



Laboratório de Pesquisa em Redes e Multimídia

Processos

Escalonamento de Processos



Universidade Federal do Espírito Santo
Departamento de Informática

Objetivos do Escalonamento

- **Maximizar a taxa de utilização da UCP.**
- Maximizar a vazão ("*throughput*") do sistema.
- Minimizar o **tempo de execução** ("*turnaround*").
 - *Turnaround*: tempo total para executar um processo.
- Minimizar o **tempo de espera** ("*waiting time*):"
 - *Waiting time*: tempo de espera na fila de prontos.
- Minimizar o **tempo de resposta** ("*response time*").
 - *Response time*: tempo entre requisição e resposta.

Algoritmos de Escalonamento

- Vários algoritmos podem ser empregados na definição de uma estratégia de escalonamento.
- Os algoritmos buscam:
 - Obter bons tempos médios ao invés de maximizar ou minimizar um determinado critério.
 - Privilegiar a variância em relação a tempos médios.
- As políticas de escalonamento podem ser:
 - Preemptivas;
 - Não-preemptivas.

Políticas de Escalonamento

■ Preemptivas:

- O processo de posse da UCP pode perdê-la a qualquer momento, na ocorrência de certos eventos, como fim de fatia de tempo, processo mais prioritário torna-se pronto para execução, etc.
- Não permite a monopolização da UCP.

■ Não-Preemptivas:

- O processo em execução só perde a posse da UCP caso termine ou a devolva deliberadamente, isto é, uma vez no estado *running*, ele só muda de estado caso conclua a sua execução ou bloqueie a si mesmo emitindo, p.ex., uma operação de E/S.

Exemplos de Algoritmos

- FIFO (First-In First-Out) ou FCFS (First-Come First-Served).
- SJF (Shortest Job First) ou SPN (Shortest Process Next).
- SRTF (Shortest Remaining Time First).
- HRRN (Highest Response Rate Next);
- Round-Robin.
- Priority.
- Multiple queue

First-Come First-Served ⁽¹⁾

- Algoritmo de baixa complexidade.
- Exemplo de abordagem não-preemptiva.
- Processos que se tornam aptos para execução são inseridos no final da fila de prontos.
- O primeiro processo da fila é selecionado para execução.
- O processo executa até que:
 - Termina a sua execução;
 - Realiza uma chamada ao sistema.

First-Come First-Served (2)

- Processos pequenos podem ter que esperar por muito tempo, atrás de processos longos, até que possam ser executados ("*convoy effect*").
- Favorece processos CPU-bound.
 - Processos I/O-bound têm que esperar até que processos CPU-bound terminem a sua execução.
- Algoritmo particularmente problemático para sistemas de tempo compartilhado, onde os usuários precisam da CPU a intervalos regulares.

First-Come First-Served ⁽³⁾

<u>Process</u>	<u>Burst Time</u>
----------------	-------------------

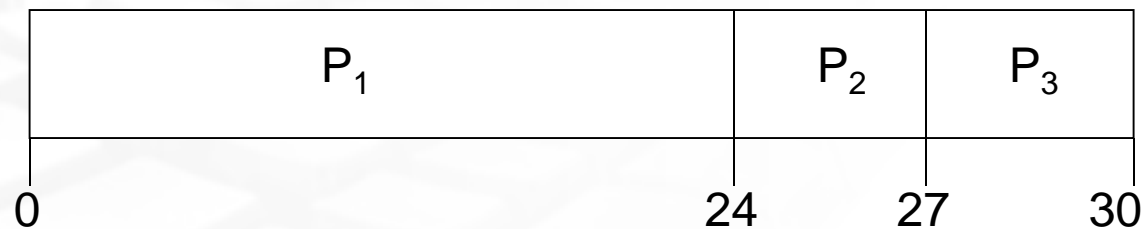
P_1	24
-------	----

P_2	3
-------	---

P_3	3
-------	---

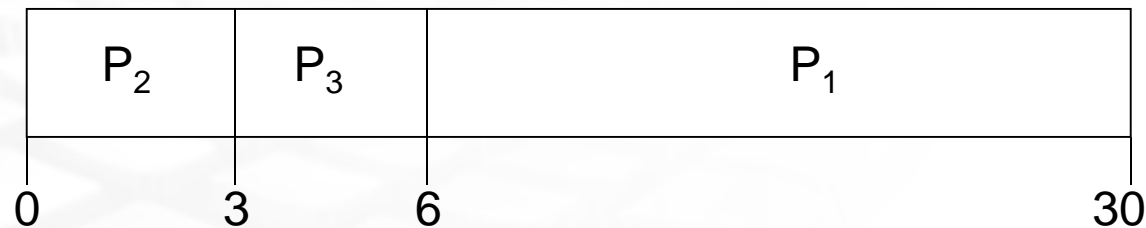
- Suponha que os esses processos cheguem na seguinte ordem:
 - P_1, P_2, P_3

First-Come First-Served (4)



- Tempo de espera para cada processo:
 - *Waiting time: $P_1 = 0$; $P_2 = 24$; $P_3 = 27$*
- Tempo médio de espera:
 - *Average waiting time: $(0 + 24 + 27)/3 = 17$*

First-Come First-Served (5)



- Suponha que os mesmos processos cheguem agora na seguinte ordem:
 - P_2, P_3, P_1
- Tempo de espera de cada processo:
 - *Waiting time:* $P_1 = 6; P_2 = 0; P_3 = 3$
- Tempo médio de espera:
 - *Average waiting time:* $(6 + 0 + 3)/3 = 3$

Shortest Job First ⁽¹⁾

- Baseia-se no fato de que privilegiando processos pequenos o tempo médio de espera decresce.
 - O tempo de espera dos processos pequenos decresce mais do que o aumento do tempo de espera dos processos longos.
- É um algoritmo ótimo, de referência.

Shortest Job First (2)

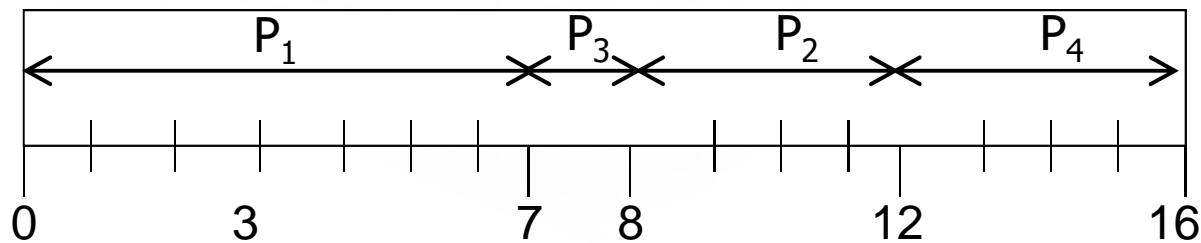
- Abordagem 1:
 - Processo com menor expectativa de tempo de processamento é selecionado para execução.
- Abordagem 2:
 - Associado com cada processo está o tamanho do seu próximo *CPU burst*.
 - Esse tamanho é usado como critério de escalonamento, sendo selecionado o processo de menor próximo *CPU burst*.

Shortest Job First ⁽³⁾

- Dois esquemas:
 - Não-preemptivo – uma vez a CPU alocada a um processo ela não pode ser dada a um outro antes do término do CPU burst corrente.
 - Preemptivo – se chega um novo processo com CPU burst menor que o tempo remanescente do processo corrente ocorre a preempção. Esse esquema é conhecido como Shortest-Remaining-Time-First (SRTF).

Exemplo de SJF Não-Preemptivo

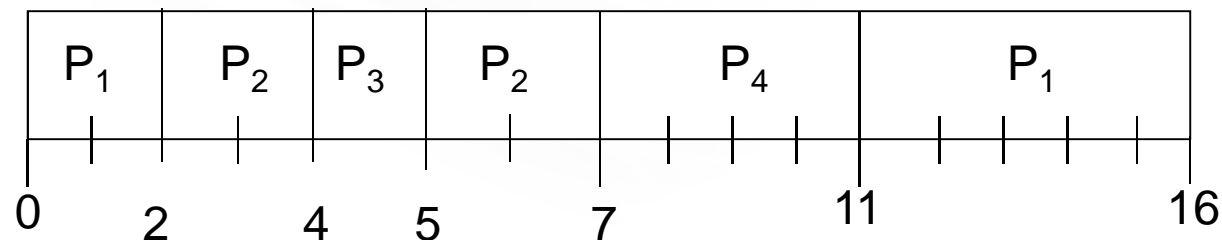
<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



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

Exemplo de SJF Preemptivo (Algoritmo SRTF)

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



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

Tamanho do Próximo *CPU burst*

- A real dificuldade do algoritmo é conhecer o tamanho da próxima requisição de CPU.
 - Para escalonamento de longo prazo num sistema batch, podemos usar como tamanho o limite de tempo de CPU especificado pelo usuário quando da submissão do job.
 - No nível de escalonamento de curto prazo sua implementação pode ser apenas aproximada, já que não há como saber o tamanho da próxima requisição de CPU.

Previendo o Tamanho do *Burst* (1)

- Uma maneira de se aproximar do SJF é *prevendo* o tamanho do próximo CPU burst.
- Normalmente isso é feito usando uma média exponencial das medidas dos *bursts* anteriores.

1. t_n = actual length of n^{th} CPU burst

2. τ_{n+1} = predicted value for the next CPU burst

3. $\alpha, 0 \leq \alpha \leq 1$

4. Define :

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

Previendo o Tamanho do *Burst* (2)

- O parâmetro α controla o peso relativo do histórico recente e passado na fórmula.
 - Se $\alpha = 0$
 - $\tau_{n+1} = \tau_n$
 - Histórico recente não é considerado relevante.
 - Se $\alpha = 1$
 - $\tau_{n+1} = t_n$
 - Apenas o último CPU burst é levado em conta.

Previendo o Tamanho do *Burst* (3)

- Se a fórmula for expandida, teremos:

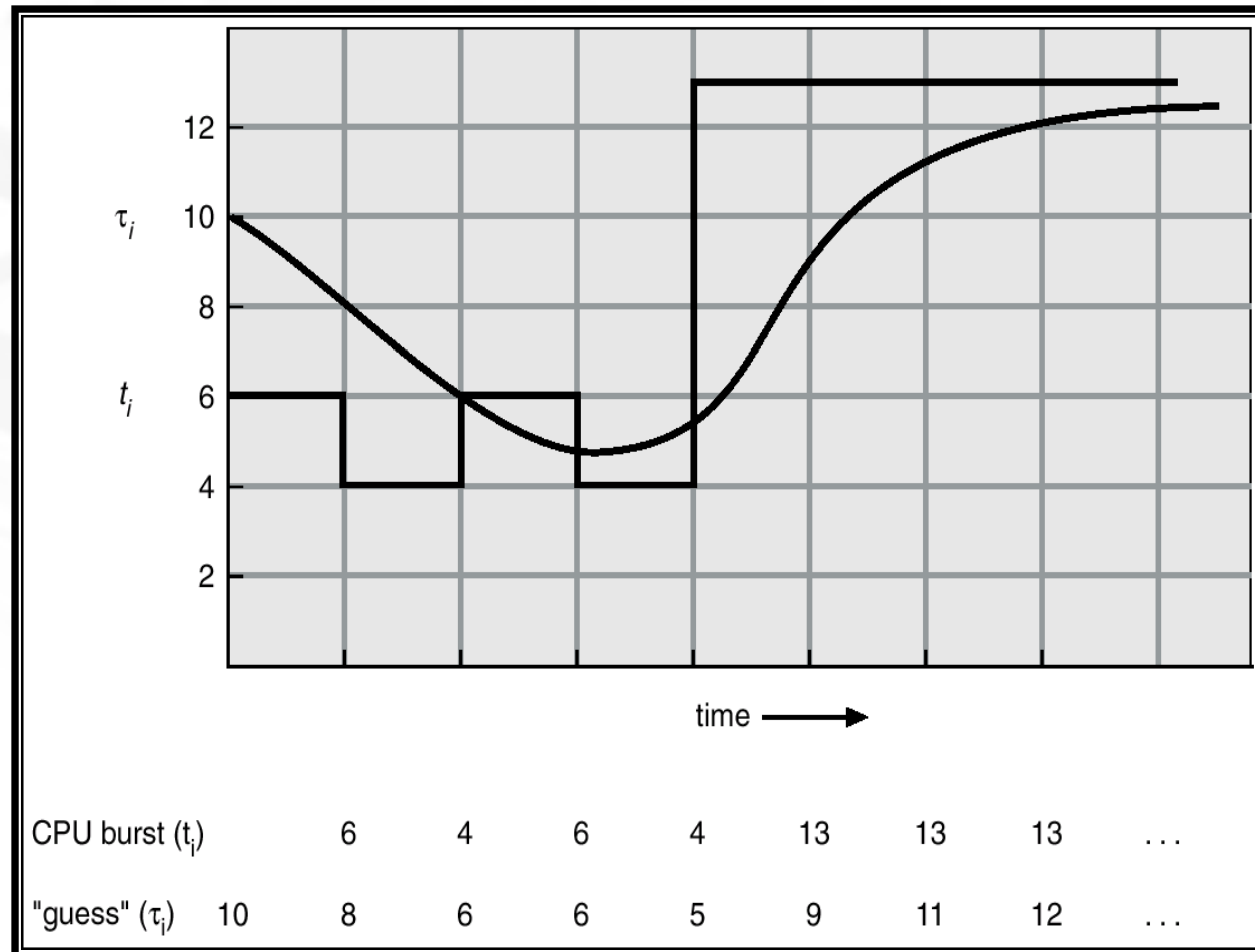
$$\begin{aligned}\tau_{n+1} &= \alpha t_n + (1 - \alpha) \alpha t_{n-1} + \dots \\ &\quad + (1 - \alpha)^j \alpha t_{n-j} + \dots \\ &\quad + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

- Como tanto α quanto $(1 - \alpha)$ são menores ou iguais a 1, cada termo sucessivo possui menos peso que o seu predecessor.

Previendo o Tamanho do *Burst* (4)

$$\alpha = 1/2$$

$$\tau_0 = 10$$



Escalonamento por Prioridade (1)

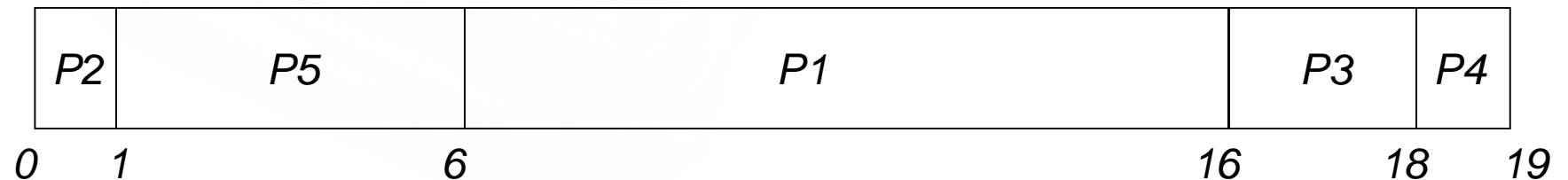
- Um número inteiro é associado a cada processo, refletindo a sua prioridade no sistema.
- A CPU é alocada ao processo de maior valor de prioridade na fila de prontos.
 - OBS: normalmente, menor valor = maior prioridade
- Estratégia muito usada em S.O. de tempo real.
- Problema: "*starvation*"
 - Processos de baixa prioridade podem nunca executar
- Solução: "*Aging*"
 - Prioridade aumenta com o passar do tempo.

Escalonamento por Prioridade (2)

- Prioridades podem ser definidas interna ou externamente.
 - Definição interna:
 - Usa alguma medida (ou uma combinação delas) para computar o valor da prioridade. Por exemplo, limite de tempo, requisitos de memória, n° de arquivos abertos, razão entre *average I/O burst* e *average CPU burst*, etc.
 - Definição externa:
 - Definida por algum critério externo ao S.O (tipo do processo, departamento responsável, custo, etc.)

Escalonamento por Prioridade (3)

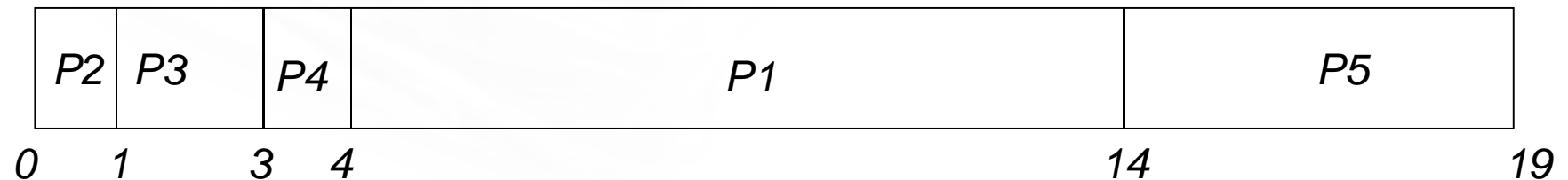
<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



$$\text{Average waiting time} = (6+0+16+18+1)/5 = 8,2\text{ms}$$

Escalonamento por Prioridade (3)

<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>
P_1 background	10	1
P_2 Interativo	1	0
P_3 Interativo	2	0
P_4 Interativo	1	0
P_5 background	5	1



$$\text{Average waiting time} = (4+0+1+3+14)/5 = 4,4\text{ms}$$

Escalonamento *Round-Robin* (1)

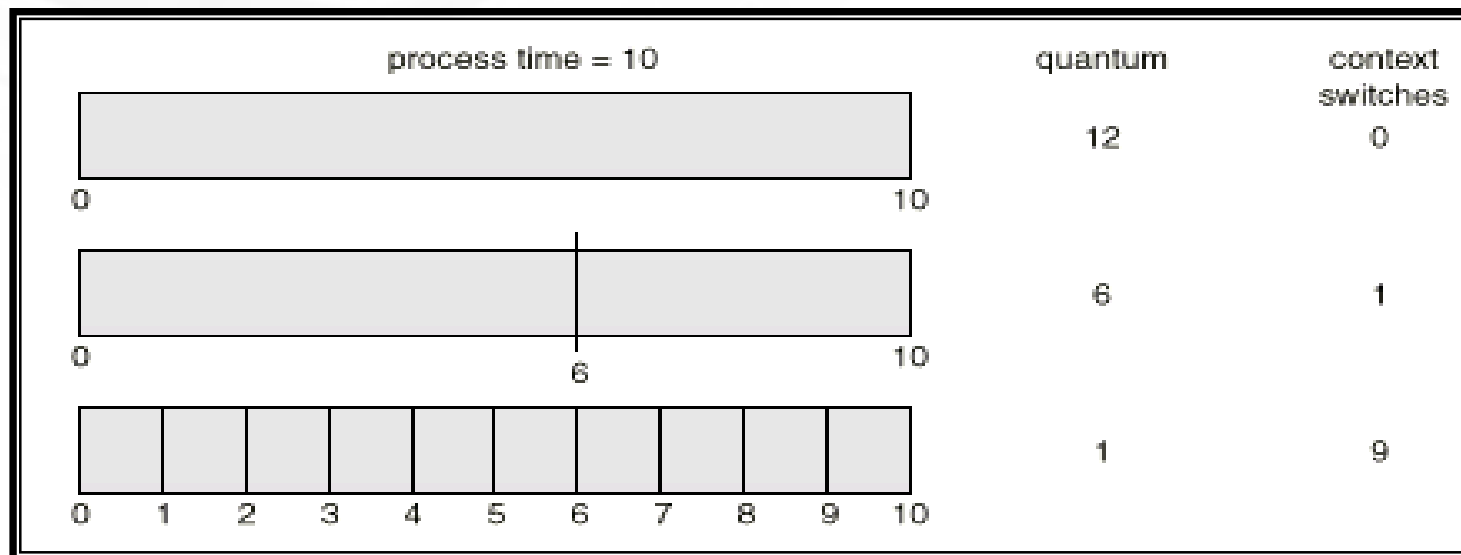
- Algoritmo típico de sistemas operacionais de tempo compartilhado.
- Cada processo recebe uma pequena fatia de tempo de CPU (quantum), usualmente entre 10 e 100 ms.
- Após o término da sua fatia de tempo o processo é “interrompido” e colocado no final da fila de prontos (“preempção” baseada na interrupção de relógio).
- É um algoritmo justo???

Escalonamento *Round-Robin* (2)

- Se n processos existem na fila de prontos e a fatia de tempo é q , então cada processo recebe $1/n$ do tempo de CPU, em fatias de q unidades de tempo de cada vez.
- Nenhum processo espera mais do que $(n-1).q$ unidades de tempo.
 - Qual a relação $c/$ o tempo de resposta?
- Tipicamente, apresenta um tempo de *turnaround* médio maior que o SJF, por que?

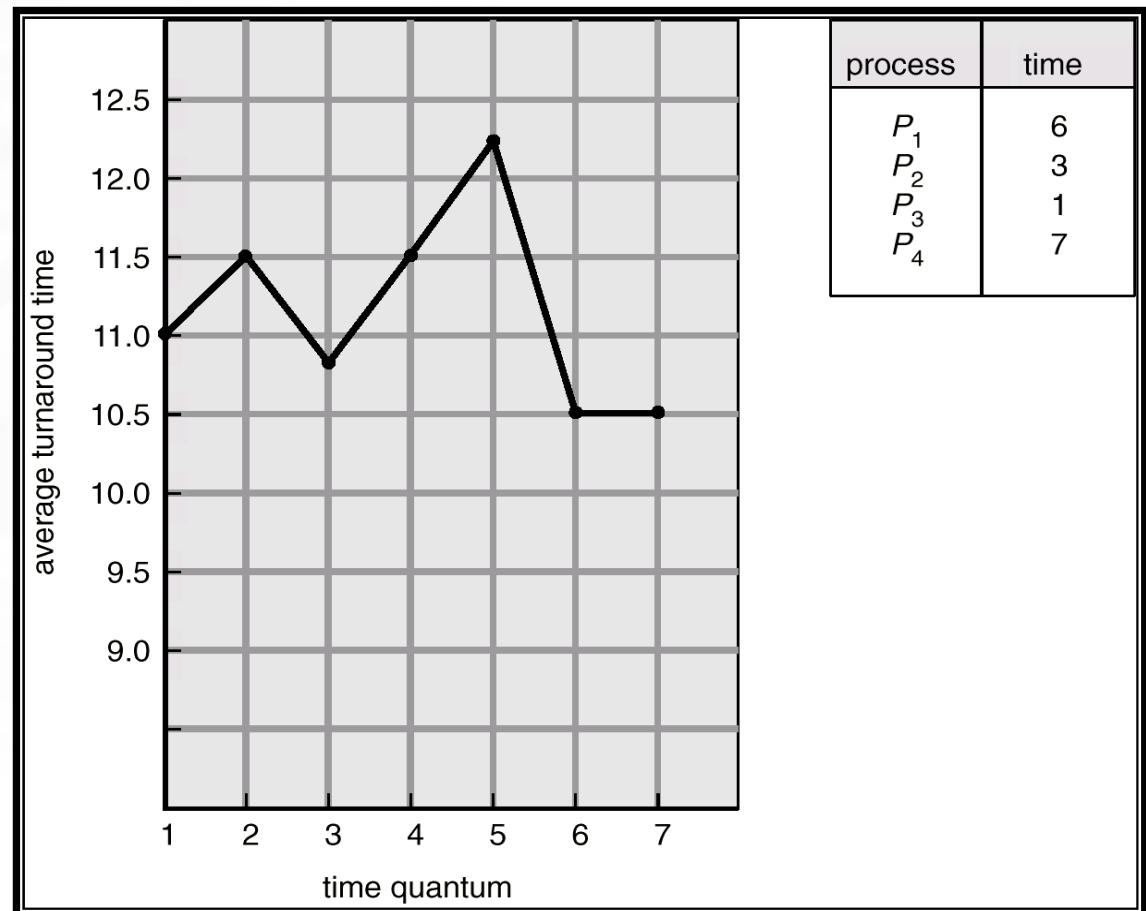
Desempenho do Algoritmo

- Dependente do tamanho do quantum:
 - q grande \Rightarrow tende a FIFO.
 - q pequeno \Rightarrow gera muito *overhead* devido às trocas de contexto.



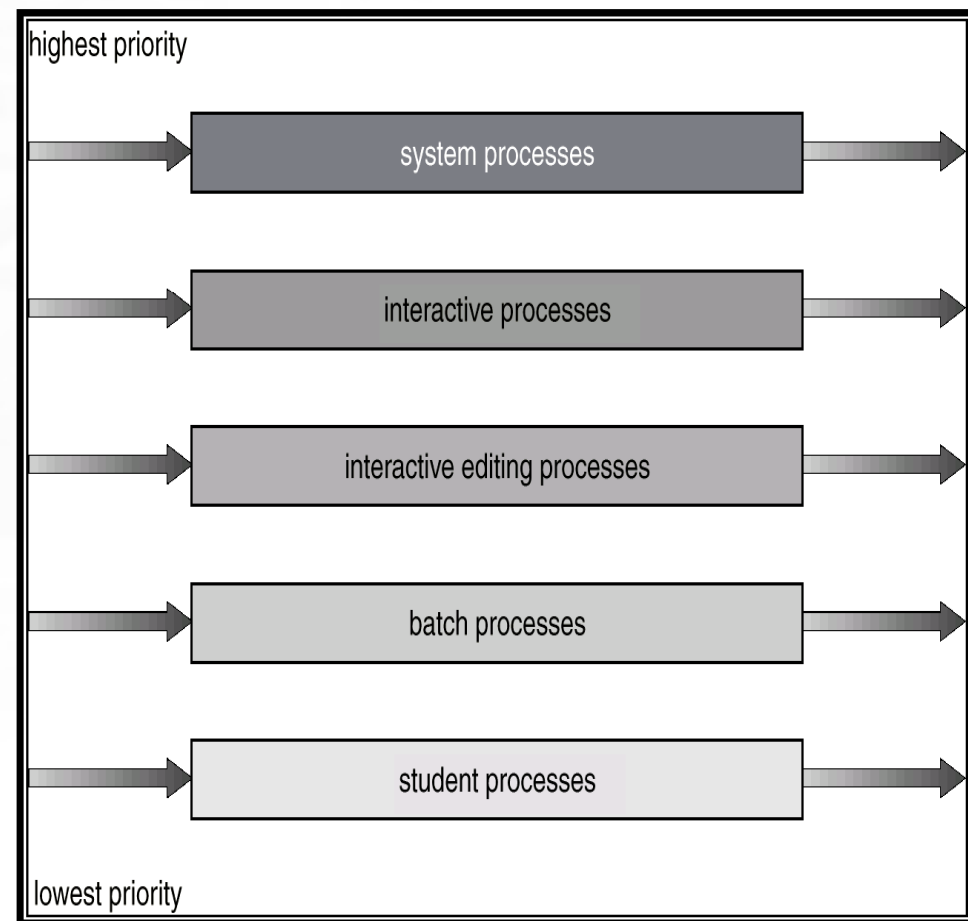
Turnaround x Tamanho do Quantum

- O tempo de *turnaround* varia com o tamanho do quantum.
- O *turnaround* de um conjunto de processos não necessariamente diminui quando o tamanho do quantum é aumentado.
- Regra geral: 80% CPU burst $< q$



Escalonamento Multinível (1)

- A idéia base é dividir os processos em diferentes grupos, com diferentes requisitos de tempos de resposta.
- A cada grupo é associada uma fila de prioridade, conforme a sua importância .



Escalonamento Multinível (2)

- Por exemplo, a fila de prontos pode ser dividida em duas filas separadas:
 - *foreground* (p/ processos interativos)
 - *background* (p/ processamento *batch*)
- Cada fila apresenta o seu próprio algoritmo de escalonamento:
 - *foreground* – RR
 - *background* – FCFS

Escalonamento Multinível (3)

- Escalonamento deve ser feito entre as filas:
 - Prioridades fixas – atende primeiro aos processos da fila *foreground* e somente depois aos da fila *background*.
 - Time slice – cada fila recebe uma quantidade de tempo de CPU para escalonamento entre os seus processos. Ex: 80% para *foreground* em RR e 20% para *background* em FCFS.

Escalonamento Multinível com Feedback

- O processo pode se mover entre as várias filas. Deste modo, a estratégia de *aging* pode ser implementada.
- O escalonador trabalha com base nos seguintes parâmetros:
 - Número de filas;
 - Algoritmo de escalonamento de cada fila;
 - Método usado para determinar quando aumentar e quando reduzir a prioridade do processo;
 - Método usado para se determinar em que fila o processo será inserido.

Exemplo ⁽¹⁾

- Suponha a existência de 3 filas:
 - Q_0 – time quantum 8 milliseconds
 - Q_1 – time quantum 16 milliseconds
 - Q_2 – FCFS
- Escalonamento:
 - Um job novo entra na fila Q_0 , que é servida segundo a estratégia RR. Quando ele ganha a CPU ele recebe 8 ms. Se não terminar em 8 ms, o job é movido para a fila Q_1 .
 - Em Q_1 o job é novamente servido RR e recebe 16 ms adicionais. Se ainda não completar, ele é interrompido e movido para a fila Q_2 .
 - Em Q_2 , FCFS

Exemplo (2)

