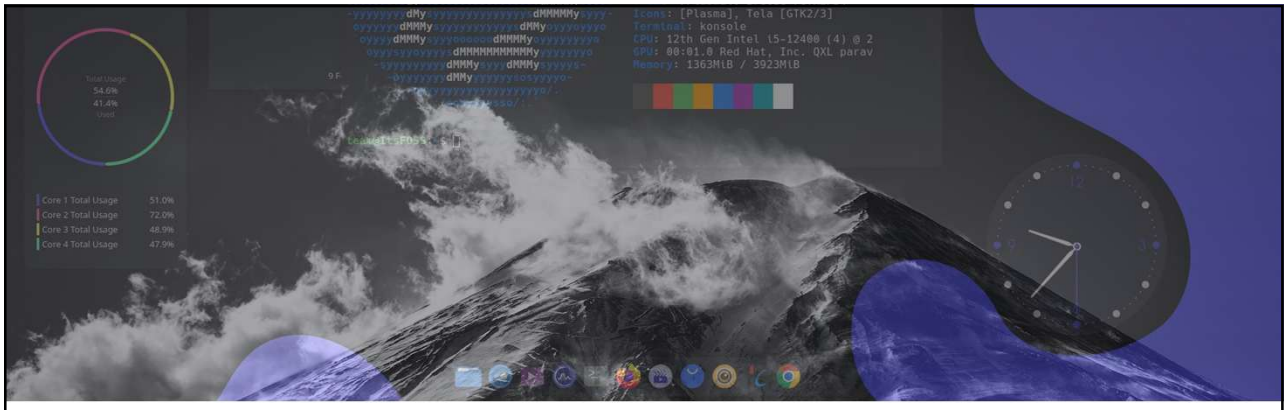


SISTEMAS OPERATIVOS

Escalonamento

António Godinho

ESCOLAMENTO PREEMPTIVO / NÃO PREEMPTIVO

Algoritmos preemptivos:

Forçam o processo a sair em situações que, de outra forma, o processo continuaria a utilizar o processador

Exemplos (depende **muito** do algoritmo de escalonamento)

- Fim de quantum
- Apareceu um processo mais prioritário
- Apareceu um processo mais curto

2

ESCOLAMENTO PREEMPTIVO / NÃO PREEMPTIVO

Algoritmos não premtivos:

Não forçam o processo a sair do processador. O processo só sai quando assim o quer

Exemplos (depende **muito pouco** do algoritmo de escalonamento)

- O programa terminou
- O programa pediu acesso a um semáforo que estava já fechado
- O programa requisitou uma operação de E/S demorada

Observações

- Sistemas de tempo partilhado são sempre preemptivos
- Nem todos os sistemas preemptivos são de tempo partilhado

3

PREEMPÇÃO

Sistemas preemptivos

O processo actualmente em execução pode ser interrompido e colocado no estado "executável" pelo S.O. sem ser por se ter bloqueado ou por ter pedido uma operação de E/S. Pode ocorrer quando:

- Um (novo) processo é posto no estado "executável"
- Uma interrupção que faz com que um processo se desbloqueie ocorre (fim de E/S)
- Periodicamente com base num timer de hardware (= tempo partilhado)

4

PREEMPÇÃO

Sistemas não preemptivos

Quando no estado "em execução", um processo continua a executar-se até:

- Terminar
- Bloquear-se:
  - À espera de conclusão de operação de E/S
  - Por requisição de serviços do S.O. (ex. sincronização para cooperação)

5

ESCALONAMENTO PREEMPTIVO

Objectivos do escalonamento preemptivo:

Permitir que processos prioritários possam reagir rapidamente a um acontecimento (ver também: "objectivos das prioridades")

(O que torna um processo "prioritário" é uma questão independente da preempção)

Indispensável em sistemas de tempo real

Aconselhável em sistemas multi-utilizador

De uma maneira geral, aumenta a estabilidade do sistema

Se um programa "encalhar", os outros não estão automaticamente impedidos de se executarem

Exemplos:

WindowsNT, Linux - Preemptivo

MS-DOS -Não preemptivo

6

ESCALONAMENTO PREEMPTIVO

Exemplo de actuação

Utilização da preempção + mecanismos de sincronização para dar resposta rápida a um acontecimento externo sem ter que estar em ciclo a testar a ocorrência do acontecimento:

1. Um processo prioritário está bloqueado num semáforo (*)
2. Um acontecimento externo provoca Assinalar() nesse semáforo (feito por um device driver, por exemplo)
3. 0 processo prioritário torna-se executável nesse instante
4. Ao processo actualmente em execução é retirado o processador
5. O processo prioritário entra em execução.

(*) semáforo: recurso de sincronização a estudar mais adiante

7

ESCALONAMENTO PREEMPTIVO

A preempção implica:

O despacho deve ser chamado na sequência de todas os acontecimentos susceptíveis de alterar o estado dos processos

- Para além dos que também ocorrem em sistemas não preemptivos:
  - Terminação de processo
  - Operações que levam ao bloqueio (pedido de E/S e sincronização)

Normalmente são:

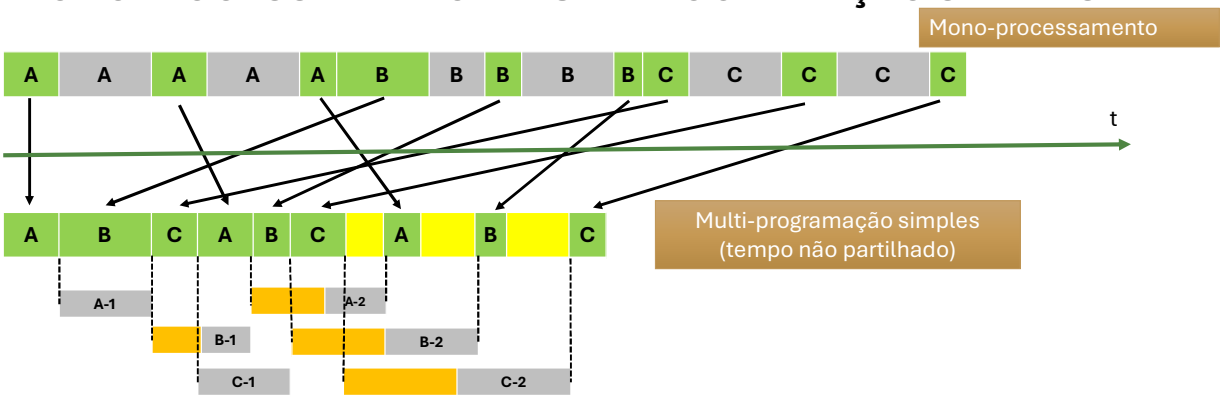
- Fim de quantum (➔ temporizador (periférico) ➔ interrupção)
  - Se o sistemas for de tempo partilhado
- Sinalização de periféricos (➔ device driver ➔ interrupção)
  - Podem levar ao desbloqueio de processos mais prioritários
- Operações de sincronização

Consequências negativas dos sistemas preemptivos

- Mais comutações de processo ➔ menor desempenho global
- Maior complexidade do S.O.

8

MONOPROCESSAMENTO X MULTIPROGRAMAÇÃO SIMPLES



Três procesos: A, B e C

X O processador está atribuído ao processo X

X O processador está inativo (o processo X está a ocupá-lo)

O processador está inativo (nenhum dos processos pode avançar)

X O processo X está à espera da conclusão de uma operação (E/S) e não pode avançar

O recurso pedido está ocupado

9

MONOPROCESSAMENTO X MULTIPROGRAMAÇÃO SIMPLES

Como ler o slide anterior

É apresentada a execução do mesmo cenário com três programas segundo monoprocessamento e segundo multiprogramação simples

- Blocos a cinzento representam momentos em que o programa em questão precisa do processador
- Blocos branco/cinzento na diagonal representam momentos em que o programa em questão está a efectuar uma operação de I/O e o processador está desocupado
- Blocos branco/cinzento entrecruzado representam momentos em que o programa em questão não está a usar o processador nem o usaria mesmo que este estivesse vazio
- Blocos branco/cinzento na vertical representam momentos em que o programa em questão estaria a usar o processador se este estivesse livre

O slide apresenta uma comparação do tempo total de execução com os mesmos processos em monoprocessmento e em multiprogramação simples.

É notória a diminuição do tempo total de execução na multiprogramação simples pelo facto de se aproveitarem os tempos d E/S de um programa para executar outro

10

TEMPO NÃO PARTILHADO X TEMPO PARTILHADO

No slide seguinte

É feita uma comparação entre multiprogramação simples (tempo não partilhado) e multiprogramação preemptiva (tempo partilhado)

É usado o mesmo cenário (os mesmos três processos)

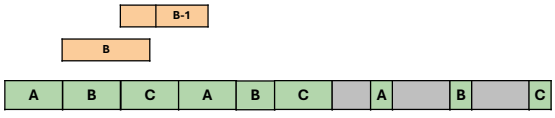
No escalonamento preemptivo está a assumir-se um tempo máximo para cada programa, findo o qual o sistema força a passagem do processador para outro programa (de uma forma circular)

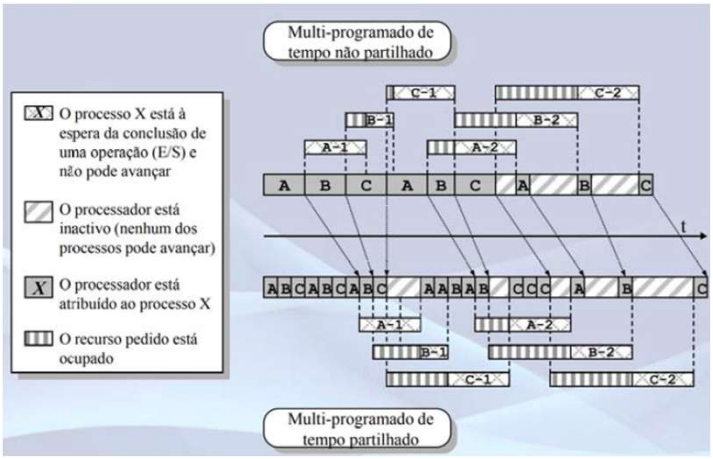
O tempo total de execução na multiprogramação preemptiva aumento face ao da programação simples.

- Este fenómeno acontece na realidade, mas não na escala vista no slide (a diferença é imperceptível)
- Este aumento é explicado pelo maior número de trocas entre processos (essas trocas não são instantâneas e gastam tempo)
- Neste exemplo, há poucos processos no exemplo e são demasiado parecidos no seu perfil usa CPU-faz E/S, levando-os a uma competição exacerbada pelo processados nos mesmos instantes.

11

TEMPO NÃO PARTILHADO X TEMPO PARTILHADO





12

TEMPO NÃO PARTILHADO X TEMPO PARTILHADO

Observações relativas aos exemplos apresentados

1. Até aos sistemas multi-programados:
   - Está-se a optimizar a utilização do processador
     - (Manter o processador sempre a trabalhar, se possível)
2. A partir dos sistemas de tempo partilhado:
   - Está-se a optimizar a interactividade
     - (rotatividade dos processos)
   - Por vezes pode levar a alguma perda de eficiência devido a:
     - Acumulação de pedidos dos mesmos recursos nos mesmos intervalos de tempo (os processos acabam por ficar bloqueados não por o processador estar ocupado mas porque um recurso qualquer está ocupado)
     - Devido a maior overhead do S.O. na comutação de processos
       - (a comutação ocorre mais vezes)
       - (este último aspecto não está representado graficamente nos exemplos)

13

TEMPO NÃO PARTILHADO X TEMPO PARTILHADO

Observações relativas aos exemplos apresentados

3. O exemplo apresentado é pouco comum (é uma simplificação):
   - Raramente todos os processos requerem o mesmo recurso
     - (nem todos os recursos são mutuamente exclusivos)
   - Geralmente existem muito mais que 3 processos a decorrer
     - Os tempos inactivos do processador seriam aproveitados para esses outros processos ⇒ Não existirão tantos intervalos de tempo de inactividade
   - Ou seja: comparativamente aos sistemas de tempo não partilhado, os sistemas de tempo partilhado não são tão ineficientes como os exemplos anteriores fazem crer

14

PRIORIDADES

Atributo inerente aos processos (e threads) e característico de alguns algoritmos de escalonamento, que permite a um processo ser atendido por outra ordem que não a de chegada (FIFO)

Existem duas variantes:

Prioridades fixas

- Estão relacionadas com o tipo de processo

Prioridades dinâmicas

- Estão relacionadas com o tipo de comportamento do processo no sistema ao longo do tempo

Prioridades não são um indicador a maximizar ou optimizar mas sim uma característica utilizada (de diversas formas) por vários algoritmos de escalonamento

15

PRIORIDADES FIXAS E DINÂMICAS

Prioridades fixas x prioridades dinâmicas

Fixas: estão associadas a acontecimentos

- Estão relacionadas com a importância dos acontecimentos.
- Sempre que tais acontecimentos se produzem, se forem muito importantes, deverão ser imediatamente atendidos, logo o processo associado deve ter uma prioridade alta

Exemplo: processo paginador

Dinâmicas: estão associadas ao comportamento do processo no sistema

- O comportamento pode variar bastante, em função disso, o processo irá ter a sua prioridade aumentada ou diminuída
- O objectivo é conseguir uma tratamento justo a todos os processos (evitar a monopolização de um recurso por um processo e a situação de starvation noutros)

16

PRIORIDADES DINÂMICAS

Utilizações possíveis - Exemplo com prioridades fixas

Cabe ao programador (utilizador) definir qual a prioridade que pretende atribuir ao processo

Exemplo (UNIX)

nice <valor> <programa>

Permite correr um determinado programa (identificado pelo comando) com uma prioridade sem ser a por omissão. Exceptuando o root, apenas se pode baixar a prioridade, sendo assim simpático (nice) para os outros processos (e utilizadores).

Desvantagens: questões de privilégio e permissões

- Não se pode permitir baixar/aumentar a prioridade de arbitrariamente

Questões:

- Quais os processos aos quais se pode mudar a prioridade?
- Quais os valores admissíveis?

17

PRIORIDADES DINÂMICAS

Utilizações possíveis - Exemplo com prioridades dinâmicas

O escalonamento atribui prioridades aos processos e modifica-as de acordo com o seu comportamento no sistema

- Que recursos está a utilizar / pretende utilizar
- Quanta atenção já teve por parte do processador
- Qual a idade do processo

Dá-se prioridade aos processos que melhor se encaixam nos critérios de optimização global da máquina

Não se pode falar de um critério universal mas sim de um critério para cada tipo de S.O. e utilização pretendida

18

ESCALONADORES

Escalonadores no sistema operativo:

- Conceptualmente existem 3:

Escalonador de longo prazo

Responsável por admitir um processo a execução (transição para o estado novo), dependendo da carga da máquina

Escalonador de médio prazo

Implementa um algoritmo de escalonamento para determinar ***quando*** e ***qual*** dos processos actualmente no estado executável deverá obter o processador.

Escalonador de curto prazo

- Também designado como ***despacho***
- Efectua a comutação entre o processo que está a executar e o próximo (já decidido pelo escalonados de médio prazo)

19

ESCALONADORES

Escalonamento de longo prazo

Responsável por admitir mais um processo a execução (aceitar o trabalho)

⇒ Controla o grau de multiprogramação

Duas decisões

Será que a máquina "aguenta" com mais um processo ?

Se não, quais é que são aceites e quais é que são recusados?

I/0 bound? ou CPU bound?

Qual a situação actual do sistema?

20

ESCALONADORES

Escalonamento de longo prazo (cont.)

Um número maior de processos em simultâneo significa maior número de tarefas a serem efectuadas em simultâneo pela máquina mas também significa maior tempo de latência e de espera pela parte de cada processo (os sistema demora mais tempo a "dar a volta" até atribuir novamente o processador a cada processo)

Escalonamento de médio prazo

Objectivo: preparar os processos executáveis para que possam executar-se imediatamente assim que forem seleccionados

- Relacionado com a gestão de memória (em particular com o swaping)
  Traz o processo do disco para memória (sistemas com memória virtual)
  Relacionado com o facto de a memória ser também um recurso cuja utilização deve ser planeada

21

ESCALONADORES

Escalonamento de curto prazo

- Também conhecido apenas por "**despacho**"
- Invocado sempre que pode ocorrer uma comutação de processos
  - Aspecto fortemente dependente do algoritmo de escalonamento implementado pelo S.O.

Exemplos:

- Interrupção do relógio (fim de quantum)
- Interrupção de um periférico (fim de operação de E/S)
- Chamadas ao sistema
- Envio/Recepção de sinais

Trabalho realizado

- Retira o processo actual de execução, obtém o próximo processo da lista e coloca-o em execução

22

DESPACHO

Tempo de latência do despacho

(tempo que demora a comutação de processos)

= tempo de parar um processo + tempo de recomeçar outro processo

Acções envolvidas na comutação de processo:

- Invocação despacho: passar o processador a modo de execução núcleo e saltar para o código do despacho
- Salvaguardar o contexto de software e de hardware do processo anteriormente em execução
- Recuperar o contexto de software e de hardware do processo a por em execução
- Saltar para a localização adequada (onde ia antes) dentro do processo que está a ser posto em execução, colocando o processador em modo de execução utilizador

23

ALGUMAS QUESTÕES DE REVISÃO

- Qual a diferença entre multiprogramação simples e multiprogramação preemptiva
- Que tipos de preempção pode haver no escalonamento preemptivo?
- Como é que um computador só com um processador/núcleo consegue executar vários programas "em simultâneo"?
- Se um programa entrar em ciclo infinito "encalha" a máquina toda? Dê a sua resposta no contexto de multiprogramação simples e depois multiprogramação preemptiva
- O que é o pseudo-paralelismo?
- Quais são as vantagens que a multiprogramação preemptiva traz em relação à multiprogramação simples?
- Num cenário real, poderá um sistema preemptivo ter menos performance global em relação a um sistema multiprogramado mas não preemptivo? Explique.
- Qual destes é mais estável e porquê: monoprocesamento, multiprogramação simples, multiprogramação preemptiva?
- Que tipo de prioridades conhece. Para que serve cada tipo. Dê exemplos de aplicação. Existem vários escalonadores? Quais/para quê?

24

25

GESTÃO DO PROCESSADOR - ESCALONAMENTO

Conceito de estado de processos

A retirar aqui:

- **Conceito de estado de processo**
- **Lógica de cada estado e transições que podem ocorrer**
- **Relevância do conceito de estado de processo para o escalonamento**

PROCESSOS - ESTADOS

Estado do processo

- Descreve a situação de um processo num dado instante em termos de "prontidão" para usufruir do processador

Até aos sistemas de processamento em lote

- Só existe um processo em cada instante

  Esse está sempre em execução, mesmo que bloqueado à de qualquer operação demorada (nenhum outro processo é posto em execução enquanto o anterior não tiver terminado).

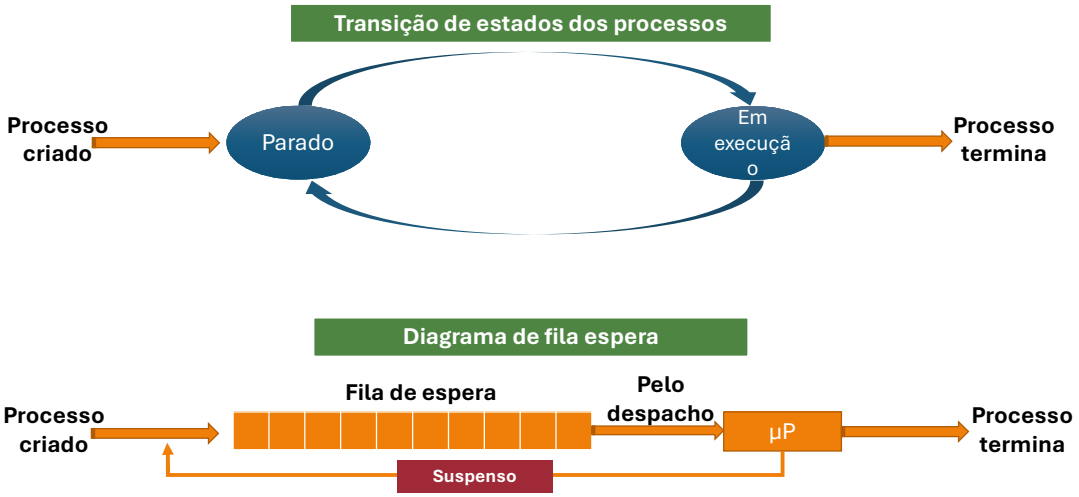
  Considera-se apenas um estado: "em execução"

A partir dos sistemas multi-programados

- Existem vários processos em execução pseudo-paralela. Interessa distinguir a situação exacta em que cada um se encontra.
- Um processo não consegue ser descrito por apenas por "em execução"
- Pode estar em execução, bloqueado não executável, ou ainda executável à espera de vez no processador

26

PROCESSOS – ESTADOS NUM SISTEMA MONOPROGRAMADO

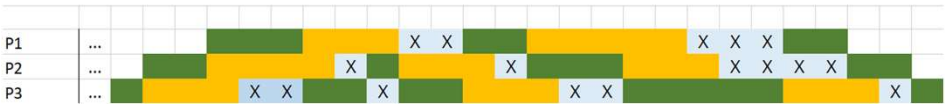


27

PROCESSOS – ESTADOS NUM SISTEMA MONOPROGRAMADO

Dois estados são insuficientes para descrever processos em sistemas multiprogramados

Exemplo visual:



- O processo está em execução. Apenas um processo pode estar neste estado num dado instante
  Corresponde ao estado em execução
- O processo está bloqueado à espera da conclusão de uma operação demorada que não necessita do processador. Ao entrar neste estado, o processo liberta o processador
  Corresponde ao estado bloqueado
- X O processo já está pronto para se executar (novamente) mas o µP ainda está ocupado por outro processo
  Corresponde ao estado executável

28

PROCESSOS – ESTADOS NUM SISTEMA MONOPROGRAMADO

Observações ao exemplo anterior

No exemplo apresentado, existem intervalos de tempo em que o processador não faz nada porque todos os três processos estão à espera da conclusão da operação que iniciaram

As "operações demoradas" não têm necessariamente que ser de E/S.

- Podem ser vulgares esperas de sincronização com outros processos

Questão importante

Quanto dois processos se encontram prontos para serem executados, qual deles será o primeiro a ficar com o processador ?

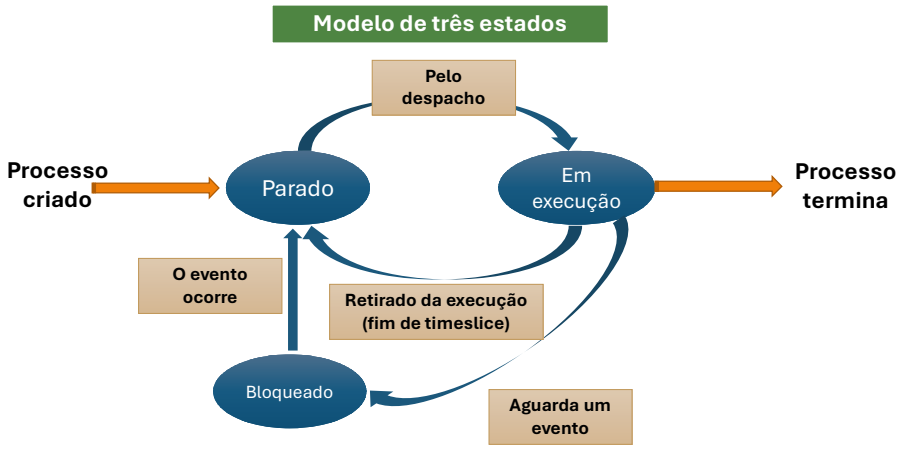
(Esta situação ocorre no exemplo apresentado)

➤ Esta questão é determinada pelo **algoritmo de escalonamento**

(Mais adiante irão ser vistos vários algoritmos de escalonamento)

29

PROCESSOS – ESTADOS



Timesplice = quantum

Alguns sistemas permitem a criação de processos no estado inicial de bloqueado

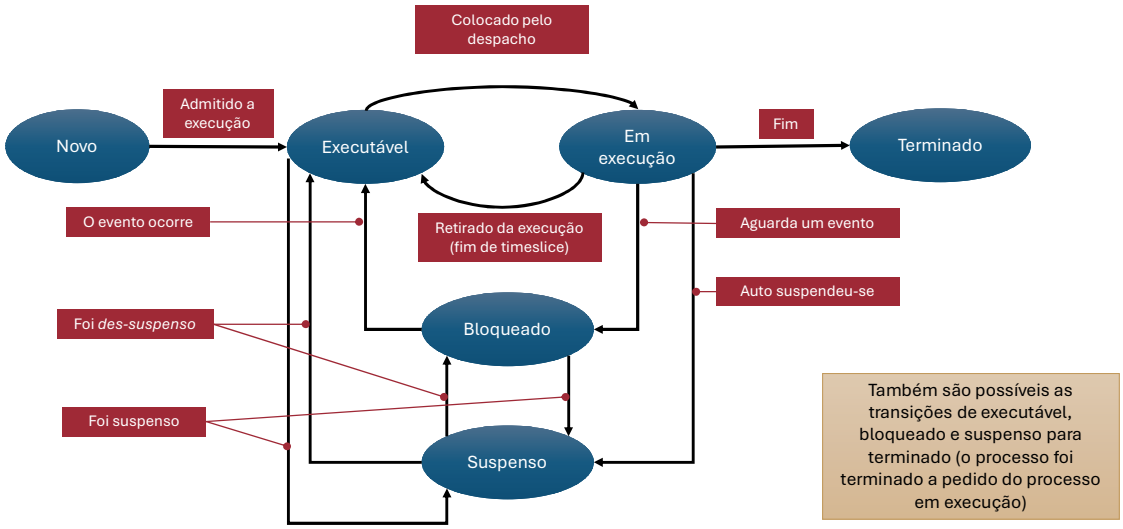
30

PROCESSOS – ESTADOS ADICIONAIS

- Geralmente é necessário manter informação acerca de processos:
  - Cuja criação já foi pedida mas ainda não foram admitidos a execução por falta de recursos da máquina ➔ estado "**novo**"
  - Já terminados mas a informação resultante da sua terminação ainda não foi aproveitada ➔ estado "**terminado**"
- Um processo pode ter a sua execução suspensa:
  - Esta situação pode ocorrer a pedido do próprio processo ou a pedido de outro processo
  - Corresponde a uma situação semelhante ao estado bloqueado. No entanto, não é equivalente pois o processo não está necessariamente à espera de um acontecimento
  - Corresponde a: ➔ estado "**suspenso**"

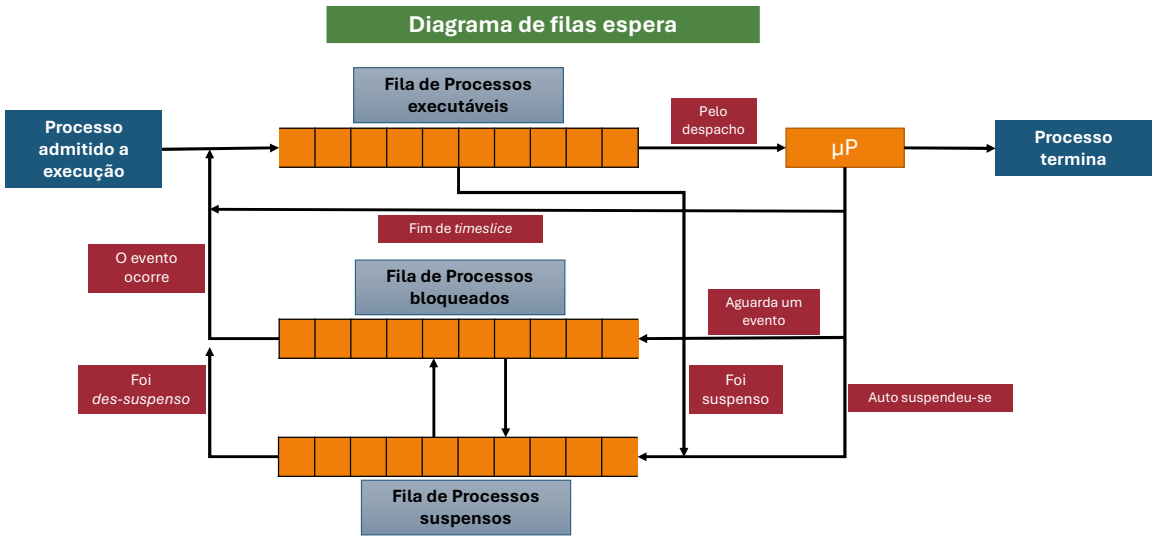
31

PROCESSOS – ESTADOS – MODELO COMPLETO



32

PROCESSOS – ESTADOS: GESTÃO POR FILAS DE ESPERA



33

ALGUMAS QUESTÕES DE REVISÃO

- Descreva o conceito de "estado de um processo" e indique a sua utilidade.
- Quais os estados que conhece e respectivo significado?
- Que transições de estado podem ocorrer e em que circunstâncias?
- Um processo pode passar directamente de bloqueado para em execução? Em que circunstâncias/porquê?
- Qual a diferença entre suspenso e bloqueado?
- O que é o escalonamento por níveis e para que serve?

34

35

GESTÃO DO PROCESSADOR - ESCALONAMENTO

Parte - Conceitos usados em escalonamento

A retirar aqui:

- **Dependência entre escalonador e restante gestão da máquina**
- **Variáveis na escolha/configuração do escalonamento**
- **Tipos de Processos: cpu bound e I/O bound**
- **Conceitos de Justiça e de starvation**
- **Previsibilidade e equilíbrio de recursos**

GESTÃO DO PROCESSADOR - ESCALONAMENTO

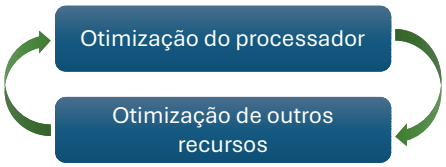
Uma gestão verdadeiramente optimizada da máquina implica gerir todos os recursos em sintonia. O processador é apenas mais um recurso

➔ Pode ser impossível gerir eficientemente um recurso se não se tiver em conta a utilização dos restantes.

➔ A gestão do processador está relacionada com a gestão dos outros recursos da máquina.

Exemplos:

- Não vale a pena tentar optimizar o ***processador*** colocando vários processos em execução se não houver ***memória*** para todos
- Evitar colocar em execução simultânea processos que utilizam o mesmo determinado recurso que os levaria a ficar em espera uns pelos outros



São mutuamente dependentes

36

GESTÃO DO PROCESSADOR / ESCALONAMENTO

➔A gestão processador está directamente relacionada com o tipo de utilização que se pretende dar à máquina

Exemplos de questões / objectivos

- A máquina vai ser utilizada de forma interactiva?

  Diminuir a latência ou diminuir quantum
- A máquina vai ser utilizada para processamento não-interactiva?

  Diminuir número de comutação de processos / aumentar quantum
- A máquina vai ser usada em ambiente de controlo?

  Garantir o cumprimento de prazos para a conclusão das tarefas
- Existirão processos/threads ou utilizadores mais importantes que outros?

  Implementar o conceito de prioridades

Razão da existencia de vários algoritmos de escalonamento diferentes:

Não é possível optimizar todos os objectivos em simultâneo. Cada algoritmo optimiza um ou alguns objectivos específicos em detrimento de outros.

37

GESTÃO DO PROCESSADOR / ESCALONAMENTO

➔ Diferentes algoritmos de escalonamento estão relacionados com respostas diferentes a questões tais como:

- Como é escolhido o próximo processo (ou thread) a executar ?
- Deverá existir um quantum ?
- Qual a duração do quantum (se existir)?
- Deverá ser implementado o conceito de prioridade?

➔ A forma como a gestão do processador é feita constitui uma dos factores mais decisivos relativamente a características importantes tais como:

- Desempenho
- Suporte para multi-utilizador
- Robustez e fiabilidade

38

GESTÃO DO PROCESSADOR / ESCALONAMENTO

Alguns conceitos utilizados na gestão do processador

- Previsibilidade
- Equilíbrio de recursos
- Justiça e Starvation
- Processos I/O bound e CPU bound

Importante:

- A matéria deste capítulo tanto se aplica a processos como threads
- Por simplicidade de escrita apenas é referido "processos" quando poderia ser "*processos/threads*"

39

CONCEITOS DE ESCALONAMENTO

Tipos de processos

Processos I/O bound

São os processos que fazem muitas operações E/S, bloqueando-se frequentemente

Processos CPU bound

São os processos que utilizam intensivamente o processador, quase nunca se bloqueando

Equilíbrio de recursos

O algoritmo de escalonamento deve tentar manter todos os recursos do sistema igualmente ocupados.

➔ Objectivo: tentar evitar os picos de utilização (que levam a esperas)

Exemplo

Processos que utilizem pouco recursos frequentemente requisitados por outros processos devem ser beneficiados sempre que esses outros recursos estejam sobrecarregados.

40

CONCEITOS DE ESCALONAMENTO

Justiça: algoritmos de escalonamento justos

Na ausência de informação em contrário, todos os processos devem ter a mesma utilização dos recursos do sistema (incluindo o processador)

Devem ser evitadas situações de ***starvation***

Um processo que nunca obtém o recurso pretendido porque quando o requer já está outro processo com esse recurso, e quando o recurso está livre, ou está bloqueado, ou não é o próximo a executar-se

A utilização de prioridades dinâmicas permite evitar facilmente situações de starvation, aumentando a prioridade dos processos que já estejam bloqueados à mais tempo que o devido.

41

ALGUMAS QUESTÕES DE REVISÃO

- O que entende por conceito de justiça no escalonamento?
- No contexto de escalonamento, indique cenários em que a justiça é desejável e cenários em que a justiça não é desejável.
- O que entende pelo conceito de starvation?
- Descreva o que entende por processo CPU Bound e por processo I/O Bound.
- Um processo pode ser simultaneamente CPU bound e I/O bound? Explique. O que entende por equilíbrio de recursos?
- Qual a relação entre a gestão do processador e a gestão do resto da máquina?

42

43

GESTÃO DO PROCESSADOR / ESCALONAMENTO

Parte - Indicadores de desempenho

A retirar aqui:

Capacidade de observar um algoritmo de escalonamento e medir o seu desempenho

Capacidade de comparar algoritmos de escalonamento em cenários específicos, escolhendo e justificando qual o melhor nesse cenário

Entender:

- **Dependência entre escalonador e restante gestão da máquina**
- **Quais as variáveis na escolha/configuração do escalonamento**
- **Tipos de Processos: cpu bound e I/O bound**
- **Conceitos de Justiça e de starvation**
- **Previsibilidade e equilíbrio de recursos**

INDICADORES DE DESEMPENHO

Indicadores do desempenho do escalonamento

- **Permitem aferir a eficiência com que o sistema gere a utilização do processador**

  Baseiem-se em observações simples relativas à execução dos processos

  Permitem comparar:

  Algoritmos de escalonamento diferentes (em situações semelhantes)

  Eficiência de um algoritmo de escalonamento em situações diferentes

44

GESTÃO DO PROCESSADOR / ESCALONAMENTO

Métricas/indicadores de desempenhos:

- Utilização do processador
- Throughput
- Tempo de conclusão (turnaround)
- Prazos
- Tempo de resposta (latência)
- Razão de penalização
- Tempo de espera

Importante:

Esta matéria tanto se aplica a processos como threads

Por simplicidade de escrita apenas é referido "processos" quando poderia ser "processos/threads"

45

INDICADORES DE DESEMPENHO

Utilização do processador

Percentagem de tempo em que o processador está a ser utilizado

- Em sistemas (hardware) muito caros torna-se um indicador importante pois indica em que grau o investimento está a ser aproveitado
- Indicador muito importante em sistemas multiprogramados

  (Por se terem vários processos em execução simultânea)
- Indicador menos importante em sistemas:

  De tempo real: existem outros indicadores mais importantes

  Mono-utilizador: por só se ter um utilizador

  ➔ Idealmente tem-se 100% de utilização. No entanto este valor é enganador:

  Se se tiver menos: pode ser apenas sinal de pouca carga e não de um mau algoritmo

  100%: pode ser sinal de carga exagerada e não de um escalonamento eficaz

  Pouco interessante em sistemas interactivos ou com aplicações interactivas

46

INDICADORES DE DESEMPENHO

Throughput

Quantidade de processos concluídos por unidade de tempo

- Mede a quantidade de trabalho efectuado (esperas e bloqueios não constituem trabalho útil)
- É bastante variável em função da carga e duração dos processos
  - Pode vir a ser uma medida de comparação enganadora
- Varia bastante consoante o tipo de algoritmo
  - Constitui uma boa medida para comparar diferentes algoritmos de escalonamento ***desde que com carga idêntica (número e tipo de processos)***
- Interessa maximizar este valor tanto quanto possível

47

INDICADORES DE DESEMPENHO

Tempo de conclusão (*Turnaround*)

Intervalo de tempo desde a criação do processo até à sua conclusão (Inclui tempos de espera por recursos, inclusive pelo processador)

- Este indicador varia bastante com
  - Duração (natureza) dos processos
  - Carga
- Em sistemas interactivos
  - Um processo longo pode ser tolerado desde que a interactividade com o utilizador não se degrade
  - ➔ O indicador "Tempo de resposta" torna-se mais interessante para estes sistemas

48

INDICADORES DE DESEMPENHO

Prazos / *deadlines* (aplica-se mais a sistemas de tempo real)

Se existirem prazos de execução (ou de resposta) e esses prazos forem especificados, então esses prazos devem ser levados em conta pelo algoritmo de escalonamento

Interessa maximizar a percentagem de prazos cumpridos

Exemplo

Três processos com um prazo de 15 min e outros 3 com um prazo de 120 min

Algoritmo A: todos os prazos são cumpridos à tangente

Algoritmo B: os prazos dos processos de 15 min são **ligeiramente** excedidos enquanto os de 120 min são cumpridos com **grande** margem

➔ Aparentemente o algoritmo ***B*** seria melhor pois, **em média**, o ***throughput*** eo ***turnaround*** são melhores, mas a percentagem de prazos cumpridos é pior, pelo que o algoritmo **A**, **segundo este indicador**, é o melhor.

49

INDICADORES DE DESEMPENHO

Tempo de resposta (ou latência)

Tempo que demora a obtenção de uma resposta a uma acção do utilizador (ou 1ª resposta desde o inicio da execução)

Só se aplica a sistemas multiprogramados interactivos

É possível a um processo produzir a resposta esperada enquanto vai fazendo outras coisas (pode fazer mais trabalho que o que se "vê")

- É um indicador mais difícil de medir que o tempo de conclusão

Em sistemas interactivos é mais significativo que o throughput

- A qualidade apercebida pelo utilizador torna-se mais importante

Objectivos a atingir:

Diminuir tanto quanto possível o tempo de resposta

Manter o maior número possível de processos com tempo de resposta aceitável

➔ Reparar que são objectivos parcialmente contraditórios

50

INDICADORES DE DESEMPENHO

Razão de penalização (*penalty ratio*)

Razão entre o tempo de existência total do processo no sistema sobre o tempo que o processo levaria a executar-se se nunca tivesse que esperar por outros processos

➔ Indica a penalização sofrida devido à existência de outros processos

É um bom indicador para comparação entre diferentes algoritmos de escalonamento (desde que com carga e processos idênticos)

➔ Idealmente deve ter o valor 1.0

51

INDICADORES DE DESEMPENHO

Exemplo: uma situação concreta e objectiva

Dado um cenário constituído por processos, sendo conhecidos (dados);

- A natureza do processos: CPU bound (não fazem operações I/O)
- A duração dos processos
- Os instantes em que os processos chegaram
- O algoritmo de escalonamento usado (FCFS: atendo os processos por ordem de chegada a ver mais adiante)

É feita uma pergunta:

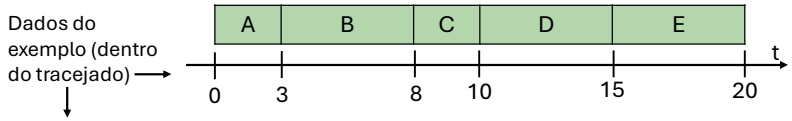
- Em que medida os processos foram penalizados por estarem a partilhar a máquina com uns com os outros?
- Resolução
  - Determinar os instantes de execução dos processos
  - Calcular o penalty ratio médio
- ➔ *Slide seguinte*

52

INDICADORES DE DESEMPENHO

Exemplo

5 processos do tipo CPU bound: os tempos de chegada e duração de execução são dados O escalonamento é multiprogramado simples do tipo first in first out não preemptivo



	Chegada	Duração	Atendimento	Conclusão	*Penalty Ratio*
A	0	3	0	3	3/3 = **1.0**
B	1	5	3	8	7/5 = **1.4**
C	3	2	8	10	7/2 = **3.5**
D	9	5	10	15	6/5 = **1.2**
E	12	5	15	20	8/5 = **1.6**

(Média = 1.74)

53

INDICADORES DE DESEMPENHO

Tempo de espera

Tempo total que um processo passou no estado executável (pronto para correr mas à espera do processador)

- Mede o tempo que os processos passam à espera que outros processos libertem o processador
- Permite também ver em que grau o processador está ou não a ser aproveitado
- Idealmente, os processos nunca teriam que esperar uns pelos outros

54

INDICADORES DE DESEMPENHO

Previsibilidade

Idealmente, um processo deve sempre necessitar de aproximadamente o mesmo tempo para se executar independentemente da carga do sistema

- É uma medida mais qualitativa que quantitativa
- É de implementação complexa

  O SO terá que manter registo dos tempos de execução normais dos processos para saber se estão atrasados (se estiverem, serãoseleccionados para execução mais frequentemente que os que não estão)

55

GESTÃO DO PROCESSADOR

Objectivos a atingir pelo escalonador do S.O.:

- Minimizar a latência
- Maximizar o throughput
- Maximizar o equilíbrio da utilização dos periféricos
  - Manter os dispositivos ocupados
- Justiça
  - Todos (processos / utilizadores) obtêm o mesmo progresso

Dificuldades

Dispositivos de E/S podem ficar inactivos devido à dificuldade em prever o tipo dos processos (CPU bound ou I/O bound)

A optimização pode causar inadvertidamente starvation de processos

Possibilidade de ocorrência de ***deadlocks***

56

ALGUMAS QUESTÕES DE REVISÃO

Estas questão são de âmbito aberto (não têm um texto fixo)

- Explique o indicador de desempenho X
  - (substitua X por cada um dos indicadores vistos aqui)

- Dado um cenário com alguns processos e dois algoritmos de escalonamento (tal como o exemplo atrás), verifique que consegue:
  - Calcular cada uma das várias métricas e indicadores explicados
  - Para cada indicador, identificar qual o melhor algoritmo

- Verifique que, dado o cenário de utilização da máquina, consegue identificar quais os indicadores de desempenho mais adequados e quais os menos adequados, sempre justificando.

57

58

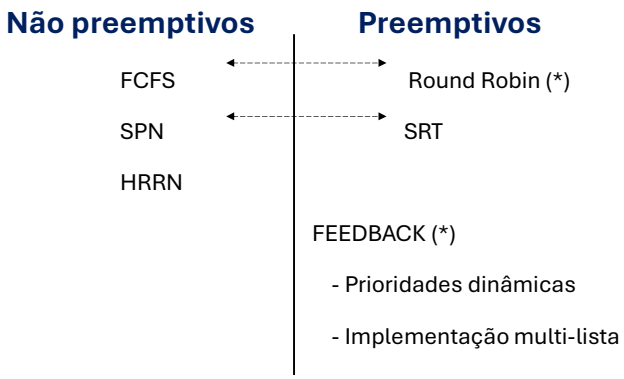
GESTÃO DO PROCESSADOR - ESCALONAMENTO

Parte 7 - Algoritmos de escalonamento

A retirar aqui:

- **Funcionamento dos algoritmos: FCFS, Round Robin, SPN, SRT, HRRN**
- **Escalonamento com feedback, escalonamento multilista, escalabilidade no escalonamento**

ALGORITMOS DE ESCALONAMENTO



(*) = baseados num *quantum* (tempo partilhado)

59

ALGORITMOS DE ESCALONAMENTO

Algoritmo ***FCFS - First Come First Served***

- Os processos são atendidos pela ordem de chegada à lista de executáveis. Não é preemptivo
  - ➔ Não serve para sistemas multiprogramados de tempo partilhado
- Os processos muito curtos podem ficar desnecessariamente à espera da conclusão de processos muito longos; tem um tempo médio de espera mau
- **Favorece os processos *CPU Bound*** à custa do bloqueio frequente dos processos *I/O Bound*; não optimiza a utilização dos dispositivos
- Adequado apenas para situações em que todos os processos são não interactivos, têm aproximadamente a mesma duração, o tempo de conclusão não é importante e não se pretende tempo partilhado
- Para arquitectura com apenas um processador é conveniente utilizar uma variante que utilize prioridades

60

ALGORITMOS DE ESCALONAMENTO

Algoritmo ***Round Robin***

Versão preemptiva do algoritmo FCFS com base num intervalo de tempo (*quantum*)

Objectivo: reduzir a penalização sofrida pelos processos pequenos face aos processos longos

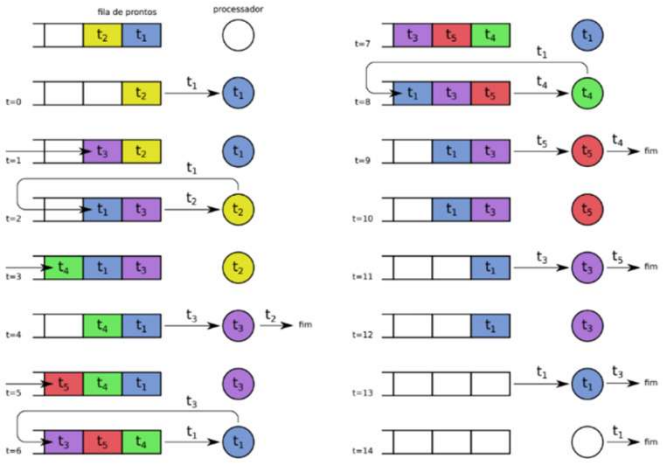
Solução para um processo a monopolizar a CPU: interrompe-se o processo

- Os processos continuam a ser executados por ordem de chegada à lista dos executáveis
- Quando findo o seu *quantum*, o processo volta para o fim da lista dos executáveis
- Pode-se visualizar a lista dos processos executáveis como sendo circular
- Corresponde ao algoritmo mais simples através do qual se consegue obter o tempo partilhado em sistemas multiprogramados

61

ALGORITMOS DE ESCALONAMENTO

Algoritmo ***Round Robin***



62

ALGORITMOS DE ESCALONAMENTO

Principal questão neste algoritmo: qual o tamanho do quantum?

Se for demasiado pequeno, haverá perda de eficiência causada pelo aumento de comutações de processos

Se for demasiado grande, perde-se a interactividade

- Se o *quantum* for maior ou igual que a duração do processo mais demorado, então este algoritmo decai no FCFS

Se for inferior (por pouco) a uma *interação típica*, os processos tenderão a necessitar sempre de mais *quantuns* para completarem o mesmo trabalho

Idealmente o *quantum* deve ser ligeiramente superior à duração de uma interacção típica

Algumas variantes deste algoritmo permitem quantum variáveis com o tempo

carga muito grande ➔ quantuns maiores ➔ menos overhead

63

ALGORITMOS DE ESCALONAMENTO

O algoritmo Round Robin é justo: todos os processos, longos ou curtos, importantes ou não, são atendidos da mesma forma

Mas isto pode não ser uma vantagem: pode ser importante ou necessário dar mais importância a alguns processos (ex. processos do próprio S.O. ou do superuser)

Vantagens do algoritmo Round Robin

Nenhum processo consegue por si só causar *starvation* de outro

Turnaround e Tempo de espera (valores médios) bons (melhores que o FCFS) quando a duração dos processos varia

64

ALGORITMOS DE ESCALONAMENTO

Exemplo: uma situação concreta e objectiva

Dado um cenário (com duas variantes A e B) constituído por processos, sendo conhecidos (dados):

A natureza do processos: CPU bound (não fazem operações I/O)

A duração dos processos

Os instantes em que os processos chegaram

É feita uma pergunta:

Qual o algoritmo mais vantajoso para este cenário: o FCFS ou o Round Robin?

Utilize como critérios:

Turnaround

Tempo de espera

➔ Slides seguintes

65

EXEMPLO: ROUND ROBIN X FCFS

Ilustração para o turnaround e tempo de espera (médios)

Cenário A

Três processos A,B,C do tipo *CPU bound*

Tamanho do quantum ➔ "q" = ⊢⊣ ; Tempo total considerado = 18q

Duração dos processos: A=12q; B=3q; C=3q

Ordem de chegada:

- A: instante 0,
- B: instante 1
- C: instante 2

Qual o melhor algoritmo: FCFS ou Round Robin?

Resolução

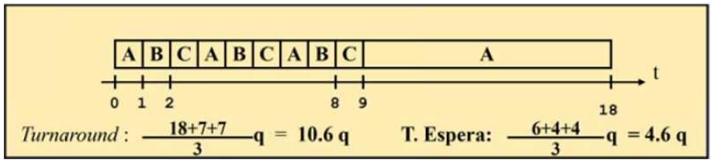
Determinar os instantes de execução dos processos em cada algoritmo

Ver qual o algoritmo que oferece os melhores tempos de turnaround e tempo de espera (o enunciado pediu esses)

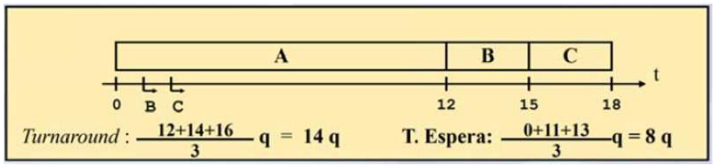
66

EXEMPLO: ROUND ROBIN X FCFS – CENÁRIO A

Round Robin:



FCFS:



Neste cenário, o RR oferece melhores tempos (melhor desempenho)

67

EXEMPLO: ROUND ROBIN X FCFS

Ilustração para o turnaround e tempo de espera (médios)
Cenário B

Duração dos processos: igual para todos = 6q

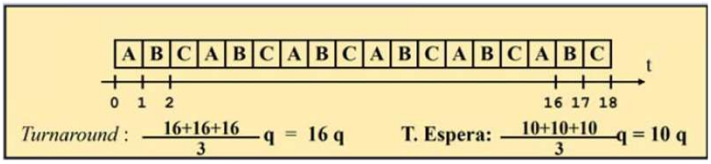
Tudo o resto mantém-se como no cenário A

A mesma pergunta e a mesma lógica de resolução

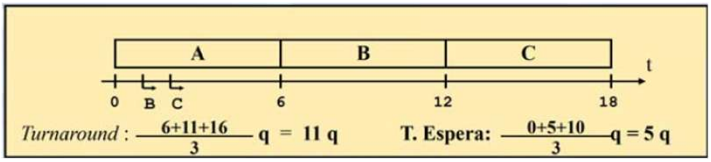
68

EXEMPLO: ROUND ROBIN X FCFS – CENÁRIO A

Round Robin:



FCFS:



Neste cenário, o FCFS oferece melhores tempos (melhor desempenho)

69

EXEMPLO: ROUND ROBIN X FCFS

Ilustração para o *turnaround* e tempo de espera (médios)

Acerca dos resultados

Normalmente o RR oferece melhores desempenho do que o FCFS: a realidade é como no cenário A

No cenário B o FCFS aparentemente é melhor que o RR porque:

- Trata-se de um exemplo muito artificial:
- Apenas 3 processos
- Todos os processos iguais (mesma duração, mesmo tipo de execução, etc.)
- Todos os processos lançados quase ao mesmo tempo

Na realidade os processos são muitos, diferentes uns dos outros e lançados em alturas diferentes, mais parecido com o cenário A

Foram usados: tempos de espera e *turnaround*, porque "o enunciado pediu esses" - podiam ter sidos outros indicadores.

70

ALGORITMOS DE ESCALONAMENTO

Algoritmo ***SPN - Shortest Process Next***

Funcionamento semelhante ao FCFS, mas em que o próximo processo a ser executado é aquele que se espera que demore menos tempo a executar.

Objectivo: diminuir a penalização sofrida pelos processos curtos face aos processos mais demorados

Não é preemptivo

➔Não serve para sistemas multiprogramados de tempo partilhado

- Continua a favorecer os processos *CPU Bound* à custa do bloqueio frequente dos processos *I/O Bound*; no entanto, melhora a utilização dos dispositivos de E/S
- Se estiverem sempre a aparecer novos processos pequenos, os processos longos são bastante penalizados (situação de *starvation*)

71

ALGORITMOS DE ESCALONAMENTO

Algoritmo com o melhor *turnaround*

Atender os processos mais curtos em primeiro lugar melhora o *turnaround* médio mais do que atender processos longos mais tarde o piora

Principal dificuldade neste algoritmo: Como saber qual o tempo que os processos vão demorar?

Possíveis soluções

Utilização de estatísticas quando o mesmo programa é executado muitas vezes

➔ Escalonamento mais pesado e complexo;

➔ Pode ser enganador: alturas diferentes = situações de carga diferentes;

Valores fornecidos pelo utilizador

➔ Podem não ser correctos; o S.O. poderá terminar o processo nestes casos

72

ALGORITMOS DE ESCALONAMENTO

Algoritmo *SRT - Shortest Remaining Time (ou Shortest Time to Completion First - STCF)*

Versão preemptiva do algoritmo SPN

O próximo processo a executar é aquele cujo tempo **restante** de processamento (esperado) é menor

Preemtivo ➔ Se um processo que estava bloqueado passar a executável e tiver um tempo restante de processamento menor do que actualmente em execução, ocorrerá preempção.

Este algoritmo não é baseado num *quantum*:

A situação anterior, o bloqueio do processo anterior e a criação de um novo processo mais curto que o que está em execução são as únicas situações que levam à preempção

Ocorrem menos comutações de processos que no algoritmo *Round Robin*

Menos interactividade que no *Round Robin* mas também menos *overhead*

73

ALGORITMOS DE ESCALONAMENTO

Vantagem sobre o algoritmo FCFS:

- Os processos curtos não são penalizados face aos processos longos

Relativamente ao algoritmo SPN:

- Obteve-se preempção (com as vantagens associadas)
- Mantém-se a possibilidade de *starvation* para os processos longos
- Mantém-se a característica de " melhor *turnaround*“

Principal dificuldade neste algoritmo:

- (Analogamente ao algoritmo SPN)
  ➔ Como saber qual o tempo que os processos vão demorar?

- Adicionalmente
  É necessário manter registo dos tempos de execução, quanto é que já decorreu, e quanto é que falta
  ➔ Escalonamento mais pesado

74

ALGORITMOS DE ESCALONAMENTO

Algoritmo *HRRN - Highest Response Ratio Next*

Objectivo: Eliminar a possibilidade dos processos mais longos poderem sofrer starvation tal como pode acontecer nos algoritmos SPN e SRT

Ideia chave: valoriza-se não só tempo esperado de processamento baixos mas também a idade do processo no sistema

RR = Response Ratio

$$RR = \frac{w + s}{s}$$

Onde

w= Tempo à espera do processador

s = Tempo de execução esperado

75

ALGORITMOS DE ESCALONAMENTO

Quando um processo se bloqueia ou termina (o algoritmo HRRN não é preemptivo) é escolhido de entre os restantes aquele que tiver **maior** Response Ratio

Vantagens

Os processos de duração curta são facilmente escolhidos (s é pequeno) pelo que não são penalizados face aos processos mais longos

Os processos longos não correm o risco de ficaram em *starvation* (w vai aumentando)

Principal dificuldade

Estimar o tempo de execução esperado (tal como nos algoritmos SPN e SRT)

76

ALGORITMOS DE ESCALONAMENTO

Algoritmo ***Feedback***

Quando não é possível estimar o tempo de processamento esperado não se podem adoptar os algoritmos SPN, SRT e HRRN

Solução

A análise da situação na máquina no passado recente fornece um bom indicador do que se irá passar nos próximos instantes

"Feedback": vai-se "realimentar" o algoritmo de escalonamento com a informação do que se tem estado a passar no sistema

Um factor tipicamente analisado é a quantidade de tempo de processamento já atribuído ao processo

➔ Como evitar prejudicar processos que, dada a sua natureza, sejam de duração longa (ex., serviços) e portanto tenham tido muito tempo de execução?

➔ Solução: Apenas se analisa o passado recente

77

ALGORITMOS DE ESCALONAMENTO

Implementação típica de algoritmos de *feedback*

- Prioridades dinâmicas
  - A prioridade vai diminuindo ao longo da idade do processo
    - Se for um processo curto- nunca fica pouco prioritário
    - Se for um processo longo - vai perdendo prioridade e não impede os mais curtos de se executarem
- Quantum variável
  - Atribui-se um *quantum* maior para recuperar um processo que esteja a ficar penalizado e vice-versa
    - Não é viável estar a modificar a frequência da interrupção associada à duração dos *quantuns*.
    - Normalmente a interrupção de relógio ocorre a uma frequência mais elevada que os *quantuns*. A duração de um *quantum* será um número inteiro de interrupções de relógio seguidas.
    - Variar a duração de um *quantum* serão então simplesmente variar um contador de interrupções

É menos provável causar *starvation* mudando a duração dos *quantuns* do que mudar as prioridades

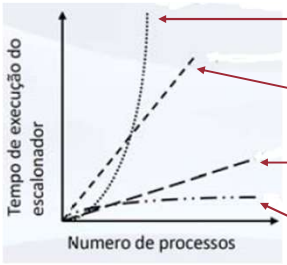
78

ALGORITMOS DE ESCALONAMENTO - ESCALABILIDADE

Escalabilidade

Uma solução em engenharia é escalável (é adaptável a qualquer dimensão do problema que vai tratar) se o custo (tempo, espaço ocupado, etc.) for proporcional à dimensão do problema.

Um algoritmo de escalonamento é escalável se o seu overhead de execução for proporcional ao número de processos em execução.



Algoritmo não escalável. Pequenos aumentos no número de processos levam a grandes aumentos (desproporcionais) no custo de execução do escalonador (mais provável de existir)

Algoritmo escalável pesado. Aumentos no número de processos levam a aumentos grandes mas proporcionais no custo de execução do escalonador (muito comum)

Algoritmo escalável leve. Aumentos no número de processos levam a aumentos pequenos e proporcionais no custo de execução do escalonador (melhor cenário que se consegue)

Algoritmo muito bom mas que não existe. Cada aumento no número de processos causa pesos de execução cada vez menores (era bom, mas não existe)

79

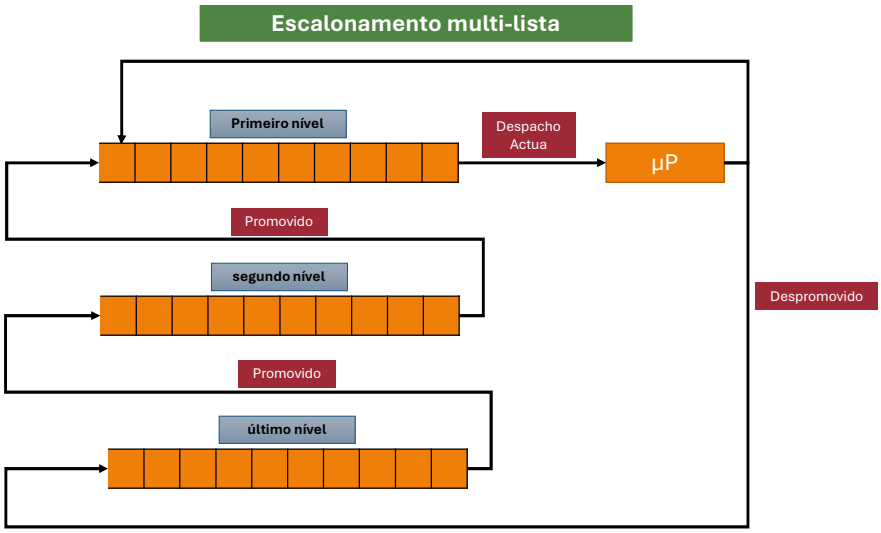
ALGORITMOS DE ESCALONAMENTO - ESCALABILIDADE

Como impedir que um algoritmo de escalonamento se torne não escalável

- Ideia: impedir que haja muitos processos em análise de cada vez.
- Método: agrupar os processos executáveis em conjuntos.
  - Apenas se analisa e põe em execução os processos de um dos conjuntos (o primeiro nível)
    - Sendo assim os processos em análise num dado instante são em menor número (reduziu-se a dimensão aparente do problema de escalonamento)
  - E os outros processos?
    - Periodicamente são promovidos para os conjuntos mais perto daquele que é analisado (em direcção ao primeiro nível)
  - E os processos que estão no conjunto do primeiro nível?
    - Periodicamente são despromovidos para o nível mais baixo
- Esta técnica tem o nome de **escalonamento multi-lista**
  - Cada conjunto de processos corresponde a uma lista de processos

80

ALGORITMOS DE ESCALONAMENTO - ESCALABILIDADE



81

ALGUMAS QUESTÕES DE REVISÃO

Explique o algoritmo X. Indique:

- Como funciona
- Quais as vantagens
- Quais as suas desvantagens
- Qual o cenário onde o algoritmo X é o mais indicado
- Quais os cenários onde o algoritmo X não é de todo indicado

➔ Substitua X por cada um dos algoritmos vistos aqui.

Construa alguns cenários com X processos de duração Y e início Z. Para cada cenário obtenha os valores das métrica W. Defina valores para Ye Z, e substitua X por cada um dos algoritmos estudados e W por cada uma das métricas estudadas.

O que é o escalonamento por níveis (multi-lista) e para que serve?

O que entende pela questão de escalabilidade no escalonamento. Indique uma forma de lidar com essa questão?

82

83

GESTÃO DO PROCESSADOR / ESCALONAMENTO

Parte 8 – Escalonamento no Unix, Linux e Windows

Lógica de funcionamento do escalonamento em alguns sistemas reais

ALGORITMOS DE ESCALONAMENTO – UNIX (GENÉRICO)

Escalonamento em Unix

Tempo partilhado baseado num esquema Round Robin com prioridades dinâmicas

- Valores menores de prioridade significa "mais prioritário"

Tenta privilegiar processos *I/O Bound* e ao mesmo tempo tenta não atrasar excessivamente a execução dos processos *CPU Bound*

Quantum típico (mas varia muito): 100ms

84

ALGORITMOS DE ESCALONAMENTO – UNIX (GENÉRICO)

Escalonamento em Unix

A prioridade varia consoante o comportamento recente do processo

- p_cpu-tempo acumulado de utilização do processador do processo
- p_pri - prioridade actual do processo

A cada 10ms a interrupção do relógio actua e faz recalcular p_cpu

A cada segundo, p_cpu é dividido em dois (forma simples de focar apenas o comportamento recente do processo)

prioridade = p_pri = prioridade base + nice + p_cpu/2

Este algoritmo é pouco escalável

85

ALGORITMOS DE ESCALONAMENTO – LINUX

Escalonamento em Linux

Tempo partilhado baseado num esquema *Round Robin* com prioridades dinâmicas

Introduz o conceito de *época* para melhorar a escalabilidade

- O algoritmo concentra a maior parte do *overhead*, não na transição de quantum para outro, mas na transição de uma época para outra
- Uma época é o conjunto de quantuns que demora a que todos os processos deixem de estar executáveis ou terem esgotado o seu *quantum*.

Nas versões mais recentes do Linux introduziu-se o escalonamento multi-lista para ser ter ainda mais escalabilidade

86

ALGORITMOS DE ESCALONAMENTO – LINUX

Escalonamento em Linux

- Em Linux um quantum é um conjunto de notificações do relógio (normalmente 20)
  - Desta forma evita-se ter que reprogramar a interrupção de relógio nas situações em que um processo deixa de estar executável antes do fim do seu quantum
- Em cada época o quantum atribuído a um processo é modificado
  - Quantum inicial = quantum base + quantum por usar na época anterior
  - (O quantum por usar pode ser, por exemplo, por o processo se ter bloqueado, "compensando" assim um processo I/O Bound na época seguinte)
  - Prioridade = prioridade base + quantum por usar actual - nice

87

ALGORITMOS DE ESCALONAMENTO – WINDOWS NT

Escalonamento em Windows (NT)

Unidade de execução: thread

Tempo partilhado baseado num esquema Round Robin com prioridades fixas e dinâmicas e escalonamento em multi-lista

Um quantum é um conjunto de notificações do relógio

Edição desktop 2 notificações ➔ 15ms

Edição server: 12 notificações ➔ 90ms

(pode mudar de versão para versão)

O sistema permite aumentar a dimensão do quantum às tarefas de um processo cuja janela esteja activa (= "esteja em execução foreground")

➔ significa mais notificações de relógio por quantum

88

ALGORITMOS DE ESCALONAMENTO – WINDOWS NT

Escalonamento em Windows (NT)

- Existem 32 níveis de prioridades fixas. As threads (processos) são colocadas em níveis pelo utilizador ou pelo sistema de acordo com a sua importância
- As prioridades dinâmicas são usadas da forma habitual (para compensar threads I/O Bound, por exemplo) mas sempre tendo como base as prioridades base (fixa) da thread
- As prioridades são ajustadas
  - No fim de uma operação I/O
  - No fim de uma espera de sincronização
  - No fim de um quantum

  Se uma thread se interromper voluntariamente, a sua prioridade será aumentada de acordo com uma tabela (disco +1, rede +2, etc.).

  Se uma thread usar o seu quantum na totalidade, a sua prioridade converge novamente para a sua prioridade base

89

ALGUMAS QUESTÕES DE REVISÃO

Explique como funciona o escalonamento no Unix (genericamente), no Linux, no Windows NT. Relacione com os conceitos dados nesta matéria.

Explique de que forma o Windows NT evita situações de starvation.

Explique de que forma o Linux evita situações de starvation.

Poder-se-á dizer que os algoritmos de escalonamento do Linux e do Windows NT são 100% justos? E isso é bom ou é mau? Explique.

Explique como são usadas as prioridades no Windows NT.

Os quantuns no Linux têm sempre a mesma duração? Porquê?

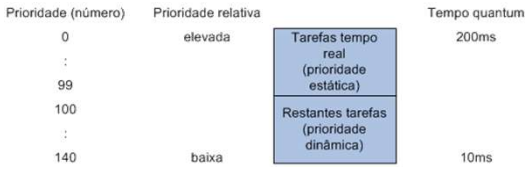
No contexto de escalonamento, indique uma diferença entre a edição desktop e a edição server do Windows NT. Explique essa diferença.

90

ALGORITMO DE ESCALONAMENTO - LINUX

Kernel 2.5

- O escalonamento é preemptivo e baseado em prioridades organizadas em duas classes de valores: [0..99] para tarefas de tempo real e [100..140] para as restantes tarefas. Um valor numérico baixo indica prioridade relativa elevada.
- Ao contrário dos restantes SO, o Linux atribui quantums elevados para tarefas de elevada prioridade.

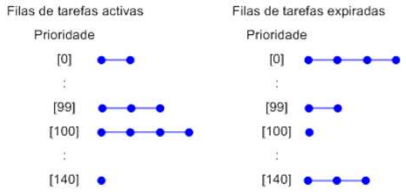


- Quando uma tarefa comuta para o estado pronto a executar é colocada numa fila de acordo com a sua prioridade. Um processo é considerado elegível para ser escalonado enquanto o seu quantum não tiver esgotado. Quando o quantum tiver esgotado, é considerado expirado e não é novamente escalonado enquanto as restantes tarefas não tiverem esgotado o seu quantum.
- Para cada prioridade são mantidas duas filas: a fila de tarefas activas (tarefas que não esgotaram o seu quantum) e a fila de tarefas expiradas (tarefas que já esgotaram o seu quantum).
- O escalonador escolhe a tarefa mais prioritária de entre as filas de tarefas activas. Quando não existirem tarefas na fila de tarefas activas, estas são trocadas.

ALGORITMO DE ESCALONAMENTO - LINUX

Kernel 2.5

- Tarefas activas e expiradas



- Para as tarefas de tempo real, a prioridade é estática e atribuída na fase de inicialização da tarefa.
- Para as restantes tarefas a prioridade é dinâmica. No fim de uma tarefa esgotar o seu quantum, aactual prioridade é incrementada ou decrementada em função do grau de interactividade.
- O grau de interactividade é determinado pelo tempo que a tarefa esteve bloqueada à espera deuma operação de E/S.
- As tarefas mais interactivas (I/O bound), tipicamente, apresentam tempos de espera elevados e,por isso, a sua prioridade é decrementada. O resultado é o aumento da prioridade relativa dastarefas interactivas.
- Às tarefas que fazem uso intensivo do processador (CPU bound)(menores tempos de espera deE/S), é incrementada a prioridade, diminuindo a prioridade relativa.