Gerenciamento de Processos

LICENCIATURA EM COMPUTAÇÃO

Escalonamento

- Computador multiprogramado tem múltiplos processos ou *threads* competindo pela CPU ao mesmo tempo.
  - Essa situação ocorre sempre que dois ou mais deles estão simultaneamente **no estado pronto.**
- Se apenas uma CPU está disponível, uma escolha precisa ser feita sobre qual processo será executado em seguida.
  - A parte do sistema operacional que faz a escolha é chamada de **escalonador**, e o algoritmo que ele usa é chamado de **algoritmo de escalonamento**.
- **Muitas das mesmas questões que se aplicam ao escalonamento de processos também se aplicam ao escalonamento de *threads*, embora algumas sejam diferentes.**
  - Quando o núcleo gerencia *threads*, o escalonamento é geralmente feito por *thread*, com pouca ou nenhuma consideração sobre o processo ao qual o *thread* pertence.
- **Nos concentraremos nas questões de escalonamento que se aplicam a ambos, processos e *threads*.**

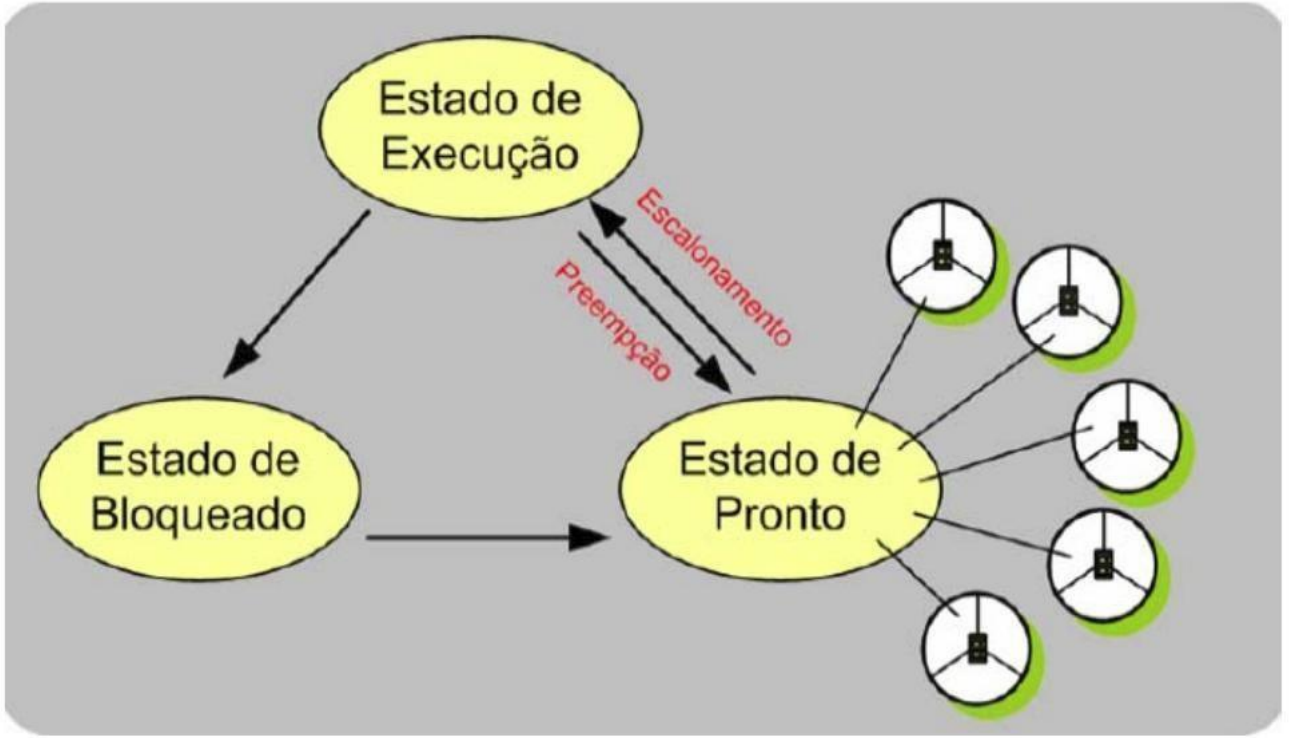
Políticas de Escalonamento

- SO escolhe qual processo da fila de PRONTOS irá executar.
- Despachante designa um processador a um dado processo.
- **Política de escalonamento** (ou disciplina de escalonamento):
  - Critério do SO para escolher o processo que executará.
- Deve garantir: **justiça, previsibilidade e escalabilidade.**
- Deve considerar o **comportamento de um processo**:
  - Processos CPU-*Bound* x I/O-*Bound*
  - Processos Em Lote x Interativo

Políticas de Escalonamento

- Maximizar: *throughput* (total de processos terminados na unidade de tempo).
- Minimizar: *turnaround* (tempo da criação até o término do processo).
- Maximizar: taxa de utilização de CPU (tempo total de ocupação da CPU).
- Minimizar: tempo de resposta (tempo da criação até o início da execução).
- Minimizar: tempo de espera (soma dos tempos na fila de prontos).
- Minimizar: tempo de resposta em processos interativos.
- Maximizar: utilização dos recursos.
- Minimizar: a sobrecarga de gerenciamento do SO.
- Favorecer: rotinas de maior prioridade e importância.
- Garantir: previsibilidade no atendimento.

Escalonamento



Políticas de Escalonamento

- Poderá ser classificada como (em relação a como lidar com interrupções de relógio):
  - Preemptiva.
  - Não-preemptiva.
- **Não-preemptiva (cooperativo)**
  - SO **NÃO** pode **interromper** um processo.
  - Processo executa até concluir ou parar voluntariamente (ser bloqueado por E/S ou esperando outro processo).
- **Preemptiva:**
  - SO **pode** interromper um processo a **qualquer instante.**
    - Para executar outro processo (chaveamento de contexto).
  - Escalonador escolhe um processo e o deixa executar por no máximo um certo tempo fixado.
- Há vantagens e desvantagens em ambas.

Políticas de Escalonamento com Prioridade

- **Mecanismos de prioridades** são de dois tipos:
  - **Prioridade estática** e **Prioridade dinâmica**
- **Prioridade estática:**
  - Não varia ao longo da execução do processo.
  - Fácil de implementar.
- **Prioridade dinâmica** (mais inteligente):
  - Varia ao longo da execução do processo.
  - Permite uma maior responsividade à mudanças no ambiente.
  - Exemplos: Técnica de *aging*, algoritmo HRRN (*Highest Response Ratio Next*)

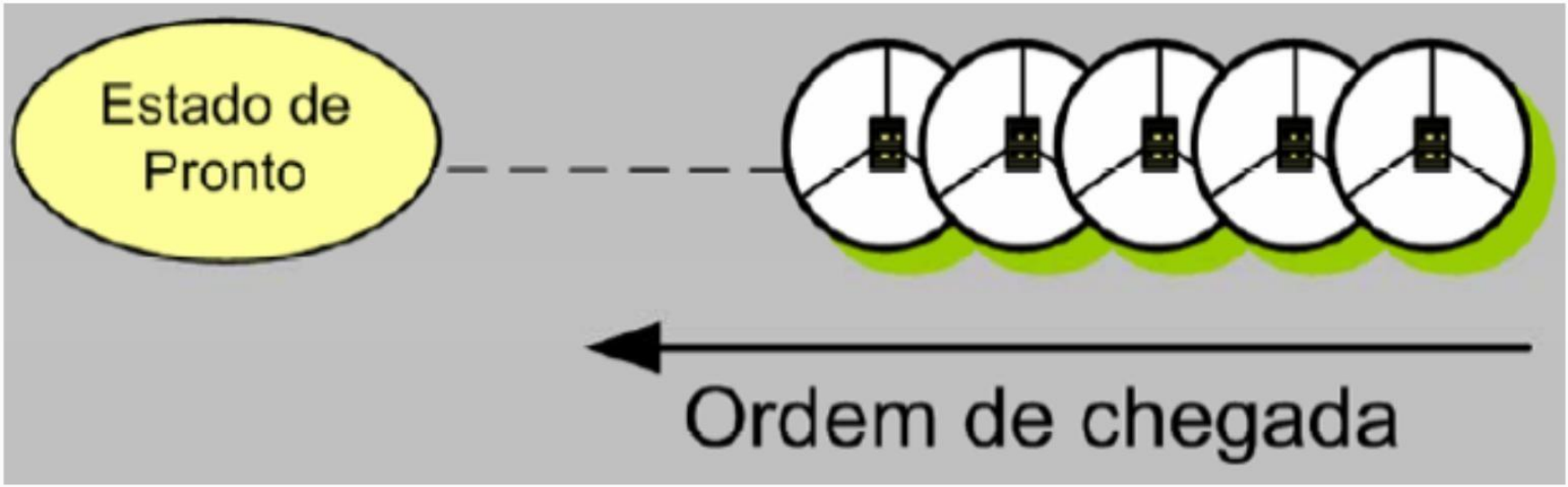
Políticas de Escalonamento - critérios

- Tentam implementar “justiça” na escolha dos processos.
- Primeiros SO eram colaborativos: ineficientes!
  - Exemplo: Windows95 (multitarefa cooperativa).
  - Processo rodava o tempo que desejava.
- **Regra geral:**
  - **Processo que chega vai para o final da fila de “prontos”.**
  - **Fila é rearranjada** conforme o **critério de escalonamento.**
- Atualmente, na prática, algoritmos usados nos SO:
  - Combinam dois ou mais critérios e são adaptativos.
  - Variam dinamicamente conforme os estados dos processos.

FIFO (*First In First Out*)

- ***First-Come First-Served* (primeiro chegar, primeiro a ser servido).**
- Critério: **ordem de chegada.**
- **Não-preemptivo** (processo executa enquanto quiser).
- **Vantagens:**
  - Justo: atende pela ordem de chegada.
  - Impede adiamento indefinido (*starvation*).
  - Fácil de implementar.
- **Desvantagens:**
  - Processos longos fazem os curtos esperarem muito.
  - Não se mostra eficiente para processos interativos.
  - Não considera a importância de uma tarefa.

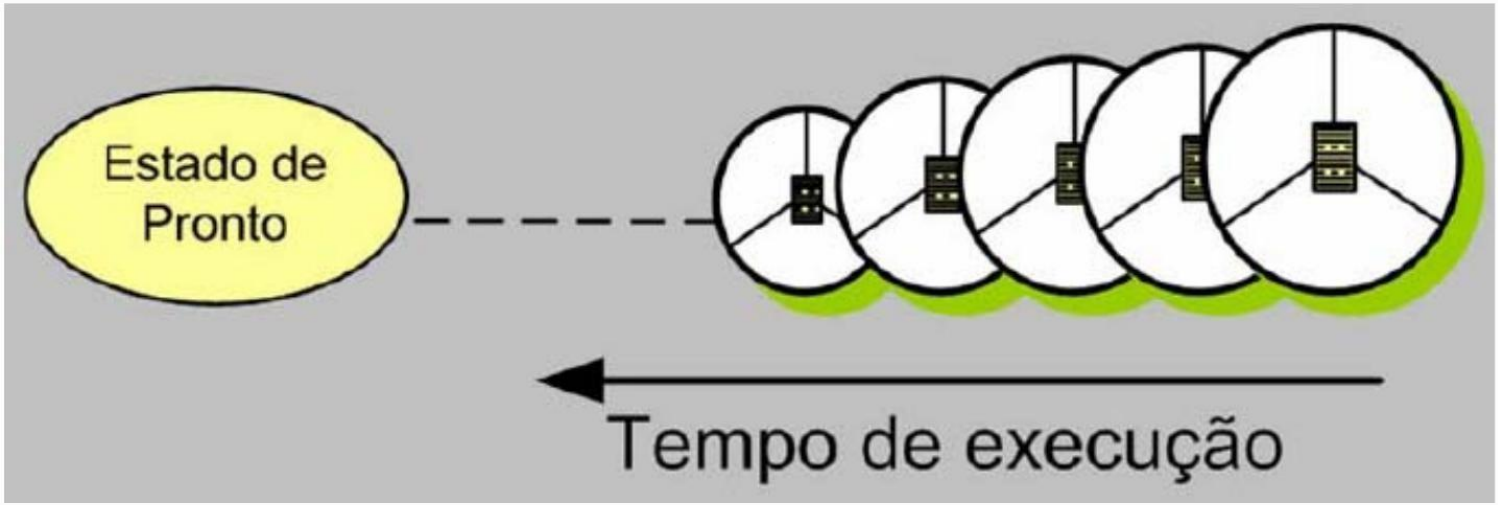
FIFO (*First In First Out*)



SJF (*Shortest Job First*) - SPF (*Shortest Process First*)

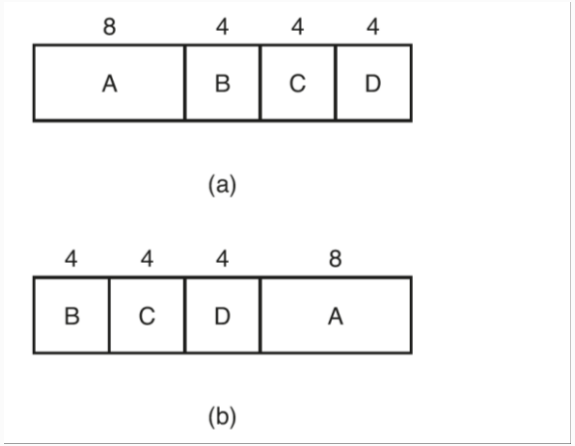
- **Processo mais curto primeiro.**
- **Presume que os tempos de execução são conhecidos antecipadamente.**
- **Critério:** tempo de execução restante (*burst*): menor primeiro.
- Não-preemptivo.
- **Vantagens:**
  - Favorece os processos mais curtos.
  - Aumenta o rendimento (*throughput*).
  - Menor tempo médio de espera.
- **Desvantagens:**
  - Baseado em estimativas de tempo.
  - *Não impede o adiamento indefinido (starvation*).
  - Maior variância no tempo de espera (+ imprevisibilidade).

SJF (*Shortest Job First*) - SPF (*Shortest Process First*)



SJF (*Shortest Job First*) - SPF (*Shortest Process First*)

- **Exemplo:** Quatro tarefas *A*, *B*, *C* e *D* com tempos de execução de 8, 4, 4 e 4 minutos, respectivamente.
  - (a) **Ordem original:** tempo de retorno para *A* é 8 minutos, para *B* é 12 minutos, para *C* é 16 minutos e para *D* é 20 minutos → resultando em uma média de 14 minutos.
  - (b) **Mais curto primeiro:** tempos de retorno são agora 4, 8, 12 e 20 minutos → resultando em uma média de 11 minutos.



SRT (*Shortest Remainning Time*)

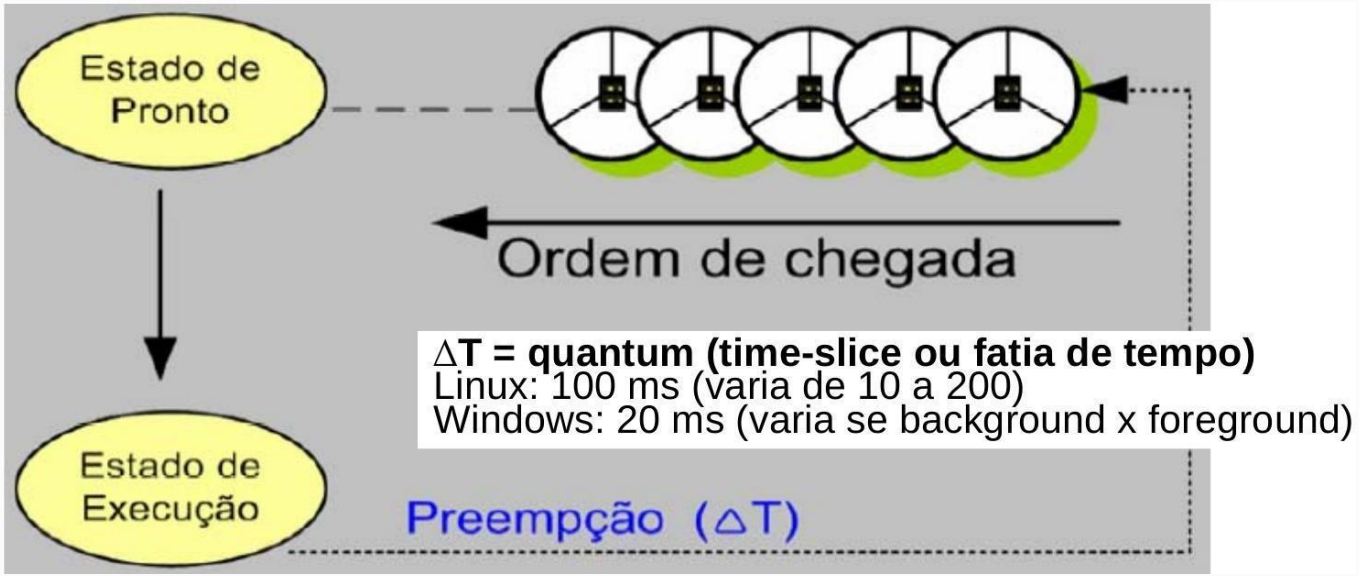
- **Tempo restante mais curto em seguida.**
- **Versão preemptiva do SPF/SJF.**
- **Critério:** tempo de execução restante (*burst*): escolhe o mais curto primeiro.
- **Preemptivo** (se chegar processo de menor *burst*).
- Tempo de execução **precisa ser conhecido antecipadamente.**
  - Quando uma nova tarefa chega, seu tempo total é comparado com o tempo restante do processo atual. Se a nova tarefa precisa de menos tempo para terminar do que o processo atual, este é suspenso e a nova tarefa iniciada.
- **Vantagens:**
  - Busca minimizar tempo de espera.
- **Desvantagens:**
  - Baseado em estimativas de tempo.
  - Gera sobrecarga desnecessária nas troca de contexto.
  - Usado em SO antigos (para processamento em lote).

RR (*Round Robin*)

- **Alternância circular.**
- **Critério: ordem de chegada (como no FIFO porém, é uma Fila Circular).**
- **Preemptivo por *quantum* de tempo.**
  - A cada processo é designado um intervalo (*quantum*) durante o qual ele é deixado executar. Se o processo ainda está executando ao fim do *quantum*, a CPU sofrerá uma preempção e receberá outro processo. Se o processo foi bloqueado ou terminado antes de o *quantum* ter decorrido, o chaveamento de CPU será feito quando o processo bloquear.
- **Vantagens:**
  - Efetivo com processos interativos.
  - Impede adiamento indefinido.
- **Desvantagens:**
  - Mais complexo que FIFO.
  - Adiciona sobrecarga no chaveamento de contexto.

RR (*Round Robin*)

- Obs.: Estabelecer o *quantum* curto demais provoca muitos chaveamentos de processos e reduz a eficiência da CPU, mas estabelecê-lo longo demais pode provocar uma resposta ruim a solicitações interativas curtas.
- Um *quantum* em torno de 20-50 ms é muitas vezes bastante razoável.

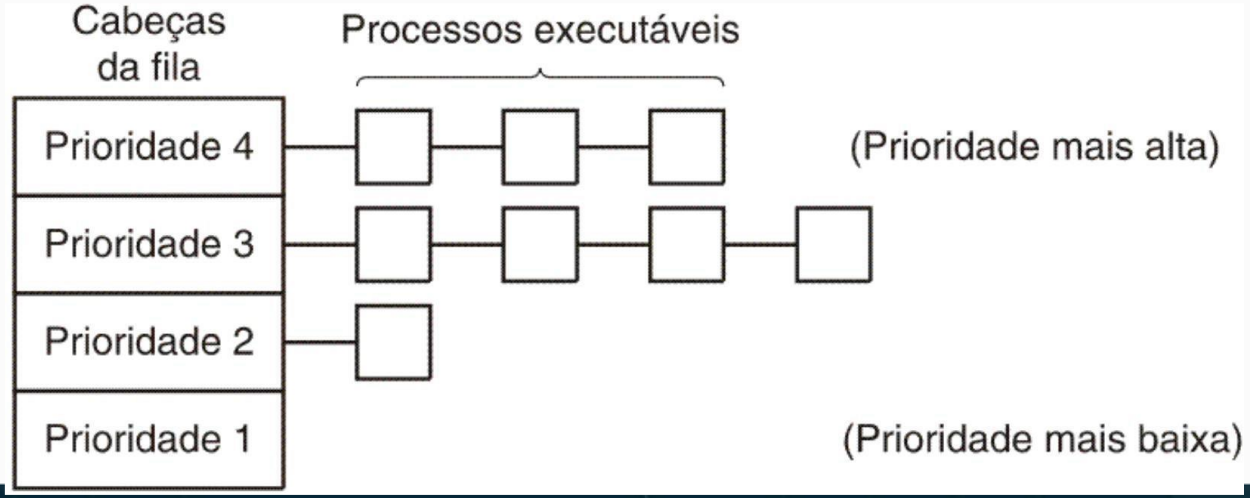


Por Prioridades

- **Critério:** maior prioridade primeiro.
- **Preemptivo** (por tempo e/ou prioridade de execução).
  - Para evitar que processos de prioridade mais alta executem indefinidamente, o escalonador pode diminuir a prioridade do processo que está sendo executado em cada tique do relógio. Quando a prioridade cair abaixo daquela do próximo processo com a prioridade mais alta, ocorre um chaveamento de processo.
  - Também pode ser designado a cada processo um *quantum* de tempo máximo no qual ele é autorizado a executar.
- Prioridade pode ser **fixa** ou **dinâmica**.
- Agrupa processos em **filas de prioridades decrescentes.**
  - Em cada fila, aplica-se RR (*Round Robin.*)

Por Prioridades (Fila de Prioridades)

- **Algoritmo:** desde que existam processos executáveis na classe de prioridade 4, apenas execute cada um por um quantum, estilo circular, e jamais se importe com classes de prioridade mais baixa. Se a classe de prioridade 4 estiver vazia, então execute os processos de classe 3 de maneira circular. Se ambas as classes 4 e 3 estiverem vazias, então execute a classe 2 de maneira circular e assim por diante.
- Se as prioridades não forem ajustadas ocasionalmente, classes de prioridade mais baixa podem todas morrer famintas.



Filas Múltiplas - Multinível com *Feedback*

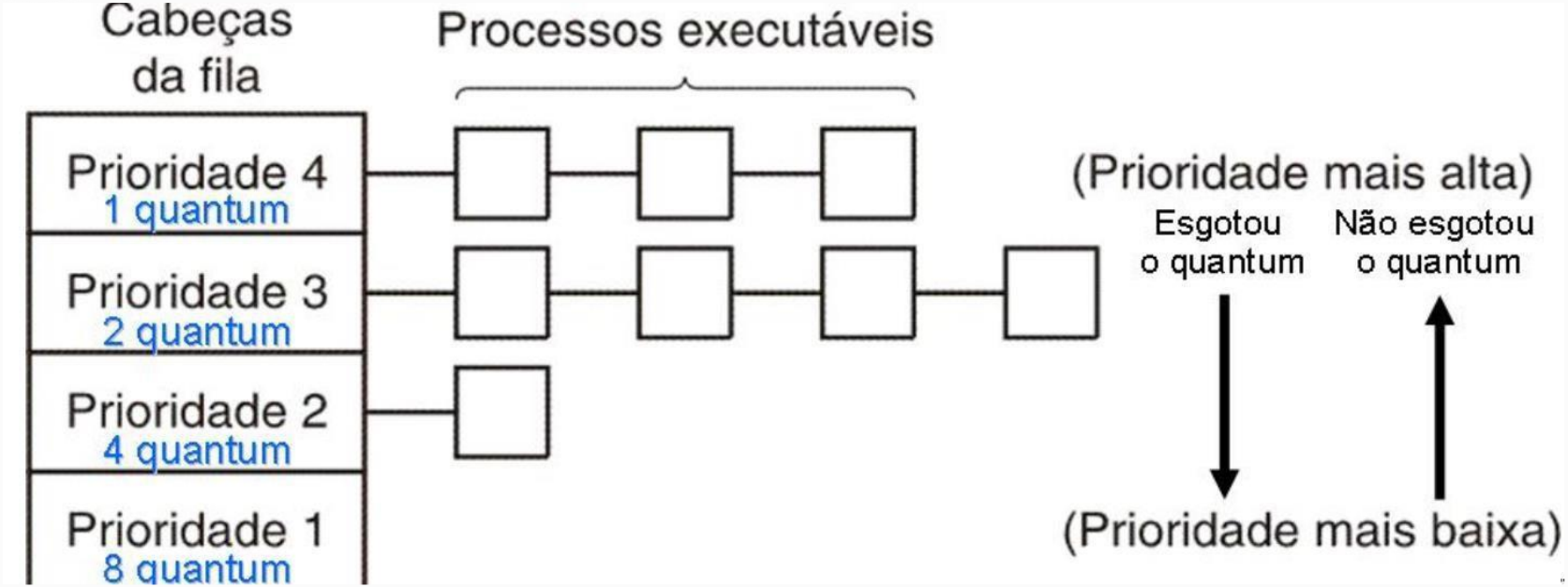
- **Critério:** Filas de prioridade distintas e *quantum* crescente.
  - Processos na classe mais alta seriam executados por dois *quantuns*. Processos na classe seguinte seriam executados por quatro *quantuns* etc. Sempre que um processo consumia todos os *quantuns* alocados para ele, era movido para uma classe inferior.
- **Preemptivo** (em cada fila usa RR ou outro critério).
- Filas mais altas são executadas primeiro.
- *Processos mudam de fila: pelo uso do quantum.*
  - Se esgota, **desce** (menor prioridade).
  - Senão esgota, **sobe** (maior prioridade) ou mantém a fila.

Filas Múltiplas - Multinível com *Feedback*

- **Vantagens:**
  - Mais justo e inteligente que os algoritmos básicos.
  - Prioriza processos interativos e rápidos (ou *IO-Bound*).
  - Aumenta o *quantum* dos processos *CPU-Bound*.

- **Desvantagens:**
  - Complexidade de implementação.

Filas Múltiplas - Multinível com *Feedback*



Outras Políticas de Escalonamento

- **Escalonamento Garantido**
  - Faz promessas reais ao usuário sobre desempenho e cumpre (% alocação de CPU).
    - **Por exemplo:** em um sistema de usuário único com *n* processos sendo executados, todos os fatores permanecendo os mesmos, cada um deve receber *1/n* dos ciclos da CPU - **justo**.
  - Para isso, o sistema deve **controlar quanta CPU** cada processo teve **desde sua criação,** calculando o montante de CPU a que cada um tem direito (tempo desde a criação dividido por *n*).
  - Montante de tempo da CPU que cada processo teve é conhecido → calcular o índice de tempo de CPU real consumido com o tempo de CPU ao qual ele tem direito é direto.
    - Um índice de 0,5 significa que o processo teve apenas metade do que deveria, e um índice de 2,0 significa que teve duas vezes o montante de tempo ao qual ele tinha direito. O algoritmo então executará o processo com o índice mais baixo até que seu índice aumente e se aproxime do de seu competidor. Então, este é escolhido para executar em seguida.
  - Difícil de implementar!!!

Outras Políticas de Escalonamento

- **Escalonamento por Loteria**
  - SO distribui *tokens* (fichas, bilhetes de loteria) numerados entre os processos para vários recursos do sistema, como tempo de CPU. Escalonador sorteia um *token* aleatório e o processo com o *token* fica com recurso.
  - Processos importantes podem receber ***tokens* extras**.
  - Processos com mais *tokens* têm mais chance de escolha.
  - Ideal para processos cooperativos (doação de *tokens*).
    - Por exemplo, quando um processo cliente envia uma mensagem para um processo servidor e então bloqueia, ele pode dar todos os seus bilhetes para o servidor a fim de aumentar a chance de que o servidor seja executado em seguida. Quando o servidor tiver concluído, ele devolve os bilhetes de maneira que o cliente possa executar novamente.

Outras Políticas de Escalonamento

- **Escalonamento de Tempo Real**
  - **Um sistema de tempo real é aquele em que o tempo tem um papel essencial.**
  - Divide-se em 2 tipos:
    - **Crítico:** prazos devem ser rigorosamente cumpridos.
    - **Não-crítico:** descumprimentos de prazo são tolerados.
  - **Prioriza os processos** em detrimento do próprio SO.
  - Busca produzir resultados em tempos determinados.
  - O comportamento em tempo real é conseguido dividindo o programa em uma série de processos, cada um dos quais é previsível e conhecido antecipadamente. Esses processos geralmente têm vida curta e podem ser concluídos em bem menos de um segundo. Quando um evento externo é detectado, cabe ao escalonador programar os processos de uma maneira que todos os prazos sejam atendidos.

Ler!!!

- Capítulo 2 do Livro Texto.

Atividade

- **Atividade no AVA.**