Algoritmos de Escalonamento

Marcelo Johann

Na aula anterior…

Threads
Sincronização
Semáforos

Plano da aula de Hoje

1. **FIFO**
2. **SJF**
3. **Prioridade**
4. **Round-Robin**
5. **Múltiplas Filas**
6. **Garantido**
7. **Lotérico**
8. **Tempo-Real**

Escalonamento

- Definição e critérios
- Quando ocorre?

1. Ao acontecer a transição “executando” para “bloqueado” (pergunta: quando isso acontece?)
2. Ao acontecer a transição “executando” para “pronto” (pergunta: quando isso acontece?)
3. Ao encerrar-se um processo.

1-2-3: o processo para de executar, e isso dispara o escalonador.

4. Ao acontecer a transição “bloqueado” para “pronto”.
   - Pode ter sido liberado um processo importante.

Vários níveis de escalonamento

- Escalonador de curto prazo
  - Decide da alocação da CPU
  - Tempo de resposta: faixa da ms
- Escalonador de médio prazo
  - Decide do gerenciamento de memória
    - Swap
  - Tempo de resposta: faixa de 100 ms
- Escalonador de longo prazo
  - Gerencia a criação dos processos
  - Escalona os acessos ao disco

Escalonador vs. dispachante

- Distingue-se duas partes:
  - O escalonador é responsável pela escolha do processo “eleito” para usar a CPU.
  - O dispachante é responsável pelo lado técnico de
    - Salvar o processo que estava usando a CPU;
    - Executar o processo eleito na CPU.
    - Isso se chama efetuar a **troca de contexto**.

Preemptar ou não preemptar?

- **Não-preempção**: quando um processo executa, ele não pode ser interrompido por um fator externo.
  - Deixa a CPU ou por vontade própria (sleep/yield), ou porque terminou.
  - Windows 3.1, Apple Mac. OS antigo (<8)
  - Os casos 1-2-3 tinham a ver com não-preempção.
- **Preempção**: ato de interromper a execução de um processo para executar um outro.
  - Sis. Op. modernos (Windows 95+, Linux)
  - Necessita sincronização!
  - Casos 3 e 4 implicam, potencialmente, em pre-empção.

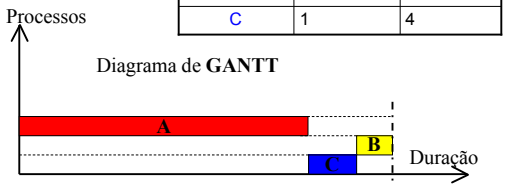
Alguns algoritmos não-preemptivos

- FIFO (First in, First out)
  - Primeiro chegado, primeiro atendido;
  - O mais simples;
- SJF (Shortest Job First)
  - Necessita informações a respeito dos processos e de sua duração!
  - Extremamente eficiente.
- Prioridades
  - Generalização do SJF

Algoritmo First in – First out 1

- O processo pronto há mais tempo passa a usar a CPU.
- Exemplo:

Processo	Hora Chegada	Duração
A	0	24
B	2	3
C	1	4



FIFO – Limitações

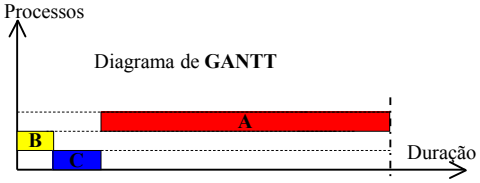
- A duração total não depende da ordem de escalonamento!
- Espera por processo:
  - A espera 0 sec.
  - B espera 22 sec.
  - C espera 23 sec.
- Espera média: (0+22+23)/3 = 15 sec.
- Que tal com a ordem de chegada C-B-A?
  - C espera 0 sec.
  - B espera 2 sec.
  - A espera 6 sec.
- Espera média: (0+2+6)/3 = 2,66 sec!
- FIFO não é nada igualitário, pelo contrário é muito “ditatorial”.
- Outro parâmetro: é inteligente escalonar primeiro os processos I/O bound!

Algoritmo Shortest Job First 2

- Existem sistemas onde se tem informação a respeito dos processos (*jobs*);
  - Sistemas de batches não-interativos
- Em geral, quer-se minimizar o *turnaround* (tempo de espera dos “clientes”);
- Seja t_i o tempo de uso da CPU do processo i (no momento da tomada de decisão!)
  - O dono de i espera $W_i = \Sigma_{j=1,\ldots,i} t_j$ sec.
  - Mostra-se por indução que W_i é mínimo quando os t_j estão em ordem crescente.

SJF em nosso exemplo

Processo	Hora Chegada	Duração
A	0	24
B	0	3
C	0	4



SJF: problema e paliativo

- Só tem um probleminha… é preciso **prever o futuro** tempo de execução do processo!
- Pode-se tentar estimar o tempo t_i baseando-se sobre o passado.
  - Execução passada de um job;
  - Analisando os surtos de CPU no histórico do processo.
- Qual estimação?
  - $t_{i+1} = t_i$: a história se repete constantemente…
  - Mais inteligente: ponderar o histórico global $W_{i-1} = \Sigma_{j=1,\ldots,i-1} t_j$ com a última medição t_j : $t_{i+1} = \alpha t_i + (1-\alpha) W_{i-1}$
  - Fala-se de média exponencial.

Escalonamento por prioridades 3

- No caso do SJF, usou-se o critério "duração" para priorizar os processos.
- Versão mais geral:
  - Decide-se uma prioridade por processo;
  - Processos com mesma prioridade podem ser desempatados através do FIFO (por exemplo).
- Prioridades podem ser atribuídas por ordem crescente (0 é a mais baixa) ou decrescente (0 é a mais alta).
- A prioridade pode ser determinada:
  - Pelo Sis. Op. (a partir do histórico, de seus I/O,…);
  - Pelo usuário (a partir da avaliação da importância do processo);
  - Pelos dois juntos!

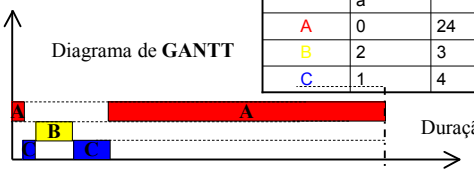
Prioridades - limitações

- **Starvation** (postergação indefinida)
  - Um processo de baixa prioridade pode sempre ser deixado para mais tarde.
  - Não significa que ele nunca será executado, mas que ele PODE nunca ser executado.
  - Solução: aumentar a prioridade com o passar do tempo **(envelhecimento/*aging*)**
- **Inversão de prioridade**
  - Processo que aguarda tem mais prioridade do que o que libera
- Sobretudo: faz sentido que um processo altamente prioritário tenha que esperar até o processo em execução deixe a CPU?
  - Ou seja, prioridade tem a ver com pre-empção!

Preempção ou não-preempção?

- No fundo:
  - Prioridades são mais compatíveis com preempção…
  - SJF pode ser usado com preempção!
    - Se surgir um job mais curto, ele interrompe o que está executando.

Processo	Hora Chegada	Duração
A	0	24
B	2	3
C	1	4



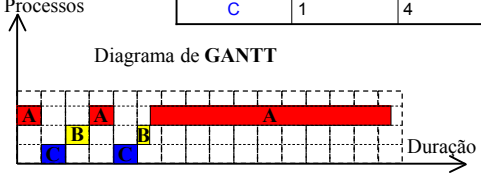
Escalonamento Round Robin 4

- O escalonador define um *quantum* (fatia de tempo) para cada processo.
- Após se encerrar a fatia, o processo escalonado deve ceder o lugar na CPU a um outro.
  - Também se perde a CPU se terminar antes do fim do quantum, ou faz um pedido de E/S, …!
- Mantém-se uma lista circular de processos prontos.
- Deve-se usar um mecanismo de interrupção regular, ocorrendo a cada fatia de tempo.
  - Usa o clock.

Algoritmo R. R. – Diagrama de Gantt

- *Quantum* de 2 unidades de tempo.
- Exemplo:

Processo	Hora Chegada	Duração
A	0	24
B	2	3
C	1	4



Round-Robin: observações

- Deve-se ter um algoritmo de desempate entre os processos prontos quando há uma interrupção…
  - FIFO, SJF, prioridades…
- Se o quantum aumenta muito, se obtém novamente um FIFO.
- Espera média:
  - A: 7
  - B: 6
  - C: 5

  (7+6+5)/3 = 6.
  (comparar com 4 (SJF) e 17 (FIFO))

Round-Robin: limitações e problemas

- O grande interesse do R.R. é sua "equidade"
  - Todos os processos acabam tendo uma chance regular de executar.
- Como definir o *quantum*?
  - Muito grande: = FIFO!
  - Muito pequeno: só se faz troca de contexto…
- Processos *I/O bound* são prejudicados!
  - Esperam tanto como os outros, mas não chegam a usar seu quantum todo!
- Solução: juntar Round Robin e prioridades com preempção

Múltiplas Filas: Prioridade preemptiva 5

- Define-se **prioridades** para os processos.
- Logo que surgir um processo com maior prioridade que o que está executando, ele **preempta** o mesmo que volta para a fila dos "prontos".
- Caso haja mais de um processo com uma dada prioridade, se aplica um segundo algoritmo de desempate.
  - Tipicamente Round-Robin;
  - Também pode ser um FIFO ou SJF.
- Neste caso, obtém-se **uma lista por nível** de prioridades.
  - Quando uma fila está vazia, considera-se a lista de prioridade inferior.
  - Múltiplas filas, com realimentação.
  - Pode ter um algoritmo distinto de desempate em cada fila.

Definição das prioridades

- Estática: a prioridade é dada na criação do processo
  - Pelo Sis. Op., pelo usuário…
  - Problema: há risco de postergação indefinida (starvação) para um processo com baixa prioridade.
- Dinâmica: a prioridade evolui durante o ciclo de vida do processo.
  - Começa com um valor estático
  - Evolui depois:
    - Aumenta a medida que o processo usa a CPU;
    - Aumenta proporcionalmente à fração do *quantum* que **não usou**.
    - Assim, os processos I/O bound voltam na fila de espera com alta prioridade!

Escalonamento Garantido 6

- Em um sistema com *n* processos, idealmente cada processo deve receber *1/n* do tempo
- Nenhum dos algoritmos anteriores oferece qualquer garantia ou tenta se aproximar disso
- Seja *h* o tempo total gasto por um processo na CPU
- Seja *t* o tempo transcorrido desde sua criação
- Então *t/n* é o tempo que esse processo deveria ter usado
- Assim, *f=h/(t/n)* é um fator que determina se o processo usou mais ou menos do que deveria.
- Se um processo tem *f=0.5*, usou metade do tempo justo.
- Pode-se escalonar os processos por menor valor de *f*.

Escalonamento Lotérico 7

(Waldspurger e Weihl, 1994)

- O sistema distribui bilhetes aos processos, e faz um sorteio cada vez que precisa selecionar um processo para a CPU.
- Se cada processo tiver x% dos bilhetes, deve ganhar a CPU x% das vezes
- Em teoria, um processo pode nunca ser sorteado.
- Na prática, as probabilidades garantem que isso não ocorre.
- É um escalonamento responsivo (sem filas).
- Processos prioritários podem ganhar mais bilhetes
- Processos cooperativos podem trocar bilhetes:
  Ex: clientes podem passar bilhetes para servidores

Escalonamento para Tempo Real 8

- Utiliizados em Sistemas Operacionais de Tempo Real

- Hard Real Time
- Soft Real Time
- Processos curtos, previsíveis, tempo conhecido
- Eventos periódicos e aperiódicos

- Sistemas Escalonável: somatórios de tempos/periodos < 1

- Algoritmo de Taxa Monotônico (RM, Liu e Layland, 1973)
- Prazo Final mais Cedo Primeiro (EDF)
- Folga Mínima Primeiro (MLF)

Mecanismo vs Política

- Mecanismo é o Algoritmo e/ou a Estrutura

- Política é o conjunto de regras que é usado para atribuir os números ou classes