Sistemas Operacionais

Memória virtual (cap. 10)

Sumário

- Princípios de paginação sob demanda
- Impacto da paginação sob demanda na criação de processos
- Algoritmos de substituição de páginas
- Alocação de quadros
- *Thrashing*

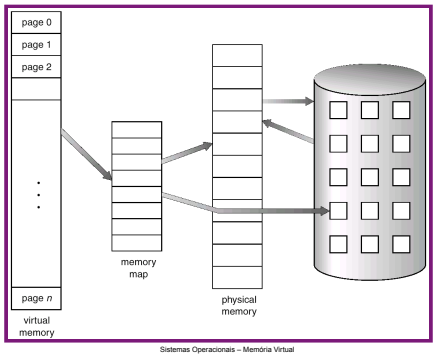
Princípios gerais

- Memória segmentada/paginada já fizeram a separação entre endereço lógico e físico
- Estendendo a funcionalidade da tabela de páginas, nem todas as páginas precisam estar na memória física o tempo todo
  - Apenas a parte do programa em execução
- Espaço de endereços lógicos pode ser muito maior que o físico

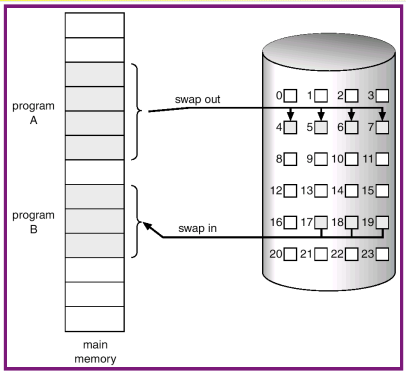
Paginação sob demanda

- Trazer as páginas para a memória física apenas quando necessárias
  - Menos E/S
  - Menor área de memória
  - Resposta mais rápida
  - Mais processos/usuários
- Bit válido/inválido usado para controlar presença

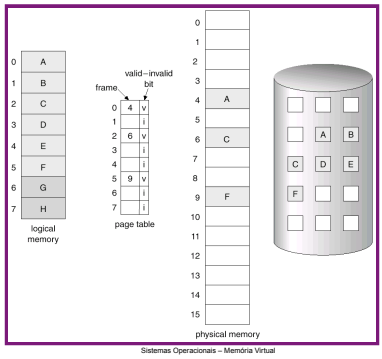
Paginação sob demanda



Extensão do princípio de *swap*



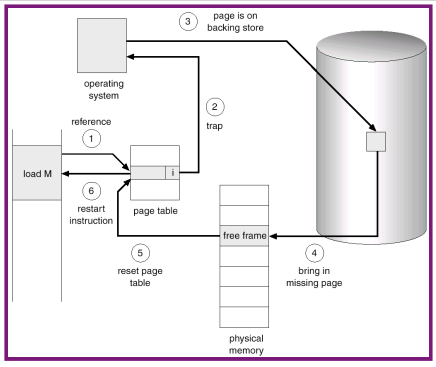
Paginação sob demanda



Falha de página (*page fault*)

- Na primeira referência a uma página: TRAP
- S.O. consulta tabela de paginação virtual
  - Referência inválida → aborta o processo
  - Pág. ausente → localiza a página no disco
- Obtém um quadro vazio
- Carrega a página no quadro
- Atualiza a tabela de páginas
- Reinicia a instrução que gerou a página

Falha de página (*page fault*)



Paginação sob demanda: desempenho

- Taxa de falta de páginas: $0 \leq p \leq 1.0$
- Tempo de acesso efetivo (TAE)

$$TAE = (1 - p) \times T_{acesso\_à\_mem} + p \times T_{falha}$$

$$T_{falha} = (\textit{page fault overhead} + \textit{swap page in} + \textit{restart overhead})$$

Paginação sob demanda: desempenho – exemplo

- Tempo de acesso à mem.: 100 ns
- Tempo de *swap*: 25 ms
- TAE = (1 – p) x 100 + p x 25.000.000
- TAE = 100 + 24.999.900 x p ns

p	TAE (ns)	TAE/Ta
0,100000	250090	2500,90
0,010000	25099	250,99
0,001000	2600	26,00
0,000100	350	3,50
0,000010	125	1,25
0,000001	102	1,02

Benefícios da memória virtual para a gerência de processos

- Redução do custo de criação de processos
  - fork() → *copy-on-write*
- Acesso a arquivos mapeados em memória
  - mmap() → mem. virtual associa mem. e arquivo

Copy-on-Write (COW)

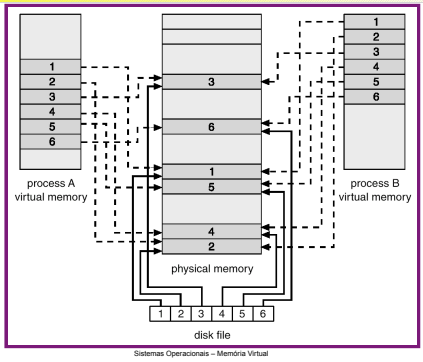
- Duplicação de página alterável sob demanda
- Processos recebem referência à mesma página
  - Página (alterável) tem permissão de escrita = 0
- Se algum dos processos tenta escrever
  - *Page fault* → nova página/quadro é alocada(o)
  - A partir daí, cada processo tem sua cópia

Podem ignorar a discussão sobre vfork()

Arquivos mapeados em memória

- Mem. virtual: mapeamento do endereço lógico do processo a uma área em disco
  - Normalmente, o sentido é mem. → disco
- Com mmap(), pode-se fazer o oposto:
  - Conteúdo do disco (arquivo) → espaço de mem.
- Exige a capacidade de usar outras áreas além do espaço de *swap* usual
- Simplifica o acesso a arquivos de dados
  - Podem ser tratados como vetores na memória
- Permite que processos compartilhem arquivos

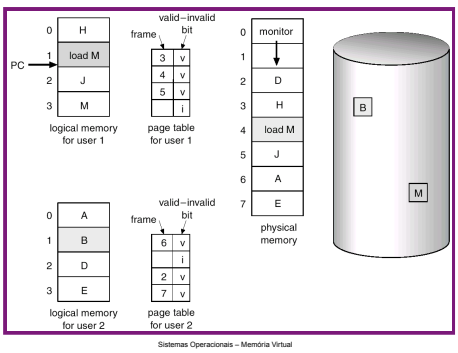
Arquivos mapeados em memória



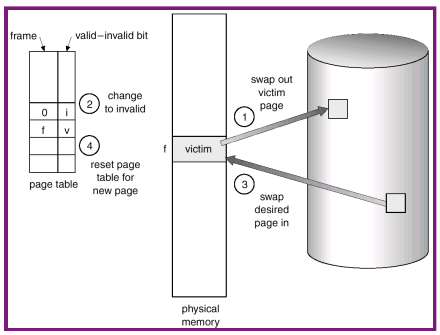
E se não houver um quadro vazio?

- Substituição/reposição de páginas
  - Encontre uma página que não esteja em uso
  - Libere o quadro (pode exigir escrita no disco)
- Desempenho:
  - Página retirada pode vir a ser acessada de novo
  - Deseja-se um algoritmo que minimize as falhas

Reposição de páginas



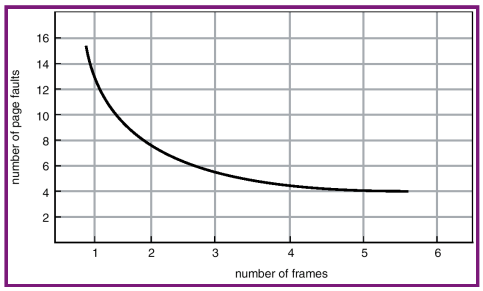
Reposição de páginas



Reposição de páginas

- Essencial para completar o desacoplamento entre memória física e lógica
- Tabela de páginas inclui bits extras para auxiliar no processo de escolha de candidatos
  - *dirty bit*: página foi modificada em relação ao disco
  - *access bit*: página foi acessada “recentemente”

Relação entre falhas e num. de quadros disponíveis



Algoritmos de reposição de páginas

- Deseja-se a menor taxa de falhas possível
- Avaliação é feita contra uma sequência de referências à memória que seja característica
  - Computa-se o num. de falhas para cada algor.
- Nos exemplos a seguir, a sequência é sempre

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

First-In-First-Out (FIFO) Algorithm

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- 3 quadros: 9 *page faults*

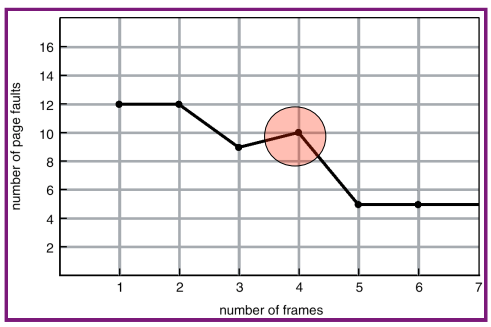
1	1	4	5	
2	2	1	3	
3	3	2	4	

Anomalia de Belady: Mais quadros não deveriam gerar mais *page faults*!

- 4 quadros: 10 *page faults*

1	1	5	4	
2	2	1	5	
3	3	2		
4	4	3		

Anomalia de Belady



Algoritmo ótimo (presciente)

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- Substitua a página que levará mais tempo para ser acessada novamente
- Exemplo: 4 quadros

1	4	
2		
3		6 *page faults*
4	5	

Least Recently Used (LRU)

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- Substitui página há mais tempo sem referências
- Exemplo: 4 quadros

1	5		
2			
3	5	4	
4	3		

8 *page faults*

- Implementação: contagem de tempo ou pilha

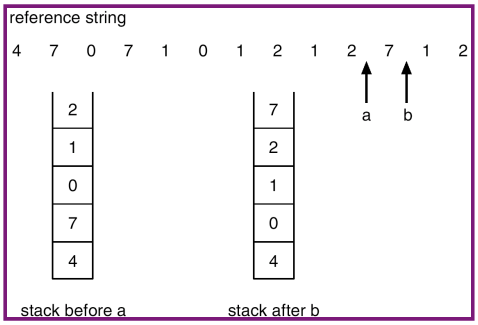
Least Recently Used (LRU)

- Contagem de tempo
  - Cada página tem registro de tempo de acesso
  - A cada acesso, relógio é copiado para o registro da página acessada
  - Quando se executa o LRU, busca-se a página com registro mais antigo (menor)

Least Recently Used (LRU)

- Implementação com pilha
  - Mantém-se uma pilha de acessos (dupl. encad.)
  - A cada acesso, página referenciada vai p/ o topo
  - Requer mudanças em 6 apontadores
  - Não requer busca no momento da substituição

LRU: uso da pilha



LRU: aproximações

- Processamento a cada acesso pode ser excessivamente oneroso
- Aproximações reduzem o processamento a eventos periódicos
- Utiliza bit de referência oferecido pela MMU para cada página

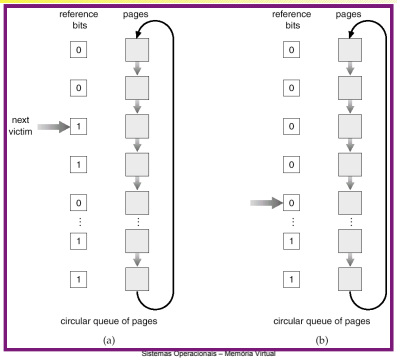
LRU: aproximações

- Bit de referência
  - Inicialmente, bit de referência = 0
  - Bit é alterado pelo hardware a cada acesso
  - Substitui-se alguma página com bit nulo
    - Não há noção de ordem entre as pág. referenciadas

LRU: aproximações

- Segunda chance
  - ° Usa lista circular (FIFO) e bit de referência
  - ° Se página a ser substituída (ordem de relógio) tem bit de referência setado,
    - faz bit de referência igual a zero,
    - deixa a pág. na memória, atualiza seu tempo de acesso
    - substitui a próxima na ordem que não foi referenciada
  - ° Ao invés de manter "tempo de acesso", basta percorrer a lista de forma circular

Algoritmo de segunda chance



Algoritmos de contagem

- Opções já propostas ao LRU
- Manter um contador do num. de referências feitas a cada página
- MFU (*most frequently used*)
  - ° Quem foi pouco usado ainda deve ser mais usado
- LFU (*least frequently used*)
  - ° Privilegia págs. com muitos acessos
  - ° Exige "envelhecimento" (decaimento exponencial)

Alocação de quadros para cada processo

- Cada processo precisa de um mínimo de págs.
- Dois esquemas principais:
  - ° alocação fixa
  - ° alocação por prioridade

Alocação fixa

- Igual: todos os processos recebem o mesmo no. de quadros na memória física
- Proporcional: função do tamanho do processo

s_i = size of process p_i
$S = \sum s_i$
m = total number of frames
a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 64$
$s_i = 10$
$s_2 = 127$
$a_1 = \frac{10}{137} \times 64 \approx 5$
$a_2 = \frac{127}{137} \times 64 \approx 59$

Substituição global x local

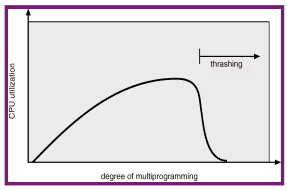
- Global:
  - ° Página a ser substituída escolhida do total
  - ° Quadros pode ser transferidos entre processos
- Local
  - ° Página tem que ser do próprio processo que gerou a falha
  - ° No. de quadros de cada processo não muda

Thrashing (atividade improdutiva)

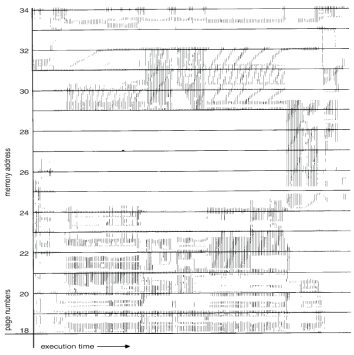
- Se processos não têm páginas "suficientes"
  - *Page faults* aumentam
  - Utilização de CPU diminui
  - S.O. pensa que é preciso aumentar multiprog.
  - Outro processo é trazido para a memória
  - Demanda por quadros aumenta
- *Thrashing* → processos estão ocupados apenas fazendo *swap* de páginas

Localidade de referência e *Thrashing*

- Por que mem. virtual funciona? → localidade
  - Processo migram entre áreas de acesso
  - Áreas podem se sobrepor ao longo do tempo
- Por que *thrashing* ocorre?
  ∑ localidades > memória total



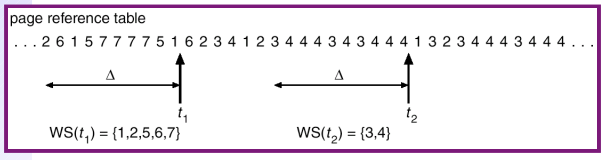
Localidade em padrão de acesso



Modelo de conjunto ativo (*working-set*)

- janela ativa (*working-set window)*
  um num. de referências a páginas (instr.)
- WSSi (*working set* do processo Pi) = total de páginas referenciadas na mais recente (varia com o tempo)
  - Se muito pequeno: não cobre a localidade
  - Se muito grande: cobre várias localidades
  - Se = cobre todo o programa

Modelo de conjunto de trabalho (*working-set*)



Modelo de conjunto de trabalho (*working-set*)

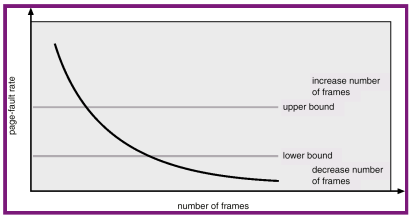
- D = WSSi demanda total por quadros
- D > m *Thrashing*
  - Nesse caso, suspende um dos processos

Identificação do *working set*

- Aproximado com temporizador e bit de referência
- Exemplo: = 10.000 unid. tempo
  - Temporização acionada a cada 5.000 unid. tempo
  - Mantém dois bits por página
  - A cada temporização, copia e zera bits de ref.
  - Se um dos bits == 1 pág. no *working set*
- Melhoria: 10 bits e interrupção a cada 1.000 unid.

Frequência de falhas de página

- Opção ao algoritmo de *working set*
- Determina-se taxa de falhas "aceitável"
  - Se taxa real é baixa, processo perde quadros
  - Se taxa real é alta, processo ganha quadros



Outras considerações

- Pré-paginação (*prefetching*)
- Determinação do tamanho da página
  - fragmentação interna (págs. menores)
  - tamanho das tabelas de página (págs. maiores)
  - custo de E/S (*seek* x transferência)
  - localidade (págs. pequenas melhoram a resolução, mas geram mais faltas)

Outras considerações

- Alcance do TLB: memória acessível por ele
  - TLB Reach = (TLB Size) X (Page Size)
- Idealmente, *working set* deve caber na TLB
  - Caso contrário, aumentam as falhas

Impacto do padrão de acesso

```
int A[][] = new int[1024][1024];
// linha = uma pág.; memória < 1024 quadros
for (i = 0; i < A.length; i++)
    for (j = 0; j < A.length; j++)
        A[i,j] = 0;
```

1024 *page faults*

```
for (j = 0; j < A.length; j++)
    for (i = 0; i < A.length; i++)
        A[i,j] = 0;
```

1024 x 1024 *page faults*

Outras considerações

- Inter-relação com E/S
  - Páginas precisam ser "presas" na memória física p/ permitir DMA