TRƯỜNG ĐẠI HỌC CÔNG NGHỆ ĐÔNG Á
EAST ASIA UNIVERSITY OF TECHNOLOGY

Chương 4

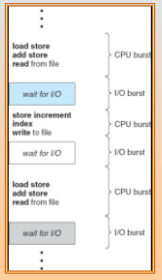
4.1 Lập lịch CPU

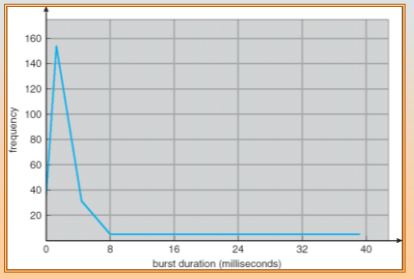
Nội dung:

- Các khái niệm cơ bản
- Các tiêu chuẩn lập lịch
- Các giải thuật lập lịch
- Lập lịch multiprocessor
- Lập lịch thời gian thực
- Lựa chọn giải thuật

4.1. Các khái niệm cơ bản

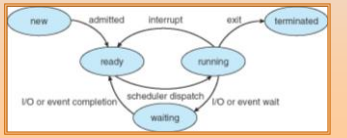
- multi-programming giúp đạt được sự sử dụng CPU tối đa
- Chu kỳ sử dụng CPU–I/O (CPU–I/O Burst Cycle): Sự thực hiện tiến trình gồm một chu kỳ thực hiện của CPU và chờ vào-ra.
- Sự phân phối sử dụng CPU giúp lựa chọn giải thuật lập lịch CPU





Trình lập lịch CPU - CPU Scheduler

- Mỗi khi CPU rỗi, HĐH cần chọn trong số các tiến trình đã sẵn sàng thực hiện trong bộ nhớ (ready queue), và phân phối CPU cho một trong số đó.
- Tiến trình được thực hiện bởi trình lập lịch ngắn kỳ (short-term scheduler, CPU scheduler)
- Các quyết định lập lịch CPU có thể xảy ra khi một tiến trình:
  1. Chuyển từ trạng thái chạy sang trạng thái chờ (vd: I/O request)
  2. Chuyển từ trạng thái chạy sang trạng thái sẵn sàng (vd: khi một ngắt xuất hiện)
  3. Chuyển từ trạng thái đợi sang trạng thái sẵn sàng (vd: I/O hoàn thành)
  4. Kết thúc



Preemptive/nonpreemptive Scheduling

- Lập lịch CPU khi 1 và 4 là *không được ưu tiên trước* (*nonpreemptive*):
  - Không có sự lựa chọn: phải chọn 1 tiến trình mới để thực hiện.
  - Khi 1 tiến trình được phân phối CPU, nó sẽ sử dụng CPU cho đến khi nó giải phóng CPU bằng cách kết thúc hoặc chuyển sang trạng thái chờ.
  - Các tiến trình sẵn sàng nhường điều khiển của CPU.
- Lập lịch khi 2 và 3 là *được ưu tiên trước* (*preemptive*)
  - Khi 2: tiến trình đá bật CPU ra. Cần phải chọn tiến trình kế tiếp.
  - Khi 3: tiến trình có thể đá bật tiến trình khác ra khỏi CPU.

Trình điều vận - Dispatcher

- Môđun trình điều vận trao quyền điều khiển của CPU cho tiến trình được lựa chọn bởi trình lập lịch CPU; các bước:
  - chuyển ngữ cảnh
  - chuyển sang user mode
  - nhảy tới vị trí thích hợp trong chương trình của người sử dụng để khởi động lại chương trình đó
- *Trễ điều vận (Dispatch latency)* – thời gian cần thiết để trình điều vận dừng một tiến trình và khởi động một tiến trình khác chạy.

4.1.2. Các tiêu chuẩn lập lịch

Max
- **CPU utilization** – giữ cho CPU càng bận càng tốt (0-100%)

Max
- **Throughput** – số tiến trình được hoàn thành trong một đơn vị thời gian

Min
- **Turnaround time** – tổng lượng thời gian để thực hiện một tiến trình: t/g chờ được đưa vào bộ nhớ + t/g chờ trong ready queue + t/g thực hiện bởi CPU + t/g thực hiện vào-ra

Min
- **Waiting time** – lượng thời gian mà một tiến trình chờ đợi ở trong ready queue

Min
- **Response time** – lượng thời gian tính từ khi có một yêu cầu được gửi đến khi có sự trả lời đầu tiên được phát ra, không phải là thời gian đưa ra kết quả của sự trả lời đó. → là tiêu chuẩn tốt.

4.1.3. Các giải thuật lập lịch

- First Come, First Served (FCFS)
- Shortest Job First (SJF)
- Priority
- Round Robin (RR)
- Multilevel Queue
- Multilevel Feedback-Queue

1) Giải thuật First-Come, First-Served (FCFS)

- Tiến trình nào yêu cầu CPU trước sẽ được phân phối CPU trước
  → Giải thuật FCFS là không được ưu tiên trước.
- Là giải thuật đơn giản nhất

Process	Burst Time (thời gian sử dụng CPU, ms)
P_1	24
P_2	3
P_3	3

- Giả định rằng các tiến trình đến theo thứ tự: P_1, P_2, P_3 thì biểu đồ Gantt (Gantt Chart) của lịch biểu như sau:

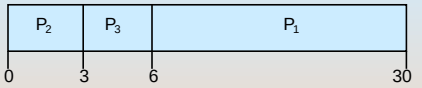
Thời gian chờ đợi của các tiến trình: $P_1 = 0$; $P_2 = 24$; $P_3 = 27$

- Thời gian chờ đợi trung bình: (0 + 24 + 27)/3 = 17

Giải thuật FCFS (tiếp)

Giả định rằng các tiến trình đến theo thứ tự P_2 , P_3 , P_1

- Biểu đồ Gantt của lịch biểu như sau:



- Thời gian chờ đợi của các tiến trình: $P_1 = 6$; $P_2 = 0$;, $P_3 = 3$
- Thời gian chờ đợi trung bình: (6 + 0 + 3)/3 = 3
- Tốt hơn nhiều so với trường hợp trước
- *Convoy effect* (hiệu ứng hộ tống): tiến trình ngắn đứng sau tiến trình dài, như là các xe máy đi sau xe buýt vậy.

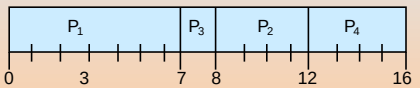
2) Giải thuật Shortest-Job-First (SJF)

- Gắn với mỗi tiến trình là thời gian sử dụng CPU *tiếp sau* của nó. Thời gian này được sử dụng để lập lịch các tiến trình với thời gian ngắn nhất.
- Hai phương pháp:
  - không ưu tiên trước (non-preemptive) – một tiến trình nếu sử dụng CPU thì không nhường cho tiến trình khác cho đến khi nó kết thúc.
  - có ưu tiên trước (preemptive) – nếu một tiến trình đến có thời gian sử dụng CPU *ngắn hơn* thời gian còn lại của tiến trình đang thực hiện thì ưu tiên tiến trình mới đến trước. Phương pháp này còn được gọi là Shortest-Remaining-Time-First (SRTF).
- SJF là tối ưu – cho thời gian chờ đợi trung bình của các tiến trình là nhỏ nhất.

Ví dụ SJF không ưu tiên trước

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SJF (non-preemptive)

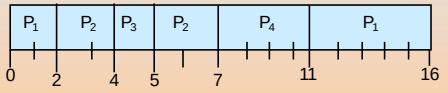


- Thời gian chờ đợi của các tiến trình: $P_1 = 0$; $P_2 = 6$;, $P_3 = 3$, $P_4 = 7$
- Thời gian chờ đợi trung bình = (0 + 6 + 3 + 7)/4 = 4

Ví dụ SJF có ưu tiên trước

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SJF (preemptive)



- Thời gian chờ đợi trung bình = (9 + 1 + 0 +2)/4 = 3

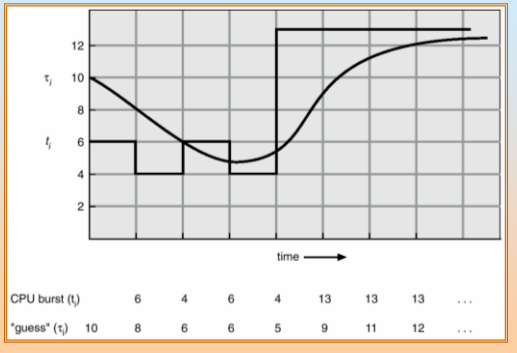
Xác định thời gian sử dụng CPU tiếp sau

- Không thể biết chính xác thời gian sử dụng CPU tiếp sau của tiến trình nhưng có thể đoán giá trị xấp xỉ của nó dựa vào thời gian sử dụng CPU trước đó và sử dụng công thức đệ quy:

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n.$$

  - τ_{n+1} = giá trị dự đoán cho thời gian sử dụng CPU tiếp sau
  - t_n = thời gian thực tế của sự sử dụng CPU thứ n
  - α, $0 \le \alpha \le 1$
  - τ_0 là một hằng số
- $\alpha = 0$: $\tau_{n+1} = \tau_n = \tau_0$.
  - Thời gian thực tế sử dụng CPU gần đây không có tác dụng gì cả.
- $\alpha = 1$: $\tau_{n+1} = t_n$.
  - Chỉ tính đến thời gian sử dụng CPU thực tế ngay trước đó.

Minh họa khi $\alpha = 1/2$ và $\tau_0 = 10$



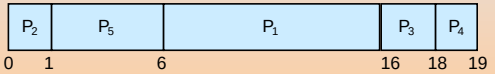
3) Lập lịch theo mức ưu tiên

- Mỗi tiến trình được gắn một số ưu tiên (số nguyên). VD: 0-127
- CPU được phân phối cho tiến trình có mức ưu tiên cao nhất (có số ưu tiên nhỏ nhất)
  - Preemptive
  - nonpreemptive
- SJF là trường hợp riêng của lập lịch theo mức ưu tiên: mức ưu tiên chính là thời gian sử dụng CPU tiếp sau dự đoán được.
- Vấn đề gặp phải là: những tiến trình có mức ưu tiên thấp có thể không bao giờ được thực hiện (starvation).
- Giải pháp ≡ Aging: kỹ thuật tăng mức ưu tiên của các tiến trình chờ đợi lâu trong hệ thống.
  - VD: Sau 1-15 phút giảm số ưu tiên một lần.

Ví dụ lập lịch theo mức ưu tiên

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Nonpreemptive:



- T/gian chờ đợi trung bình = (0 + 1 + 6 + 16 + 18)/5 = 8.2

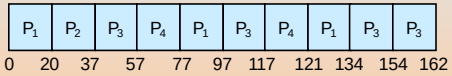
4) Giải thuật Round-Robin (RR)

- Mỗi tiến trình sử dụng một lượng nhỏ thời gian của CPU (*time quantum – định lượng thời gian, q*), thường là 10-100 ms. Sau đó nó được ưu tiên đưa vào cuối của ready queue.
- Ready queue được tổ chức dạng FIFO (FCFS)
- Nếu tiến trình có thời gian sử dụng CPU < $q \Rightarrow$ Tiến trình sẽ tự nguyện nhường CPU khi kết thúc. Trình lập lịch sẽ chọn tiến trình kế tiếp trong ready queue.
- Nếu tiến trình có thời gian sử dụng CPU > $q \Rightarrow$ bộ định thời (timer) sẽ đếm lùi và gây ngắt HĐH khi nó = 0. Việc chuyển ngữ cảnh được thực hiện và tiến trình hiện tại được đưa xuống cuối ready queue để nhường CPU cho tiến trình kế tiếp.

Ví dụ lập lịch RR với q = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

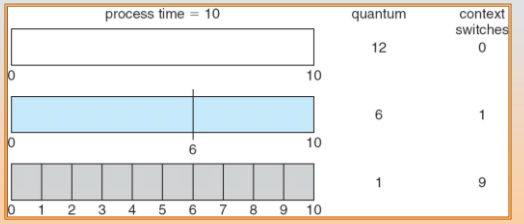
- Biểu đồ Gantt:



- T/g chờ đợi TB = ((57+24)+20+(37+40+17)+(57+40))/4 = 73
- Thường thì *RR* có turnaround trung bình cao hơn SJF, nhưng có *response* tốt hơn (thấp hơn).

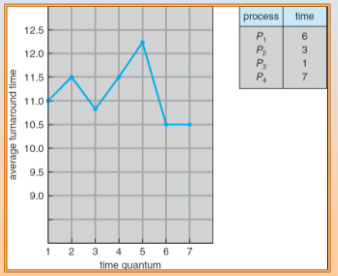
Quan hệ giữa q và hiệu năng

- Nếu q lớn $\Rightarrow$ tương tự như FCFS.
- Nếu q nhỏ $\Rightarrow$ số lần chuyển ngữ cảnh càng nhiều, làm giảm hiệu năng.

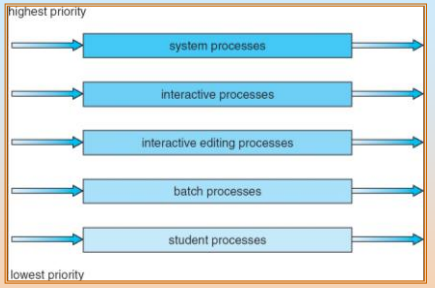


Phụ thuộc của Turnaround Time vào q

- Turnaround Time trung bình của một tập tiến trình không phải luôn được cải thiện khi q tăng lên.
- Luật: 80% các CPU burst nên nhỏ hơn q.



Multilevel Queue Scheduling



➢ Tiến trình trong queue có mức ưu tiên thấp hơn chỉ có thể chạy khi các queue có mức ưu tiên cao hơn rỗng.

➢ Tiến trình có mức ưu tiên cao hơn khi vào ready queue không ảnh hưởng đến tiến trình đang chạy có mức ưu tiên thấp hơn.

5) Lập lịch đa mức hàng đợi

- Ready queue được chia thành nhiều queue riêng biệt:
  foreground (chứa các *interactive process*)
  background (chứa các *batch process*)
- Mỗi hàng đợi có giải thuật lập lịch riêng:
  - foreground – RR
  - background – FCFS
- Phải có sự lập lịch giữa các queue:
  - Lập lịch với mức ưu tiên cố định; vd: phục vụ tất cả tiến trình từ foreground, tiếp theo từ background (có thể xảy ra starvation).
  - Phân chia thời gian: mỗi queue nhận được một lượng thời gian CPU nào đó mà nó có thể lập lịch các tiến trình của nó.
    - vd: 80% cho foreground và 20% cho background queue

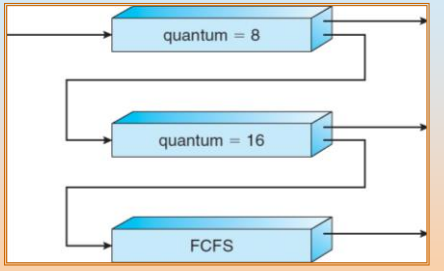
6) Lập lịch đa mức hàng đợi có hoàn ngược

- Một tiến trình có thể di chuyển giữa các queue khác nhau
- Trình lập lịch đa mức hàng đợi có hoàn ngược được xác định bởi các tham số sau:
  - số lượng queue
  - giải thuật lập lịch cho mỗi queue
  - phương pháp được sử dụng để xác định khi nào thì tăng/giảm mức ưu tiên của một tiến trình
  - phương pháp được sử dụng để xác định queue nào mà tiến trình sẽ đến khi nó cần được phục vụ.

Ví dụ Multilevel Feedback Queue

- Ba queue:
  - Q_0 – thời gian định lượng 8 ms
  - Q_1 – thời gian định lượng 16 ms
  - Q_2 – FCFS
- Lập lịch:
  - Một tiến trình vào queue Q_0 và được phục vụ FCFS. Khi nó giành được CPU, tiến trình nhận được 8 ms. Nếu nó không hoàn thành trong 8 ms, tiến trình được chuyển tới queue Q_1.
  - Tại Q_1 tiến trình tiếp tục được phục vụ FCFS với 16 ms nữa. Nếu nó vẫn chưa hoàn thành thì nó được ưu tiên và được chuyển đến queue Q_2.

Multilevel Feedback Queues



4.1.4. Lập lịch multiprocessor

- Lập lịch CPU khi có nhiều processor phức tạp hơn nhiều
- Các loại processor trong multiprocessor
  - *Đồng nhất (Homogeneous):* tất cả có cùng kiến trúc.
  - *Không đồng nhất (Heterogeneous)*: một số tiến trình có thể không tương thích với kiến trúc của các CPU.
- *Cân bằng tải (Load balancing/sharing)*: một ready queue cho tất cả các processor, CPU nhàn rỗi được gán cho tiến trình ở đầu queue.
- *Đa xử lý không đối xứng - Asymmetric multiprocessing:*
  - chỉ một processor (master processor) truy nhập các cấu trúc dữ liệu hệ thống, làm giảm sự cần thiết bảo vệ dữ liệu chia sẻ.

5.1.5. Lập lịch thời gian thực

- *Hard real-time* systems – yêu cầu hoàn thành một tác vụ găng (critical task) trong thời gian được đảm bảo.
  - **resource reservation**: khi tiến trình được gửi đến cùng với lệnh cho biết thời gian cần thiết của nó, trình lập lịch có thể chấp nhận và đảm bảo nó sẽ kết thúc đúng hạn, hoặc từ chối tiến trình.
- *Soft real-time* computing – yêu cầu các tiến trình găng nhận mức ưu tiên trên các tiến trình kém may mắn hơn.
  - có thể phân phối tài nguyên không hợp lý, thời gian trễ lâu, starvation.

  → phải cẩn thận trong thiết kế trình lập lịch và các khía cạnh liên quan của HĐH:
    - lập lịch có ưu tiên, các tiến trình thời gian thực có mức ưu tiên cao nhất.
    - trễ điều vận (dispatch latency) phải nhỏ.

4.1.6. Lập lịch luồng

- Lập lịch cục bộ (Local Scheduling):
  - Bằng cách nào Thư viện luồng quyết định chọn luồng nào để đặt vào một CPU ảo khả dụng:
    - Thường chọn luồng có mức ưu tiên cao nhất
  - Sự cạnh tranh CPU diễn ra giữa các luồng của cùng một tiến trình.
  - Trong các HĐH sử dụng mô hình Many-to-one, Many-to-many.
- Lập lịch toàn cục (Global Scheduling)
  - Bằng cách nào kernel quyết định kernel thread nào để lập lịch CPU chạy tiếp.
  - Sự cạnh tranh CPU diễn ra giữa tất cả các luồng trong hệ thống.
  - Trong các HĐH sử dụng mô hình One-to-one (Windows XP, Linux, Solaris 9)

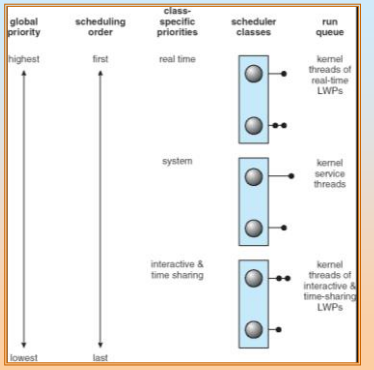
4.1.7. Các ví dụ lập lịch trong HĐH

- Solaris scheduling
- Windows XP scheduling
- Linux scheduling

1) Solaris 2 Scheduling

- Lập lịch dựa trên mức ưu tiên
- Xác định 4 lớp lập lịch có thứ tự ưu tiên như sau
  1. Real time
  2. System
  3. Time sharing
  4. Interactive
- Trong mỗi lớp có các mức ưu tiên khác nhau và các giải thuật lập lịch khác nhau.

Solaris 2 Scheduling (tiếp)



Lớp cho các luồng tương tác và chia sẻ thời gian

- Lớp lập lịch mặc định cho tiến trình là Time sharing. Chính sách lập lịch tự động thay đổi mức ưu tiên và gán các q khác nhau nhờ dùng một multilevel feedback queue.
- Mặc định: Quan hệ tỷ lệ nghịch: mức ưu tiên càng cao, q càng nhỏ và ngược lại.
- Các tiến trình tương tác có mức ưu tiên cao hơn các tiến trình CPU-bound
  - Thời gian đáp ứng tốt cho các tiến trình tương tác
  - Thông lượng tốt cho các tiến trình CPU-bound

Bảng điều vận của Solaris cho các luồng tương tác và chia sẻ thời gian

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

Giải thích

- **Priority**: mức ưu tiên phụ thuộc lớp của các lớp tương tác và chia sẻ thời gian.
  - Số càng lớn mức ưu tiên càng cao.
- **Time quantum**: lượng thời gian cho mức ưu tiên tương ứng.
  - Tỷ lệ nghịch với mức ưu tiên
- **Time quantum expired**: mức ưu tiên mới của một luồng mà đã sử dụng hết lượng thời gian của nó mà chưa kết thúc.
  - Thấp hơn mức ưu tiên gốc
- **Return from sleep**: mức ưu tiên của một luồng đang trở về từ trạng thái ngủ (vd đợi vào/ra).
  - Cao hơn mức ưu tiên gốc, giúp giảm thời gian đáp ứng cho các tiến trình tương tác.

Lớp cho các luồng thời gian thực và luồng hệ thống

- Lớp hệ thống
  - Để chạy các tiến trình kernel, ví dụ như trình lập lịch, trình quản lý phân trang bộ nhớ.
  - Các tiến trình người sử dụng chạy trong kernel mode không ở trong lớp hệ thống.
  - Mức ưu tiên của các tiến trình hệ thống là không thay đổi.
- Lớp thời gian thực
  - Chứa các luồng có mức ưu tiên cao nhất ⇒ tiến trình thời gian thực sẽ được chạy trước tiến trình thuộc các lớp khác
  - Ít tiến trình thuộc lớp này.

Solaris 9 scheduling

- Có thêm 2 lớp lập lịch mới cho các luồng:
- Fixed priority:
  - Dải mức ưu tiên giống như lớp chia sẻ thời gian
  - Khác ở chỗ mức ưu tiên không được điều chỉnh tự động
- Fair share:
  - Sử dụng sự chia sẻ CPU thay vì dùng mức ưu tiên

Chọn luồng tổng thể?

- Mỗi lớp có tập mức ưu tiên riêng (cục bộ).
- Trình lập lịch
  - chuyển đổi mức ưu tiên cục bộ thành mức ưu tiên toàn cục
  - chọn luồng có mức ưu tiên toàn cục lớn nhất để chạy.
- Luồng được chọn sẽ dùng CPU cho đến khi:
  - Nó chuyển trạng thái khóa (vd để vào/ra)
  - Nó sử dụng hết lượng thời gian q
  - Bị một luồng có mức ưu tiên cao hơn giành quyền
- Khi có nhiều luồng có cùng mức ưu tiên, trình lập lịch sử dụng một hàng đợi round-robin.

2) Windows XP Scheduling

- Sử dụng giải thuật lập lịch có ưu tiên trước dựa trên mức ưu tiên.
- Chịu trách nhiệm lập lịch luồng: trình điều vận (dispatcher)
- Một luồng được chọn sẽ chạy cho đến khi:
  - Nó gọi system call khóa (vd để vào/ra)
  - Nó sử dụng hết lượng thời gian q
  - Bị một luồng có mức ưu tiên cao hơn giành quyền
- Trình điều vận sử dụng dải ưu tiên 32 mức, chia làm 2 lớp:
  - **Variable class**: chứa các luồng có mức từ 1 đến 15
  - **Real-time class**: chứa các luồng có mức từ 16 đến 31.
- Mỗi mức ưu tiên có 1 queue. Trình điều vận duyệt qua các queue từ cao nhất đến thấp nhất cho đến khi tìm thấy 1 luồng sẵn sàng. Nếu không có luồng như vậy, trình điều vận thực hiện luồng đặc biệt – **idle thread**.

Quan hệ mức ưu tiên của kernel và Win32 API

- Win32 API định nghĩa các lớp ưu tiên:
  - REALTIME_PRIORITY_CLASS
  - HIGH_PRIORITY_CLASS
  - ABOVE_NORMAL_PRIORITY_CLASS
  - NORMAL_PRIORITY_CLASS
  - BELOW_NORMAL_PRIORITY_CLASS
  - IDLE_PRIORITY_CLASS

  Mức ưu tiên của một luồng thuộc một trong những lớp này là có thể thay đổi
- Trong mỗi lớp ưu tiên gồm các giá trị ưu tiên:
  - TIME_CRITICAL
  - HIGHEST
  - ABOVE_NORMAL
  - NORMAL
  - BELOW_NORMAL
  - LOWEST
  - IDLE

Quan hệ mức ưu tiên của kernel và Win32 API

	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

Base Priority →

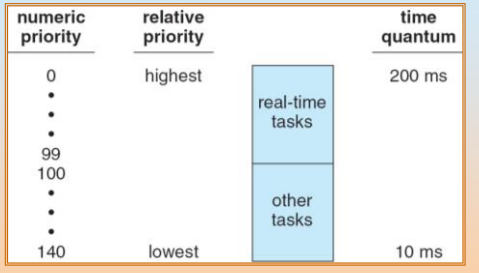
Chiến lược tăng, giảm mức ưu tiên

- Khi lượng thời gian q của luồng hết, luồng bị ngắt. Nếu luồng thuộc lớp có thể thay đổi mức ưu tiên thì mức của nó bị giảm. Tuy nhiên không giảm đến dưới mức ưu tiên cơ sở.
  ⇒ có xu hướng giới hạn sử dụng CPU của các luồng CPU-bound
- Khi luồng thuộc lớp có thể thay đổi mức ưu tiên được giải phóng khỏi hoạt động đợi, trình điều vận tăng mức ưu tiên cho nó. Lượng tăng phụ thuộc luồng đã đợi cái gì, ví dụ:
  - Đợi vào/ra bàn phím ⇒ tăng mạnh
  - Đợi hoạt động đĩa ⇒ tăng vừa
- Chiến lược này có xu hướng cho thời gian đáp ứng tốt với các luồng tương tác sử dụng chuột và cửa sổ, cho phép các luồng I/O-bound giữ các thiết bị vào/ra "bận rộn", và các luồng CPU-bound sử dụng các chu kỳ CPU dư thừa trong "lặng lẽ".
- Chiến lược này một số HĐH chia sẻ thời gian sử dụng, vd UNIX.
- Ngoài ra, cửa sổ mà user đang tương tác được nâng mức ưu tiên.

3) Linux Scheduling

- Linux kernel từ phiên bản 2.5,
  - Cung cấp giải thuật lập lịch chạy trong thời gian hằng - *O*(1) – bất kể số lượng luồng trong hệ thống.
  - Hỗ trợ tốt hơn cho các hệ thống đa xử lý đối xứng (SMP)
  - Cung cấp sự công bằng và hỗ trợ cho các tác vụ (task) tương tác
- Trình lập lịch Linux dùng giải thuật có ưu tiên trước dựa trên mức ưu tiên với 2 dải riêng:
  - Dải **real-time**: có mức từ 0 đến 99
  - Dải **nice**: có mức từ 100 đến 140.
- Số càng nhỏ, mức ưu tiên càng cao
- Không giống với nhiều HĐH khác, Linux gán các task có mức ưu tiên càng cao càng được nhiều lượng thời gian q.

Quan hệ giữa Priorities và Time-slice length



Linux Scheduling (tiếp)

- Kernel duy trì danh sách các task sẵn sàng trong một **runqueue**
  - Nếu có nhiều bộ xử lý, mỗi BXL có 1 runqueue và lập lịch độc lập
- Mỗi runqueue có 2 mảng ưu tiên:
  - **active**: chứa tất cả task vẫn còn trong lượng thời gian của chúng
  - **expired**: chứa tất cả task đã hết lượng thời gian
- Trình lập lịch chọn task có mức ưu tiên cao nhất trong mảng active. Khi mảng active rỗng, 2 mảng sẽ tráo đổi cho nhau.
- Các real-time task được gán mức ưu tiên tĩnh
- Các task khác có mức ưu tiên động: = nice value ± 5
  - Task có thời gian ngủ đợi vào/ra dài hơn thường là tương tác hơn
    ⇒ tăng mức ưu tiên bằng cách -5
  - Task có thời gian ngủ ngắn hơn thường là CPU-bound
    ⇒ giảm mức ưu tiên bằng cách +5

4.1.8. Lựa chọn giải thuật

- Chọn giải thuật lập lịch CPU nào cho hệ thống cụ thể?
- Trước tiên, xác định sử dụng tiêu chuẩn nào? Vd:
  - Tối đa CPU utilization với ràng buộc response time lớn nhất là 1s
  - Tối đa throughput để turnaround time là tỷ lệ tuyến tính với thời gian thực hiện

1. Phân tích hiệu năng của từng giải thuật đối với các tiến trình
2. Sử dụng chuẩn hàng đợi: công thức Little: $n = \lambda \times W$
   - n: độ dài queue trung bình
   - W: thời gian chờ đợi trung bình trong queue
   - λ: tốc độ đến queue của tiến trình (số tiến trình/giây)
3. Mô phỏng: lập trình mô hình hệ thống để đánh giá
4. Thực hiện: đặt giải thuật cụ thể trong hệ thống thực để đánh giá

BÀI TẬP