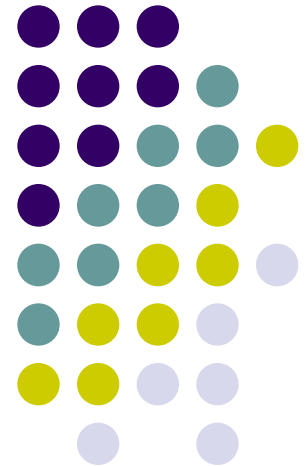
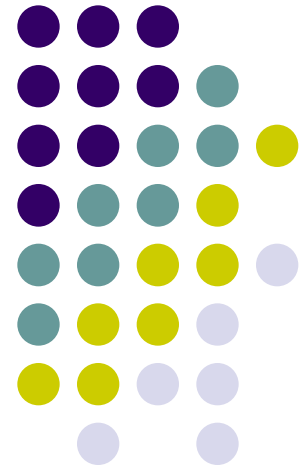


Nguyên lý hệ điều hành

Lê Đức Trọng
DS&KTLab, Khoa CNTT
Trường Đại học Công nghệ
(Slide credit: PGS. TS. Nguyễn Hải Châu)



Lập lịch CPU

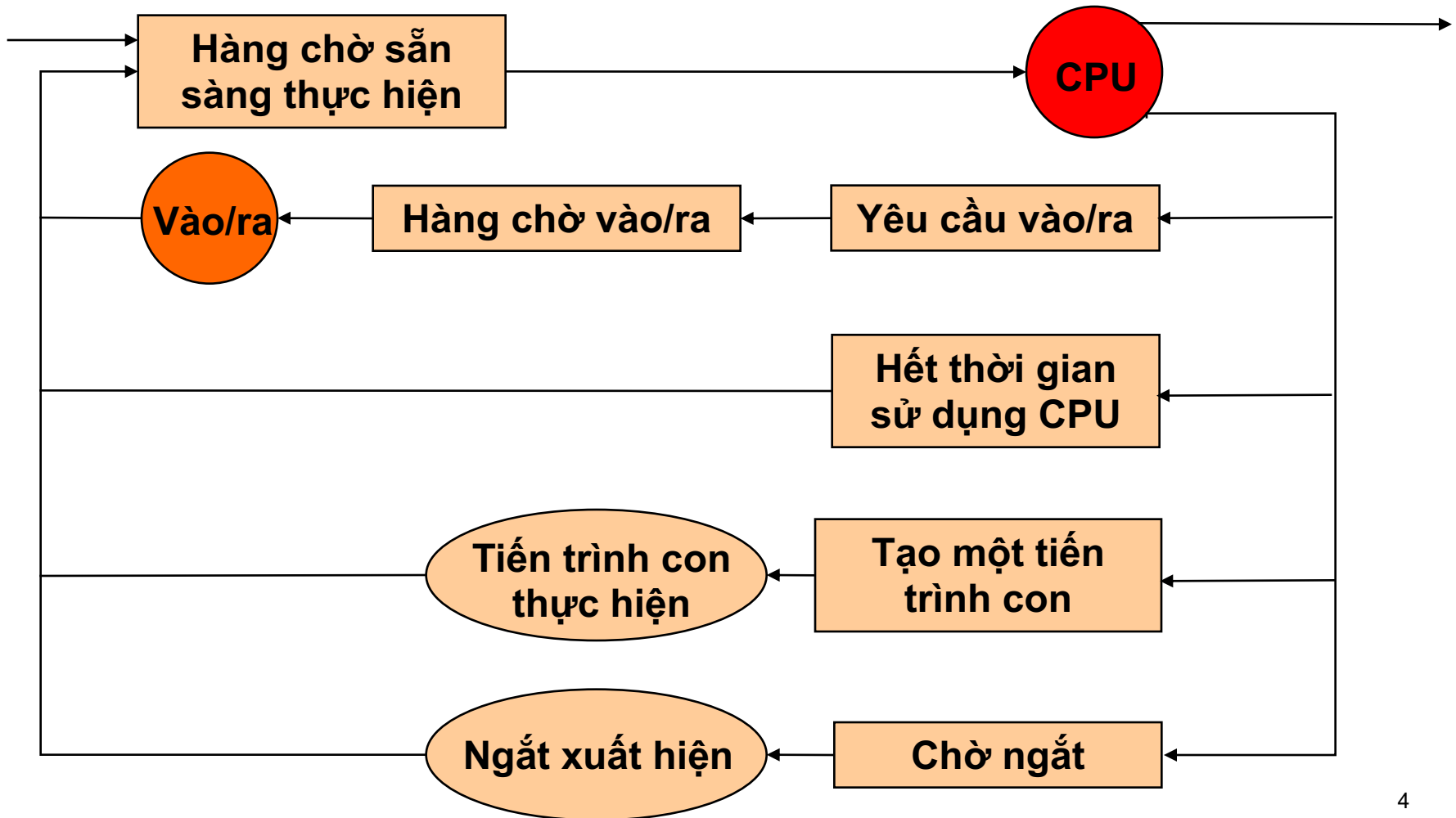
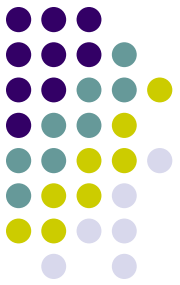




Tại sao phải lập lịch CPU?

- Số lượng NSD, số lượng tiến trình luôn lớn hơn số lượng CPU của máy tính rất nhiều
- Tại một thời điểm, chỉ có duy nhất một tiến trình được thực hiện trên một CPU
- Vấn đề:
 - Nhu cầu sử dụng nhiều hơn tài nguyên (CPU) đang có
 - Do đó cần lập lịch để phân phối thời gian sử dụng CPU cho các tiến trình của NSD và hệ thống

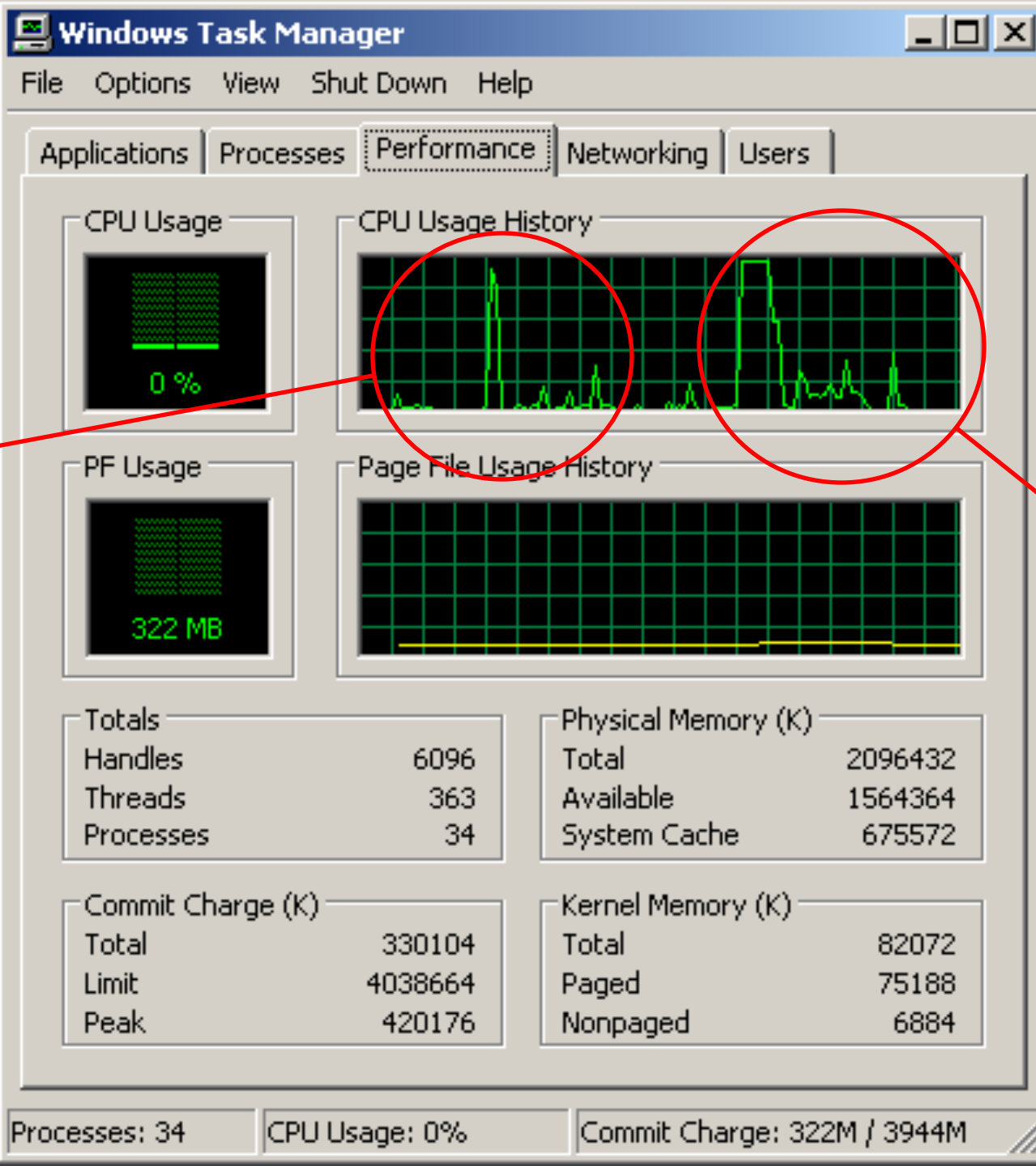
Hàng chờ lập lịch tiến trình





CPU-burst và IO-burst

- Trong suốt thời gian tồn tại trong hệ thống, tiến trình được xem như thực hiện hai loại công việc chính:
 - Khi tiến trình ở trạng thái running: **Sử dụng CPU** (thuật ngữ: *CPU-burst*)
 - Khi tiến trình thực hiện các thao tác vào ra: **Sử dụng thiết bị vào/ra** (thuật ngữ: *I/O burst*)



Microsoft
Office
Outlook
CPU-burst

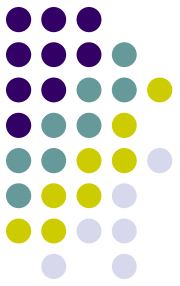
Adobe
Photoshop
CPU-burst



Hai loại tiến trình chính

- Căn cứ theo cách sử dụng CPU của tiến trình, có hai loại tiến trình:
 - Tiến trình loại CPU-bound: Tiến trình có một hoặc nhiều phiên sử dụng CPU dài
 - Tiến trình loại I/O-bound: Tiến trình có nhiều phiên sử dụng CPU ngắn (tức là thời gian vào ra nhiều)

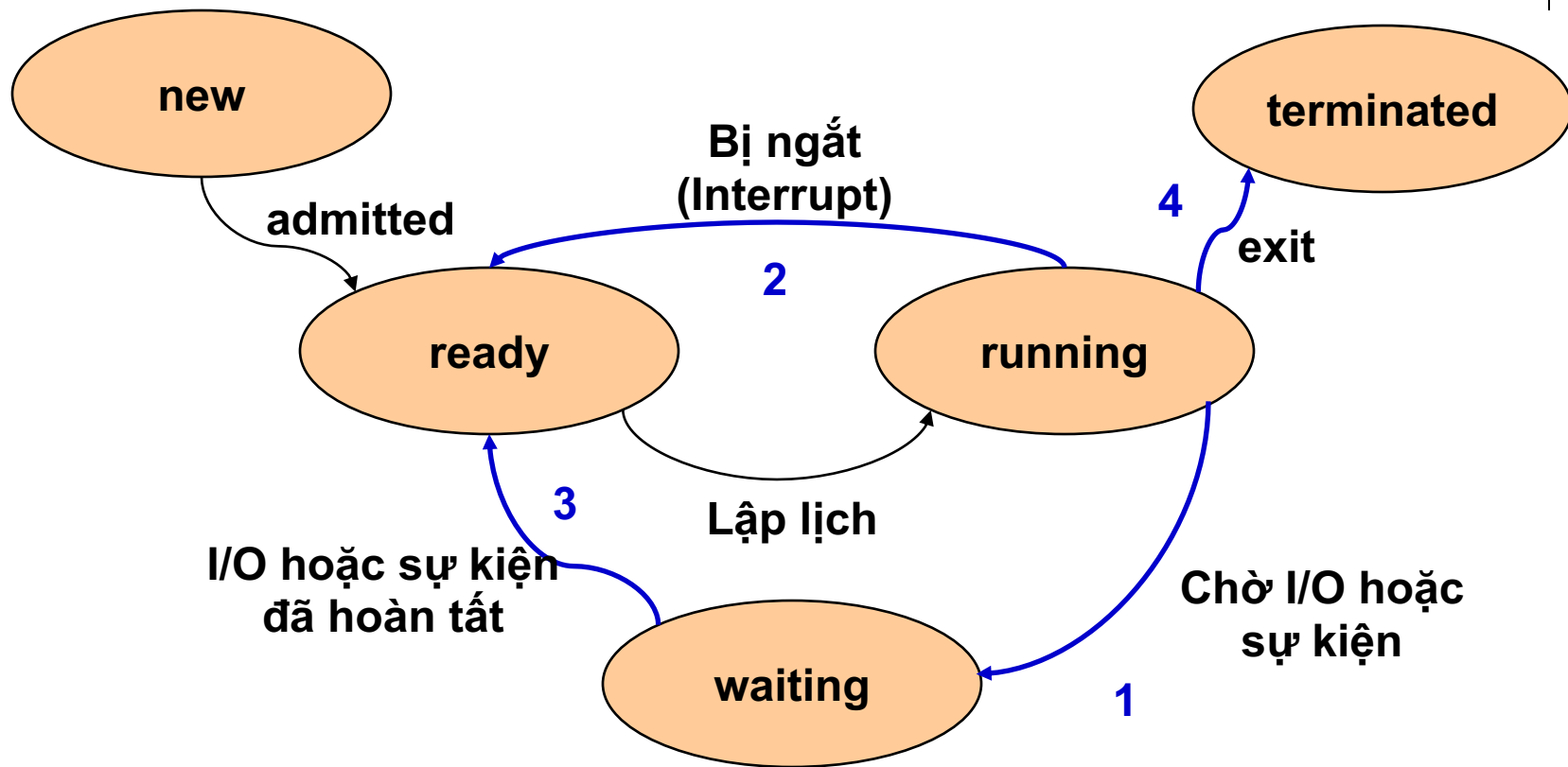
Bộ lập lịch ra hoạt động khi...



1. Một tiến trình chuyển từ trạng thái running sang waiting
2. Một tiến trình chuyển từ trạng thái running sang ready
3. Một tiến trình chuyển từ trạng thái waiting sang ready
4. Một tiến trình kết thúc



Các phương pháp lập lịch



- 1 và 4: Lập lịch non-preemptive
- Ngược lại: Lập lịch preemptive



Lập lịch non-preemptive

- Một tiến trình giữ CPU đến khi nó kết thúc hoặc chuyển sang trạng thái waiting.
- Ví dụ: Microsoft Windows 3.1, Apple Macintosh sử dụng lập lịch non-preemptive
- Có thể sử dụng trên nhiều loại phần cứng vì không đòi hỏi timer



Lập lịch preemptive

- Hiệu quả hơn lập lịch non-preemptive
- Thuật toán phức tạp hơn non-preemptive và sử dụng nhiều tài nguyên CPU hơn
- Ví dụ: Microsoft Windows XP, Linux, UNIX sử dụng lập lịch preemptive



Bộ điều phối (dispatcher)

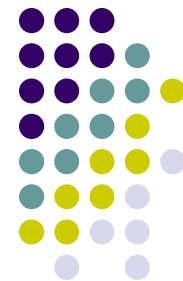
- Nhiệm vụ:
 - Chuyển trạng thái (context switch)
 - Chuyển về user-mode
 - Thực hiện tiến trình theo trạng thái đã lưu
- Cần hoạt động hiệu quả (tốc độ nhanh)
- Thời gian cần để bộ điều phối dừng một tiến trình và thực hiện tiến trình khác gọi là độ trễ (latency) của bộ điều phối



Các tiêu chí đánh giá lập lịch

- Khả năng tận dụng CPU (*CPU utilization*):
Thể hiện qua tải CPU – là một số từ 0% đến 100%.
 - Trong thực tế các hệ thống thường có tải từ 40% (tải thấp) đến 90% (tải cao)
- Thông lượng (*throughput*): Là số lượng các tiến trình hoàn thành trong một đơn vị thời gian

Các tiêu chí đánh giá lập lịch

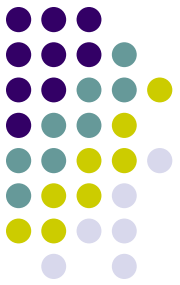


- Thời gian hoàn thành (*turnaround time*):
 - $t_{turnaround} = t_o - t_i$ với t_i là thời điểm tiến trình vào hệ thống, t_o là thời điểm tiến trình ra khỏi hệ thống (kết thúc thực hiện)
 - Như vậy $t_{turnaround}$ là tổng: thời gian tải vào bộ nhớ, thời gian thực hiện, thời gian vào ra, thời gian nằm trong hàng chờ...



Các tiêu chí đánh giá lập lịch

- Thời gian chờ (*waiting time*): Là tổng thời gian tiến trình phải nằm trong hàng chờ ready ($t_{waiting}$)
 - Các thuật toán lập lịch CPU không có ảnh hưởng đến tổng thời gian thực hiện một tiến trình mà chỉ có ảnh hưởng đến thời gian chờ của một tiến trình trong hàng chờ ready
 - Thời gian chờ trung bình (*average waiting time*):
 $t_{averagewaiting} = t_{waiting} / n$, n là số lượng tiến trình trong hàng chờ



Các tiêu chí đánh giá lập lịch

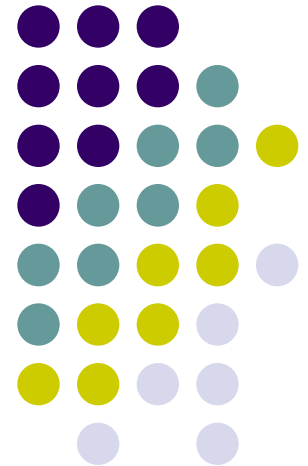
- Thời gian đáp ứng (*response time*): Là khoảng thời gian từ khi tiến trình nhận được một yêu cầu cho đến khi *bắt đầu đáp ứng* yêu cầu đó
- $t_{res} = t_r - t_s$, trong đó t_r là thời điểm nhận yêu cầu, t_s là thời điểm bắt đầu đáp ứng yêu cầu



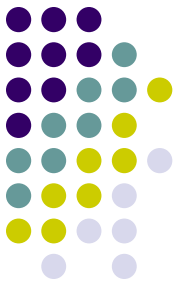
Đánh giá các thuật toán lập lịch

Tiêu chí	Giá trị thấp	Giá trị cao
Khả năng tận dụng CPU (CPU utilization)	Xấu	Tốt
Thông lượng (throughput)	Xấu	Tốt
Thời gian hoàn thành (turnaround time)	Tốt	Xấu
Thời gian chờ (waiting time) -> Thời gian chờ trung bình	Tốt	Xấu
Thời gian đáp ứng (response time)	Tốt	Xấu

Các thuật toán lập lịch



FCFS (First Come First Served)

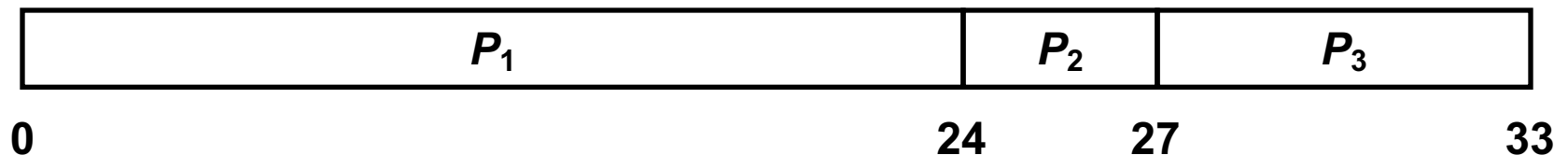


- Tiến trình nào có yêu cầu sử dụng CPU trước sẽ được thực hiện trước
- Ưu điểm: Thuật toán đơn giản nhất
- Nhược điểm: Hiệu quả của thuật toán phụ thuộc vào *thứ tự* của các tiến trình trong hàng chờ, vì thứ tự này ảnh hưởng rất lớn đến *thời gian chờ trung bình (average waiting time)*



Ví dụ FCFS 1a

- Giả sử có 3 tiến trình P_1 , P_2 , P_3 với thời gian thực hiện tương ứng là 24ms, 3ms, 6ms
- Giả sử 3 tiến trình xếp hàng theo thứ tự P_1 , P_2 , P_3 . Khi đó ta có biểu đồ Gantt như sau:

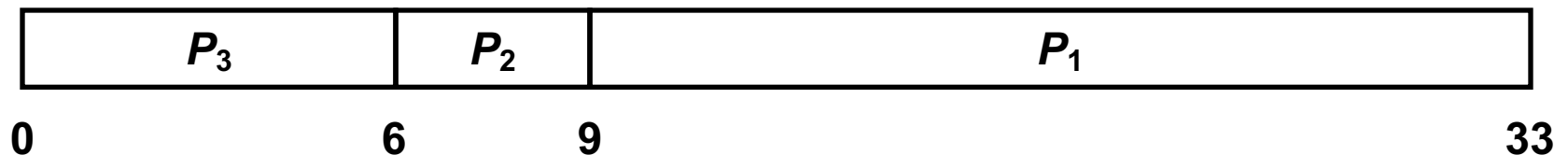


- Thời gian chờ của các tiến trình là: P_1 chờ 0ms, P_2 chờ 24ms, P_3 chờ 27ms
- Thời gian chờ trung bình: $(0+24+27)/3=17\text{ms}$



Ví dụ FCFS 1b

- Xét ba tiến trình trong ví dụ 1a với thứ tự xếp hàng P_3, P_2, P_1
- Biểu đồ Gantt:



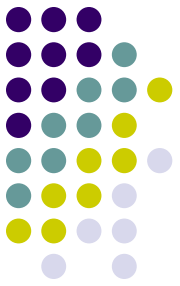
- Thời gian chờ của các tiến trình là: P_3 chờ 0ms, P_2 chờ 6ms, P_1 chờ 9ms
- Thời gian chờ trung bình: $(0+6+9)/3=5\text{ms}$
- Thời gian chờ trung bình thấp hơn thời gian chờ trung bình trong ví dụ 1a



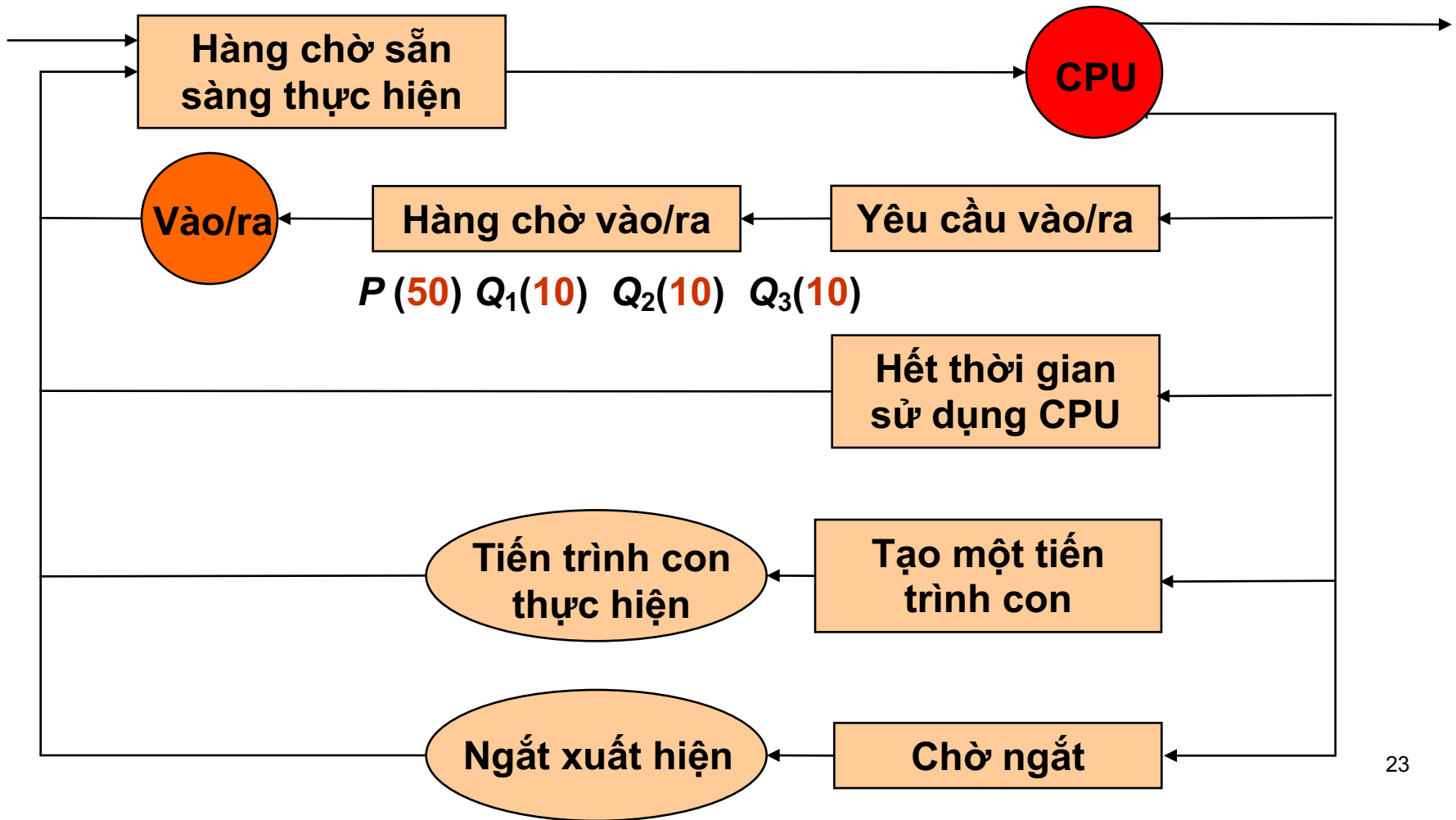
Hiện tượng “đoàn hộ tống”

- Thuật ngữ: *convoy effect*
- Xảy ra khi có một tiến trình “lớn” P nằm ở đầu hàng chờ và nhiều tiến trình “nhỏ” Q_i xếp hàng sau P .
- “Lớn”: Sử dụng nhiều thời gian CPU và vào ra
- “Nhỏ”: Sử dụng ít thời gian CPU và vào ra
- Thuật toán lập lịch được sử dụng là FCFS.:
- Hiện tượng xảy ra: CPU, thiết bị vào ra có nhiều thời gian rỗi, thời gian chờ trung bình của các tiến trình cao

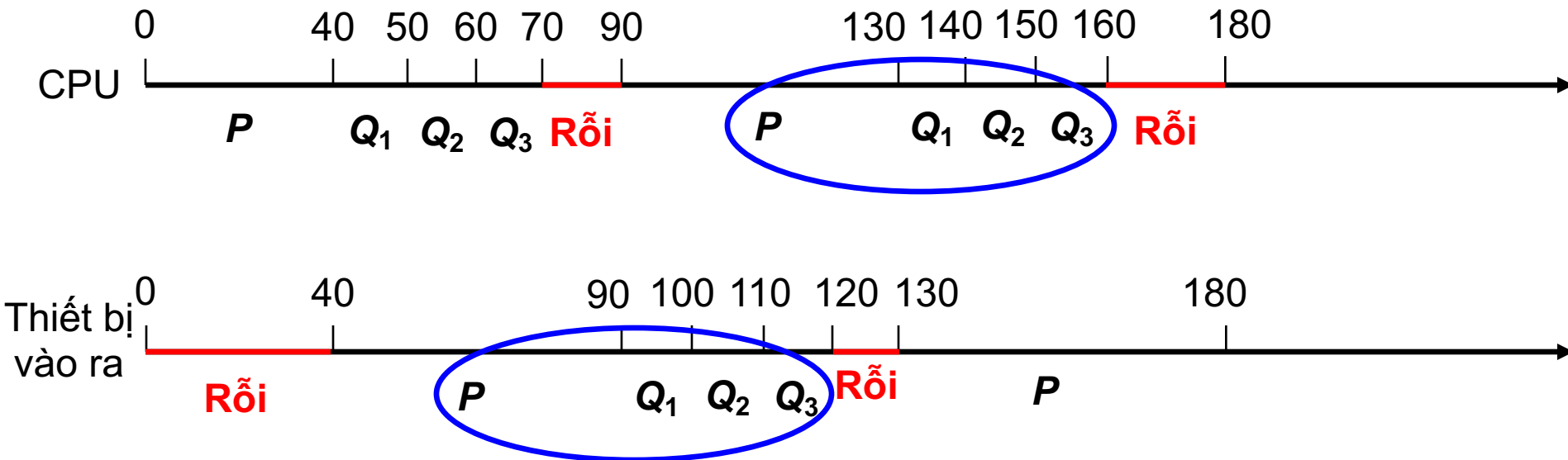
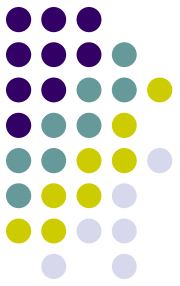
Ví dụ convoy effect



$Q_3(10)$ $Q_2(10)$ $Q_1(10)$ $P(40)$



Convoy effect: Thời gian sử dụng CPU và thiết bị vào ra



- Giả sử P là tiến trình “lớn” có chu kỳ sử dụng CPU trong 40ms và vào ra trong 50ms
- Q_1 , Q_2 , Q_3 là 3 tiến trình “nhỏ” có chu kỳ sử dụng CPU trong 10ms và vào ra trong 10ms
- Thứ tự xếp hàng: P , Q_1 , Q_2 , Q_3



SJF (Shortest Job First)

- Với SJF, tham số lập lịch có thêm *độ dài của phiên sử dụng CPU tiếp theo* $t_{nextburst}$
- Tiến trình có $t_{nextburst}$ nhỏ nhất sẽ được lập lịch sử dụng CPU trước
- Nếu hai tiến trình có $t_{nextburst}$ bằng nhau thì FCFS được áp dụng



SJF (Shortest Job First)

- Với lập lịch dài hạn: Có thể biết $t_{nextburst}$ vì có tham số chỉ ra thời gian chạy của tiến trình khi đưa tiến trình vào hệ thống (job submit)
- Khó khăn: Chỉ có thể phỏng đoán được $t_{nextburst}$ với lập lịch ngắn hạn
- **Đọc ở nhà:** Dự đoán $t_{nextburst}$ bằng công thức trung bình mũ (exponential average) trang 160-162 trong giáo trình
- SJF không tối ưu được với lập lịch ngắn hạn
- SJF thường được áp dụng cho lập lịch dài hạn



Lập lịch có độ ưu tiên

- Thuật ngữ: Priority scheduling
- Mỗi tiến trình được gán một tham số lập lịch gọi là độ ưu tiên p , với p là một số thực
- Tiến trình có độ ưu tiên cao nhất được sử dụng CPU
- Qui ước trong môn học này:
 - Tiến trình P_1 và P_2 có độ ưu tiên p_1, p_2 tương ứng
 - $p_1 > p_2$ có nghĩa là tiến trình P_1 có độ ưu tiên thấp hơn P_2
- SJF là trường hợp đặc biệt của lập lịch có ưu tiên với ưu tiên của các tiến trình là nghịch đảo độ dài phiên sử dụng



Lập lịch có độ ưu tiên

- Hai cách xác định độ ưu tiên:
 - Độ ưu tiên trong: Được tính toán dựa trên các tham số định lượng của tiến trình như giới hạn về thời gian, bộ nhớ, số file đang mở, thời gian sử dụng CPU
 - Độ ưu tiên ngoài: Dựa vào các yếu tố như mức độ quan trọng, chi phí thuê máy tính...
- Chờ không xác định: Một tiến trình có độ ưu tiên thấp có thể nằm trong hàng chờ trong một khoảng thời gian dài nếu trong hàng chờ luôn có các tiến trình có độ ưu tiên cao hơn



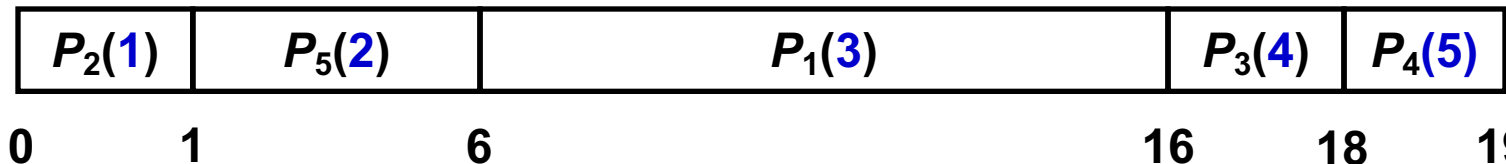
Lập lịch có độ ưu tiên

- Để tránh hiện tượng chờ không xác định, có thêm tham số *tuổi* để xác định thời gian tiến trình thời gian nằm trong hàng chờ
- Tham số *tuổi* được sử dụng để làm tăng độ ưu tiên của tiến trình
- Ví dụ thực tế: Khi bảo dưỡng máy tính IBM 7094 của MIT năm 1973, người ta thấy một tiến trình nằm trong hàng chờ từ năm 1967 nhưng vẫn chưa được thực hiện

Ví dụ lập lịch có độ ưu tiên

- Xét 5 tiến trình như trong bảng với thứ tự trong hàng chờ là P_1, P_2, P_3, P_4, P_5
- Sử dụng thuật toán lập lịch có ưu tiên, ta có thứ tự thực hiện của các tiến trình như sau biểu đồ dưới
- Thời gian chờ trung bình là $(1+6+16+18)/5=8.2ms$

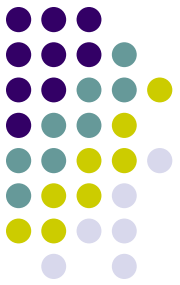
Tiến trình	Thời gian thực hiện (ms)	Độ ưu tiên
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2





Round-robin (RR)

- Còn gọi là lập lịch quay vòng
- Được thiết kế để áp dụng cho các hệ phân chia thời gian (time-sharing)
- RR hoạt động theo chế độ preemptive
- Tham số lượng tử thời gian (time quantum)
 $t_{quantum}$: Mỗi tiến trình được sử dụng CPU trong nhiều nhất bằng $t_{quantum}$, sau đó đến lượt tiến trình khác



Round-robin (RR)

- Hiệu quả của RR phụ thuộc độ lớn của $t_{quantum}$
 - Nếu $t_{quantum}$ nhỏ thì hiệu quả của RR giảm vì bộ điều phối phải thực hiện nhiều thao tác chuyển trạng thái, lãng phí thời gian CPU
 - Nếu $t_{quantum}$ lớn thì số thao tác chuyển trạng thái giảm đi
- Nếu $t_{quantum}$ rất nhỏ (ví dụ 1ms) thì RR được gọi là processor sharing
- Nếu $t_{quantum} = \infty$ thì RR trở thành FCFS



Ví dụ RR

- Giả sử có 3 tiến trình P_1 , P_2 , P_3 với thời gian thực hiện tương ứng là 24ms, 3ms, 6ms, thứ tự trong hàng chờ P_1 , P_2 , P_3 , vào hàng chờ cùng thời điểm 0
- Giả sử $t_{quantum}=4ms$
- RR lập lịch các tiến trình như sau:

P_1	P_2	P_3	P_1	P_3	P_1	P_1	P_1	P_1	
0	4	7	11	15	17	21	25	29	33

- Thời gian chờ
 - P_1 : $0+(11-4)+(17-15)=9ms$
 - P_2 : 4ms
 - P_3 : $7+(15-11)=11ms$
- Thời gian chờ trung bình: $(9+4+11)/3=8ms$



Lập lịch với hàng chờ đa mức

- Thuật ngữ: Multilevel queue scheduling
- Được sử dụng khi ta có thể chia các tiến trình thành nhiều lớp khác nhau để lập lịch theo các tiêu chí khác nhau, ví dụ:
 - Lớp các tiến trình có tương tác (interactive hoặc foreground process) cần có độ ưu tiên cao hơn
 - Lớp các tiến trình chạy nền (background) thường không có tương tác với NSD: Độ ưu tiên thấp hơn



Lập lịch với hàng chờ đa mức

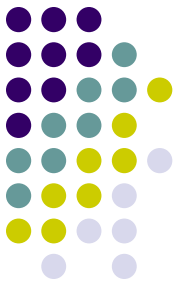
- Thuật toán lập lịch với hàng chờ đa mức chia hàng chờ ready thành nhiều hàng chờ con khác nhau, mỗi hàng chờ con được áp dụng một loại thuật toán khác nhau, ví dụ:
 - Hàng chờ các tiến trình background: FCFS
 - Hàng chờ các tiến trình có tương tác: RR
- Các tiến trình được phân lớp dựa vào đặc tính như bộ nhớ, độ ưu tiên, ...
- Cần có thuật toán lập lịch cho các hàng chờ con, ví dụ: preemptive có độ ưu tiên cố định



Ví dụ hàng chờ đa mức

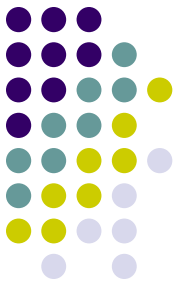
- Ví dụ các hàng chờ đa mức có độ ưu tiên giảm dần:
 - Hàng chờ các tiến trình hệ thống
 - Hàng chờ các tiến trình có tương tác
 - Hàng chờ các tiến trình là editor
 - Hàng chờ các tiến trình hoạt động theo lô
 - Hàng chờ các tiến trình thực tập của sinh viên

Lập lịch với hàng chờ đa mức có phản hồi



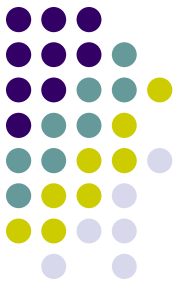
- Thuật ngữ: Multilevel feedback-queue scheduling
- Thuật toán lập lịch kiểu này nhằm khắc phục nhược điểm *không mềm dẻo* của lập lịch với hàng chờ đa mức
- Ý tưởng chính: Cho phép tiến trình chuyển từ hàng chờ này sang hàng chờ khác, trong khi lập lịch với hàng chờ đa mức không cho phép điều này

Lập lịch với hàng chờ đa mức có phản hồi

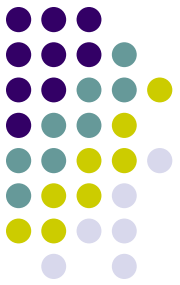


- Cách thực hiện:
 - Độ dài phiên sử dụng CPU và thời gian đã nằm trong hàng chờ là tiêu chuẩn chuyển tiến trình giữa các hàng chờ
 - Tiến trình chiếm nhiều thời gian CPU sẽ bị chuyển xuống hàng chờ có độ ưu tiên thấp
 - Tiến trình nằm lâu trong hàng chờ sẽ được chuyển lên hàng chờ có độ ưu tiên cao hơn

Các tham số của bộ lập lịch hàng chờ đa mức có phản hồi



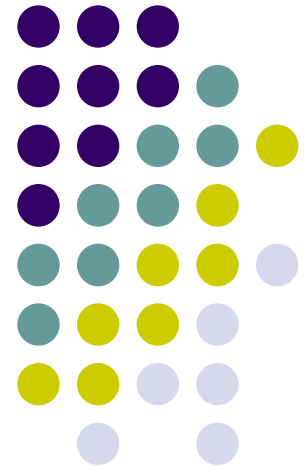
- Số lượng các hàng chờ
- Thuật toán lập lịch cho mỗi hàng chờ
- Phương pháp tăng độ ưu tiên cho một tiến trình
- Phương pháp giảm độ ưu tiên cho một tiến trình
- Phương pháp xác định hàng đợi nào để đưa một tiến trình vào



Các thuật toán lập lịch khác

- Sinh viên tìm hiểu trong giáo trình:
 - Lập lịch đa xử lý
 - Lập lịch thời gian thực

Các phương pháp đánh giá thuật toán lập lịch





Mô hình xác định

- Thuật ngữ: Deterministic modeling
- Dựa vào các trường hợp cụ thể (chẳng hạn các ví dụ đã nói trong bài giảng này) để rút ra các kết luận đánh giá
- Ưu điểm: Nhanh và đơn giản
- Nhược điểm: Không rút ra được kết luận đánh giá cho trường hợp tổng quát



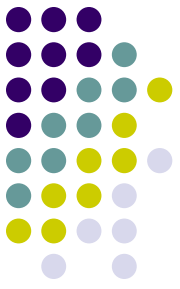
Mô hình hàng chờ

- Thuật ngữ: Queueing model
- Dựa trên lý thuyết xác suất, quá trình ngẫu nhiên. Tài liệu tham khảo:
 - Leonard Kleinrock, *Queueing Systems: Volume I – Theory* (Wiley Interscience, New York, 1975)
 - Leonard Kleinrock, *Queueing Systems: Volume II – Computer Applications* (Wiley Interscience, New York, 1976)
- Ưu điểm: So sánh được các thuật toán lập lịch trong một số trường hợp
- Hạn chế: Phức tạp về mặt toán học, lớp các thuật toán phân tích so sánh được còn hạn chế



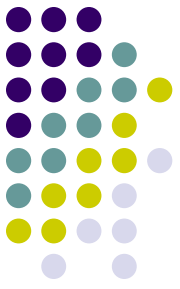
Mô phỏng

- Thuật ngữ: Simulation
- Thực hiện các tình huống giả định trên máy tính để đánh giá hiệu quả của các thuật toán lập lịch, kiểm nghiệm các kết quả lý thuyết
- Mô phỏng thường tốn thời gian CPU và không gian lưu trữ
- Được sử dụng nhiều trong công nghiệp
- Ví dụ các hệ mô phỏng mạng (có module hàng chờ): OPNET, NS-2, Qualnet



Cài đặt thử trong thực tế

- Mô phỏng có thể xem như “qui nạp không hoàn toàn”
- Có thể xây dựng hệ thống thử trong thực tế
- Ưu điểm: Đánh giá được hiệu quả thực sự khi sử dụng
- Nhược điểm:
 - Chi phí cao
 - Hành vi của người sử dụng có thể thay đổi theo môi trường hệ thống



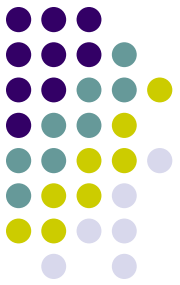
Tóm tắt

- Khái niệm lập lịch, các tiêu chí đánh giá thuật toán lập lịch
- Các phương thức hoạt động preemptive và non-preemptive
- Các thuật toán lập lịch FCFS, SJF, ưu tiên, RR
- Lập lịch với hàng chờ đa mức, có và không có phản hồi
- Các phương pháp đánh giá thuật toán lập lịch



Bài tập

- Thực hiện ví dụ RR với lượng tử thời gian 2ms, 6ms và 6ms.
 - Tính thời gian chờ trung bình của các tiến trình trong các trường hợp này.
 - Khi lượng tử thời gian thay đổi, thời gian chờ trung bình thay đổi thế nào?
 - Tính thời gian hoàn thành (turnaround time) của tất cả các tiến trình trong các trường hợp trên
 - Nhận xét về sự thay đổi thời gian hoàn thành của các tiến trình khi lượng tử thời gian thay đổi



Bài tập

- Hãy xây dựng một ví dụ về hiện tượng convoy effect sao cho thời gian rồi của thiết bị vào ra ít hơn thời gian rồi của CPU. Giả sử có 4 tiến trình, trong đó P là tiến trình “lớn”, Q_1 , Q_2 , Q_3 là các tiến trình “nhỏ” được nằm trong hàng chờ theo thứ tự: P , Q_1 , Q_2 , Q_3
- Tìm hai ví dụ trong thực tế về hàng chờ đa mức và hàng chờ đa mức có phản hồi và giải thích các ví dụ này.