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

HỆ ĐIỀU HÀNH

BÀI TẬP LỚN

SYSTEM CALL

GVHD : Phạm Trung Kiên
Lớp : L06
Sinh viên : Lê Văn Nam - 1712237

Tp. Hồ Chí Minh
Ngày 5 tháng 5 năm 2019

Mục lục

1 Biên dịch và cài đặt linux kernel

1.1 Chuẩn bị

Tải và cài đặt **Ubuntu 18.04**. Cài gói **build-essential** và **kernel package**

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

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

Trả lời : Kernel-package là một công cụ(tool) được phát triển để biên dịch(compile) và cài đặt(install) một nhân hệ điều hành(kernel) được tùy biến. Có những lợi ích sau :

- Tính tiện lợi, kernel-package được viết để thực hiện một chuỗi các công việc theo trình tự (theo cách thủ công thì phải làm theo từng bước).
- Kernel-package cho phép giữ nhiều phiên bản của kernel-image trên thiết bị mà không gây xung đột, hay lỗi.
- Tự động di chuyển các thư mục tới vị trí thích hợp, cũng như tự động lựa chọn các cài đặt phù hợp với từng kiến trúc phần cứng.
- Cho phép hệ thống quản lý các phiên bản kernel đã cài đặt.

Tiếp theo tạo thư mục kernelbuild, sau đó tải kernel về thư mục vừa tạo:

```
$ mkdir kernelbuild
$ cd kernelbuild
$ wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.0.7s.tar.xz
```

Câu hỏi : Tại sao phải sử dụng mã nguồn kernel được tải từ http://www.kernel.org, ta có thể biên dịch trực tiếp trên nhân hệ điều hành gốc(original kernel) được hay không?

Trả lời : Ta sử dụng mã nguồn kernel để dễ dàng thêm, xóa, thay đổi mã nguồn(ví dụ như thêm một system call như yêu cầu của bài tập lớn). Ta không thể biên dịch một kernel trực tiếp từ chính nó.

1.2 Cấu hình

Sao chép file cấu hình Kernel Linux hiện có trong máy tính vào thư mục chứa mã nguồn Linux

```
$ cp /boot/config-$(uname -r) ~/kernelbuild/linux-5.0.7/.config
```

Trong thư mục linux-5.0.7 gõ lệnh `$ make menuconfig` hoặc `$ make nconfig` để mở Kernel Configuration:

```
$ make nconfig
```

Thay đổi tên kernel version: chọn General setup option, truy cập đến dòng "(-ARCH) Local version - append to kernel release". Sau đó nhập ".1712237"(MSSV) và nhấn F6 để lưu.

1.3 Trim the kernel

Mã nguồn linux được ta tải có thể dùng cho rất nhiều máy tính phổ biến hiện nay. Nhưng không cần thiết cho bài tập lớn này vì ta chỉ biên dịch và chạy trên một máy tính cá nhân. Nên để tiết

kiệm thời gian biên dịch và cài đặt ta cần thay đổi file cấu hình để chỉ biên dịch thành phần cần thiết. Từ bản Linux 2.6.32 ta có thể tự động hóa công việc trên thông qua lệnh:

```
$ make localmodconfig
```

1.4 Biên dịch kernel linux

```
$ make -j 8
$ make -j 8 modules
```

Câu hỏi: Ý nghĩa của 2 câu lệnh trên?

Trả lời:

- `make` tạo ra một bản kernel-image được sử dụng bởi trình boot loader, chứa system call vừa được thêm vào.
- `make modules` biên dịch lại các modules trong mã nguồn kernel. Khi biên dịch ta có thể bỏ qua lệnh này vì lệnh `make` đã thực hiện lệnh `make modules`.

1.5 Cài đặt kernel mới

Cài đặt modules và cài đặt kernel:

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

Khởi động lại máy để kiểm tra:

```
$ sudo reboot
```

Sau khi khởi động lại, chạy lệnh:

```
$ uname -r
```

Output: "linux-5.0.7.1712237". (Quá trình biên dịch và cài đặt thành công).

2 System call

2.1 The role of system call

System call `get_proc_info` xác định thông tin về các process.
Thông tin về những process được thể hiện theo cấu trúc sau:

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

- `proc`: process được tìm kiếm với pid hoặc process hiện tại.
- `parent_proc`: cha của process `proc`.
- `oldest_child_proc`: con lớn nhất của process `proc`.

Thông tin về một process được thể hiện qua cấu trúc sau:

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

- `pid`: định danh process.
- `name`: tên process.

2.2 Prototype

Prototype của system call như sau:

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

- **Input:** PID được user cung cấp là pid của process hoặc -1 nếu là process hiện tại.
- **Output:** Thông tin các process được lưu vào `struct procinfos* info` và trả về cho user.
- **Return:** Trả về 0 nếu lấy được thông tin process, `EINVAL` nếu không thể tìm thấy process.

2.3 Implementation

Vào thư mục linux-5.0.7, tạo thư mục `get_proc_info`

```
    $ cd ~/kernelbuild/linux-5.0.7
    $ mkdir get_proc_info
```

Vào thư mục `get_proc_info` vừa tạo, tạo Makefile và `get_proc_info.c`

```
    $ cd get_proc_info
    $ touch Makefile
    $ touch get_proc_info.c
```

Trong Makefile

```
obj-y := get_proc_info.o
```

Trong get_proc_info.c

```
SYSCALL_DEFINE2(get_proc_info, int, pid, struct procinfos*,
info)
{
    //--------------------- INITIALIZE --------------------
    struct task_struct * proc, *parent_proc, *oldest_child_proc;
    struct procinfos kinfo;
    kinfo.studentID = 1712237; printk("1712237");

    //-------------- CURRENT PROCESS-------------------------
    if(pid == -1)
        proc = current;
    else if(pid == 0)
        proc = find_task_by_vpid(2)->parent;
    else
        proc = find_task_by_vpid(pid);

    if(!proc)
        return EINVAL;

    kinfo.proc.pid = proc->pid;
    strcpy(kinfo.proc.name ,proc->comm);

    //---------------PARENT PROCESS---------------------------
    parent_proc = proc->parent;
    if(parent_proc == proc){
        kinfo.parent_proc.pid = -1;
        strcpy(kinfo.parent_proc.name, "NULL");
    }
    else{
        kinfo.parent_proc.pid = parent_proc->pid;
        strcpy(kinfo.parent_proc.name, parent_proc->comm);
    }

    //-------------- OLDEST_CHILDREN_PROCESS-----------------
    if(list_empty(&proc->children)) {
        kinfo.oldest_child_proc.pid = -1;
        strcpy(kinfo.oldest_child_proc.name, "NULL");
    }
    else{
        oldest_child_proc = list_first_entry(&proc->children,
            struct task_struct, sibling);
        kinfo.oldest_child_proc.pid = oldest_child_proc->pid;
        strcpy(kinfo.oldest_child_proc.name,
            oldest_child_proc->comm);
    }
    //-------------------------------------------------------

    copy_to_user(info, &kinfo, sizeof(struct procinfos));
    return 0;
}
```

- `SYSCALL_DEFINE2` Hàm thực hiện nhận 2 đối số để truyền vào trong system call.
- `struct task_struct` Bao gồm các trường thông tin của một process. Trong đó có 2 trường được sử dụng trong bài này: `pid`(PID của process) và `comm`(tên của process).
- `printk("1712237")` In MSSV vào kernel.
- `current` là con trỏ trỏ đến process hiện tại, sử dụng khi đối số PID = -1.
- `find_task_by_vpid:` Trả về con trỏ trỏ đến process với pid tương ứng.
- `list_empty` Kiểm tra danh sách chứa các process con có rỗng hay không.
- `list_first_entry` Trả về oldest children nếu process có con.
- `copy_to_user:` Copy data từ kernel space sang user space.

Thêm thư mục `get_proc_info` vào kernel Makefile
Tìm đến dòng sau:

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

Thêm `get_proc_info/` vào cuối:

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

Trong file `syscall_64.tbl` trong `/arch/x86/entry/syscalls` thêm vào dòng cuối

```
548 64 get_proc_info __x64_sys_get_proc_info
```

Câu hỏi: ý nghĩa của từng trường trong dòng vừa thêm vào file `syscall_64.tbl` ?

Trả lời:

- number : Tất cả các syscall được xác định bởi một số duy nhất. Để gọi một syscall, ta nói với kernel để gọi syscall theo số của nó chứ không phải bằng tên của nó.
- abi : Application Binary Interface - là interface giữa hai chương trình, một trong số đó thường là thư viện hoặc hệ điều hành, và cái còn lại là chương trình ở user space. Mức mã máy phổ biến là x64, x32, i386...
- name : Tên của syscall.
- entry point : là tên của hàm được gọi đầu tiên khi system call được nạp lên hệ thống. Quy ước đặt tên cho hàm này là tên của syscall có tiền tố với `sys_`. Ví dụ, điểm truy nhập của syscall "write" là `sys_write`.

Thêm system call mới vào trong header file:

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

Câu hỏi: Ý nghĩa của mỗi dòng trên?

Trả lời:

- `struct proc_info;` và `struct procinfos;` là các cấu trúc được sử dụng trong system call.
- `asmlinkage` thông báo rằng hàm chỉ nhận tham số được truyền vào từ thanh ghi.
- `long sys_get_proc_info(pid_t pid, struct procinfos* info);` Chỉ định kiểu tham số truyền vào và kiểu dữ liệu trả về của system call.

Biên dịch lại kernel và khởi động lại hệ thống để áp dụng kernel mới.

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

2.4 Tạo API cho system call

Tạo 2 file `get_proc_info.h` và `get_proc_info.c`
File `get_proc_info.h`

```
#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_
```

File `get_proc_info.c`

```
#include "get_proc_info.h"
long get_proc_info(pid_t pid, struct procinfos* info){
    assert(pid >= -1 && info != NULL);
    return syscall(548, pid, info);
}
```

Câu hỏi: Tại sao phải định nghĩa lại `struct proc_info` và `struct procinfos` khi mà đã định nghĩa nó ở trong kernel?

Trả lời: Các hệ điều hành có 2 phần. Một phần là kernel (gọi là kernel space trong *nix) và user-mode(gọi là user space trong *nix). Kernel và user-mode nói chuyện với nhau thông qua một thứ gọi là system call. Vì vậy các cấu trúc đã được định nghĩa trong kernel space thì user space không biết. Ta phải tái định nghĩa trong user space.

2.5 Validation

Biên dịch library file(.so)

```
$ gcc -shared -fpic get_proc_info.c -o libget_proc_info.so
$ sudo cp libget_proc_info.so /usr/lib
```

Copy header file(.h) và library file(.so) vừa tạo vào hệ thống.
Câu hỏi: Tại sao phải dùng đặc quyền root(super user) để thực hiện copy header file(.h) và library file(.so) vào thư mục `/usr/include` và `/usr/lib`?

Trả lời: Vì thư mục `/usr/` thuộc quyền của super user nên ta cần xin quyền super user(bằng lệnh sudo) để thêm các file vào thư mục.

Câu hỏi: Tại sao phải sử dụng 2 option `-shared` và `-fpic` khi biên dịch bằng gcc?

Trả lời:

- -shared dùng để chỉ cho trình biên dịch(gcc) biết rằng ta sẽ biên dịch ra một thư viện động(dynamic library)
- -fpic sinh ra code không phụ thuộc vào vị trí, để sử dụng cho thư viện động, địa chỉ của thư viện động sau khi được nạp lên hệ thống sẽ là hằng số được lưu trong global offset table (GOT). GOT là 1 thành phần của hệ điều hành.

3 Testing

Tạo file test.c với chương trình sau:

```
#include <get_proc_info.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdint.h>

int main(int argc, char const *argv[]) {
    pid_t mypid;
    struct procinfos info;

    if(argc < 2) pid = getpid();
    else pid = atoi(&*argv[1]);
    printf("PID: %d\n", mypid);

    if(get_proc_info(mypid, &info) == 0) {
        printf("StudentID : %ld\n", info.studentID);

        printf("Current_proc     : %i - %s\n",
            info.proc.pid, info.proc.name);

        printf("Parent_proc      : %i - %s\n",
            info.parent_proc.pid, info.parent_proc.name);

        printf("Oldest_child_proc      : %i - %s\n",
            info.oldest_child_proc.pid,
            info.oldest_child_proc.name);

    }else{
        printf("Cannot get information from the process %d\n",
            mypid);
    }
}
```

Trong thư mục chứa file test.c vừa tạo, chạy lệnh:

```
    $ gcc test.c -lget_proc_info -o test
    $ ./test -1
```

Ouput:

```
PID: 3043
StudentID : 1712237
Current_proc          : 3043 - test
Parent_proc           : 1905 - bash
Oldest_child_proc     : -1 - NULL
```

```
    $ ./test 0
```

Ouput:

```
PID: 0
StudentID : 1712237
Current_proc          : 0 - swapper/0
Parent_proc           : -1 - NULL
Oldest_Child_proc     : 1 - systemd
```

```
$ ./test 1203
```

Ouput:

```
PID: 1203
StudentID : 1712237
Current_proc          : 1203 - gnome-shell
Parent_proc           : 1073 - gnome-session-b
Oldest_child_proc     : 1250 - ibus-daemon
```

Tài liệu

[1] Silberschatz, Galvin, and Gagne, Operating System Concepts

[2] Tanenbaum, Modern Operating Systems

[3] Stallings, Operating Systems – Internals and Design Principles

[4] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, Operating Systems: Three Easy Pieces