TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN
MÔN: **HỆ ĐIỀU HÀNH**
LỚP: 16CNTN

ĐỒ ÁN I

LẬP TRÌNH LINUX KERNEL MODULE

LÊ THÀNH CÔNG

MSSV: 1612842

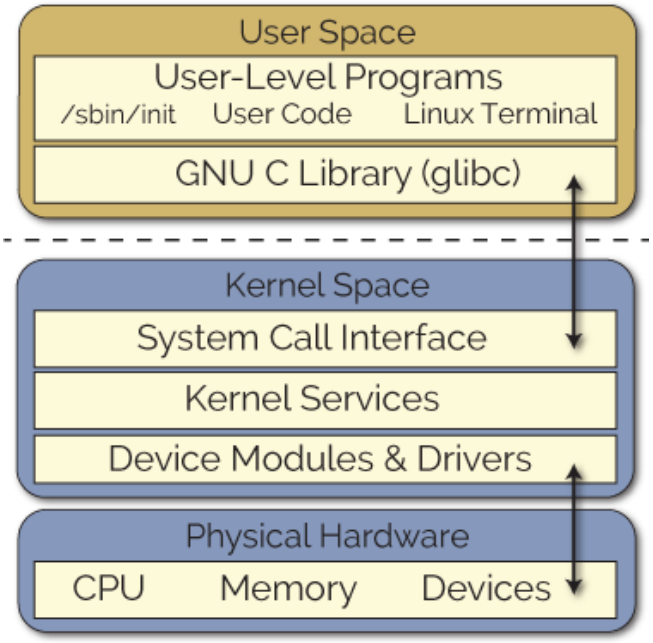
TP.HCM, ngày 29 tháng 09 năm 2018

Mục lục

I. Nội dung

1. Linux kernel module

- "Kernel module" là một phần mở rộng cho hệ điều hành. Module nằm trong cùng một mức đặc quyền của hệ điều hành (mức cao nhất) và do đó có thể truy cập mọi tài nguyên của hệ thống. Trong hệ điều hành Linux, một module không gì hơn một chương trình C với một giao diện được xác định rõ ràng để giao tiếp với các tiến trình của người dùng và với các phần khác của hệ điều hành.
- "Device driver" là một kernel module chuyên về giao tiếp I/O với một số loại thiết bị. Từ "device" có một ý nghĩa rất rộng và nó không chỉ đề cập đến một hệ thống bên ngoài hoặc vật lý. Nói chung, "device" là một số loại tài nguyên như đĩa mềm, máy in, chuột, nhưng cũng có thể là vùng bộ nhớ đặc biệt, một virtual terminal hoặc message box.
- "Loadable kernel module (LKM)" là một kỹ thuật có thể thêm code vào hoặc lấy code ra khỏi Linux kernel ngay tại run time. Điều này giúp kernel giao tiếp được với phần cứng mà không cần biết nó làm việc như thế nào. Nếu không có khả năng đó thì Linux kernel sẽ rất lớn và sẽ phải build lại kernel mỗi lần muốn thêm phần cứng hay update một device driver. Tuy nhiên, điểm bất lợi của LKM đó là phải được duy trì cho mỗi thiết bị. LKM được nạp tại run time và không thực thi trong user space, về cơ bản thì chúng là 1 phần của kernel.



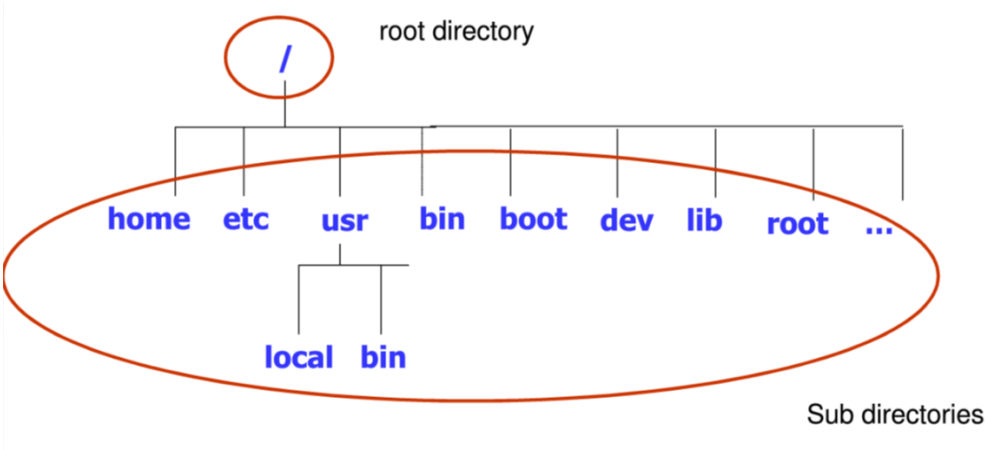
Hình 1. Linux user space và kernel space

- Như hình trên, kernel module chạy trong kernel space còn ứng dụng người dùng chạy trong user space. Kernel space và user space có riêng biệt không gian địa chỉ và không trùng lắp. Thông qua việc sử dụng system call mà các kernel services được cung cấp cho user space. Kernel cũng ngăn các ứng dụng trên user space xung đột với nhau hoặc truy cập các tài nguyên hạn chế.

2. Hệ thống quản lí file và device trong Linux

a. *Hệ thống quản lí file trong Linux*

- Linux sử dụng hệ thống file có cấu trúc cây phân cấp: có thứ bậc, giống một cấu trúc cây từ trên xuống dưới, với root (/) tại cơ sở của hệ thống file và tất cả các thư mục khác trải ra từ đó, không có khái niệm ổ đĩa như Windows.



Hình 2. Cây thư mục hệ thống quản lí file trên Linux

- Cách bố trí thư mục tuân theo FHS (Filesystem Hierarchy Standard):
  - /bin: chứa các tập tin lệnh chủ yếu
  - /boot: chứa các tập tin tĩnh của bộ nạp khởi động
  - /dev: chứa các tập tin thiết bị
  - /etc: chứa các file cấu hình hệ thống
  - /lib: chứa các kernel module và các thư viện được chia sẻ chủ yếu
  - /media: điểm lắp đặt cho các phương tiện tháo lắp vật lý
  - /mnt: điểm lắp đặt để lắp đặt một hệ thống tập tin tạm thời
  - /opt: các gói phần mềm ứng dụng bổ sung
  - /proc: thư mục giả giúp truy xuất thông tin trạng thái của hệ thống
  - /sbin: chứa các tập tin lệnh hệ thống
  - /srv: dữ liệu cho các dịch vụ được hệ thống cung cấp
  - /tmp: nơi lưu các tập tin tạm
  - /usr: hệ thống phân cấp thứ cấp
  - /var: lưu dữ liệu biến đổi, các log file, hàng đợi in
  - /home: thư mục cá nhân của người dùng thông thường
  - /root: thư mục cá nhân của tài khoản root
- Như vậy, một hệ thống file trong Linux là một tập hợp của các file và thư mục có các đặc tính:
  - Nó có một thư mục gốc (/) mà chứa các file và thư mục khác.
  - Mỗi file và thư mục được xác định duy nhất bởi tên của nó, thư mục mà trong đó nó cư trú, và một sự nhận diện duy nhất, được gọi theo cách đặc trưng là inode.
  - Theo quy ước, thư mục gốc có số inode là 2 và thư mục lost+found có số inode là 3. Số inode 0 và 1 không được sử dụng. Các số inode có thể được gửi bởi trình xác định trong chức năng -i của lệnh ls.

    - Nó có đặc tính khác nữa là tự chứa. Không có sự phụ thuộc giữa một hệ thống file này với một hệ thống file khác.
- **inode** là một cấu trúc dữ liệu chứa các metadata của mỗi file, thư mục trong các hệ thống file Linux. Trong một inode có các metadata sau:
    - Dung lượng file tính bằng bytes.
    - Device ID: mã số thiết bị lưu file.
    - User ID: mã số chủ nhân của file.
    - Group ID: mã số nhóm của chủ file.
    - File mode: gồm kiểu file và các quyền truy cập file (permissions).
    - Hệ thống phụ và các cờ hạn chế quyền truy cập file.
    - Timestamps: các mốc thời gian khi bản thân inode bị thay đổi (ctime), nội dung file thay đổi (mtime) và lần truy cập mới nhất (atime).
    - Link count: số lượng hard links trỏ đến inode.
    - Các con trỏ (từ 11-15 con trỏ) chỉ đến các blocks trên ổ cứng dùng lưu nội dung file. Theo các con trỏ này mới biết file nằm ở đâu để đọc nội dung.
- Chú ý trong inode:
    1. Inode không chứa tên file, thư mục.
    2. Các con trỏ là thành phần quan trọng nhất: nó cho biết địa chỉ các block lưu nội dung file và tìm đến các block đó có thể truy cập được nội dung file.

b. Quản lý device trong Linux

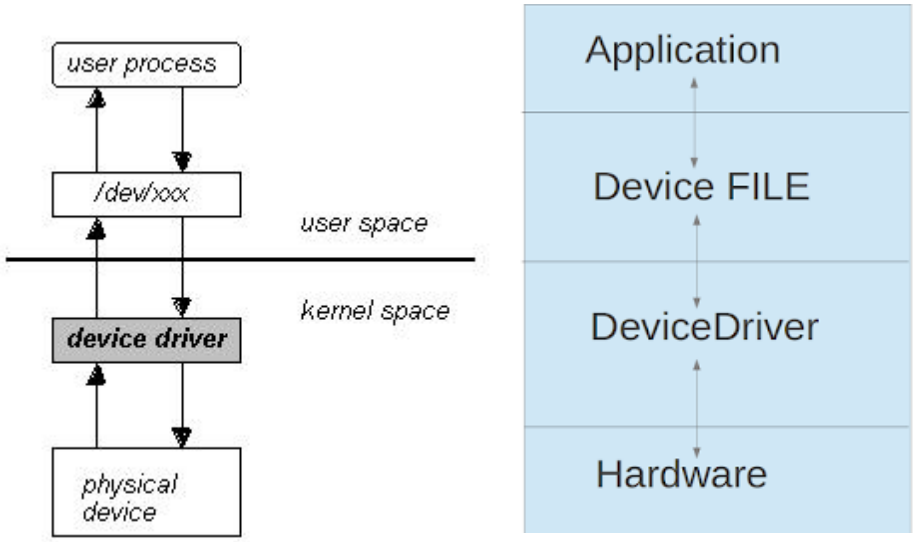
- Trong hệ thống Linux, các thiết bị được truy xuất thông qua các device file
- Thư mục /dev quản lý tất cả các device file gồm character device file và block device file
- Mỗi device file gồm 2 chỉ số:
    - Major number: trỏ tới loại driver tương ứng với thiết bị trong Linux kernel
    - Minor number: xác định thiết bị cụ thể, phụ thuộc vào device
- Về "device" trong Linux thì chúng được chia thành 3 loại device. Các module chỉ thực thi một trong các loại này và chúng được phân loại như sau:
    - Character device:
        - Là device có thể được truy cập dưới dạng một luồng các bytes, như một file
        - Việc đọc và ghi dữ liệu được thực hiện ngay lập tức trong luồng từng ký tự một
        - Chúng được đặt trong thư mục /dev
        - Ví dụ: /dev/tty (terminal)
    - Block device
        - Là device tương tự character device với các tệp thông thường và có thể truy cập bằng bội số của một block (có thể là một mảng đệm dữ liệu được lưu trong bộ nhớ cache), mà kích thước block đó dựa trên device.
        - Loại này bao gồm hard disks và floppy disks với kích thước của 1 block là 2 sectors (1KB).
        - Ví dụ: /dev/floppy/0u1440 (first floppy, 1.4MB)
    - Network device
        - Chịu trách nhiệm trao đổi dữ liệu với các host khác
        - Không được hiển thị trong file system, không tìm thấy được trong thư mục /dev

- Một số ví dụ về device file:

Thiết bị	Device files
Terminals	/dev/tty1, /dev/tty2, …
IDE Hard disk partitions	/dev/hda1, /dev/hda2, /dev/hdb1, …
SCSI Hard disk partitions	/dev/sda1, /dev/sdb1, /dev/sdb2, …
SCSI cdroms	/dev/sr0, /dev/sr1
Floppy disk	/dev/fd0
Printer	/dev/lp0

3. Giao tiếp giữa tiến trình ở User space và Kernel space

- Đối với Linux, hầu như tất cả resource trong hệ thống đều như những tập tin bao gồm cả các devices. Mỗi device file liên kết với một device driver hay kernel module riêng biệt trong kernel space.



Hình 3. Giao tiếp giữa user space và kernel space

- Sự khác biệt giữa device driver và device file:
  - Device driver là một chương trình phần mềm nằm dưới tầng kernel, nhiệm vụ của nó là nhận các chỉ thị từ user space để điều khiển hardware hoạt động. Device driver được viết theo cơ chế module của kernel, theo một form nhất định được kernel quy định rõ. Để có thể điều khiển được hardware, nó thực hiện việc đọc và ghi các thanh ghi register của phần cứng. Khi compile source code driver thì kết quả cho ra một file .ko (kernel object), driver hệ thống nằm trong gói source code của kernel.
  - Device file là một file đặc biệt nằm trên user space, có nhiệm vụ đại diện cho một thiết bị phần cứng, hay nói cách khác nó là interface giữa user application với device driver. Device file được biểu diễn với dạng **/dev/xxx**.
- Bởi vì device là các file nên có các thao tác trên đó như một file: open(), read(), write(), close(). Cứ mỗi lần thực thi một trong các thao tác đó từ user space lên trên device file (/dev/xxx) thì kernel module liên kết với device file đó cũng phải thực thi thao tác đó.

- Ví dụ: fd = open("/dev/hda", O_RDONLY);
  Câu lệnh trên thực hiện open /dev/hda (first hard disk) với quyền read only. Khi đó hệ điều hành biết /dev/hda là device file nên kernel module liên kết với device file này sẽ gọi dev_open() để thực thi trên module đó. Tùy thuộc vào device driver để khởi tạo thiết bị hoặc trả về lỗi.
- Tóm lại, một character device hay device file (/dev/xxx) có thể được dùng như trung gian để truyền nhận thông tin giữa chương trình Linux ở user space và Loadable kernel module (LKM) chạy trong kernel space. Tuy nhiên, vì tính chất của character device truyền nhận tức thì nên khi truyền dữ liệu có thể xảy ra vấn đề đồng bộ hóa, tức là có thể một lúc có 2 hay nhiều chương trình ở user space muốn gửi dữ liệu vào LKM chạy trong kernel space nên dữ liệu có thể bị ghi đè. Có thể sử dụng mutex locks đề cập ở phần sau để giải quyết vấn đề đó.
- Các hàm tương ứng với open(), read(), write(), close() ở user space mà kernel cần có đối với một character device:
  - dev_open(): gọi mỗi lần device được mở từ user space
  - dev_read(): gọi mỗi lần data được gửi từ device đến user space
  - dev_write(): gọi mỗi lần data được gửi từ user space đến kernel space
  - dev_release(): gọi khi device đóng trong user space

4. Module dùng để tạo ra số ngẫu nhiên

- Một module luôn có thông tin về class name và device name. Khi cài đặt random_number.ko vào hệ thống thì device này sẽ xuất hiện tại /sys/class/random_class/random_number và device file tương ứng tại /dev/random_number.

```
#define DEVICE_NAME "random_number"
#define CLASS_NAME "random_class"
```

- Module phải có hàm init() và exit(), trong đó hàm init() có nhiệm vụ khởi tạo hoặc đăng ký các giá trị ban đầu cần thiết khi install module và hàm exit() có nhiệm vụ tháo và hủy đăng ký khi remove module. Nếu muốn tạo một character device phải có đăng ký majorNumber, device class, device driver với hệ thống khi init(), vì vậy khi exit() tương tự cần hủy những gì đã đăng ký majorNumber, device class, device driver.

```
static int __init random_number_init(void){
    mutex_init(&random_number_mutex);       /// Initialize the mutex lock dynamically at runtime

    printk(KERN_INFO "RandomNumber: Initializing the RandomNumber LKM\n");

    //try to dynamically a major number for the device
    majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
    if (majorNumber<0){
      printk(KERN_ALERT "RandomNumber failed to register a major number\n");
      return majorNumber;
   }
   printk(KERN_INFO "RandomNumber: registered correctly with major number %d\n", majorNumber);

   //register device class
   randomClass=class_create(THIS_MODULE, CLASS_NAME);
   if (IS_ERR(randomClass)){
        unregister_chrdev(majorNumber,DEVICE_NAME);
        printk(KERN_ALERT "Failed to register a device class\n");
        return PTR_ERR(randomClass);
   }
   printk(KERN_INFO "RandomNumber: device class registerd correctly\n");
```

```
    //register the device driver
    randomDevice = device_create(randomClass,NULL, MKDEV(majorNumber,0),NULL,DEVICE_NAME);
    if (IS_ERR(randomDevice)){
     class_destroy(randomClass);
     unregister_chrdev(majorNumber,DEVICE_NAME);
     printk(KERN_ALERT "Failed to create the device\n");
     return PTR_ERR(randomDevice);
    }
    printk(KERN_INFO "RandomNumber: device class created correctly\n");

    return 0;

}

static void __exit random_number_exit(void){
    device_destroy(randomClass,MKDEV(majorNumber,0));
    class_unregister(randomClass);
    class_destroy(randomClass);
    unregister_chrdev(majorNumber,DEVICE_NAME);
    mutex_destroy(&random_number_mutex);
    printk(KERN_INFO "RandomNumber: Goodbye from the LKM\n");
}
```

- Các thao tác đối với character device như đã đề cập phần trước có các thao tác sau: open, read, write và release. Để một module tạo một character device có thể open và read các số ngẫu nhiên thì ta chỉ cần viết xử lí các hàm dev_open(), dev_read(), dev_release().

```
//we only need to create a character device with open, read, and close operations
//we do not need write operation because we only need to read random number
static struct file_operations fops =
{
    .open = dev_open,
    .read = dev_read,
    .release = dev_release,
};
```

- Trong hàm dev_open(), sử dụng hàm get_random_bytes(&x, sizeof(x)) từ thư viện linux/random.h để phát sinh số ngẫu nhiên. Nếu x là kiểu int (4 bytes) thì get_random_bytes() sẽ điền ngẫu nhiên các bit 0 và 1 vào 32 bit để phát sinh số ngẫu nhiên.

```
static int dev_open(struct inode *inodep, struct file *filep){
    if(!mutex_trylock(&random_number_mutex)){
        printk(KERN_ALERT "RandomNumber: Device in use by another process");
        return -EBUSY;
    }

    numberOpens++;
    printk(KERN_INFO "RandomNumber: Device has been opened %d time(s)\n",numberOpens);

    get_random_bytes(&rand, sizeof(int));
```

- Mục đích của việc để hàm sinh số ngẫu nhiên ở dev_open() để khi mỗi lần chương trình ở user space thực hiện open(/dev/random_number) thì kernel module sẽ phát sinh số ngẫu nhiên khác nhau.

- Ở hàm dev_read() dùng sprintf() để thực hiện ép kiểu int về char* lưu vào biến message. Nhờ đó, chúng ta có thể sử dụng hàm copy_to_user() để thực hiện gửi số random dạng chuỗi đến chương trình ở user space để chương trình đó có thể đọc được và in ra cho user.

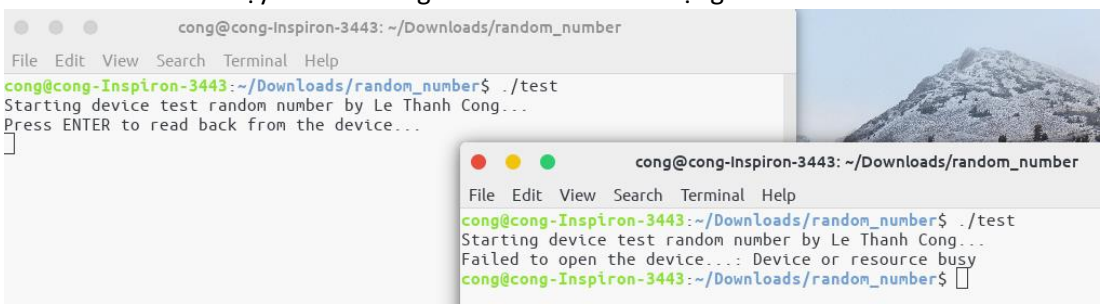
```
static ssize_t dev_read(struct file *filep, char* buffer,size_t len, loff_t *offset){
    int error_count = 0;

   sprintf(message,"%d",rand);

   // copy_to_user has the format ( * to, *from, size) and returns 0 on success
   error_count = copy_to_user(buffer, message, strlen(message));

   if (error_count==0){
      printk(KERN_INFO "RandomNumber: Sent random number to the user successfully\n");
      return 0;
   }
   else {
      printk(KERN_INFO "RandomNumber: Failed to send the random number to the user\n");
      return -EFAULT;
   }
}
```

- Tuy nhiên, nếu 2 chương trình ở user space trở lên thực hiện các thao tác với device file (/dev/xxx) thì đôi khi sẽ bị ghi đè thông tin. Ví dụ nếu nhập chuỗi và ghi xuống kernel module thì chương trình sau sẽ ghi đè chuỗi lên chuỗi của chương trình trước. Do vậy, để giải quyết vấn đề này ta có thể chèn thêm mutex locks vào file code C module (cụ thể random_number.c)

```
static DEFINE_MUTEX(random_number_mutex);

static int __init random_number_init(void){
    ...
    mutex_init(&random_number_mutex);       /// Initialize the mutex lock dynamically at runtime
    ...
}

static void __exit random_number_exit(void){
    ...
    mutex_destroy(&random_number_mutex);
    ...
}

static int dev_open(struct inode *inodep, struct file *filep){
    ...
    if(!mutex_trylock(&random_number_mutex)){
        printk(KERN_ALERT "RandomNumber: Device in use by another process");
        return -EBUSY;
    }
    ...
}

static int dev_release(struct inode *inodep,struct file* filep){
    ...
    mutex_unlock(&random_number_mutex);
    ...
}
```

- Mutex locks giúp tại một thời điểm chỉ một chương trình ở user space có thể thực hiện các thao tác open, read, write, close với device file (/dev/xxx) nên sẽ tránh được việc nhiều chương trình thao tác với device file và gây xáo trộn dữ liệu khi truyền và nhận dữ liệu từ kernel module.
- Kết quả chạy file test kiểm tra khả năng open và read số ngẫu nhiên:

- Mở 2 terminal và chạy file test cùng lúc để kiểm tra tác dụng của mutex locks:



II. Nguồn tham khảo

- Dr. Derek Molly : http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/
- Filesystem in Linux: https://www.gocit.vn/bai-viet/file-va-he-thong-file-tren-linux/
- Linux kernel module: http://cs.smith.edu/~nhowe/262/oldlabs/module.html
- Get_random_bytes function: https://stackoverflow.com/questions/40961482/how-to-use-get-random-bytes-in-linux-kernel-module

-HẾT-