



# HỆ ĐIỀU HÀNH

## CHƯƠNG 4: ĐỊNH THỜI CPU (PHẦN 3)

Định thời CPU là hoạt động quan trọng của thành phần quản lý tiến trình và có ảnh hưởng rất lớn đến hiệu suất máy tính cũng như trải nghiệm của người dùng. Trong chương này, người học được trình bày về mục đích và các tiêu chuẩn định thời, cũng như các chiến lược định thời CPU cơ bản.



# NỘI DUNG

5. Định thời tiêu trình (Thread scheduling)
6. Định thời đa bộ xử lý (Multiple-processor scheduling)
7. Định thời theo thời gian thực (Real-time CPU scheduling)
8. Định thời trên một số hệ điều hành
  - Linux
  - Windows
  - Solaris



# ĐỊNH THỜI TIỄU TRÌNH

05.



## 5. Định thời tiêu trình

- Trên các hệ điều hành hiện đại có hỗ trợ tiêu trình, **tiểu trình được định thời**, không phải tiến trình.
- Có sự phân biệt giữa tiêu trình người dùng và tiêu trình hạt nhân khi định thời:
- **Tiêu trình người dùng (user thread)** được định thời thông qua các thư viện quản lý tiêu trình:
  - Phạm vi định thời là bên trong tiến trình (process-contention scope - PCS).
  - Thường được thực hiện bằng cách thiết lập độ ưu tiên (bởi người lập trình).
- **Tiêu trình hạt nhân (kernel thread)** được định thời trên tất cả các CPU khả dụng. Phạm vi định thời là toàn hệ thống (system-contention scope - SCS).



# ĐỊNH THỜI ĐA BỘ XỬ LÝ

06.



## 6. Định thời đa bộ xử lý

- Định thời CPU trở nên phức tạp hơn khi hệ thống có nhiều bộ xử lý.
- Khái niệm đa bộ xử lý có thể là một trong các dạng sau:
  - CPU có nhiều lõi vật lý (Multicore CPUs)
  - CPU có nhiều luồng xử lý trên một lõi (Multithreaded cores)
  - Hệ thống NUMA (non-uniform memory access)
  - Đa xử lý không đồng nhất (Heterogeneous multiprocessing)
- Có hai cách tiếp cận phổ biến: **đa xử lý bất đối xứng** (asymmetric multiprocessing) và **đa xử lý đối xứng** (symmetric multiprocessing - SMP).



# **ĐỊNH THỜI ĐA BỘ XỬ LÝ**

## **6.1. Đa xử lý bất đối xứng**

**06.**



## 6.1. Đa xử lý bất đối xứng

- Tất cả các thao tác lập lịch, xử lý I/O được thực hiện bởi một bộ xử lý – master server.
- Các bộ xử lý còn lại chỉ thực thi user code.
- Ưu điểm: đơn giản, chỉ một bộ xử lý truy xuất dữ liệu hệ thống, không cần chia sẻ dữ liệu.
- Nhược điểm: master server có thể bị nghẽn cổ chai (bottleneck), làm giảm hiệu năng của hệ thống.



# **ĐỊNH THỜI ĐA BỘ XỬ LÝ**

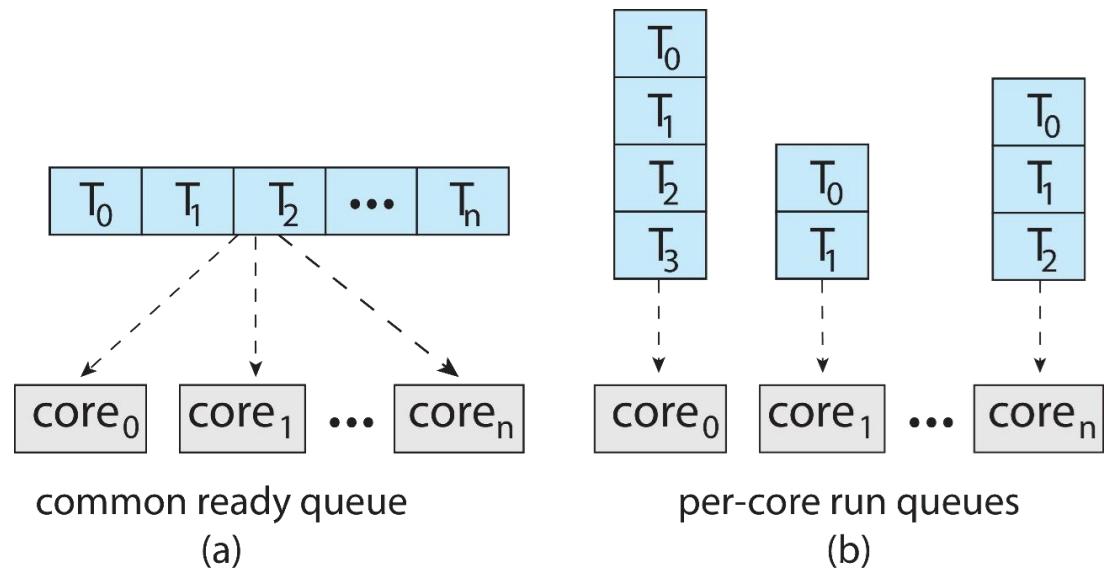
## **6.2. Đa xử lý đối xứng**

**06.**



## 6.2. Đa xử lý đối xứng

- Mỗi bộ xử lý tự định thời cho chính nó.
- Hai hướng tiếp cận để tổ chức các tiểu trình cần định thời:
  - Tất cả tiểu trình nằm trong cùng một *ready queue* (a)
  - Mỗi bộ xử lý tự tổ chức hàng đợi của riêng nó (b)





## 6.2. Đa xử lý đối xứng

- Tất cả tiểu trình nằm trong cùng một *ready queue*:
  - Tiểu trình có thể không được bộ xử lý nào chọn ?
  - Xuất hiện vùng tranh chấp: Nhiều bộ xử lý có thể chọn định thời cùng một tiểu trình => Cần có cơ chế kiểm tra và khóa (lock) việc truy xuất tiểu trình => Hiệu năng hệ thống có thể giảm do nghẽn cổ chai.
- Mỗi bộ xử lý tự tổ chức hàng đợi của riêng nó:
  - Hiệu năng không bị ảnh hưởng do các vấn đề khi dùng chung một hàng đợi => Hướng tiếp cận phổ biến trên các hệ thống SMP.
  - Vấn đề: Khối lượng công việc của các bộ xử lý khác nhau?



## 6.2. Đa xử lý đối xứng

### Cân bằng tải (Load balancing)

- Một bộ xử lý có quá nhiều tải, trong khi các bộ xử lý khác rỗi => Cần đảm bảo các bộ xử lý đều được sử dụng hiệu quả.
- Mục tiêu của cân bằng tải là phân phối khối lượng công việc (workload) đều nhau cho các CPU.
- Có hai cách cân bằng tải:
  - **Push migration:** Một tác vụ đặc biệt sẽ kiểm tra định kỳ tải của từng CPU. Nếu tình trạng quá tải xuất hiện, hệ thống sẽ di chuyển (đẩy) tác vụ từ CPU bị quá tải sang các CPU khác.
  - **Pull migration:** CPU rỗi kéo (pull) tác vụ đang chờ từ CPU bận.



## 6.2. Đa xử lý đối xứng

### Processor affinity

- Khi một tác vụ chạy trên một bộ xử lý, bộ nhớ đệm (cache) của bộ xử lý đó lưu trữ dữ liệu được truy xuất bởi tác vụ => tác vụ có tính liên kết với bộ xử lý (processor affinity).
- Cân bằng tải sẽ ảnh hưởng đến processor affinity, cụ thể là khi một tác vụ được dời sang bộ xử lý khác:
  - Cache của bộ xử lý mới phải nạp lại (repopulate)
  - Cache của bộ xử lý cũ phải được giải phóng (invalidate)

→ Phí tổn



## 6.2. Đa xử lý đối xứng

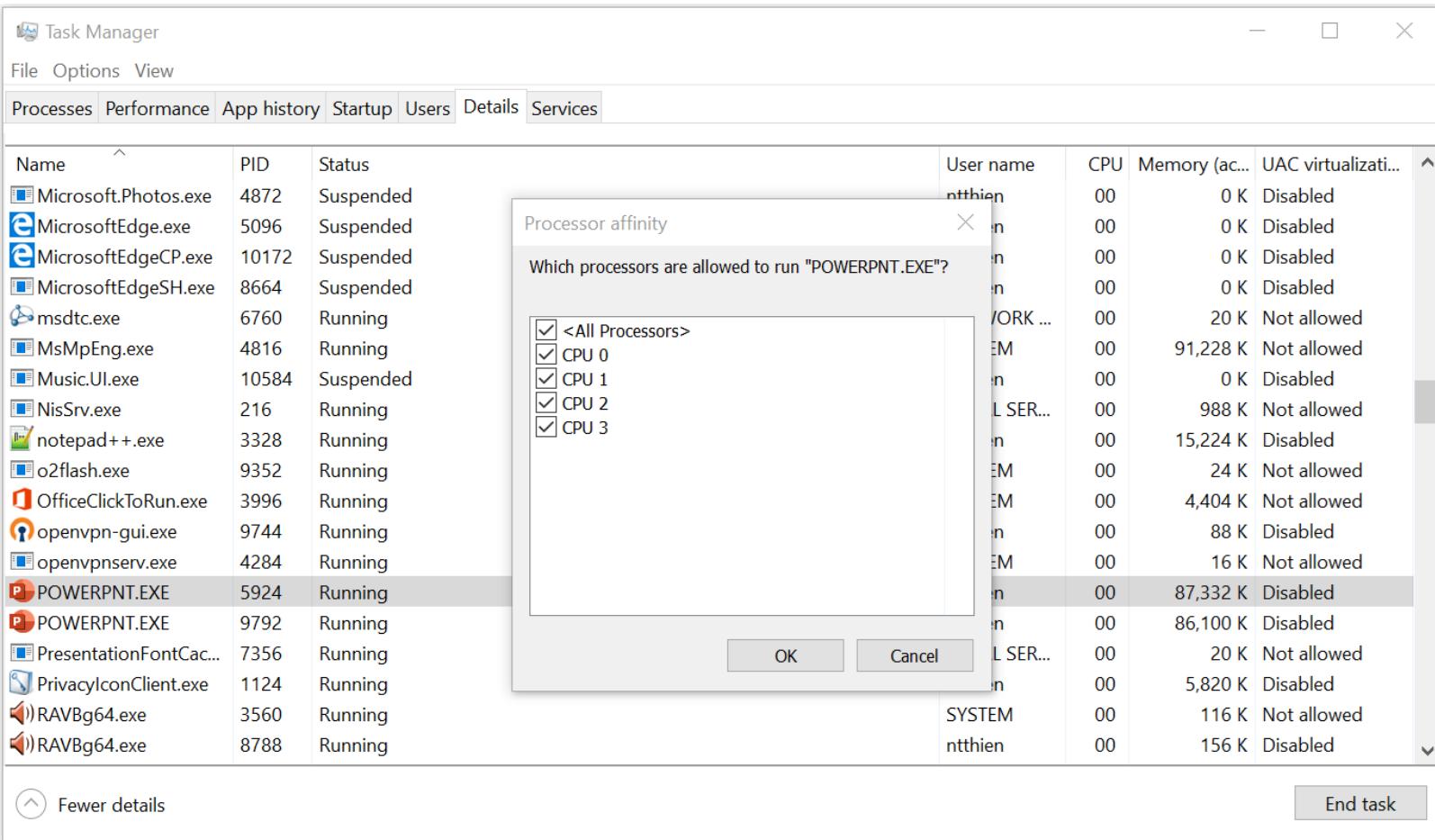
### Processor affinity

- Có 2 dạng liên kết:
  - **Liên kết mềm (Soft affinity)**: Hệ thống sẽ cố giữ tác vụ chỉ chạy trên bộ xử lý đó (nhưng không đảm bảo).
  - **Liên kết cứng (Hard affinity)**: Cho phép tiến trình chọn một tập các bộ xử lý mà nó có thể chạy trên đó.



# 6.2. Đa xử lý đối xứng

## Processor affinity





# **ĐỊNH THỜI THEO THỜI GIAN THỰC (ĐỌC THÊM)**

**07.**



## 7. Định thời theo thời gian thực

- Có nhiều thách thức do yêu cầu về tính chất thời gian thực.
- Có 2 dạng hệ thống thời gian thực:
  - **Soft real-time systems:** Các tác vụ quan trọng sẽ được cấp độ ưu tiên lớn nhất, nhưng không đảm bảo bất cứ điều gì khác.
  - **Hard real-time systems:** Tác vụ phải hoàn thành trong deadline của nó.



# 7. Định thời theo thời gian thực

## Định thời theo thời gian thực dựa trên độ ưu tiên

- Hệ thống thời gian thực phải phản hồi ngay lập tức yêu cầu CPU của một tiến trình => Bộ định thời phải hỗ trợ định thời theo độ ưu tiên với chế độ trưng dụng.
- Tiến trình có thêm một đặc trưng mới: tính chu kỳ - yêu cầu CPU trong một khoảng thời gian cố định.
- Khi một tiến trình có chu kỳ yêu cầu CPU, nó có thời gian xử lý C, thời gian deadline d (thời gian nó sẽ được phục vụ bởi CPU) và thời gian chu kỳ T.
  - $0 \leq C \leq d \leq T$

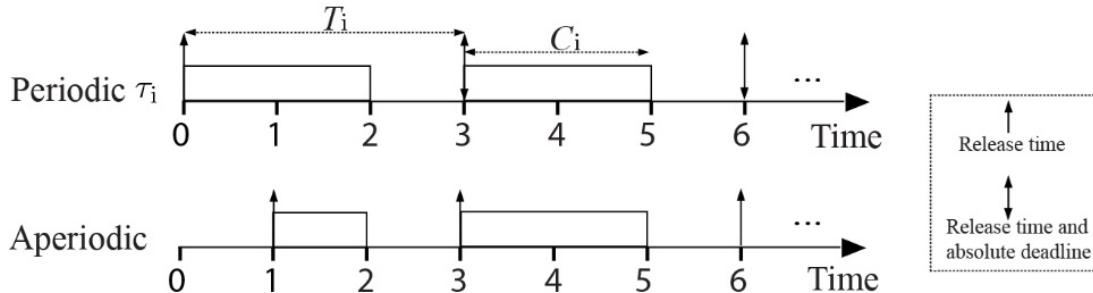


# 7. Định thời theo thời gian thực

## Mô hình tác vụ trọng thời gian thực

### Periodic tasks

- $n$  periodic tasks;
- Task:  $\tau_i = (\mu_i, T_i, C_i) (0 \leq i < n)$ 
  - $\mu_i$ : task utilization,
  - $\sum_{i=0}^{n-1} \mu_i \leq m$  (number of processors)
  - $T_i$ : task period
  - $C_i = \mu_i * T_i$ : worst-case execution time
- Job release times and absolute deadlines are predictable.



### Aperiodic tasks

- Entering times are unknown a priori
- Soft real-time aperiodic tasks
- No deadline constraints
- Responsiveness is targeted.

Self-Study

### Sporadic tasks



# 7. Định thời theo thời gian thực

## Các giải thuật định thời

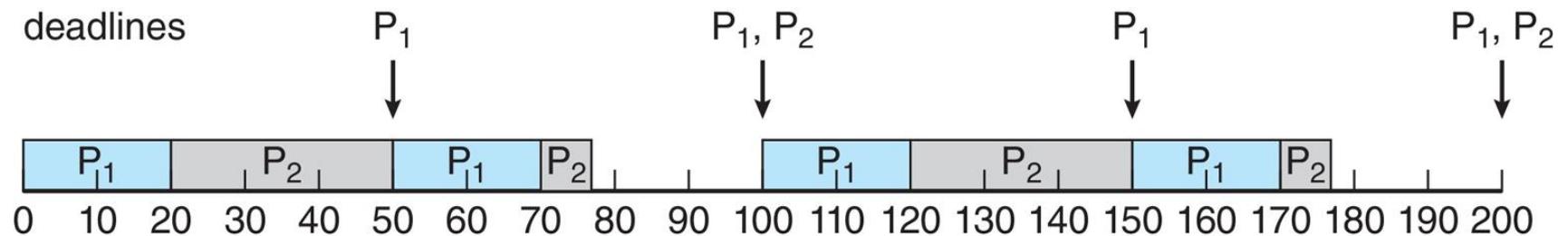
- Tác vụ chu kỳ - Periodic Tasks
  - Rate Monotonic - RM
  - Earliest Deadline First - EDF
- Tác vụ phi chu kỳ - Aperiodic Task
  - Total Bandwidth Server – TBS
  - Enhanced Virtual Release Advancing TBS
  - Constant Bandwidth Server



# 7. Định thời theo thời gian thực

## Định thời Rate Monotonic

- Độ ưu tiên được gán dựa trên nghịch đảo của chu kỳ => Chu kỳ ngắn thì độ ưu tiên cao và ngược lại.
- Tần suất của tác vụ là  $1/p$ .
- $P_1$  được gán độ ưu tiên cao hơn  $P_2$ .

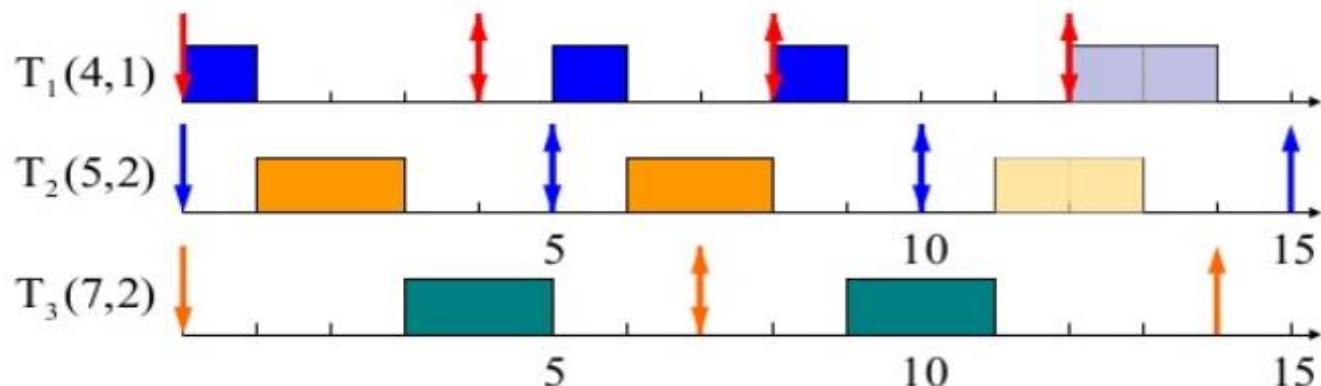




# 7. Định thời theo thời gian thực

## Định thời EDF

- Tổng hệ số sử dụng CPU (Utilization bound):  $U = \sum_{i=0}^n \mu_i \leq 1$
- Thời hạn tuyệt đối (Absolute deadline):  $d_i = r_i + T_i$
- Tác vụ có thời hạn hoàn thành nhỏ hơn sẽ có độ ưu tiên lớn hơn.

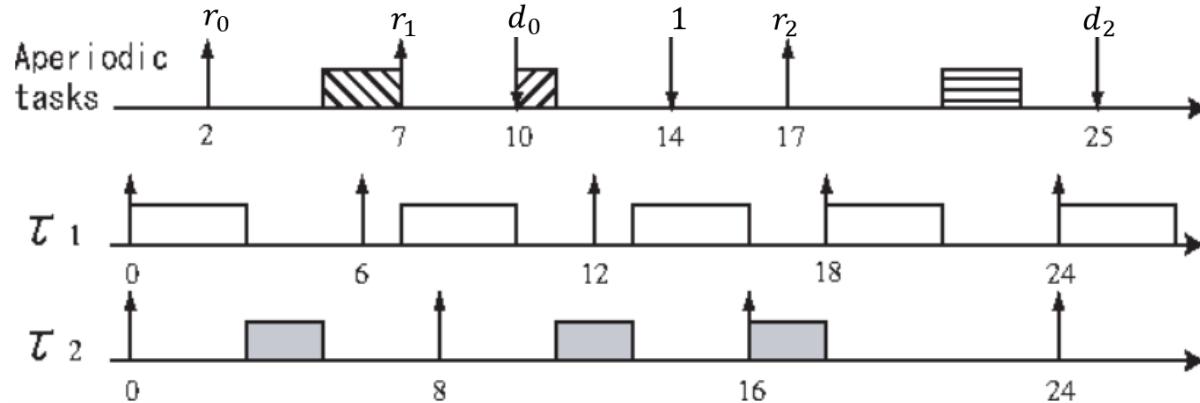




# 7. Định thời theo thời gian thực

## Định thời Total Bandwidth Server (TBS)

- Dựa trên EDF, định thời tác vụ phi chu kỳ.
- Hệ số sử dụng CPU của tác vụ có chu kỳ:  $U_p = \sum_{i=0}^{n_p} \mu_i^p \leq 1$
- Hệ số sử dụng CPU của tác vụ phi chu kỳ:  $U_a = 1 - U_p$
- Tác vụ phi chu kỳ:  $d_j = r_j + \frac{c_j^{wcet}}{U_a}$





# Định thời trên hệ thống đa vi xử lý

- Partition Scheduling
  - Partition EDF
- Semi-Partition Scheduling
  - Semi Partition Reservation
- Global Scheduling
  - Global EDF
  - PFAIR – Proportionate Fairness
  - RUN – Reduce to Uniprocessor
  - Local Assignment Algorithm (Duy D. et. al)



# ĐỊNH THỜI TRÊN MỘT SỐ HỆ ĐIỀU HÀNH

## 8.1. Định thời trên Linux

08.



## 8.1. Định thời trên Linux

- Nhân Linux 2.5 trở về trước sử dụng các phiên bản định thời UNIX tiêu chuẩn.
  - Không hỗ trợ tốt các hệ thống nhiều bộ xử lý.
  - Hiệu năng kém nếu có số lượng lớn các tiến trình trong hệ thống.
- Nhân Linux 2.5 sử dụng bộ định thời O(1):
  - Chạy với thời gian hằng số.
  - Định thời theo độ ưu tiên với chế độ trưng dụng.
  - Có hai khoảng ưu tiên: time-sharing và real-time.
  - Giá trị số nhỏ hơn biểu diễn độ ưu tiên lớn hơn.
  - Hoạt động tốt với các hệ thống SMP nhưng đáp ứng kém với các tiến trình interactive.



# 8.1. Định thời trên Linux

## Bộ định thời CFS

- Nhân Linux từ 2.6.23 sử dụng bộ định thời CFS (Completely Fair Scheduler)
  - Định thời theo lớp:
    - Mỗi lớp được gán một độ ưu tiên cụ thể.
    - Bộ định thời chọn tác vụ có độ ưu tiên cao nhất trong lớp có độ ưu tiên cao nhất.
    - Thời gian sử dụng CPU của mỗi tác vụ không dựa trên quantum time cố định mà dựa trên tỷ lệ giờ CPU.
    - Nhân Linux cài đặt sẵn 2 lớp: default và real-time. Các lớp khác có thể được thêm vào.



# 8.1. Định thời trên Linux

## Bộ định thời CFS

- Thời gian sử dụng CPU:
  - Được tính dựa trên giá trị nice được gán cho mỗi tác vụ, có giá trị từ -20 đến 19.
  - Giá trị thấp hơn có độ ưu tiên cao hơn.
  - Target latency – khoảng thời gian mà một tiến trình cần được chạy ít nhất một lần.
  - Target latency có thể tăng lên nếu số lượng tiến trình tăng lên.



# 8.1. Định thời trên Linux

## Bộ định thời CFS

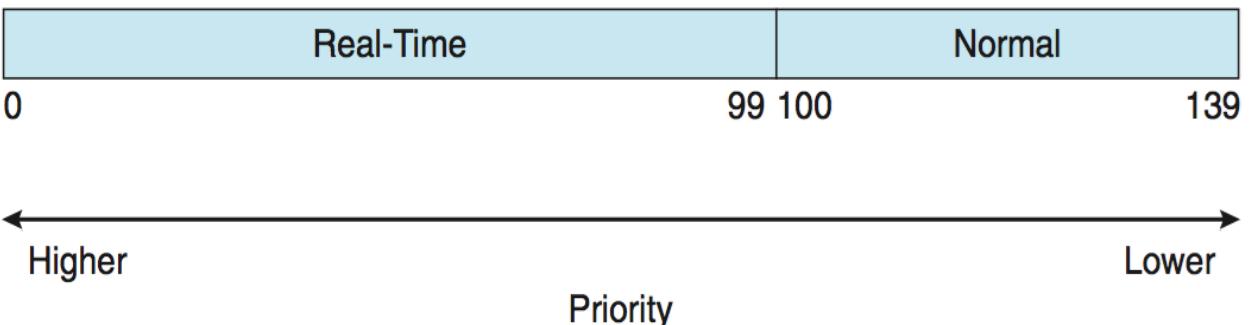
- CFS xác định tác vụ được thực thi kế tiếp qua virtual run time:
  - Mỗi tác vụ có giá trị virtual run time riêng, được kết hợp với một hệ số đặc biệt dựa trên độ ưu tiên.
  - Các tiến trình có độ ưu tiên bình thường có virtual run time tương đương với thời gian chạy thực tế.
  - Chọn tiến trình có virtual run time nhỏ nhất để thực thi tiếp.



# 8.1. Định thời trên Linux

## Bộ định thời Real-time

- Định thời real-time dựa trên tiêu chuẩn POSIX.
  - Các tác vụ real-time có độ ưu tiên tĩnh.
- Độ ưu tiên được chia thành 2 phần: real-time (từ 0 đến 99) và normal (từ 100 đến 139).
- Giá trị nice -20 tương ứng với độ ưu tiên 100, +19 tương ứng với độ ưu tiên 139.

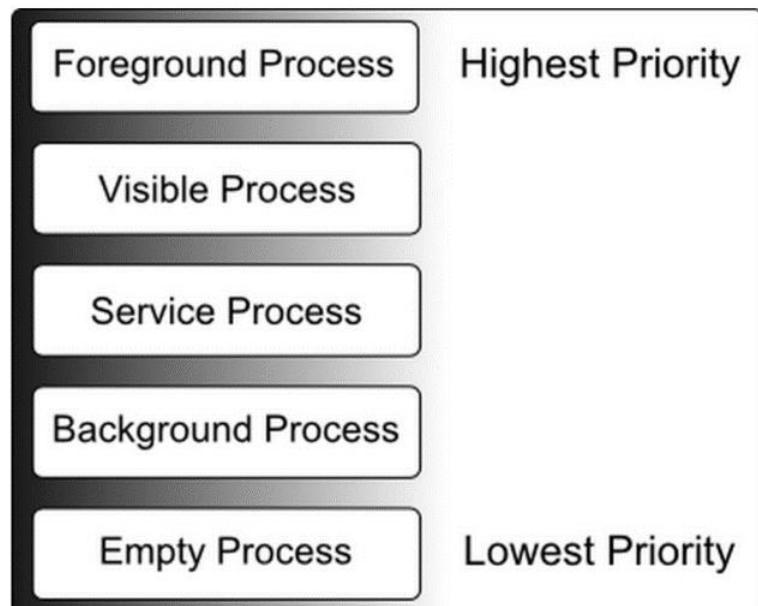




## 8.1. Định thời trên Linux

### Định thời trên Android

- Sử dụng bộ định thời của Linux.
- Độ ưu tiên được phân chia theo nhóm của các tiến trình.
- Để thu hồi tài nguyên, Android có thể hủy (kill) các tiến trình dựa trên độ ưu tiên của chúng.





## ĐỊNH THỜI TRÊN MỘT SỐ HỆ ĐIỀU HÀNH

### 8.2. Định thời trên Windows

08.



## 8.2. Định thời trên Windows

- Định thời theo độ ưu tiên với chế độ trưng dụng.
- Tác vụ có độ ưu tiên cao nhất luôn được chạy tiếp.
- Tiến trình sẽ được thực thi cho đến khi (1) block bởi system call, (2) hết quantum time, (3) bị thay thế bởi một tiến trình khác có độ ưu tiên cao hơn.
- Sử dụng 32 độ ưu tiên, được chia thành 2 lớp: variable (1-15) và real-time (16-31). Độ ưu tiên 0 dành cho quản lý bộ nhớ.
- Mỗi độ ưu tiên có hàng đợi riêng.
- Idle thread được chạy nếu không có bất cứ tác vụ nào trong hàng đợi.



## 8.2. Định thời trên Windows

- Các hàm thư viện Windows API cung cấp cho tiến trình các lớp ưu tiên sau:
  - REALTIME\_PRIORITY\_CLASS, HIGH\_PRIORITY\_CLASS, ABOVE\_NORMAL\_PRIORITY\_CLASS, NORMAL\_PRIORITY\_CLASS, BELOW\_NORMAL\_PRIORITY\_CLASS, IDLE\_PRIORITY\_CLASS.
- Tiến trình có thể có các độ ưu tiên tương đối sau:
  - TIME\_CRITICAL, HIGHEST, ABOVE\_NORMAL, **NORMAL**, BELOW\_NORMAL, LOWEST, IDLE



## 8.2. Định thời trên Windows

- Lớp ưu tiên và độ ưu tiên tương đối có thể kết hợp để xác định giá trị ưu tiên.
- Độ ưu tiên cơ sở (lúc khởi tạo) là **NORMAL** bên trong lớp.
- Khi hết quantum, độ ưu tiên có thể giảm nhưng không nhỏ hơn độ ưu tiên cơ sở.



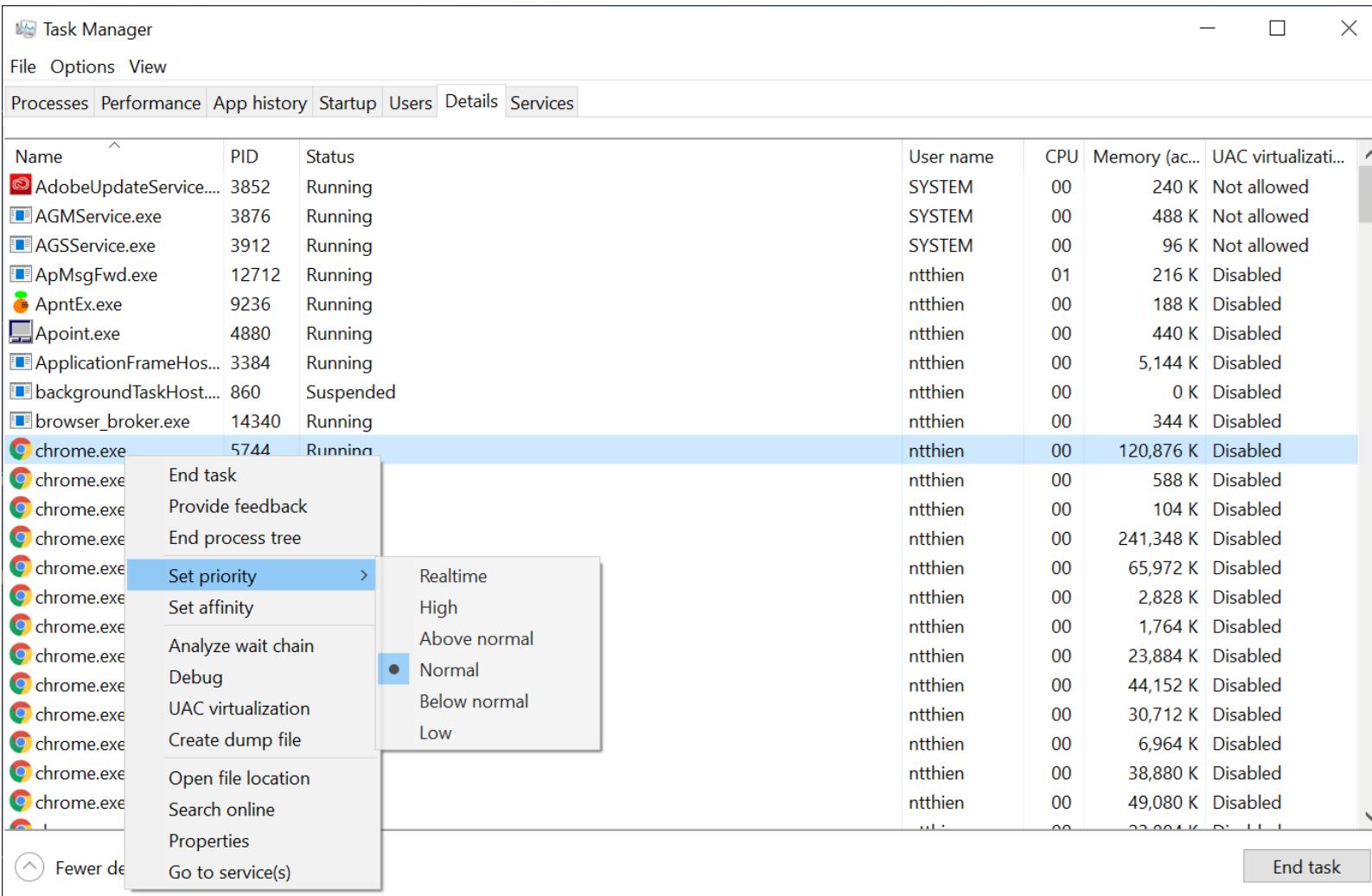
## 8.2. Định thời trên Windows

- Các độ ưu tiên trên Windows

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1



## 8.2. Định thời trên Windows





## 8.2. Định thời trên Windows

- Windows 7 có thêm user-mode scheduling (UMS):
  - Ứng dụng tạo và quản lý tiểu trình độc lập với nhau.
  - Hiệu quả hơn trong trường hợp có nhiều tiểu trình.
  - Định thời UMS được thực hiện với sự hỗ trợ của các thư viện như C++ Concurrent Runtime (ConcRT).



## ĐỊNH THỜI TRÊN MỘT SỐ HỆ ĐIỀU HÀNH

### 8.3. Định thời trên Solaris (đọc thêm)

08.



## 8.3. Định thời trên Solaris

- Định thời theo độ ưu tiên
- Có 6 lớp, mỗi lớp có độ ưu tiên khác nhau và giải thuật định thời khác nhau:
  - Time sharing (TS) – mặc định
  - Interactive (IA)
  - Real time (RT)
  - System (SYS)
  - Fair Share (FSS)
  - Fixed priority (FP)
- Lớp TS sử dụng giải thuật định thời MFQ.
- Độ ưu tiên càng lớn thì time slice càng nhỏ.



## 8.3. Định thời trên Solaris

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

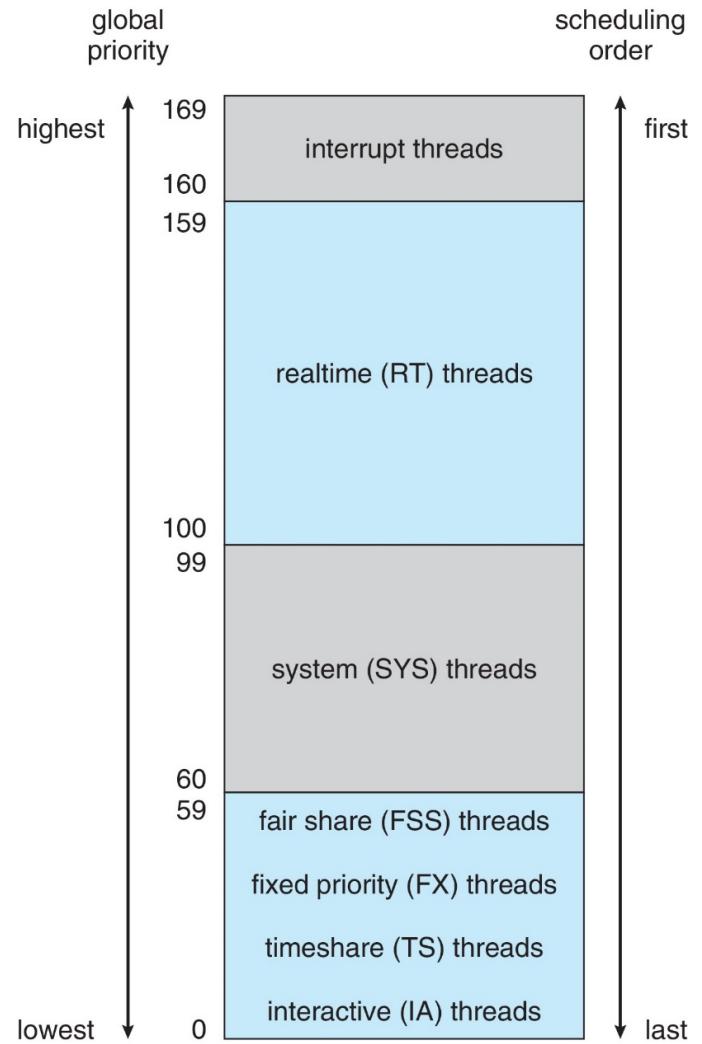


## 8.3. Định thời trên Solaris

- Bộ định thời chuyển đổi độ ưu tiên theo lớp thành độ ưu tiên toàn cục:
  - Tác vụ có độ ưu tiên cao nhất được chọn chạy tiếp.
  - Tiến trình sẽ được thực thi cho đến khi (1) block, (2) hết quantum time, (3) bị thay thế bởi một tiến trình khác có độ ưu tiên cao hơn.
  - Nếu có nhiều tiến trình có cùng độ ưu tiên, bộ định thời sẽ sử dụng hàng đợi round-robin.



## 8.3. Định thời trên Solaris





# Tóm tắt nội dung buổi học

- Định thời tiêu trình (Thread scheduling)
- Định thời đa bộ xử lý (Multiple-processor scheduling)
- Định thời theo thời gian thực (Real-time CPU scheduling)
- Định thời trên một số hệ điều hành
  - Linux
  - Windows
  - Solaris



# Câu hỏi ôn tập

- Định thời tiển trình như thế nào?
- Có các cách tiếp cận nào để thực hiện định thời đa bộ xử lý?  
Ưu nhược điểm của từng cách tiếp cận?
- Cân bằng tải là gì? Tại sao phải cân bằng tải?
- Định thời theo thời gian thực như thế nào?
- Mô tả CFS?
- Trình bày đặc điểm của bộ định thời trên Windows?

# THẢO LUẬN



sinhvienit.net