

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



MÔN HỌC: HỆ ĐIỀU HÀNH

ASSIGNMENT 1

System Call

Danh sách thành viên:

1. Nguyễn Trung Tính
2. Nguyễn Nhật Tân
3. Đặng Văn Dũng
4. Cao Đăng Dũng

MSSV

1713521
1713074
1710853
1710849



Mục lục

1	Thêm syscall mới	2
1.1	Chuẩn bị Linux Kernel	2
1.2	File configuration	3
1.3	Dùng Kernel Module để thử nghiệm	4
2	Hiện thực System Call	7
3	Biên dịch và cài đặt	9
3.1	Biên dịch	9
3.2	Cài đặt	9
4	Kiểm thử	10
5	Tạo API cho System Call	11
5.1	Tạo API	11
5.2	Xác minh	12

Các bước thực hiện thêm một system call mới để lấy thông tin lịch trình (schedule) của một process bất kì.

1 Thêm syscall mới

1.1 Chuẩn bị Linux Kernel

Tải Ubuntu image bản 12.04 cho VirtualBox và khởi động máy ảo. Sau đó update và cài đặt các gói package cần thiết, chủ yếu là `build-essential` và `kernel-package`.

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

Câu hỏi: Tại sao phải cài đặt `kernel-package`?

Trả lời: Cài đặt `kernel-package` để giúp cho việc cá nhân hóa kernel dễ dàng hơn, nó cũng giúp cho việc compile kernel thuận tiện hơn bằng các script thực hiện tự động.

Tạo thư mục để build kernel và tải bản linux-4.4.56 về.

```
$ mkdir ~\kernelbuild
$ cd ~\kernelbuild
$ wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.4.56.tar.xz
```

Câu hỏi: Tại sao phải dùng kernel source từ `http://www.kernel.org`?, ta có thể biên dịch trực tiếp bằng kernel cục bộ có trong OS hay không?

Trả lời: Ta hoàn toàn có thể biên dịch trực tiếp bằng kernel có trong OS, nhưng với điều kiện là đủ can đảm và đảm bảo rằng hệ thống đều đã back up khi có lỗi xảy ra. Thay vào đó nếu ta muốn kiểm tra xem có lỗi không và sửa lỗi thì ta nên dùng kernel source tải từ `http://www.kernel.org` sẽ an toàn hơn nhiều.

Cài đặt `openssl` package? và giải nén kernel:

```
$ sudo apt-get install openssl libssl-dev
$ tar -xvJf linux-4.4.56.tar.xz
```



1.2 File configuration

Copy file /boot/config sang thư mục linux-4.4.56

```
$ cp /boot/config -x.x.x-generic ~\kernelbuild/linux-4.4.56/.config
```

Cài đặt package libncurses5-dev:

```
$ sudo apt-get install libncurses5-dev
```

Trong thư mục thư mục linux-4.4.56, mở kernel configuration:

```
$ make nconfig
```

Chọn General setup -> Local version - append to kernel release , sau đó nhập .1713521 vào và lưu lại.

1.3 Dùng Kernel Module để thử nghiệm

Dùng Kernel Module để kiểm tra các bước tìm task struct của process. Hiện thực file test.c để kiểm tra system call như sau:

Với Makefile như sau:

```
EXTRA_CFLAGS = -Wall
obj-m        = test.o
KDIR = /lib/modules/$(uname -r)/build
all:
    make -C $(KDIR) M='pwd'
clean:
    make -C $(KDIR) M='pwd' clean
```

```
#include <linux/module.h> // included for all kernel modules
#include <linux/kernel.h> // included for KERN_INFO
#include <linux/init.h> // included for __init and __exit macros
#include <linux/proc_fs.h>
#include <linux/sched.h>
#include <linux/pid.h>
static int pid;
struct task_struct *task;
static int __init procsched_init(void)
{
    printk(KERN_INFO "Starting kernel module!\n");
    struct pid *get_pid;
    get_pid = find_get_pid(pid);
    if (get_pid == NULL)
        return -1;
    else {
        task = pid_task(get_pid, PIDTYPE_PID);
        printk(KERN_INFO "pid = %d\n", task->pid);
        printk(KERN_INFO "pcount = %lu\n", task->sched_info.pcount);
        printk(KERN_INFO "run_delay = %llu\n", task->sched_info.run_delay);
        printk(KERN_INFO "last_arrival = %llu\n",
                task->sched_info.last_arrival);
        printk(KERN_INFO "last_queued = %llu\n",
                task->sched_info.last_queued);
        return 0;
    }
    return 1;
}
static void __exit procsched_cleanup(void)
{
    printk(KERN_INFO "Cleaning up module.\n");
}
module_init(procsched_init);
module_exit(procsched_cleanup);
module_param(pid, int, 0);
```



Sau đó biên dịch kernel module trong thư mục chứa test.c và Makefile, ta thu được kết quả:

```
[24680.492720] Starting kernel module!  
[24680.492725] pid: 3716  
[24680.492727] pcount: 514  
[24680.492728] run_delay: 147886838  
[24680.492729] last_arival: 24676626156164  
[24680.492730] last_queued: 0
```

Sau đó ta gỡ module ra khỏi kernel, ta có thêm dòng trong output.

```
[24753.860454] Cleaning up module.
```

2 Hiện thực System Call

Để thêm syscall mới, đầu tiên ta phải thêm thông tin của nó vào `syscall_32.tbl` và `syscall_64.tbl` trong đường dẫn `arch/x86/entry/syscalls`.

```
377 i386 procsched sys_procsched # trong file syscall_32.tbl.  
546 x32 procsched sys_procsched # trong file syscall_64.tbl.
```

Câu hỏi: Các phần `i386`, `procsched`, `sys_procsched` nghĩa là gì?

Trả lời: `i386` là ABI (application binary interface); `procsched` là tên của syscall; `sys_procsched` là điểm nhập (entry point) - tên của hàm gọi khi xử lý syscall, đặt theo cú pháp `sys_ + tên syscall`.

Mở file `include/linux/syscalls.h` và thêm 2 dòng sau vào:

```
struct proc_segs;  
asmlinkage long sys_procsched(int pid, struct proc_segs *info);
```

Câu hỏi: Ý nghĩa của 2 dòng trên là gì?

Trả lời: 2 dòng này dùng để khai báo prototype của `struct proc_segs` và hàm syscall `sys_procsched` - những hàm mà ta sẽ hiện thực ở phần sau trong file `sys_procsched.c`

Ta hiện thực file `sys_procsched.c` dựa trên file `test.c` ở trên và lưu vào thư mục

`/kernelbuild/linux-4.4.56/arch/x86/kernel/`.


```
#include <linux/linkage.h>
#include <linux/sched.h>
#include <linux/syscalls.h>
#include <linux/pid.h>
struct proc_segs {
    unsigned long mssv;
    unsigned long pcount;
    unsigned long long run_delay;
    unsigned long long last_arrival;
    unsigned long long last_queued;
};
asmlinkage long sys_procsched(int pid, struct proc_segs *info) {
    struct task_struct *task;
    struct pid *get_pid;
    get_pid = find_get_pid(pid);
    if (get_pid == NULL)
        return -1;
    else {
        task = pid_task(get_pid, PIDTYPE_PID);
        info->mssv = 1713521;
        info->pcount = task->sched_info.pcount;
        info->run_delay = task->sched_info.run_delay;
        info->last_arrival = task->sched_info.last_arrival;
        info->last_queued = task->sched_info.last_queued;
        return 0;
    }
}
```

Sau khi hiện thực xong, ta thêm dòng sau vào Makefile.

```
obj-y += sys_procsched.o
```

3 Biên dịch và cài đặt

3.1 Biên dịch

Biên dịch kernel, quá trình này có thể tốn vài giờ đồng hồ:

```
$ make # hoặc make -j 4
```

Sau đó build loadable kernel module:

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

Câu hỏi: Ý nghĩa của 2 giai đoạn trên là gì?

Trả lời: 2 giai đoạn trên để biên dịch lại toàn bộ kernel source và tạo kernel image chứa syscall mà ta vừa thêm vào.

3.2 Cài đặt

Đầu tiên ta cài đặt các modules:

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

Sau đó cài đặt kernel:

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

Cuối cùng ta khởi động lại máy ảo:

```
$ sudo reboot
```

Sau khi khởi động máy xong, ta có thể kiểm tra lại phiên bản bằng lệnh:

```
$ uname -r
```

Kết quả thu được chuỗi 4.4.56-MSSV thì chúng ta đã thành công.

```
4.4.56.1713521
```

4 Kiểm thử

Sau khi cài đặt xong, ta có thể kiểm thử syscall bằng chương trình c ngắn sau:

```
#include <sys/syscall.h>
#include <stdio.h>
#define SIZE    10
int main() {
    long sysvalue;
    unsigned long info[SIZE];
    sysvalue = syscall(546,1,info);
    printf("My MSSV: %lu\n",info[0]);
}
```

Biên dịch và chạy thử:

```
$ gcc test.c -o test
$ ./test
```

Kết quả thu được từ màn hình:

```
My MSSV: 1713521
```

Câu hỏi: Tại sao chương trình trên có thể cho ta biết được syscall có hoạt động hay không?

Trả lời: Syscall hoạt động đúng thì việc gán MSSV vào info mới thành công.

5 Tạo API cho System Call

5.1 Tạo API

Viết file header `procsched.h` với nội dung sau:

```
#ifndef _PROC_SCHED_H
#define _PROC_SCHED_H
#include <unistd.h>
struct proc_segs {
    unsigned long mssv;
    unsigned long long pcount;
    unsigned long long run_delay;
    unsigned long long last_arrival;
    unsigned long long last_queued;
};
long procsched(pid_t pid, struct proc_segs *info);
#endif // _PROC_SCHED_H
```

Câu hỏi: Tại sao phải định nghĩa lại `struct proc_segs` trong khi đã có sẵn trong kernel?

Trả lời: Ta phải định nghĩa lại và mới chỉ khai báo trong prototype của `struct proc_segs` trong `include/linux/syscalls.h`, ta chỉ có thể khai báo con trỏ đến struct này mà không sử dụng được các members của nó cũng như tạo một instance, bởi vì size hay members của `struct proc_segs` chưa được biết bởi trình biên dịch.

Hiện thực API trong file `procsched.c` (546 là index của syscall mà ta thêm vào).

```
#include <procsched.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
long procsched(pid_t pid, struct proc_segs *info) {
    return syscall(546, pid, info);
}
```

5.2 Xác minh

Sau khi tạo API xong, ta chỉ còn bước cuối là xác minh tính đúng đắn của thông tin nhận được khi gọi syscall. Đầu tiên ta copy file `procsched.h` vào thư mục `/usr/include`:

```
$ sudo cp /procsched.h /usr/include
```

Câu hỏi: Tại sao lại đặt quyền root lại cần thiết cho việc copy file header vào `/usr/include`?

Trả lời: Vì thư mục `/usr/include` thuộc quyền của root nên phải thêm `sudo` vào trước lệnh.

Sau đó biên dịch file `procsched.c` được tạo trong phần đóng gói:

```
$ sudo -shared -fpic procsched.c -o libprocsched.so
```

Câu hỏi: Tại sao phải thêm `-share`, `-fpic` vào gcc cmd?

Trả lời: Vì `-share` giúp chia sẻ file object được biên dịch ra để linked với các file object khác tạo nên file thực thi, còn `-fpic` dùng để sinh code cho tương thích với option `-shared` vì chúng cùng tập option.

Sau đó copy file output `libprocsched.so` vào thư mục `/usr/lib`:

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

Bước tiếp theo ta viết file final.c có nội dung như sau:

```
#include <procsched.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdint.h>
int main() {
    pid_t mypid = getpid();
    printf("PID: %d\n", mypid);
    struct proc_segs info;
    if (procsched(mypid, &info) == 0) {
        printf("Student ID: %lu \n", info.mssv);
        printf("pcount: %lu \n", info.pcount);
        printf("run_delay: %llu \n", info.run_delay);
        printf("last_arrival: %llu \n", info.last_arrival);
        printf("last_queued: %llu \n", info.last_queued);
    } else {
        printf("Cannot get information from the process %d\n", mypid);
    }
}
```

Biên dịch và chạy thử:

```
$ gcc final.c -lprocsched -o final
$ ./final
```

Kết quả thu được:

```
pid: 2385
Student ID: 1713521
pcount: 4
run_delay: 124105
last_arrival: 271938562300
last_queued: 0
```



Tài liệu

- [1] *Linux source code - Bootlin.*
- [2] *Compile and run kernel modules.*
- [3] *Adding a New System Call*