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)

Bài tập lớn 01

System Call

GVHD: Nguyễn Minh Trí
SV thực hiện: Thân Đức Tài – 1613027

Tp. Hồ Chí Minh, Tháng 3/2018

Mục lục

Báo cáo trình bày các bước thực hiện thêm system call mới để giúp các ứng dụng lấy được thông tin về lịch trình (schedule) của một process đã cho.

1 Thêm system call 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) cốt yếu là build-essential và kernel-package:

$ sudo apt-get update
$ 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 giúp cho việc cá nhân hóa kernel dễ dàng hơn, nó cũng giúp cho việc compile kernel thuận tiện hơn bằng việc script các bước thực hiện.

Tiếp theo tạo thư mục để build kernel, sau đó tải kernel về:

$ mkdir /home/kernelbuild
$ cd /home/kernelbuild
$ wget http://cdn.kernel.org/pub/linux/kernel/v4.x/linux -4.4.56.tar.xz

Câu hỏi: Tại sao phải dùng kernel source tải 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 backed up khi có bug xảy ra. Thay vào đó nếu ta muốn kiểm tra xem có bug không và sửa lỗi thì ta nên dùng kernel source tải từ *http://www.kernel.org* thì sẽ an toàn hơn nhiều.

Nếu gặp lỗi thiếu các openssl package thì cài đặt chúng:

$ sudo apt-get install openssl libssl -dev

Unpack kernel:

$ tar -xvJf linux -4.4.56.tar.xz

1.2 File configuration

Copy file config có sẵn trong /boot/ sang thư mục vừa unpack (thư mục top) với đuôi .config:

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

Cài đặt package libcurses5-dev:

$ sudo apt-get install libncurses5 -dev

Trong thư mục top mở kernel Configuration bằng lệnh:

$ make nconfig

Chọn General setup -> Local version - append to kernel release rồi nhập ".1613027" (MSSV) vào và lưu lại.

1.3 Dùng Kernel Module để thử nghiệm

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

```
#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_namespace.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");
   for_each_process(task){
      if((int)task->pid==pid){
         printk(KERN_INFO "%d\n",task->pid);
         printk(KERN_INFO "%lu\n",task->sched_info.pcount);
         printk(KERN_INFO "%llu\n",task->sched_info.run_delay);
         printk(KERN_INFO "%llu\n",task->sched_info.last_arrival);
         printk(KERN_INFO "%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_LICENSE("GPL");
module_init(procsched_init);
module_exit(procsched_cleanup);
module_param(pid, int , 0);
```

Với Makefile có nội dung:

```
obj-m += test.o

all:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Sau đó ta biên dịch kernel module trong thư mục chứa file test.c và Makefile:

$ make

Tiếp đến ta thêm module mới này vào kernel bằng lệnh:

$ insmod ./test.ko

Và kiểm tra output của module bằng lệnh:

$ dmesg | tail

Nội dung in ra sẽ có dạng:

```
[  803.332674] Starting kernel module!
[  803.332737] 3577
[  803.332741] 1
[  803.332744] 182029
[  803.332747] 748120461832
[  803.332750] 0
```

Vậy là ta đã kiểm tra được rằng các lệnh cần thiết để viết system call hoạt động tốt, bấy giờ ta gỡ bỏ module test ra khỏi kernel:

$ rmmod test

Kiểm tra lại xem đã gỡ bỏ module thành công hay chưa bằng lệnh:

$ dmesg | tail

So với lần gọi dmesg trước thì output có thêm dòng:

```
[ 2086.303518] Cleaning up module.
```

Và ta đã gờ module thành công.

1.4 Hiện thực system call

Đầu tiên ta phải thêm system call sắp hiện thực vào danh sách các system call. Trong thư mục arch/x86/entry/syscalls tìm file syscall_32.tbl và thêm dòng sau vào cuối file:

377 i386 procsched sys_procsched

Câu hỏi: Các phần i386, procsched và 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 của syscall.

Tương tự thêm vào file syscall_64.tbl:

546 x32 procsched sys_procsched

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 khai báo prototype của struct proc_segs và hàm syscall sys_procsched - những gì sẽ được hiện thực trong file sys_procsched.c bên dưới.

Ta hiện thực system call trong file sys_procsched.c dựa vào file test.c ở trên, cụ thể trình bày trong phần 2. Sau khi hiện thực xong, ta copy file trên vào thư mục /home/kernelbuild/linux-4.4.56/arch/x86/kernel/, rồi thêm dòng sau vào Makefile trong thư mục này:

obj-y += sys_procsched.o

1.5 Biên dịch và cài đặt

Sau khi hiện thực system call, ta phải biên dịch và cài đặt lại kernel để thêm system call vào hệ thống. Các bước biên dịch và cài đặt được mô tả cụ thể ở phần 3.

1.6 Chạy thử

Sau khi đã biên dịch và cài đặt xong, ta kiểm tra nhanh xem system call có thực sự hoạt động hay không. Viết file testing.c có nội dung như sau:

```
#include<sys/syscall.h>
#include<stdio.h>
#define SIZE 10
int main(){
   long sysvalue;
   unsigned long info[SIZE];
   sysvalue = syscall([number_32], 1, info);
   printf("My MSSV: %ul\n", info[0]);
}
```

Biên dịch và chạy thử:

$ gcc -c testing.c
$ gcc testing.o -o testing
$ ./testing

Kết quả in ra màn hình:

```
My MSSV: 1613027
```

Câu hỏi: Tại sao chương trình trên có thể cho ta biết được system call có hoạt động hay không ?
Trả lời: System call hoạt động mà không có lỗi thì việc gán MSSV vào biến info mới thành công.

1.7 Đóng gói

Sau khi chắc chắn rằng system call chạy được, ta đóng gói (tạo API) cho system call mới này. Các bước tạo API cho system call mới được nêu cju thể ở phần 4.

1.8 Xác minh

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

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

Câu hỏi: Tại sao đặc quyền của root (thêm sudo vào trước command) lại cần thiết cho việc copy file header vào /usr/include/ ?
Trả lời: Vì thư mục /usr/include thuộc quyền sở hữu của root nên muốn copy file vào thư mục này phải thêm sudo vào trước command để copy bằng đặc quyền của root (và phải nhập đúng mật khẩu của root).

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

$ gcc -shared -fpic procsched.c -o libprocsched.so

Câu hỏi: Tại sao ta phải thêm các option -shared và -fpic vào gcc command ?
Trả lời: Phải thêm các option này vào vì -shared 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.

Rồi copy file output libprocsched.so vào thư mục /usr/lib/:

$ sudo cp libprocsched.so /usr/lib

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

```c
#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);
   }
   sleep (100);
}
```

Biên dịch file trên và chạy thử:

$ gcc -lprocsched -c validation.c
$ gcc validation.o -lprocsched -o validation
$ ./validation

Kết quả in ra màn hình có dạng:

```
PID: 6081
Student ID: 1613027
pcount: 1
run_delay: 162504
last_arrival: 746239438239
last_queued: 0
```

Mở một terminal khác và dùng kernel modules để kiểm tra như đã trình bày ở phần 1.3, chỉ khác là để thêm kernel modules thì ta dùng lệnh:

$ insmod ./test.ko pid=6081

Sau đó gõ lệnh *dmesg | tail*, output sẽ là:

```
[  803.332674] Starting kernel module!
[  803.332737] 6081
[  803.332741] 1
[  803.332744] 162504
[  803.332747] 746239438239
[  803.332750] 0
```

Thông tin in ra giống với output ở phía trên nên system call của chúng ta đã hoạt động đúng.

2 Hiện thực system call

Hiện thực system call trong file sys_procsched.c cụ thể như sau:

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

   /* Find task_struct of provided pid */
   for_each_process (task)
   {
      if ((int)task->pid == pid)
      {
         info->mssv = 1613027;
         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;
      }
   }

   /* Not found */
   return 1;
}
```

3 Quá trình biên dịch và cài đặt

3.1 Biên dịch

Biên dịch kernel:

cd /home/kernelbuild/linux-4.4.56/
$ make -j 4

Sau đó build loadable kernel modules

$ make -j 4 modules
Câu hỏi: Ý nghĩa của 2 giai đoạn *make* và *make modules* trên là gì ?
Trả lời: 2 giai đoạn *make* và *make modules* trên biên dịch lại toàn bộ kernel sources và tạo ảnh kernel (kernel image) chứa system call mà ta vừa thêm vào.

3.2 Cài đặt

Đầu tiên cài đặt các module:

$ sudo make -j 4 modules_install

Rồi cài đặt kernel:

$ sudo make -j 4 install

Khởi động lại máy ảo bằng lệnh:

$ sudo reboot

Sau khi máy khởi động lại xong mở terminal lên và gõ:

$ uname -r

Trên màn hình in ra:

```
4.4.56.1613027
```

Chuỗi in ra chứa MSSV đã lưu trong file config chứng tỏ ta đã biên dịch và cài đặt kernel thành công.

4 Tạo API cho system call

Viết file header procsched.h với nội dung:

```
#ifndef _PROC_SCHED_H_
#define _PROC_SCHED_H_
#include<unistd.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;
};
long procsched(pid_t pid, struct proc_segs * info);
#endif // _PROC_SCHED_H_
```

Câu hỏi: Tại sao ta phải định nghĩa lại *struct proc_segs* trong khi đã có sẵn định nghĩa trong kernel ?
Trả lời: Ta phải định nghĩa lại vì ta mới chỉ khai báo 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. Vì vậy ta phải định nghĩa lại struct proc_segs trong file procsched.h.

Hiện thực API trong file procsched.c (546 là index của system call mà ta thêm vào trong file syscall_64.tbl, do tiến hành trên máy tính 64-bit):

```
#include"procsched.h"
#include<linux/kernel.h>
#include<sys/syscall.h>

long
procsched(pid_t pid , struct proc_segs * info)
{
   unsigned long long info_uld[5];

   long sysvalue = syscall(546, pid, &info_uld);

   info->mssv = info_uld[0];
   info->pcount = info_uld[1];
   info->run_delay = info_uld[2];
   info->last_arrival = info_uld[3];
   info->last_queued = info_uld[4];

   return sysvalue;
}
```

Tài liệu

[1] Implementing a system call in Linux Kernel 4.7.1
https://medium.com/@ssreehari/implementing-a-system-call-in-linux-kernel-4-7-1-6f98250a8c38

[2] Adding a New System Call
https://www.kernel.org/doc/html/v4.12/process/adding-syscalls.html

[3] Compile and run kernel modules
http://www.tldp.org/LDP/lkmpg/2.6/html/x181.html