INT2214 Bài tập #1: Kernel Module

Nguyễn Thành Đô - 19020250

Giảng viên: Bùi Duy Hiếu

Hạn nộp: Thứ hai, 14/03/2022 trước **23h:59'**

1 Giới thiệu vấn đề

Hệ điều hành là một hệ thống lớn và phức tạp, do đó cần được thiết kế cẩn trọng để nó có thể hoạt động tốt và có thể dễ dàng sửa đổi. Một hướng tiếp cận tự nhiên đó là chia hệ điều hành thành nhiều thành phần nhỏ, hay còn gọi là modules, hơn là chỉ xây dựng một hệ thống duy nhất. Thử thách của hướng tiếp cận này đó là cần phải xác định vai trò của từng module thật rõ ràng, cũng như là các chức năng và giao diện của nó.

Tuy rằng vậy, cấu trúc của hệ điều hành cũng có nhiều loại từ đơn giản đến phức tạp. Đầu tiên là kiểu cấu trúc **monolithic**, nó được coi là cấu trúc đơn giản nhất để tổ chức một hệ điều hành. Cụ thể, tất cả các tính năng của kernel được đặt vào một file nhị phân duy nhất chạy ở một không gian địa chỉ duy nhất. Ý tưởng nghe có vẻ đơn giản nhưng cấu trúc này lại rất khó để triển khai và mở rộng. Tuy nhiên cấu trúc monolithic vẫn có được lợi thế về mặt hiệu suất khi tốc độ để truyền thông trong kernel là rất nhanh và chi phí để thực hiện system call cũng thấp.

Một hướng tiếp cận khác là thiết kế hệ thống **phân lớp**, mà mỗi lớp sẽ có những chức năng cụ thể và giới hạn. Lợi thế của cách tiếp cận này đó là thay đổi xảy ra ở một lớp này sẽ chỉ ảnh hưởng tới lớp đó, cho phép người triển khai hệ thống tự do hơn trong việc tạo ra và thay đổi hoạt động bên trong của hệ thống. Tuy nhiên mỗi lớp sẽ "giấu đi" các cấu trúc dữ liệu, các toán tử và phần cứng khỏi lớp bên trên nó, dẫn đến hiệu suất của hệ thống bị giảm đi đáng kể khi mà nó bắt một chương trình người dùng phải phải đi qua nhiều lớp khác nhau mới có thể thực thi được một dịch vụ hệ thống.

Vào giữa những năm 1980, các nhà nghiên cứu tại Đại học Carnegie Mellon đã phát triển hệ điều hành *Mach* có thể module hóa kernel bằng cách sử dụng hướng tiếp cận **microkernel**. Ý tưởng của microkernel là rời bỏ tất cả những thành phần được cho là không thiết yếu ra khỏi kernel và cài đặt chúng như là chương trình người dùng, nằm ở các không gian địa chỉ riêng biệt. Kernel do đó nhỏ gọn hơn rất nhiều. Một chức năng chính của microkernel là cung cấp truyền thông giữa chương trình người dùng và các dịch vụ khác cùng nằm ở không gian người dùng thông qua phương pháp truyền thông điệp. Một lợi ích của microkernel là nó giúp mở rộng hệ điều hành dễ dàng hơn. Tất cả các dịch vụ mới được thêm vào không gian người dùng và do đó không yêu cầu sửa đổi kernel. Vì kích thước microkernel nhỏ gọn, nên việc sửa đổi kernel (nếu bắt buộc) sẽ ít hơn, đồng thời tính bảo mật, tính tin cậy cũng sẽ cao hơn do phần lớn dịch vụ được chạy ở không gian người dùng hơn là không gian kernel. Tuy nhiên, hiệu năng của hệ thống sử dụng microkernel lại rất tệ, do việc truyền thông điệp giữa các chương trình người dùng phải đi qua trung gian là microkernel. Cụ thể hơn, chi phí cho việc sao chép thông điệp và thay đổi ngữ cảnh giữa các dịch vụ nằm ở các không gian địa chỉ khác nhau là rất đáng kể.

Phương pháp tốt nhất để thiết kế hệ điều hành hiện tại có lẽ liên quan đến việc sử dụng **loadable kernel modules (LKMs)**. Ở đây, kernel có một tập hợp các thành phần cốt lõi và có thể liên kết với các dịch vụ bổ sung thông qua các module, tại thời điểm boot hệ thống hoặc trong thời gian chạy. Ý tưởng của thiết kế là để kernel cung cấp các dịch vụ cốt lõi, trong khi các dịch vụ khác được cài đặt động (implemented dynamically), khi kernel đang chạy. Liên kết động các dịch vụ tốt hơn là thêm các tính năng mới trực tiếp vào kernel, điều này sẽ yêu cầu biên dịch lại kernel mỗi khi có thay đổi. Tổng thể hệ điều hành sẽ giống như một hệ thống phân lớp trong đó mỗi phần kernel có các giao diện được xác định, được bảo vệ; nhưng nó linh hoạt hơn hệ thống phân lớp, bởi vì bất kỳ module nào cũng có thể trực tiếp gọi bất kỳ module nào khác.

Cách tiếp cận này cũng tương tự như cách tiếp cận microkernel ở chỗ các module chính chỉ có các chức năng cốt lõi và kiến thức về cách nạp và truyền thông với các module khác; nhưng nó hiệu quả hơn, bởi vì các module không cần phải truyền thông điệp để giao tiếp.

Kernel của Linux là monolithic, nó chạy ở chế độ kernel mode toàn bộ trong một không gian địa chỉ duy nhất, nhưng đồng thời Linux cũng có thiết kế sử dụng module bởi vì nó cho phép sửa đổi kernel trong thời gian chạy. Hai bài tập dưới đây sẽ minh họa cho việc tạo ra, nạp vào và rời bỏ LKMs trong Linux.

2 Bài toán

Nhiệm vụ của hai bài tập dưới đây là viết hai kernel modules và nạp chúng vào trong kernel. Sau khi đọc tài liệu hướng dẫn thì em nhận thấy rằng tất cả công việc cần làm chỉ là chỉnh sửa file *hello.c* đã được viết sẵn, tận dụng thêm biến *jiffies* lưu ở trong file *linux/jiffies.h* và biến *HZ* lưu ở trong file *asm/param.h*

Do đó trong báo cáo này, em cho rằng các giải thích chi tiết về code là không cần thiết. Lời giải cụ thể được cho ở bên dưới.

2.1 Module jiffies

Yêu cầu: *Thiết kế một kernel module cho HDH Linux tạo ra file /proc/jiffies khi module được nạp, hiển thị giá trị jiffies khi file này được đọc bằng lệnh:*

```
$ cat /proc/jiffies
```

Xóa file /proc/jiffies khi bỏ module khỏi kernel.

Answer: Kernel lưu giữ một biến toàn cục là jiffies, nó đếm số lần timer đã interrupt kể từ khi hệ thống được boot. Biến jiffies được khai báo trong file <linux/jiffies.h>
Code:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include <linux/jiffies.h>
#define BUFFER_SIZE 128
#define PROC_NAME "jiffies"
#define MESSAGE "Jiffies\n"

/**
 * Function prototypes
 */
ssize_t proc_read(struct file *file, char *buf, size_t count, loff_t *pos);

static struct file_operations proc_ops = {
        .owner = THIS_MODULE,
        .read = proc_read,
};
```

```c
/* This function is called when the module is loaded. */
int proc_init(void)
{

        // creates the /proc/jiffies entry
        // the following function call is a wrapper for
        // proc_create_data() passing NULL as the last argument
        proc_create(PROC_NAME, 0, NULL, &proc_ops);
        printk(KERN_INFO "/proc/%s created\n", PROC_NAME);
        return 0;
}

/* This function is called when the module is removed. */
void proc_exit(void) {

        // removes the /proc/jiffies entry
        remove_proc_entry(PROC_NAME, NULL);
        printk( KERN_INFO "/proc/%s removed\n", PROC_NAME);
}

ssize_t proc_read(struct file *file, char __user *usr_buf,
                                        size_t count, loff_t *pos)
{
        int rv = 0;
        char buffer[BUFFER_SIZE];
        static int completed = 0;
        if (completed) {
                completed = 0;
                return 0;
        }
        completed = 1;
        rv = sprintf(buffer, "Current jiffies: %lu\n", jiffies);
        // copies the contents of buffer to userspace usr_buf
        copy_to_user(usr_buf, buffer, rv);
        return rv;
}
/* Macros for registering module entry and exit points. */
module_init( proc_init );
module_exit( proc_exit );

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Jiffies Module");
MODULE_AUTHOR("SGG");
```

Hình 1: jiffies.c

2.2 Module seconds

Yêu cầu: *Thiết kế một kernel module cho HĐH Linux tạo ra file /proc/seconds khi module được nạp, khi chạy lệnh*

```
$ cat /proc/seconds
```

sẽ hiển thị thời gian tính bằng giây kể từ khi module được nạp lên hệ thống. Xóa file /proc/seconds khi module được bỏ khỏi hệ thống.

Answer: Tốc độ mà timer ticks được lưu trong biến HZ trong file <asm/param.h>. Kết hợp với biến jiffies ta có thể tính được tính bằng giây kể từ khi module được nạp vào hệ thống.
Code:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include <linux/jiffies.h>
#include <asm/param.h>

unsigned long start;

#define BUFFER_SIZE 128

#define PROC_NAME "seconds"
#define MESSAGE "Seconds\n"


/**
    * Function prototypes
    */
ssize_t proc_read(struct file *file, char *buf, size_t count, loff_t *pos);

static struct file_operations proc_ops = {
        .owner = THIS_MODULE,
        .read = proc_read,
};


/* This function is called when the module is loaded. */
int proc_init(void)
{

        // creates the /proc/seconds entry
        // the following function call is a wrapper for
        // proc_create_data() passing NULL as the last argument
        proc_create(PROC_NAME, 0, NULL, &proc_ops);
        printk(KERN_INFO "/proc/%s created\n", PROC_NAME);
```

```
        start = jiffies;

    return 0;
}

/* This function is called when the module is removed. */
void proc_exit(void) {

        // removes the /proc/seconds entry
        remove_proc_entry(PROC_NAME, NULL);

        printk( KERN_INFO "/proc/%s removed\n", PROC_NAME);
}

ssize_t proc_read(struct file *file, char __user *usr_buf,
                                        size_t count, loff_t *pos)
{
        int rv = 0;
        char buffer[BUFFER_SIZE];
        static int completed = 0;

        if (completed) {
                completed = 0;
                return 0;
        }
        completed = 1;
        rv = sprintf(buffer, "Seconds elapsed: %lu\n", (jiffies - start)/HZ);

        // copies the contents of buffer to userspace usr_buf
        copy_to_user(usr_buf, buffer, rv);

        return rv;
}

/* Macros for registering module entry and exit points. */
module_init( proc_init );
module_exit( proc_exit );

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Seconds Module");
MODULE_AUTHOR("SGG");
```

Hình 2: seconds.c

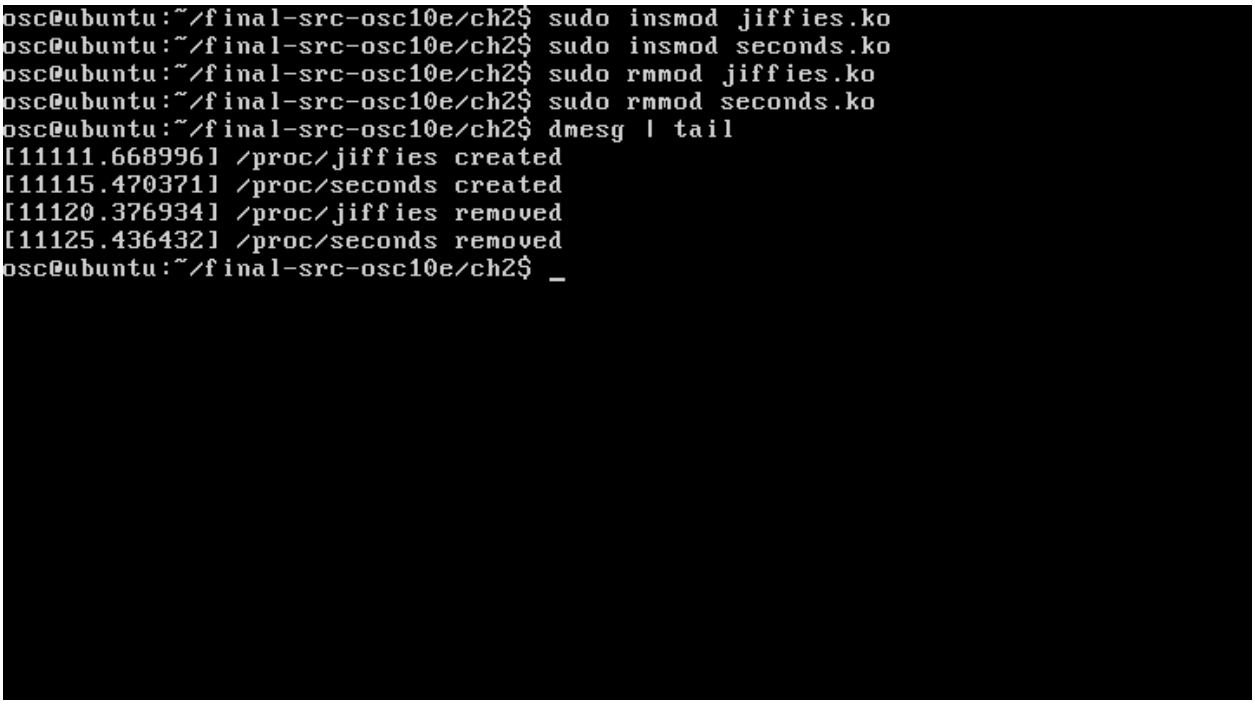
2.3 Kết quả

```
osc@ubuntu:~/final-src-osc10e/ch2$ ls
hello.c  jiffies.c  Makefile  seconds.c  simple.c
osc@ubuntu:~/final-src-osc10e/ch2$ make
make -C /lib/modules/4.4.0-21-generic/build M=/home/osc/final-src-osc10e/ch2 mod
ules
make[1]: Entering directory '/usr/src/linux-headers-4.4.0-21-generic'
  CC [M]  /home/osc/final-src-osc10e/ch2/seconds.o
  CC [M]  /home/osc/final-src-osc10e/ch2/jiffies.o
  Building modules, stage 2.
  MODPOST 2 modules
  CC      /home/osc/final-src-osc10e/ch2/jiffies.mod.o
  LD [M]  /home/osc/final-src-osc10e/ch2/jiffies.ko
  CC      /home/osc/final-src-osc10e/ch2/seconds.mod.o
  LD [M]  /home/osc/final-src-osc10e/ch2/seconds.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.4.0-21-generic'
osc@ubuntu:~/final-src-osc10e/ch2$ ls
hello.c     jiffies.mod.c  Makefile        seconds.c      seconds.mod.o
jiffies.c   jiffies.mod.o  modules.order   seconds.ko     seconds.o
jiffies.ko  jiffies.o      Module.symvers  seconds.mod.c  simple.c
osc@ubuntu:~/final-src-osc10e/ch2$ _
```

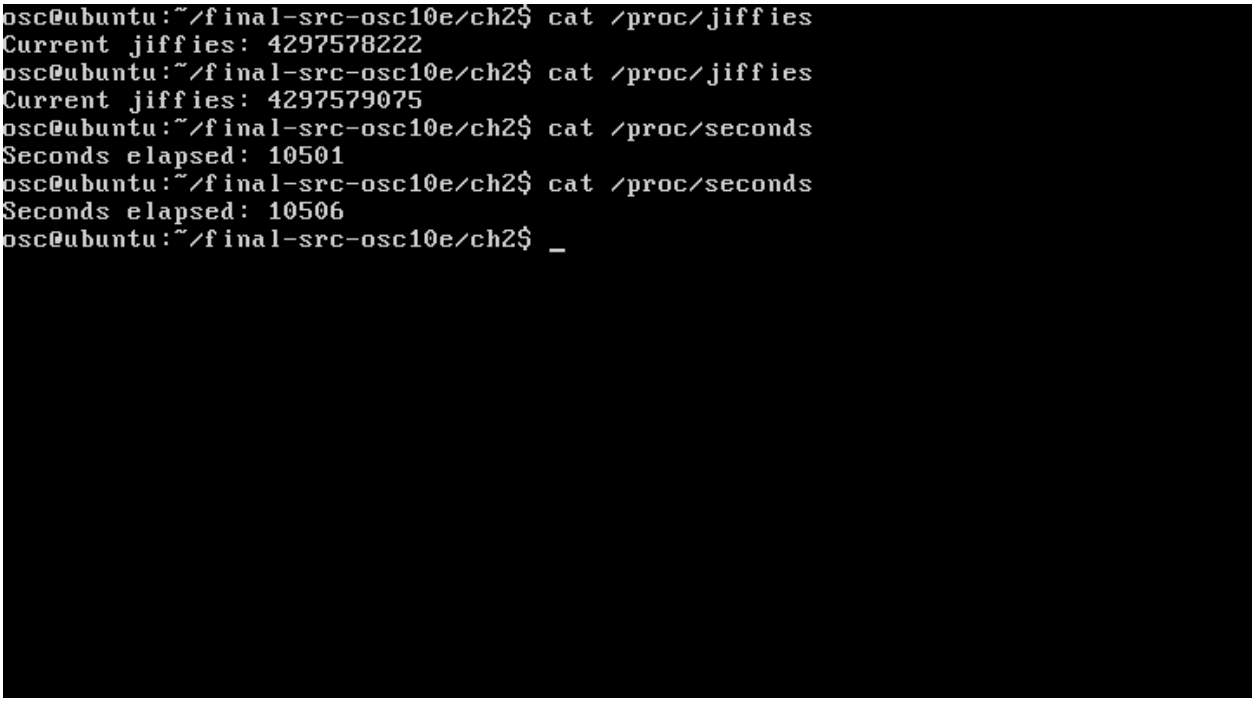
Hình 3: Tạo modules

Tài liệu

[1] A. Silberschatz, P.B. Galvin, G. Gagne (2018) *Operating System Concepts*, John Wiley & Sons, Inc, 10th ed.



Hình 4: Nạp vào và rời bỏ modules



Hình 5: Chạy modules