**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

**Môn: Hệ điều hành**

**Đồ án**

**TÌM HIỂU VỀ LINUX**

**Phần 1: Linux kernel module, hệ thống quản lý file và device, giao tiếp giữa tiến trình ở user space và kernel space**

**Sinh viên:**

**Bùi Chí Dũng – 1712364**

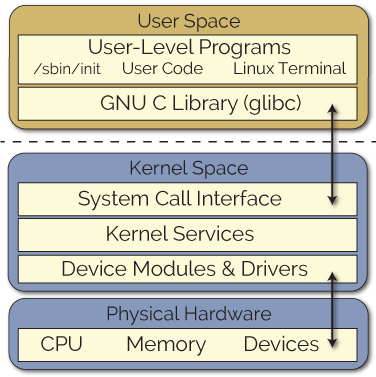
**Phạm Đặng Đăng Khoa - 1712533**

**I. Linux kernel module và hệ thống quản lý file và device trong linux, giao tiếp giữa tiến trình ở user space và kernel space**

**1. Linux kernel module**

**Linux** được thiết kế để làm việc với hang tỉ thiết bị. Nhưng ta không thể đưa tất cả các driver vào trong kernel được, vì sẽ làm cho kích thước kernel rất lớn. Giải pháp cho vấn đề này là thiết kế driver dưới dạng module tách rời với kernel. Trong quá trình hoạt động, driver nào cần thiết sẽ được lắp vào kernel, còn driver nào không cần thiết sẽ bị tháo ra khỏi kernel (gọi là dynamic loading).

**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. Chính vì vậy, nó còn có một tên gọi khác là **loadable kernel module**. Một trong những kiểu loadable kernel module phổ biến đó là driver. Các lợi ích của việc thiết kế driver theo kiểu loadable module là giúp giảm kích thước của kernel, từ 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. Một lợi ích tiếp theo đó là 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 đối với các máy server.



**Loadable kernel module (LKM)** là một cơ chế để thêm hoặc xóa mã khỏi Linux kernel khi chạy. Điều này rất lý tưởng cho các trình điều khiển thiết bị, cho phép kernel giao tiếp với phần cứng mà không cần phải biết phần cứng hoạt động như thế nào. Giải pháp thay thế cho các LKM sẽ là xây dựng mã cho mỗi và mọi trình điều khiển vào nhân Linux.

Nếu không có loadable module này, nhân Linux sẽ rất lớn, vì nó sẽ phải hỗ trợ mọi trình điều khiển cần thiết trên thiết bị BBB. Chúng ta cũng sẽ phải xây dựng lại kernel mỗi lần bạn muốn thêm phần cứng mới hoặc cập nhật trình điều khiển thiết bị. Nhược điểm của các LKM là các tệp trình điều khiển phải được duy trì cho mỗi thiết bị. Các LKM được tải trong thời gian chạy, nhưng chúng không thực thi trong không gian người dùng – về cơ bản chúng là một phần của kernel.

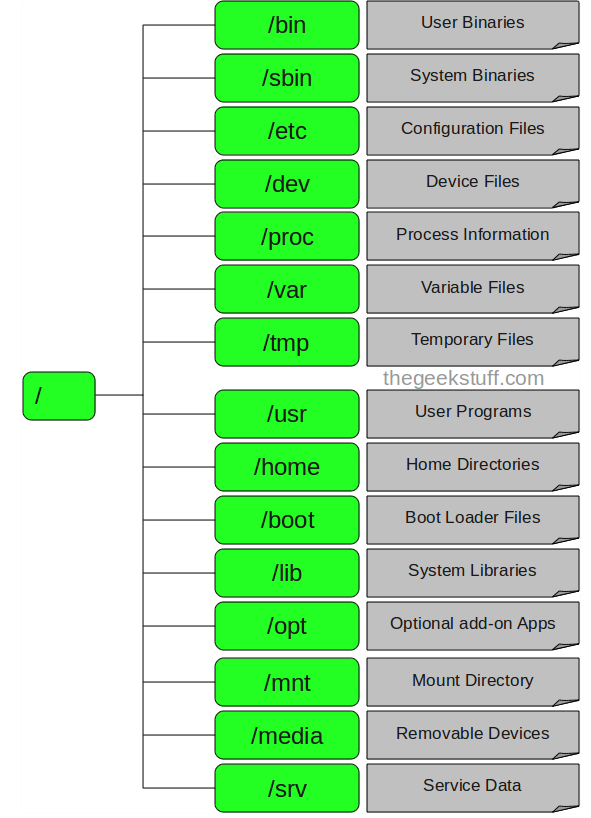
Các **kernel module** chạy trong không gian nhân (**kernel space**) và các ứng dụng chạy trong không gian người dùng (**user space**) được minh hoạt trên hình trên. Cả không gian nhân và không gian người dùng đều có không gian địa chỉ bộ nhớ duy nhất của riêng chúng, và không trùng nhau. Cách tiếp cận này đảm bảo rằng các ứng dụng chạy trong không gian người dùng có giao diện nhất quán về phần cứng, bất kể nền tảng phần cứng. Các dịch vụ kernel (**kernel services**) sau đó được cung cấp cho không gian người dùng theo cách được kiểm soát thông qua việc sử dụng các cuộc gọi hệ thống. Kernel cũng ngăn các ứng dụng không gian người dùng các nhân xung đột với nhau hoặc truy cập các tài nguyên bị hạn chế thông qua việc sử dụng các mức bảo vệ (ví dụ: super user so với quyền của user thông thường).

**2. Hệ thống quản lý file và device trong linux**

Hệ thống các tập tin của **Linux** và **Unix** được tổ chức theo một hệ thống phân bậc tương tự cấu trúc của một cây phân cấp. Bậc cao nhất của hệ thống tập tin là thư mục gốc, được kí hiệu bằng gạch chéo **“/”** (**root directory**).

Đối với các hệ điều hành **Unix** và **Linux**, tất cả các thiết bị kết nối vào máy tính đều được nhận dạng như các tập tin, kể cả những linh kiện như ổ đĩa cứng, các phân vùng đĩa cứng và các ổ USB. Điều này có nghĩa là tất cả các tập tin và thư mục đều nằm dưới thư mục gốc, ngay cả những tập tin biểu tưởng cho các ổ đĩa cứng.

Nằm dưới thư mục gốc **(/)** có một loại các thư mục quan trọng của hệ thống tập tin được công nhận ở tất cả các bản phân phối Linux khác nhau. Dưới đây là danh sách các thư mục thông thường đươc nhìn thấy dưới thư mục gốc **(/):**



Trong đó:

1. **/-Root**: mở từng tập tin và thư mục từ thư mục Root, chỉ có những Root user mới có quyền viết dưới thư mục này. Lưu ý rằng **/root** là thư mục gốc của Root user.

2. **/bin – User Binaries**: chứa các tập tin thực thi nhị phân **(binary executables)**, lệnh Linux phổ biến sử dụng ở chế độ Singer-user mode nằm trong thư mục này và tất cả user trên hệ thống nằm tại thư mục này đều có thể sử dụng lệnh. Ví dụ: ps, ls, ping, grep, cp.

3. **/sbin – Systemp Binaries**: cũng giống như **/bin** hay **/sbin** cũng chứa tập tin thực thi nhị phân **(binary executables)**, lệnh Linux nằm trong thư mục này được sử dụng bởi Admin hệ thống, nhằm mục đích duy trì hệ thống. Ví dụ: iptables, reboot, fdisk, ifconfig, swapon.

4. **/etc – Connfiguration Files**: chứa cấu hình các tập tin cấu hình của hệ thống, các tập tin lệnh để khởi động các dịch vụ của hệ thống, ngoài ra **/etc** còn chưa shell scripts startup và shutdown, sử dụng để chạy/ngừng các chương trình cá nhân. Ví dụ: /etc/resolv.conf, /etc/logrolate.conf.

5. **/dev – Files device**: chứa các tập tin để nhận biết cho các thiết bị của hệ thống (device files), bao gồm thiết bị đầu cuối, USB hoặc các thiết bị được gắn trên hệ thống. Ví dụ: /dev/tty1, /dev/usbmon0.

6. **/proc – Process Information**: chứa các thông tin về **System Process**, đây là hệ thống tập tin giả có chứa thông tin về các quá trình đang chạy, chẵng hạn như thư mục /proc/{pid} có chứa thông tin về quá trình đặc biệt của pid. Đây là một hệ thống tập tin ảo có thông tin về tài nguyên hệ thống, chẵng hạn như /proc/uptime.

7. **/var – Variable Files**: Var là viết tắt của variable file, lưu lại tập tin ghi các số hiệu biến đổi **(variable files)**. Nội dung các tập tin được dự kiến sẽ tăng lên tại thư mục này. Bao gồm: hệ thống tập tin log (/var/log), các gói và các file dữ liệu (/var/lib), email (/var/mail), print queues (/var/spool), look files (/var/lock), các file tạm thời cần khi reboot (/var/tmp).

8. **/tmp – Temporary Files** (các tập tin tạm thời): thư mục chứa các tập tin tạm thời được tạo bởi hệ thống và user, các tập tin tạo thư mục này được xóa khi hệ thống được khởi động lại (reboot).

9. **/usr – User Programs**: chứa các ứng dụng, thư viện, tài liệu và mã nguồn các chương trình thứ cấp. /usr/bin chứa các tập tin của các ứng dụng chính đã được cài đặt cho user. Nếu bạn không tìm thấy user binary tại thư mục /bin, bạn có thể tìm lại thư mục /usr/bin. Ví dụ như at, awk, cc, less, scp. /usr/sbin có chứa các tập tin ứng dụng cho Admin hệ thống. Nếu không tìm thấy hệ nhị phân tại /sbin, có thể tìm tại /usr/sbin. Chẵng hạn như atd, cron, sshd, useradd, userdel. /usr/lib chứa thư viện /usr/bin và /usr/sbin. /usr/local chứa các chương trình user mà bạn cài đặt từ nguồn. Chẵng hạn khi bạn cái đặt apache từ nguồn, apache nằm dưới /usr/local/apache2.

10. **/home – thư mục Home**: thư mục chính lưu trữ các tập tin cá nhân của tất cả user. Ví dụ: /home/leonardbui, /home/nothing.

11. /**boot – Boot Loader Files**: chứa các tập tin cấu hình cho quá trình khởi động hệ thống. Các file Kernel initrd, vmlinux, grub nằm trong /boot. Ví dụ: ntrd.img-2.6.32-24-generic, vmlinux-2.6.32-24-generic.

12. **/lib – Systemp Libraries**: chứa các file thư viện hỗ trợ các thư mục nằm dưới /bin và /sbin. Tên file thư viện có thể là ld\* hoặc lb\*.so.\*. Ví dụ: ld-2.11.1.so, libncurses.so.5.7.

13. **/opt – Optional add-on Applications**: Opt là viết tắt của Optional (tùy chọn). Chứa các ứng dụng Add-on từ các nhà cung cấp, ứng dụng add-on được cài đặt dưới thư mục /opt/ hoặc thư mục /opt/sub.

14. **/mnt: Mount Directory**: gắn kết các thư mục hệ thống tạm thời (thư mục Temporary) nơi Sysadmins có thể gắn kết các file hệ thống.

15. **/media – Removable Media Devices**: gắn kết các thư mục Temporary (thư mục tạm thời) được hệ thống tạo ra khi một thiết bị lưu đồng (removable media) được cắm vào như đĩa CD, Digital Camera, … Ví dụ /media/cdrom for CD-ROM, /media/floppy for floppy drives, /media/cdrecorder for CD writer.

16. **/srv – Service Data**: chứa các service của máy chủ cụ thể liên quan đến dữ liệu. Ví dụ: /srv/cvs chứa dữ liệu liên quan đến CVS

**Hệ thống file** của linux gồm các thành phần:

- **Boot block**: thường chứa đoạn mã để nạp hệ điều hành lúc máy khởi động.

- **Super block**: khối nhớ lưu trữ các thông tin của hệ thống file như: kích thước hệ thống file, số khối còn trống, danh sách khối trống, số inode còn trống, danh sách inode còn trống, trường khóa của danh sách khối inode.

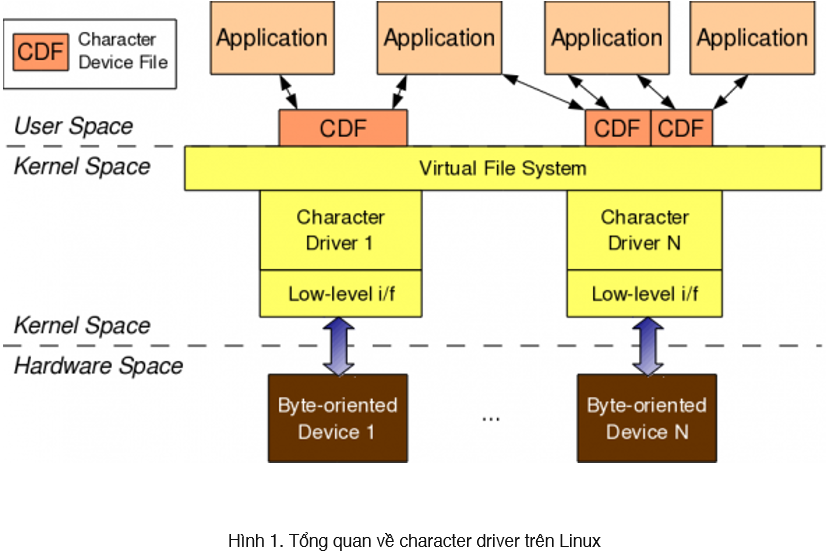
- **Inode table**: bảng chứa các inode của các file.

- **Data block**: Vùng lưu trữ phần dữ liệu của các file.

**Linux** hiển thị cho người dùng hệ thống file dưới dạng cây thư mục với thư mục gốc là **root (/)**, chứa các thư mục con và các file chứa trong các thư mục. Hệ điều hành sẽ quản lý các file thông qua tên file và inode của nó. Mỗi file là một khối dữ liệu được lưu liên tục hoặc không liên tục trên bộ nhớ, và được liên kết với một inode chứa các thông tin về file, device ID, user ID, group ID, kiểu file và các quyền truy cập, timestamps (các thời điểm file bị thay đổi), số hardlink liên kết đến file, và quan trọng nhất là con trỏ trỏ đến vùng nhớ lưu file, Inode không chứa tên file và tên thư mục.

Khi một file được tạo, nó sẽ được gán cho một file nưm và một inode (với inode number duy nhất trong hệ thống). Cặp file name – inode number được lưu thành các mục. Khi người dùng cần truy cập đến một file, hệ điều hành sẽ tìm đến inode tương ứng và thông qua con trỏ vùng nhớ để đến được vị trí file lưu trong bộ nhớ.

Ngoài quản lý file, hệ điều hành còn có hệ thống quản lý các device, thông qua device file, device driver và device thật.



- **Device file** là các file logic, nó nằm ở không gian người dùng (user space), đại diện cho một device nằm ở không gian phần cứng. Khi ứng dụng cần thao tác với một thiết bị thì ứng dụng nó sẽ làm việc thông qua device file tương ứng.

- **Device driver**: là các thành phần nằm ở kernel của hệ điều hành, có vai trò tương tự như kernel module, giúp liên kết giữa device file và các thiết bị phần cứng. Khi các device file tiếp nhận yêu cầu từ ứng dụng, Virtual file system trong kernel diễn giải và gửi thông tin đến device driver tương ứng.

- **Device** là thiết bị phần cứng, mỗi device sẽ có một device driver riêng nằm trong tầng nhân.

**3. Giao tiếp giữa tiến trình ở user-space và kernel space**

Khi một chương trình khởi tạo trên User Mode, Windows sẽ tạo một tiến trình (process) cho chương trình đó. Một tiến trình cung cấp cho chương trình đó một không gian địa chỉ ảo (Virtual address space). Bởi vì mỗi không gian địa chỉ ảo của mỗi tiến trình là riêng biệt, nên một chương trình khó hay còn có thể nói là không thể thay đổi dữ liệu về một chương trình khác.

Không gian địa chỉ ảo của mỗi tiến trình ở User Mode bị giới hạn. Một chương trình chạy tại User Mode sẽ không xử lý được những địa chỉ ảo (Virtual address) được dự trữ để sử dụng bởi hệ điều hành. Sự giới hạn không gian địa chỉ nhằm bảo vệ những chương trình khỏi việc xử lý nhầm vào những vùng dữ liệu quan trọng, có thể gây hại tới hệ điều hành.

Tiến trình ở User space và Kernel space giao tiếp với nhau thông qua các hệ thống file ảo (virtual file system) và các lệnh gọi hệ thống (system call). Tầng nhân sẽ cung cấp các hàm gọi hệ thống như các API cho các ứng dụng ở user space sử dụng. Ngược lại, khi ứng dụng các hàm này, VFS sẽ đưa các lệnh đến các lớp tương ứng bên dưới.

**II. Quá trình thực hiện**

**1. Cài đặt môi trường làm việc**

Mở terminal và chạy lệnh bên dưới để cài đặt công cụ cần thiết và kernel headers:

Apt-get install build-essential linux-headers-‘uname -r’

**2. Tổ chức thiết kế**

**Ý tưởng:**

- Đầu tiên, tiến trình gọi system call open để mở device file tương ứng của char device này.

- Sau đó, tiến trình gọi system call read để đọc dữ liệu từ device file này. Khi đó, Linux kernel sẽ kích hoạt entry point read của char driver tương ứng.

- Cuối cùng, entry point read đọc dữ liệu từ char device ra rồi trả về cho tiến trình trên user space.

**Thiết kế hàm:** static ssize\_t Read(struct file \*filep, char \*buffer, size\_t size, loff\_t \*offset)

- Chức năng: đọc dữ liệu từ buffer của char device vào kernel buffer, sau đó sao chép dữ liệu từ kernel buffer vào user buffer của tiến trình thông qua hàm copy\_to\_user

- Tham số đầu vào:

+ \*filep: địa chỉ của cấu trúc file, mô tả một device file đang mở.

+ \*buff: địa chỉ của user buffer.

+ \*offset: địa chỉ của cấu trúc loff\_t, cấu trúc này cho biết vị trí trên buffer của character device mà dữ liệu bắt đầu được đọc ra.

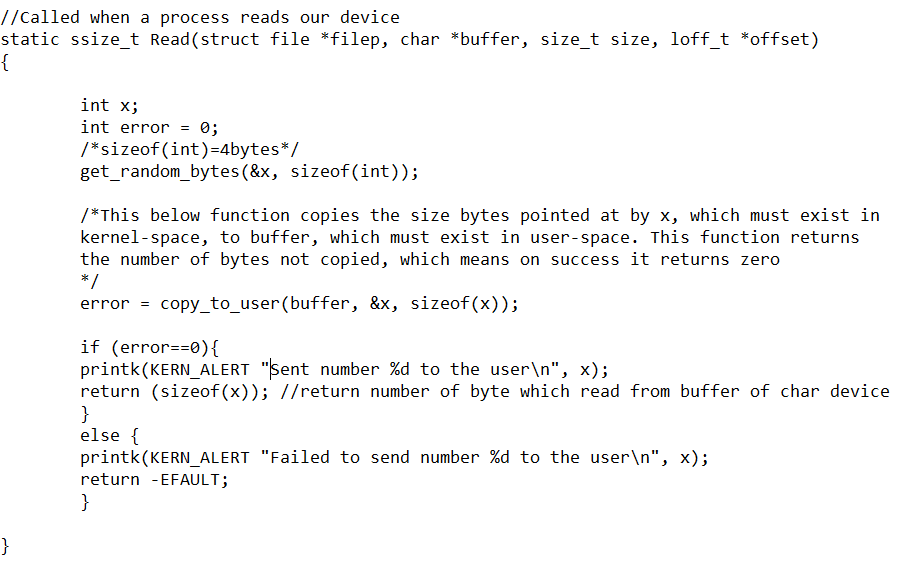
- Giá trị trả về:

+ Trả về một số 4 byte đã đọc được từ thiết bị

+ System call sẽ được gọi cho đến khi dữ liệu đọc được bằng tham số size hoặc cho tới khi toàn bộ dữ liệu trong buffer của character device được đọc hết.

+ Trả về 0 thể hiện rằng toàn bộ dữ liệu đã được đọc hết.

+ Trả về một số âm nếu có lỗi xảy ra.



Giải thích hàm trên:

- Hàm **get\_random\_bytes(&x, sizeof(int))** điền 4 byte số ngẫu nhiên bắt đầu tại địa chỉ &x.

- Sau đó dùng hàm **copy\_to\_user** để sao chép **size** byte dữ liệu từ kernel buffer vào user buffer. Kết quả trả về của hàm copy\_to\_user được lưu vào biến error.

- Nếu giá trị biến error = 0, thì sẽ trả về số byte đã được sao chép thành công, ngược lại, trả về dòng thông báo “Failed …”

**Giải thích**

Theo cơ chế quản lý bộ nhớ, kernel buffer luôn luôn nằm trên RAM, còn user buffer có thể tạm thời bị đưa ra ổ cứng bất cứ lúc nào, khi nào cần thì nó mới được đưa trở lại RAM. Nếu sao chép dữ liệu thẳng vào user buffer, nhưng user buffer lại đang không nằm trên RAM, khi đó, Linux kernel phải tìm cách đưa user buffer quay trở lại RAM. Việc này rất mất thời gian, khiến cho việc lấy dữ liệu từ character device không kịp thời. Chính vì vậy, ta phải sử dụng kernel buffer để chứa tạm dữ liệu đọc từ buffer của character device, sau đó sử dụng hàm **copy\_to\_user** để đưa dữ liệu từ kernel buffer lên user buffer

**III. Tài liệu tham khảo**

- <http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction>

- <https://blog.sourcerer.io/writing-a-simple-linux-kernel-module-d9dc3762c234>

- <https://vimentor.com/vi/lesson/linux-kernel-module>

- <https://www.quora.com/Linux-Kernel-How-does-copy_to_user-work>

- File hướng dẫn LẬP TRÌNH MODULE KERNEL LINUX.