ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

MÔN HỌC: HỆ ĐIỀU HÀNH

CHỦ ĐỀ: TẠO CHARACTER DEVICE HỖ TRỢ READ & WRITE

Danh sách thành viên

1. Nguyễn Huỳnh Thảo Nhi - 1712117
2. Nguyễn Anh Thư - 1712177

GVHD: ..... Thầy Lê Giang Thanh
..... Cô Nguyễn Thị Thanh Huyền
Lớp: ..... Hệ điều hành CQ2017/21

Năm học: 2019 – 2020

Mục lục

I. TỔNG QUAN ĐỒ ÁN:

1. Yêu cầu:

- Mục tiêu hiểu về Linux Character Device và hệ thống quản lý file và device trong linux, giao tiếp giữa tiến trình ở user space và kernel space, đồng bộ các tiến trình dùng semaphore.
    - Tạo một character device có tên /dev/mylog hỗ trợ hai thao tác read và write
        - write sẽ cho phép tiến trình ghi thêm dữ liệu vào phần cuối của buffer.
        - read sẽ bắt đầu đọc dữ liệu từ đầu buffer và tăng offset, khi không còn dữ liệu để đọc read sẽ bị block cho tới khi có một tiến trình khác ghi thêm dữ liệu mới.

II. PHÂN CÔNG CÔNG VIỆC:

STT	Công việc	Thành viên thực hiện
1	Cài đặt chương trình	Nguyễn Huỳnh Thảo Nhi
2	Tìm hiểu, nghiên cứu đề, test chương trình	Nguyễn Anh Thư
3	Hoàn thành báo cáo và tổng hợp	Nguyễn Huỳnh Thảo Nhi Nguyễn Anh Thư

III. MÔI TRƯỜNG CÀI ĐẶT:

- Hệ điều hành : Ubuntu 19.04 LTS 64 bit.
- Ngôn ngữ lập trình : C

IV. TÌM HIỂU CÁC KHÁI NIỆM:

1. Linux Kernel Module:

- Linux kernel module là một file với tên mở rộng là (.ko). Nó sẽ được lắp vào hoặc tháo ra khỏi kernel khi cần thiết. Việc thiết kế driver theo kiểu loadable module mang lại 3 lợi ích:
    - Giúp giảm kích thước kernel. Do đó, giảm sự lãng phí bộ nhớ và giảm thời gian khởi động hệ thống.
    - Không phải biên dịch lại kernel khi thêm mới driver hoặc khi thay đổi driver.
    - Không cần phải khởi động lại hệ thống khi thêm mới driver. Trong khi đối với Windows, mỗi khi cài thêm driver, ta phải khởi động lại hệ thống, điều này không thích hợp với các máy server.

2. Character Device Driver:

- Driver là một trình điều khiển có vai trò điều khiển, quản lý, giám sát một thực thể nào đó dưới quyền của nó. Bus driver làm việc với một đường bus, device driver làm việc với một thiết bị (chuột, bàn phím, màn hình, đĩa cứng, camera, …)
- Trên Linux, device driver cung cấp một giao diện "system call" (giao diện gọi các hàm hệ thống) đến tầng ứng dụng cho người dùng; đây được coi là một ranh giới giữa tầng nhân (kernel space) và tầng người dùng (user space) của Linux
- Tùy thuộc vào đặc trưng của của driver với hệ điều hành, driver trên Linux được phân chia thành 3 loại
    - Packet-oriented or the network vertical (driver hướng gói dữ liệu)
    - Block-oriented or the storage vertical (driver hướng khối dữ liệu)
    - Byte-oriented or the character vertical (driver hướng byte/ký tự)
- Hầu hết các thiết bị đều thuộc kiểu thiết bị hướng byte (byte-oriented), do vậy hầu hết các device driver là các character device drivers.

V. THIẾT KẾ VÀ TỔ CHỨC CHARACTER DEVICE DRIVER HỖ TRỢ READ & WRITE:

1. Khai báo các thư viện cần thiết:


```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/version.h>
#include <linux/types.h>
#include <linux/kdev_t.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/cdev.h>
#include <asm/uaccess.h>
#include <linux/slab.h>
#include <linux/semaphore.h>
#include <linux/init.h>
#include <linux/miscdevice.h>
#include <linux/platform_device.h>
#include <linux/rtc.h>
#include <linux/sched.h>
```

2. Khai báo các biến toàn cục:

```
static dev_t first;
static struct cdev c_dev;
static struct class *cl;
static char c;

static char write_time = 0;
static char* message;
static int length = 0;

static int open_count = 0;
static int buffer_empty_slots;

static struct semaphore full;
static struct semaphore empty;

static struct semaphore read_op_mutex;
static struct semaphore write_op_mutex;
```

3. Khởi tạo ofcd_init() và ofcd_exit():

```
static int __init ofcd_init(void)
{
        printk(KERN_INFO "DRIVER LOADED SUCCESSFULLY");
        if (alloc_chrdev_region (&first, 0, 1, "Shweta") < 0)
                return -1;
        if ((cl = class_create(THIS_MODULE, "chardrv")) == NULL)
        {
                unregister_chrdev_region (first, 1);
                return -1;
        }
        if (device_create(cl, NULL, first, NULL, "mylog") == NULL)
        {
                class_destroy(cl);
                unregister_chrdev_region(first, 1);
                return -1;
        }
        cdev_init(&c_dev, &pugs_fops);
        if (cdev_add(&c_dev, first, 1) == -1)
        {
                device_destroy(cl, first);
                class_destroy(cl);
                unregister_chrdev_region(first, 1);
                return -1;
        }

        sema_init(&full, 0);
        sema_init(&empty, 1);
        sema_init(&read_op_mutex, 1);
        sema_init(&write_op_mutex, 1);

        buffer_empty_slots = 1;
        open_count = 0;
        return 0;
}
static void __exit ofcd_exit(void)
{
        cdev_del(&c_dev);
        device_destroy(cl, first);
        class_destroy(cl);
        unregister_chrdev_region(first, 1);
        kfree(message);
        printk(KERN_INFO "DRIVER UNLOADED SUCCESSFULLY");
}
module_init(ofcd_init);
module_exit(ofcd_exit);
```

4. Khởi tạo thông tin lớp thiết bị:

```
static struct file_operations pugs_fops = {
        .owner = THIS_MODULE,
        .open = my_open,
        .release = my_close,
        .read = my_read,
        .write = my_write,
};
```

5. Bổ sung các thao tác với file device:

```
static int my_open(struct inode *i, struct file *f)
{
        printk(KERN_INFO "Driver: open()\n");
        open_count++;
        return 0;
}
```

```c
static int my_close(struct inode *i, struct file *f)
{
        printk(KERN_INFO "Driver: close()\n");
        open_count--;
        return 0;
}


static ssize_t my_read(struct file *f, char __user *buf, size_t len, loff_t *off)
{
        printk(KERN_INFO "Driver: read() %s string\n", message);

        if((*off) > 0)
                return 0;

        down_interruptible(&read_op_mutex);
        down_interruptible(&full);

        copy_to_user(buf, message, length);

        *off += strlen(buf);
        buffer_empty_slots++;
        up(&empty);
        up(&read_op_mutex);

        return strlen(buf);
}



static ssize_t my_write(struct file *f, const char __user *buf, size_t len, loff_t *off)
{
        printk(KERN_INFO "Driver: write() %d\n", length);

        down_interruptible(&write_op_mutex);
        down_interruptible(&empty);

        if (length == 0) {
                message = (char*) kmalloc(len * sizeof(char), GFP_KERNEL);
                length += len;
        } else {
                length += len;
                message = (char*) krealloc(message, length*sizeof(char), GFP_KERNEL);
        }

        int pos = length-len;
        pos -= write_time;
        write_time++;
        copy_from_user(message + pos, buf, len);

        buffer_empty_slots--;
        up(&full);
        up(&write_op_mutex);

        return len;
}
```

VI. HIỆN THỰC CHARACTER DEVICE DRIVER HỖ TRỢ READ & WRITE:

1. Viết Makefile và chạy make để biên dịch driver tạo ra file driver.ko:

```
nganhthu99@nganhthu99-VirtualBox:~/driver$ ls
driver.c  Kbuild  Makefile
nganhthu99@nganhthu99-VirtualBox:~/driver$ make
make -C /lib/modules/`uname -r`/build M=`pwd`
make[1]: Entering directory '/usr/src/linux-headers-5.0.0-29-generic'
  CC [M]  /home/nganhthu99/driver/driver.o
/home/nganhthu99/driver/driver.c: In function ‘my_write’:
/home/nganhthu99/driver/driver.c:85:2: warning: ISO C90 forbids mixed declarations and code [-Wdeclaration-after-statement]
  os = length-len;
  ^~
At top level:
/home/nganhthu99/driver/driver.c:21:13: warning: ‘c’ defined but not used [-Wunused-variable]
 static char c;
             ^
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/nganhthu99/driver/driver.mod.o
  LD [M]  /home/nganhthu99/driver/driver.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.0.0-29-generic'
nganhthu99@nganhthu99-VirtualBox:~/driver$ ls
driver.c  driver.ko  driver.mod.c  driver.mod.o  driver.o  Kbuild  Makefile  modules.order  Module.symvers
nganhthu99@nganhthu99-VirtualBox:~/driver$
```

2. Nạp driver vào nhân hệ thống sử dụng lệnh insmod:

```
nganhthu99@nganhthu99-VirtualBox:~$ cd driver
nganhthu99@nganhthu99-VirtualBox:~/driver$ ls
c Files .c  driver.ko  driver.mod.c  driver.mod.o  driver.o  Kbuild  Makefile  modules.order  Module.symvers
nganhthu99@nganhthu99-VirtualBox:~/driver$ sudo insmod driver.ko
[sudo] password for nganhthu99:
nganhthu99@nganhthu99-VirtualBox:~/driver$ lsmod
Module                  Size  Used by
driver                 16384  0
```

3. Test driver:

- **Chạy lệnh sudo –i.**

```
root@nganhthu99-VirtualBox:/home/nganhthu99/driver# echo -n "Hello from the other side. " > /dev/mylog
root@nganhthu99-VirtualBox:/home/nganhthu99/driver# cat /dev/mylog
Hello from the other side. root@nganhthu99-VirtualBox:/home/nganhthu99/driver# cat /dev/mylog
^C
root@nganhthu99-VirtualBox:/home/nganhthu99/driver# echo -n "Hello there!" > /dev/mylog
root@nganhthu99-VirtualBox:/home/nganhthu99/driver# cat /dev/mylog
Hello from the other side.Hello there!root@nganhthu99-VirtualBox:/home/nganhthu99/driver#
```

4. Gỡ bỏ driver:

- **Chạy lệnh insmod driver.ko và kiểm tra dmesg log.**

```
[T7 Thg 1 11 15:39:19 2020] DRIVER LOADED SUCCESSFULLY
[T7 Thg 1 11 15:39:45 2020] Driver: open()
[T7 Thg 1 11 15:39:45 2020] Driver: write() 0
[T7 Thg 1 11 15:39:45 2020] Driver: close()
[T7 Thg 1 11 15:39:50 2020] Driver: open()
[T7 Thg 1 11 15:39:50 2020] Driver: read() Hello from the other side. \xb9\xf5 string
[T7 Thg 1 11 15:39:50 2020] Driver: read() Hello from the other side. \xb9\xf5 string
[T7 Thg 1 11 15:39:50 2020] Driver: close()
[T7 Thg 1 11 15:40:04 2020] Driver: open()
[T7 Thg 1 11 15:40:04 2020] Driver: read() Hello from the other side. \xb9\xf5 string
[T7 Thg 1 11 15:40:05 2020] Driver: close()
[T7 Thg 1 11 15:40:20 2020] Driver: open()
[T7 Thg 1 11 15:40:20 2020] Driver: write() 27
[T7 Thg 1 11 15:40:20 2020] Driver: close()
[T7 Thg 1 11 15:40:26 2020] Driver: open()
[T7 Thg 1 11 15:40:26 2020] Driver: read() Hello from the other side.Hello there! string
[T7 Thg 1 11 15:40:26 2020] Driver: read() Hello from the other side.Hello there! string
[T7 Thg 1 11 15:40:26 2020] Driver: close()
[T7 Thg 1 11 15:40:55 2020] DRIVER UNLOADED SUCCESSFULLY
```

VII. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH:

CHỨC NĂNG	MỨC ĐỘ HOÀN THÀNH
Tạo một character device có tên /dev/mylog hỗ trợ hai thao tác read và write như sau:	
- write sẽ cho phép tiến trình ghi thêm dữ liệu vào phần cuối của buffer.	**100%**

- read sẽ bắt đầu đọc dữ liệu từ đầu buffer và tăng offset, khi không còn dữ liệu để đọc read sẽ bị block cho tới khi có một tiến trình khác ghi thêm dữ liệu mới	**100%**
TỔNG KẾT	**100%**

--------- The End ---------