Escalonamento de Processos

Escalonamento de processos

Multiprogramação → visa maximizar uso da CPU

Sempre que processos estão prontos → disputam CPU

Algoritmo de escalonamento: maneira de escolher o processo que vai executar

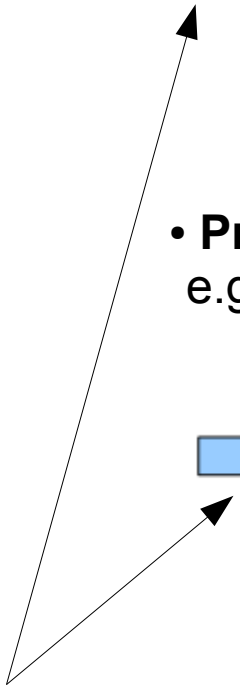
- Escalonador: parte do SO que decide qual processo será escalonado
- *Dispatcher*: parte do SO que transfere o controle da CPU para o processo escolhido pelo escalonador (envolve o chaveamento de contexto)

Trocar processos (chaveamento de contexto) é tarefa de alto custo

Comportamento dos processos

- **Processos orientados à CPU (*CPU-bound*):** muito uso da CPU e pouco uso de E/S
  e.g. programas de processamento científico

- **Processos orientados a E/S (*I/O-bound*):** muito uso de E/S e pouco uso da CPU
  e.g. banco de dados



Surtos de CPU
(ciclos de máquina)

Critérios de escalonamento

- **Maximizar**
  - uso CPU
  - *Throughput* (vazão): número de processos que completam sua execução por unidade tempo

- **Minimizar**
  - Tempo de *turnaround*: tempo total entre a criação de um processo e seu término
  - Tempo de espera: tempo que processo ficou esperando na fila prontos
  - Tempo de resposta: tempo que o processo demora para produzir alguma resposta a uma requisição (importante para processos interativos)

Algoritmos de escalonamento

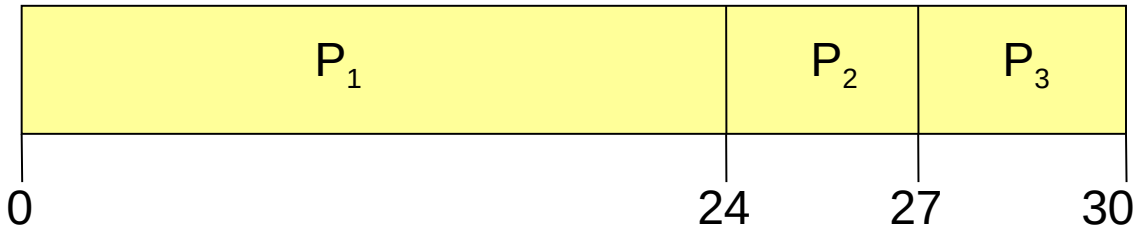
First-come, first-served (FCFS): primeiro a chegar, primeiro a ser servido (FIFO)

- Quando o processo deseja usar CPU, é colocado no fim da fila
- Processo só libera CPU quando é bloqueado → não preemptivo

Process	Duração surto
P_1	24
P_2	3
P_3	3

Supondo que processos cheguem na ordem: P_1 , P_2 , P_3

Diagrama de Gantt é:



Tempo de espera para P1=0; P2=24; P3=27

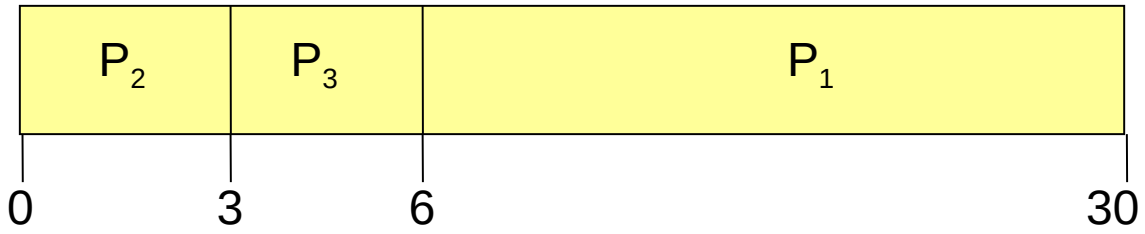
Tempo espera médio: (0 + 24 + 27)/3 = 17

FCFS (cont)

Supondo que processos cheguem na ordem:

P_2 , P_3 , P_1

Diagrama Gantt é:



Tempo espera para $P_1 = 6$; $P_2 = 0$, $P_3 = 3$

Tempo espera médio: (6 + 0 + 3)/3 = 3

Muito melhor que anterior!

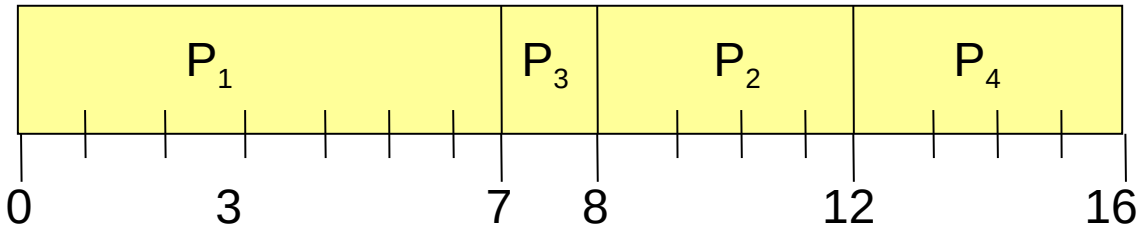
Algoritmos de escalonamento

***Shortest job first* (SJF)**: o processo mais curto primeiro

- Processo que tiver menor próximo surto CPU será escolhido
- É ótimo → produz tempo espera mínimo
- Dois esquemas:
  - Não-preemptivo: processo em execução não é interrompido até que termine seu surto
  - Preemptivo: se chegar novo processo com surto menor do que tempo restante processo atual, interrompe atual → *Shortest remaining time first* (SRTF)

SJF não-preemptivo

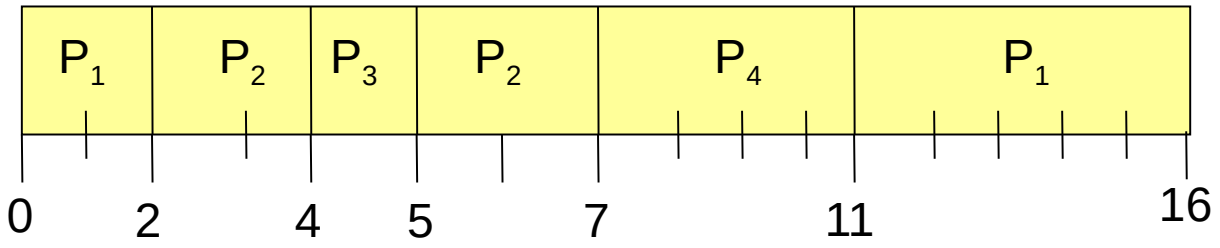
Processo	Tempo chegada	Duração surto
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



Tempo de espera para: P1=0-0=0, P2=8-6=2; P3=7-4=3; P4=12-5=7
Tempo espera médio = (0 + 6 + 3 + 7)/4 = 4

SJF preemptivo

Processo	Tempo chegada	Duração surto
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



Tempo de espera para:

P1 = (0-0)+(11-2) = 9
P2 = (2-2)+(5-4) = 1
P3 = (4-4) = 0
P4 = (7-5) = 2

Tempo espera médio = (9 + 1 + 0 +2)/4 = 3

Como saber tamanho próximo surto?

- Usuário informa ou sistema prevê?
  - Usuário informa: e.g. RTOS estático
    mas, muitas vezes, o usuário não dispõe desta informação
  - Sistema prevê → estimativa → média exponencial

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

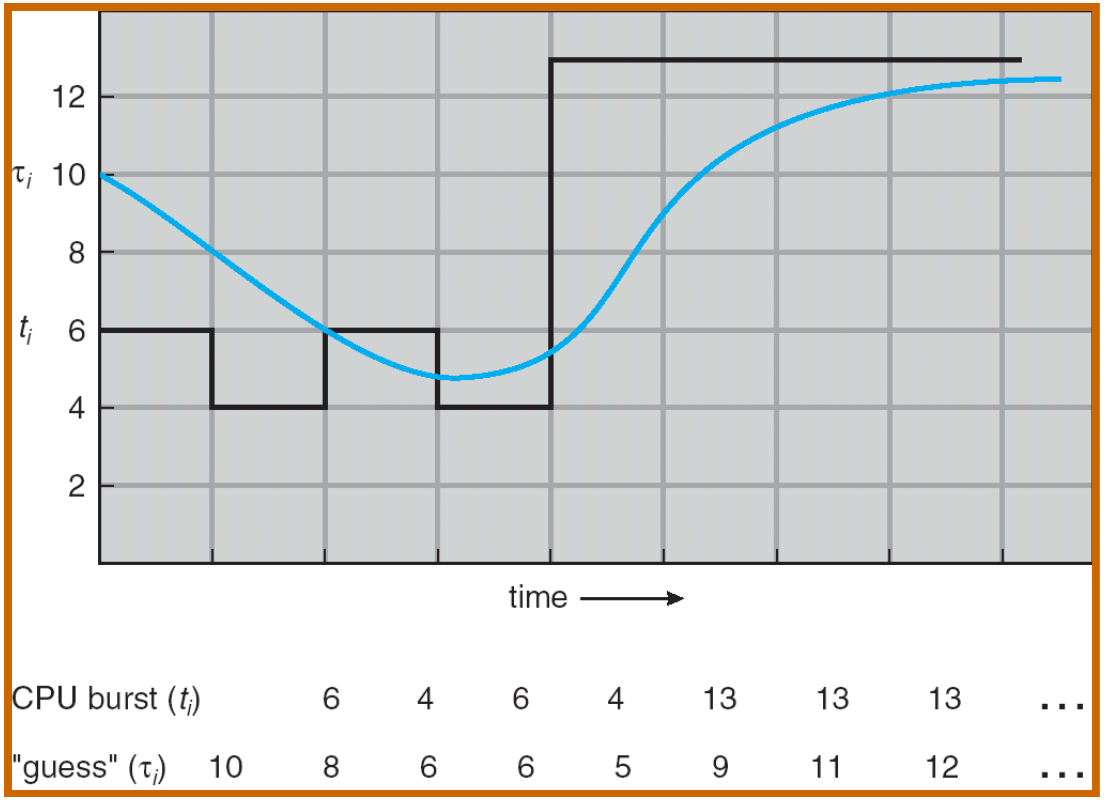
onde:

t_n: último surto de CPU

α: peso ($0 \le \alpha \le 1$)

τ_{n+1}: próximo surto previsto

τ_n: histórico de surtos anteriores



Exemplos do uso da média exponencial

- $\alpha = 0$

  $\tau_{n+1} = \tau_n \rightarrow$ não leva em conta o presente, mas apenas o passado (histórico)

- $\alpha = 1$

  $\tau_{n+1} = t_n \rightarrow$ leva em conta apenas o último surto CPU (presente)

- Se expandirmos a fórmula:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha).\tau_n$$
$$= \alpha\, t_n + (1 - \alpha).[\alpha\, t_{n-1} + (1 - \alpha).\tau_{n-1}]$$
$$= \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j \alpha\, t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

- Como α e $(1 - \alpha)$ variam entre 0 e 1 (inclusive), cada termo tem menos peso que seu predecessor → quanto mais antigo é o surto, menos peso ele tem na previsão

Algoritmos de escalonamento

***Round robin* (revezamento circular)**

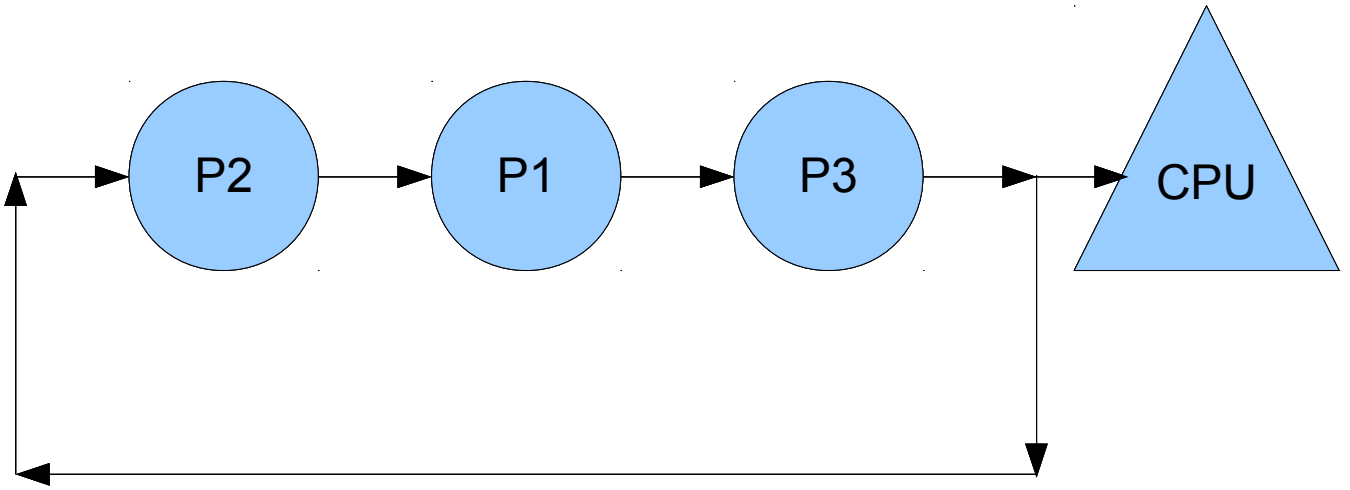
Para prevenir que processos de usuário monopolizem a CPU
introduz-se uma interrupção por tempo, conhecida como *quantum* ou *time-slice*
Tipicamente: 10 a 100ms

Um processo de usuário, uma vez interrompido por término do seu *quantum*, volta à fila de prontos e permite que outro processo seja escalonado

- Similar FCFS, mas com preempção
- *quantum* (q): pequena fatia tempo da CPU
- Se houver n processos na fila de prontos, cada processo obterá 1/n do tempo da da CPU em montantes de, no máximo, q unidades de cada vez. Nenhum processo espera mais do que (n-1).q
- Desempenho
  - q grande → FIFO
  - q pequeno → q deve ser grande em relação ao tempo chaveamento de contexto → *overhead* muito grande

Algoritmos de escalonamento

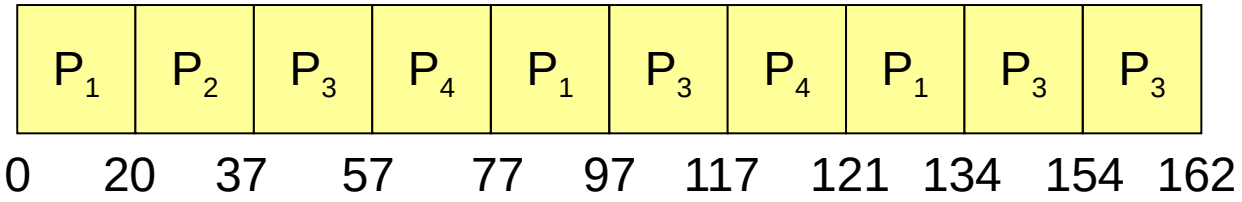
Round robin **(revezamento circular)**



Exemplo RR com q=20

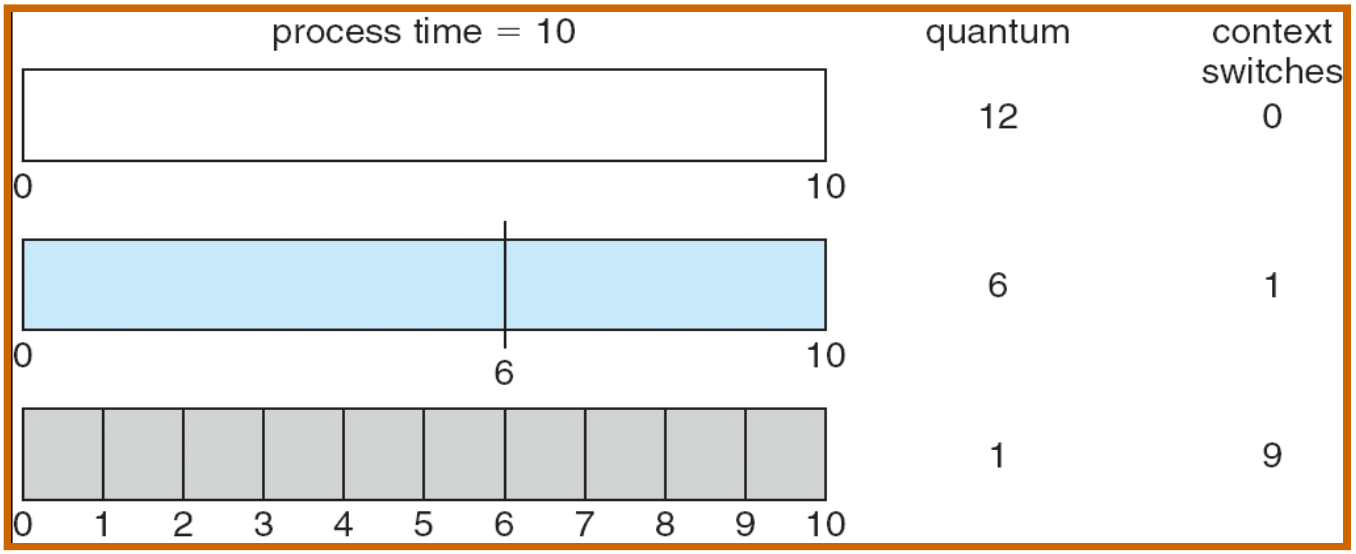
Processo	Duração surto
P_1	53
P_2	17
P_3	68
P_4	24

Diagrama de Gantt:

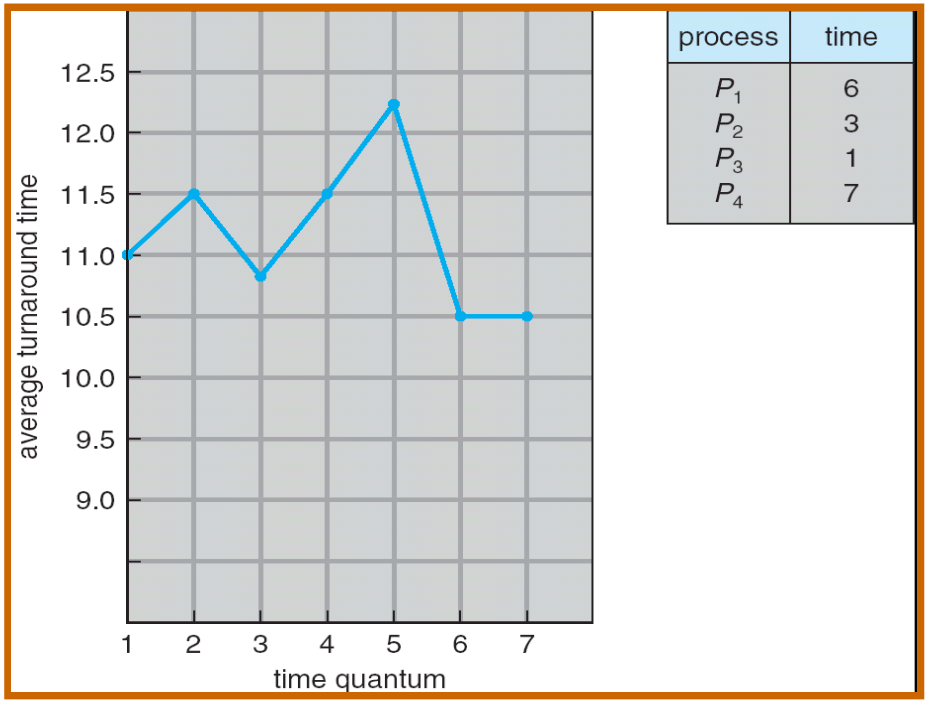


Tipicamente tem *turnaround* maior que SJF, mas resposta melhor

Relação entre o tempo de quantum e de chaveamento de contexto



Turnaround varia com o tempo quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

O tempo de *turnaround* de um conjunto de processos não necessariamente melhora com o aumento do *quantum*

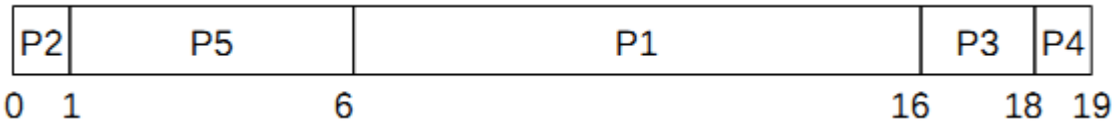
Exercício: Reproduza o gráfico ao lado.
Dicas:
- Todos os processos são criados em t=0
- O primeiro cálculo (q=1) fica:
  t = (3+9+15+17)/4 = 11

Algoritmos de escalonamento

Por prioridade

- Cada processo possui uma prioridade (número inteiro)
- A CPU é alocada ao processo maior prioridade
- SJF é um algoritmo por prioridade onde prio é o próx. surto CPU
- Valor *prio* depende SO
- Premptivo ou não
- Problema: *starvation* (bloqueio por tempo indefinido)
  - Lenda: quando IBM 7094 do MIT desligado 1973, havia processo baixa prio esperando desde 1967
  - Solução → *aging* (envelhecimento) e.g Linux

Processo	Duração do Surto	Prioridade
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2



Tempo espera médio = (0+1+6+16+18)/5 = 8,2ms

Bibliografia

[1] *Real-Time Systems and Programming Languages*. Burns A., Wellings A. 2nd edition

[2] Análise de Sistemas Operacionais de Tempo Real Para Applicações de Robótica e Automação. Aroca R. V. Dissertação de Mestrado.

[3] *Operating System Concepts*. Silberschatz, Galvin, Gagne. 8th edition

[4] Sistemas Operacionais Modernos. Tanenbaum 2a edição