Substituição de Páginas

Marcelo Johann

Lembrando: paginação sob demanda

- O Sis. Op. aloca as páginas na RAM à medida que os processos as pedem.
  - O resto do tempo, elas são copiadas no disco (*paging*).
- Uma tentativa de acesso a uma página que não está na RAM provoca uma falta de página
  - Interrupção de software
  - Tratamento muito lento
- Para minimizar o ônus:
  - O programador deve usar o princípio de localidade e agrupar os acessos lógicos dentro de uma página.
  - O Sis. Op. deve tentar manter na RAM o conjunto de páginas mais acessas pelos processos.

Plano da aula

- Políticas de alocação de quadros
  - Quantos quadros por processo?
- Políticas de substituição de páginas.
  - Global vs. Local
  - Exemplos de algoritmos globais
  - Exemplos de algoritmos locais
- Trashing

Quais são os problemas com demanda?

- O sis. Op. deve tratar de dois problemas:
  - Alocação de páginas/quadro:
    - Quantos quadros alocar a um processo?
  - Substituição de páginas:
    - Quando não há espaço na RAM, quais páginas devem ser descartadas para possibilitar o *page-in* de outras?
    - Determinar uma página vítima
    - Otimização pelo uso de alguns bits
      - Bit de sujeira ; direitos de acesso (RO)

Alocação de quadros

- Quantos quadros alocar a um processo?
  - Existe um número mínimo!
    - Depende da arquitetura e de suas instruções!
    - Exemplo 1: OPCODE @1 @2 => 3 páginas
    - Exemplo 2: OPCODE reg, @1 => 2 páginas
    - Abaixo deste número mínimo, o processo pode nunca executar (por falta de página)
  - Existe um número máximo!
    - Tamanho da memória real.
  - Existe um meio-termo...
    - Quanto mais quadros alocados, menos falta de página.
  - Dois métodos
    - Igualitária vs. proporcional.

Alocação igualitária

- A cada decisão de escalonamento, o sistema determina o número *n* de processos prontos.
- O Sis. Op. mantém alocados *m*/n quadros, onde *m* é o número de quadros.
- É um mecanismo **igualitário** pois todos os processos recebem o mesmo número de quadros.
- O número de quadros por processo é ajustado conforme for o grau de multiprogramação.
- Problema: todos ganham o mesmo número de quadros, independente de seu uso real da memória!

Alocação proporcional

- Estima-se a memória virtual S_i usada pelo processo i;
- Aloca-se (S_i / Σ S_i) x *m* quadros ao processo i.
  - O número de frames alocado é proporcional ao uso de memória.
- S_i deve ser atualizado ao decorrer do tempo!
- Pode-se ponderar esse esquema através do uso de prioridades
  - Processos com maior prioridade recebem mais frames.

Categorias de políticas de substituição de páginas (*frames*)

- Onde/como escolher as páginas vítimas?
- Basicamente, 2 categorias de algoritmos:
  - **Globais**
    - Todas as páginas são consideradas como um *pool* único.
    - Qualquer página, de qualquer processo, pode ser escolhida.
  - **Locais**
    - Para cada processo, se mantém um conjunto de *frames* em uso ("*working set*")
    - Para efetuar um *page-in*, a página substituída será escolhida dentro do working set do processo que pediu o *page-in*.

Pros e contras

- As políticas globais:
  - Acarretam no "roubo" de páginas de processos menos prioritários, pelos processos mais "gulosos".
  - Assim, o comportamento de alguns processos se repercute no andamento de outros. :o(
- As políticas locais:
  - Isolam os processos :o)
  - Provocam desperdiço :o(

Catálogo de algoritmos de substituição

- Política global
  - Algoritmo ótimo
  - FIFO
  - LRU - *Least Recently Used*
    - Segunda chance
- Política local
  - Algoritmo ótimo
  - Working Set
  - Page Fault Frequency Replacement (PFF)

Como modelar e avaliar os algoritmos de substituição?

- Obter rastros de execução dos programas para analisar seu padrão de acessos na memória.
- O padrão pode ser resumido à cadéia de referências:
  - Sequência dos números das páginas acessadas.
  - Obtida a partir dos endereços lógicos acessados pelo processo.
  - Desde que se quer avaliar as faltas de página, apenas se rastreiam os números de páginas que mudam.
  - Exemplo: 0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103... => 1, 4, 1, 6, 1, 6, 1...
- A avaliação foca:
  - O número de falta de páginas
  - ... Comparado com o número de quadros.

Algoritmo FIFO

- A página mais antigamente carregada na memória é descartada.
- Exemplo: 3 quadros, cadeia

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

7	7	7	2		2	2	4	4	4	0		0	0		7	7	7		
	0	0	0		3	3	3	2	2	2		1	1		1	0	0		
		1	1		1	0	0	0	3	3		3	2		2	2	1		

15 faltas de página

- Anomalia de Belady: com 3 e 4 quadros, a cadeia:
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Anomalia de Belady

Implementação do FIFO

- Não é preciso guardar a data de carga da página na memória...
- Basta saber qual foi a ordem de carga.
  - Uma fila pode ser usada para rastrear as páginas mais antigas.

Algoritmo ótimo

- Pode-se comprovar que o algoritmo que menos provoca faltas de páginas é o que escolhe:
  Página vítima = página mais remotamente (no futuro) acessada.
- Por exemplo, a cadeia de teste do FIFO provoca 9 faltas de página só.
- Infelizmente... É preciso conhecer o futuro para implementar este algoritmo!
  - Vide mesma situação com SJF!
- Porém, útil para ter uma meta de qualidade!

Algoritmo LRU

- Uma aproximação do ótimo é usar o passado para prever o futuro.
- LRU = "ótimo invertido": **a página mais remotamente acessada no passado será a vítima.**
- A cadeia de teste do FIFO provoca 12 faltas de página só.
- **Diferente de FIFO**!
  - A mais antigamente carregada pode ter sido acessada recemente!
  - Não sofre da anomalia de Belady.
- É preciso guardar alguma informação a respeito da data do último acesso a cada página.

Implementação do LRU

- Para rastrear a ordem dos últimos acessos, é preciso acrescentar informações às páginas.
- Duas soluções:
  - Uso de **registradores** (em SW ou HW) para armazenar um "time-stamp".
    - É preciso procurar na tabela de páginas o time-stamp para obter a página mais antiga...
    - É preciso salvar os registradores nas trocas de contexto...
  - Uso de uma **pilha**
    - As páginas são ordenadas pela ordem de uso.
    - A mais recente está sempre no topo.
    - É preciso de uma lista duplamente encadeada.
    - Facilita a busca!

LRU aproximado

- Nem sempre o HW possibilita o uso do LRU total.
- Pelo menos, o HW sempre possibilita o uso de **UM** bit: o **bit de referência**.
  - Quando houve um acesso, o bit é setado.
- Pode-se usar este bit como ponto de partida para atualizar a ordem de acesso às páginas.
  - O Sis. Op. mantém uma **lista de Bytes**, sendo um byte associado a cada página.
  - Cópia regular, pelo Sis. Op, do valor deste bit para cada página, no bit de maior peso na entrada da lista.
  - Shift para a direita dos demais bits do Byte (descarta o bit de peso menor).
  - A página com menor byte é a mais antiga!

Algoritmo da segunda chance

- Caso particular da lista de Bytes: usa-se apenas o bit de referência.
- Aplica-se primeiro um FIFO para achar uma página vítima.
- Se seu bit de referência vale 0, a página é *paged-out*
- Se o bit vale 1, a página foi "recentemente" usada, e lhe é dada uma **segunda chance**: procura-se uma outra página vítima.
  - Em compensação, **seu bit de referência é zerado**.
- Na pior dos hipóteses, todas as outras páginas também ganham uma 2a chance, e a página vítima acaba sendo escolhida novamente...

Catálogo de algoritmos de substituição

- Política global
  - Algoritmo ótimo
  - FIFO
  - LRU - *Least Recently Used*
    - Segunda chance
- Política local
  - Algoritmo ótimo
  - Working Set
  - Page Fault Frequency Replacement (PFF)

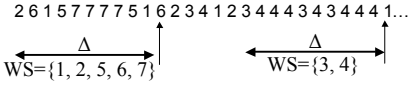
Algoritmo ótimo

- Prever os Δ próximos acessos a páginas de um dado processo.
- Mantê-las na RAM.
- Problema: necessita de antecipar o futuro.

Working Set

- Baseado na localidade:
  - Define-se uma janela de análise de duração Δ
  - As páginas que foram acessadas nas Δ últimas referências formam o *working set*.
  - Exemplo: Δ=10
    - Seqüência de acessos:

2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1...



Implementação do WS

- Para manter o histórico, pode-se usar os mecanismos usados para o algoritmo LRU:
  - Usa-se um registrador de n bits (por exemplo um byte) por quadro;
  - Regularmente, o bit de residência dos quadros é copiado (com deslocamento) para os registradores;
  - Cada quadro que tem pelo menos 1 bit aceso no registrador pertence ao WS
    - Além disso, mantém-se uma contabilidade dos quadros mais acessados.

Dificuldade com o *Working Set*

- Problema: como definir Δ?
  - Se for pequeno, ele não dará conta das páginas acessadas...
  - Se for muito grande, o WS não será útil
- Trata-se de um algoritmo local, porém:

  $D = \Sigma WS(i)$

  Representa a demanda total por quadro ativamente usados.

Exemplo de WS: Windows

- O Windows usa um meio-termo entre uma política global e uma política local:
  - Usa um Working Set para gerenciar quadros / processo;
  - Quadros vítimas podem vir de qualquer processo.
- Working-Set
  - Na criação, default entre 50 e 345 quadros;
  - Quando há Falta de Página, aloca mais páginas ou substitui uma.
    - Procura vítimas nos processos que não usaram todo seu WS há tempo; nos "grandes processos" que não executaram.
- (Obs: no Linux, usa-se o LRU/2a chance)

Quando a demanda aumenta...

- Considere um processo P que não tem o número suficiente (< WS) de quadros alocados:
  - Vai provocar uma falta de página rapidamente;
  - Para carregar a página faltando, vai ser descartado um quadro de seu WS;
  - Por definição, este quadro estava sendo usado frequentemente...
  - Logo, o processo vai rapidamente efetuar uma nova falta de página!
- Um processo que vive fazendo faltas de página é dito em "***thrashing***".

Thrashing e efeito bola de neve

- Um processo em *thrashing* já é preocupante... Mas pode ficar pior!
  - Se algum mecanismo global é usado, ele pode levar ao "roubo" de quadros de outros processos, os quais podem entrar também em *thrashing*...
  - Considere o escalonador de CPU:
    - As faltas de página bloqueiam os processos;
    - A fila de processos prontos se esvazia;
    - Logo, o escalonador pode **liberar** novos processos para usar a CPU.
    - Os mesmos vão precisar de quadros... Que não se encontram disponíveis!
    - O *thrashing* vai piorando em todo o sistema.
- Todos os processos acabam bloqueados em espera de *paging*.

Ilustração do *thrashing*

- Uso da CPU vs. Número de processos no sistema

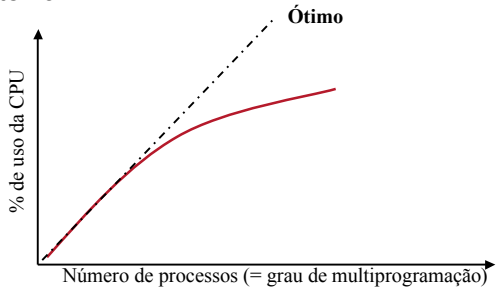
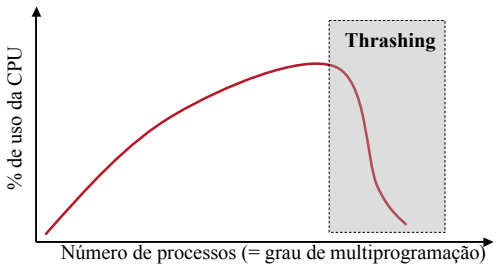


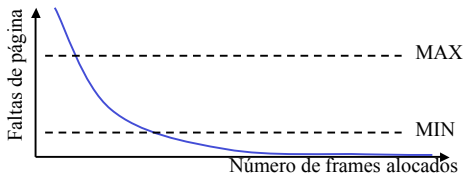
Ilustração do *thrashing*

- Uso da CPU vs. Número de processos no sistema



Page-Fault Frequency (PFF): tentando prevenir o *thrashing*

- Idéia: usar o **número de faltas de páginas** como indicador da necessidade de mais ou menos *frames* / processo.
- Alteração dinâmica do Δ:
  - Quando um processo provoca um aumento no número de FP, aloca-se mais um frame a ele.
  - Quando um processo efetua pouquíssimas FP, libera-se um frame de seu WS.



Fazer o que quando não tem mais quadros?

- Se realmente não se tem mais quadro para um dado processo...
  - Ele efetua muitas FP...
  - O algoritmo é local ou global, mas não tem mais quadro disponíveis
- Única solução:
  - Efetuar o ***swap-out*** de todo o processo!
  - Recupera-se seus quadros para dar um alívio à memória e aos outros processos.
- Qual processo vítima?
  - O que provoca FP;
  - Um outro, que usa muita memória?
  - Um outro, que usa pouca memória?
  - O último que executou?

Sobre o tamanho da página/frame

- Argumentos a favor de uma **página pequena**:
  - Diminui a fragmentação interna;
  - Diminui o tempo de escrita/leitura no disco (paging);
- Argumentos a favor de uma **página grande**:
  - Diminui o número de entradas na tabela de páginas;
  - Diminui o número de faltas de páginas;
  - Melhora a taxa de acerto na TLB;

Próxima aula…

Gerência de entrada/saída!