HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

CHƯƠNG 3

Các thành phần phần mềm của hệ thống nhúng và vi xử lý 8051

Giảng viên: Ths. Đinh Xuân Trường

Bộ môn: Khoa học máy tính - Khoa CNTT1

Email: truongdx@ptit.edu.vn

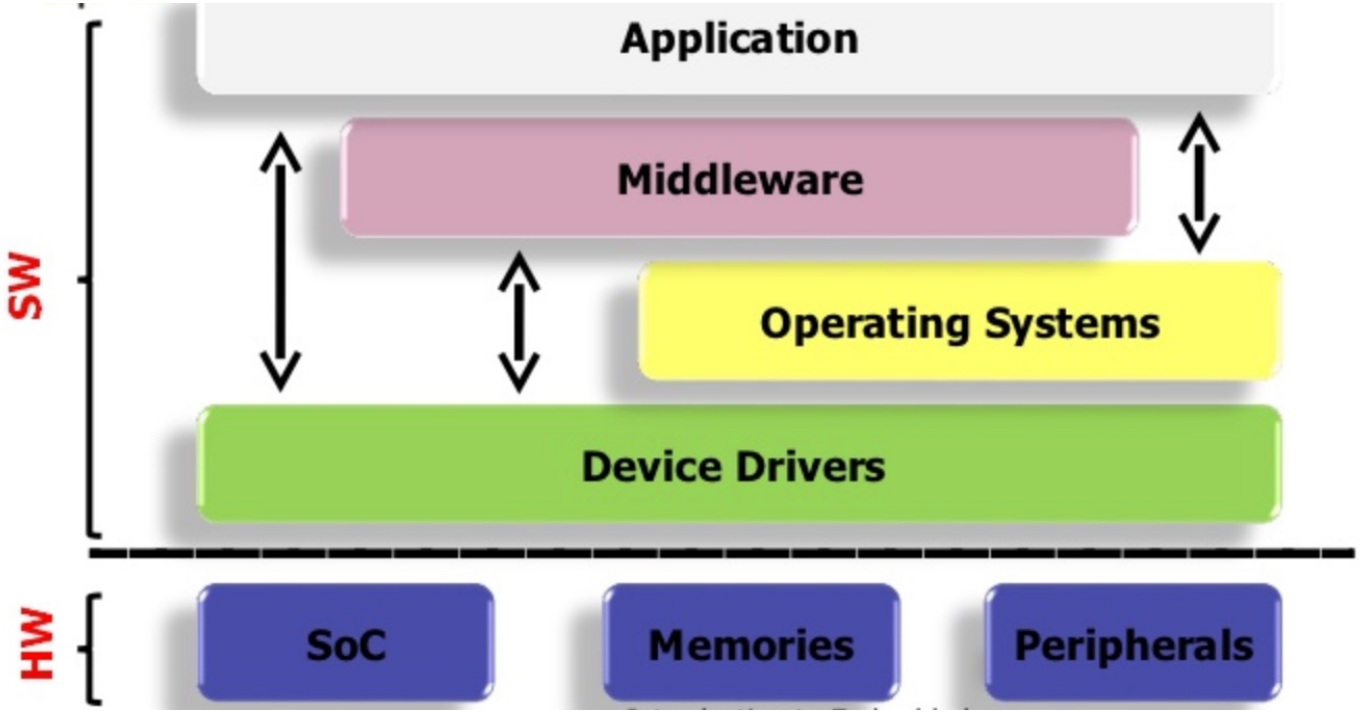
NỘI DUNG CHÍNH

3.1. Trình điều khiển thiết bị

3.2. Hệ điều hành thời gian thực

3.3. Middleware và các phần mềm ứng dụng

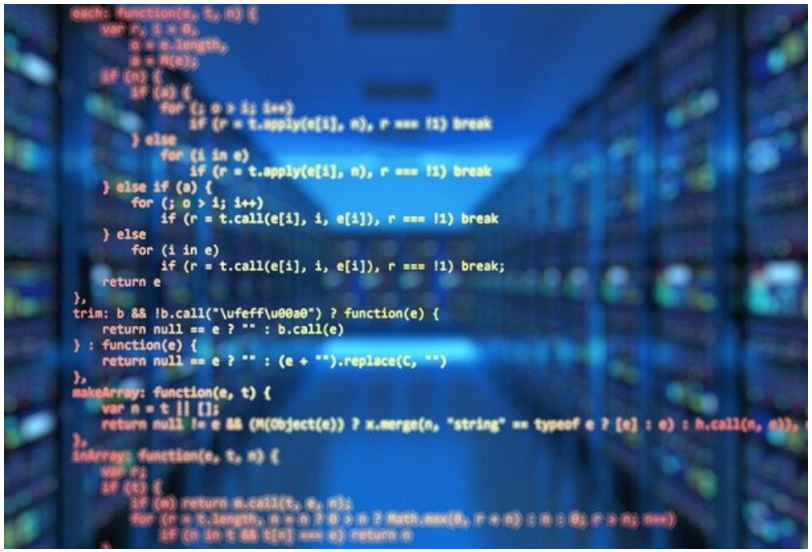
- Tổng quan phần mềm của Hệ thống nhúng
  - **Operating System:** Phần mềm hệ thống
  - **Application**: Phần mềm ứng dụng



Giới thiệu về phần mềm nhúng

Giới thiệu về phần mềm nhúng

- Hai cách tiếp cận xây dựng phần mềm nhúng:
  - Thiết kế phần mềm dựa vào thủ tục thông thường.
  - Thiết kế dựa vào Hệ điều hành nhúng.



- ***Thiết kế phần mềm dựa vào thủ tục thông thường***.
  - Sử dụng siêu vòng lặp cho các ứng dụng không phụ thuộc vào thời gian đáp ứng của hệ thống.
  - Mỗi thủ tục được thực thi nối tiếp:
    - 1. Cấu hình các tham số dùng chung và thực hiện khởi tạo cho các thành phần phần cứng như là bộ nhớ, thanh ghi.
    - 2. Bắt đầu với tác vụ đầu tiên và thực thi nó.
    - 3. Thực thi tác vụ thứ 2.
    - 4. Thực thi tác vụ tiếp theo.
    - 5. ….
    - 6. ….
    - 7. Thực thi tác vụ cuối cùng.
    - 8. Nhảy về tác đầu tiên và thực thi theo luồng tương tự.

```
void main ()
{
Configurations ();
Initialisations ();
while (1)
{
Task 1 ();
Task 2 ();
:
:
Task n ();
}
}
```

❖ *Thiết kế phần mềm dựa vào thủ tục thông thường.*

- Ưu điểm:
  - Không yêu cầu hệ điều hành
  - Thiết kế đơn giản
  - Giá thành rẻ
- Nhược điểm:
  - Dễ bị ảnh hưởng toàn hệ thống nếu có một tác vụ lỗi
  - Thiếu thời gian xử lý

```
void main ()
{
Configurations ();
Initialisations ();
while (1)
{
Task 1 ();
Task 2 ();
:
:
Task n ();
}
}
```

❖ ***Thiết kế phần mềm dựa vào thủ tục thông thường.***

Giới thiệu về phần mềm nhúng

- ***Thiết kế phần mềm dựa vào Hệ điều hành nhúng:***
- Một hệ thống hoạt động chứa hệ điều hành.
- Tạo và chạy ứng dụng trên hệ điều hành.
- Có thể là hệ điều hành chung hoặc hệ điều hành thời gian thực (RTOS)
  - Hệ điều hành chung:
    - hợp nhất hệ thống nhúng với hệ thống tính toán chung của hệ điều hành
    - hỗ trợ các giao diện lập trình có thể được sử dụng
  - Hệ điều hành thời gian thực:
    - sản phẩm nhúng yêu cầu thời gian đáp ứng
    - chứa một phần mềm chịu trách nghiệm phòng ngừa đa nhiệm, lập kế hoạch cho lịch trình thực hiện, đa luồng, một hệ điều hành thời gian thực cho phép linh hoạt đặt lịch các tài nguyên hệ thống như CPU và bộ nhớ và đưa ra một vài cách để giao tiếp giữa các nhiệm vụ

❖ *Thiết kế phần mềm dựa vào Hệ điều hành nhúng.*

- Ưu điểm:
  - Không yêu cầu hệ điều hành
  - Thiết kế đơn giản
  - Giá thành rẻ
- Nhược điểm:
  - Dễ bị ảnh hưởng toàn hệ thống nếu có một tác vụ lỗi
  - Thiếu thời gian xử lý

```
void main ()
{
Configurations ();
Initialisations ();
while (1)
{
Task 1 ();
Task 2 ();
:
:
Task n ();
}
}
```

3.1 Trình điều khiển thiết bị

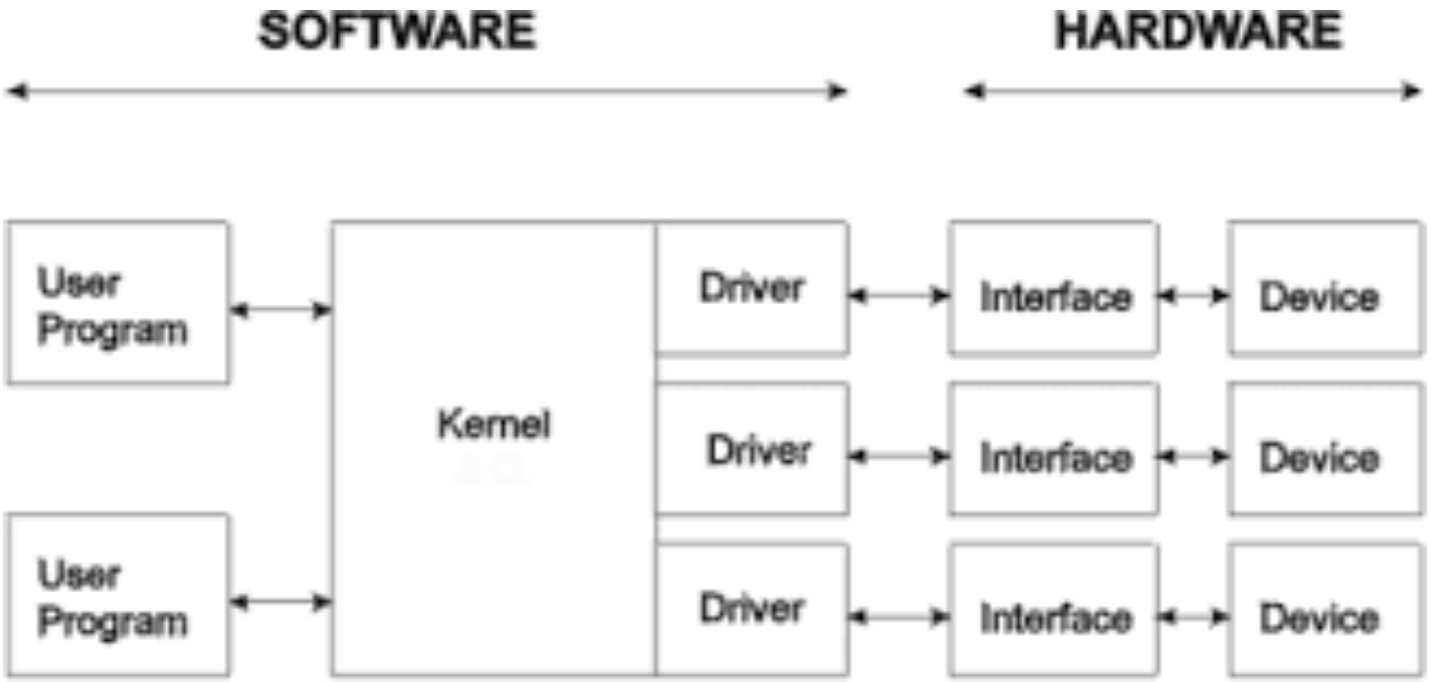
I. ĐỊNH NGHĨA TRÌNH ĐIỀU KHIỂN THIẾT BỊ (Device driver)

- Device driver: kết nối giữa OS và các thiết bị I/O.
- Chức năng: dịch, convert các yêu cầu từ OS thành các câu lệnh mà bộ điều khiển ngoại vi có thể hiểu.
- Là một phần của nhân OS (drivers là các module phần mềm được xây dựng riêng biệt và cài vào OS khi cần thiết).

ĐỊNH NGHĨA TRÌNH ĐIỀU KHIỂN THIẾT BỊ (Device driver)

- Các ví dụ: driver máy in, Bluetooth driver …
- Một hệ thống hoàn chỉnh có rất nhiều driver được cài đặt trên hệ điều hành.
- Device drivers hoạt động ở mode privileged: cần được thiết kế, bảo mật tốt, tránh mã độc.

ĐỊNH NGHĨA TRÌNH ĐIỀU KHIỂN THIẾT BỊ (Device driver)



Ưu điểm của Device driver

1. Đơn giản hóa hoạt động của OS

2. Không có device driver, OS chịu trách nhiệm giao tiếp trực tiếp với phần cứng ⇒ gây quá tải cho OS.

3. Nếu một thiết bị mới được cài đặt, dẫn đến phải thay đổi OS.

Các vấn đề thiết kế chính

1. OS / Driver Communication
   - Trao đổi thông tin (command, data)
   - Các hàm hỗ trợ mà kernel cung cấp

2. Driver / Hardware Communication
   - Trao đổi thông tin (command, data)
   - Phần mềm giao tiếp với Phần cứng
   - Phần cứng giao tiếp với Phần mềm

3. Driver Internal Operations
   - Dịch command nhận được từ OS
   - Sắp xếp thứ tự các yêu cầu
   - Quản lý truyền dữ liệu giữa các interfaces (OS và hardware)
   - Chấp nhận và xử lý các ngắt
   - Duy trì sự thống nhất của cấu trúc dữ liệu giữa driver và kernel.

Các loại Device Drivers

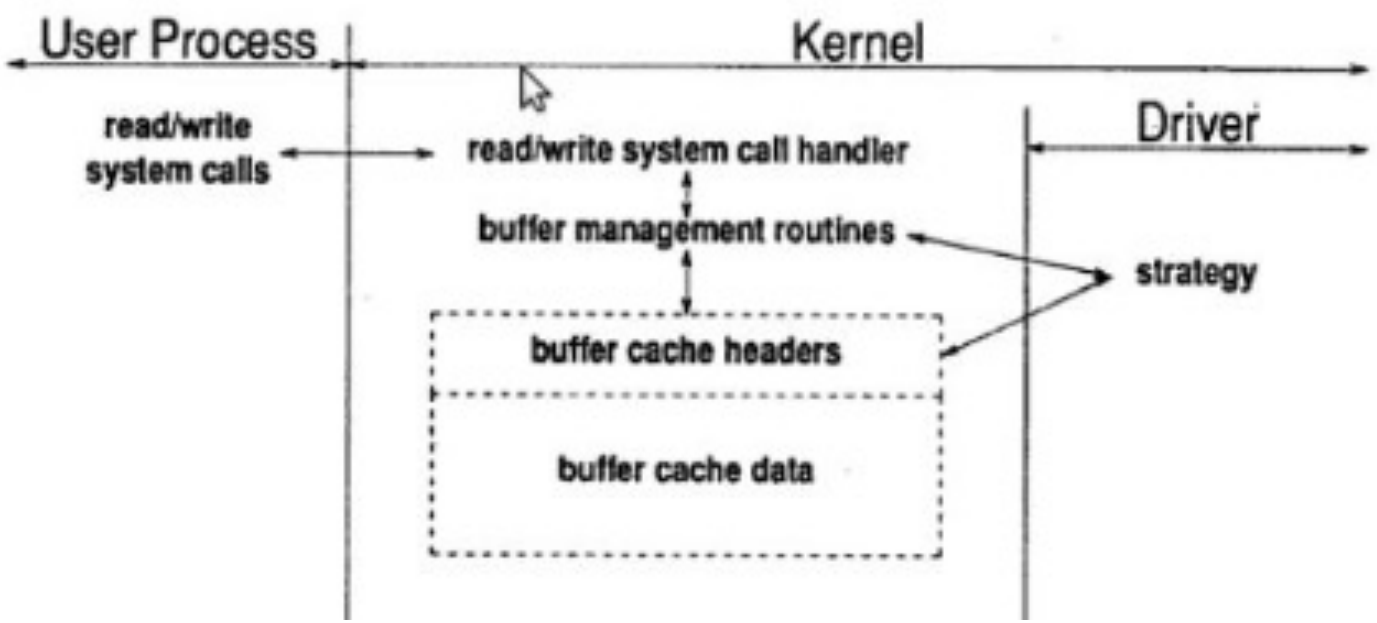
- Block Driver
- Character Driver
- Terminal Driver
- Stream Driver

Block Drivers

- Giao tiếp với OS thông qua tập các bộ đệm với kích thước cố định (block of data).
- OS quản lý bộ nhớ cache của các bộ đệm và đáp ứng yêu cầu người dùng về dữ liệu bằng cách truy cập vào cache.
- Driver chỉ được gọi khi các dữ liệu yêu cầu không có trong cache hoặc khi bộ đệm trong cache thay đổi.
- Driver làm việc với các yêu cầu từ OS để lấp đầy hoặc xóa các bộ đệm với kích thước cố định.

Block Drivers

❖ Ví dụ: disks được ứng dụng như là block devices.

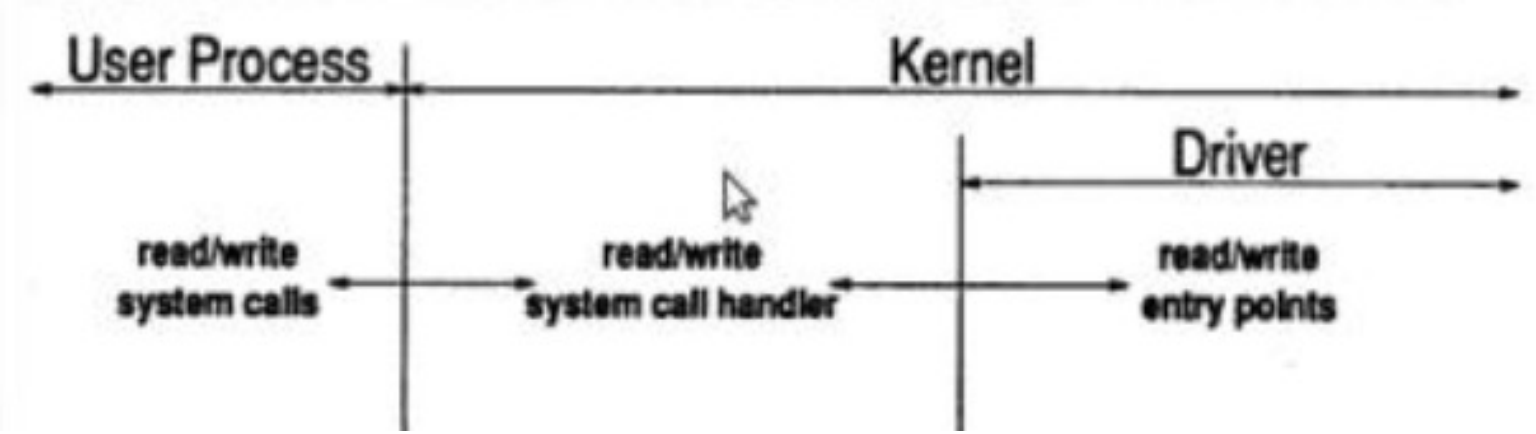


Character Drivers

- Các yêu cầu với kích thước gói tin độ dài bất kì của I/O.
- Được dùng cho các thiết bị giao tiếp theo byte tại 1 thời điểm hoặc với khối data lớn hoặc nhỏ hơn khối đệm dữ liệu của block drivers

  VD: máy in
- Các yêu cầu từ I/O được chuyển trực tiếp đến driver và driver chuyển dữ liệu đến bộ nhớ VXL.

Character Drivers



Block driver vs Character Drivers

Block Driver	Character Driver
Truyền dữ liệu đến kernel buffer cache	Truyền dữ liệu trực tiếp đến tiến trình người dùng
Không phù hợp cho các thiết bị giao tiếp với gói tin kích thước thay đổi	Phù hợp với các thiết bị giao tiếp theo gói tin có kích thước thay đổi
Có thể được dùng để hỗ trợ thiết bị chạy trên nền UNIX	Không hỗ trợ thiết bị chạy nền UNIX
Phù hợp hỗ trợ các ổ đĩa, ổ cứng	Phù hợp hỗ trợ cổng nối tiếp, cổng song song, card mạng, bàn phím…

Đặc điểm của một device driver

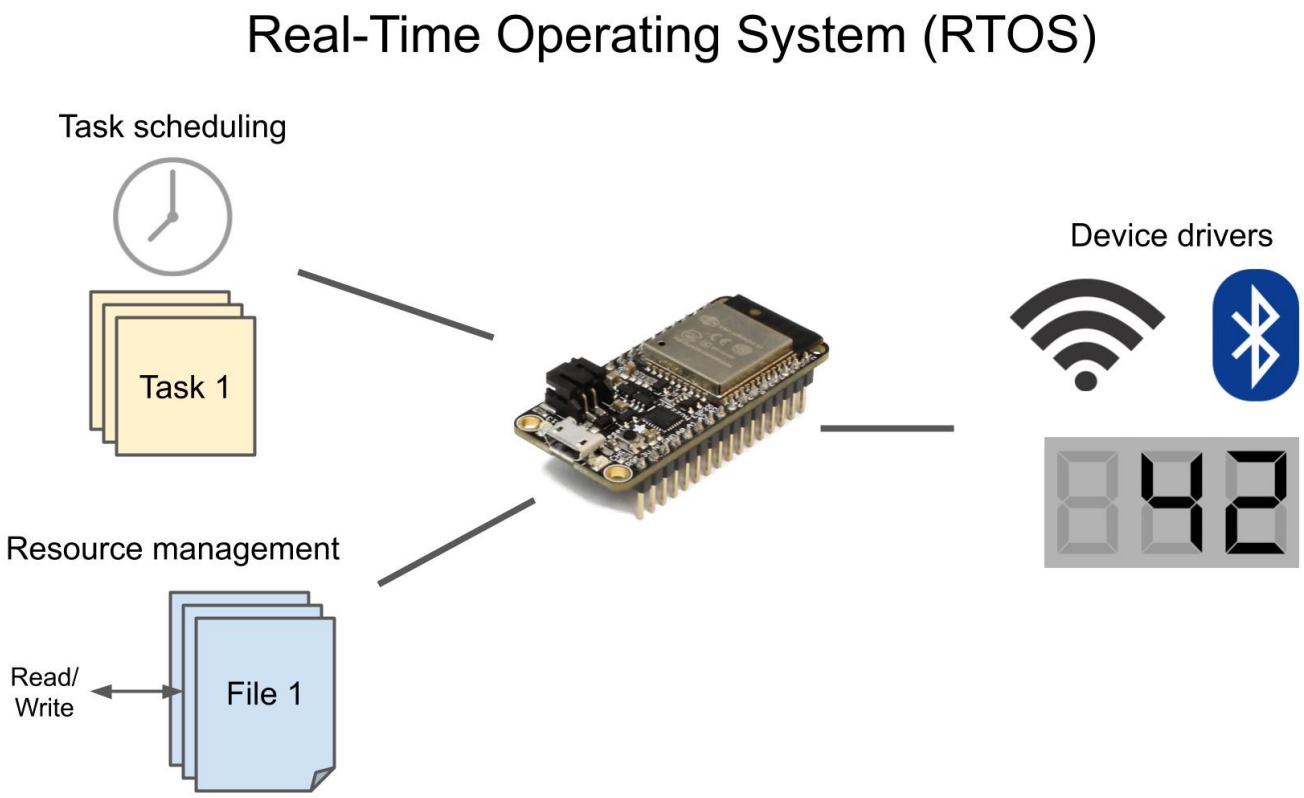
- Driver là tập các entry points được gọi bởi OS.
- Một driver bao gồm:
  - Cấu trúc data riêng biệt cho từng thiết bị.
  - Giao thức riêng biệt cho từng driver
- Hầu hết các drivers được viết trên 1 file nguồn.
- Phần đầu của driver được gọi là khai báo prologue.

Đặc điểm của một device driver

- Khai báo (prologue) gồm:
  - #include
  - #define
  - Khai báo các biến và đặc điểm các biến

- Phần còn lại của driver:
  - Entry points (các hàm C được tham chiếu bởi OS)
  - Routines (các hàm C tương tác với driver)

3.2 . Real Time Operating System (RTOS)



II. Real Time Operating System (RTOS)

- Định nghĩa: là hệ điều hành đa tác vụ cho các ứng dụng với các ràng buộc (constraints) thời gian thực.

- Ràng buộc về thời gian thực: yêu cầu một độ trễ nhất định cho một sự kiện và đáp ứng của hệ thống.

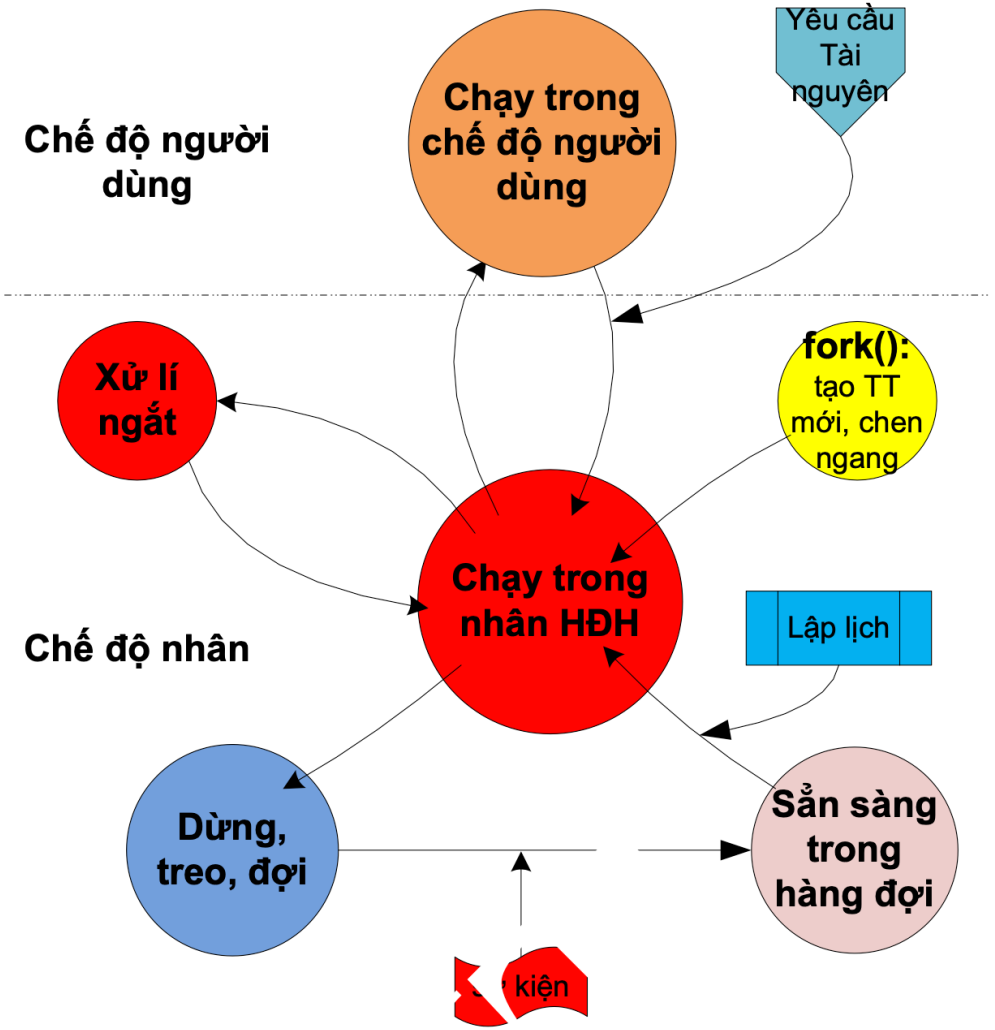
Lợi ích Real Time Operating System (RTOS)

- Chia nhỏ project để "trị".
- Dễ dàng phân chia công việc trong làm việc nhóm.

Các chức năng cơ bản

- Nhân HĐH (kernel)
- Quản lý tiến trình
- Quản lý tài nguyên
- Quản lý thiết bị
- Quản lý hệ thống thiết bị ngoại vi I/O
- Quản lý các thiết bị mạng

Quản lý tiến trình



Phân cấp các tiến trình ưu tiên

- Phân cấp các mức độ ưu tiên người dùng, được gọi là phân cấp ưu tiên tĩnh, hay phân cấp ưu tiên thời gian thực.
- Phân cấp ưu tiên thời gian thực cao hơn phân cấp ưu tiên tĩnh và phân cấp ưu tiên rỗi.
- Các tác vụ ưu tiên rỗi chạy khi không có các tác vụ ưu tiên cao đang chạy.

Quản lý tiến trình bằng can thiệp

- RTOS can thiệp một tiến trình ưu tiên thấp khi một tin nhắn hoặc sự kiện tiến trình ưu tiên cao đang đợi .

- RTOS có các điểm can thiệp tại cuối của critical code, và do đó RTOS có thể được can thiệp tại những điểm này bởi các tác vụ ưu tiên cao thời gian thực.

- Một phần nhỏ các hàm RTOS là không thể can thiệp.

Quản lý tiến trình bằng thừa hưởng mức ưu tiên (priority inheritance)

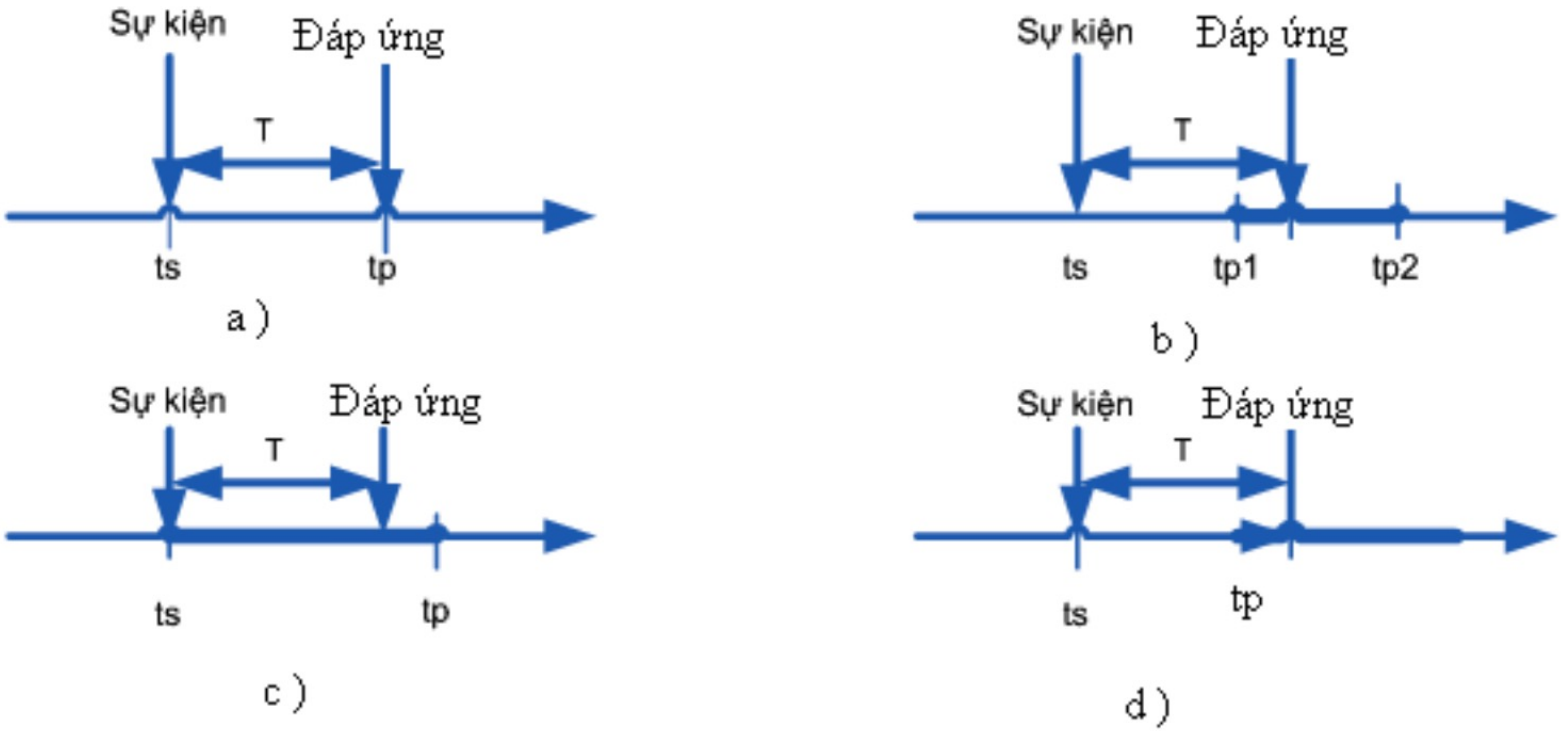
- Thừa hưởng ưu tiên cho phép chia sẻ tài nguyên với các tác vụ ưu tiên thấp.
- Một tác vụ mức ưu tiên trung bình sẽ không thể can thiệp vào một tác vụ ưu tiên thấp khi nó đang bị khóa để chạy cùng với tác vụ mức ưu tiên cao
- Giúp cải thiện hiệu suất hệ thống.

Thiết kế của RTOS

- Một hệ thống nhúng với 1 CPU đơn có thể chạy duy nhất 1 tiến trình tại 1 thời điểm.
- Một tiến trình tại một thời điểm bất kì có thể do ISR, hoặc hàm kernel hoặc task.
- Chạy các luồng trong nhân để tăng tốc độ xử lý.

Thiết kế RTOS

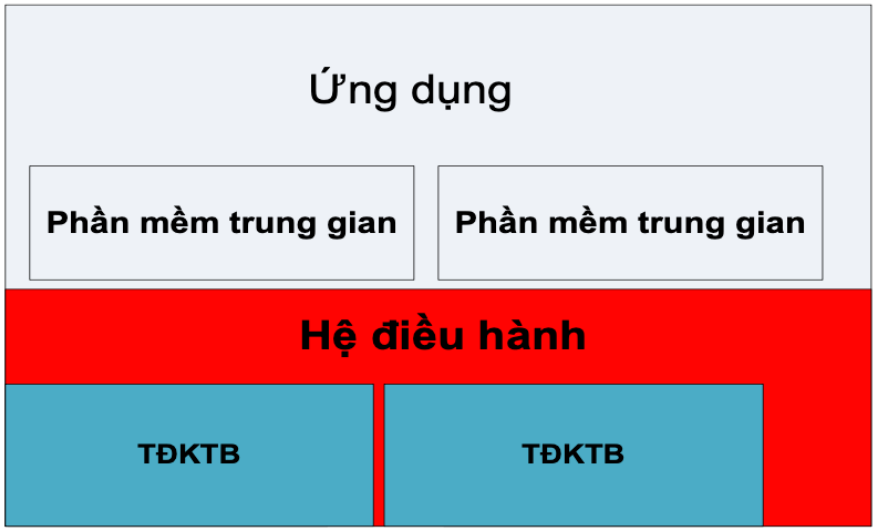
- Cung cấp xử lý hiệu quả của ISRs, drivers, ISTs, luồng …
- Bật tắt các ngắt
- Phân phát, thu hồi bộ nhớ tại thời điểm cố định và các khối bộ nhớ.
- Cung cấp hiệu quả việc lên lịch, chạy và dừng các tasks in trường hợp nhiều tasks.


Sự kiện
Đáp ứng
T
ts
tp
a)
Sự kiện
Đáp ứng
T
ts
tp1
tp2
b)
Sự kiện
Đáp ứng
T
ts
tp
c)
Sự kiện
Đáp ứng
T
ts
tp
d)

Thiết kế RTOS

- Việc quản lý I/O (thiết bị, files, mailboxes …) trở nên dễ dàng với RTOS
- Quản lý hiệu quả nhiều trạng thái của CPU, thiết bị như ổ cứng trong ngoài, thiết bị ảo…
- Một hệ thống nhúng chạy OS giúp hỗ trợ làm việc với nhiều đầu vào một cách đơn giản hơn (thông qua drivers) và xử lý dữ liệu dễ dàng hơn.

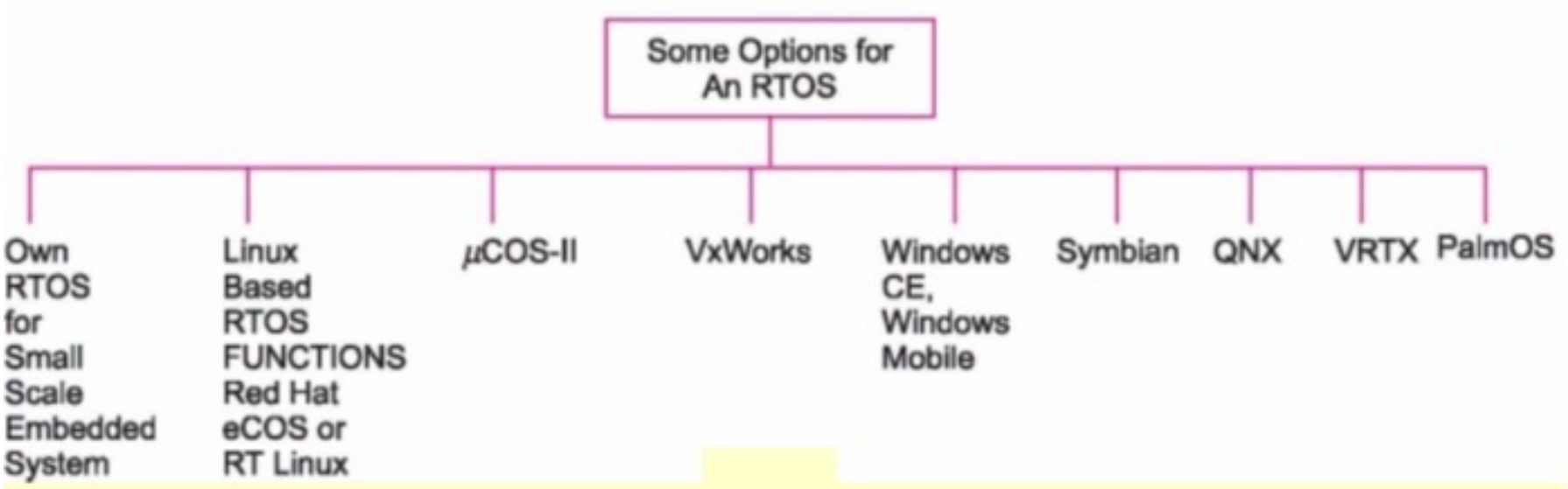

Ứng dụng
Phần mềm trung gian
Phần mềm trung gian
TĐKTB
TĐKTB
nhân Thời gian thực


Ứng dụng
Phần mềm trung gian
Phần mềm trung gian
Hệ điều hành
TĐKTB
TĐKTB

Thiết kế RTOS

- Việc quản lý I/O (thiết bị, files, mailboxes …) trở nên dễ dàng với RTOS
- Quản lý hiệu quả nhiều trạng thái của CPU, thiết bị nhu ổ cứng trong ngoài, thiết bị ảo.

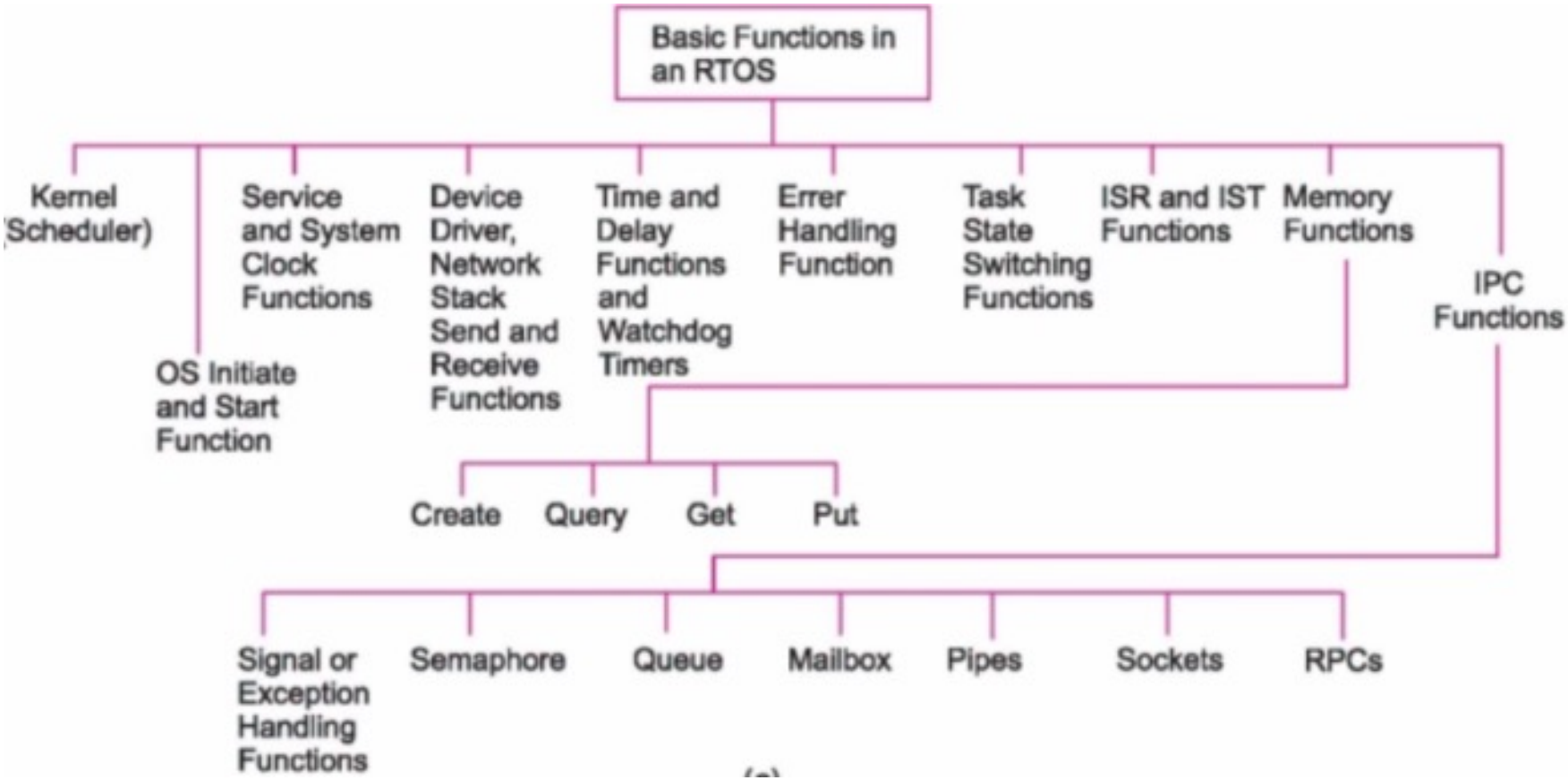
Các lựa chọn cho RTOS



Các yêu thiết kế hệ thống nhúng đa nhiệm phức tạp

- IDE
- Các hàm đa nhiệm C/C++
- Đồng hồ thời gian thực – timers
- Lập lịch (scheduler)
- Device drivers, device manager
- Các hàm giao tiếp giữa các tiến trình, hàm xử lý đa luồng …
- Các hàm khác: TCP/IP, USB port, mạng
- Phần mềm test, debug cho testing RTOS

Các hàm cơ bản trong RTOS



Phân loại RTOSes

- In-house Developed RTOSes

1. Codes được viết cho các yêu cầu ứng dụng hoặc sản phẩm cụ thể.
2. Có thể tinh chỉnh lại các yêu cầu.

Phân loại RTOSes

- RTOSes thương mại

1. Cung cấp các công cụ test, debug.
2. Hỗ trợ nhiều cấu trúc VXL (ARM, x86 …)
3. Hỗ trợ GÚI
4. Hỗ trợ giao tiếp nhiều thiết bị, nhiều giao thức kết nối khác nhau.
5. Hỗ trợ tối ưu phần mềm – thiết bị.
6. Đơn giản hóa quá trình code của nhà phát triển.
7. Tăng tốc độ phát triển sản phẩm.
8. Tiết kiệm thời gian, chi phí bảo dưỡng.

Phân loại RTOSes

- RTOSes thương mại

1. Cung cấp các công cụ test, debug.
2. Hỗ trợ nhiều cấu trúc VXL (ARM, x86 …)
3. Hỗ trợ GÚI
4. Hỗ trợ giao tiếp nhiều thiết bị, nhiều giao thức kết nối khác nhau.
5. Hỗ trợ tối ưu phần mềm – thiết bị.
6. Đơn giản hóa quá trình code của nhà phát triển.
7. Tăng tốc độ phát triển sản phẩm.
8. Tiết kiệm thời gian, chi phí bảo dưỡng.

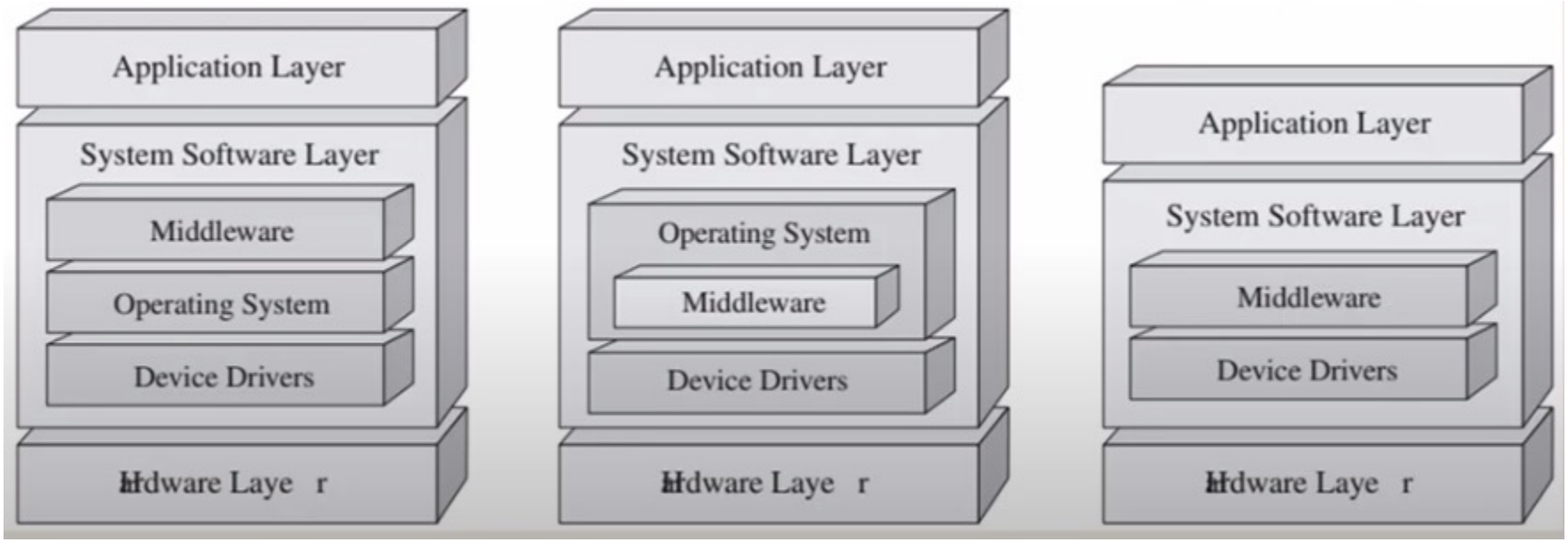
Phân loại RTOSes

- General purposes OS with RTOS

1. Linux, Windows OS.
2. Có cấu hình GUIs mạnh, nhiều giao diện multimedia, giá thành thấp.
3. Hỗ trợ GUIs

III. Middleware, Application

1. Middleware: nằm ở nhiều nơi, tách biệt với app, OS, driver.
2. Application: phần mềm nằm ở lớp trên cùng, giao tiếp với người dung.

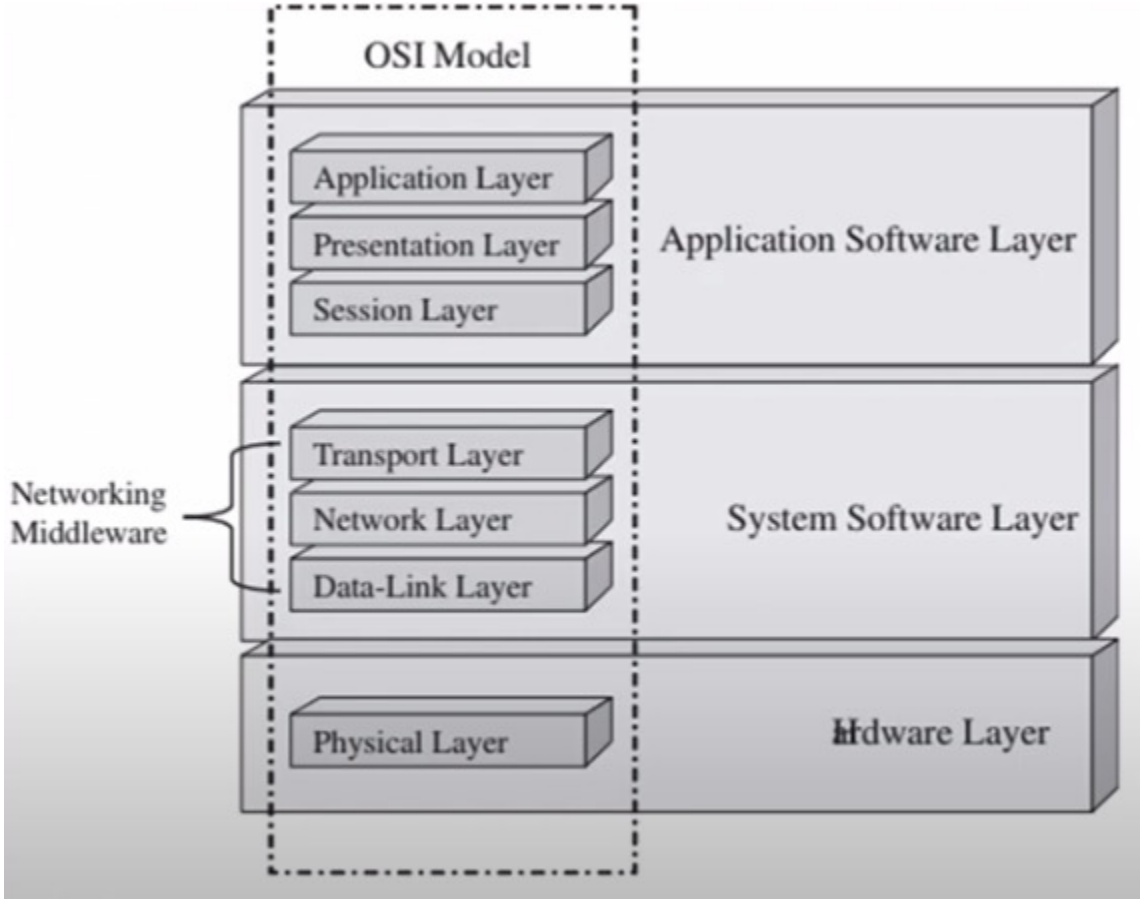


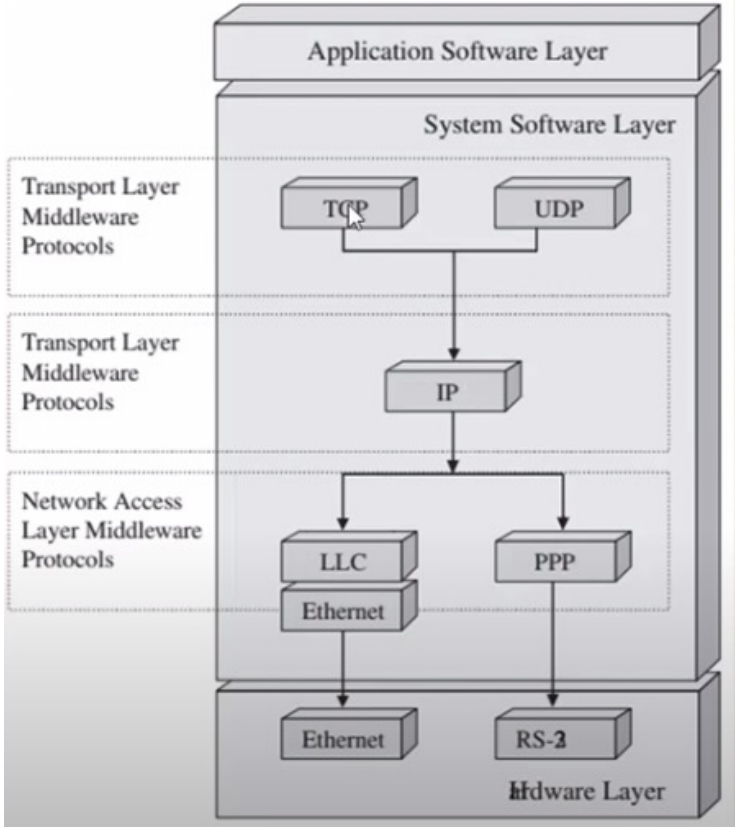
Middleware

1. Middleware: xem như phần mềm trung gian, đảm bảo linh hoạt, an ninh và giúp liên kết các phần mềm ở các lớp kề cận.

2. Chứa các hàm dịch vụ, được gọi sử dụng nhiều lần bởi các phần mềm khác.

Middleware (ví dụ)





Phát triển phần mềm nhúng cần những gì?

1. Phần mềm biên dịch
2. Board phát triển
3. Bộ nạp/Bộ gỡ lỗi
4. Thư viện (cho lõi ARM, driver ngoại vi …)
5. Tài liệu (datasheet)

1. Tạo project
2. Cấu hình phần mềm biên dịch (chọn loại chip, chọn loại mạch nạp)
3. Viết chương trình và biên dịch
4. Nạp vào bộ nhớ flash của board phát triển
5. Chạy chương trình và kiểm tra lỗi