**Trường Đại học Khoa học Tự Nhiên**

Khoa Công nghệ thông tin

\*\*\*\*\*\*\*\*\*\*



**BÁO CÁO ĐỒ ÁN**

MÔN HỌC: HỆ ĐIỀU HÀNH

TÊN ĐỒ ÁN: LINUX KERNEL MODULE

Nhóm:

18120168 - Nguyễn Đại Dương

18120183 – Trần Quang Huy

TP.HCM, ngày 14 tháng 12 năm 2020

MỤC LỤC

[1. Thông tin thành viên nhóm: 3](#_Toc58789664)

[2. Phân công công việc: 3](#_Toc58789665)

[3. Nội dung tìm hiểu: 3](#_Toc58789666)

[3.1. Linux Kernel Module: 3](#_Toc58789667)

[3.2. Hệ thống quản lý file và device trong linux: 5](#_Toc58789668)

[3.2.1. Hệ thống quản lý file: 5](#_Toc58789669)

[3.2.2. Hệ thống quản lý device trong linux 5](#_Toc58789670)

[3.3. Giao tiếp giữa tiến trình ở User space và Kernel space: 7](#_Toc58789671)

[4. Chương trình cài đặt: 9](#_Toc58789672)

[4.1. Cấu trúc: 9](#_Toc58789673)

[4.2. Cách chạy chương trình và kết quả minh hoạ: 10](#_Toc58789674)

[5. Đánh giá mức độ hoàn thành: 11](#_Toc58789675)

[Tài liệu tham khảo 11](#_Toc58789676)

# Thông tin thành viên nhóm:

|  |  |  |
| --- | --- | --- |
| MSSV | Sinh viên | Email |
| 18120168 | Nguyễn Đại Dương | daiduongnguyen168@gmail.com |
| 18120183 | Trần Quang Huy | huytrquang1103@gmail.com |

# Phân công công việc:

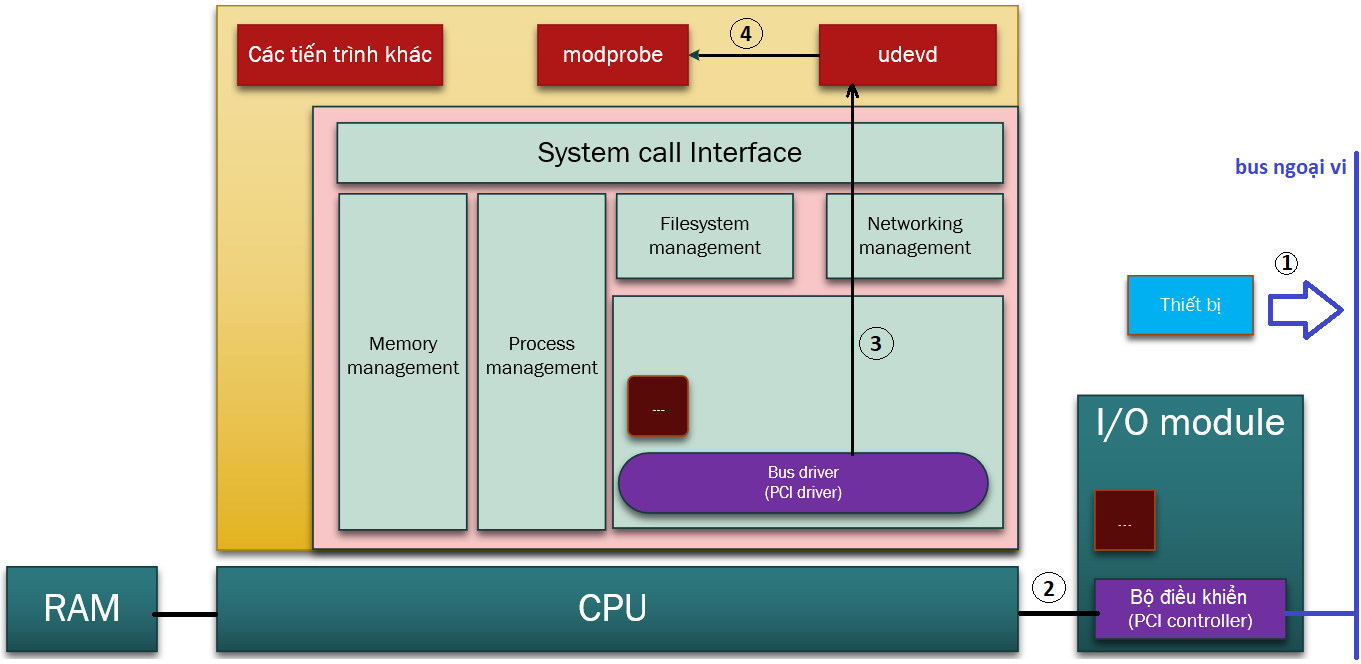
|  |  |  |
| --- | --- | --- |
| **Nội dung** | **Dương** | **Huy** |
| **Tìm hiểu lý thuyết** | X |  |
| **Viết module và readme cho module** |  | X |

# Nội dung tìm hiểu:

## Linux Kernel Module:

* Linux Kernel Module là 1 file với tên mở rộng là (.ko). Nó sẽ được lắp vào hoặc tháo ra khỏi kernel khi cần thiết. Do đó, nó còn có tên gọi là loadable kernel module (LKM). Một trong những kiểu loadable kernel module là driver.
* Lợi ích của việc thiết kế driver theo kiểu loadable module:
* Giúp giảm kích thước của 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.
* Loadable kernel module:
* Phần lớn các driver đều là các loadable kernel module, nhưng không phải là tất cả. Vẫn có một số driver được tích hợp luôn vào trong kernel, đặc biệt là các bus driver hay còn gọi là built-in driver. Các device driver thường sẽ là các loadable kernel module.
* Ngược lại, không phải loadable kernel module nào cũng là driver. Trên thực tế, loadable kernel module được chia làm 3 loại chính: device driver, system call và file system.
* Khi cần một module nhưng nó lại chưa có trong kernel space, kernel sẽ đưa module ấy vào (dưới sự giúp đỡ của tiến trình modprobe). Kernel kích hoạt tiến trình modprobe bằng kmod hoặc udevd.
* Vd:

Nếu cắm một thiết bị vào hệ thống máy tính, thì điện trở trên bus ngoại vi (ví dụ PCI bus hoặc USB bus) sẽ thay đổi và bộ điều khiển (controller) sẽ biết điều này. Khi đó, bus driver sẽ gửi một bản tin lên cho tiến trình udevd. Bản tin này chứa thông tin về thiết bị. Tiến trình udevd sẽ tra cứu file /lib/modules/<kernel-version>/modules.alias để tìm ra driver nào tương thích với thiết bị. Sau đó, udevd sinh ra tiến trình modprobe.



*Quá trình kích hoạt modprobe bằng udevd.*

* Kết luận:
* Driver có thể được tích hợp luôn vào trong kernel hoặc được thiết kế dưới dạng module tách rời. Driver cho các thiết bị cố định, ví dụ các thiết bị trong smartphone, sẽ được tích hợp luôn vào trong kernel. Driver cho các thiết bị hay phải thay đổi, sẽ được thiết kế dưới dạng loadable module.
* Kernel module có thể được đưa vào trong kernel một cách tự động, hoặc thủ công.
  + - Đối với trường hợp tự động, kernel kích hoạt tiến trình modprobe thông qua kmod hoặc udevd. Sau đó, modprobe sẽ đưa module cần thiết vào.
    - Đối với trường hợp thủ công, ta sẽ sử dụng lệnh insmod hoặc modprobe. Còn để đưa kernel module ra khỏi kernel space, ta sẽ sử dụng lệnh rmmod.

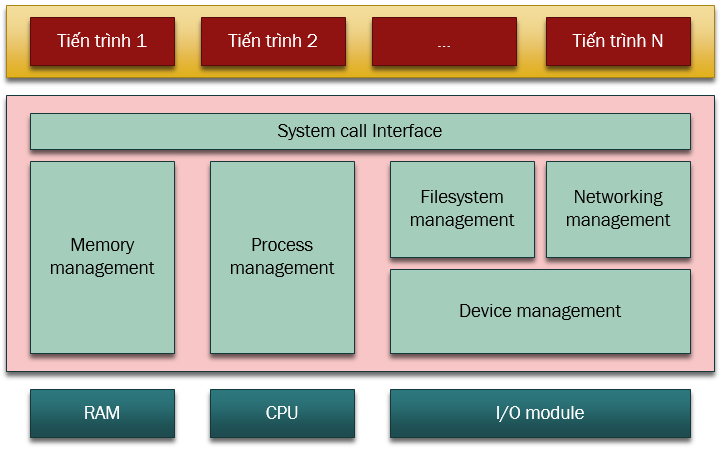
## Hệ thống quản lý file và device trong linux:

### Hệ thống quản lý file:

* Tất cả dữ liệu trong linux được tổ chức trong file. Tất cả các tập tin được tổ chức vào thư mục. Những thư mục này được tổ chức thành một cấu trúc dạng cây gọi là filesystem.
* Có 3 loại file cơ bản:
* Ordinary File: File thường, là một file trên hệ thống mà lưu trữ dữ liệu, văn bản hoặc chỉ dẫn chương trình.
* Directory: Thư mục, lưu giữ cả file thường và file đặc biệt.
* Special File: File đặc biệt, một số file đặc biệt cung cấp quyền truy cập vào phần cứng như các ổ cứng, CD-ROM, modem và các đầu đọc Ethernet. Một số file đặc biệt khác là tương tự như các phím tắt và khiến bạn có thể truy cập vào một file riêng nào đó bằng cách sử dụng các tên khác nhau.

### Hệ thống quản lý device trong linux

* Dựa vào chức năng của hệ điều hành, Linux Kernel được chia làm 6 thành phần:
* **Process management**: có nhiệm vụ quản lý các tiến trình.
* **Memory management**: có nhiệm vụ quản lý bộ nhớ.
* **Device management**: có nhiệm vụ quản lý thiết bị.
* **File system management**: có nhiệm vụ quản lý dữ liệu trên thiết bị lưu trữ (như ổ cứng, thẻ nhớ). Quản lý dữ liệu gồm các công việc: thêm, tìm kiếm, sửa, xóa dữ liệu.
* **Networking management**: có nhiệm vụ quản lý các gói tin (packet) theo mô hình TCP/IP.
* **System call Interface**: có nhiệm vụ cung cấp các dịch vụ sử dụng phần cứng cho các tiến trình. Mỗi dịch vụ được gọi là một **system call**.

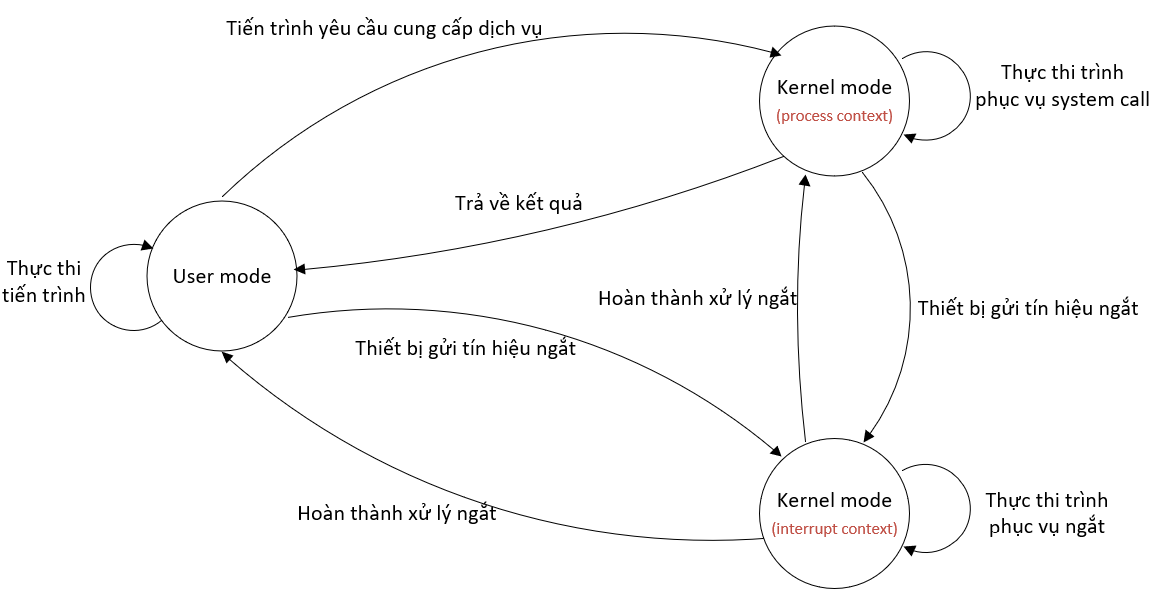


*Kiến trúc Linux Kernel ở góc độ quản lý (management point of views)*

* Trong 6 phần trên, phần device management gồm các driver, mỗi driver chịu trách nhiệm điều khiển, giám sát, trao đổi dữ liệu với một thiết bị
* Giới thiệu về linux driver:
* Driver là một phần mềm, gồm các lệnh, hướng dẫn CPU cách tương tác với thiết bị. Các thiết bị có thể là chuột, bàn phím, ổ cứng, card mạng, loa, màn hình,v.v Tuy nhiên, các thiết bị này không được nối trực tiếp với CPU, bởi vì:
* Hệ thống có nhiều thiết bị, nhưng số lượng nhân của CPU hữu hạn.
* Tốc độ làm việc của các thiết bị thấp hơn nhiều so với CPU.
* Các thiết bị kết nối với CPU thông qua một thiết bị khác, gọi là **bộ điều khiển** (**device controller**). Có nhiều cách để phân loại bộ điều khiển:
* Phân loại theo chức năng:
* Hard disk controller: trợ giúp CPU điều khiển ổ cứng.
* Graphic controller: trợ giúp CPU điều khiển các thiết bị hiển thị.
* Keyboard controller: trợ giúp CPU điều khiển bàn phím.
* Phân loại theo kỹ thuật giao tiếp với các thiết bị:
* PCI controller: hỗ trợ CPU giao tiếp với các thiết bị khác, mà các thiết bị này được thiết kế theo chuẩn PCI.
* USB controller: hỗ trợ CPU giao tiếp với các thiết bị khác, mà các thiết bị này được thiết kế theo chuẩn USB.
* I2C controller: hỗ trợ CPU giao tiếp với các thiết bị khác, mà các thiết bị này được thiết kế theo chuẩn I2C.
* Thiết bị (device) và bộ điều khiển (device controller) chỉ mang tính chất tương đối. Đứng từ góc độ của CPU, bộ điều khiển cũng chỉ là một thiết bị. Do đó, cần có driver hướng dẫn CPU làm việc với bộ điều khiển. Driver này được gọi là **bus driver**. Còn driver hướng dẫn CPU làm việc với thiết bị thì được gọi là **device driver**
* **Device drive** gồm 2 phần:
* Thành phần **OS specific**. Thành phần này cung cấp cho hệ điều hành các dịch vụ đọc/ghi dữ liệu của thiết bị. Điều này cho phép chúng ta xây dựng hệ điều hành độc lập với cấu trúc của thiết bị.
* Thành phần **device specific**. Thành phần này chứa các lệnh hướng dẫn CPU điều khiển thiết bị, giám sát thiết bị, trao đổi dữ liệu với thiết bị.
* Dựa vào lượng dữ liệu mỗi lần thiết bị trao đổi với CPU, thiết bị được chia làm 3 loại:
* **Character device**: lượng dữ liệu nhỏ nhất mà CPU và thiết bị trao đổi với nhau là 1 byte. Ví dụ về các thiết bị thuộc loại này là chuột, bàn phím, loa, v.v
* **Block device**: lượng dữ liệu nhỏ nhất mà CPU và thiết bị trao đổi với nhau là một khối, gồm nhiều byte (ví dụ 1 khối gồm 512 byte). Thông thường, block device là các thiết bị lưu trữ, như ổ cứng chẳng hạn.
* **Network device**: lượng dữ liệu nhỏ nhất mà CPU và thiết bị trao đổi với nhau là một gói tin, gồm nhiều byte. Gói tin có kích thước không cố định. Thông thường, network device là các thiết bị mạng, như NIC card, Wifi chip.

## Giao tiếp giữa tiến trình ở User space và Kernel space:

* Bộ nhớ RAM chứa các lệnh/dữ liệu dạng nhị phân của Linux kernel và các tiến trình. RAM được chia làm 2 miền
* **Kernel space** là vùng không gian chứa các lệnh và dữ liệu của kernel.
* **User space** là vùng không gian chứa các lệnh và dữ liệu của các tiến trình.
* CPU có 2 chế độ thực thi:
* Khi CPU thực thi các lệnh của kernel, thì nó hoạt động ở chế độ **kernel mode**. Khi ở chế độ này, CPU sẽ thực hiện bất cứ lệnh nào trong tập lệnh của nó, và CPU có thể truy cập bất cứ địa chỉ nào trong không gian địa chỉ.
* Khi CPU thực thi các lệnh của tiến trình, thì nó hoạt động ở chế độ **user mode**. Khi ở chế độ này, CPU chỉ thực hiện một phần tập lệnh của nó, và CPU cũng chỉ được phép truy cập một phần không gian địa chỉ.
* Có 2 cách để giao tiếp giữa User space và Kernel space:
* **System call**: Khi một tiến trình cần sử dụng một dịch vụ nào đó của kernel, tiến trình sẽ gọi một **system call**. Các system call được cung cấp bởi kernel. Do đó khi tiến trình gọi các system call, CPU phải chuyển sang chế độ kernel mode để thực thi các lệnh của kernel. Lúc này, ta nói rằng, CPU đang thực thi ở chế độ kernel mode, trong ngữ cảnh **process context**. Sau khi kernel thực hiện xong yêu cầu, kernel gửi trả kết quả cho tiến trình. Lúc này, CPU lại chuyển sang chế độ user mode để thực thi tiếp các lệnh của tiến trình.
* **Interrupt:** ngắt cũng là một nguyên nhân khiến CPU chuyển chế độ thực thi sang kernel mode. Khi có một thiết bị muốn trao đổi dữ liệu với CPU, nó sẽ gửi một tín hiệu ngắt tới CPU bằng cách nâng điện áp trên chân INT của CPU. Khi đó, CPU sẽ ngừng thực thi các lệnh của tiến trình lại, chuyển sang chế độ kernel mode rồi thực thi một chương trình đặc biệt của kernel để xử lý tín hiệu ngắt đó. Lúc này, ta nói CPU đang thực thi ở chế độ kernel mode, trong ngữ cảnh **interrupt context**. Sau khi xử lý xong, CPU trở lại chế độ user mode và tiếp tục thực hiện các lệnh tiếp theo của tiến trình.



*Các chế độ hoạt động của CPU.*

# Chương trình cài đặt:

* 1. *Cấu trúc:*

File mã nguồn gồm các hàm:

* dev\_open
* dev\_release
* dev\_read
* rnm\_init
* rnm\_exit

1. *Hàm dev\_open và dev\_release:* chỉ đơn giản là hàm ghi lại log khi mở và đóng thiết bị
2. *Hàm dev\_read:*

- Đây là hàm thực hiện việc tạo số ngẫu nhiên bằng cách dùng hàm get\_random\_bytes()

- Sau đó trả số đó về cho user bằng cách gọi hàm API copy\_to\_user()

1. *Hàm rnm\_init:* Đây là hàm chạy đầu tiên khi lắp module vào kernel, thực hiện các bước sau:

- Đăng ký thông tin số hiệu <major, minor>

- Tạo file thiết bị tự động bằng cách thực hiện: tạo lớp thiết bị, sau đó tạo thông tin <major, minor> cho thiết bị

- Đăng ký các thao tác file với VFS (virtual file system)

1. *Hàm rnm\_exit:* Đây là hàm chạy ngay trước khi module được tháo khỏi kernel, thực hiện các bước sau (theo thứ tự ngược so với rnm\_init):

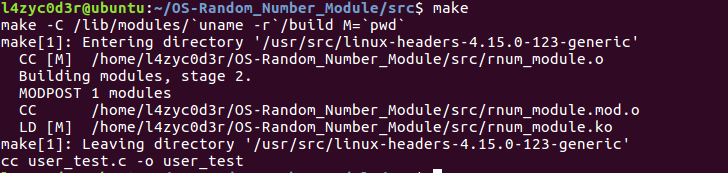
- Huỷ đăng ký các thao tác file với VFS

- Huỷ file thiết bị bằng cách thực hiện: huỷ thông tin <major, minor> cho thiết bị, sau đó huỷ lớp thiết bị

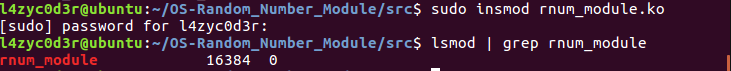
- Huỷ đăng ký thông tin số hiệu <major, minor>

* 1. *Cách chạy chương trình và kết quả minh hoạ:*
* Cách chạy chương trình đã được mô tả trong file README.md
* Kết quả minh hoạ:

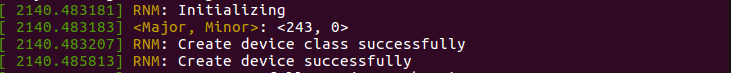
- Biên dịch chương trình:



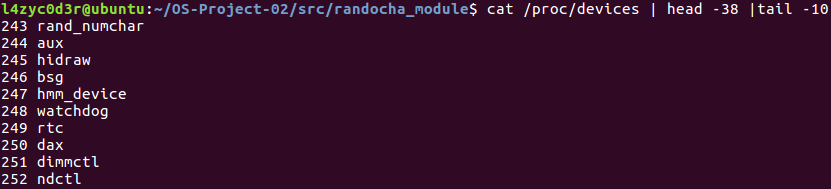
- Lắp module và kiểm tra kết quả



- Xem log sau khi lắp module vào kernel:



- Xem thông tin số hiệu major: 243 rand\_numchar



- Kết quả khi test module :



# Đánh giá mức độ hoàn thành:

|  |  |  |
| --- | --- | --- |
| Yêu cầu | | Mức độ hoàn thành |
| Tìm hiểu | Linux kernel module | 100% |
| Hệ thống quản lý file và device trong linux | 100% |
| Giao tiếp giữa tiến trình ở user space và code kernel space | 100% |
| Lập trình | Viết một module dùng để tạo ra số ngẫu nhiên. | 100% |
| Module này sẽ tạo một character device để cho phép các tiến trình ở userspace có thể open và read các số ngẫu nhiên. | 100% |

# Tài liệu tham khảo

1. Tài liệu đã được cung cấp
2. <https://www.tutorialspoint.com/unix/unix-file-management.htm>
3. <https://vimentor.com/vi/lesson/gioi-thieu-ve-linux-kernel-1>
4. <https://stackoverflow.com/questions/20182691/linux-get-random-bytes-error-implicit-declaration-of-function>
5. [Github repository của @84436](https://github.com/84436/OS-Project-02)
6. [Github repository của @npkhang99](https://github.com/npkhang99/CSC10007-Project2)