****

**BÁO CÁO**

***ĐỒ ÁN THỰC HÀNH 2***

***LINUX KERNEL MODULE***

Ung Tiến Đạt – 18120314

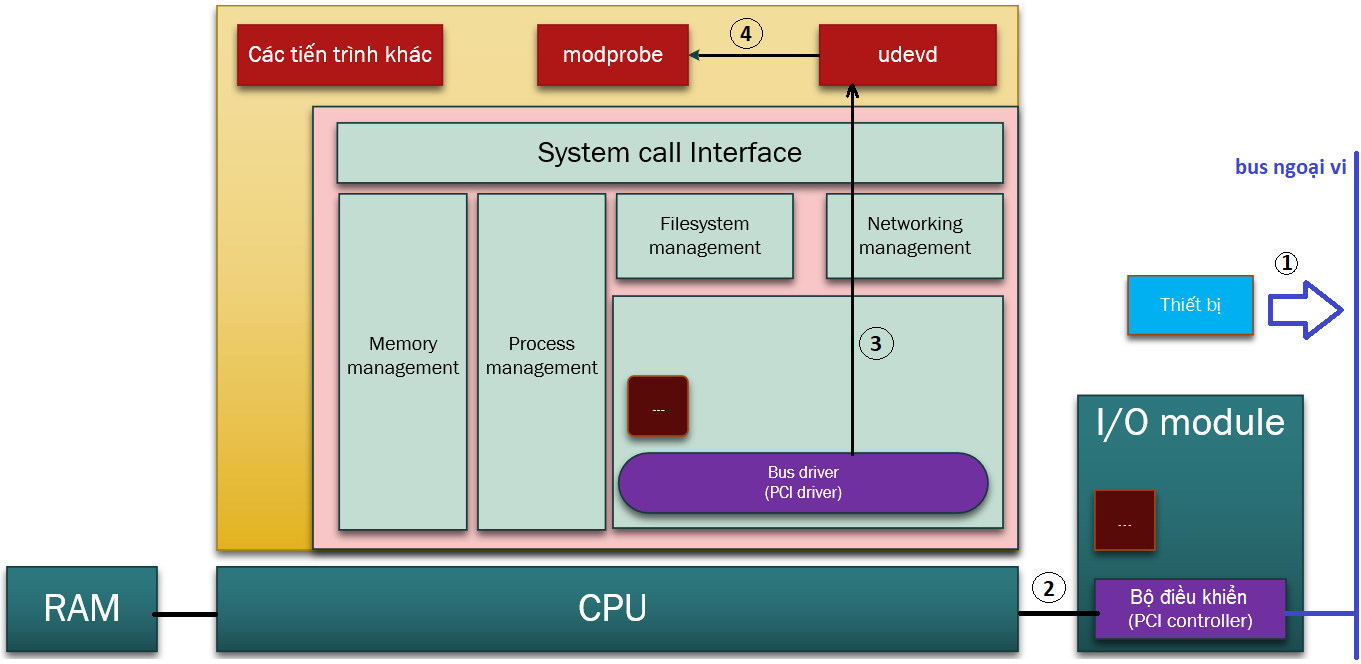
Võ Ngọc Đức – 18120327

1. **Thành viên:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **HỌ & TÊN** | **MSSV** | **PHÂN CÔNG** | **Mức độ hoàn thành đồ án.** |
| 1 | Ung Tiến Đạt | 18120314 | * Tìm hiểu về Linux Kernel Module. * Tìm hiểu về Driver trong Linux. * Viết Driver để ẩn file trên Linux. | 100% |
| 2 | Võ Ngọc Đức | 18120327 | * Tìm hiểu cách nạp và hủy nạp 1 Module. * Tìm hiểu cách sử dụng và hủy Driver để ẩn file. * Viết Driver để ẩn file trên Linux. | 100% |

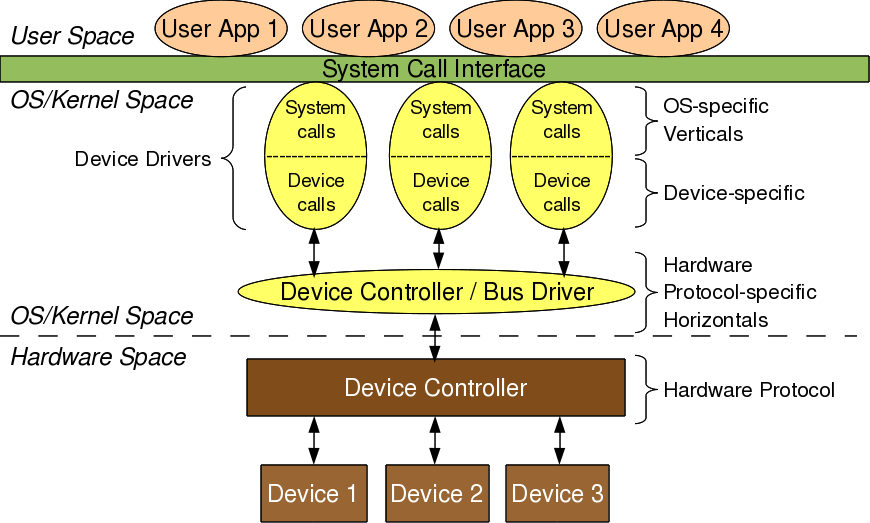
1. **Nội dung đồ án**
   1. **Linux Kernel Module:**
      1. **Linux Kernel là gì?**
   2. **Các thành thần chính:**

* **Process management:** có nhiệm vụ quản lý các tiến trình, bao gồm các công việc:
  + Tạo/hủy các tiến trình.
  + Lập lịch cho các tiến trình.
  + Hỗ trợ các tiến trình giao tiếp với nhau.
  + Đồng bộ hoạt động của các tiến trình để tránh xảy ra tranh chấp tài nguyên.
* **Memory management:** có nhiệm vụ quản lý bộ nhớ, bao gồm các công việc:
  + Cấp phát bộ nhớ trước khi đưa chương trình vào, thu hồi bộ nhớ khi tiến trình kết thúc.
  + Đảm bảo chương trình nào cũng có cơ hội được đưa vào bộ nhớ.
  + Bảo vệ vùng nhớ của mỗi tiến trình.
* **Device management:** có nhiệm vụ quản lý thiết bị, bao gồm các công việc:
  + Điều khiển hoạt động của các thiết bị.
  + Giám sát trạng thái của các thiết bị.
  + Trao đổi dữ liệu với các thiết bị.
  + Lập lịch sử dụng các thiết bị, đặc biệt là thiết bị lưu trữ (ví dụ ổ cứng).
* **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.
  1. **Các khái niệm thường gặp:**
* Bộ nhớ RAM chứa các lệnh/dữ liệu dạng nhị phân của Linux kernel và các tiến trình, 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.
    1. **Module trong Kernel:**
  1. **Khái niệm:**
* **Linux Kernel Module** là một file với tên mở rộng là “**.ko**” và 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**. 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.
* 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**. Chúng được 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, ví dụ **kvm.ko** là Loadable Kernel Module nhưng không phải là Driver. Trên thực tế, Loadable Kernel Module được chia làm 3 loại chính: **device driver**, **system call** và **file system**.
* Để đơn giản hóa, các thuật ngữ Device Driver, Loadable Kernel Module, Linux Kernel Module, Loadable Module, Kernel Module trong báo cáo đều được hiểu là **module**.
  1. **Các thành phần:**
* Dù viết bất cứ Kernel Module nào, ta cũng cần phải tham chiếu tới file của Linux Kernel là **<linux/module.h>**, chứa 2 macro quan trọng:
  + **module\_init()** giúp xác định hàm nào sẽ được thực thi ngay sau khi lắp module vào kernel.
  + **module\_exit()** giúp xác định hàm nào được thực thi ngay trước khi tháo module ra khỏi kernel.
* Macro **\_\_init** thường đi kèm với hàm khởi tạo, xuất hiện trước tên một hàm. Macro này giúp Kernel biết rằng, hàm trên chỉ phải thực thi lúc khởi tạo, nên vùng nhớ chứa hàm này có thể được giải phóng sau khi nó thực thi xong mà không ảnh hưởng gì.
* Tương tự, macro **\_\_exit** thường đi kèm với hàm kết thúc, xuất hiện trước tên một hàm. Macro này cho kernel biết, khi lắp module vào kernel thì chưa cần đưa hàm trên vào trong bộ nhớ RAM. Chỉ khi chuẩn bị tháo module ra khỏi kernel, hàm này mới cần được đưa vào RAM và thực thi.
* Ngoài ra, còn một số macro khác cung cấp thông tin về module như:
  + **MODULE\_AUTHOR()** cho biết ai là người tạo ra module.
  + **MODULE\_DESCRIPTION()** cho biết module làm được những gì.
  + **MODULE\_SUPPORTED\_DEVICE()** cho biết module này hỗ trợ làm việc với những thiết bị nào.
  + **MODULE\_LICENSE()** cho biết người dùng có cần phải trả phí nếu sử dụng module hay không. Với giấy phép sử dụng GPL, người dùng có thể sử dụng module miễn phí. Ngoài GPL, còn có các loại license như GPL v2, BSD/GPL, MIT/GPL, MPL/GPL.
  1. **Cách nạp và gỡ module:**
* Khi cần một module nhưng nó lại chưa có trong kernel space, kernel sẽ đưa module ấy vào. Quá trình này có thể diễn ra một cách tự động, với trình tự sau:
  + Bước 1: Kernel kích hoạt tiến trình **modprobe** cùng với tham số truyền vào là tên của module (abc.ko).
  + Bước 2: Tiến trình **modprobe** kiểm tra file **/lib/modules/<kernel-version>/modules.dep** xem abc.ko có phụ thuộc vào module nào khác không. Giả sử abc.ko phụ thuộc vào module xyz.ko.
  + Bước 3: Tiến trình **modprobe** sẽ kích hoạt tiến trình **insmod** để đưa các module phụ thuộc vào trước (xyz.ko), rồi mới tới module cần thiết (abc.ko).
* Kernel sẽ kích hoạt tiến trình **modprobe** bằng 2 cách:
  + Cách 1: sử dụng **kmod**. Đây là một thành phần của Linux Kernel, hoạt động trong Kernel Space. Khi một thành phần nào đó của Kernel cần đưa một module vào trong Kernel Space, nó sẽ truyền tên module cho hàm **request\_module** của **kmod**. Hàm request\_module sẽ gọi hàm **call\_usermodehelper\_setup** để sinh ra tiến trình **modprobe**.
  + Cách 2: sử dụng **udevd** (hình 1). Đây là một tiến trình hoạt động trong User Space. Nếu một thiết bị cắm 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**.



(Hình 1)

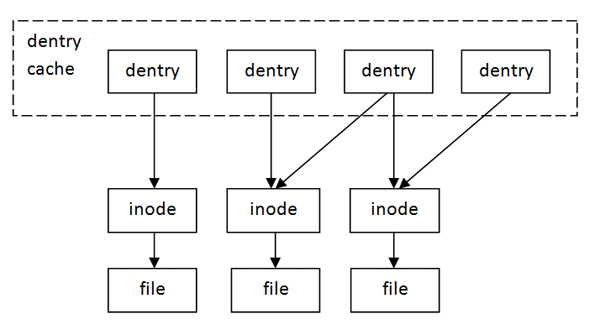
* Để tạo ra module, ta dùng lệnh **make** hoặc **make** **all**. Khi đó tiến trình **make** sẽ dựa vào **Makefile** để biên dịch mã nguồn, tạo ra module. Sau khi biên dịch xong, ta sẽ thấy xuất hiện môt file có tên mở rộng là “.ko” (ko là viết tắt của kernel object). Đây chính là kernel module.
* Để lắp module vào trong Kernel, ta có thể thực hiện thủ công bằng cách gõ lệnh **insmod**. Sau khi lắp xong, ta sẽ dùng lệnh **ismod** để kiểm tra xem module đã được load thành công chưa. Ngoài ra, ta có thể dùng lệnh **dmesg** để theo dõi quá trình hoạt động của module. Cuối cùng, chúng ta sẽ dùng lệnh **rmmod** để tháo module ra khỏi Kernel.
  + 1. **Driver trong Linux:**
  1. **Khái niệm:**
* **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, …).
* Một thành phần phần cứng có thể được điều khiển bởi một **driver** hoặc được điều khiển bởi một phần cứng khác mà được quản lý bởi một **driver**. Trường hợp này, phần cứng có vai trò điều khiển được gọi là một **device controller**. Bản thân các **controller** cũng cần **driver**.
  1. **Nhiệm vụ:**
* Các **bus driver** cung cấp giao diện đặc tả cho các giao thức phần cứng tương ứng. Nó nằm ở tầng dưới cùng trong mô hình phân lớp phần mềm của hệ điều hành.
* Nằm trên nó là các **device driver** thực sự để vận hành các thiết bị, mang đặc trưng của từng thiết bị xác định. Ngoài ra, mục đích quan trọng của các driver thiết bị là cung cấp một giao diện lên tầng trên của hệ điều hành.



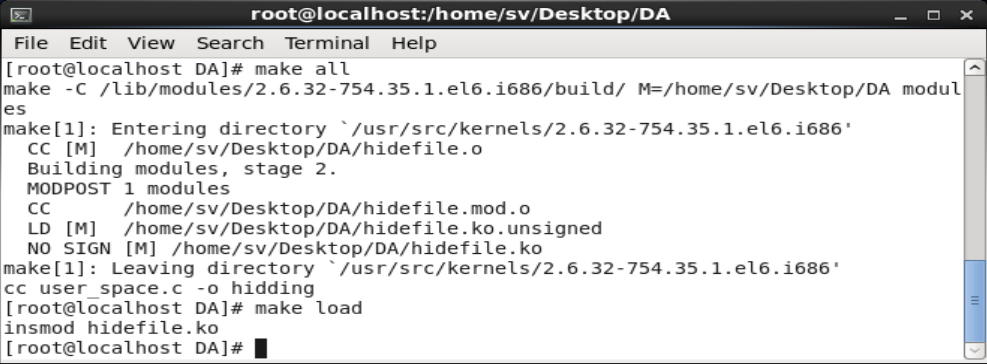
* Một cách tổng quan, một driver sẽ bao gồm 2 phần quan trọng:
  + Thành phần giao tiếp với thiết bị (**device-specific**) của một driver là giống nhau đối với tất cả các hệ điều hành. Nó có thể hiểu và giải mã các thông tin về thiết bị (chi tiết kỹ thuật, kiểu thao tác, hiệu năng, cách lập trình giao tiếp với thiết bị, …).
  + Thành phần giao tiếp với hệ điều hành (**OS-specific**) gắn kết chặt chẽ với các cơ chế của hệ điều hành, và do vậy sẽ là khác nhau giữa một driver trên Linux và một driver trên Windows, hoặc MacOS, …

1. **Cách thức dùng driver để ẩn file, hủy driver thì show lại file:**

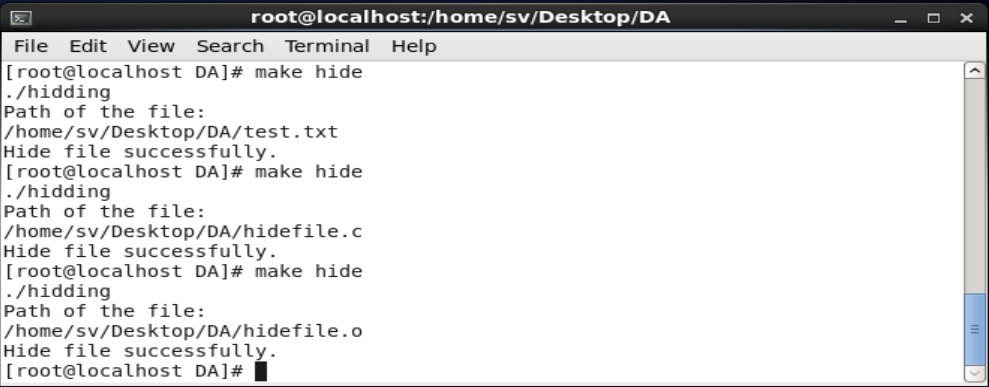
* Virtual File System (VFS) là tầng phần mềm trong kernel giúp cho người dùng có thể tương tác với hệ thống tập tin. Khi mà truyền tham số filePath vào thì VFS sẽ dùng nó để tìm thông qua directory entry cache (dentry cache). Như vậy ta có thể chuyển đường dẫn tệp thành một dentry cụ thể, phục vụ những tác vụ tiếp theo.
* Một dentry thường có một con trỏ trỏ đến 1 inode. Inode là một cấu trúc dữ liệu chứa các metadata của mỗi file, thư mục. Mỗi 1 inode được đánh số riêng biệt.

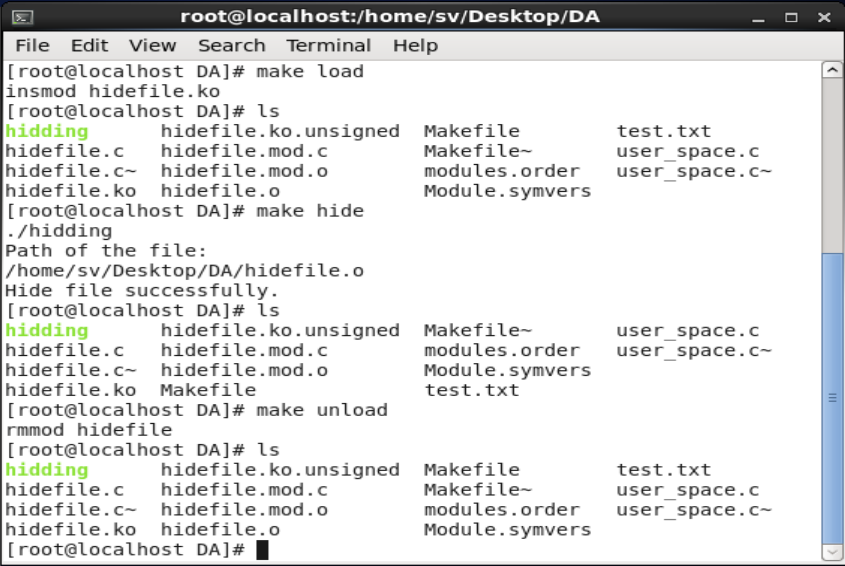


* Tận dụng cách thức này, driver ẩn file sẽ thực hiện thay đổi con trỏ trên cấu trúc inode và file operations của 1 file thành hàm do ta định nghĩa lại. (thay đổi với file cần ẩn và thư mục cha của nó)
* Để ẩn 1 file, ta hook hàm filldir() cho thư mục cha. Hàm này được gọi từ hàm readdir(). Để thay đổi lời gọi hàm filldir(), ta định nghĩa lại hàm readdir(), và gọi filldir() từ hàm này. (hàm parent\_readdir trong hidefile.c sẽ ghi đè hàm readdir())
* Tại hàm new\_filldir() thì với các file cần ẩn thì trả về 0, nếu không thì ta gọi đến hàm filldir() gốc.
* Việc thay đổi hàm filldir chỉ giúp ẩn tệp khỏi thư mục. Nhưng với các lệnh system call thì vẫn hoạt động. Nên ta thay đổi các inode và file operations cho file. Bằng cách ta trả về giá trị -2 cho tất cả các thao tác vào file bị ẩn.
* Sử dụng hàm hook\_funtion để hook inode, file operations và lấy inode number của file chỉ định thông qua hàm path\_lookup. Tại đây cấu trúc nameidata sẽ được dùng để chứa dentry của đường dẫn chỉ định. Sau đó ta cấp phát, lưu lại các dữ liệu về con trỏ cũ, inode, inode number bằng hàm reallocate\_memmory(). Cuối cùng ta thay đổi dữ liệu thành cấu trúc mà ta đã tạo mới.
* Khi unload driver để có thể hiển thị lại file đã bị ẩn, thì trong hàm hidefile\_exit. Ta gọi hàm backup\_funtion.
* Backup\_funtion có chức năng khôi phục lại con trỏ trỏ đến các inode cũ và file operations cũ.
  1. **Cài đặt Module:**
     1. **Biên dịch và cài đặt:**

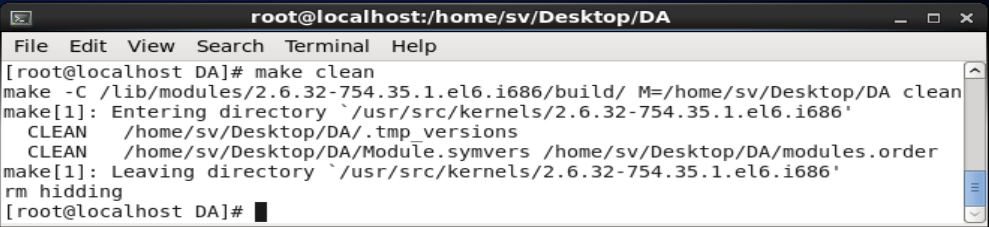


* + 1. **Chạy chương trình kiểm tra:**





* + - Ẩn được file hidefile.o thành công. Sau đó khi unload driver thì file sẽ hiện lại.
    1. **Gỡ Module khỏi Kernel:**



* 1. **Tài liệu tham khảo:**

[1] <https://www.apriorit.com/dev-blog/195-simple-driver-for-linux-os>

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

[3] https://www.codeproject.com/Articles/444995/Driver-to-hide-files-in-Linux-OS