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)

Báo cáo Bài tập lớn số 1

SYSTEM CALL

GV ra đề: Phạm Trung Kiên
Sinh viên thực hiện : Võ Minh Toàn 1915570
Quang Chấn Vĩ 1915961

Hồ Chí Minh, ngày 10 tháng 5 năm 2020

Mục lục

1 Adding new system call **2**

2 System call Implementation **3**

3 Compilation and Installation process **5**

3.1 Chuẩn bị . . . 5
3.2 Cấu hình . . . 5
3.3 Build kernel đã được cấu hình . . . 6
3.4 Cài đặt kernel mới . . . 6
3.5 Trim the kernel . . . 7

4 Making API for system call **7**

4.1 Wrapper . . . 7
4.2 Validation . . . 8

Danh sách hình vẽ

1 Test system call thành công . . . 3
2 Cài đặt kernel mới thành công . . . 7
3 Hoàn thành validation . . . 9

C Code

1 Test Code . . . 2
2 Hiện thực System call . . . 3
3 Header File . . . 7
4 Hiện thực Wrapper . . . 8
5 Validation Code . . . 8

1 Adding new system call

Tạo thư mục *get_proc_info* trong thư mục *kernelbuild*,sau đó tạo file *sys_get_proc_info.c* chứa phần hiện thực của syscall cần thêm. Trong thư mục kernelbuild, mở Makefile:

```
$ vim Makefile
```

Trong Makefile, tìm dòng:

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

và bổ sung thành:

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

Tiếp theo, bổ sung system call vào system call table:

```
$ pwd $ ~/kernelbuild/ $ cd arch/x86/entry/syscalls/ $ echo "548 64 get_proc_info
sys_get_proc_info" » syscall_64.tbl
```

Câu hỏi: Ý nghĩa của từng thông tin được thêm vào trong system call table (548 64 get_proc_info sys_get_proc_info)?

Trả lời: Các system call trong system call table có định dạng: <number> <abi> <name> <entry point>

- <number> 548 là số thứ tự của system call thêm vào.
- <abi> 64 ứng với hệ thống 64 bit.
- <name> *get_proc_info* là tên của system call.
- <entry point> *sys_get_proc_info* là tên chương trình cần gọi để xử lý system call.

Tiếp theo, chúng ta bổ sung một system call mới vào trong file system call header:

```
$ ~/kernelbuild/include/linux
```

Mở file *syscalls.h* và bổ sung dòng sau vào trước *#endif:*

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

Câu hỏi: Ý nghĩa của những dòng trên?

Trả lời: Các dòng trên được bổ sung vào file header (syscalls.h) nhằm mục đích khai báo các struct proc_info, procinfos và hàm sys_get_proc_info(pid_t pid, struct procinfos ∗ info).

Cuối cùng, biên dịch lại kernel và khởi động lại hệ thống để áp dụng kernel mới:

```
$ make -j 8
$ make modules -j 8
$ sudo make modules_install
$ sudo make install
$ sudo reboot
```

Sau khi khởi động vào kernel mới, chúng ta tạo một chương trình C nhỏ để kiểm tra system call đã được tích hợp chưa vào kernel hay chưa:

```
#include <sys/syscall.h>
#include <stdio.h>
#include <unistd.h>

#define SIZE 200

```

```c
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;
}

```

Listing 1: Test Code

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

Trả lời: Chương trình này in ra MSSV đã được thêm vào trong file sys_get_proc_info.c. Nếu không in ra đúng MSSV, system call đã thất bại.

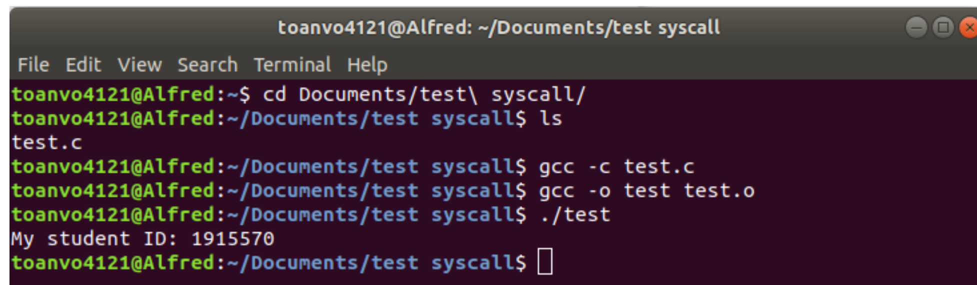

Figure 1: Test system call thành công

2 System call Implementation

Trong thư mục *kernelbuild*, tạo mới một thư mục có tên *get_proc_info*, và sau đó tạo một file *sys_get_proc_info.c* bên trong thư mục *get_proc_info*:

```
$ cd ~/kernelbuild
$ mkdir get_proc_info
$ cd get_proc_info
$ touch sys_get_proc_info.c
```

Sau đó viết code vào file *sys_get_proc_info.c*:

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/string.h>
#include <linux/syscalls.h>
#include <linux/sched/signal.h>
#include <asm/current.h>
#include <asm/uaccess.h>

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

```

```
16    struct procinfos
17    {
18      long studentID;
19      struct proc_info proc;
20      struct proc_info parent_proc;
21      struct proc_info oldest_child_proc;
22    };
23
24    SYSCALL_DEFINE2(get_proc_info, pid_t, pid, struct procinfos *, info)
25    {
26      struct task_struct *process = NULL, *child_process = NULL;
27      struct procinfos process_infos;
28
29      process_infos.studentID = 1913944;
30      if (pid == -1)
31      {
32        pid = current->pid;
33      }
34      for_each_process (process)
35      {
36        if (process->pid == pid)
37        {
38          process_infos.proc.pid = process->pid;
39          strcpy (process_infos.proc.name, process->comm);
40
41          // PARENT
42          if (process->real_parent != NULL)
43          {
44            process_infos.parent_proc.pid = process->real_parent->pid;
45            strcpy (process_infos.parent_proc.name, process->real_parent->comm);
46          }
47          else
48          {
49            process_infos.parent_proc.pid = 0;
50            strcpy (process_infos.parent_proc.name, "\0");
51          }
52
53          // CHILD
54          child_process = list_first_entry_or_null (&process->children,
55                                                    struct task_struct,
    sibling);
56
57          if (child_process != NULL)
58          {
59            process_infos.oldest_child_proc.pid = child_process->pid;
60            strcpy (process_infos.oldest_child_proc.name, child_process->comm);
61          }
62          else
63          {
64            process_infos.oldest_child_proc.pid = 0;
65            strcpy (process_infos.oldest_child_proc.name, "\0");
66          }
67
68          copy_to_user (info, &process_infos, sizeof(struct procinfos));
69          return 0;
70        }
71      }
72      return EINVAL;
73    }
74
```

Listing 2: Hiện thực System call

Sau đó, tạo một Makefile:

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

3 Compilation and Installation process

3.1 Chuẩn bị

Thiết lập máy ảo (Virtual machine): Trong bài tập lớn này, chúng ta sẽ cài đặt Ubuntu 18.04 trên máy ảo VMWare và cấp phát cho máy ảo đủ RAM cần thiết, dung lượng ổ cứng tối thiểu 40GB.

Cài đặt core packages: Sau khi cài xong máy ảo, chúng ta cài Ubuntu's toolchain (gcc, make):

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

Sau đó là cài kernel-package:

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

Câu hỏi: Tại sao chúng ta phải cài kernel-package?

Trả lời: Bởi vì kernel-package có rất nhiều phiên bản hạt nhân (kernel) để chúng ta lựa chọn sao cho phù hợp với cấu hình phần cứng của máy ảo (hay máy thật) của bạn.

Tạo thư mục biên dịch kernel: Tiếp theo, ta tạo mới một thư mục kernelbuild trong Home. Sau đó chúng ta tải kernel source về và giải nén, trong bài tập lớn này, ta sử dụng phiên bản kernel 5.0.5:

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

Câu hỏi: Tại sao chúng ta phải sử dụng những kernel source khác từ những server như http://www.kernel.org. Chúng ta có thể biên dịch kernel source gốc (kernel của OS hiện hành) trực tiếp được không?

Trả lời:Chúng ta có thể biên dịch kernel gốc của OS hiện hành bởi kernel mặc định được vận chuyển với Debian xử lí với mọi cấu hình. Tuy nhiên, việc sử dụng kernel source khác từ server giúp ta có thể sử dụng những tùy chọn không được hỗ trợ trong kernel sẵn có cũng như loại bỏ các tùy chọn không cần thiết để tăng tốc. Ngoài ra còn giúp giải quyết xung đột phần cứng với kernel được cung cấp trước và xử lí các nhu cầu phần cứng đặc biệt.

3.2 Cấu hình

Cấu hình của kernel nằm trong file .config, bằng cách cài đặt lại các tùy chỉnh trong file cấu hình sẽ giúp kernel và máy tính hoạt động một cách hiệu quả.

Chúng ta có thể copy file cấu hình của kernel hiện tại của OS sang thư mục linux-5.0.5:

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

Sau đó, chúng ta đổi tên lại phiên bản kernel. Để tùy chỉnh file cấu hình thông qua terminal interface, ta phải cài đặt thêm các package cần thiết:

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

Và chạy lệnh make menuconfig hoặc make nconfig để mở Kernel Configuration:

```
$ make nconfig
```

Để thay đổi phiên bản kernel, chọn General setup option → (-ARCH) Local version - append to kernel release, sau đó nhập vào .1915570

Lưu file lại và thoát ra.

3.3 Build kernel đã được cấu hình

Đầu tiên ta phải biên dịch kernel và tạo máy ảo vmlinuz. Việc này sẽ mất một khoảng thời gian khá dài. Trong terminal, di chuyển đến thư mục linux-5.0.5 và command:

```
$ make
Hoặc
$ make -j 4
```

Sau đó là xây dựng loadable kernel modules:

```
$ make modules
Hoặc
$ make -j 4 modules
```

Câu hỏi: Ý nghĩa của command make và make modules là gì? Những gì được tạo ra và để làm gì?

Trả lời:

- make: biên dịch và liên kết kernel image. Kết quả tạo ra một file có tên vmlinuz.
- make modules: biên dịch các module. Kết quả tạo ra các file nhị phân.

3.4 Cài đặt kernel mới

Đầu tiên là cài đặt các modules:

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

Sau đó là cài đặt kernel mới:

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

Sẽ mất một ít thời gian để hoàn thành việc cài đặt kernel mới. Sau khi cài đặt xong, khởi động lại máy:

```
$ sudo reboot
```

Sau khi khởi động lại, kiểm tra việc cài đặt bằng command sau:

```
$ uname -r
```

Kết quả: kết quả có chứa MSSV, do đó việc biên dịch và cài đặt đã thành công.

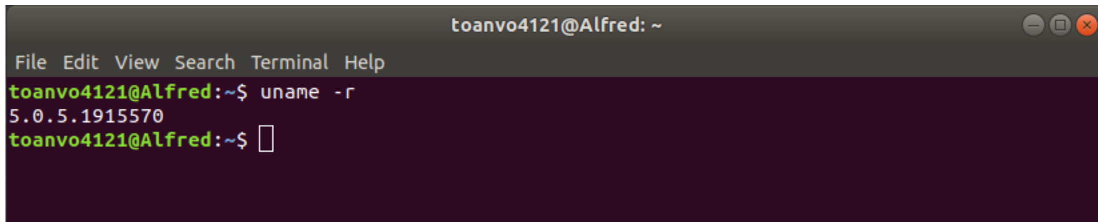

Figure 2: Cài đặt kernel mới thành công

3.5 Trim the kernel

Sau khi cài đặt kernel thành công, chúng ta có kernel mới với cấu hình mặc định. Nó sẽ bao gồm các gói hỗ trợ hầu như tất cả mọi thứ, dẫn đến sự dư thừa không cần thiết. Với cấu hình này, chúng ta sẽ mất rất nhiều thời gian để biên dịch. Trong khuôn khổ bài tập lớn này, chúng ta cần một cấu hình kernel khác cho máy. Cụ thể, chúng ta nên mở lại make nconfig để chọn cấu hình phù hợp cho máy của mình. Nó sẽ cung cấp cho chúng ta một loạt các menu, từ đó sẽ chọn các tùy chọn ta muốn đưa vào. Sau khi cấu hình, chúng ta sẽ biên dịch lại kernel.

4 Making API for system call

4.1 Wrapper

Mặc dù system call *get_proc_info* đã hoạt động đúng, chúng ta vẫn cần phải cải tiến nó để thuận tiện hơn cho các lập trình viên khác. Chúng ta cần triển khai C wrapper để dễ dàng sử dụng hơn. Để tránh biên dịch kernel lại một lần nữa, chúng ta sẽ tạo một thư mục khác để lưu trữ mã nguồn cho chương trình wrapper. Trước tiên chúng ta tạo một header file của chương trình wrapper và các cấu trúc *procinfos, proc_info*. Chúng ta đặt tên cho header file là *get_proc_info.h* và nó chứa nội dung như sau:

```
#ifndef _GET_PROC_INFO_H_
#define _GET_PROC_INFO_H_

#include <unistd.h>
#include <unistd.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;
};

long get_proc_info(pid_t pid, struct procinfos * info);
#endif // _GET_PROC_INFO_H_

```

Listing 3: Header File

Câu hỏi: Tại sao chúng ta phải định nghĩa lại các cấu trúc procinfos và proc_info trong khi đãđịnh nghĩa nó bên trong kernel?

Trả lời: Chúng ta cần định nghĩa lại nó để có thể sử dụng chúng ở bên ngoài.

Sau đó chúng ta tạo một file get_proc_info.c để giữ mã nguồn cho wrapper. Nội dung của file này như sau:

```c
#include "get_proc_info.h"
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>

long get_proc_info(pid_t pid, struct procinfos * info) {
  long sysvalue;
  sysvalue = syscall(548, pid, info);
  return sysvalue;
}

```

Listing 4: Hiện thực Wrapper

4.2 Validation

Tiếp theo, chúng ta copy file header *get_proc_info.h* vào */usr/include*:

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

Câu hỏi: Tại sao root lại đặc quyền? (Ví dụ phải thêm sudo trước cp để copy file header sang /usr/include)?

Trả lời: vì thư mục /usr thuộc quền sử hữu của root nên khi cần copy thì phải đc sự cho phép của root.

Sau đó ta biên dịch source code như một đối tượng chia sẻ để cho phép người dùng truy cập system call của ta thông qua ứng dụng của họ:

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

Sau khi biên dịch thành công, copy file *libget_proc_info.so* sang */usr/lib*:

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

Câu hỏi: Tại sao ta phải thêm -shared và -fpic trong gcc command?

Trả lời: -shared để biên dịch tạo ra thư viện liên kết động, -fpic để biên dịch ra mã không phụ thuộc vị trí đây là option cần thiết khi tạo thư viện liên kết.

Bước cuối cùng để kiểm tra toàn bộ công việc, chúng ta sử dụng chương trình sau và biên dịch với tùy chọn *get_proc_info* option:

```c
#include <get_proc_info.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 procinfos info;

  if(get_proc_info(mypid, &info) == 0){
    printf("studentID: %ld\n", info.studentID);
    printf("proc.pid: %d\n", info.proc.pid);
    printf("proc.name: %s\n", info.proc.name);
```

```
16          printf("parent_proc.pid: %d\n", info.parent_proc.pid);
17          printf("parent_proc.name: %s\n", info.parent_proc.name);
18          printf("oldest_child_proc.pid: %d\n", info.oldest_child_proc.pid);
19          printf("oldest_child_proc.name: %s\n", info.oldest_child_proc.name);
20      } else {
21          printf("Cannot get infomation from the process %d\n", mypid);
22      }
23  }
24
```

Listing 5: Validation Code

Kết quả thu được:

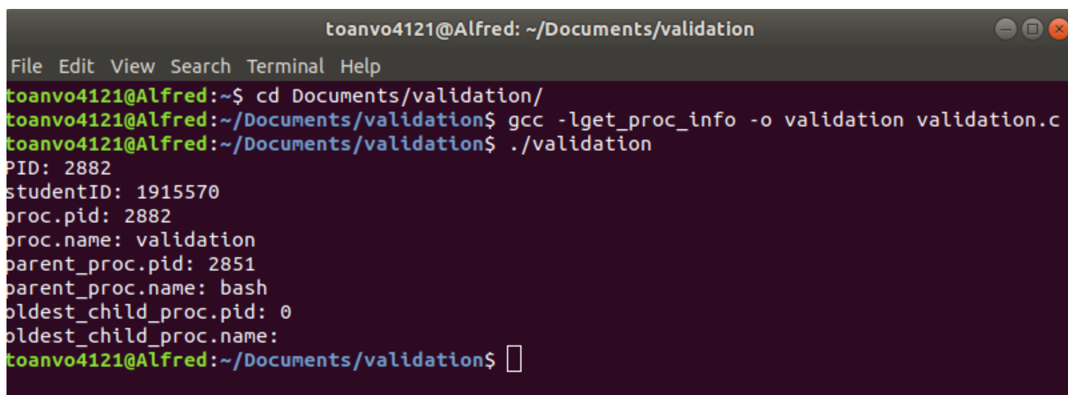

Figure 3: Hoàn thành validation