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

HỆ ĐIỀU HÀNH

Bài tập lớn 1

SYSTEM CALL

GVHD: Phạm Trung Kiên
SV: Bùi Anh Nhật - 1612377

TP. HỒ CHÍ MINH, THÁNG 3/2018

Mục lục

1 Mô tả tiến trình thực hiện

1.1 Thêm một System call mới

Đầu tiên, cần một máy ảo Ubuntu sử dụng VirtualBox và cài đặt các gói dữ liệu cần thiết như kernel-package, libncurses5-dev,openssl, libssl-dev,...

Tiếp theo ta sẽ chỉnh sửa 2 tệp là `system_32.tbl` và `system_64.tbl` tại đường dẫn `arch/x86/entry/syscalls` để có thể chạy trên mọi máy có cấu trúc x86. Bước này đã thông báo triển khai system call vừa tạo cho `kernel`.

Tuy nhiên, ta cần phải chỉnh sửa tệp `include/linux/syscalls.h` để cho `kernel` biết được định nghĩa của system call như giá trị trả về, các giá trị nhận vào,... cùng với và cấu trúc struct đã sử dụng.

1.2 Hiện thực System call

Tạo tệp mã nguồn là `sys_procmem.c` ở trong đường dẫn `arch/x86/kernel`. Khi biên dịch `kernel`, ta phải thông báo cho `linker` cần phải tạo tệp object `sys_procmem.o` cũng như vị trí lưu các tệp mã nguồn thông qua việc chỉnh sửa tệp `Makefile`.

Mục đích của system call này là nhận thông tin của một process đang chạy thông qua `pid` của process. Vì vậy ta sử dụng cấu trúc `proc_segs` để lưu các thông tin nhận được.

Trước hết, ta cần tìm process tương ứng thông qua `for_each_process (struct task_struct *)`, được định nghĩa trong `linux/sched.h`. Cụ thể hơn, do mỗi process có một `task_struct` riêng, thông qua cấu trúc này, ta có thể truy cập được mm_struct mà có các thuộc tính lưu giữ thông tin về vùng nhớ của process. Trong `task_struct` cũng chứa `pid` của process, ta sẽ tìm thấy process tương ứng thông qua giá trị này. Tiếp theo, ta có thể lấy thông tin của process và lưu vào cấu trúc `proc_segs` đã được định nghĩa.

1.3 Quá trình biên dịch và cài đặt

Sau khi chỉnh sửa, thiết lập các thông tin cần thiết ta tiến hành thứ tự biên dịch, cài đặt kernel, sử dụng 4 process :
`make -j 4` biên dịch kernel
`make -j 4 modules` biên dịch tất cả modules
`sudo make -j 4 modules_install` cài đặt modules, cần thức hiện với quyền `root` (`sudo`)
`sudo make -j 4 install` cài đặt kernel, cần thức hiện với quyền `root` (`sudo`)

1.4 Tạo Wrapper cho System call

Do system call khi được gọi phải gọi thông qua số (được quy định trong tệp `system_32.tbl` và `system_64.tbl`), gây bất tiện, khó khăn cho người sử dụng và các lập trình viên khác. Nên ta cần phải viết wrapper để tiện sử dụng.

Đầu tiên, tạo một tệp header `procmem.h`, ở đây ta phải định nghĩa lại cấu trúc proc_segs, với thứ tự thuộc tính trùng với thứ tự đã định nghĩa trong kernel. Sau đó ta định nghĩa wrapper cho system call, mục đích là gọi system call, nhưng với cú pháp dễ hơn, các tham số truyền vào dễ quản lý bởi người dùng. Chi tiết hiện thực trong tệp `procmem.c`.

2 Trả lời câu hỏi

2.1 Phần 3.4 : Why we need to install kernel-package ?

Khi cài đặt gói `kernel-package`, có ích lợi :

- 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). Có nghĩa là tự động thực hiện các bước cần thiết để tạo một kernel tùy chỉnh.
- `kernel-package` cho phép việc giữ nhiều phiên bản của `kernel-image` trên thiết bị mà không gây rối.
- Có khả năng giữ nhiều phiên bản của cùng một `kernel` trên thiết bị.
- Tự động lựa chọn các cài đặt phù hợp với từng kiến trúc (ví dụ vmlinuz hay vmlinux, zImage hay bzImage) và tự động di chuyển các thư mục tới vị trí thích hợp cũng như .
- Hỗ trợ dpkg. Được quản lý bằng trình quản lý của hệ thống. (bằng `Package Management System`)
- Các `kernel-module` được liên kết với nhau, nên có thể biên dịch dễ dàng, đảm bảo tính tương thích.
- Theo dõi các tệp thiết đặt cho từng `kernel-image`
- Đảm bảo các tệp cài đặt cùng với `kernel-image` luôn đi cùng nhau.
- Cho phép việc tạo các gói cài đặt (`package`) với các tệp header, các tệp mã nguồn, cũng như các tệp `.deb`.
- Cho phép việc biên dịch `kernel` trên nhiều kiến trúc con.
- Cho phép việc biên dịch `kernel` trên các máy tính khác.

2.2 Phần 3.4 : Why we have to use another kernel source from the server, can we compile the original kernel (the local kernel on the running OS) directly ?

Khi sử dụng một mã nguồn `kernel` khác từ server, ta có thể dễ dàng tùy chỉnh, thêm một vài syscall cần thiết như trong asssignment.

Không thể biên dịch một `kernel` trực tiếp từ chính nó. Có thể biên dịch một phiên bản `kernel` tương tự với bản hiện tại, mã nguồn được lấy thông qua : `apt-get source linux-image-$(uname -r)`.

2.3 Phần 3.5 : What is the meaning of i386, procmem, and sys_procmem?

i386 : là ABI (Application binary interface) , mô tả các tương tác ở mức thấp (low-level) giữa ứng dụng và hệ điều hành với các ứng dụng khác. i386, hay intel 80386 vẫn được sử dụng các tập lệnh, mã hóa nhị phân trong 32 bit processor
procmem : dùng để khai báo khi sử dụng ở mức người dùng (`user-space`)
sys_procmem : là tên hàm của `kernel` mà hiện thực `system call`.

2.4 Phần 3.5 : What is the meaning of each line above ?

struct `proc_segs;` Khai báo cấu trúc được sử dụng trong system call.
`asmlinkage` **long** `sys_procmem (int pid, struct proc_segs __user *info)` Chỉ định kiểu tham số truyền vào, cũng như kiểu dữ liệu của system call. `asmlinkage` thông báo rằng hàm chỉ nhận tham số được truyền vào từ stack của CPU, không nhận tham số từ thanh ghi.

2.5 Phần 4.1 : What is the meaning of these two stages, `make` and `make modules`?

`make` biên dịch và liên kết `kernel-image`, tạo ra file `vmlinuz`
`make modules` biên dịch từng file riêng biệt đã được đánh dấu trong kernel config. Object sẽ được liên kết với kernel mới.

2.6 Phần 4.3 : Why this program could indicate whether our system call works or not?

Để biết `system call` có hoạt động hay không ? Ta phụ thuộc vào giá trị trả về của hàm

```
sysvalue = syscall([syscall_number_in_tbl], 1, info);
```

Hàm sẽ trả về giá trị của `system call` được gọi. Nếu không có, ở trường hợp chung, sẽ trả về giá trị return theo như trong hiện thực nếu gọi thành công, -1 nếu

không thành công.
Chương trình sẽ in ra MSSV đúng nếu thành công, còn không nó sẽ trả về một giá trị nào đó khác.

2.7 Phần 5 : Why we have to re define proc_segs struct while we have already defined it inside the kernel?

Chúng ta phải định nghĩa lại cấu trúc `proc_segs` vì khi truyền tham số mảng `info` vào `syscall()`, ta chỉ thu được một mảng lưu các thông tin của cấu trúc `proc_segs`, theo thứ tự như định nghĩa trong `kernel`. Để sử dụng `proc_segs` ở mức người dùng, ta phải tái cấu trúc lại `proc_segs`.

2.8 Phần 5.1 : Why root privilege (e.g. adding sudo before the cp command) is required to copy the header file to /usr/include?

Vì thư mục đang truy cập không nằm trong `user space`, nên khi sao chép thư mục vào trong ta phải thực hiện với quyền `root`.

2.9 Phần 5.1 : Why we must put -shared and -fPIC option into gcc command?

file

-fPIC nói với trình biên dịch tạo ra các mà không phụ thuộc vị trí (Position Independent Code), đoạn mã có thể được nạp lên bất kì địa chỉ nhớ ảo nào lúc thực thi, được sử dụng khi tạo `shared object` trên các kiến trúc thích hợp.
-shared tạo ra tệp đối tượng (object) dùng cho các thư viện chia sẻ.

Tài liệu

[1] kernel-package "**http://man.he.net/man5/kernel-package**",

[2] Linux Documentation "**http://linux.die.net**",

[3] GNU Operating System "**http://gnu.org/**",

[4] Make Linux "**http://makelinux.net/**",

[5] UBUNTU Wiki "**http://wiki.ubuntu.com/**",