

AULA 10 - Gerência do Processador

Conforme comentamos, em alguns momentos podemos ter 2 ou mais processos aptos a utilizar o processador para ser executado. Nesse instante, o sistema operacional deve decidir qual dos processos aptos, armazenados em uma fila, será escolhido para rodar primeiro. Essa tarefa e a tomada de decisão é feita pelo *escalonador* de processos (parte do sistema operacional) através da implementação de alguns algoritmos de seleção, denominados *algoritmos de escalonamento*.

Escalonadores

O escalonador é a entidade do sistema operacional responsável por selecionar um processo apto a executar no processador e dividir o tempo do processador de forma justa entre os processos que estão aptos. Em outras palavras, o objetivo dos escalonadores é implementar uma política de escalonamento de processos.

O sistema operacional possui um módulo responsável por efetuar a troca de contexto entre a execução de processos distintos, chamado de *Dispatcher*. Já o escalonador está relacionado com a implementação e aplicação das políticas de seleção adotadas.

Objetivos do escalonamento:

1. Maximizar a utilização do processador
2. Privilegiar aplicações que são críticas
3. Maximizar a produção do sistema (*throughput*)
 - a. Número de processos executados por unidade de tempo
4. Minimizar o tempo de execução (*turnaround* – tempo que um processo gasta desde a sua criação até seu término)
 - a. Tempo total para executar um determinado processo
5. Minimizar o tempo de espera
 - a. Tempo que um processo permaneça na lista de aptos
6. Minimizar o tempo de resposta
 - a. Tempo decorrido entre uma requisição e a sua realização

Tipos de escalonadores

Existem 2 tipos de escalonadores:

1. **não-preemptivo**: escalonadores que permitem que os processos rodem até o fim de sua execução sem ser interrompidos por eventos externos.
2. **preemptivo**: escalonadores que são capazes de suspender processos que poderiam continuar executando.

Para cada um desses tipos, os processos poderão utilizar o processador até que:

Não preemptivo:

1. Término de execução do processo
2. Execução de uma requisição de entrada/saída ou sincronização
3. Liberação voluntária do processador a outro processo (*yield*)

Preemptivo:

1. Término de execução do processo
2. Execução de uma requisição de entrada/saída ou sincronização
3. Liberação voluntária do processador a outro processo (*yield*)
4. Interrupção de relógio
5. Processo de mais alta prioridade esteja pronto para executar

Algoritmos de escalonamento

O algoritmo de escalonamento deve selecionar qual processo irá executar em um determinado instante de tempo. Na literatura existem vários algoritmos destinados a este objetivo, todos buscam obter bons tempos médios ao invés de maximizar ou minimizar alguns dos critérios citados anteriormente.

Algoritmos não preemptivos

1. FIFO
2. SJF
3. Cooperativo

Algoritmos preemptivos

1. Round robin (circular)
2. Múltiplas filas

FIFO - First In First Out

É o algoritmo mais simples de implementar, onde o processador possui uma fila associada para armazenar os processos que estão aptos a executar.

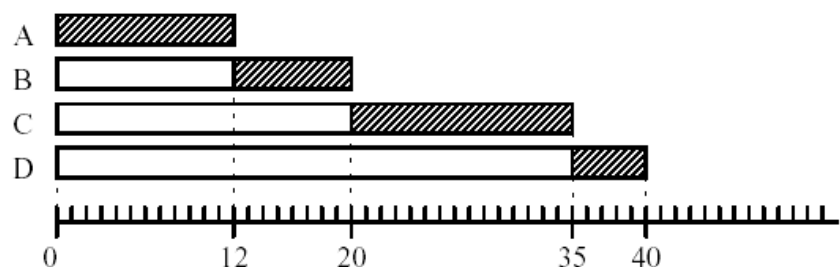
Funcionamento:

1. Processos que se tornam aptos são inseridos no final da fila
2. Processo que está no início da fila é o próximo a executar
3. Processo executa até que:
 - a. Libere explicitamente o processador
 - b. Realize uma chamada de sistema (bloqueado)
 - c. Termine sua execução

Desvantagem:

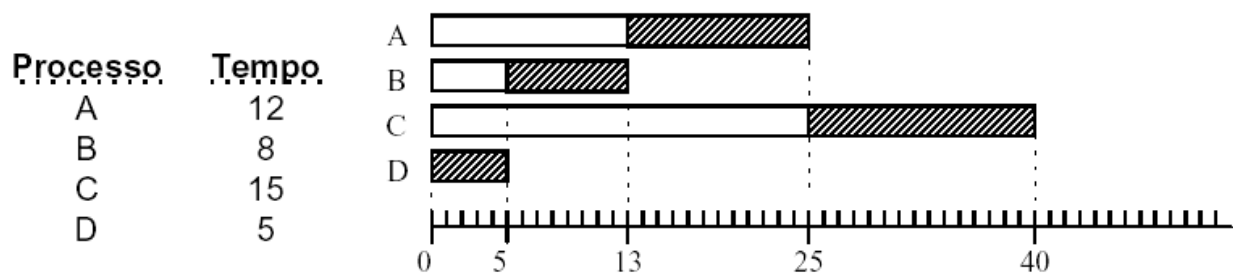
1. Prejudica processos *I/O bound*
2. Tempo médio de espera na fila de execução:
 - a. Ordem A-B-C-D = $(0 + 12 + 20 + 35) / 4 = 16.75$ u.t.
 - b. Ordem D-A-B-C = $(0 + 5 + 17 + 25) / 4 = 11.7$ u.t.

Processo	Tempo
A	12
B	8
C	15
D	5



SJF - Shortest Job First

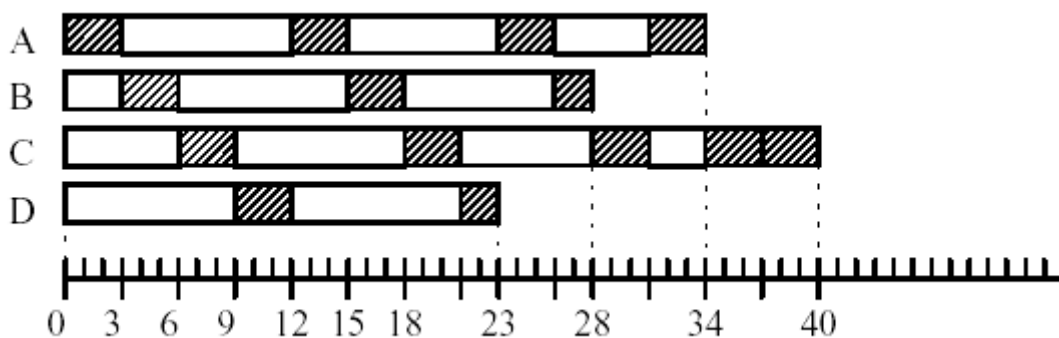
Para usar este algoritmo precisamos conhecer antecipadamente o tempo de execução de cada processo, o que é difícil. A idéia é alocar o processador para o menor job da fila. O fato é que o menor tempo médio é obtido quando se executa primeiro os processos de menor ciclo de processador (*I/O bound*). Portanto, os processos *I/O bound* são favorecidos.



$$\text{Tempo médio: } (0 + 5 + 13 + 25)/4 = 10.75 \text{ u.t}$$

RR - Round Robin

A cada processo atribuímos um tempo durante o qual ele poderá utilizar o processador. Denominamos este intervalo de tempo de quantum. Além disso, este algoritmo é similar ao algoritmo FIFO, pois também mantemos uma fila, agora circular, para armazenar os processos. Para a execução existe a necessidade de um relógio para delimitar as fatias de tempo.



O processo perde o processador quando:

1. Libera explicitamente o processador
2. Realize uma chamada de sistema (bloqueado)
3. Termina sua execução
4. Quando sua fatia de tempo é esgotada

Escalonamento do tipo *round-robin* é preemptivo: se o *quantum* ? 8 obtém-se o comportamento de um escalonador FIFO.

Problema 1: Dimensionar o *quantum*

1. Compromisso entre *overhead* e tempo de resposta em função do número de usuários
2. Compromisso entre tempo de chaveamento e tempo do ciclo de processador (*quantum*)

Imagine que a cada 20 ms de processamento útil seja necessário gastar 5 ms com tarefas de troca de contexto para rodar processos de outro usuário, por exemplo. Com isso, 20% do tempo de processador será gasto com estes overheads.

Problema 2: Processos *I/O bound* são prejudicados

1. Esperam da mesma forma que processos *CPU bound* porém muito provavelmente não utilizam todo o seu *quantum*
2. Solução: prioridades

Escalonadores por prioridades

1. Associar prioridades a processos *I/O bound* para compensar o tempo gasto em estado de espera (apto)
2. Sempre que um processo de maior prioridade que o processo atualmente em execução entrar no estado apto deve ocorrer uma preempção
3. Escalonamento com prioridades é inerente a preempção
4. É possível haver prioridade não-preemptiva
5. Escalonador deve sempre selecionar o processo de mais alta prioridade

Prioridade estática versus dinâmica

Prioridade estática:

Um processo é criado com uma determinada prioridade e esta prioridade é mantida durante todo o tempo de vida do processo

Prioridade dinâmica:

Prioridade do processo é ajustada de acordo com o estado de execução do processo e/ou do sistema

Ex.: ajustar a prioridade em função da fração do *quantum* que foi realmente utilizada pelo processo

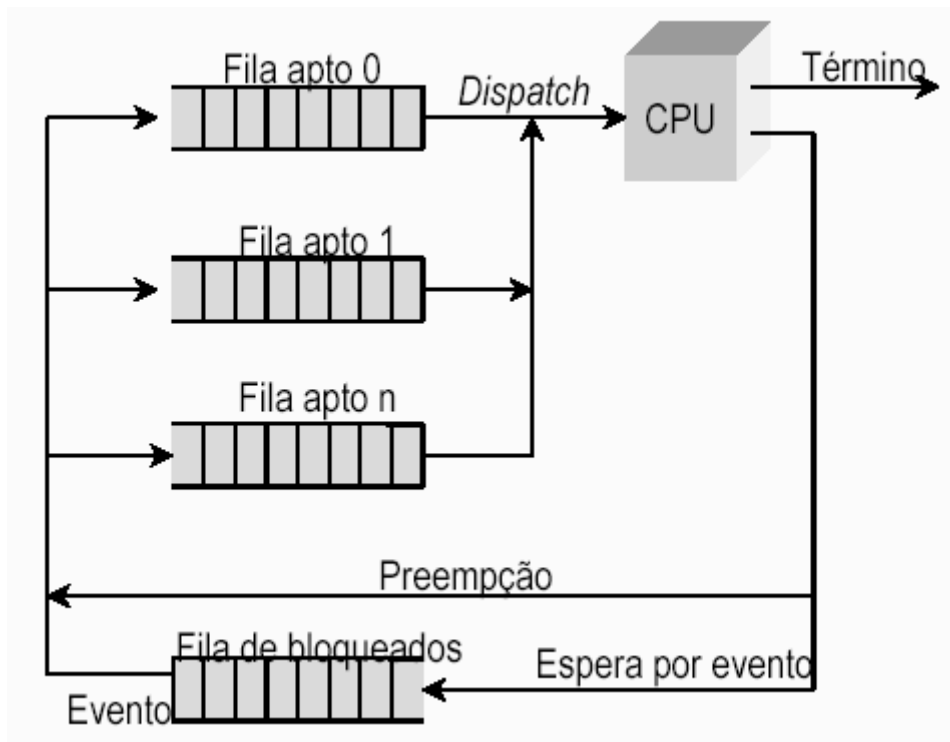
$$q = 100 \text{ ms}$$

$$\text{Processo A utilizou 2ms !nova prioridade} = 1/0.02 = 50$$

$$\text{Processo B utilizou 50ms !nova prioridade} = 1/0.5 = 2$$

Implementação de escalonador com prioridades

1. Múltiplas filas associadas ao estado apto
2. Cada fila uma prioridade
3. Pode ter sua própria política de escalonamento (FIFO, SJF, RR)

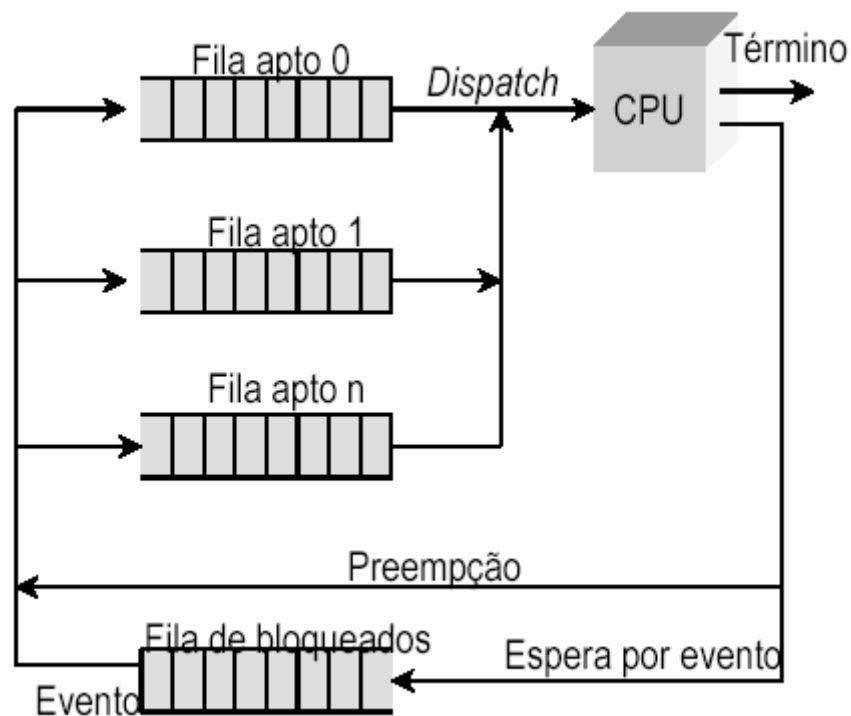


Problemas com prioridades

1. Com prioridades um processo de baixa prioridade pode não executar (*starvation*)
2. Um processo que durante sua execução troca de comportamento (*CPU bound* a *I/O bound*) pode ficar mal classificado a nível de prioridades e ser penalizado
3. Solução: Múltiplas filas com realimentação

Múltiplas filas com realimentação

1. Baseado em prioridades dinâmicas.
2. Em função do tempo de uso da CPU a prioridade do processo aumenta e diminui
3. Sistema de envelhecimento (*aging*) evita postergação indefinida.



Exercícios

1. Faça uma pesquisa e explique o funcionamento dos seguintes algoritmos de escalonamento:
 - a. Escalonamento por loteria
 - b. Escalonamento por fração justa (*fair-share*)
 - c. Escalonamento garantido
 - d. Escalonamento por menor tempo de execução restante (SRT)
2. Apresente uma definição sobre o problema de inversão de prioridades.
3. Quando um escalonador não preemptivo é mais adequado que um escalonador preemptivo?
4. Um programa que entra em laço infinito pode monopolizar um sistema preemptivo?
5. Alguns livros comentam sobre três níveis de escalonadores, comente a respeito de cada um:
 - a. Escalonador de alto nível
 - b. Escalonador de nível intermediário
 - c. Escalonador de baixo nível

Bibliografia

OLIVEIRA, R. S.; CARISSIMI, A. S. e TOSCANI, S. S. *Sistemas Operacionais*. Sagra Luzzatto, Porto Alegre, 2001. (Cap. 4)