Chương IV: Định thời CPU

- Khái niệm cơ bản
- Các bộ định thời
  - long-term, mid-term, short-term
- Các tiêu chuẩn định thời CPU
- Các giải thuật định thời
  - First-Come, First-Served (FCFS)
  - Shortest Job First (SJF) và Shortest Remaining Time First (SRTF)
  - Priority Scheduling
  - Round-Robin (RR)
  - Highest Response Ratio Next (HRRN)
  - Multilevel Queue
  - Multilevel Feedback Queue

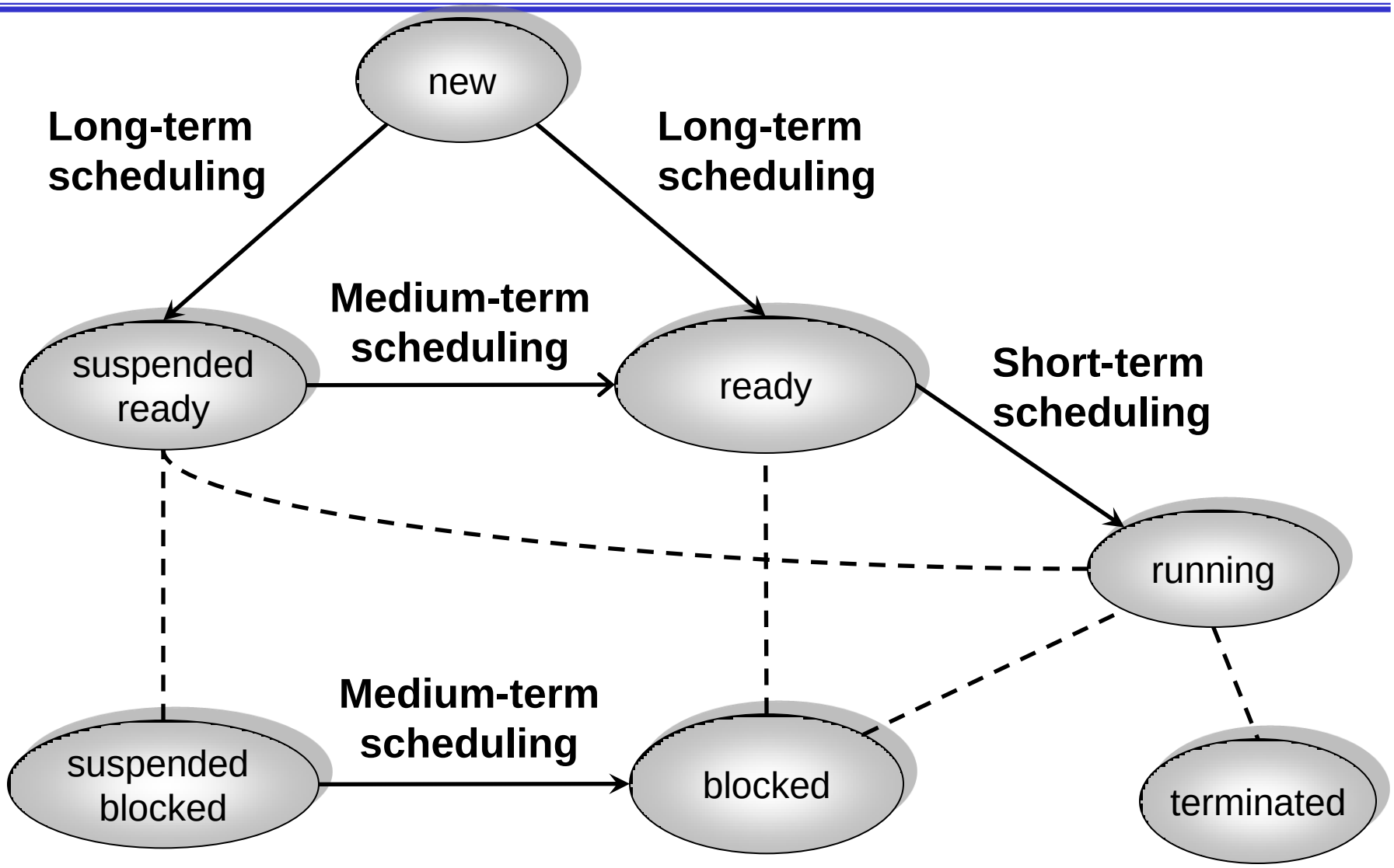
Mục tiêu

- Hiểu được:
  - Tại sao phải định thời.
  - Các tiêu chí định thời
  - Một số giải thuật định thời

Khái niệm cơ bản

- Trong các hệ thống multitasking
  - Thực thi nhiều chương trình đồng thời làm tăng hiệu suất hệ thống.
  - Tại mỗi thời điểm, chỉ có một process được thực thi. Do đó, cần phải giải quyết vấn đề phân chia, lựa chọn process thực thi sao cho được hiệu quả nhất → chiến lược định thời CPU.
- *Định thời CPU*
  - Chọn một process (từ ready queue) thực thi.
  - Với một multithreaded kernel, việc định thời CPU là do OS chọn kernel thread được chiếm CPU.

Các bộ định thời



Các bộ định thời

- *Long-term scheduling*
  - Xác định chương trình nào được chấp nhận nạp vào hệ thống để thực thi
  - Điều khiển mức độ multiprogramming của hệ thống
  - Long term scheduler thường cố gắng duy trì xen lẫn CPU-bound và I/O-bound process
- *Medium-term scheduling*
  - Process nào được đưa vào (swap in), đưa ra khỏi (swap out) bộ nhớ chính
  - Được thực hiện bởi phần quản lý bộ nhớ và được thảo luận ở phần quản lý bộ nhớ.

Các bộ định thời (tt)

Short term scheduling

- Xác định process nào trong ready queue sẽ được chiếm CPU để thực thi kế tiếp (còn được gọi là định thời CPU, CPU scheduling)
- Short term scheduler còn được gọi với tên khác là *dispatcher*
- Bộ định thời short-term được gọi mỗi khi có một trong các sự kiện/interrupt sau xảy ra:
  - t thời gian (clock interrupt)
  - Ngắt ngoại vi (I/O interrupt)
  - Lời gọi hệ thống (operating system call)

Chương này sẽ tập trung vào định thời ngắn hạn

Dispatcher

- Dispatcher sẽ chuyển quyền điều khiển CPU về cho process được chọn bởi bộ định thời ngắn hạn
- Bao gồm:
  - Chuyển ngữ cảnh (sử dụng thông tin ngữ cảnh trong PCB)
  - Chuyển chế độ người dùng
  - Nhảy đến vị trí thích hợp trong chương trình ứng dụng để khởi động lại chương trình (chính là program counter trong PCB)
- Công việc này gây ra phí tổn
  - *Dispatch latency*: thời gian mà dispatcher dừng một process và khởi động một process khác

Các tiêu chuẩn định thời CPU

- User-oriented
  - *Thời gian đáp ứng (Response time):* khoảng thời gian process nhận yêu cầu đến khi yêu cầu đầu tiên được đáp ứng (time-sharing, interactive system) → cực tiểu
  - *Thời gian quay vòng (hoàn thành) (Turnaround time):* khoảng thời gian từ lúc một process được nạp vào hệ thống đến khi process đó kết thúc → cực tiểu
  - *Thời gian chờ (Waiting time):* tổng thời gian một process đợi trong ready queue → cực tiểu
- System-oriented
  - *Sử dụng CPU (processor utilization):* định thời sao cho CPU càng bận càng tốt → cực đại
  - *Công bằng (fairness):* tất cả process phải được đối xử như nhau
  - *Thông lượng (throughput):* số process hoàn tất công việc trong một đơn vị thời gian → cực đại.

Hai yếu tố của giải thuật định thời

- *Hàm chọn lựa* (selection function): dùng để chọn process nào trong ready queue được thực thi (thường dựa trên độ ưu tiên, yêu cầu về tài nguyên, đặc điểm thực thi của process,…), ví dụ
  - w = tổng thời gian đợi trong hệ thống
  - e = thời gian đã được phục vụ
  - s = tổng thời gian thực thi của process (bao gồm cả “e”)

Hai yếu tố của giải thuật định thời (tt)

- *Chế độ quyết định* (decision mode): chọn thời điểm thực hiện hàm chọn lựa để định thời. Có hai chế độ
  - Không trưng dụng (Non-preemptive)
    - Khi ở trạng thái running, process sẽ thực thi cho đến khi kết thúc hoặc bị blocked do yêu cầu I/O
  - Trưng dụng (Preemptive)
    - Process đang thực thi (trạng thái running) có thể bị ngắt nửa chừng và chuyển về trạng thái ready bởi hệ điều hành
    - Chi phí cao hơn non-preemptive nhưng đánh đổi lại bằng thời gian đáp ứng tốt hơn vì không có trường hợp một process độc chiếm CPU quá lâu.

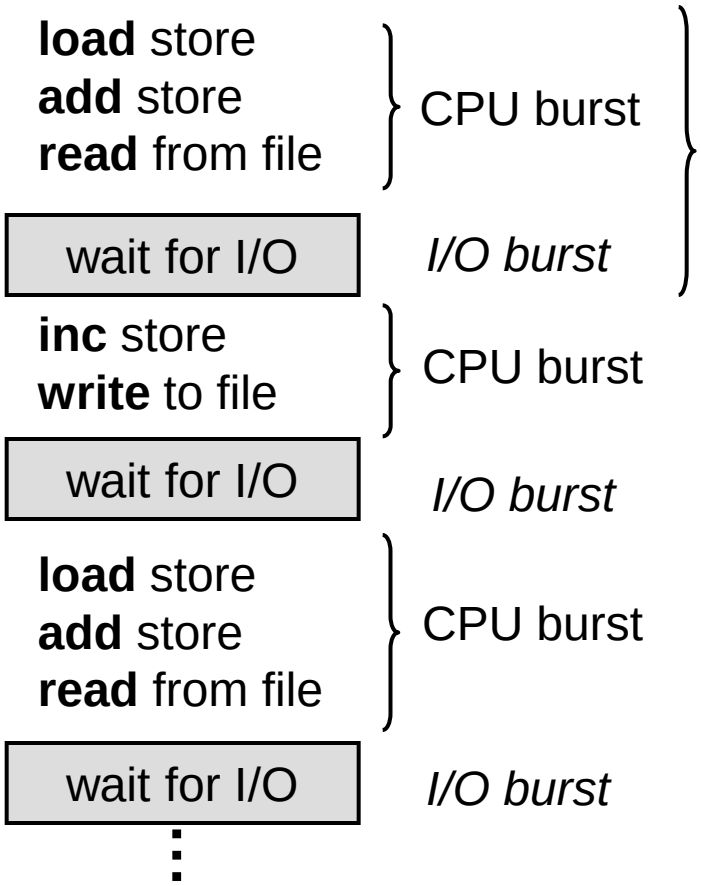
Preemptive và Non-preemptive

- Hàm định thời được thực hiện khi
  - ( ) n từ ng i running sang waiting
  - ( ) n từ ng i running sang ready
  - ( ) n từ ng i waiting, new sang ready
  - ( ) t c c thi

  1 và 4 không cần lựa chọn loại định thời biểu, 2 và 3 cần
- ng p , c i nh i *nonpreemptive*
- ng p , c i nh i *preemptive*

c n cơ o hơn? i sao?

Khảo sát giải thuật định thời

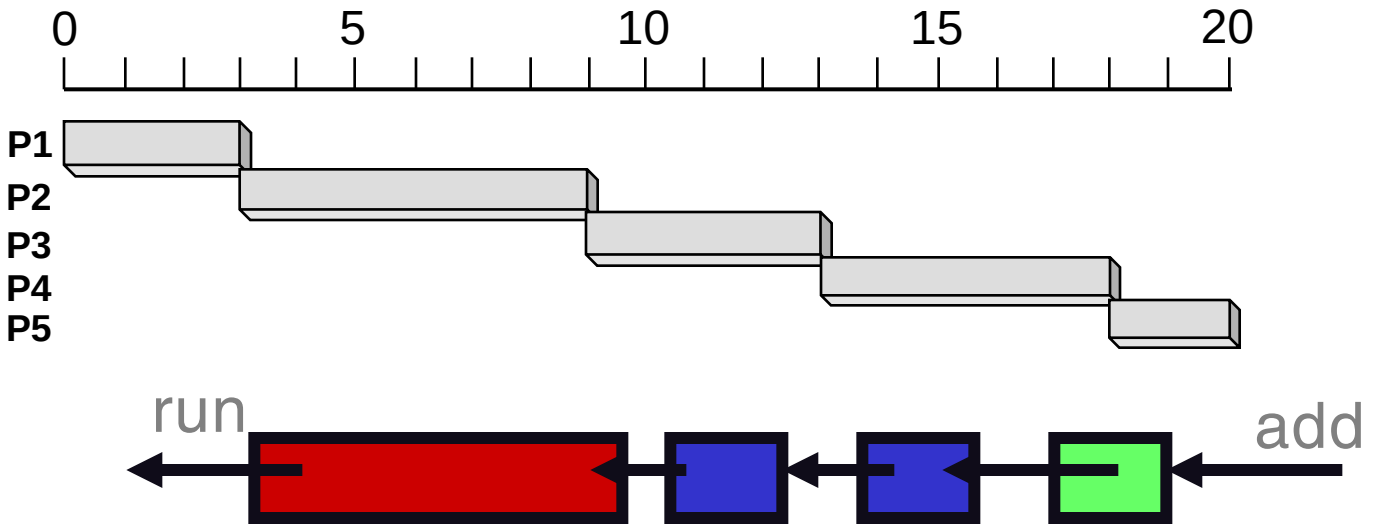


Process	*Arrival Time*	*Service Time*
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

- *Service time* = thời gian process cần CPU trong một chu kỳ CPU-I/O
- Process có service time lớn là các CPU-bound process

1. First-Come First-Served (FCFS)

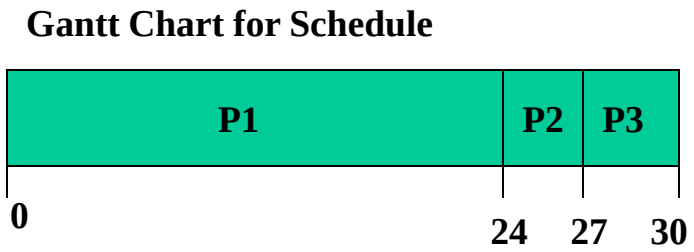
- Hàm lựa chọn: Tiến trình nào yêu cầu CPU trước sẽ được cấp phát CPU trước; Process sẽ thực thi đến khi kết thúc hoặc bị blocked do I/O
- Chế độ quyết định: non-preemptive algorithm
- Hiện thực : sử dụng hàng đợi FIFO (FIFO queues)
  - Tiến trình đi vào được thêm vào cuối hàng đợi
  - Tiến trình được lựa chọn để xử lý được lấy từ đầu của queues



FCFS Scheduling

- Ví dụ :

Process	Burst Time
P1	24
P2	3
P3	3

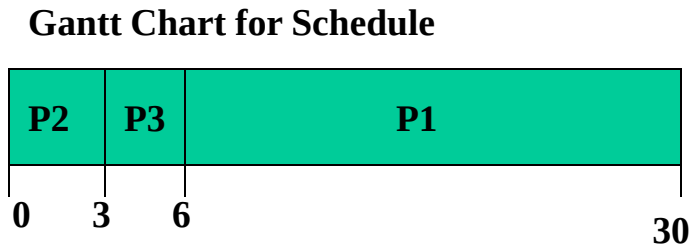


- Giả sử thứ tự vào của các tiến trình là
  - P1, P2, P3
- Thời gian chờ
  - P1 = 0;
  - P2 = 24;
  - P3 = 27;
- Thời gian chờ trung bình
  - (0+24+27)/3 = 17

FCFS Scheduling

- Ví dụ:

Process	Burst Time
P1	24
P2	3
P3	3



- Giả sử thời gian vào của các tiến trình là
  - P2, P3, P1
- Thời gian chờ :
  - P1 = 6; P2 = 0; P3 = 3;
- Thời gian chờ trung bình
  - (6+0+3)/3 = 3 , tốt hơn..

Liệu có xảy ra trường hợp trì hoãn vô hạn định (starvation hay indefinte blocking) với giải thuật FCFS?

Nhận xét

2. Shortest-Job-First(SJF) Scheduling

- Định thời biểu **công việc ngắn nhất trước**
- Khi CPU được tự do, nó sẽ cấp phát cho tiến trình yêu cầu ít thời gian nhất để kết thúc (tiến trình ngắn nhất)
- Liên quan đến chiều dài thời gian sử dụng CPU cho lần tiếp theo của mỗi tiến trình. Sử dụng những chiều dài này để lập lịch cho tiến trình với thời gian ngắn nhất.

2. Shortest-Job-First(SJF) Scheduling

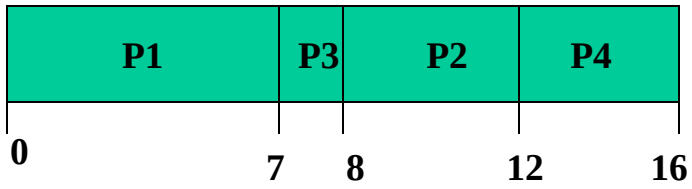
- Hai hình thức (Schemes):
  - *Scheme 1: Non-preemptive*(tiến trình độc quyền CPU)
    - Khi CPU được trao cho quá trình nó không nhường cho đến khi nó kết thúc chu kỳ xử lý của nó
  - *Scheme 2: Preemptive*(tiến trình không độc quyền)
    - Nếu một tiến trình CPU mới được đưa vào danh sách với chiều dài sử dụng CPU cho lần tiếp theo nhỏ hơn thời gian còn lại của tiến trình đang xử lý nó sẽ dừng hoạt động tiến trình hiện hành (hình thức này còn gọi là Shortest-Remaining-Time-First (SRTF).)
  - **SJF là tối ưu** – cho thời gian chờ đợi trung bình tối thiểu với một tập tiến trình cho trước

Non-Preemptive SJF Scheduling

- Ví dụ :

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for Schedule



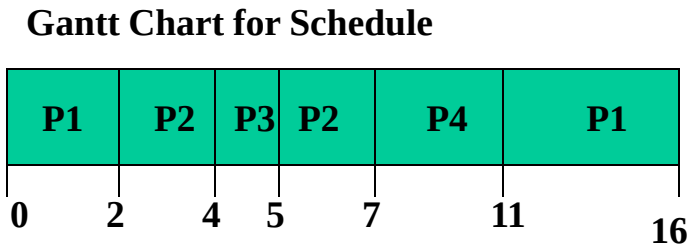
Average waiting time = (0+6+3+7)/4 = 4

Preemptive SJF Scheduling (hay Sortest Remaining Time First - SRTF)

- Ví dụ 1:

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Thực hiện ở chế độ nào?



Average waiting time = (9+1+0+2)/4 = 3

VD2:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Nhận xét về giải thuật SJF

- Có thể xảy ra tình trạng "đói" (starvation) đối với các process có CPU-burst lớn khi có nhiều process với CPU-burst nhỏ đến hệ thống.
- Cơ chế non-preemptive không phù hợp cho hệ thống time sharing (interactive)
- Giải thuật SJF ngầm định ra độ ưu tiên theo burst time
- Các CPU-bound process có độ ưu tiên thấp hơn so với I/O-bound process, nhưng khi một process không thực hiện I/O được thực thi thì nó độc chiếm CPU cho đến khi kết thúc

Nhận xét về giải thuật SJF

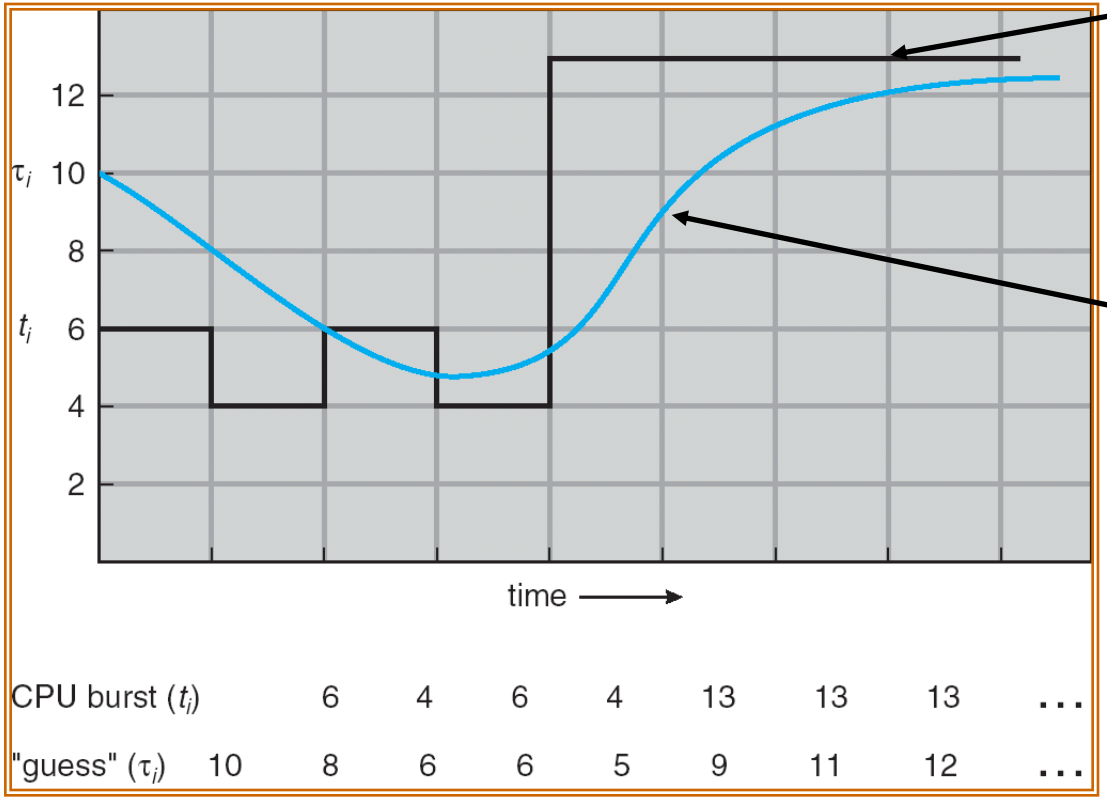
- Tương ứng với mỗi process cần có độ dài của CPU burst tiếp theo
- Hàm lựa chọn: chọn process có độ dài CPU burst nhỏ nhất
- Chứng minh được: SJF tối ưu trong việc giảm thời gian đợi trung bình
- Nhược điểm: Cần phải ước lượng thời gian cần CPU tiếp theo của process
- Giải pháp cho vấn đề này?

Nhận xét về giải thuật SJF

(Thời gian sử dụng CPU chính là độ dài của CPU burst)

- Trung bình tất cả các CPU burst đo được trong quá khứ
- Nhưng thông thường những CPU burst càng mới càng phản ánh đúng hành vi của process trong tương lai
- Một kỹ thuật thường dùng là sử dụng *trung bình hàm mũ* (exponential averaging)
  - $\tau_{n+1} = a\, t_n + (1 - a)\, \tau_n$, $0 \le a \le 1$
  - $\tau_{n+1} = a\, t_n + (1- a)\, a\, t_{n-1} + \ldots + (1- a)^j a\tau_{n-j} + \ldots + (1- a)^{n+1} a\tau_0$
  - Nếu chọn $a = ½$ thì có nghĩa là trị đo được t_n và trị dự đoán τ_n được xem quan trọng như nhau.

Dự đoán thời gian sử dụng CPU



Độ dài CPU burst đo được

Độ dài CPU burst dự đoán, với a = ½ và τ_0 = 10

3. Priority Scheduling

- Mỗi process sẽ được gán một độ ưu tiên
- CPU sẽ được cấp cho process có độ ưu tiên cao nhất
- Định thời sử dụng độ ưu tiên có thể:
  - Preemptive hoặc
  - Nonpreemptive

Gán độ ưu tiên

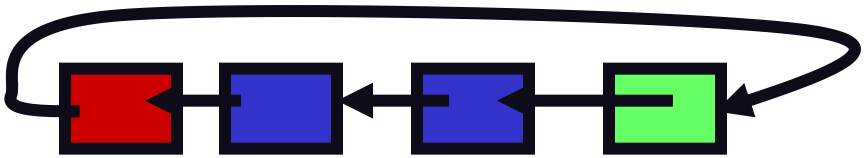
- SJF là một giải thuật định thời sử dụng độ ưu tiên với độ ưu tiên là thời-gian-sử-dụng-CPU-dự-đoán.
- Gán độ ưu tiên còn dựa vào:
  - Yêu cầu về bộ nhớ
  - Số lượng file được mở
  - Tỉ lệ thời gian dùng cho I/O trên thời gian sử dụng CPU
  - Các yêu cầu bên ngoài ví dụ như: số tiền người dùng trả khi thực thi công việc

3. Priority Scheduling

- Vấn đề: trì hoãn vô hạn định – process có độ ưu tiên thấp có thể không bao giờ được thực thi
- Giải pháp: làm mới (*aging)* – độ ưu tiên của process sẽ tăng theo thời gian
- Vd:
  - IBM, MIT 1973 – process 1967.
  - 0 – 127. Sau 15’ tăng độ ưu tiên 1 lần ➔ khoãng bao lâu thì process được thực thi.

4. Định thời luân phiên (Round Robin -RR)

- Mỗi process nhận được một đơn vị nhỏ thời gian CPU (time slice, quantum time), thông thường từ 10-100 msec để thực thi. Sau khoảng thời gian đó, process bị đoạt quyền và trở về cuối hàng đợi ready.
- Nếu có n process trong hàng đợi ready và quantum time = q thì không có process nào phải chờ đợi quá (n – 1)q đơn vị thời gian.



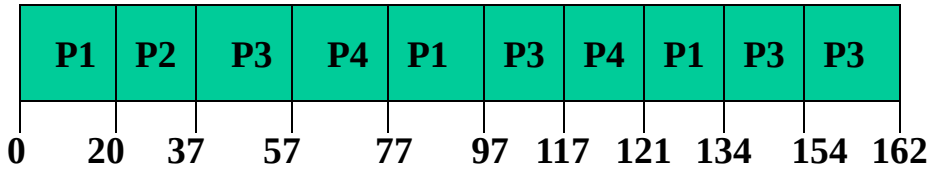
- Hiệu suất
  - Nếu q lớn: RR $\Rightarrow$ FCFS
  - Nếu q nhỏ (q không được quá nhỏ bởi vì phải tốn chi phí chuyển ngữ cảnh)
- Thời gian chờ đợi trung bình của giải thuật RR thường khá lớn nhưng thời gian đáp ứng nhỏ

Ví dụ Round Robin

- Time Quantum = 20

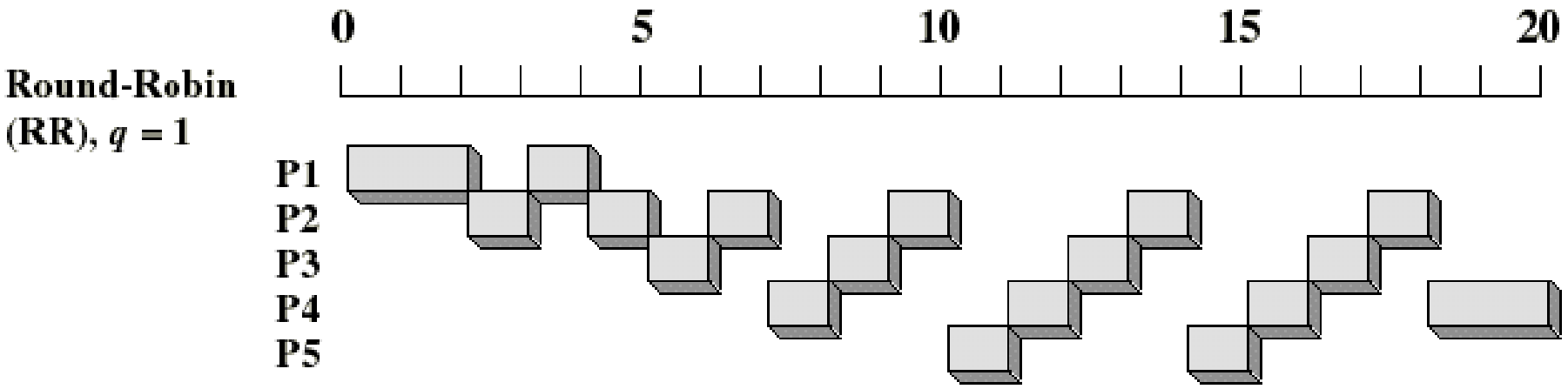
Process	Burst Time
P1	53
P2	17
P3	68
P4	24

Gantt Chart for Schedule



turnaround time trung bình lớn hơn SJF, nhưng đáp ứng tốt hơn

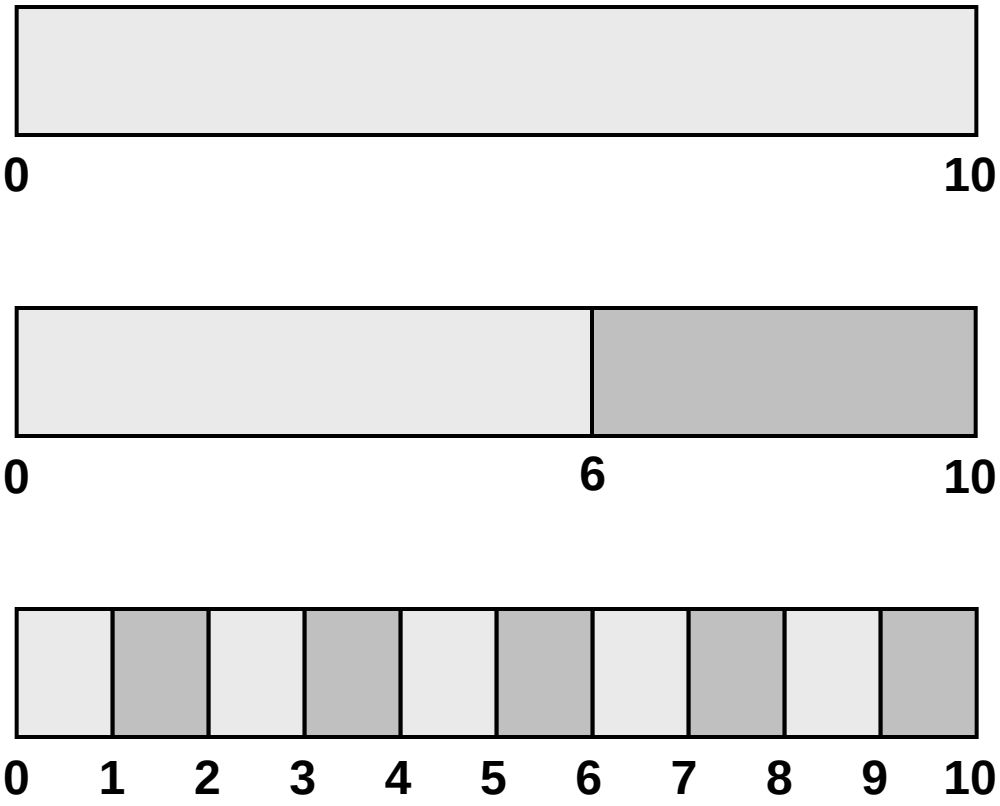
RR với time quantum = 1



- Thời gian turn-around trung bình cao hơn so với SJF nhưng có thời gian đáp ứng trung bình tốt hơn.
- Ưu tiên CPU-bound process
  - I/O-bound process thường sử dụng rất ít thời gian của CPU, sau đó phải blocked đợi I/O
  - CPU-bound process tận dụng hết quantum time, sau đó quay về ready queue ⇒ được xếp trước các process bị blocked

Time quantum và context switch

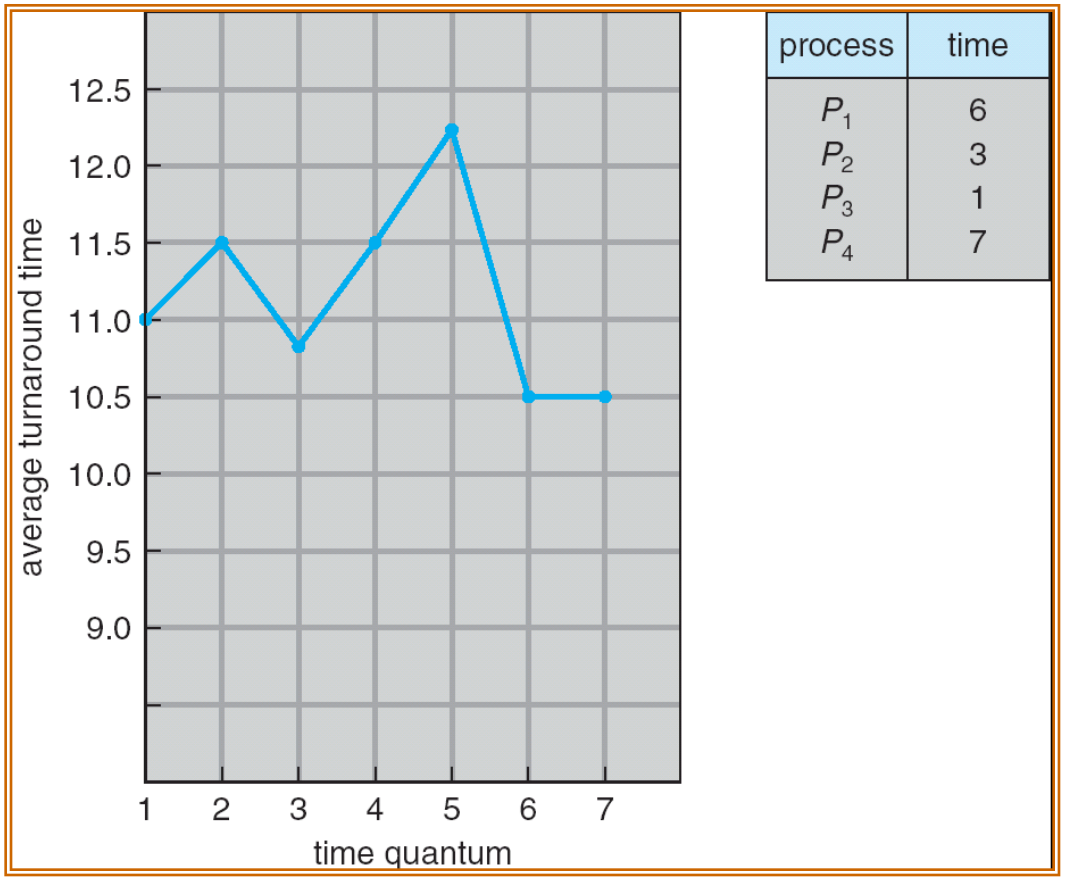
Process time = 10



quantum	context switch
12	0
6	1
1	9

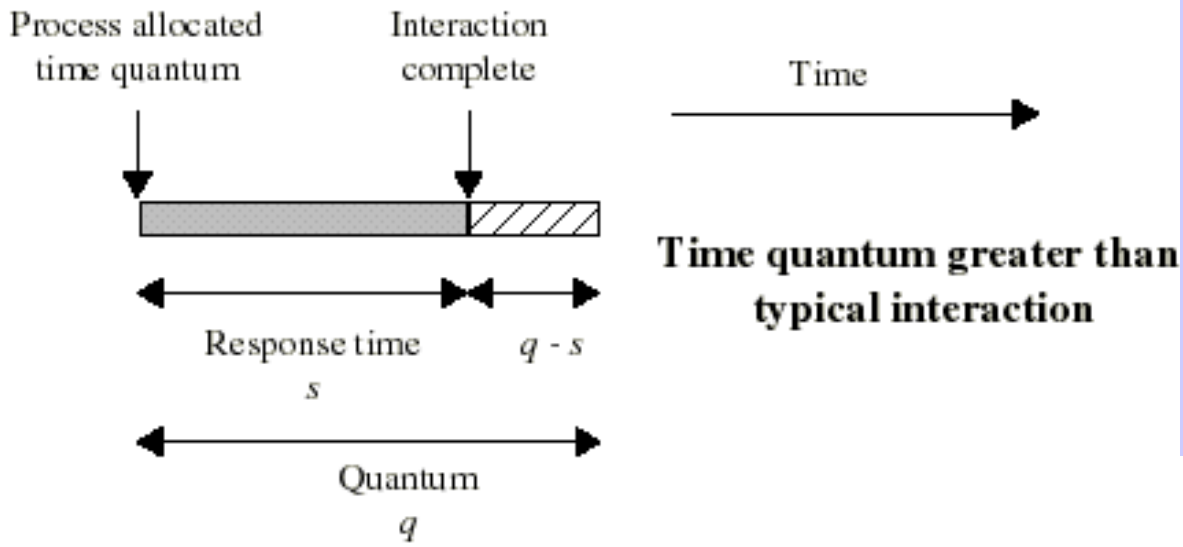
Thời gian hoàn thành và quantum time

- Thời gian hoàn thành trung bình (average turnaround time) không chắc sẽ được cải thiện khi quantum lớn

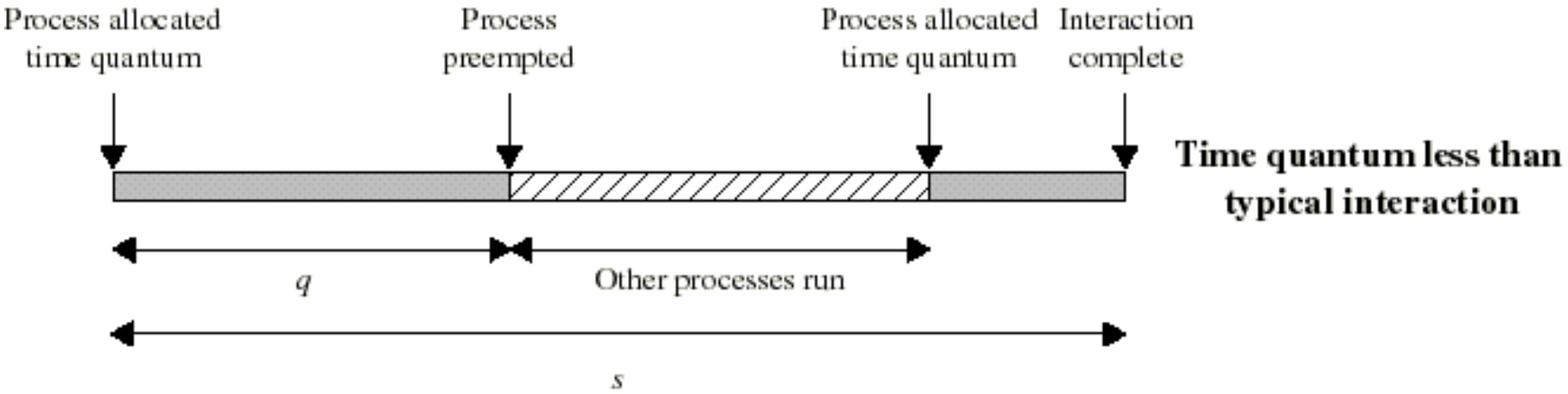


process	time
P_1	6
P_2	3
P_3	1
P_4	7

Quantum và response time



- Quantum time phải lớn hơn thời gian dùng để xử lý clock interrupt (timer) và thời gian dispatching
- Nên lớn hơn thời gian tương tác trung bình (typical interaction)



Quantum time cho Round Robin*

- Khi thực hiện process switch thì OS sẽ sử dụng CPU chứ không phải process của người dùng (*OS overhead*)
  - Dừng thực thi, lưu tất cả thông tin, nạp thông tin của process sắp thực thi
- Performance tùy thuộc vào kích thước của quantum time (còn gọi là time slice), và hàm phụ thuộc này không đơn giản
- Time slice ngắn thì đáp ứng nhanh
  - Vấn đề: có nhiều chuyển ngữ cảnh. Phí tổn sẽ cao.
- Time slice dài hơn thì throughput tốt hơn (do giảm phí tổn OS overhead) nhưng thời gian đáp ứng lớn
  - Nếu time slice quá lớn, RR trở thành FCFS.

Quantum time cho Round Robin

- Quantum time và thời gian cho process switch:
  - Nếu quantum time = 20 ms và thời gian cho process switch = 5 ms, như vậy phí tổn OS overhead chiếm 5/25 = 20%
  - Nếu quantum = 500 ms, thì phí tổn chỉ còn ≈ 1%
    - Nhưng nếu có nhiều người sử dụng trên hệ thống và thuộc loại interactive thì sẽ thấy đáp ứng rất chậm
  - Tùy thuộc vào tập công việc mà lựa chọn quantum time
  - Quantum time nên lớn trong tương quan so sánh với thời gian cho process switch
  - Ví dụ với 4.3 BSD UNIX, quantum time là 1 giây

Round Robin

- Nếu có *n* process trong hàng đợi ready, và quantum time là *q*, như vậy mỗi process sẽ lấy 1/*n* thời gian CPU theo từng khối có kích thước lớn nhất là *q*
  - Sẽ không có process nào chờ lâu hơn (*n* - 1)*q* đơn vị thời gian
- RR sử dụng một giả thiết ngầm là tất cả các process đều có tầm quan trọng ngang nhau
  - Không thể sử dụng RR nếu muốn các process khác nhau có độ ưu tiên khác nhau

Round Robin: nhược điểm

- Các process dạng CPU-bound vẫn còn được "ưu tiên"
  - Ví dụ:
    - Một I/O-bound process sử dụng CPU trong thời gian ngắn hơn quantum time và bị blocked để đợi I/O. Và
    - Một CPU-bound process chạy hết time slice và lại quay trở về hàng đợi ready queue (ở phía trước các process đã bị blocked)

5. Highest Response Ratio Next

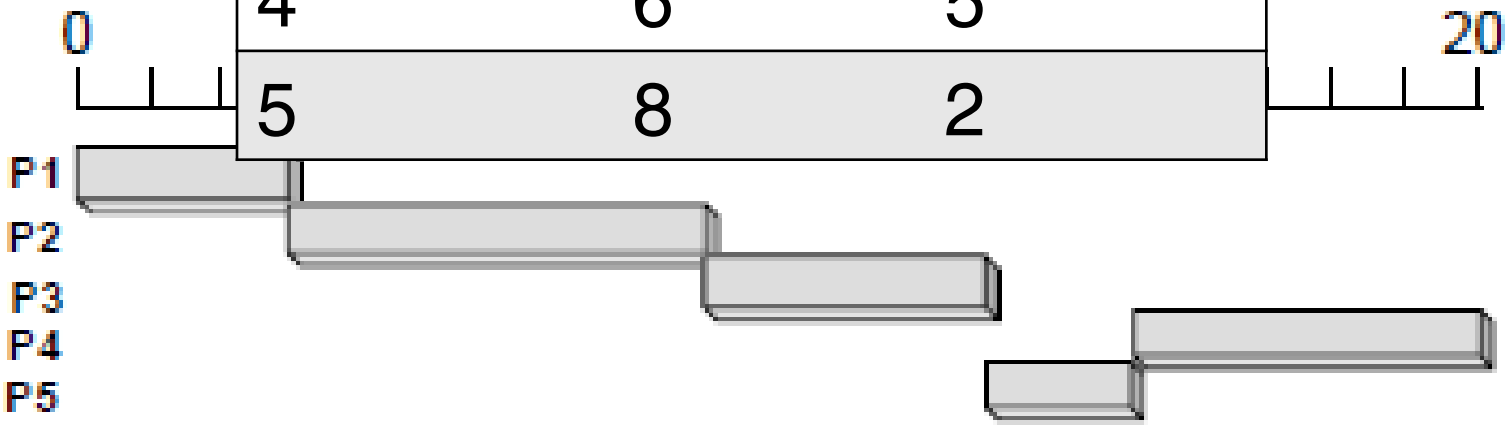
$$RR = \frac{\text{time spent waiting} + \text{expected service time}}{\text{expected service time}}$$

- Chọn process kế tiếp có giá trị *RR* (Response ratio) lớn nhất
- Các process ngắn được ưu tiên hơn (vì service time nhỏ)

Highest Response Ratio Next

Quantumn time = 3

Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

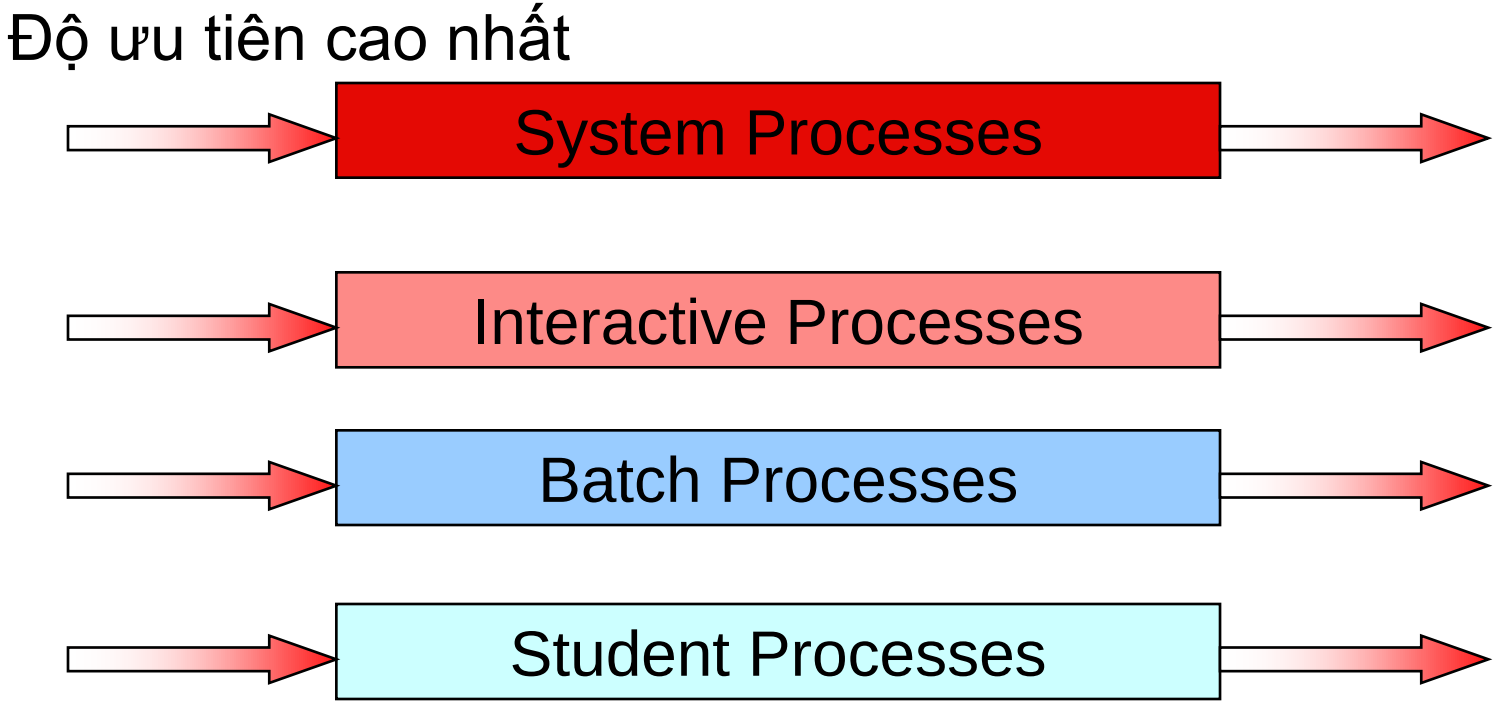


6. Multilevel Queue Scheduling

- Hàng đợi ready được chia thành nhiều hàng đợi riêng biệt theo một số tiêu chuẩn như
  - Đặc điểm và yêu cầu định thời của process
  - Foreground (interactive) và background process,…
- Process được gán cố định vào một hàng đợi, mỗi hàng đợi sử dụng giải thuật định thời riêng
- Hệ điều hành cần phải định thời cho các hàng đợi.
  - *Fixed priority scheduling:* phục vụ từ hàng đợi có độ ưu tiên cao đến thâp. Vấn đề: có thể có starvation.
  - *Time slice:* mỗi hàng đợi được nhận một khoảng thời gian chiếm CPU và phân phối cho các process trong hàng đợi khoảng thời gian đó. Ví dụ:
    - 80% cho hàng đợi foreground định thời bằng RR.
    - 20% cho hàng đợi background định thời bằng giải thuật FCFS.

Multilevel Queue Scheduling*

- Ví dụ phân nhóm các quá trình

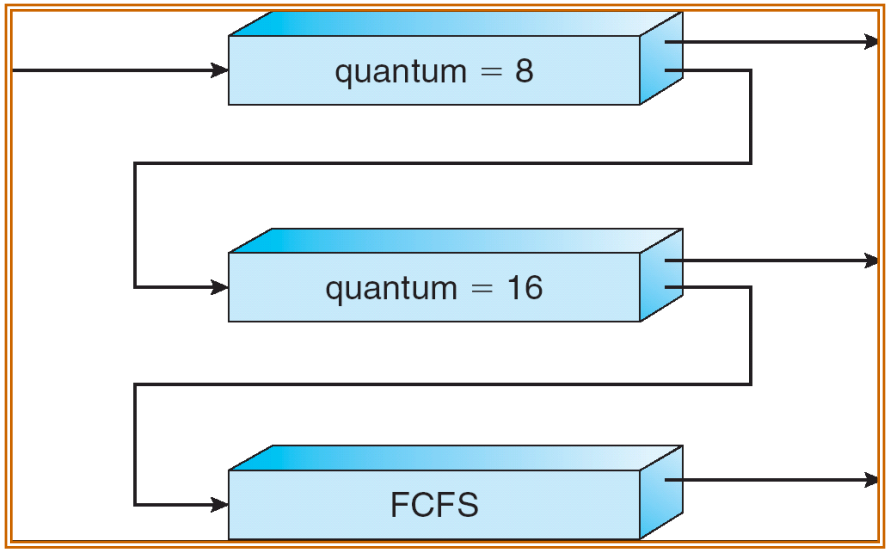


7. Hàng đợi phản hồi đa cấp
Multilevel Feedback Queue

- Vấn đề của multilevel queue
  - process không thể chuyển từ hàng đợi này sang hàng đợi khác $\Rightarrow$ khắc phục bằng cơ chế feedback: cho phép process di chuyển một cách thích hợp giữa các hàng đợi khác nhau.
- *Multilevel Feedback Queue*
  - Phân loại processes dựa trên các đặc tính về CPU-burst
  - Sử dụng decision mode preemptive
  - Sau một khoảng thời gian nào đó, các I/O-bound process và interactive process sẽ ở các hàng đợi có độ ưu tiên cao hơn còn CPU-bound process sẽ ở các queue có độ ưu tiên thấp hơn.
  - Một process đã chờ quá lâu ở một hàng đợi có độ ưu tiên thấp có thể được chuyển đến hàng đợi có độ ưu tiên cao hơn (cơ chế *niên hạn*, aging).

7. Multilevel Feedback Queue

- Ví dụ: Có 3 hàng đợi
  - *Q*0 , dùng RR với quantum 8 ms
  - *Q*1 , dùng RR với quantum 16 ms
  - *Q*2 , dùng FCFS



7. Multilevel Feedback Queue (tt)

- Định thời dùng multilevel feedback queue đòi hỏi phải giải quyết các vấn đề sau
  - Số lượng hàng đợi bao nhiêu là thích hợp?
  - Dùng giải thuật định thời nào ở mỗi hàng đợi?
  - Làm sao để xác định thời điểm cần chuyển một process đến hàng đợi cao hơn hoặc thấp hơn?
  - Khi process yêu cầu được xử lý thì đưa vào hàng đợi nào là hợp lý nhất?

So sánh các giải thuật

- Giải thuật định thời nào là tốt nhất?
- Câu trả lời phụ thuộc các yếu tố sau:
  - Tần xuất tải việc (System workload)
  - Sự hỗ trợ của phần cứng đối với dispatcher
  - Sự tương quan về trọng số của các tiêu chuẩn định thời như response time, hiệu suất CPU, throughput,…
  - Phương pháp định lượng so sánh

Đọc thêm

- Policy và Mechanism
- Định thời trên hệ thống multiprocessor
- Đánh giá giải thuật định thời CPU
- Định thời trong một số hệ điều hành thông dụng

- Nguồn:

Operating System Concepts. Sixth Edition. John Wiley & Sons, Inc. 2002. Silberschatz, Galvin, Gagne

Bài tập

Xét 1 tập các process sau có thời gian thực thi CPU tính bằng mili giây:

Process	Burst - time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
p5	5	2

Giả sử thứ tự đến để thực thi của các process là P1, P2, P3, P4, P5. (tất cả process này đều đến tại thời điểm bằng 0).

1. Vẽ sơ đồ Gautt thực thi của các process theo giải thuật định thời: FCFS, SJF, RR (quantumn = 1). Ở cả 2 chế độ preemptive và nonpreemptive.
2. Thời gian đợi của các process trong từng giải thuật định thời?

Bài tập

3. Tính thời gian hoàn thành(turnaround time) của từng process cho từng giải thuật?

Trong các giải thuật sau, giải thuật nào có thể gây ra trường hợp 1 process có thể không bao giờ được thực thi:

1. First come, first serve
2. Shortest job first
3. Round robin
4. Priority.

Giải thích lý do tại sao xảy ra trường hợp trên?

Bài tập

Process	Arrival time	Burst time (ms)	Priority
P1	0	8	3
P2	1	4	2
P3	2	3	1
P4	4	5	4

1. RR với q = 2
2. Preemptive Priority với số càng lớn càng ưu tiên
3. Điều phối ưu tiên nhiều cấp xoay vòng, sử dụng 2 cấp: Cấp 1 sử dụng giải thuật robin round với quantumn = 3ms. Cấp 2 sử dụng giải thuật SRTF. Một process nếu đã ở cấp I 5ms sẽ được chuyển xuống cấp II nếu đang ở trạng thái waiting còn nếu đang ở trạng thái running thì sau khi ra khỏi sẽ chuyển. Ngược lại một process đang ở cấp II sau khoãng thời gian 10ms sẽ được chuyển lên I. Khi các process vào bộ nhớ chính thì điều vào hàng đợi cấp I.

Bài tập

- Cho 4 tiến trình A, B, C, D với thời gian vào ready list và thời gian cần CPU cho các lần thứ 1, thứ 2, thứ 3 và thời gian thực hiện I/O tương ứng như bản sau:

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

- Vẽ giản Gantt, Tính thời gian đợi trung bình, Thời gian đáp ứng trung bình, Thời gian lưu lại trong hệ thống trung bình cho các giải thuật (câu 1, 2 chỉ dùng thời gian thực thi 1st execution time như là burst time)

1. FCFS
2. RR với q = 3
3. SRFT cho cả 3 lần exec