**BC1518 - Sistemas Operacionais**

# Aula 6: Escalonamento de CPU

# Tópicos desta aula

- Conceitos Básicos
- Critérios de Escalonamento
- Algoritmos de Escalonamento
- Escalonamento de Threads Java

# Conceitos básicos

- Multiprogramação
  - Múltiplos processos na memória compartilhando o uso da CPU
  - Objetivo: ter sempre um processo executando na CPU → **Maximizar a utilização da CPU**
  - Requer o gerenciamento do processador

- Escalonamento de processos
  - Havendo vários processos para serem executados (prontos) é preciso estabelecer critérios para selecionar um deles para utilizar a CPU
  - É função do escalonador de processos decidir qual será o próximo processo a executar na CPU – define a ordem de execução dos processos na fila de prontos
  - Função fundamental do Sistema Operacional, é a base do gerenciamento de processador e da multiprogramação

# Conceitos básicos

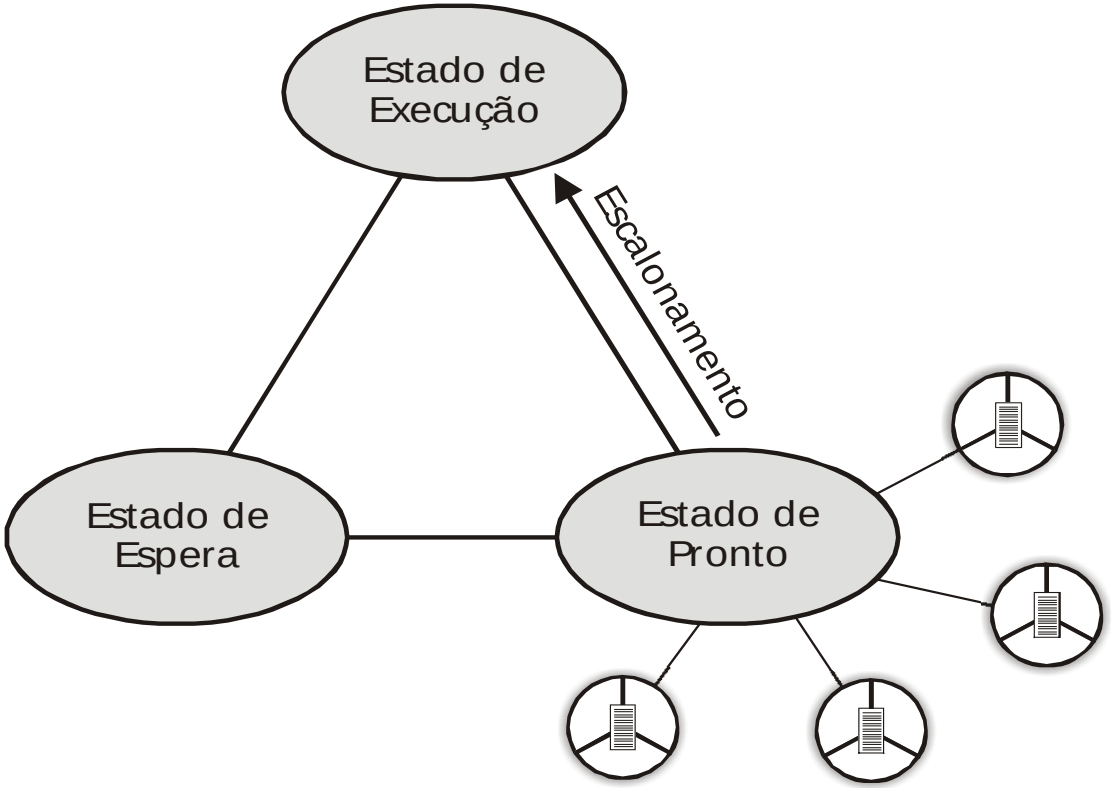


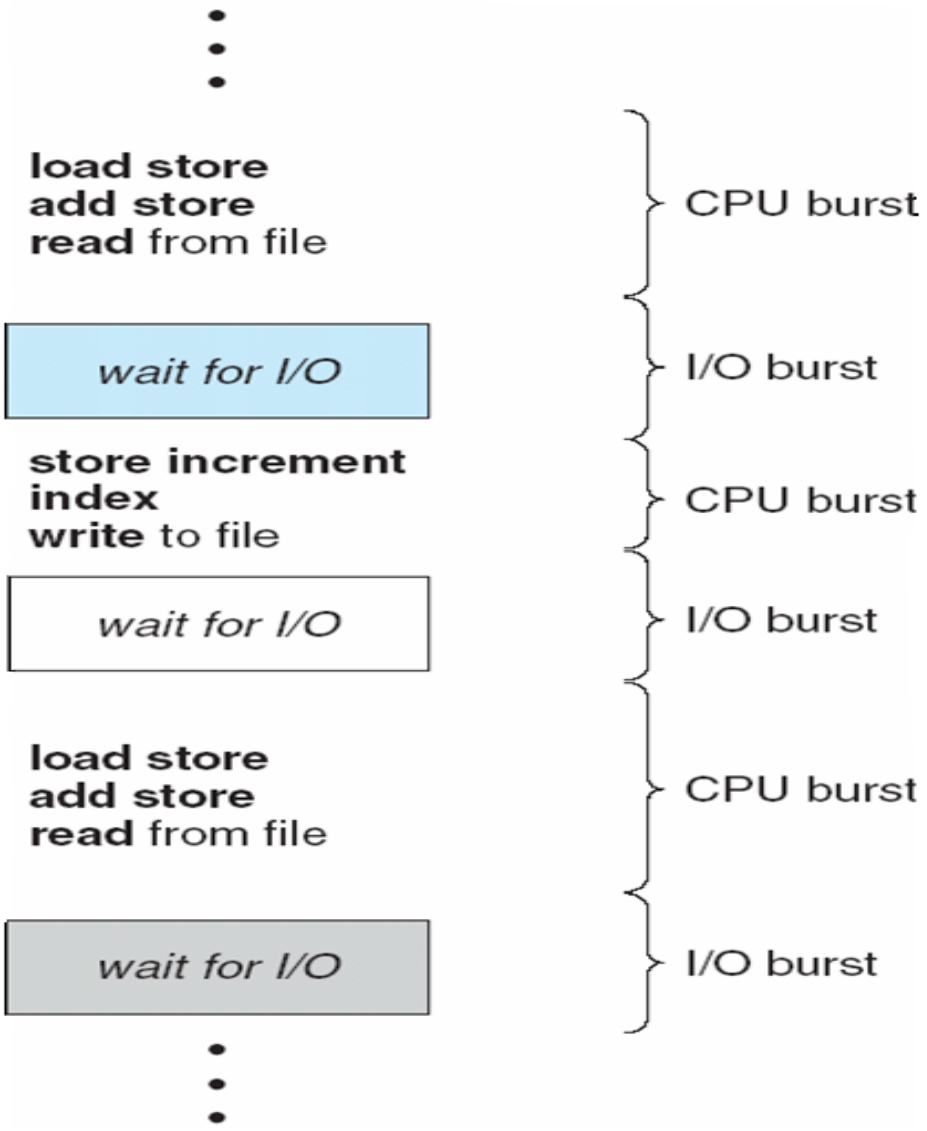

Ilustração de escalonamento [Machado]

# Conceitos básicos

- A execução de processos consiste de: execução de CPU (computação) e espera de E/S (operação de E/S)
- A execução de um processo alterna entre esses dois estados: execução de CPU (*burst* ou surto de uso da CPU) e espera de E/S (*burst* ou surto de E/S)
  - Consiste em um ciclo de surto de uso da CPU e surto de E/S
  - (Início) → surto de CPU → surto de E/S → surto de CPU → surto de E/S → … → surto de CPU (Fim)
  - Exemplo:

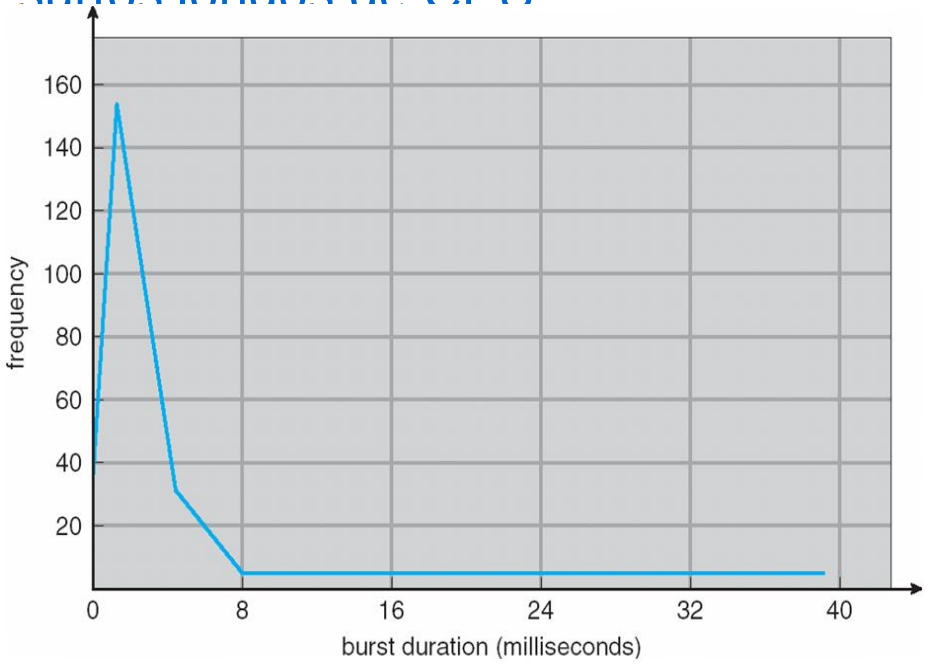
```
BEGIN                       }
A = 2                       }
B = A + 5                   surto de uso da CPU (CPU burst)
READ C                      espera de E/S (I/O burst)
B = A + C                   }
D = (A * B) - C             surto de uso da CPU
E = D / C
WRITE A, B, C, D, E         espera de E/S
STOP                        termina a execução
```

# Sequência de troca de surtos de CPU e E/S




Alternando a sequência de surtos CPU-E/S [Silberschatz]

- As durações dos surtos de CPU foram medidas exaustivamente
  - Variam de um processo para outro, de um computador para outro
  - Em geral há uma grande quantidade de surtos curtos de CPU e uma menor quantidade de surtos longos de CPU



# Comportamento escalonamento-processo

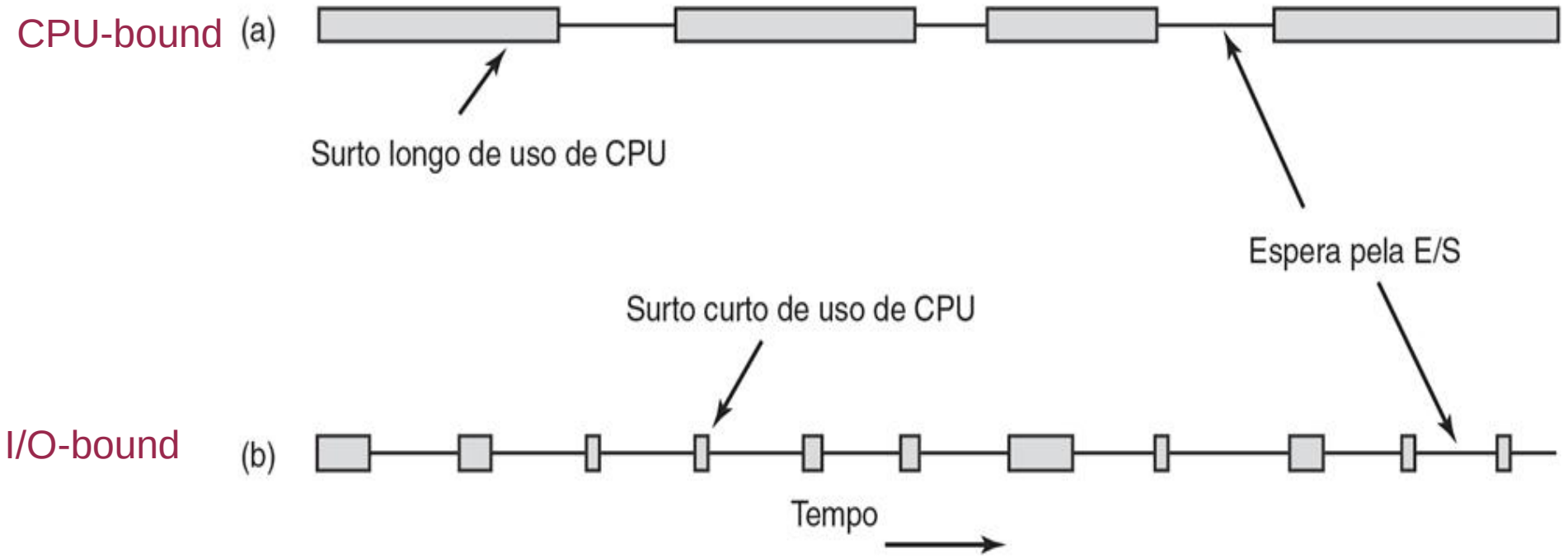



**Figura 2.31** Usos de surtos de CPU se alternam com períodos de espera por E/S. (a) Um processo orientado à CPU. (b) Um processo orientado à E/S. [Tanenbaum]

(a) Processos *CPU-bound*: gastam a maior parte do tempo computando, possuem longos surtos de uso da CPU e esporádicas esperas por E/S

(b) Processos *I/O-bound*: passam a maior parte do tempo aguardando por E/S, possuem pequenos surtos de utilização da CPU e frequentes esperas por E/S, ou seja pouca utilização da CPU

# Escalonador de CPU

➢O escalonador de curto prazo (ou escalonador de CPU) seleciona dentre os processos na memória aqueles que estão **prontos para executar** e **aloca a CPU a um deles**

➢As decisões de escalonamento de CPU podem ocorrer em **quatro circunstâncias**:

- ➢ 1. Um processo passa do estado **executando para esperando**
- ➢ 2. Um processo passa do estado **executando para pronto**
- ➢ 3. Um processo passa do estado **esperando para pronto**
- ➢ 4. U


Diagrama de estado do processo [Silberschatz]

# Tipos de Escalonamento

➢O escalonamento pode ser:

- **Não-preemptivo**
  - Nenhum evento externo pode provocar a perda do uso da CPU por um processo em execução
  - O escalonamento ocorre somente nas situações (1) e (4)

- **Preemptivo**
  - Um processo pode ser interrompido para a execução de outro
  - O escalonamento pode ocorrer em qualquer uma das situações (1) a (4)

# Escalonamento Não-preemptivo

## ➢Não-preemptivo

- Primeiro tipo de escalonamento implementado nos sistemas multiprogramáveis onde predominava o processamento *batch*
- Depois que a CPU foi alocada a um processo, este processo executa ininterruptamente até que:
  - O processo termina a sua execução ou
  - O processo entra para o estado de espera (op. E/S) ou
  - O processo libera voluntariamente a CPU
- Também conhecido como **escalonamento cooperativo, cada processo precisa ceder o controle voluntariamente** para que outros possam executar
- Tem a vantagem de tornar o projeto do SO muito mais simples
- Era usado nos sistemas mais antigos como Windows 3.x

# Tipos de Escalonamento

## ➢Preemptivo

- Um **processo pode ser interrompido para a execução de outro**
- Amplamente utilizado em **sistemas de tempo compartilhado**
  - Um processo executa na CPU por um tempo máximo fixado
  - Ao final desse tempo, o processo é suspenso e o escalonador seleciona um outro processo para executar
- É possível priorizar a execução de processos
  - O processo em execução pode ser interrompido caso um outro processo (mais importante ou mais urgente) entre na fila de processos prontos e esse possa ser escalonado
- Utilizado na maioria dos SOs atuais (a partir do Windows 95, UNIX, Linux), ênfase em interatividade, em satisfazer as necessidades imediatas dos usuários interativos
- Permite a implementação de políticas de escalonamento que compartilhem a CPU de maneira mais uniforme, distribuindo o uso da CPU de forma balanceada entre os processos

# Dispatcher (Executor ou Despachante)

➢Um outro componente envolvido na função de escalonamento de CPU é o **despachante (*dispatcher*)**

➢O ***dispatcher* passa o controle da CPU ao processo selecionado** pelo escalonador de curto prazo; envolve:

- Mudança de contexto
- Comutação para o modo usuário
- Desvio para a posição adequada no programa do usuário de modo a reiniciá-lo

➢O ***dispatcher*** precisa ser o **mais veloz possível**, pois é invocado a cada troca de processo

➢***Latência de despacho*** – tempo necessário para o *dispatcher* interromper um processo e iniciar a execução de outro

# Critérios de Escalonamento

➢Para distintos ambientes são necessários diferentes algoritmos de escalonamento, que melhor possam atender seus objetivos e requisitos

- **Sistemas interativos**
  - Requerem respostas rápidas a eventos (dos usuários, rede)
  - Editores de texto, navegadores Web, jogos, servidores (e-mail,...)
- **Sistemas em lote (*batch*)**
  - Normalmente executam sem intervenção do usuário (procedimentos de *backup*, cálculos numéricos longos, etc.)
  - Não há restrição de tempo
- **Sistemas de tempo real**
  - Há restrição de tempo, o escalonamento deve priorizar os processos críticos em detrimento de outros

# Critérios de Escalonamento

➢Que critérios usar para comparar diferentes algoritmos de escalonamento?

- **Utilização da CPU:** desejável que a CPU esteja ocupada a maior parte do tempo possível; (varia de 0 a 100%, na teoria); (em sistemas reais, de 40 a 90%)
- ***Throughput* (vazão):** número de processos completados por unidade de tempo (menor para processos curtos, maior para processos longos)
- **Tempo de retorno (*turnaround*)** – intervalo entre a submissão de um processo até seu tempo de conclusão (soma dos tempos gastos esperando na fila de prontos, executando na CPU e realizando E/S)
- **Tempo de espera** – tempo que um processo gasta esperando na fila de processos prontos (escalonamento afeta apenas o tempo que um processo permanece na fila de prontos)
- **Tempo de resposta** – tempo entre a submissão de um pedido até a primeira resposta ser produzida (não é o tempo que leva para gerar a resposta completa) – interessante para sistemas interativos

# Critérios de Otimização

- Justiça
  - Distribuição justa do processador entre os processos prontos na fila (processos semelhantes devem ter tempos de CPU semelhantes)
- Em geral, qualquer algoritmo de escalonamento busca:

| | |
|---|---|
| Maximizar | utilização da CPU |
| | *throughput* (vazão) |
| Minimizar | tempo de retorno |
| | tempo de espera |
| | tempo de resposta |

# Objetivos do algoritmo de escalonamento

**Todos os sistemas**

- Justiça — dar a cada processo uma porção justa da CPU
- Aplicação da política — verificar se a política estabelecida é cumprida
- Equilíbrio — manter ocupadas todas as partes do sistema

**Sistemas em lote**

- Vazão (*throughput*) — maximizar o número de tarefas por hora
- Tempo de retorno — minimizar o tempo entre a submissão e o término
- Utilização de CPU — manter a CPU ocupada o tempo todo

**Sistemas interativos**

- Tempo de resposta — responder rapidamente às requisições
- Proporcionalidade — satisfazer às expectativas dos usuários

**Sistemas de tempo real**

- Cumprimento dos prazos — evitar a perda de dados
- Previsibilidade — evitar a degradação da qualidade em sistemas multimídia

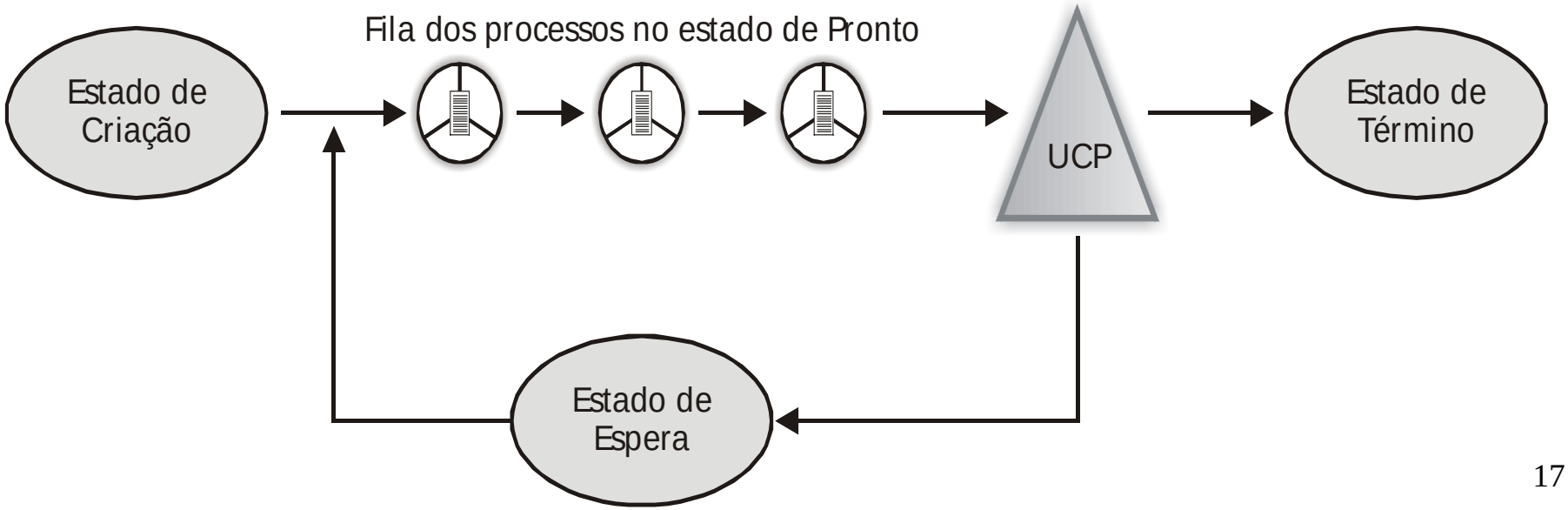
**Tabela 2.8** Alguns objetivos do algoritmo de escalonamento sob diferentes circunstâncias.



[Tanenbaum]

# Escalonamento First-Come, First-Served (FCFS)
## (Primeiro a chegar, primeiro a ser atendido)

- Este é o mais simples dos algoritmos de escalonamento, a CPU é atribuída aos processos na ordem requisitada por eles
- A implementação utiliza uma fila FIFO (*First In, First Out*) – Fila de processos prontos
  - Processos que entram para o estado **pronto** entram para o final da fila
  - Quando a CPU fica livre, o primeiro processo na fila é escalonado e é removido da fila




Escalonamento FCFS (ou FIFO) [Machado]

# Escalonamento FCFS (cont.)

| Processo | Duração de Surto |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

- Se os processos chegarem na ordem: $P_1$ , $P_2$ , $P_3$

O **Diagrama de Gantt** para este escalonamento é:


- Tempo de espera para $P_1$ = 0; $P_2$ = 24; $P_3$ = 27
- Tempo de espera médio: (0 + 24 + 27)/3 = 17

# Escalonamento FCFS (cont.)

Suponha que os processos cheguem na ordem:

$P_2$ , $P_3$ , $P_1$

- O **Diagrama de Gantt** para este escalonamento é:


- Tempo de espera para $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Tempo de espera médio: (6 + 0 + 3)/3 = 3
- Muito melhor que o caso anterior
- ***Efeito comboio*** – os processos pequenos ficam prejudicados, esperando que um grande processo saia da CPU
- **Não-preemptivo**

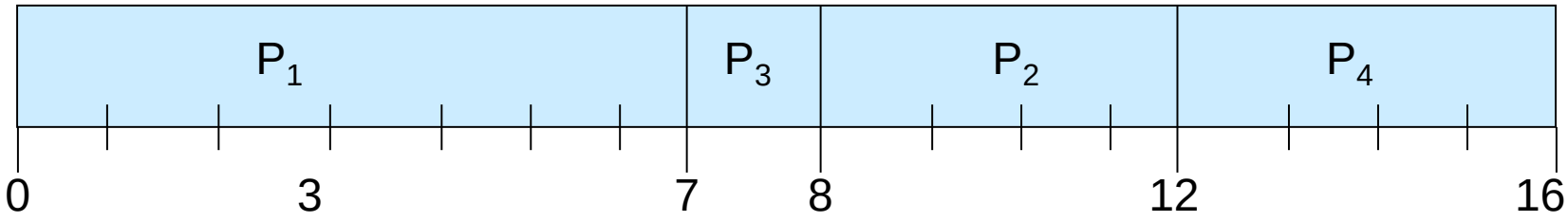
# Escalonamento job mais curto primeiro (Shortest-Job-First - SJF)

- Associa a cada processo a duração de seu próximo surto de CPU, e concede a CPU ao processo que tem o próximo menor surto de CPU
- Duas alternativas:
  - **Não-preemptivo** – uma vez que a CPU é alocada para um processo, ele a mantém até que se complete sua fase de utilização da CPU
  - **Preemptivo** – se um novo processo chega com um surto de CPU menor que o tempo restante para o processo em execução, a CPU é alocada para este novo processo. Esta alternativa é conhecida como *Shortest-Remaining-Time-First* (SRTF)
- **SJF é ótimo** – apresenta o **menor tempo médio de espera** para um determinado conjunto de processos

# Exemplo de SJF Não-Preemptivo

| Processo | Hora de chegada | Duração |
|---|---|---|
| $P_1$ | 0 | 7 |
| $P_2$ | 2 | 4 |
| $P_3$ | 3 | 1 |
| $P_4$ | 5 | 4 |

➢**SJF (não-preemptivo)**




➢Tempo de espera médio = [0 + (8 – 2) + (7 – 3) + (12 - 5)]/4 = 4

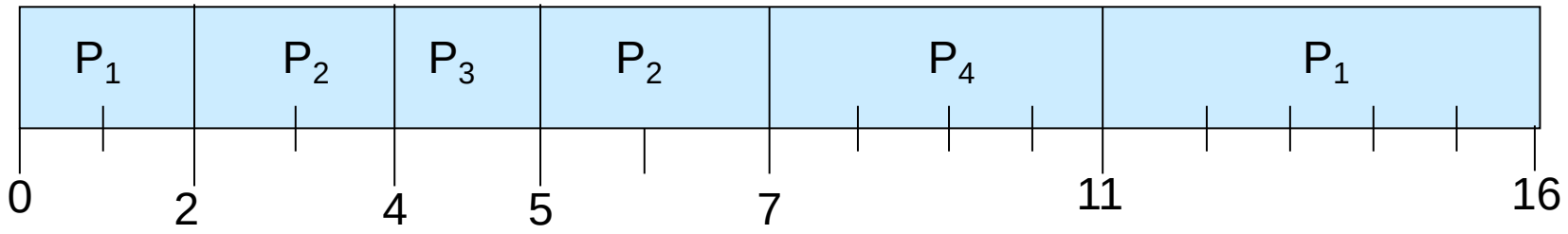


➢(Para cada processo, calcular o tempo de espera como: tempo inicio – tempo chegada)

# Exemplo de SJF Preemptivo

| Processo | Hora de Chegada | Duração |
|---|---|---|
| $P_1$ | 0 | 7 |
| $P_2$ | 2 | 4 |
| $P_3$ | 4 | 1 |
| $P_4$ | 5 | 4 |

- **SJF (preemptivo)**




- Tempo de espera médio = [(11 – 2) + (5 – 4) + 0 +(7 - 5)]/4 = 3

# Previsão da duração do próximo surto de CPU

- É possível estimar a duração
- Pode ser feita por meio do tempo de surto anterior da CPU, usando **média exponencial**

1. $t_n$ = duração do $n-ésimo$ surto da CPU
2. $\tau_{n+1}$ = valor previsto para o próximo surto da CPU
3. $\alpha,\ 0 \le \alpha \le 1$
4. Defina :

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

# Escalonamento por prioridade

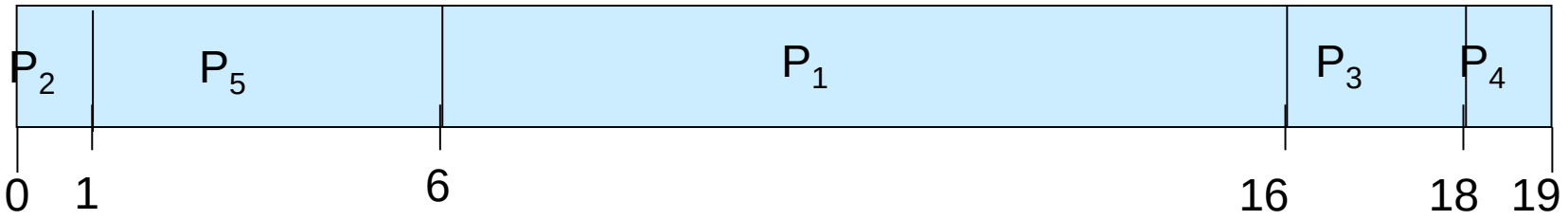
- Uma **prioridade** (nº. inteiro) é **associada a cada processo**
  - A CPU é alocada ao processo com a mais alta prioridade (quanto menor o inteiro => maior a prioridade)
  - Processos com mesma prioridade => FCFS para desempatar
  - Prioridades podem ser definidas por fatores externos (importância do processo, usuário, departamento, etc.) ou internos (requisitos de memória, limites de tempo, número de arquivos abertos, razão entre surto de E/S médio e surto de CPU médio, etc.

- Quando um novo processo chega na fila de prontos, se a sua prioridade for maior do que o processo em execução, o procedimento dependerá do tipo de escalonamento:
  - **Não-preemptivo:** o processo é simplesmente inserido na primeira posição da fila (execução do processo corrente não é afetada)
  - **Preemptivo:** o processo em execução será suspenso para liberar a CPU para o novo processo de maior prioridade

# Escalonamento por prioridade (não preemptivo)

| Processo | Duração de Surto | Prioridade |
|---|---|---|
| $P_1$ | 10 | 3 |
| $P_2$ | 1 | 1 |
| $P_3$ | 2 | 4 |
| $P_4$ | 1 | 5 |
| $P_5$ | 5 | 2 |

O Diagrama de Gantt para este escalonamento é:



# Escalonamento por prioridade

- **SJF** é um **escalonamento por prioridade** em que a prioridade é a duração do próximo de surto da CPU
  - Quanto menor a duração => maior a prioridade
- **Problema: *Starvation* (Abandono ou bloqueio indefinido)**
  - Processos com baixa prioridade podem nunca ser executados
  - Em um sistema sobrecarregado, processos de prioridade mais alta podem entrar frequentemente na fila e impedir que processo de baixa prioridade acesse a CPU
- **Solução: *Aging* (Envelhecimento)** – a prioridade dos processos cresce com o passar do tempo
  - Técnica para aumentar gradualmente a prioridade de processos que estão aguardando por um longo tempo
  - Exemplo:
    - Prioridades variando de 0 (alta) a 127 (baixa)
    - Aumentar em 1 a prioridade do processo a cada 15 minutos
    - Um processo com prioridade 127 levaria 32 horas para “envelhecer” e chegar a prioridade 0

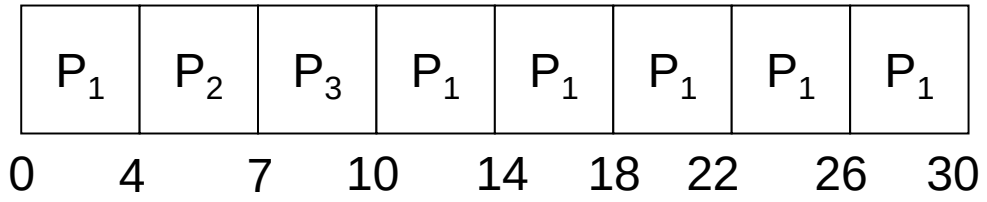
# Escalonamento Round Robin (RR)
## (Revezamento Circular)

- Foi projetado para sistemas de tempo compartilhado
- É semelhante ao FCFS (FIFO), com preempção para alternar entre os processos
- Cada processo utiliza a CPU por um tempo fixo chamado *time quantum* ou fatia de tempo (*time slice*), normalmente 10 a 100 milissegundos
- A fila de processos prontos é implementada como uma fila FIFO
  - Novos processos são incluídos no fim da fila
  - Quando a CPU fica livre, o escalonador obtém o primeiro processo da fila
  - Define um temporizador para interromper após 1 *quantum* de tempo e despacha o processo
  - Quando este tempo se expira, o processo é interrompido e colocado no final da fila de processos prontos

# Exemplo de RR com quantum = 4

| Processo | Tempo de surto |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

O Diagrama de Gantt é:

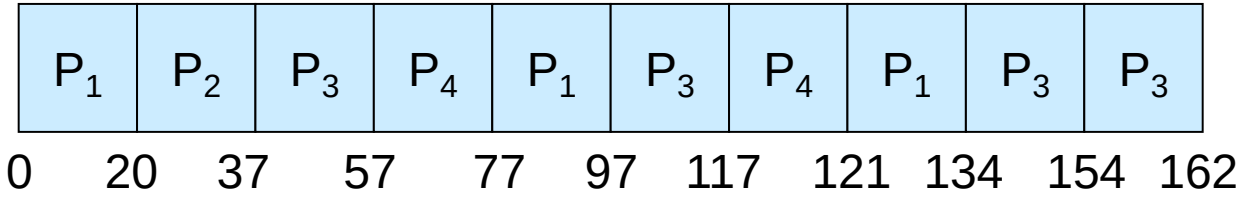



➢Tempo de espera médio = [(10 – 4) + 4 + 7]/3 = 5.66

# Exemplo de RR com quantum = 20

| Processo | Tempo de Surto |
|---|---|
| $P_1$ | 53 |
| $P_2$ | 17 |
| $P_3$ | 68 |
| $P_4$ | 24 |

➢O **Diagrama de Gantt** é:

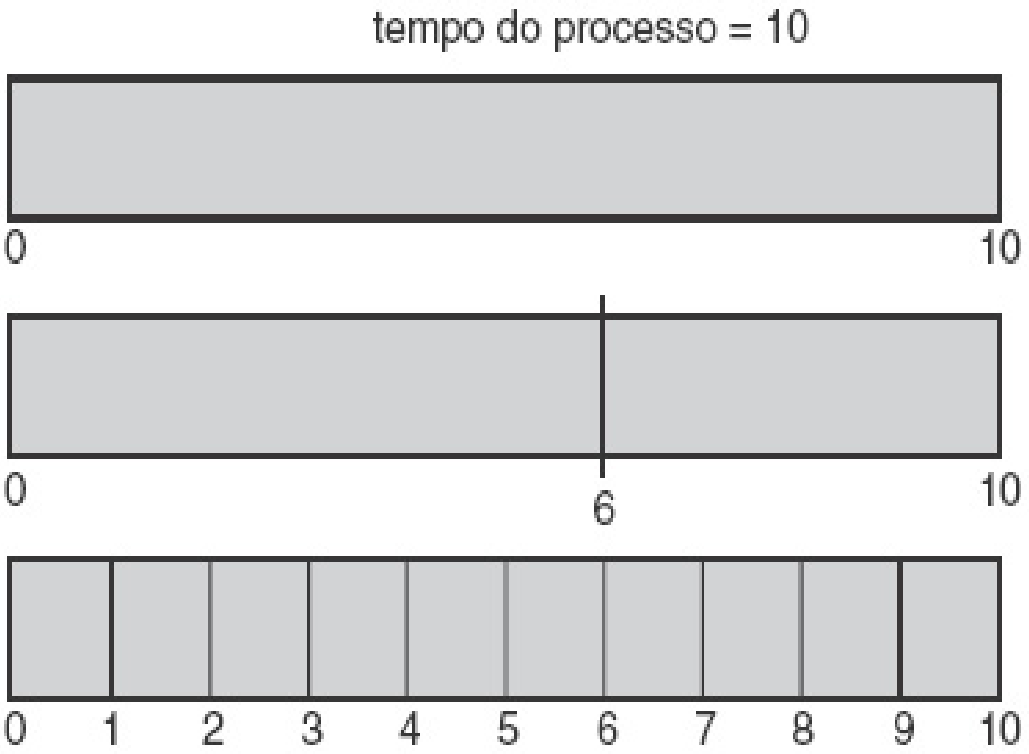



➢Normalmente, o **tempo de retorno médio é maior que o SJF**, **mas o tempo de resposta é melhor**

# Escalonamento Round Robin (RR)

- Se houver *n* processos na fila de prontos e o quantum for *q*, então cada processo obterá 1/*n* do tempo de CPU em lotes de no máximo *q* unidades de tempo
- Nenhum processo espera mais que (*n*-1)*q unidades de tempo*
- O desempenho do algoritmo RR depende muito do tamanho do *quantum q* de tempo
  - A troca de contexto requer um certo tempo para ser executada (carregar e descarregar registradores, atualizar listas e tabelas, carregar e descarregar memória, etc.)
  - *q* muito grande ⇒ política RR = política FIFO (ou FCFS) – a preempção raramente ocorrerá (resposta pobre para requisições interativas curtas)
  - *q* muito pequeno ⇒ *overhead* é muito alto; *q* precisa ser grande o suficiente com relação ao tempo requerido para a troca de contexto
    - Exemplo1: tempo para troca de contexto: 1 ms, q = 4 ms (20% do tempo da CPU gasto em troca de contexto)
    - Exemplo2: tempo para troca de contexto: 1 ms, q = 100 ms (1% do

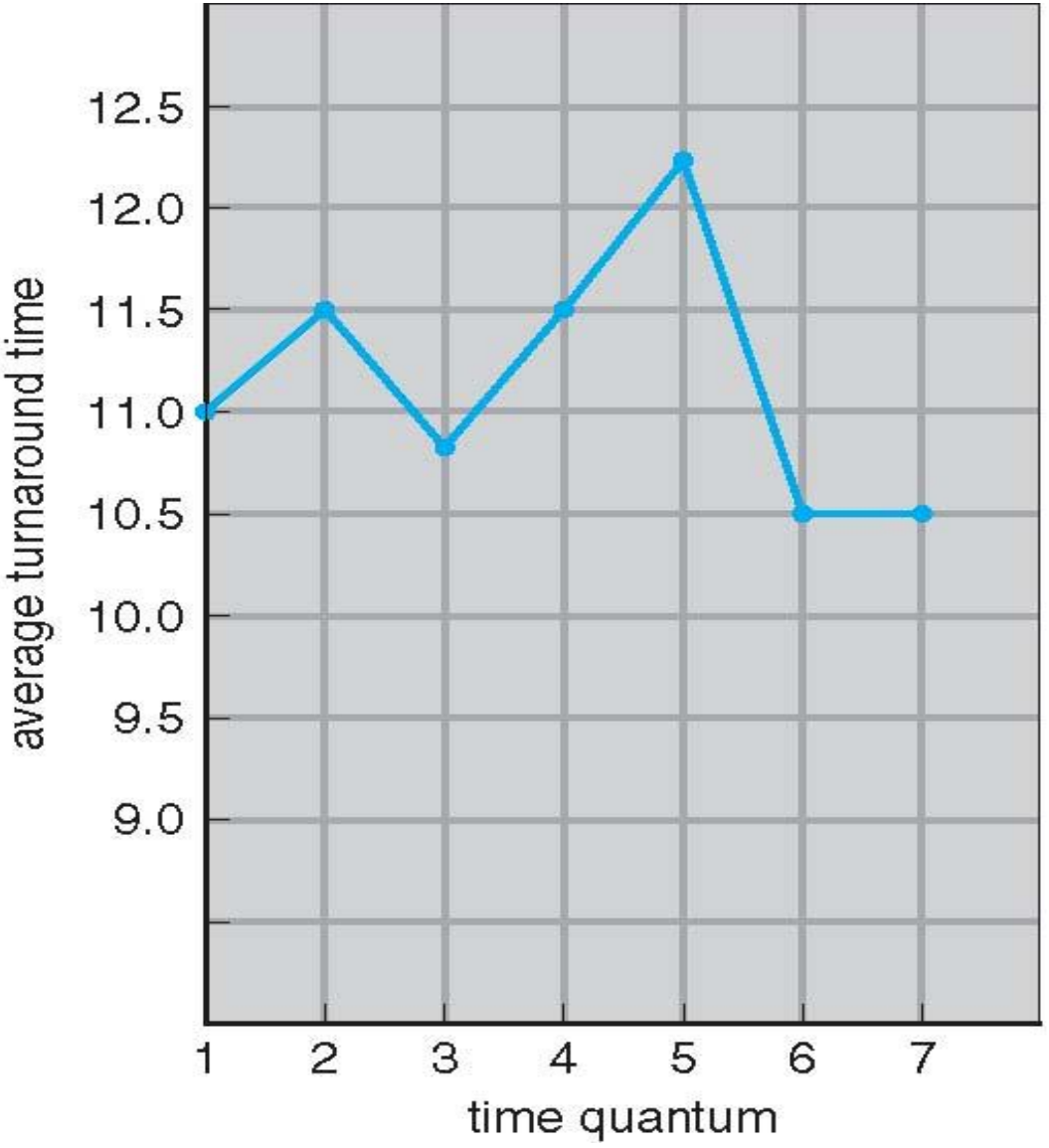
# Quantum e troca de contexto




| quantum | trocas de contexto |
|---|---|
| 12 | 0 |
| 6 | 1 |
| 1 | 9 |

Um quantum de tempo menor aumenta as trocas de contexto [Silberschatz]

# Tempo de retorno varia com o quantum




| process | time |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 3 |
| $P_3$ | 1 |
| $P_4$ | 7 |

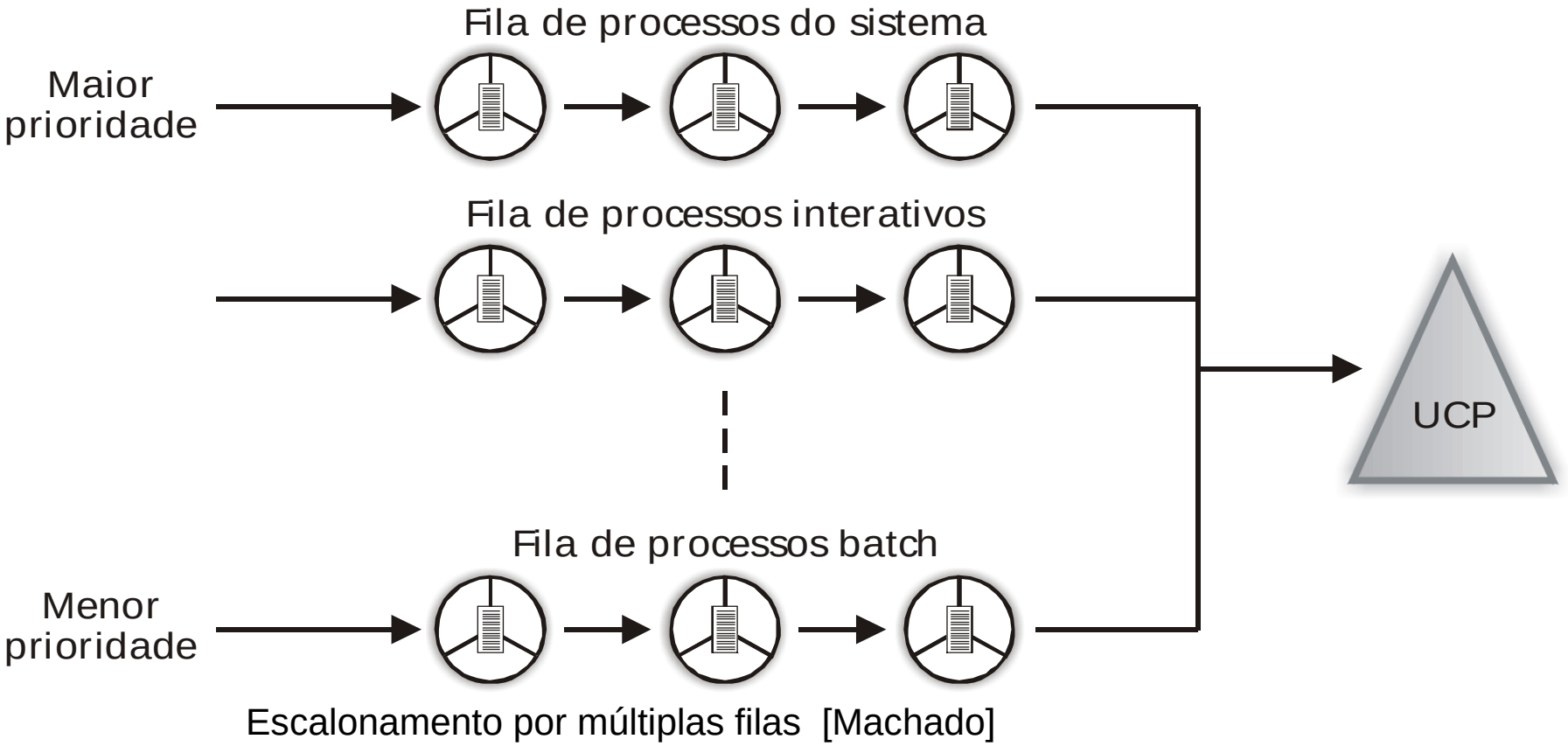
Variação do tempo de retorno com o quantum [Silberschatz]

# Escalonamento por múltiplas filas

- Processos são classificados em diferentes grupos, possuem diferentes requisitos de tempo de resposta e por isso podem ter necessidades de escalonamento diferentes
- Uma divisão comum é separar processos entre:
  - **Interativos**: de primeiro plano (*foreground)*
  - **Não-interativos**: de segundo plano (*background)* (*batch*)

- A fila de prontos é dividida em várias filas separadas (para processos *foreground* e processos *background*, por exemplo)
- Cada fila tem seu próprio algoritmo de escalonamento
  - *foreground* – RR
  - *background* – FCFS

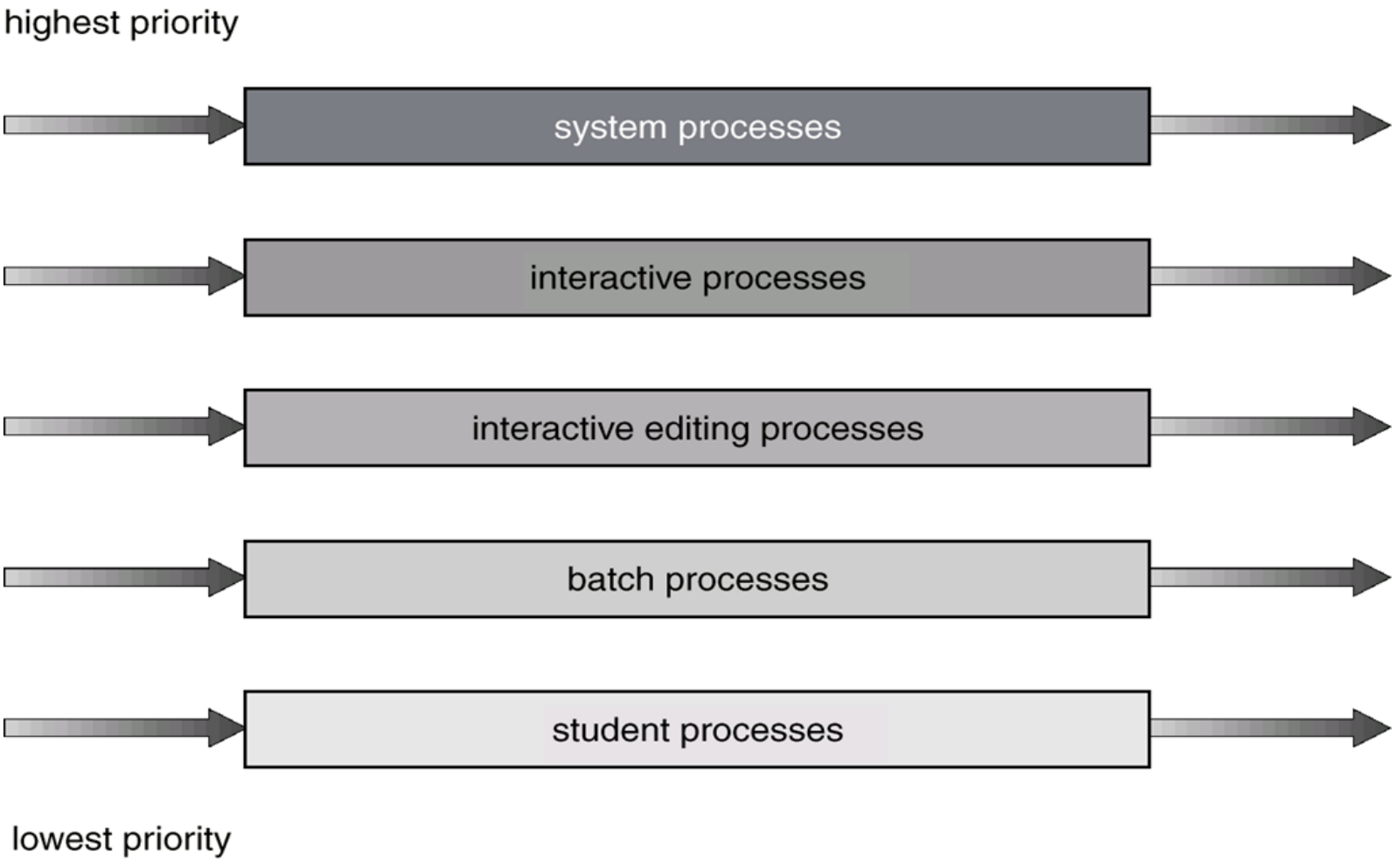
# Escalonamento por múltiplas filas




Escalonamento por múltiplas filas [Machado]

- É necessário também um escalonamento entre as filas
  - **Escalonamento com prioridade fixada** (i.e., atende primeiro todos os processos do *foreground* e depois *background*) - há possibilidade de *starvation*
  - ***Time slice*** (Fatia de Tempo) – cada fila recebe uma quantidade de tempo da CPU para escalonar os processos; i.e., 80% para *foreground* em RR e 20% para *background* em FCFS

# Escalonamento por múltiplas filas




Escalonamento por múltiplas filas (com 5 filas de acordo com a ordem de prioridade) [Silberschatz]

# Escalonamento de *Threads* Java

➢Todas as ***Threads* Java** recebem uma **prioridade** e a JVM escalona a *Thread* executável com a prioridade mais alta para execução

➢Uma **Fila FIFO** é usada se houver **múltiplas *Threads* com a mesma prioridade**

➢A JVM escalona uma *Thread* para ser executada quando:

1. A ***Thread* sendo executada deixa o estado "Executável"**
2. Uma ***Thread* de prioridade mais alta entra no estado "Executável"**

# Escalonamento de Threads Java (cont.)

➢A JVM não especifica se as *threads* tem o tempo repartido por meio de um escalonador de revezamento (Round Robin) – isso cabe à implementação específica da JVM

➢Se as *threads* tiverem fatias de tempo, uma *thread* Executável será executada durante seu *quantum* de tempo ou ocorrer um dos dois eventos acima

➢

➢Se as *threads* não tiverem fatia de tempo, uma *thread* será executada até que um dos dois eventos listados acima ocorra

# Fatia de tempo

Como a JVM não garante o fracionamento de tempo, o **método *yield*()** pode ser usado:

```
while (true) {
    // realiza uma tarefa de uso intensivo da CPU
    . . .
    // passa o controle da CPU
    Thread.yield();
}
```

Ao chamar o método *yield*(), uma *thread* abandona o controle da CPU, permitindo que outra *thread* com a mesma prioridade seja executado

# Prioridades de *Thread*

➢Todos as ***threads* Java**, quando criadas, recebem como **prioridade** um **inteiro positivo** dentro de um determinado intervalo (1 ~ 10), semelhante à prioridade da *thread* que a criou

| **Prioridade** | **Comentário** |
|---|---|
| Thread.MIN_PRIORITY | Prioridade Mínima de Thread (1) |
| Thread.MAX_PRIORITY | Prioridade Máxima de Thread (10) |
| Thread.NORM_PRIORITY | Prioridade *Default* de Thread (5) |

➢ Prioridades podem ser definidas usando o **método setPriority():**

setPriority(*Thread*.NORM_PRIORITY + 2) => (nova prioridade = 7)

# Referências


- [Silberschatz] SILBERCHATZ, A., GALVIN, P. B. e GAGNE, G. **Sistemas Operacionais com Java**. 7ª ed., Rio de Janeiro: Elsevier, 2008.
- [Tanenbaum] TANENBAUM, A. **Sistemas Operacionais Modernos**. 3ª ed. São Paulo: Prentice Hall, 2009.
- [MACHADO] MACHADO, F. B. e MAIA, L. P. **Arquitetura de Sistemas Operacionais**. 4ª ed., Rio de Janeiro: LTC, 2007.