**1) Giới thiệu Linux Kernel**

\_ Dựa theo chức năng của hệ điều hành, Linux kernel được chia làm 6 thành phần:

+ Process management: quản lý các tiến trình:

* Tạo/ hủy
* Lập lịch
* Hỗ trợ giao tiếp giữa các tiến trình
* Đồng bộ hoạt động giữa các tiến trình

+ Memory management: quản lý bộ nhớ:

* Cấp phát/ thu hồi bộ nhớ
* Đảm bảo tiến trình nào cũng có cơ hội được đưa vào bộ nhớ
* Bảo vệ vùng nhớ của mỗi tiến trình

+ Device management: quản lý thiết bị:

* Điều khiển
* Giám sát
* Trao đổi dữ liệu giữa các thiết bị
* Lập lịch sử dụng các thiết bị, đặc biệt là thiết bị lưu trữ

+ File system management: quản lý dữ liệu trên thiết bị lưu trữ:

* Thêm
* Tìm kiếm
* Sửa
* Xóa

+ Networking management: quản lý các gói tin (packet) theo mô hình TCP/IP

+ System call Interface: cung cấp các dịch vụ sử dụng phần cứng cho các tiến trình. Mỗi dịch vụ được gọi là system call

\_ Các khái niệm:

+ User space: vùng không gian chứa các lệnh và dữ liệu của tiến trình

+ Kernel space: vùng không gian chứa các lệnh và dữ liệu của kernel

+ User mode: thực thi các lệnh của tiến trình và chuyển sang kernel mode dưới 1 số lệnh đặc biệt

+ Kernel mode: dưới 1 số lệnh như system call, input/output, các loại interrupt,… user mode sẽ được chuyển qua kernel mode -> thực thi các lệnh cần thiết -> chuyển về user mode

+ Process context: ngữ cảnh khi thực thi ở chế độ kernel mode khi được gọi system call

+ Interrupt context: ngữ cảnh khi thực thi ở chế độ kernel mode khi được gọi interrupt

**2) Giới thiệu Linux Kernel Module (Loadable Kernel Module)**

\_ 1 file với tên mở rộng là (.ko), có thể tháo ra hoặc lắp vào khi cần thiết

\_ Dạng phổ biến là driver, việc thiết kế driver theo kiểu loadable kernel mang lại 3 lợi ích:

+ Giảm kích thước kernel -> giảm lãng phí bộ nhớ và thời gian khởi động hệ thống

+ Không cần biên dịch lại kernel khi thêm driver mới hoặc thi thay đổi driver

+ Không cần phải khởi động lại hệ thống khi thêm mới driver

\_ Bus driver (built-in driver) được tích hợp luôn vào trong kernel

\_ Loadable kernel được chia làm 3 loại chính:

+ Device driver

+ System call

+ File system

\_ Module được cài đặt trong kernel space 1 cách tự động theo trình tự sau:

+ Kernel kích hoạt tiến trình modprobe cùng với tham số truyền vào là tên module

+ Tiến trình modprobe kiểm tra file /lib/modules/<kernel-version>/modules.dep xem có phụ thuộc vào module khác hay không

+ Tiến trình modprobe kích hoạt insmod để đưa các module phụ thuộc vào trước, rồi mới tới module cần thiết

\_ Phương thức kích hoạt modprobe:

+ Cách 1 (kmod): khi 1 thành phần nào đó của kernel cần đưa 1 module vào trong kernel space -> truyền tên module cho hàm request\_module -> hàm request\_module sẽ gọi hàm call\_usermodehelper\_setup -> sinh ra tiến trình modprobe

+ Cách 2 (udevd): tiến trình hoạt động trên user space: 1 thiết bị cắm vào hệ thống máy tính -> điện trở trên bus ngoại vi thay đổi -> bộ điều khiển nhận biết -> bus driver gửi 1 bản tin cho udevd (bảng tin chứa thông tin về thiết bị) -> udevd sẽ tra cứu file /lib/modules/<kernel-version>/modules.alias để tìm ra driver tương thích -> sinh ra tiến trình modprobe

**3) Cách viết 1 Linux Kernel Module**

\_ Cần #include các file cần dung trong Linux kernel để sử dụng các hàm và macro

\_ <linux/module.h> với 2 macro quan trọng và bắt buộc phải có:

+ module\_init(): thường đi kèm với hàm khởi tạo - thực thi ngay sau khi lắp module vào kernel

+ module\_exit(): thường đi kèm với hàm kết thúc - thực thin gay trước khi tháo module khỏi kernel

\_ Sử dụng printk() để ghi lại quá trình hoạt động của module (gọi là logging)

\_ Sử dụng lệnh dmesg để xem quá trình hoạt động của kernel từ lúc nó hoạt động

\_ Lệnh modinfo để xem các thông tin của 1 module:

+ Macro MODULE\_AUTHOR: tác giả của module

+ Macro MODULE\_DESCRIPTION: module làm được những gì

+ Macro MODULE\_SUPPORTED\_DEVICE: module hỗ trợ làm việc với những thiết bị nào

+ MODULE\_LICENSE: có cần phải trả phí khi sử dụng module hay không ?

\_ Để biên dịch kernel ta cần dùng phương pháp Kbuild: tạo ra 2 file Makefile và Kbuild

\_ Để tạo kernel module gõ lệnh make hoặc make all -> dựa vào 2 file Makefile và Kbuild để biên dịch mã nguồn -> tạo ra kernel module

\_ Tháo/lắp module:

+ Lắp module: lệnh insmod

+ Kiểm tra module load thành công: lệnh Ismod

+ Kiểm tra hoạt động module: lệnh dmesg

+ Tháo module: lệnh rmmod

**4) Cơ bản về driver trong Linux**

\_ Vai trò driver: 1 chương trình điều khiển có vai trò điều khiển, quản lý, giám sát 1 thực thể nào đó dưới quyền của nó. Các controller cũng cần có driver

\_ 2 nhiệm vụ của driver:

+ Giao tiếp với thiết bị: hiểu và giải mã các thông tin về thiết bị

+ Giao tiếp với hệ điều hành: driver trên cách hệ điều hành khác nhau thì sẽ khác nhau

\_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.

\_ Driver trên Linux được phân chia thảnh 3 loại sau:

+ Packet-oriented or the network vertical:

* Network protocol stack
* Network interface card device drivers

+ Block-oriented or the storage vertical:

* File-system drivers: giải mã các định dạng khác nhau trên các vùng lưu trữ khác nhau
* Block device drivers: cho các giao thức phần cứng ứng với các thiết bị lưu trữ khác nhau

+ Byte-oriented or the character vertical: được chia thành các lớp con như tty driver, input driver, …

**5) Character Device Driver**

\_ Hầu hết các thiết bị thuộc kiểu thiết bị hướng byte (byte-oriented) => hầu hết các device driver là các character device driver

\_ Ứng dụng tầng người dùng (user space) muốn thao tác với thiết bị kiểu character device trong tầng phần cứng(hardware space) sẽ sử dụng character device driver tương ứng trong tầng nhân (kernel space) thông qua hệ thống file ảo(virtual file system – VFS)

\_ Việc truy cập các file thiết bị sử dụng các thao tác file thông thường như đối với file dữ liệu nhưng hiệu quả khác nhau

\_ Việc kết nối từ ứng dụng đến thiết bị được thực hiện hoàn chỉnh thông qua 4 thực thể chính liên quan gồm:

+ Application (Ứng dụng)

+ Character device file (File thiết bị)

+ Character device driver (Driver thiết bị)

+ Character device (Thiết bị)

\_ Số hiệu file thiết bị:

+ Việc kết nối giữa ứng dụng và file thiết bị được thực hiện thông qua số hiệu file thiết bị

+ Để liệt kê file thiết bị: ls -l /dev/ | grep “^c”

\_ Các hỗ trợ liên quan tới <major, minor> trong kernel 2.6

+ Đăng ký số hiệu <major, minor> cho file thiết bị

+ Kết nối các thao tác file thiết bị với các hàm tương ứng trong driver

\_ Tạo file thiết bị tự động: Lớp thiết bị (device class) được đưa ra bởi phần nhân -> thông tin (<major, minor>) được tạo ra bởi hàm API -> Các công việc còn lại được thực hiện bởi udev

\_ Các thao tác file với thiết bị: Để truyền các thao tác file thiết bị vào driver phải đăng ký thao tác file với VFS được thực hiện trong mã nguồn driver gồm 2 bước:

+ Tạo ra biến cấu trúc “struct file\_operations pugs\_fops” và điền các thao tác xử lý muốn dùng với file

+ Điều khiển cấu trúc này tới hệ thống file ảo VFS bằng cách gọi hàm “cdev\_add()”

**6) Các thao tác với file device**

\_ Thao tác đọc file device: gọi hàm đọc (system call) -> VFS -> tìm tới driver cần thiết dựa vào cặp số hiệu <major, minor> -> gọi hàm my\_read() của driver -> hàm trả về số lượng byte dữ liệu nhận được

\_ Thao tác ghi file device: cung cấp kích thước byte dữ liệu -> hàm my\_write() nhận dữ liệu -> ghi vào thiết bị thật -> hàm trả về số byte ghi thành công