apt install openssl libssl-dev
sudo apt install htop #to view tree process
```

**Câu hỏi:** Tại sao chúng ta cần cài đặt kernel-package?

**Trả lời:** Kernel-package giúp thực hiện một chuỗi các công việc theo trình tự một cách tự động giúp tiết kiệm thời gian. Ngoài ra nó còn cho phép quản lý nhiều phiên bản của kernel-image đã cài đặt trên thiết bị. Cũng như, tự động lựa chọn các cài đặt phù hợp với từng kiến trúc. Các kernel-modules được liên kết với nhau, nên có thể biên dịch dễ dàng, đảm bảo tính tương thích. Các ưu nhược điểm của kernel-package được đề cập tại đây:

<http://manpages.ubuntu.com/manpages/bionic/man5/kernel-package.5.html>

#### 1.1.3 Tạo thư mục để biên dịch kernel

```
mkdir ~/kernelbuild
```

#### 1.1.4 Tải kernel

Ở đây là dùng phiên bản 5.0.5

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

**Câu hỏi:** Tại sao chúng ta phải sử dụng một nguồn kernel khác từ máy chủ, chẳng hạn như <http://www.kernel.org>, chúng ta có thể biên dịch kernel gốc (kernel cục bộ trên HĐH đang chạy) không?

Khi sử dụng nguồn kernel tải về thì có thể tùy chỉnh, xem xét thư mục, vài đặt bản vá cần thiết. Ta không thể biên dịch kernel trực tiếp từ chính nó, ta chỉ có thể biên dịch một phiên bản tương tự với phiên bản đang chạy.

#### 1.1.5 Giải nén kernel

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

### 1.2 Thiết lập cấu hình

Sao chép tập tin cấu hình có sẵn hiện có trong thư mục `/boot/` vào thư mục vừa giải nén:

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

Thay đổi phiên bản nội bộ bằng cách cấu hình kernel:



`make nconfig`

Để thay đổi phiên bản kernel chọn **General Setup** ----> **Local version - append to kernel release** sau đó nhập **.1713484** (MSSV), nhấn **F6** để lưu và **F9** để thoát.

Sử dụng `make localmodconfig` để cấu hình cho kernel chỉ biên dịch các modules hiện đang được load trong máy.

### 1.3 Thêm system call vào kernel

Tạo Makefile và `sys_get_proc_info.c` trong thư mục `get_proc_info`

```
mkdir ~/kernelbuild/linux-5.0.5/get_proc_info
```

```
cd ~/kernelbuild/linux-5.0.5/get_proc_info
```

```
touch Makefile
```

```
echo "obj-y:=sys_get_proc_info.o"
```

```
gedit sys_get_proc_info.c
```

Lưu nội dung sau vào file `sys_get_proc_info.c`

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

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

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

```
#define STUDENT_ID 1713484
```

```
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;
```

```
};
```

```
struct task_struct *proc;
```

```
struct task_struct *parent_proc;
```

```
struct task_struct *oldest_child_proc;
```

```
SYSCALL_DEFINE2(get_proc_info, pid_t, pid, struct procinfos *, info)
```

```
{
```

```
    printk("My_student_ID: %ld\n", (long)STUDENT_ID);
```

```
    printk("pid_to_find: %d\n", pid);
```

```
    if (pid == -1)
```

```
    {
```

```
        proc = current;
```

```
}
else if (pid == 0)
{
    proc = &init_task;
}
else
{
    proc = find_task_by_vpid(pid);
    if (proc == NULL)
    {
        printk("Not_found_proc_with_pid=%d!\n", pid);
        return EINVAL;
    }
}
info->studentID = STUDENT_ID;
printk("Proc_ID: %d\n", proc->pid);
printk("Proc_name: %s\n", proc->comm);
info->proc.pid = proc->pid;
sprintf(info->proc.name, proc->comm);
parent_proc = proc->parent;
if (parent_proc != NULL)
{
    printk("Parent_ID: %d\n", parent_proc->pid);
    printk("Parent_name: %s\n", parent_proc->comm);
    info->parent_proc.pid = parent_proc->pid;
    sprintf(info->parent_proc.name, parent_proc->comm);
}
else //Khong can thiet vi proc nao cung co parent
{
    printk("This_proc_does_not_have_parent!\n");
    info->parent_proc.pid = -1;
}
if (!list_empty(&proc->children))
{
    oldest_child_proc = list_first_entry(&proc->children, struct task_struct, child_list);
    printk("Child_ID: %d\n", oldest_child_proc->pid);
    printk("Child_name: %s\n", oldest_child_proc->comm);
    info->oldest_child_proc.pid = oldest_child_proc->pid;
    sprintf(info->oldest_child_proc.name, oldest_child_proc->comm);
}
else
{
    printk("This_proc_does_not_have_any_child!\n");
    info->oldest_child_proc.pid = -1;
}
return 0;
}
```

Chỉnh sửa Makefile trong thư mục linux-5.0.5



```
cd ~/kernelbuild/linux-5.0.5  
gedit Makefile
```

Tìm dòng:

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

Sửa lại thành:

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

Thêm lệnh system call mới vào bảng system call:

```
cd ~/kernelbuild/linux-5.0.5/arch/x86/entry/syscalls  
echo "548_common_get_proc_info__x64_sys_get_proc_info" >> syscall_64.tbl
```

**Câu hỏi:** Ý nghĩa của các trường của dòng chỉ cần thêm vào bảng system call

**Trả lời:** Các trường đó là [Number] [ABI] [Name] [Entry point] trong đó:

- Number: Số dùng để xác định system call được gọi khi dùng hàm syscall.
- ABI: (Application Binary Interface) là interface giữa 2 modules
- Name: Tên system call
- Entry point: Điểm truy cập, là tên của hàm được gọi để xử lý syscall, quy ước đặt tên cho hàm này là tên của syscall với tiền tố **sys\_**.

Định nghĩa system call mới

```
cd ~/kernelbuild/linux-5.0.5/include/linux  
gedit syscalls.h
```

Thêm các dòng sau trước **endif**:

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

**Câu hỏi:** Ý nghĩa của từng dòng trên là gì?

**Trả lời:** Hai dòng đầu dùng để khai báo cấu trúc được sử dụng. Dòng cuối cùng khai báo syscall, có bao nhiêu tham số cũng như kiểu của các tham số. Từ khoá **asmlinkage** thông báo rằng hàm chỉ nhận tham số được truyền vào từ stack.

## 2 Biên dịch kernel

Chạy các lệnh sau (máy ảo có 4 nhân)

```
cd ~/kernelbuild/linux-5.0.5/  
make -j4  
make modules -j4  
sudo make modules_install  
sudo make install  
sudo reboot
```

### 3 Hiện thực system call

Phần hiện thực của system call nằm trong file `sys_get_proc_info` trong thư mục `get_proc_info` đã trình bày ở trên. Mục đích của system call là truy xuất thông tin về process cha và process con già nhất cũng như thông tin của chính process đó thông qua PID. Cấu trúc `task_struct` được định nghĩa trong `/linux/sched.h` chứa các thông tin cần thiết. Ta sử dụng hàm `find_task_by_vpid` để tìm ra `task_struct` của process đó, sau đó thông qua con trỏ `parent` để tìm `task_struct` của process cha và hàm `list_first_entry` cùng cấu trúc `list_head` tên là `children` để tìm `task_struct` của process con già nhất.

### 4 Biên dịch và cài đặt

Sau khi hiện thực xong system call chúng ta cần biên dịch và xây dựng lại kernel để system call được thêm vào kernel.

#### 4.1 Cài đặt cấu hình kernel

Đầu tiên là biên dịch kernel

```
make -j4
```

Sau đó xây dựng kernel modules

```
make -j4 modules
```

**Câu hỏi:** Ý nghĩa của hai bước này, cụ thể là `make` và `make modules` là gì? Cái gì được tạo ra và để làm gì?

**Trả lời:** `make` tạo ra `vmlinuz`, một bản nén kernel-image được sử dụng bởi boot loader. Còn `make modules` biên dịch lại các modules mà được đánh dấu M trong kernel source. Các modules này giúp kernel giao tiếp được với các thiết bị, tương tự như các driver trên hệ điều hành Windows.

#### 4.2 Cài đặt kernel mới

Đầu tiên cài đặt các modules:

```
make -j4 modules_install
```

Sau đó cài đặt kernel.

```
make -j4 install
```

**Kiểm tra kết quả:** Sau khi cài đặt kernel theo các bước trên. Khởi động lại máy ảo bằng cách:  

```
sudo reboot
```

Sau khi đăng nhập lại vào máy tính, chạy dòng lệnh sau:

```
uname -r
```

Nếu kết quả trên mà hình là 5.0.5.1713484 thì kernel đã được biên dịch và cài đặt.

### 4.3 Thử nghiệm

Sau khi khởi động với kernel mới, tạo một chương trình C nhỏ để kiểm tra system call đã nằm trong kernel hay chưa?

```
#include <sys/syscall.h>
#include <stdio.h>
#include <unistd.h>
#define SIZE 200
int main(){
    long sys_return_value;
    unsigned long info[SIZE];
    sys_return_value = syscall(548, -1, &info);
    printf("My_student_ID: %lu\n", info[0]);
    return 0;
}
```

**Câu hỏi:** Tại sao chương trình này có thể cho biết liệu hệ thống của chúng ta có hoạt động hay không?

**Trả lời:** Hàm `syscall(548, -1, &info)` sẽ trả về giá trị của system call được gọi. Nếu không có, ở trường hợp chung, sẽ trả về `EINVAL` nếu không thành công. Và vì bản chất của struct là một vùng nhớ, trong struct `procinfo` ta khai báo đầu tiên là `long studentID` và vùng nhớ được cấp phát cũng kiểu long, nên có thể truy xuất `studentID` thông qua chỉ số 0 của vùng nhớ để kiểm tra thông tin được truyền có đúng đắn hay không.

## 5 Tạo API cho system call

### 5.1 Đóng gói

Tạo một tập tin header chứa mã giả của chương trình đóng gói và khai báo struct `procinfo`, `pro_info`. Đặt tên tập tin header là `get_proc_info.h` và hiện thực nó như sau:

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

struct proc_info {
    pid_t pid;
    char name[16];
};
struct procinfo {
    long studentID;
    struct proc_info proc;
};
```



```
    struct proc_info parent_proc;  
    struct proc_info oldest_child_proc;  
};  
long sys_get_proc_info(pid_t pid , struct procinfo * info);  
#endif // _GET_PROC_INFO_H
```

**Câu hỏi:** Tại sao chúng ta phải định nghĩa lại `procinfo` và `proc_info` trong khi chúng ta đã định nghĩa chúng trong kernel?

**Trả lời:** Sau khi biên dịch, file mã nguồn chứa định nghĩa của 2 struct không được chứa trong kernel đã biên dịch, nên ta cần định nghĩa lại để có thể biên dịch được chương trình cần sử dụng 2 struct này (ở đây là chương trình wrapper dùng để tạo thư viện gọi system call tiện lợi hơn. Sau đó, tạo một chương trình C tên là `get_proc_info.c` hiện thực đóng gói kernel. Chương trình C đó có nội dung như sau:

```
#include "get_proc_info.h"  
#include <linux/kernel.h>  
#include <sys/syscall.h>  
#include <unistd.h>  
  
long get_proc_info(pid_t pid , struct procinfo *info)  
{  
    return syscall(548, pid , info);  
}
```

## 5.2 Xác nhận

Copy file header vào thư mục `/usr/include`. Chạy những câu lệnh dưới đây để thực hiện điều đó:

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

**Câu hỏi:** Tại sao đặc quyền root được yêu cầu để sao chép file header đến `/usr/include`?

**Trả lời:** Vì `/usr/include` là thư mục hệ thống, người dùng thông thường chỉ có quyền truy cập, không có quyền chỉnh sửa các file trong thư mục này, vì vậy cần có quyền của root.

Dịch file `get_proc_info.c` trở thành một đối tượng được chia sẻ để người dùng tích hợp vào hệ thống của mình. Chạy lệnh:

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

Nếu biên dịch thành công, sao chép tập tin output sang `/usr/lib` (nhớ thêm `sudo` trước lệnh `cp`).

**Câu hỏi:** Tại sao chúng ta phải đặt `-shared` và `-fpic` vào lệnh `gcc`?

**Trả lời:** Vì `-shared` tạo ra các đối tượng dùng cho các thư viện chia sẻ còn `-fpic` dùng để sinh ra code không phụ thuộc vào vị trí, đoạn mã có thể được nạp lên từ bất kì địa chỉ nhớ ảo nào lúc thực thi và dễ tương thích với `-shared`.

Bước cuối cùng, kiểm tra tất cả những gì đã làm. Để kiểm tra thì ta viết một chương trình C như sau lưu lại với tên `test_wrapper.c`, biên dịch với `-lget_proc_info` và chạy nó.

```
#include <get_proc_info.h>  
#include <stdint.h>  
#include <stdio.h>  
#include <stdlib.h>
```

```
#include <sys/types.h>
#include <unistd.h>

static struct procinfo info;

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "usage: ./a.out <integer_value>\n");
        return -1;
    }
    long return_value;
    pid_t pid = (pid_t)atoi(argv[1]);
    return_value = get_proc_info(pid, &info);
    if (return_value != 0)
    {
        fprintf(stderr, "Process_with_pid=%d_not_found!\n", pid);
        return -1;
    }
    printf("My_StudentID=%ld\n", info.studentID);
    printf("Proc_ID: %d\n", info.proc.pid);
    printf("Proc_name: %s\n", info.proc.name);
    if (info.parent_proc.pid != -1)
    {
        printf("Parent_ID: %d\n", info.parent_proc.pid);
        printf("Parent_name: %s\n", info.parent_proc.name);
    }
    else //Khong can thiet vi proc nao cung co parent
    {
        printf("This_proc_does_not_have_parent!\n");
    }
    if (info.oldest_child_proc.pid != -1)
    {
        printf("Child_ID: %d\n", info.oldest_child_proc.pid);
        printf("Child_name: %s\n", info.oldest_child_proc.name);
    }
    else
    {
        printf("This_proc_does_not_have_any_child!\n");
    }
    return 0;
}
```

### 5.3 Kết quả

Sau khi hoàn tất cả các bước trên. Ta sẽ xem kết quả kiểm tra. Chạy những lệnh sau trên terminal:



```
gcc test_wrapper.c -lget_proc_info  
./a.out 1
```

Kết quả trên màn hình:

```
tien@tien-VirtualBox:~/Desktop/test$ ./a.out 1  
My StudentID = 1713484  
Proc ID: 1  
Proc name: systemd  
Parent ID: 0  
Parent name: swapper/0  
Child ID: 276  
Child name: systemd-journal  
tien@tien-VirtualBox:~/Desktop/test$
```

## Tài liệu

- [1] The Linux Kernel API (<https://www.kernel.org/doc/html/v5.0/core-api/kernel-api.html>).
- [2] Linux source code (<https://elixir.bootlin.com/linux/v5.0.5/source>).
- [3] The Linux Kernel Module Programming Guide (<http://www.tldp.org/LDP/lkmpg/2.6/html/index.html>).
- [4] Stackoverflow (<https://stackoverflow.com>).
- [5] UBUNTU Wiki (<http://wiki.ubuntu.com>).
- [6] GNU Operating System (<http://gnu.org>).
- [7] Và một số nguồn khác