BAN CƠ YẾU CHÍNH PHỦ

HỌC VIỆN KỸ THUẬT MẬT MÃ

LẬP TRÌNH DRIVER

Ngành: Công nghệ thông tin

Chuyên ngành: Kỹ thuật phần mềm nhúng và di động

Mã số: 52.48.02.01

Nội dung

- Tóm lược về HĐH
- Khái niệm driver
- Hoạt động
- Phân lớp driver
- Vấn đề an toàn
- Phiên bản và giấy phép

Tóm lược về HĐH

- Hệ điều hành
  - Nghĩa hẹp, hệ điều hành là một phần mềm quản lý tài nguyên phần cứng. Phần mềm này còn được gọi là **nhân** (kernel).
    - ✓Ví dụ Linux kernel.
  - Nghĩa rộng, hệ điều hành là một gói phần mềm gồm:
    - ✓Phần mềm quản lý các tài nguyên phần cứng.
    - ✓Các phần mềm quan trọng khác như phần mềm giao diện dòng lệnh (CLI – Command Line Interpreter), phần mềm giao diện đồ họa (GUI – Graphic User Interface)

Nhân HĐH


Các tiến trình
Cung cấp dịch vụ sử dụng phần cứng
Hệ điều hành
Quản lý phần cứng
Phần cứng

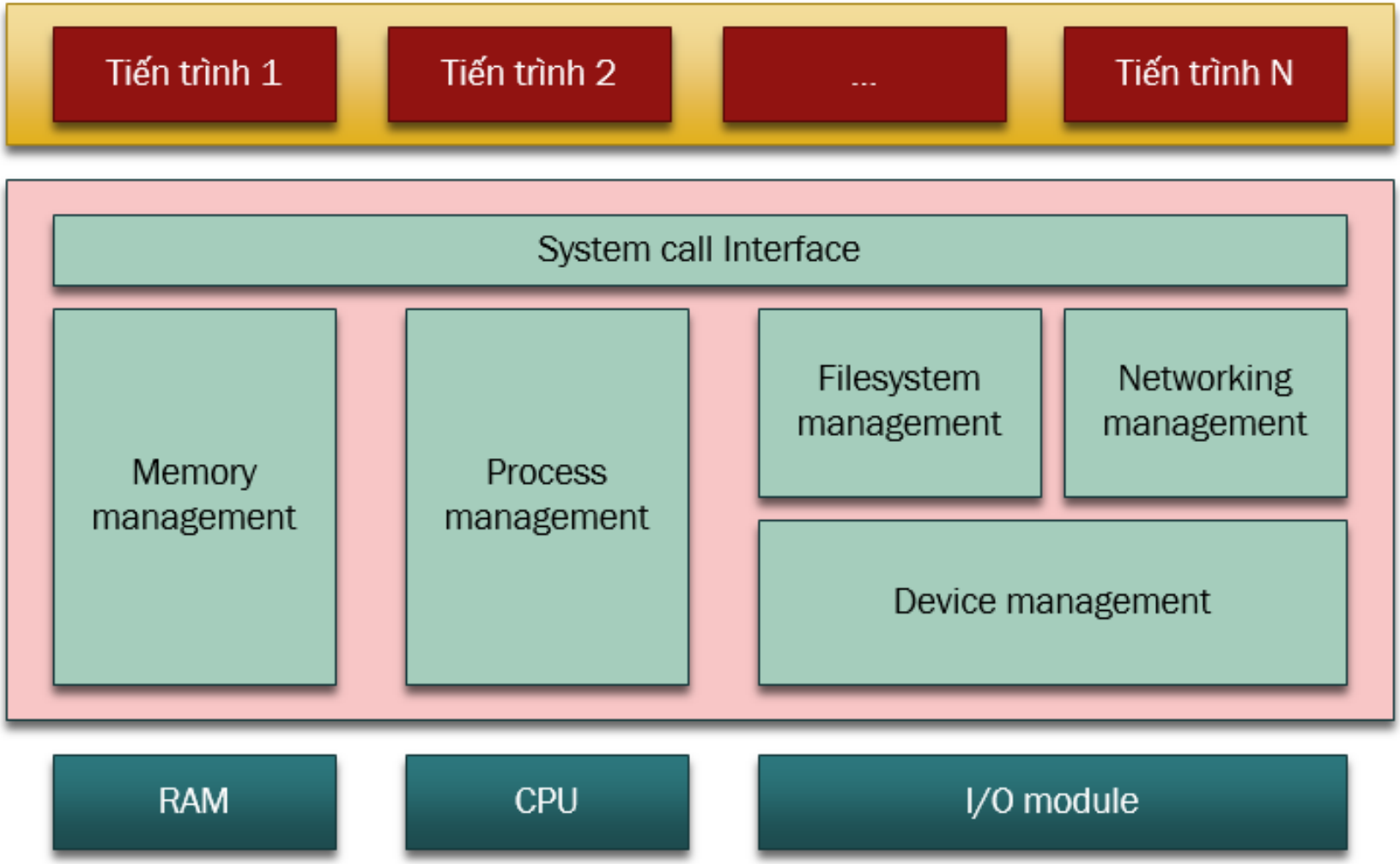
Nhân HĐH

- **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. Đây thực chất là lên kế hoạch: CPU sẽ thực thi chương trình khi nào, thực thi trong bao lâu, tiếp theo là chương trình nào.
  - 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.

Nhân HĐH

- **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**

Nhân HĐH


Tiến trình 1
Tiến trình 2
...
Tiến trình N
System call Interface
Memory management
Process management
Filesystem management
Networking management
Device management
RAM
CPU
I/O module

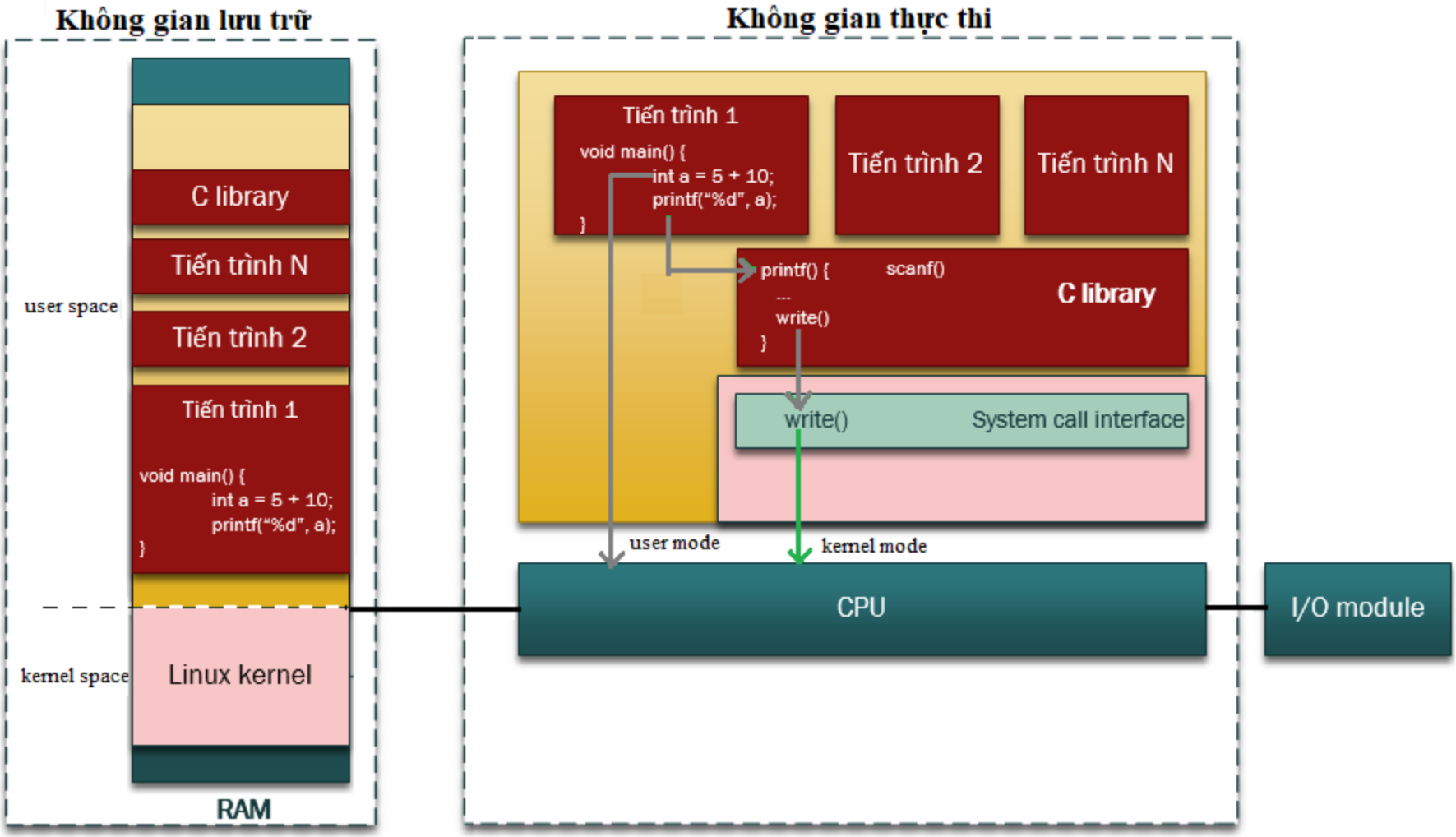
Nhân HĐH

Thư mục	Vai trò
/arch	Chứa mã nguồn giúp Linux kernel có thể thực thi được trên nhiều kiến trúc CPU khác nhau như x86, alpha, arm, mips, mk68, powerpc, sparc,…
/block	Chứa mã nguồn triển khai nhiệm vụ lập lịch cho các thiết bị lưu trữ.
/drivers	Chứa mã nguồn để triển khai nhiệm vụ điều khiển, giám sát, trao đổi dữ liệu với các thiết bị.
/fs	Chứa mã nguồn triển khai nhiệm vụ quản lý dữ liệu trên các thiết bị lưu trữ.
/ipc	Chứa mã nguồn triển khai nhiệm vụ giao tiếp giữa các tiến trình
/kernel	Chứa mã nguồn triển khai nhiệm vụ lập lịch và đồng bộ hoạt động của các tiến trình.
/mm	Chứa mã nguồn triển khai nhiệm vụ quản lý bộ nhớ
/net	Chứa mã nguồn triển khai nhiệm vụ xử lý các gói tin theo mô hình TCP/IP.

Khái niệm cơ sở

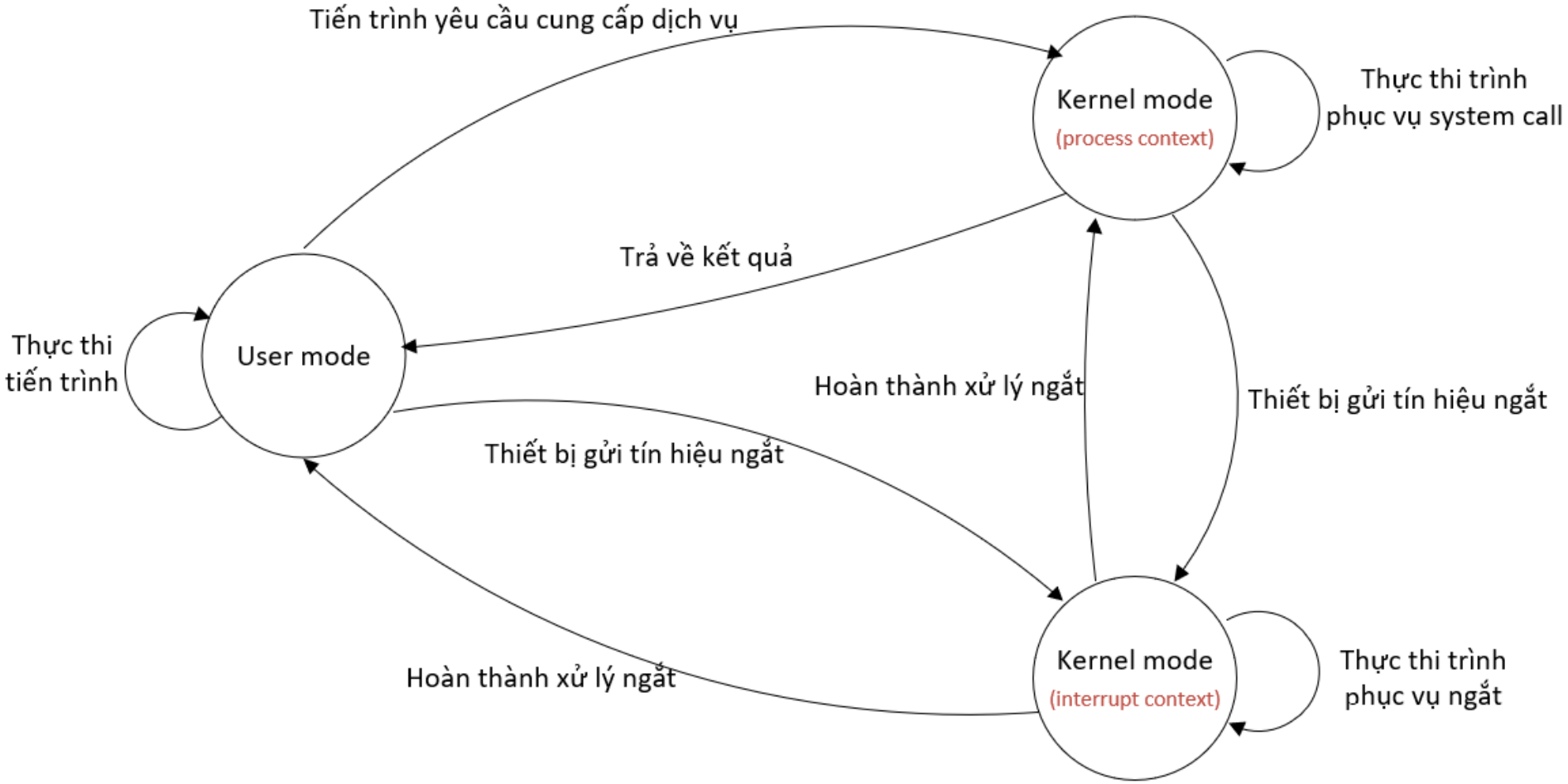
- User space và kernel space.
  - **Kernel space** là vùng không gian chứa các lệnh và dữ liệu của kernel.
  - **User space** là vùng không gian chứa các lệnh và dữ liệu của các tiến trình
- User mode và kernel mode.
  - Khi CPU thực thi các lệnh của kernel, thì nó hoạt động ở chế độ **kernel mode**. 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.

Khái niệm cơ sở



Khái niệm cơ sở

- Các chế độ hoạt động của CPU



Khái niệm cơ sở

- System call
  - Khi một tiến trình cần sử dụng một dịch vụ nào đó của kernel, tiến trình sẽ gọi một **system call**
  - Khi tiến trình gọi các system call, CPU phải chuyển sang chế độ kernel mode để thực thi các lệnh của kernel
    - ✓CPU đang trong ngữ cảnh **process context**
- Ngắt
  - 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
    - ✓CPU đang trong ngữ cảnh **interrupt context**

Module nhân Linux

- Các vấn đề liên quan tới Linux kernel module:
  - Linux kernel module là gì? Linux kernel module và Linux driver có mối quan hệ gì?
  - Cách viết một Linux kernel module.
  - Cách biên dịch (build) một Linux kernel module.
  - Cách nạp/lắp (load) module này vào và cách tháo (unload) module này ra khỏi kernel.

Module nhân Linux

- **Khái niệm Linux kernel module**
  - 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
- 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

Module nhân Linux

- Khi cần một module chưa có trong **kernel space**, kernel sẽ đưa module vào. Quá trình này có thể diễn ra một cách tự động:
  - Bước 1: Kernel kích hoạt tiến trình **modprobe** cùng với tham số truyền vào là tên của module (ví dụ xxx.ko).
  - Bước 2: Tiến trình **modprobe** kiểm tra file */lib/modules/<kernel-version>/modules.dep* xem xxx.ko có phụ thuộc vào module nào khác không. Giả sử xxx.ko phụ thuộc vào module yyy.ko.
  - Bước 3: Tiến trình **modprobe** sẽ kích hoạt tiến trình **insmod** để đưa các module phụ thuộc vào trước (yyy.ko), rồi mới tới module cần thiết (xxx.ko).

Module nhân Linux

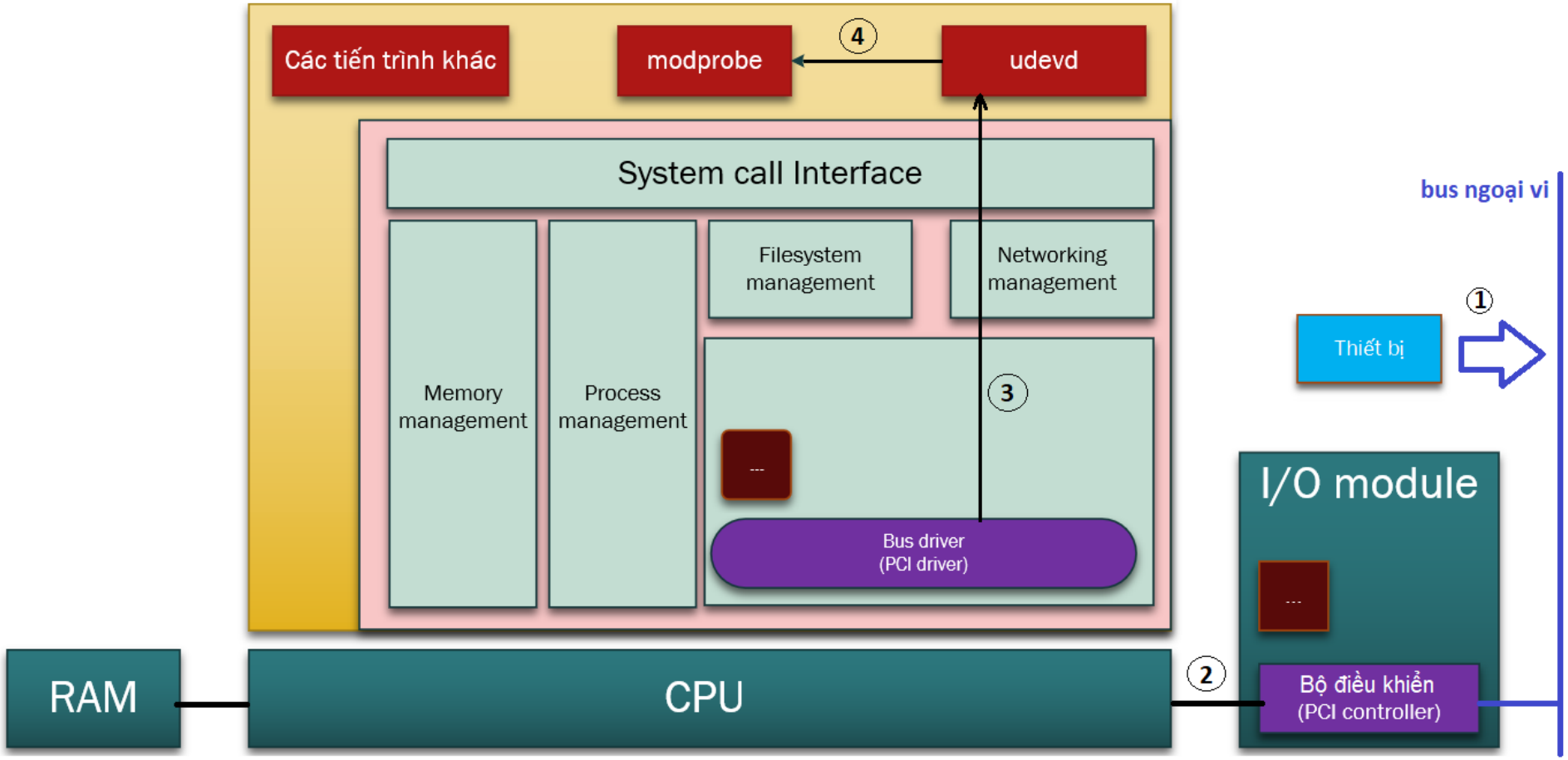
- Kernel kích hoạt tiến trình modprobe thế nào
  - Cách 1 là sử dụng **kmod**
    - ✓Là một thành phần của Linux kernel, hoạt động trong **kernel space**
    - ✓Khi một thành phần nào đó của kernel cần đưa một module vào trong kernel space, nó sẽ truyền tên module cho hàm **request_module** của kmod
    - ✓Hàm request_module sẽ gọi hàm **call_usermodehelper_setup** để sinh ra tiến trình **modprobe**
    - ✓Tham khảo mã nguồn của kmod tại /kernel/kmod.c

Module nhân Linux

- Kernel kích hoạt tiến trình modprobe thế nào
  - Cách 2 là sử dụng **udevd**
    - ✓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
    - ✓Bus driver sẽ gửi một bản tin (chứa thông tin về thiết bị) lên cho tiến trình udevd.
    - ✓Tiến trình udevd sẽ tra cứu file */lib/modules/<kernel-version>/modules.alias* để tìm ra driver nào tương thích với thiết bị
    - ✓**udevd** sinh ra tiến trình **modprobe**

Module nhân Linux

- Quá trình kích hoạt **modprobe** bằng **udevd**



Ví dụ: viết 1 linux kernel module

- Các bước:
  - Code
    - ✓Nên tạo thư mục chứa code
  - Biên dịch
  - Nạp/gỡ bỏ (Lắp/tháo) kernel module

Bước 1: tạo thư mục chứa code

```
cd /home/ubuntu
mkdir ltDriver
cd ltDriver
mkdir lab1
```

Ví dụ: viết 1 linux kernel module

- Bước 2: code

```
#include <linux/module.h>
/* thu vien nay dinh nghia cac macro nhu module_init va module_exit */

#define DRIVER_AUTHOR "Ten tac gia"
#define DRIVER_DESC   "Mo ta"

static int __init init_hello(void)
{
    printk("Hello Vietnam\n");
    return 0;
}

static void __exit exit_hello(void)
{
    printk("Goodbye Vietnam\n");
}

module_init(init_hello);
module_exit(exit_hello);

MODULE_LICENSE("GPL"); /* giay phep su dung cua module */
MODULE_AUTHOR(DRIVER_AUTHOR); /* tac gia cua module */
MODULE_DESCRIPTION(DRIVER_DESC); /* mo ta chuc nang cua module */
MODULE_SUPPORTED_DEVICE("testdevice"); /*kieu device ma module ho tro */
```

Ví dụ: viết 1 linux kernel module

- Biên dịch kernel module (Phương pháp Kbuild)
  - Tạo ra 2 file: một file có tên là **Makefile**, file còn lại có tên là **Kbuild**

```
#vim Makefile

KDIR = /lib/modules/`uname -r`/build

all:
        make -C $(KDIR) M=`pwd`

clean:
        make -C $(KDIR) M=`pwd` clean
```

- Thẻ **all** chứa câu lệnh để biên dịch các module trong thư mục hiện tại.
- Thẻ **clean** chứa lệnh xóa tất cả các object file có trong thư mục hiện tại.

```
#vim Kbuild

EXTRA_CFLAGS = -Wall

obj-m        = hello.o
```

- Biến **obj-m** chỉ ra rằng: object file sẽ được biên dịch theo kiểu kernel module.
- Cờ **-Wall** cho phép trình biên dịch hiển thị tất cả các bản tin cảnh báo trong quá trình biên dịch.

Ví dụ: viết 1 linux kernel module

- Tham khảo makefile
  - http://eslinuxprogramming.blogspot.com/2015/04/gnu-make.html
  - http://eslinuxprogramming.blogspot.com/2015/06/makefile-part-2.html
- Biên dịch tạo ra module nhân
  - Dùng lệnh **make** hoặc **make all**
  - **Kết quả:** file có tên mở rộng là **.ko** (kernel object)
- Xem thông tin module nhân
  - Sử dụng lệnh **modinfo <ten module>.ko**

Ví dụ: viết 1 linux kernel module

- Nạp module nhân
  - Để nạp module vào trong kernel, sử dụng lệnh **insmod**
  - Sau khi nạp xong, dùng lệnh **lsmod** để kiểm tra xem module đã được load thành công chưa
  - Tiếp theo, dùng lệnh **dmesg** để theo dõi quá trình hoạt động của module
- Gỡ module nhân
  - Dùng lệnh **rmmod** để tháo module ra khỏi kernel.

Driver

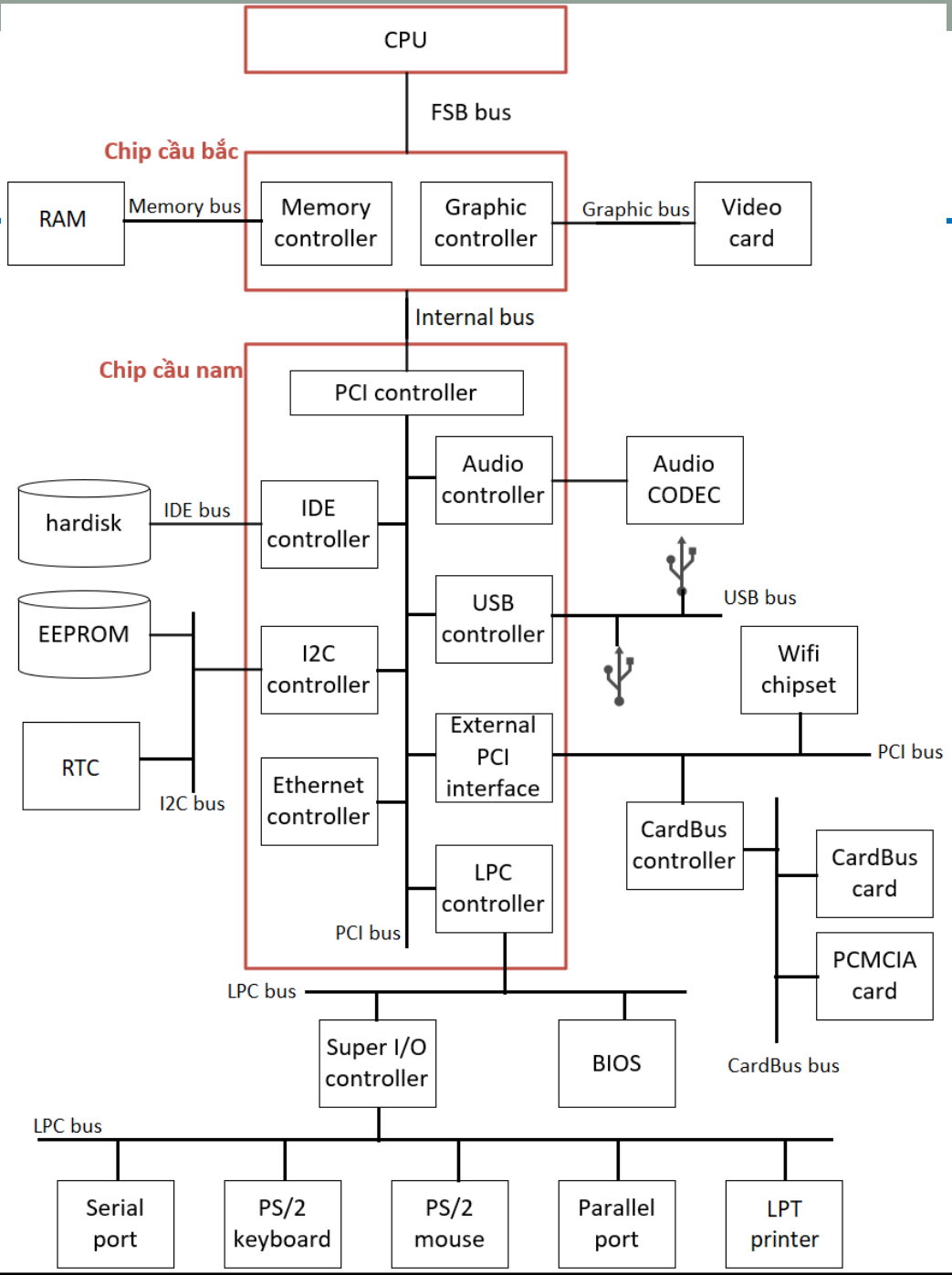
- Khái niệm driver
  - Là một phần mềm, gồm các lệnh, hướng dẫn CPU cách tương tác với thiết bị
- Các thiết bị có thể là chuột, bàn phím, ổ cứng, card mạng, loa, màn hình,…
- Các thiết bị này không được nối trực tiếp với CPU, bởi vì:
  - Hệ thống có nhiều thiết bị, nhưng số lượng chân của CPU hữu hạn.
  - Tốc độ làm việc của các thiết bị thấp hơn nhiều so với CPU

Controller

- Bộ điều khiển (device controller)
  - Là phần cứng trung gian cho phép CPU làm việc với thiết bị
- Phân loại bộ điều khiển 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 bộ điều khiển theo kỹ thuật giao tiếp với các thiết bị:
  - PCI (Peripheral Component Interconnect) 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.

Controller

- Sơ đồ khối của máy tính cá nhân sử dụng bộ xử lý Intel

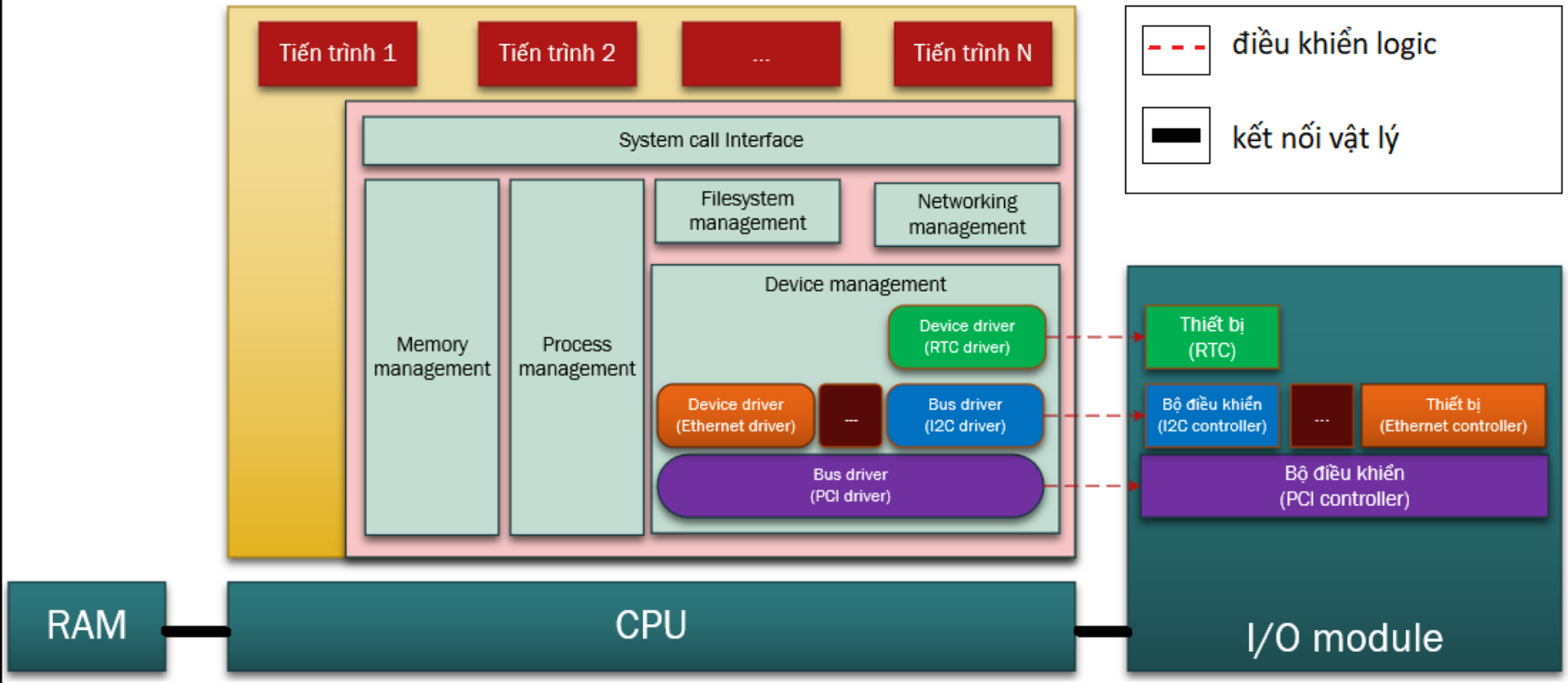


Controller

- Bộ điều khiển cũng chỉ là một thiết bị
  - 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 <=> Controller <=> CPU**

Controller

- Mối liên hệ giữa driver, bộ điều khiển và thiết bị dưới góc độ thực thi



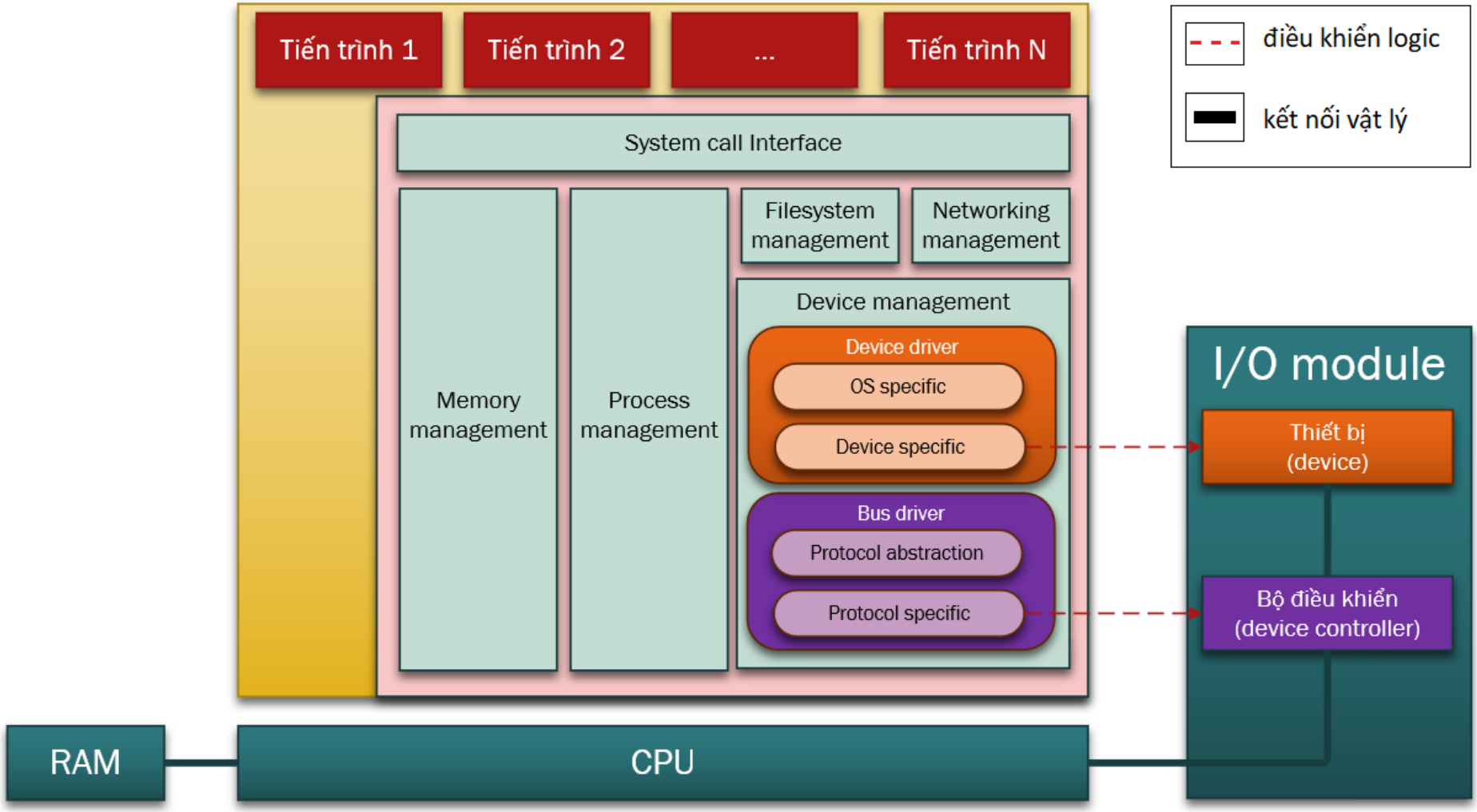
Device driver

- Device driver 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. Cho phép 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
    - ✓Chúng ta sử dụng datasheet của thiết bị để xây dựng thành phần này
    - ✓Datasheet là một tài liệu được cung cấp bởi nhà sản xuất thiết bị. Nó mô tả sơ đồ khối chức năng, nguyên lý hoạt động, hiệu suất hoạt động, đặc tính điện của thiết bị và đặc biệt là bản đồ thanh ghi (register map).

Bus driver

- Bus driver gồm 2 phần:
  - Thành phần **protocol abstraction**.
    - ✓Thành phần này che giấu đi sự phức tạp của các giao thức trên bus, cung cấp các dịch vụ cho device driver sử dụng.
    - ✓Ví dụ như đọc/ghi một thanh ghi nào đó của thiết bị.
  - Thành phần **protocol specific**.
    - ✓Thành phần này chứa các lệnh hướng dẫn CPU làm việc với bộ điều khiển, giúp đọc/ghi dữ liệu trên bus.
    - ✓Cũng được xây dựng dựa trên datasheet của bộ điều khiển.

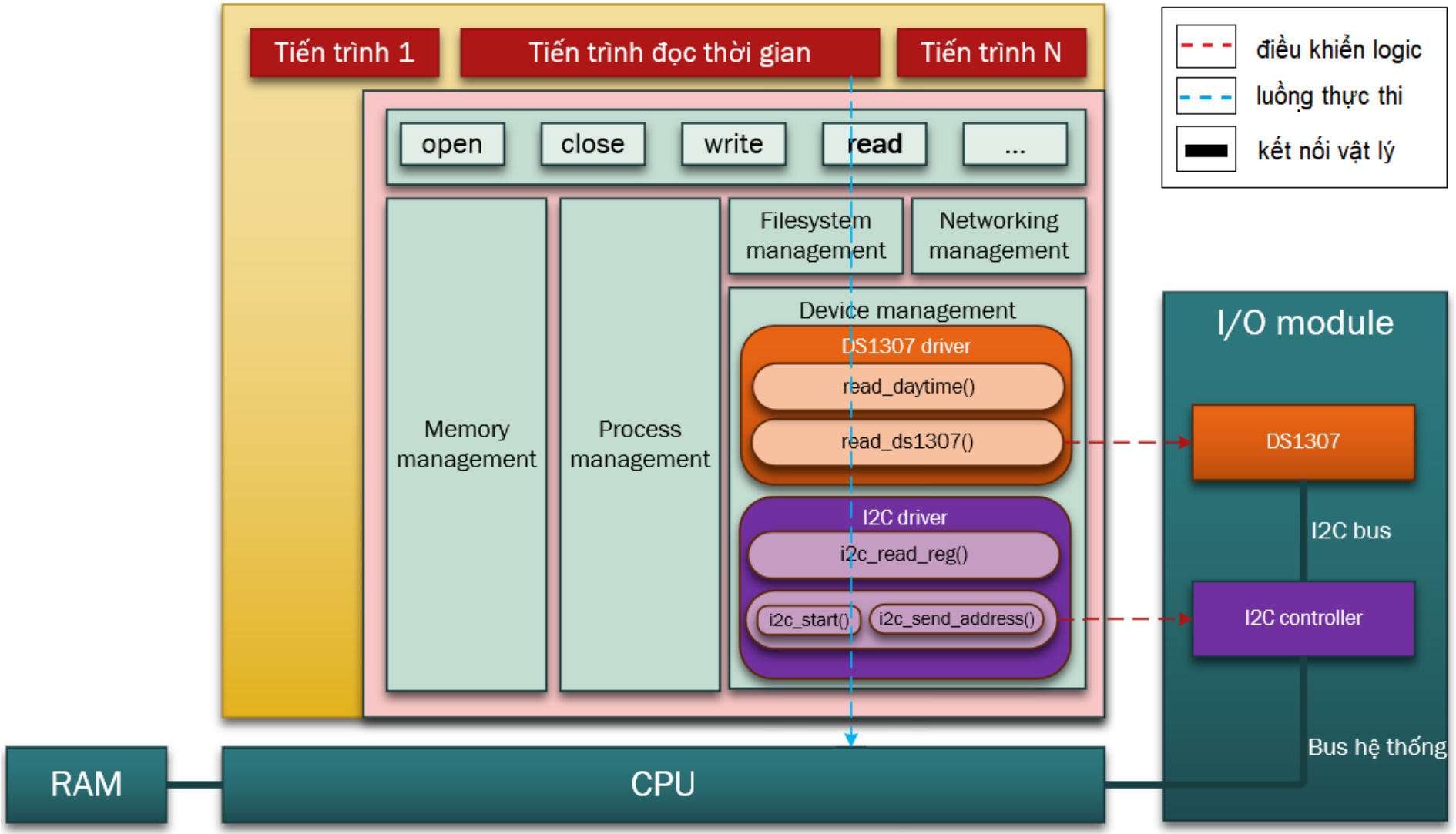
Các thành phần của bus driver và device driver



Ví dụ về driver

- Driver cho thiết bị và bộ điều khiển trên I2C (Inter-Integrated Circuit) bus
  - Xét một tiến trình đọc thời gian từ thiết bị DS1307
  - Trên I2C bus, DS1307 là thiết bị, I2C controller là bộ điều khiển. Driver của thiết bị DS1307 là DS1307 driver.
    - ✓Đây là một device driver.
  - Còn driver của I2C controller là I2C driver.
    - ✓Đây là một bus driver.

Ví dụ về driver



Ví dụ về driver

- Bước 1: Tiến trình gọi system call "read" của hệ điều hành.
- Bước 2: Hệ điều hành sẽ gọi hàm "read_daytime". Hàm này thuộc phần OS specific của DS1307 driver.
- Bước 3: Hàm "read_daytime" của OS specific tiếp tục gọi hàm "read_ds1307". Hàm này thuộc phần device specific của DS1307 driver.
- Bước 4: Hàm "read_ds1307" sẽ gọi hàm "i2c_read_reg" bảy lần để đọc dữ liệu từ 7 thanh ghi của thiết bị DS1307.
  - Các thanh ghi này chứa năm, tháng, ngày, thứ, giờ, phút, giây.
  - Hàm "i2c_read_reg" là một hàm thuộc phần protocol abstraction của I2C driver. Hàm này nhận 2 tham số đầu vào: một là địa chỉ của DS1307 trên I2C bus, và một là địa chỉ của thanh ghi cần đọc trong thiết bị DS1307 (năm, tháng, ngày, giờ,...).

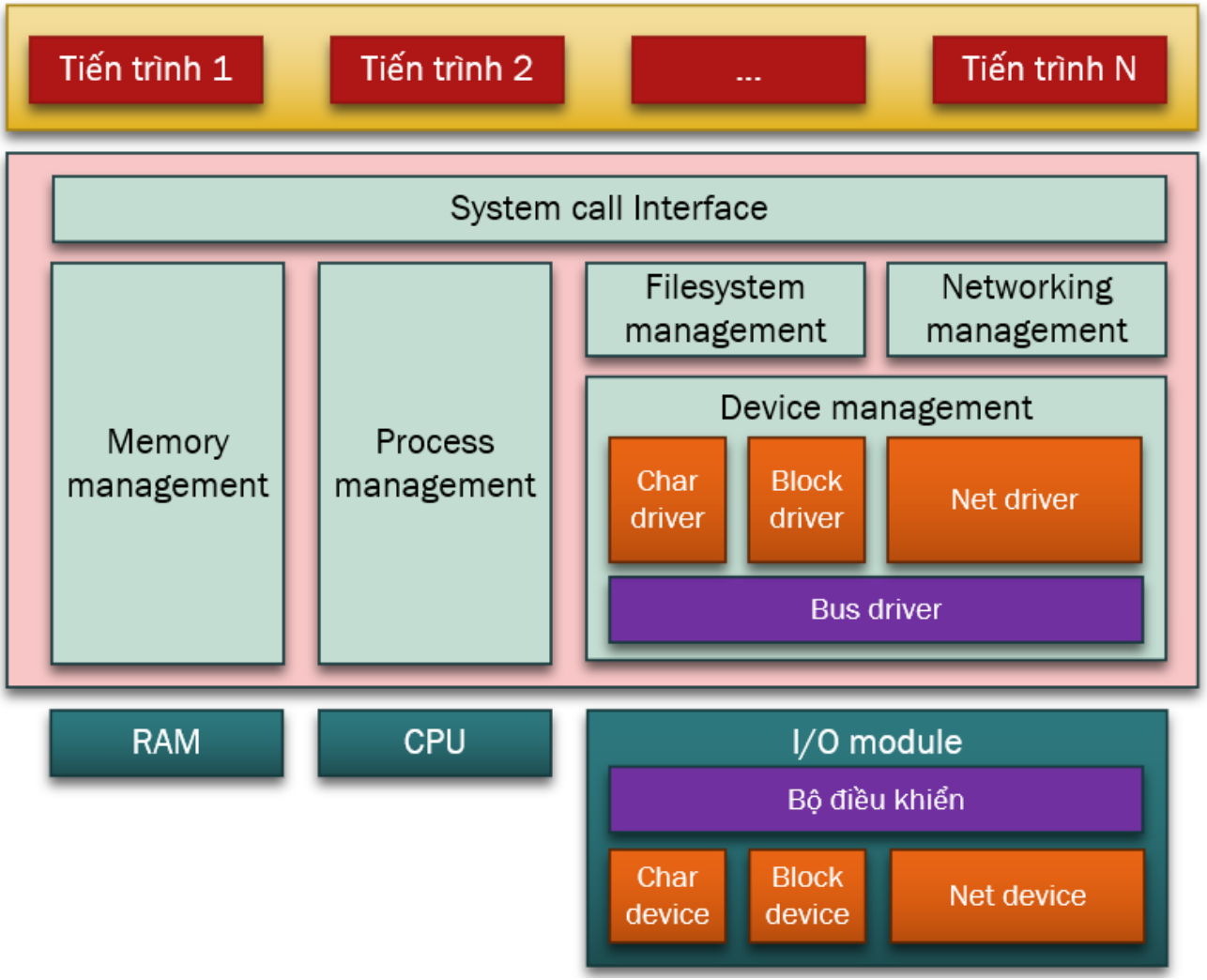
Ví dụ về driver

- Bước 5: Hàm "i2c_read_reg" sẽ lần lượt gọi các hàm "i2c_start", "i2c_send_address", "i2c_read_byte". Các hàm này thuộc phần protocol specific của I2C driver. Các hàm này sẽ hướng dẫn CPU làm việc với I2C controller thông qua bus hệ thống (bus địa chỉ, bus dữ liệu, bus điều khiển).
- Bước 6: I2C controller bắt đầu giao tiếp với DS1307 thông qua I2C bus.
- Bước 7: DS1307 gửi dữ liệu về năm, tháng, ngày, thứ, giờ, phút, giây tới cho I2C controller thông qua I2C bus.
- Bước 8: I2C controller gửi dữ liệu về cho CPU thông qua bus hệ thống.
- Bước 9: hàm "i2c_read_reg" trả dữ liệu về cho hàm "read_ds1307". Hàm "read_ds1307" trả dữ liệu về cho hàm "read_daytime". Hàm "read_daytime" trả dữ liệu về cho system call "read". Cuối cùng, system call "read" trả dữ liệu về cho tiến trình.

Linux device driver

- Bus driver
- Device driver
  - **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,…
  - **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.

Linux device driver



Vấn đề an toàn

- Bất kỳ việc kiểm tra bảo mật nào trong hệ thống đều được thực hiện bởi mã nhân.
  - Nếu trong nhân có lỗ hổng thì toàn bộ hệ thống sẽ mất an toàn
- Kiểm tra việc tải nhân
  - Lời gọi hệ thống init_module kiểm tra xem tiến trình có được phép tải mô-đun trong nhân hay không
- Kiểm tra đặc quyền truy xuất tài nguyên
- Viết trình điều khiển cần tránh các lỗi bảo mật
  - Tràn bộ đệm, khi đó lập trình viên quên kiểm tra lượng dữ liệu ghi vào bộ đệm và dữ liệu kết thúc được viết vượt quá phần cuối của bộ đệm
- Tránh các nhân được biên dịch bởi một người không tin cậy

Quản lý phiên bản và giấy phép

- Quản lý phiên bản
  - Mọi gói phần phần được dùng trong Linux đều có số phát hành riêng và thường có sự phụ thuộc lẫn nhau
  - Trình quản lý gói phân phối nói chung không cho phép nâng cấp cho đến khi các phụ thuộc được thỏa mãn
- Quản lý giấy phép
  - Linux được cấp giấy phép của cộng đồng GNU (General Public License – GPL)
    - ✓GPL cho phép bất cứ ai cũng có thể phân phối lại hoặc bán phiên bản của mình, miễn là người nhận có đầy đủ quyền truy cập vào nguồn và có thể thực hiện các quyền tương tự

HỎI - ĐÁP