



# Revisão SOP

CPU ou seus núcleos são unidimensionais

Escalonador: Decide qual processo executar a seguir e por quanto tempo

- Para sistema multiprogramador ocorre a multiplexação no tempo, ou seja, zwar o mesmo processador com um tempo próprio definido para cada processo vai
- Escalonador usa algoritmos de escalonamento

## Passamento de contexto

\* Usado para trocar processos em execução

- 1) Contexto do atual processo é salvo na tabela de processos
- 2) Contexto do novo processo é carregado da tabela de processos
- 3) Novo processo inicia sua execução

\* Tabela de processos é os registradores de CPU/memória + estrutura de dados do núcleo.

## Diferenças entre os algoritmos de escalonamento

- Minimizar custo da troca de contexto Vs minimizar tempo de espera
- Usar a CPU de maneira eficiente

## Processo orientado a CPU

- ↳ Só usos ciclos de CPU, muito tempo usando a CPU

## Processos orientados a entrada e saída.

- ↳ Usam a CPU, por pouco tempo e logo fazem uma requisição de E/S

(Quando - eralmente?)

- ↳ Crição de processos → executa o novo ou o atual?

- ↳ Encerramento de um processo → qual o próximo?

- ↳ B. Sócio → qual é próximo?

- ↳ Interrupção de E/S → pode desligar um que esteja dormindo, qual executar o novo ou o

- ↳ Interrupção de relógio

↳ preemptivo

## Encerramento não preemptivo

- ↳ só libera a CPU quando quiser

## Encerramento preemptivo

- ↳ Perde a CPU, mesmo contra a vontade

→ tempo

→ prioridade

→ não preemptivo

## Tipos de algoritmos de escalonamento

- \* **Lote**:
  - sem usuários interativos
  - ciclos longos são aceitáveis
  - estima o tempo
- \* **Interativo**:
  - com usuários interativos
  - sem tempo
  - ciclos curtos, todos progredem
- \* **Tempo real**:
  - Requisitos de tempo específico

Como medir a eficiência de um algoritmo de escalonamento?

- \* **Variável número de jobs por tempo** (qualquer sistema)
- \* **Tempo de retorno**: tempo médio do inicio de um job até o fim (sistemas em lote)
- \* **Tempo de reação**: tempo de uma emissão à sua resposta (sistemas interativos)

## Algoritmos por tipo

- \* **sistemas em lote** → FCFS
  - SJF
  - SRTN
- \* **sistemas interativos** → round-robin
  - prioridades
  - filas múltiplas
  - filação justa
- \* **sistemas em tempo real** → rate monotonic scheduling (RMS) **spiral**

Próximo a chegar, próximo a ser executado (FCFS)

\* Ordem de chegada

\* não preemptivo (quanto tempo deseja meu processo?)

\* Simples

\* Não diferencia entre orientado a CPU e orientado a E/S  
prejudica os orientados a E/S

folha mais curta primeiro (SJF)

\* Mais curto primeiro

\* não preemptivo (quanto tempo quiser)

\* menor tempo médio de retorno

\* Todos disponíveis simultaneamente

\* duração dos ciclos de CPU é menor

Próximo de menor tempo restante (SRTN)

\* Igual o SJF, mas é preemptivo

\* Se o tempo do novo é menor do que o restante do atual, então troca

\* Bom desempenho em jobs curtos

spiral

\* Pequenos tempos conhecidos de CPU

## Algoritmo circular round robin

\* Cada processo que ganha o CPU executa durante um tempo

\* Se o processo não terminar no final do tempo (quantum) → volta pro final da fila de processos prontos

### Preemptivo

\* Quanto maior o quantum pior o tempo de reação  
prejudica os orientadores E/S  
demora muito a ser escalonado  
menos preempções

\* Menor o quantum, maior o overhead  
tempo para changamento aproximação do tempo  
de execução

\* Quantum entre 20 e 100 ms

### Escalonamento por prioridades

\* Cada um recebe uma prioridade e executa de acordo com a maior

\* Pode ser periodicamente ajustada

\* Pode ser dinâmica ou estática

\* Classes de prioridades

↳ Processos com a mesma prioridade ficam em uma classe, dentro dessa classe é usado um round robin

- \* Para não ocorrer inanição, os processos que estão na fila de pronto há mais tempo e não executaram aumentam sua prioridade

## Fila Multiplex

- \* Cada Classe de prioridade tem um quantum
- \* Mais prioritários têm um quantum menor
- \* Se o quantum acaba antes do processo concluir, muda de prioridade
- \* Reduz a quantidade de desarmamentos para processos que usam a CPU
- \* Interrupções têm alta prioridade

## Fracção justa

- \* Usuários com muitos processos possuem mais tempo de CPU
- \* A ideia é distribuir a CPU para os usuários e o escalonador garante o processo de forma a respeitar essas frações

## Escalonamento de Threads de usuário

- ↳ Núcleo executa o processo e o runtime divide entre os threads desse processo
- ↳ Preempção voluntária (yield) não pode escalar threads de processos diferentes
- ↳ Qualquer algoritmo pode ser usado

## Escalonamento de threads de núcleo

- ↳ Escalonador decide uma thread de qualquer processo
- ↳ processo pode ou não ser considerado
  - \* pode ocorrer devido ao custo

Pode ter alternância de threads de processos diferentes

## Escalonamento no Linux

- \* Ordem:
  - 1) Deadline (Prazo) (Prazo)
  - 2) tempo real
  - 3) tempo compartilhado

→ ① Deadline

Cada thread needs um tempo para executar cada período e a execução deve ocorrer dentro do prazo.

- Thread com menor prazo executa primeiro
- Usa admissão para garantir todos os prazos

### ⇒ Tempo real

- Não há prazo nem garantia
- Prioridade entre 1 e 99
- Executhe a primeira da fila mais prioritária
- Cada CPU tem sua fila

### FIFO

- Em ordem
- executa de:
  - ↳ Idoquear
  - ↳ liberar a CPU
  - ↳ uma mais prioritaria ficar pronto
- Volta para fila da sua respectiva prioridade

### BR

- Igual FIFO mas:
- Quando uma thread perde seu tempo em execução, descontado
  - Quando o quantum zero, volta ao valor inicial a thread volta ao final da fila

## Multiprocessamento

- Deixar a thread na mesma CPU
  - ↳ aproxita o cache de processador
  - ↳ afinidade de processador
- Periodicamente a carga entre CPUs é均衡ada, levando em consideração afinidade e desempenho.
- Runqueues ajuda na afinidade
  - ↳ por processador
- Escalonadores considera apenas threads escalonáveis
- Threads bloqueadas ficam numa "fila de espera"
- Fila no núcleo com spinlocks, para a exclusão mútua