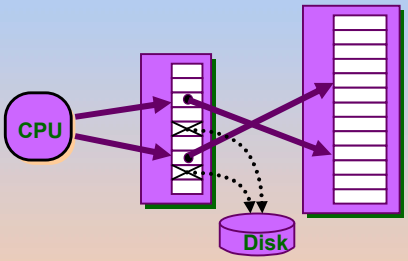
Capítulo 9 Memória Virtual

- Introdução
- Soluções Históricas
  - Overlays
  - Swapping
- Memoria Virtual
  - Demand Paging
  - Page Replacement
    - Algoritmos
- Outros Assuntos
- OS Examples



1

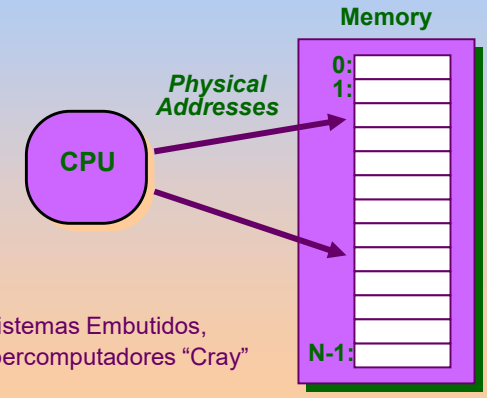
Como vencer a capacidade limitada das memórias?

- ***Como é possível executar um ou mais programas com tamanho total superior à capacidade da memória?***:
  - **Overlays** (já não se usa!)
    - mecanismo de linguagem de programação (ex.: TurboPascal)
  - **Swapping**
    - mecanismo embutido no sistema operativo.
    - Usar memoria secundária como uma memoria principal
    - mecanismo de suporte à gestão de memória virtual.
  - **Memoria Virtual**
    - Extensão ao conceito logico de memoria

2

Sistemas apenas com Memória Física

Endereços gerados pelo CPU têm uma correspondência directa (quase) aos endereços da memoria física.



- Exemplos:
  - PCs Antigos
  - A maior parte dos Sistemas Embutidos,
  - Quase todos os Supercomputadores “Cray” etc.

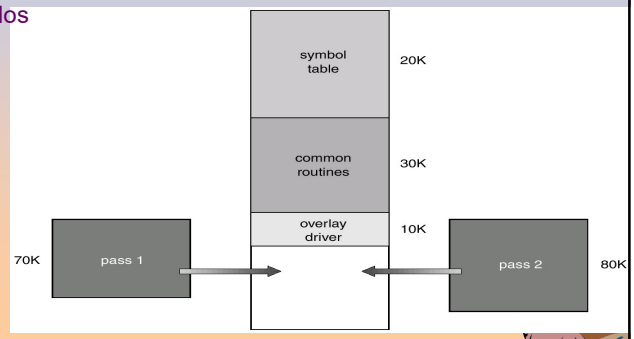
3

Overlays

- No passado, overlays foram usados
  - Necessário quando um processo era maior que a quantidade de memória que lhe foi reservada.
  - Mantém-se em memória somente aquelas instruções e dados que são necessários numa dada altura.Programa dividido em secções logicamente distintas – chamadas overlays
  - Assim mais programas podiam correr do que cabiam na memoria física se totalmente carregados

Desvantagens

- Programador tinha a responsabilidade de dividir o seu programa. O SO (pode) não oferece qualquer suporte
- Divisão dum programa é difícil e dispendioso em tempo. Uma tarefa (sem interesse) que podia resultar em muitos erros

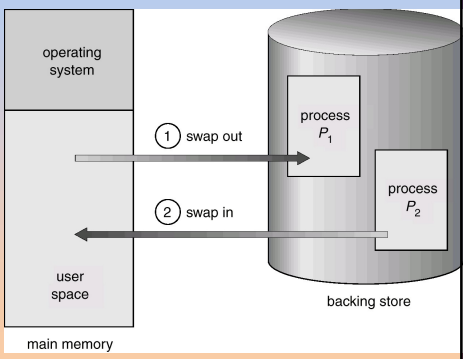


4

Swapping

Um processo pode ser temporariamente expulso (*swapped out*) da memória para um *backing store,* e depois ser re-admitido (*swapped in*) na memória para poder continuar a sua execução.

- ***Backing store*** – disco rápido capaz de alojar cópias das imagens de memória dos processos dos utilizadores; tem de fornecer acesso directo a estas imagens de memória.
- A maior parte do tempo de swapping é tempo de transferência
  - =Latência+Taxa de Transferencia
- Versões de swapping existem em sistemas UNIX, Linux, Windows, etc.



5

Memória Virtual

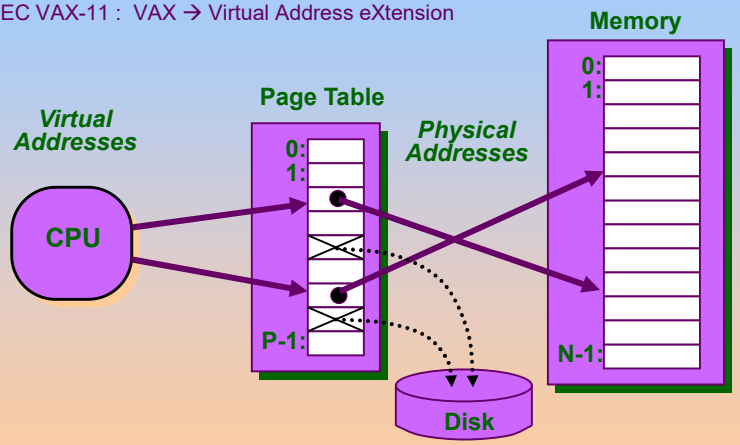
Separação de memória lógica da memória física

- O espaço de endereçamento lógico dum processo **pode ser maior do que a memória física**
- A soma dos espaços de endereçamento de todos os processos pode ultrapassar a memória física.
- Apenas a parte "ativa" (working set) do programa precisa de estar em memória – mais eficaz!
- Fornece mecanismos de simplificação de gestão de memória e proteção
- Implementações
  - Demand paging
  - (Demand Segmentation)

6

Um Sistema com Memória Virtual

- Examplos: workstations, servers, modern PCs, etc.
- DEC VAX-11 : VAX → Virtual Address eXtension



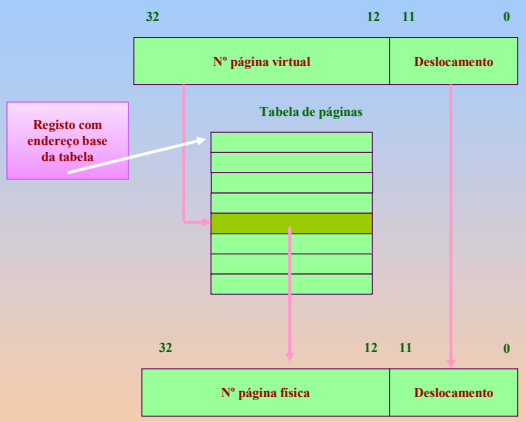
- **Tradução de Endereços: Hardware/Software traduz os endereços virtuais para endereços físicos via uma estrutura de dados (tabela) gerida pelo SO. Mem. Virtual : P>N**

7

Tamanho da tabela de páginas

- Se
  - o espaço virtual tem 32 bits (4 GB)
  - a página tem 12 bits (4KB)
- Então
  - a tabela de páginas tem 20bits de entradas de 32 bits.

  Ou seja, gasta **4 Mbytes**!
- Se a coluna de valida/invalida (ver a seguir) estiver incluída ocupa ainda mais !

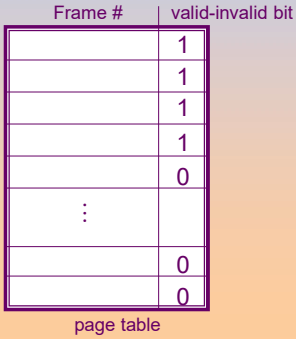


8

Implementação : Bit Válido-Inválido

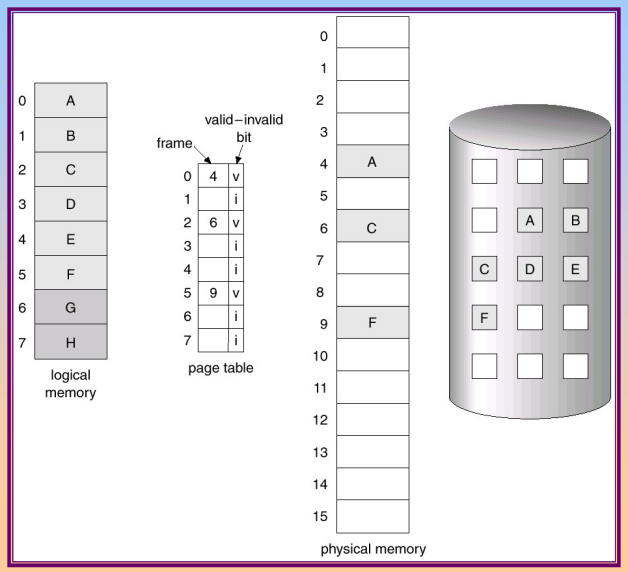
- Na tabela de páginas cada página além de indicar *um frame* terá associado um bit válido–inválido
  (1=válido ⇒ dentro da memória, 0=inválido ⇒ fora da memória)
- Inicialmente o bit válido–invalido é zero para todos as entradas.
- Durante a tradução do endereço,
  - Se o bit válido–inválido for 0 ⇒ “”page fault””.

Exemplo duma tabela de páginas



9

Tabela de Paginas quando algumas páginas não se encontram na memória principal

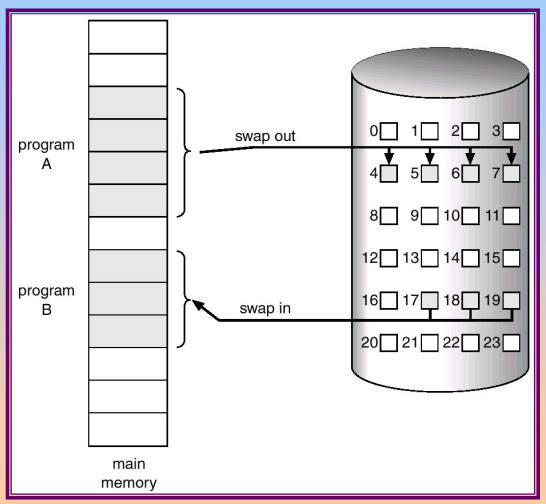


10

Demand Paging

- Trazer uma página para memória apenas quando for necessário
  - menor I/O necessário
  - Menos memória necessária
  - Tempo de resposta mais rápido
  - Mais utilizadores

•As zonas de memória virtual não carregadas em memória principal e com dados/código dos processos **estão num Backing store** (m disco num *swap file ou partition*)

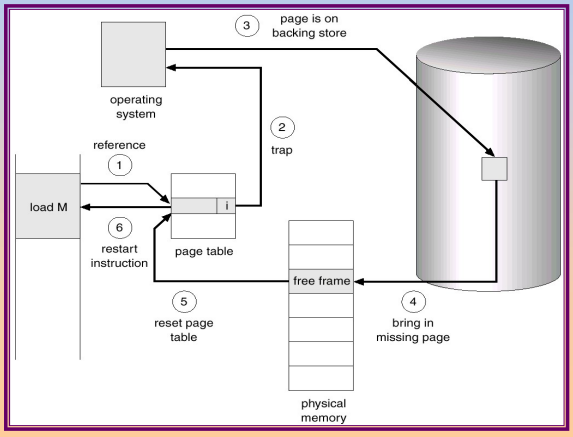


Transfer of a Paged Memory to Contiguous Disk Space

11

Page Fault

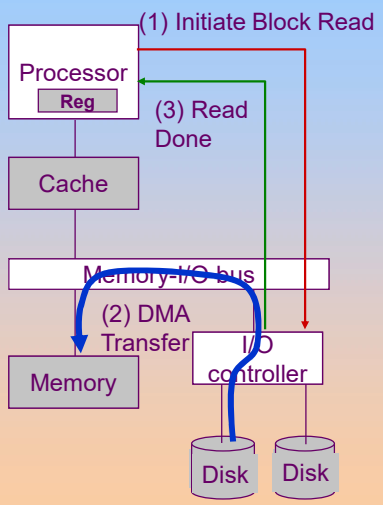
- Instrução na CPU utilize um endereço lógico traduzido pelo MMU (1) para uma página que não está na memoria → TRAP OS (2)
  - (3) Obter localização da moldura. (4) Swap (Inserir/trocar) página na moldura (5) Reset tables, validation bit = 1. (6) Re-começo da instrução:



12

Serviço duma Page Fault - Hardware

- Processor Signals Controller
  - Read block of length P starting at disk address X and store starting at memory address Y
- Read Occurs
  - Direct Memory Access (DMA)
  - Under control of I/O controller
- I / O Controller Signals Completion
  - Interrupt processor
  - OS resumes suspended process



13

Desempenho do Demand Paging

- Probabilidade duma falha de pagina $0 \le p \le 1.0$
  - se $p = 0$ sem falha
  - se $p = 1$, cada referencia é uma falha
- Effective Access Time (EAT)
  EAT = $(1 - p)$ x memory access + p x page fault overhead
- Page Fault overhead = service page fault interrupt
  + swap page out (ver depois)
  + swap page in
  + restart overhead

Swap Page In/Out → Disk Latency and Transfer Time !

14

Problemas: O que acontece se não há uma moldura disponível?

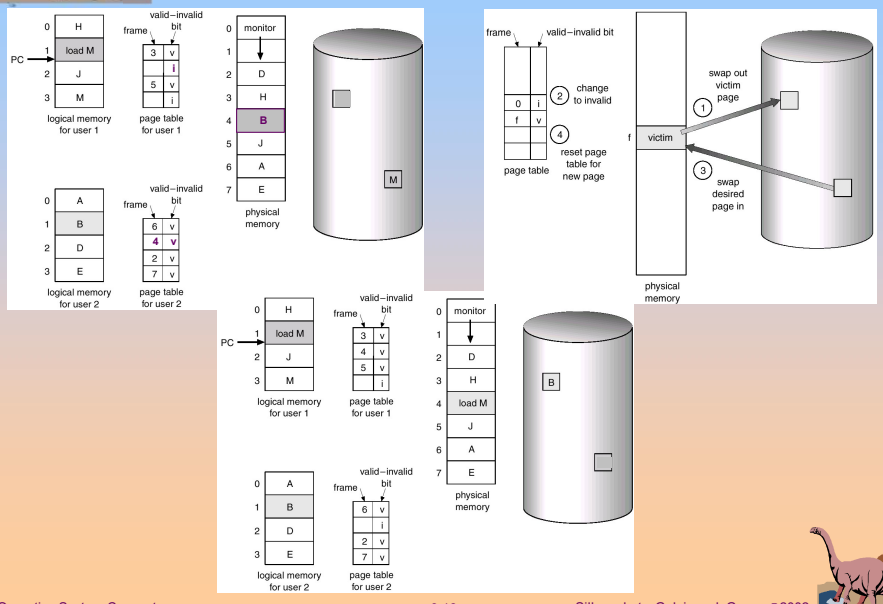
Page replacement – Substituição duma Página

- O SO terá que encontrar uma página em memória que não está a ser utilizada e substitui-la
  - Qual será a página para substituir ?
  - → necessidade de haver um algoritmo de substituição
  - Procure-se um algoritmo que minimize o numero de substituições num dado período de tempo
  - Qual é o desempenho deste processo ?

- O mecanismo da substituição das página complete a separação da memória lógica da memoria física.
- Assim um grande memória virtual pode ser fornecido usando uma memória física mais pequena.

15

Necessidade de : Page Replacement



16

Basic Page Replacement

1. Parar a execução do processo
2. Localizar a página no backing store / disco.
3. Localizar uma moldura livre
   - se existir então utilizá-la
   - se não selecionar uma vitima através de algum **algoritmo**
3. Inserir a página na moldura livre. Actualizar estruturas do SO -a tabela de páginas e tabela de molduras livre
4. Recomeçar a execução do processo.

17

Page Replacement Algorithms

- Alvo : Um algoritmo que minimize o numero de falhas
- Algoritmo Avaliação
  - Feita executando o algoritmo usando um dado sequencia de referencias a memoria, chamada *reference string*, e depois calculando o numero de *falhas de pagina*
  - Um exemplo duma sequencia de paginas (*page reference string or stream),* $\mathcal{R}$, será

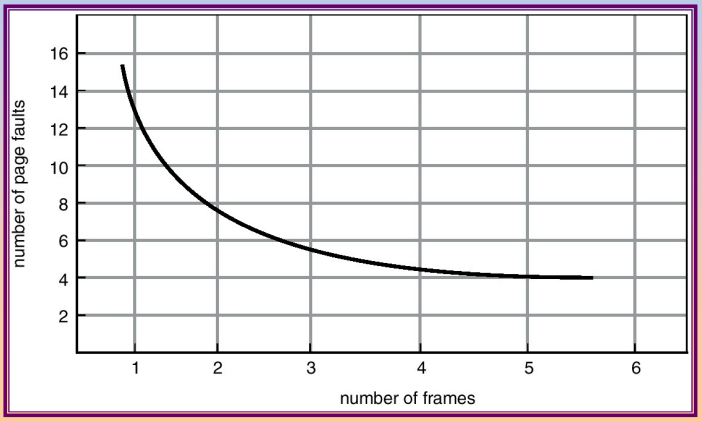
    $\mathcal{R}$ = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
- Algoritmos
  - FIFO
  - Random
  - Optimal
  - LRU

18

Graph of Page Faults Versus The Number of Frames

Antes de ver os algoritmos e exemplos considere o seguinte :

Pergunta Geral: Será que aumentando o numero de frames implica uma redução no numero de falhas de pagina ?



19

First-In-First-Out (FIFO) Algorithm

- Reference string: ℛ = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

Frame Nº: Page Nº

1	1	4	5		
2	2	1	3	9 page faults	
3	3	2	4		

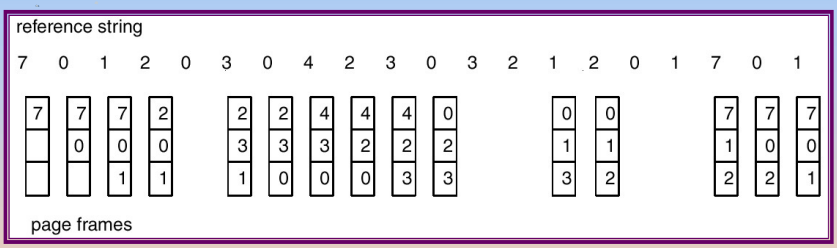
•4 frames

1	1	5	4		
2	2	1	5	10 page faults	
3	3	2			
4	4	3			

Anomalia de Belady

20

FIFO Page Replacement
Exemplo 2



Substituiçoes → ?

Hits → ?

Exercício : Com 4 frames ?

21

Random Replacement

A pagina de substituir é escolhida aleatoriamente entre os "m" molduras com probabilidade 1/m

Let page reference stream, ℛ = 2031203120316457

Frame	2	0	3	1	2	0	3	1	2	0	3	1	6	4	5	7
0	2	2	2	2	2	2	3	3	3	0	0	0	0	4	4	7
1		0	0	1	1	1	1	1	1	1	3	3	6	6	5	5
2			3	3	3	0	0	0	2	2	2	2	2	2	2	2

13 page faults

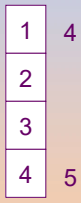
- **No knowledge of ℛ ⇒ not perform well**
- **Easy to implement**

22

(Beladys) Optimal Algorithm

- Algoritmo : Substituir a pagina que não vai ser usada durante o maior quantidade do tempo
- Exemplo 1 com 4 molduras

$\mathcal{R}$ = 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



6 page faults

- Como é que se pode saber qual é a pagina ? É Impossível prever o futuro !!
- É um benchmark contra qual outros algoritmos podem ser comparados.

23

Belady's Optimal Algorithm

- **Replace page with maximal forward distance: $y_t = \max_{x \in S_{t-1}(m)} FWD_t(x)$**

Let page reference stream, $\mathcal{R}$ = 2031203120316457

Frame	2	0	3	1	2	0	3	1	2	0	3	1	6	4	5	7
0	2	2	2													
1		0	0													
2			3													

24

Belady's Optimal Algorithm

- Replace page with maximal forward distance: $y_t = \max_{x \in S_{t-1}(m)} FWD_t(x)$

Let page reference stream, $\mathcal{R}$ = 2031203120316457

Frame	2	0	3	1	2	0	3	1	2	0	3	1	6	4	5	7
0	2	2	2													
1		0	0													
2			3													

$FWD_4(2) = 1$
$FWD_4(0) = 2$
$FWD_4(3) = 3$

25

Belady's Optimal Algorithm

- Replace page with maximal forward distance: $y_t = \max_{x \in S_{t-1}(m)} FWD_t(x)$

Let page reference stream, $\mathcal{R}$ = 2031203120316457

Frame	2	0	3	1	2	0	3	1	2	0	3	1	6	4	5	7
0	2	2	2	2												
1		0	0	0												
2			3	1												

$FWD_4(2) = 1$
$FWD_4(0) = 2$
$FWD_4(3) = 3$

26

Belady's Optimal Algorithm

- Replace page with maximal forward distance: $y_t = \max_{x \in S_{t-1}(m)} FWD_t(x)$

Let page reference stream, $\mathcal{R}$ = 2031203120316457

Frame	2	0	3	1	2	0	3	1	2	0	3	1	6	4	5	7
0	2	2	2	2	2	2										
1		0	0	0	0	0										
2			3	1	1	1										

27

Belady's Optimal Algorithm

- Replace page with maximal forward distance: $y_t = \max_{x \in S_{t-1}(m)} FWD_t(x)$

Let page reference stream, $\mathcal{R}$ = 2031203120316457

Frame	2	0	3	1	2	0	3	1	2	0	3	1	6	4	5	7
0	2	2	2	2	2	2	2									
1		0	0	0	0	0	3									
2			3	1	1	1	1									

$FWD_7(2) = 2$
$FWD_7(0) = 3$
$FWD_7(1) = 1$

28

Belady's Optimal Algorithm

- Replace page with maximal forward distance: $y_t = \max_{x \in S_{t-1}(m)} FWD_t(x)$

Let page reference stream, $\mathcal{R}$ = 2031203120316457

Frame	2	0	3	1	2	0	3	1	2	0	3	1	6	4	5	7
0	2	2	2	2	2	2	2	2	2	0	0	0	0	0	0	0
1		0	0	0	0	0	3	3	3	3	3	3	3	3	3	3
2			3	1	1	1	1	1	1	1	1	1	1	1	1	1

$FWD_{10}(2) = \infty$
$FWD_{10}(3) = 2$
$FWD_{10}(1) = 3$

29

Belady's Optimal Algorithm

- Replace page with maximal forward distance: $y_t = \max_{x \in S_{t-1}(m)} FWD_t(x)$

Let page reference stream, $\mathcal{R}$ = 2031203120316457

Frame	2	0	3	1	2	0	3	1	2	0	3	1	6	4	5	7
0	2	2	2	2	2	2	2	2	2	0	0	0	0	0	0	0
1		0	0	0	0	0	3	3	3	3	3	3	3	3	3	3
2			3	1	1	1	1	1	1	1	1	1	1	1	1	1

$FWD_{13}(0) = \infty$
$FWD_{13}(3) = \infty$
$FWD_{13}(1) = \infty$

Empate
??
FIFO
Random
LRU

30

Belady's Optimal Algorithm

- Replace page with maximal forward distance: $y_t = \max_{x \in S_{t-1}(m)} FWD_t(x)$

Let page reference stream, $\mathcal{R}$ = 2031203120316457

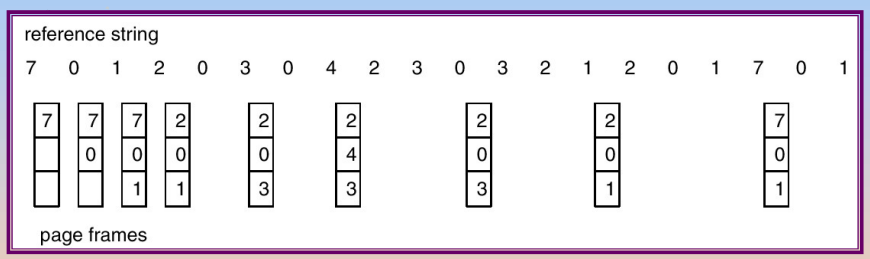
Frame	2	0	3	1	2	0	3	1	2	0	3	1	6	4	5	7
0	2	2	2	2	2	2	2	2	2	0	0	0	0	4	4	4
1		0	0	0	0	0	3	3	3	3	3	3	6	6	6	7
2			3	1	1	1	1	1	1	1	1	1	1	1	5	5

10 page faults

- Perfect knowledge of $\mathcal{R} \Rightarrow$ perfect performance
- Impossible to implement

31

Optimal Page Replacement
Exemplo 2



Usando 3 molduras

Misses/Hits → ?

Substituições → ?

32

Least Recently Used (LRU) Algorithm

O algoritmo é baseada na seguinte observação : Os programas acedem à memória com:

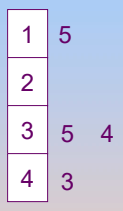
- Localidade temporal. Se um endereço for acedido agora, há uma grande probabilidade de ser acedido no futuro próximo (ciclos, rotinas de invocação frequente, dados importantes);
- Localidade espacial. Se um endereço for acedido, a probabilidade de os próximos acessos serem em endereços próximos é grande (execução sequencial, ciclos, arrays cujos dados são acedidos sequencialmente).

- ***Algoritmo de Substituição da Página Menos Utilizada Recentemente***
- *As páginas que foram muito utilizadas nas ultimas instruções serão, provavelmente, muito utilizadas novamente nas próximas instruções. Reciprocamente, as páginas que não têm sido utilizadas há longo tempo vão permanecer, provavelmente, sem uso por um longo tempo.*

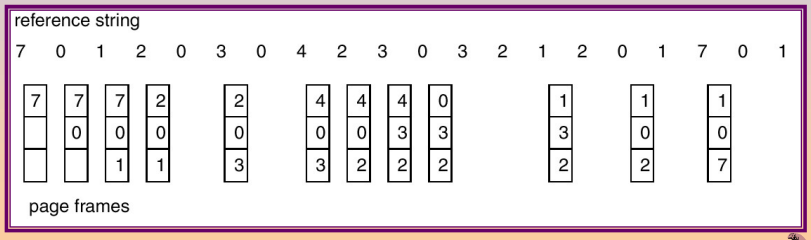
33

Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



- Mais um Exemplo



Substituiçoes → ? Hits → ?

34

Exemplo

- Num sistema de memória virtual paginada com três molduras quantas faltas de página aconteceriam usando os algoritmos de substituição de página (i) “LRU” (*least recently used*) e (ii) “Optimal” com a seguinte string de referência: (Nota: todas as molduras estão inicialmente vazias)
- 1, 3, 2, 3, 1, 1, 4, 2, 3, 5, 4, 2, 3, 4, 1.

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

35

Exemplo

- Num sistema de memória virtual paginada com três molduras quantas faltas de página aconteceriam usando os algoritmos de substituição de página (i) “LRU” (*least recently used*) e (ii) “Optimal” com a seguinte string de referência: (Nota: todas as molduras estão inicialmente vazias)
- 1, 3, 2, 3, 1, 1, 4, 2, 3, 5, 4, 2, 3, 4, 1.

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

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

36

LRU - IMPLEMENTAÇÃO

- LRU tornou-se o algoritmo preferido mas a sua implementação é difícil e dispendioso.
- Implementação com Contador
  - Cada pagina tem um relógio associado inicialmente null.
  - Cada vez que a pagina será referenciado (usada) actualize o valor deste relógio
  - Quando é necessário correr o LRU então baste comparar os valores dos relógios
  - O relógio pode ser implementado como um contador .. zero,um,dois etc cada vez que uma pagina é referenciado actualize-se o contador
- Implementação com Pilha
  - guarde-se uma estrutura de dados que represente uma pilha de paginas usadas
  - Pagina referenciada → mudar pagina para cima da pilha
  - Vantagem = não há procure linear para encontrar a pagina LRU aquando a substituição

37

Outros Algoritmos

Algoritmos para gerir substituição:

- First-In, First-Out FIFO – Bélády's Anomaly (1969)
- Random
- Optimal
- Least Recently Used LRU

…

- Least Frequently Used LFU
- Second Chance Replacement algorithms
- Not Frequently Used
- Adaptive Replacement Cache (IBM)

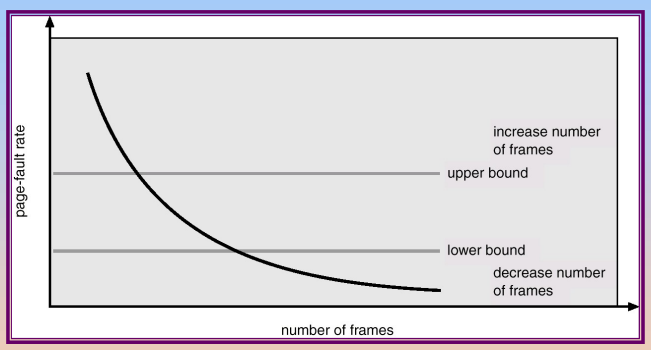
38

Modelo do Conjunto de Trabalho

- Alem duma politica de substituição **é necessário** definir uma **politica de alocação**
  - *Especificar quantas molduras um processo vai ter*
- Modelos atuis são baseados no conceito dum *working set*
- Definição : Conjunto de Trabalho (Working Set)
  - O conjuno de páginas que um processo está ativamente a referenciar
- Para que um programa processe eficientemente o seu conjunto de páginas de trabalho tem que ser mantido na memória.
- Outras páginas do processo que num dado momento não estão a ser utilizado podem estar invalidos.
  - Liberta espaço em memoria para outros processos
- Necissidade de optimizar o tamanho do conjunto de trabalho

39

Page-Fault Frequency Scheme

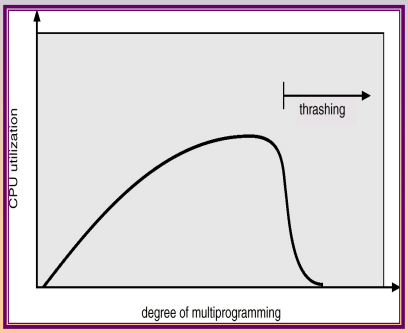


- Definir uma taxa de falhas de páginas aceitável (“acceptable” page-fault rate)
  - Se a taxa actual for demasiado baixo -> processo perde molduras.
  - Se a taxa actual for demasiado alto- > processe ganhe molduras.

40

Thrashing

- Se um processo não tiver páginas validos (frames) suficientes a taxa de falhas-de-páginas pode ser muito alto. Como consequências :
  - O sistema tem uma baixa taxa de utilização do CPU.
  - Portanto o SO julgue necessária aumentar o grau de multi-programação
  - Outro processo é adicionado ao sistema.
- **Thrashing** ≡ processes are busy swapping pages in and out.
  No useful work done!



41

Outros Assuntos

- Outras utilizações de memória virtual
  - COW (Copy on write)
  - Memory Mapped Files
- Page size selection
  - fragmentation
  - table size
- I/O Lock
- Locality

42

Outras Vantagens de Memória Virtual

Copy-On-Write

- Copy-on-Write (COW) permite que inicialmente um processo “pai” e “filho” partilham as mesmas páginas de memória.
- Se qualquer dos dois processos altera dados numa pagina partilhada então neste altura (e apenas nesta altura) é que a pagina será copiada. Sendo assim cada processo tem uma copia própria da pagina a alterar
- COW permite os processos serem criados duma maneira mais eficaz (menos memoria) e mais rápido.
- Páginas livre são alocadas dum conjunto de paginas livres mantido pelo SO.
- Exemplo: a chamada de sistema Linux fork()

43

Outras Vantagens de Memória Virtual

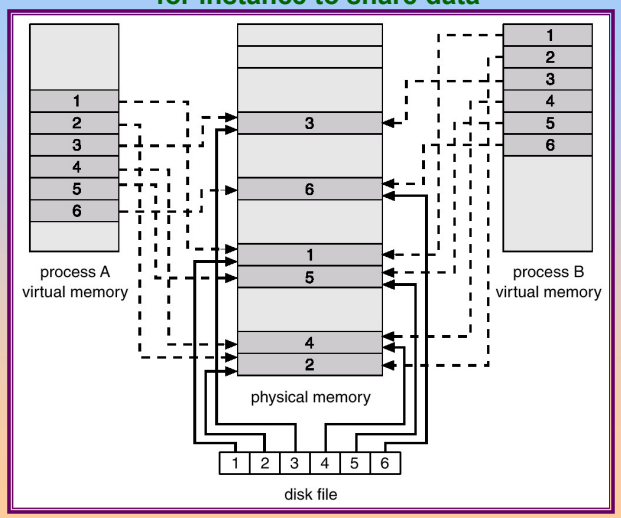
Memory-Mapped Files

- “Memory-mapped file I/O” permite “I/O dum ficheiro” ser tratado como uma rotina de acesso a memoria mapeando blocos do disco a paginas em memoria.
- Um ficheiro é inicialmente lido usando “demand paging”. Uma parte do ficheiro do tamanho duma página é lido do sistema de ficheiros para uma moldura. Depois qualquer leitura/escritura do ficheiro é tratado como acesso a memoria.
- Simplifique o acesso a um ficheiro tratando I/O via memória principal em vez das chamadas ao sistema **read() write()**
- Permite múltiplos processos partilhar um ficheiro através das páginas em memoria

44

Memory Mapped Files.

Two Processes sharing the same file via memory - for instance to share data



45

Page size selection

Determinação do tamanho de página a utilizar

- **Tamanho Grande ->**
  - Diminuição do tamanho de tabela de páginas
  - Mas em contra partida pode haver um aumento de fragmentação, nem todas as aplicações necessitam dum tamanho de página grande
- **Tamanho Pequeno ->**
  - Implica um tamanho demasiado grande da tabela de páginas

Soluções

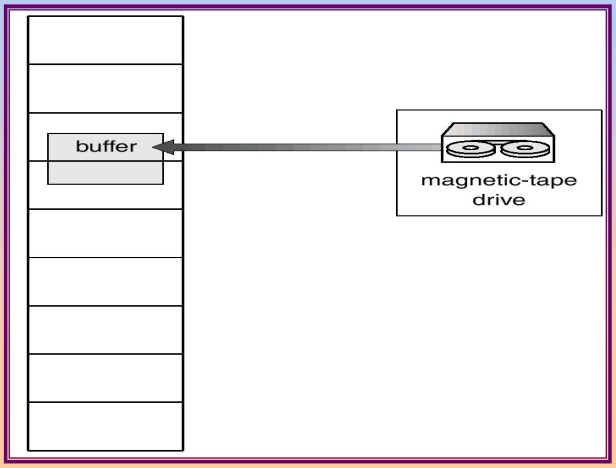
- **Fornecer possibilidade do administrador modificar o tamanho de página.**
- **Fornecer a possibilidade de ter múltiplos tamanho diferentes**
  - **Permite aplicações otimizar o tamanho de pagina para o seu caso e assim não deve haver aumento significativo de fragmentação**
- Ver o caso de windows xp !

46

I/O Interlock

I/O Interlock – As vezes as páginas tem que ser fechadas (*locked*) em memoria e não podem ser retiradas

Considere I/O. As páginas usadas durante a copia dum ficheiro dum dispositivo não podem ser vitimas dum algoritmo de substituição de páginas ...



47

Locality

- Program structure
  - **int A[][] = new int[1024][1024];**
  - **Int *A=(int *)malloc(1024.1024.sizeof(int)) ;**
  - Each row is stored in one page
  - Program 1

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

    1024 x 1024 potencial page faults

  - Program 2

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

    1024 potencial page faults

48

Operating System Example Windows NT

- Uses demand paging with **clustering**. Clustering brings in pages surrounding the faulting page.
- Processes are assigned **working set minimum** and **working set maximum**.
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory.
- A process may be assigned as many pages up to its working set maximum.
- When the amount of free memory in the system falls below a threshold, **automatic working set trimming** is performed to restore the amount of free memory.
- Working set trimming removes pages from processes that have pages in excess of their working set minimum.
- WIN-NT Internals

- IBM OS2 - Demand segmentation (Mais Complexo)

49

Resumo

- Memoria Vista dum Programador
  - Um grande espaço de endereçamento linear
    - Pode alocar blocos de memoria contíguos
  - O seu processo é “dono” da maquina
    - Tem um espaço de endereçamento privado
    - Outros processos não podem directamente interferir com o VAS do seu processo.
- Memoria Vista do Sistema
  - Virtual Address Space (VAS) dum processo dum utilizador criado mapeando paginas/partes do VAS para memoria que pode ser memoria principal ou disco.
    - Memoria dum processo dum utilizado pode não ser contigua
    - Alocação é dinâmica
    - Protecção é feito durante o processo de tradução dum endereço
  - SO gere muitos processos concorrentemente
    - Está sempre a trocar entre os processos
    - Quando o processo necessita dum recurso é trocado
      - p.ex., disk I/O para tratar dum page fault

50

References

- Additional Material from
  - Operating System Third Edition Gary Nutt
  - Tannenbaum