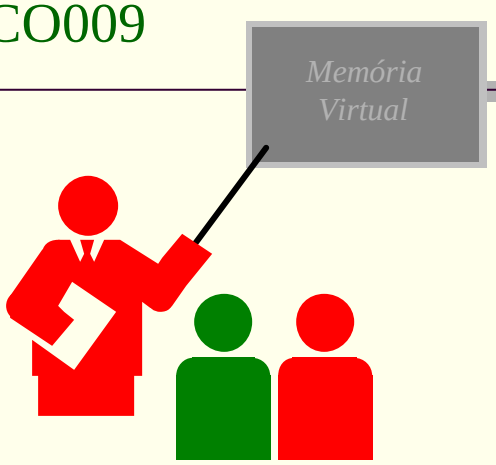
Modulo IV
Memória Virtual

Prof. Ismael H F Santos

Ementa

- Introdução aos Sistemas Operacionais
  - *Memória Virtual*
  - *Paginação*
  - *Segmentação*
  - Técnicas Mistas
  - Políticas de Realocação de Paginas
  - Working Set
  - Memory Mapped Files
  - Prepaging, Influencia TLB, etc

SOP – CO009



Gerência de Memória
Memória Virtual

- Memória Virtual

  · ***Memória Virtual (virtual memory)*** *é uma técnica sofisticada e poderosa de gerência de memória, onde as memórias principal e secundária são combinadas, dando ao usuário a ilusão de existir uma memória muito maior que a MP.*

  · *O conceito de* ***Memória Virtual*** *está baseado em desvincular o endereçamento feito pelo programa dos endererecços físicos da memória principal. Assim, os programas e suas estruturas de dados deixam de estar limitados ao tamanho da memória física disponível.*

Gerência de Memória
Memória Virtual

Memória Virtual

· *O mecanismo MVT –* ***Alocação Particionada Dinâmica****, apesar de ser uma evolução em relação ao MFT –* ***Alocação Particionada Estática****, ainda sofre o problema de* ***Fragmentação Externa****.*

· *Para que a área* **alocada a um processo seja contígua** *duas coisas podem ser feitas para reunir estas áreas*

Gerência de Memória
Memória Virtual

Memória Virtual

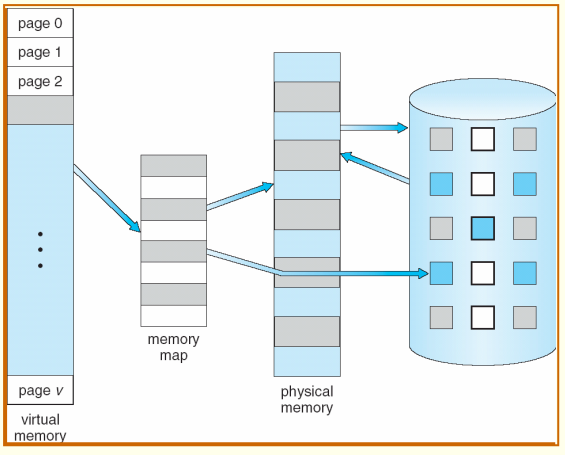
(i) *Compactação* ***- movimentação de memória livre para gerar uma região contígua através de movimentação dos programas em memória ou via Swapping.***

(ii) *Paginação* ***- permitir que a memória utilizada pelo programa seja não contígua ou ao menos contígua por partes.***

Background

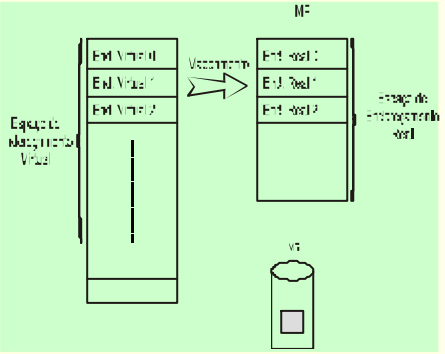
- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Virtual Memory That is Larger Than Physical Memory



Memória Virtual

- EEV, EER e Mapeamento



· *Um programa executando em ambiente de* ***memória virtual*** *não faz referência a* ***endereços reais,*** *mas apenas a* ***endereços virtuais.*** *No momento da execução de uma instrução, o* ***endereço virtual*** *é traduzido para um* ***endereço real****, pois o processador acessa apenas posições da MP. O mecanismo de tradução do endereço virtual para endereço físico é denominado* ***mapeamento*** *(****maping).***

Memória Virtual

· *O* ***EEV (espaço de endereçamento virtual)*** *não tem nenhuma relação direta com os endereços do* ***EER (espaço de endereçamento real)****. Um programa pode fazer referência a endereços virtuais que estejam fora dos limites do espaço real.*

· *Como os programas podem ser maiores que a memória física, somente parte deles pode estar residente na* ***MP*** *em um determinado instante. O SO utiliza a* ***MS*** *como extensão da* ***MP****. Quando um programa é executado, somente uma parte do código fica residente na* ***MP****, o restante permanece na* ***MS*** *até o momento de ser referenciado.*

Memória Virtual

· *Como conseqüência do mapeamento, um programa não precisa estar necessariamente contíguo na MP para ser executado. Nos SOs atuais a tarefa de tradução é realizada por um HW específico dentro da UCP (**UGM - unidade de gerenciamento de memória**), juntamente com o SO, de forma a não comprometer seu desempenho e torná-lo transparente para a aplicação.*

· *Como a maioria das aplicações tende a fazer referência a um número reduzido de endereços virtuais (**princípio da localidade**), somente uma pequena fração da tabela de mapeamento é realmente necessária, por isso criou-se um HW especial para mapear **endereços virtuais** para **endereços físicos** sem a necessidade de se fazer um acesso a tabela de mapeamento.*

Memória Virtual

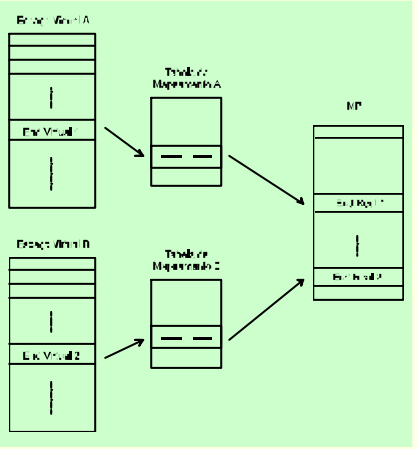
· *Este HW é chamado de **TLB - translation lookaside buffer** e tem o mesmo princípio de funcionamento da cache de memória.*

· *Quando o usuário desenvolve suas aplicações, ele ignora a existência dos endereços virtuais. Os **compiladores** e **linkers** se encarregam de gerar código executável em função desses enderêços, e o SO cuida dos detalhes de sua execução.*

Memória Virtual

· *Cada processo tem o mesmo* ***EEV (espaço endereçamento virtual)*** *como se possuísse a própria MV.*

· *O* ***mecanismo de tradução*** *se encarrega, então, de manter as tabelas de mapeamento exclusivas para cada processo. Quando um programa está sendo executado o SO utiliza a tabela de mapeamento do processo, no qual o programa executa, para realizar a tradução.*

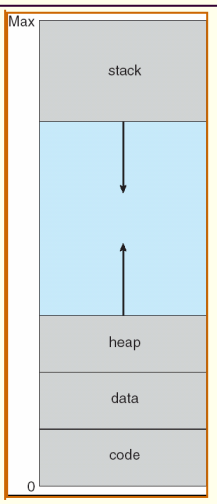


Memória Virtual

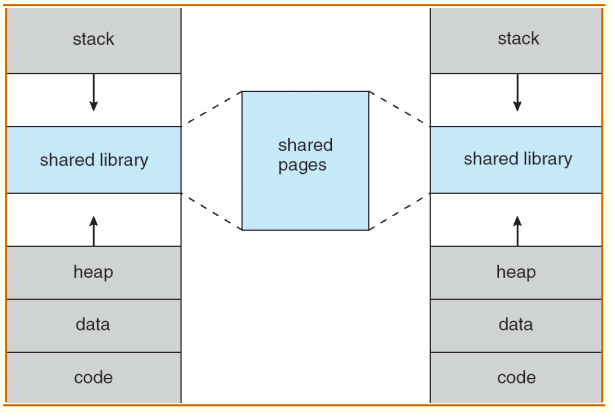
- *A informação correspondente à posição inicial da tabela de mapeamento é em geral indicada por um registrador, chamado* ***PTBR*** *(**page table base register**) e faz parte do contexto do HW do processo.*

- *A* ***tabela de mapeamento*** *mapeia* ***blocos de informação*** *cujo tamanho determina o número de entradas necessário na tabela.* **Quanto maior o bloco, menor número de entradas na tabela e, conseqüentemente, tabelas menores; entretanto blocos maiores aumentam o tempo de transferência do bloco entre MS e MP**. *Existem SOs que trabalham apenas com blocos de mesmo tamanho (**páginas**), outros que utilizam blocos de tamanho diferente (**segmentos**) e, ainda há SOs que trabalham com os dois sistemas.*

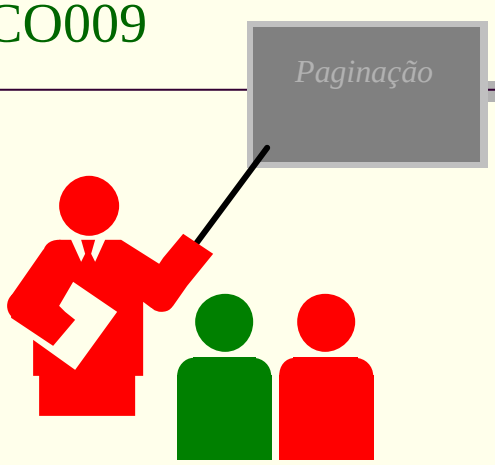
Virtual-address Space



Shared Library Using Virtual Memory



SOP – CO009



Paginação

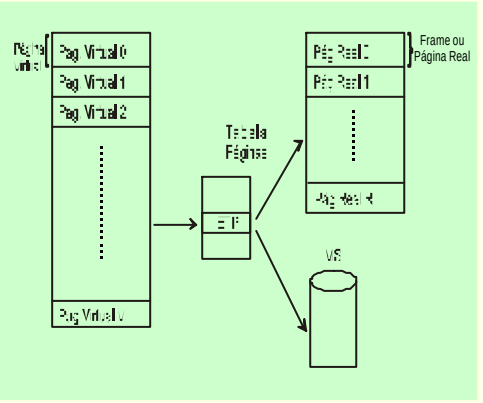
- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8192 bytes)
- Divide logical memory into blocks of same size called **pages**.
- Keep track of all free frames
- To run a program of size *n* pages, need to find *n* free frames and load program. Set up a page table to translate logical to physical addresses

Tabela de Paginas

Paginação

· *Nesta técnica o **EEV** e o EER são divididos em Blocos do mesmo tamanho, chamados **páginas.***

· *As **páginas** no **EEV** São chamadas **páginas Virtuais** e as **páginas** no **EER** são chamadas **páginas reais ou Frames.***



Memória Virtual

Paginação Antecipada - Prepaging

· *Existe uma tendência de os SOs modernos passarem a utilizar a **técnica de paginação antecipada (antecipatory paging).** Os problemas dessa técnica é que quando o SO erra na previsão das próximas páginas a serem referenciadas terá sido perdido tempo de processador e ocupado memória desnecessariamente.*

Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory

Paginação

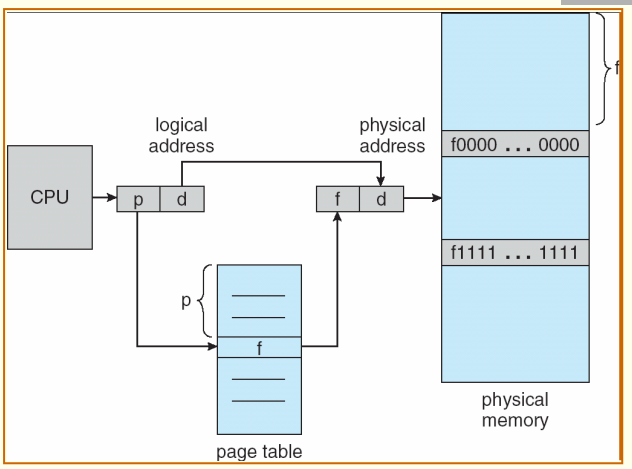
· *Quando um programa é executado, as **páginas virtuais** são carregadas da **MS** para a **MP** e colocadas nos **frames** disponíveis e a **tabela de páginas** é atualizada com os frames utilizados. Sempre que o programa fizer referência a um **endereço virtual**, o mecanismo de mapeamento localiza na **ETP** da tabela do processo o **endereço físico do frame** para ser acessado.*

· *O **tamanho da página** (e do frame) é definido pelo HW, sendo tipicamente uma potência de 2. Por exemplo, o IBM 370 usa 2k ou 4k bytes/pág. Em geral, se o **tamanho da página é P** então um **Endereço Virtual V** produz um **número de página virtual NPV** e um **deslocamento d** relacionados da seguinte forma:*

Address Translation Scheme

- Address generated by CPU is divided into:
  - *Page number (p)* – used as an index into a *page table* which contains base address of each page in physical memory
  - *Page offset (d)* – combined with base address to define the physical memory address that is sent to the memory unit

Address Translation Architecture

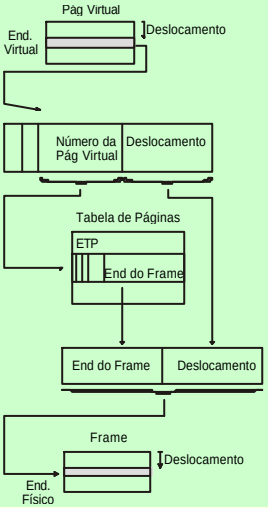


Paginação

NPV = V div P
d = V mod P

Onde:
P - tamanho da página;
V - Endereço Virtual;
NPV - número de página Virtual;
d - deslocamento dentro da página;
div *e* ***mod*** *são a divisão inteira e o resto da divisão.*

Exemplo: Pagina 66 apostila !

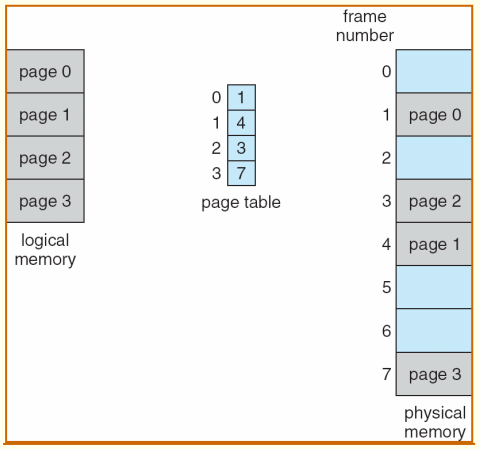


Paginação

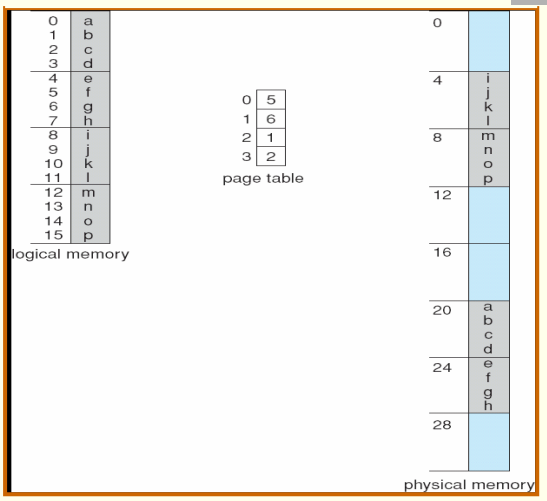
- **Paginação**
  - *Outra observação importante é que se o tamanho da página é uma potência de 2, e é 2^n, então os* ***n bits menos significativos*** *de um* ***endereço virtual*** *designam o deslocamento (offset) dentro de uma página, enquanto os* ***bits restantes mais significativos*** *designam o* ***número da página virtual (NPV).*** *Desta forma sendo o tamanho da página um múltiplo de 2 podemos evitar o processo de divisão (fórmula anterior).*

  - *É importante ressaltar que o* ***mecanismo de paginação*** *é uma forma de* ***relocação dinâmica****. Todo endereçamento virtual é mapeado pelo mecanismo de paginação para um endereço real.*

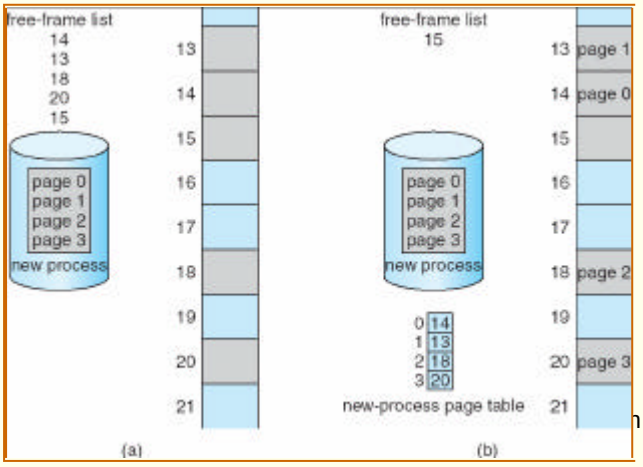
Paging Example



Paging Example



Free Frames



Implementation of Page Table

- Page table is kept in main memory
- *Page-table base register (*PTBR) points to the page table
- *Page-table length register* (PRLR) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**

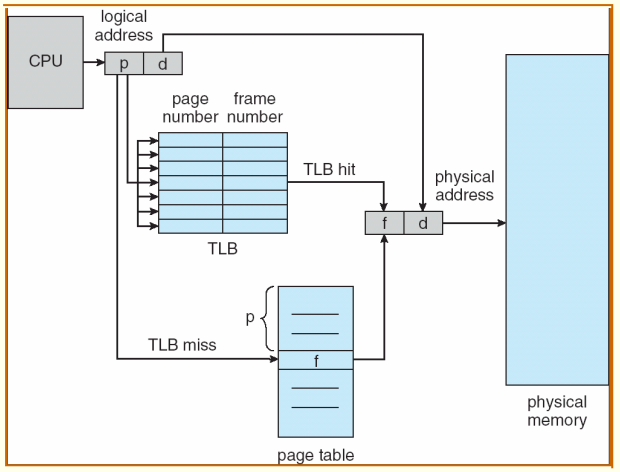
Associative Memory

- Associative memory – parallel search

Page #	Frame #

- Address translation (A´, A´´)
  - If A´ is in associative register, get frame # out
  - Otherwise get frame # from page table in memory

Paging Hardware With TLB



Effective Access Time

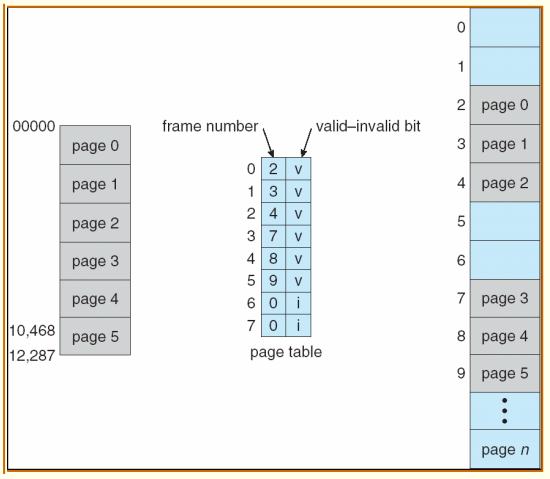
- Associative Lookup = ε time unit
- Assume memory cycle time is 1 microsecond
- Hit ratio – percentage of times that a page number is found in the associative registers; ration related to number of associative registers
- Hit ratio = α
- **Effective Access Time** (EAT)

$$EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$$
$$= 2 + \varepsilon - \alpha <= 2 \quad (\text{já que } \alpha > \varepsilon \text{ !})$$

Memory Protection

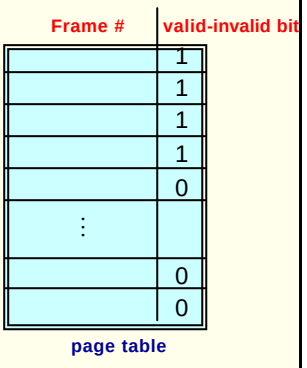
- Memory protection implemented by associating protection bit with each frame
- **Valid-invalid** bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
  - "invalid" indicates that the page is not in the process' logical address space

Valid/Invalid Bit In A Page Table

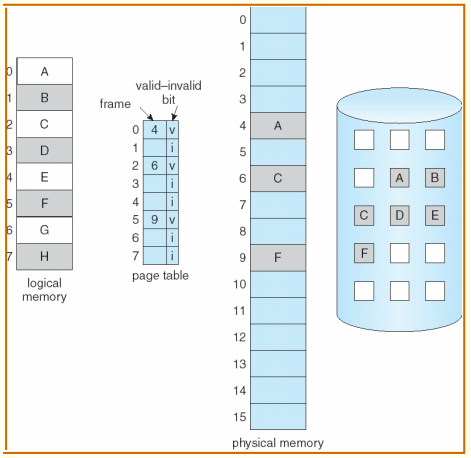


Valid-Invalid Bit

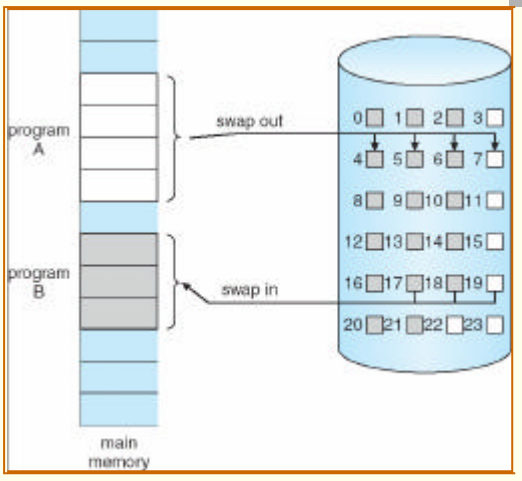
- With each page table entry a valid–invalid bit is associated (1 ⇒ in-memory, 0 ⇒ not-in-memory)
- Initially valid–invalid but is set to 0 on all entries
- During address translation, if valid–invalid bit in page table entry is 0 ⇒ page fault !



Page Table When Some Pages Are Not in Main Memory



Transfer of a Paged Memory to Contiguous Disk Space



Page-Fault

Page-Fault

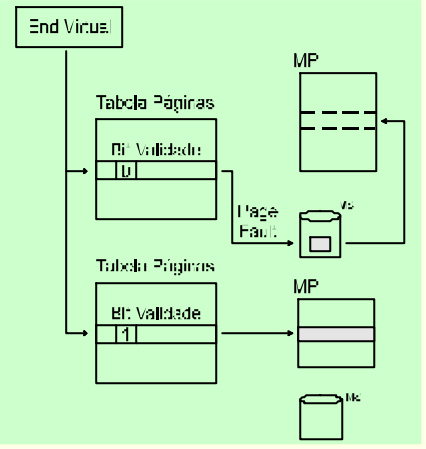
· *Além da informação sobre a localização da página virtual, a **ETP** possui outras informações, dentre elas o **bit de validade** que indica se uma página está ou não na MP (valid bit ou reference bit).*

· *Sempre que o processo faz referência a um endereço virtual o SO verifica se a página que contém o endereço referenciado está ou não na MP. Caso não esteja, o SO acusa a ocorrência de um **page-fault** (interrupção) e uma página é então transferida da MS para a MP.*

Page-Fault (cont.)

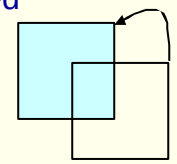
Page-Fault

· *As páginas dos processos são transferidas da MS para a MP, apenas quando são referenciadas. Este mecanismo, é chamado **paginação por demanda (demand paging)** e é conveniente, na medida em que leva para a MP apenas as páginas realmente necessárias para a execução do programa.*

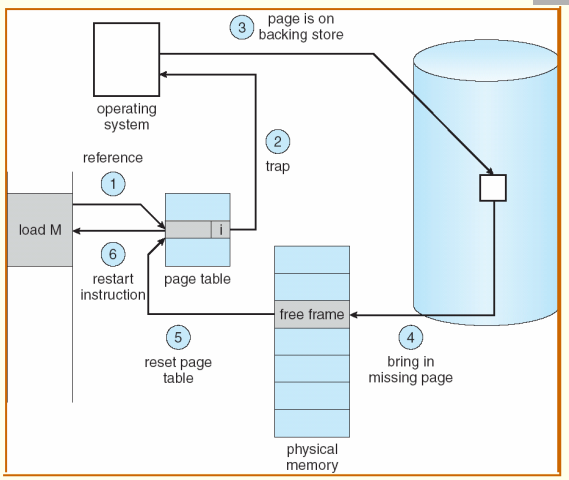


Page Fault

- If there is ever a reference to a page, first reference will trap to OS ⇒ page fault
- OS looks at another table to decide:
  - Invalid reference ⇒ abort.
  - Just not in memory.
- Get empty frame.
- Swap page into frame.
- Reset tables, validation bit = 1.
- Restart instruction: Least Recently Used
  - block move
  - auto increment/decrement location



Steps in Handling a Page Fault



Proteção

Proteção em Paginação

- *O **mecanismo de proteção** para sistemas paginados é feito através de bits de proteção associados a cada página (na ETP). Em geral um bit é reservado para identificar se a **página é read/write ou read only**. Uma tentativa de se escrever em uma página RO ocasiona uma **interrupção chamada violação de memória (memory protection violation).***

- *A arquitetura da máquina define o range de endereços válidos para um programa. Por exemplo uma máquina com MAR de 16 bits pode gerar endereços virtuais entre 0 e 65535. Com os **registradores de fronteira** (barrier registers) ou **base e limite** (base limit register) nós podemos trapear endereços gerados erroneamente por programas.*

Fragmentação

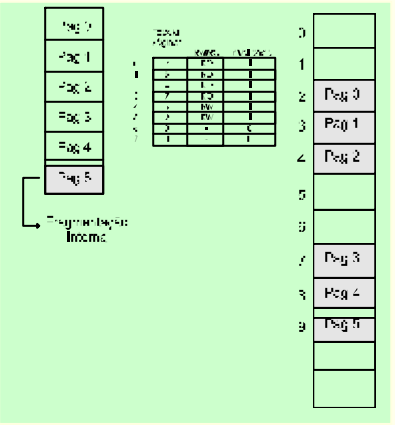
Fragmentação

- ***Fragmentação** também está presente em **sistemas paginados**, só que em menor escala, se comparada com a de outras organizações já vistas. A fragmentação só é encontrada, realmente, na última página, quando o código não a ocupa por completo. Maior ou menor fragmentação depende do tamanho da página.*

- *O tamanho da página influencia outros fatores tais como:*
  - ***tamanho das tabelas de mapeamento;***
  - ***taxa de paginação, expressa pelo número de page-faults do sistema por unidade de tempo;***
  - ***percentual de utilização da MP***

Fragmentação (cont.)

- **Fragmentação**
  - *Um problema colateral gerado pela **fragmentação interna** é que referências a pág 5 (conforme figura) mesmo que além da área definida pelo programa não gerarão **violação de endereço!***



Page Table Structure

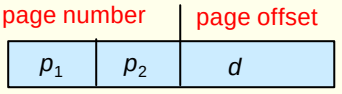
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table

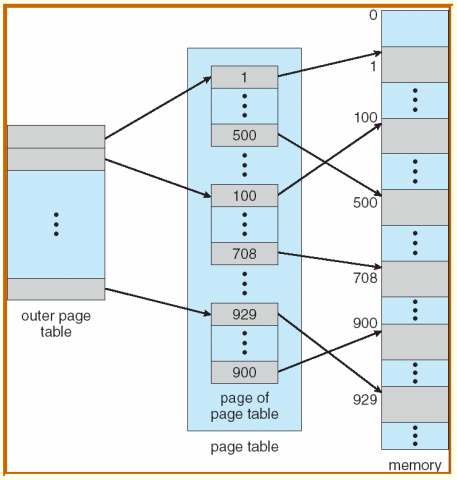
Two-Level Paging Example

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits
  - a page offset consisting of 12 bits
- Since the page table is paged, the page number is further divided into:
  - a 10-bit page number
  - a 10-bit page offset



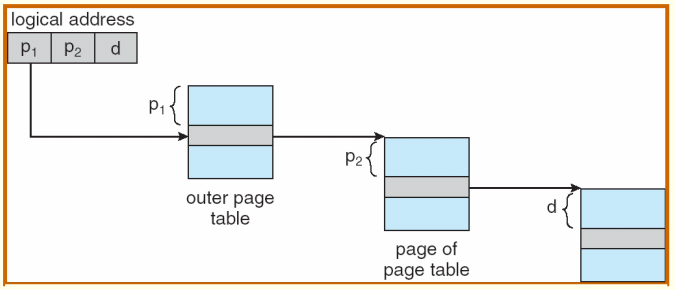
- Thus, a logical address is as follows:
  - where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

Two-Level Page-Table Scheme



Address-Translation Scheme

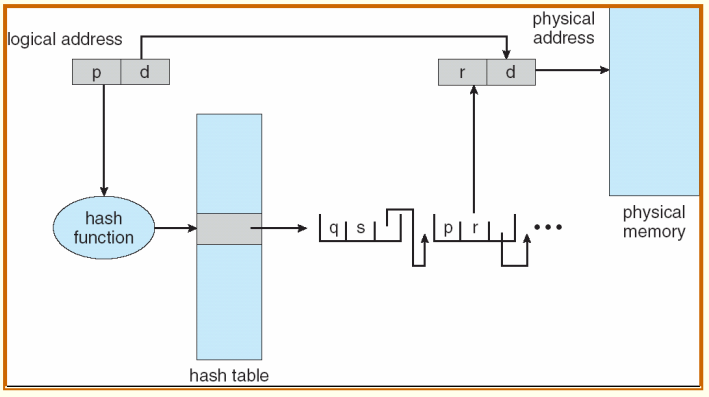
- Address-translation scheme for a two-level 32-bit paging architecture



Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location.
- Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.

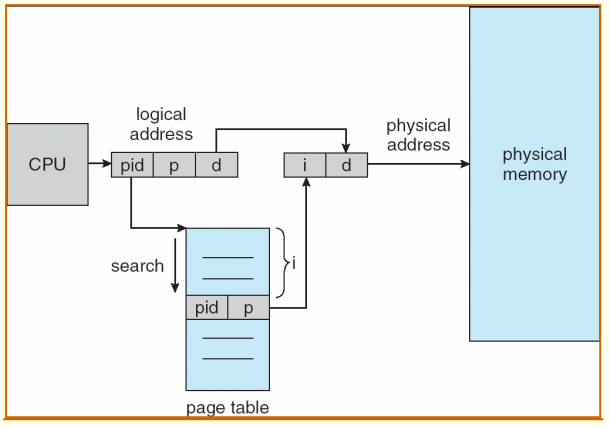
Hashed Page Table



Inverted Page Table

- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries

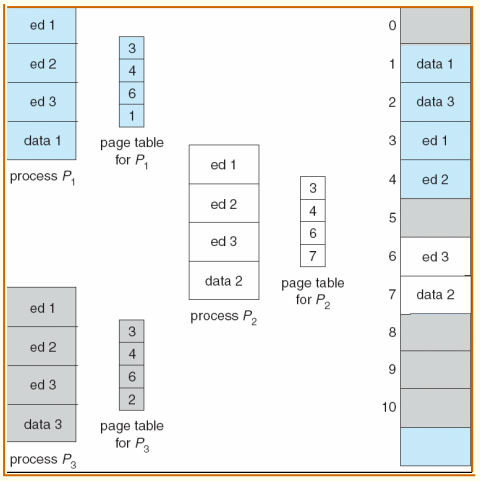
Inverted Page Table Architecture



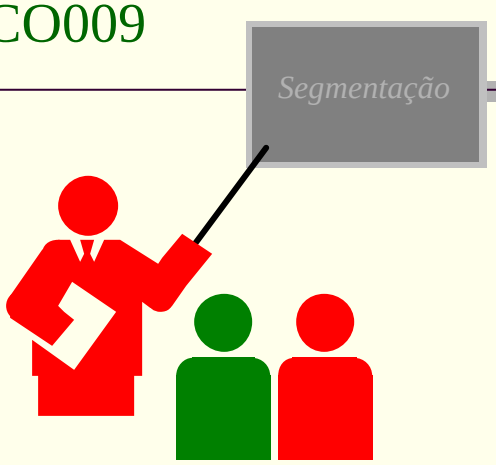
Shared Pages

- **Shared code**
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes
- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example



SOP – CO009



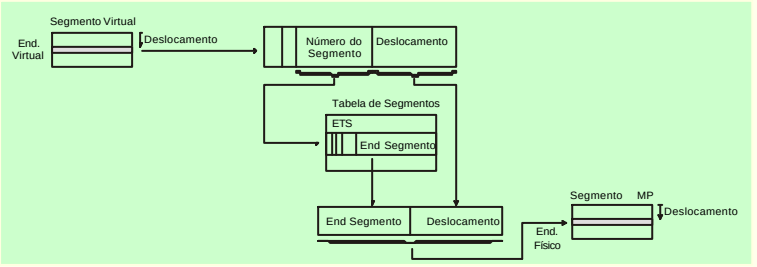
Segmentação

- **Segmentação**
  - *Técnica de gerência de MP, onde os programas são divididos logicamente em blocos (segmento de dados, código, stack). Estes blocos têm tamanhos diferentes e são chamados* ***segmentos****, cada um com seu próprio espaço de endereçamento.*
  - *A diferença em relação à* ***paginação*** *é que a* ***segmentação*** *permite uma relação entre a lógica do programa e sua divisão na MP (porque?).*

Segmentação

Segmentação

- *O mecanismo de mapeamento é semelhante ao de **paginação**. Os **segmentos** são mapeados através de tabelas de mapeamento e os endereços são compostos pelo número do segmento e um deslocamento dentro do segmento.*



Alocação de Segmentos

- *O SO mantém uma tabela com as áreas livres e ocupadas da MP. Quando um processo é carregado para MP, o SO localiza um espaço livre que o acomode. As estratégias para escolha da área livre podem ser as mesmas do MVT, ou seja, **BEST-FIT, WORST-FIT, FIRST-FIT.***

- *Na **segmentação**, apenas os segmentos referenciados são transferidos da MS para MP. Logo, para ser mais eficiente, os programas devem estar bem modularizados.*

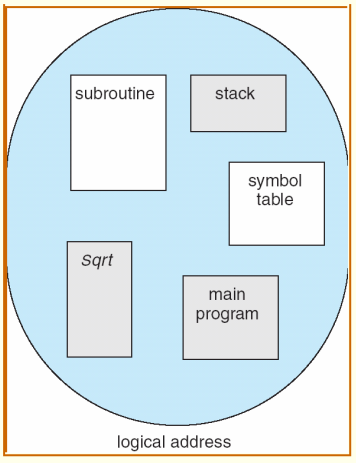
- *O problema de **fragmentação** também ocorre nesse modelo, quando as áreas livres são tão pequenas, que não acomodam nenhum segmento que necessite ser carregado.*

Segmentation

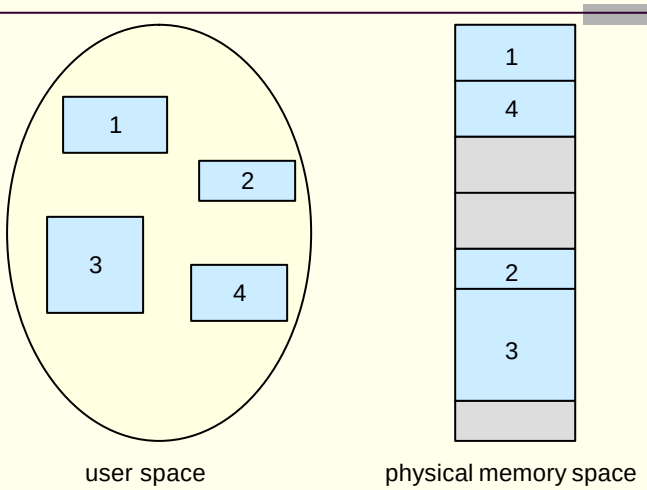
- Memory-management scheme that supports user view of memory
  - A program is a collection of segments. A segment is a logical unit such as:

    main program,
    procedure,
    function,
    method,
    object,
    local variables, global variables,
    common block,
    stack,
    symbol table, arrays

User's View of a Program



Logical View of Segmentation



Segmentation Architecture

- Logical address consists of a two tuple:
  <segment-number, offset>,
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory
  - *limit* – specifies the length of the segment
- *Segment-table base register (STBR)* points to the segment table's location in memory
- *Segment-table length register (STLR)* indicates number of segments used by a program;
- segment number *s* is legal if *s* < STLR

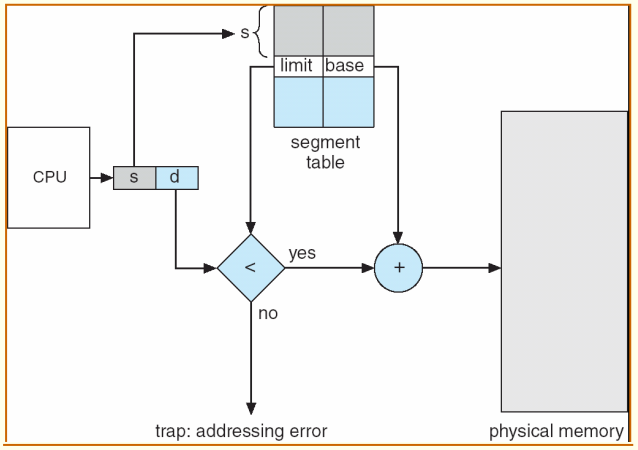
Segmentation Architecture (Cont.)

- **Relocation**.
  - dynamic
  - by segment table
- **Sharing**.
  - shared segments
  - same segment number
- **Allocation**.
  - first fit/best fit
  - external fragmentation

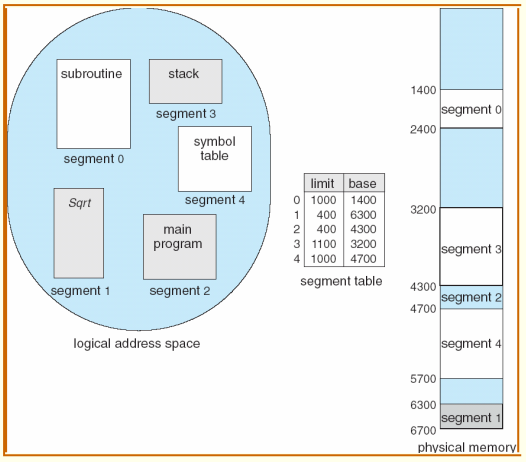
Segmentation Architecture (Cont.)

- Protection. With each entry in segment table associate:
  - validation bit = 0 ⇒ illegal segment
  - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram

Address Translation Architecture



Example of Segmentation



Compartilhamento de Segmentos

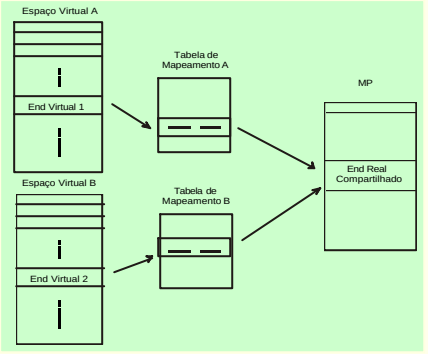
Compartilhamento de Memória

· *Em sistemas multiprogramáveis, é comum usuários utilizarem certos programas simultaneamente (**código reentrante**), o que evita que várias cópias de um mesmo programa ocupem a memória desnecessariamente. Exemplos são os utilitários do sistema, como **compiladores**, **editores de texto** ou, mesmo, algumas aplicações de usuários.*

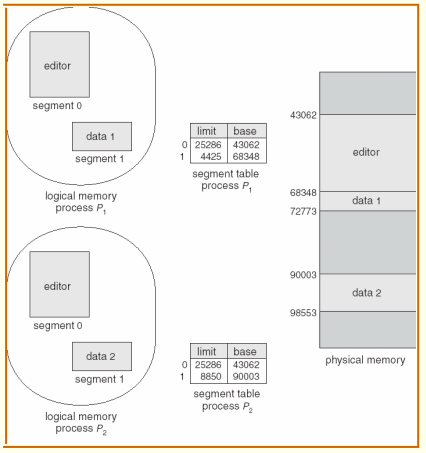
· *Para compartilharmos código e/ou dados entre vários processos basta que as entradas das **páginas/segmentos** apontem para os mesmos frames/segmentos na memória principal. Dessa forma, é possível reduzir-se o número de programas na memória e aumentar o número de usuários compartilhando o mesmo recurso conforme mostra a figura.*

Compartilhamento de Segmentos

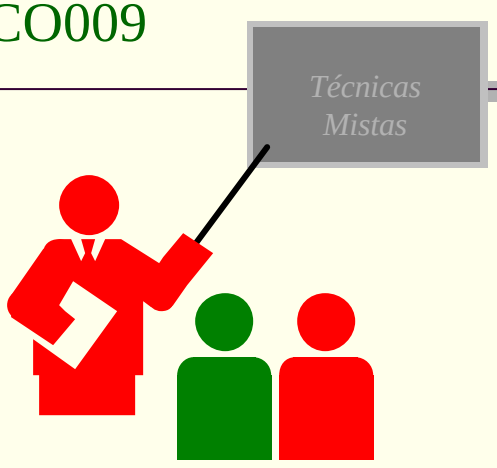
· *O **compartilhamento** de **segmentos** é mais simples de implementar do que o de **páginas**. O compartilhamento de estruturas de dados dinâmicas na **paginação** implica alocação de novas páginas e, consequentemente, o ajuste das tabelas de mapeamento, na **segmentação**, as tabelas devem ter ajustado apenas o tamanho do segmento.*



Sharing of Segments

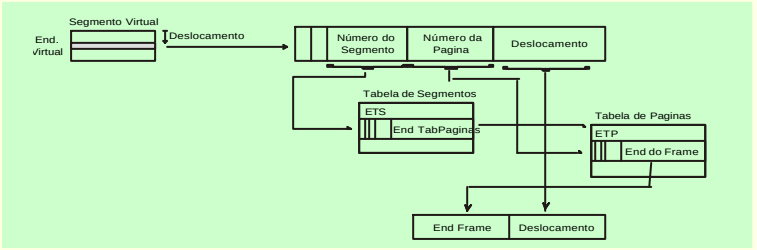


SOP – CO009



Técnicas Mistas

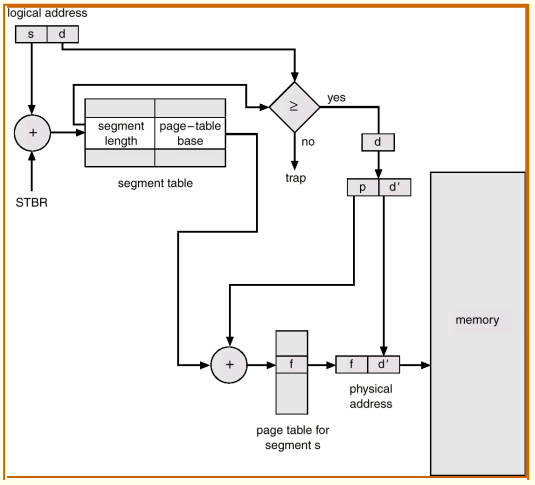
- **Segmentação com Paginação**
  - *Permite-se a divisão lógica dos programas em segmentos e, por sua vez, cada segmento é dividido, fisicamente, em páginas. Neste sistema, um endereço é formado pelo número do segmento, um número de página dentro do segmento e um deslocamento e dentro de uma página.*



Segmentation with Paging – MULTICS

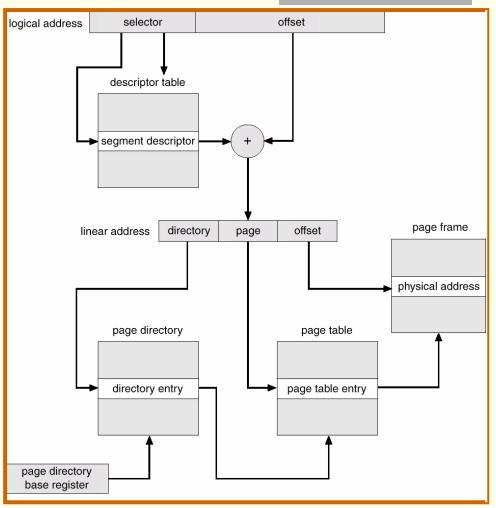
- The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments
- Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a *page table* for this segment

MULTICS Address Translation Scheme



Segmentation with Paging – Intel 386

- As shown in the following diagram, the Intel 386 uses segmentation with paging for memory management with a two-level paging scheme



Linux on Intel 80x86

- Uses minimal segmentation to keep memory management implementation more portable
- Uses 6 segments:
  - Kernel code
  - Kernel data
  - User code (shared by all user processes, using logical addresses)
  - User data (likewise shared)
  - Task-state (per-process hardware context)
  - LDT
- Uses 2 protection levels:
  - Kernel mode
  - User mode

SOP – CO009

Políticas de Relocação de Páginas

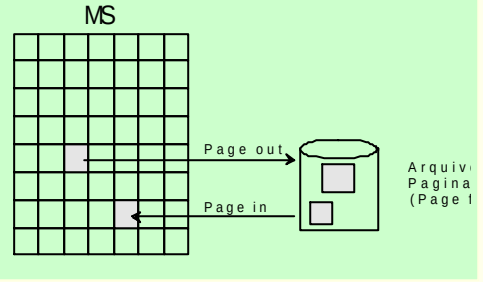
Realocação de Paginas

- **Políticas de Realocação de Páginas**
  - *O maior problema na gerência de memória virtual por paginação não é decidir que página carregar para a MP, mas quais páginas remover.*
  - *Quando o limite do ws do processo (wss) é alocado, e este necessita de frames, o SO deve intervir e escolher, dentre as diversas páginas do seu ws, quais que devem ser liberadas.*
  - *Sempre que o SO libera uma página, cujo seu conteúdo tenha sido alterado, ele antes deverá gravá-la na MS (page out) no **arquivo de paginação** (page file) onde as páginas alteradas são armazenadas.*

Realocação de Paginas

- *Sempre que uma dessas páginas for novamente referenciada, ela será trazida novamente para o ws do processo (page in). O SO controla o salvamento de páginas através do **bit de modificação** (dirty/modify bit), que existe na entrada de cada tabela de páginas.*

- *Principais algoritmos:*
  - *Aleatória*
  - ***FIFO***
  - ***LRU***
  - ***NRU***
  - ***LFU***



What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

Performance of Demand Paging

- Page Fault Rate $0 \leq p \leq 1.0$
  - if p = 0 no page faults
  - if p = 1, every reference is a fault
- Effective Access Time (EAT)

EAT = (1 – p) * memory access
+ p * (page fault overhead
+ [swap page out]
+ swap page in
+ restart overhead)

Demand Paging Example

- Memory access time = 1 microsecond
- 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out
- Swap Page Time = 10 ms = 10,000 microsec

  EAT = (1 – p) x 1 + p (15000)

  EAT = 1 + 15000p (in microsec)

Process Creation

- Virtual memory allows other benefits during process creation:
  - Copy-on-Write
  - Memory-Mapped Files (later)

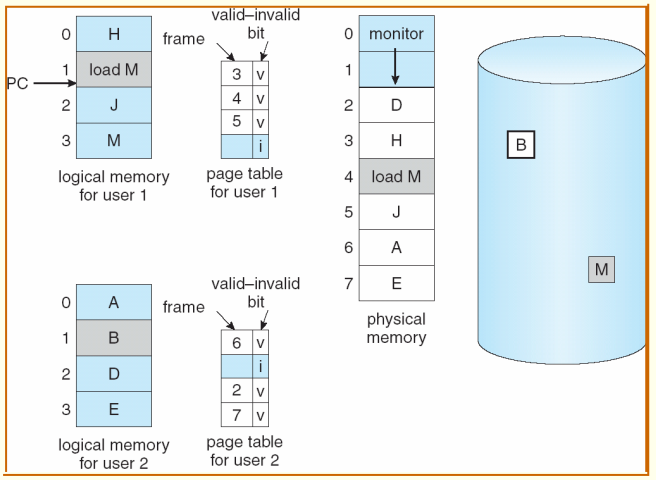
Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory
  - If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- Free pages are allocated from a **pool** of zeroed-out pages

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement
- Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

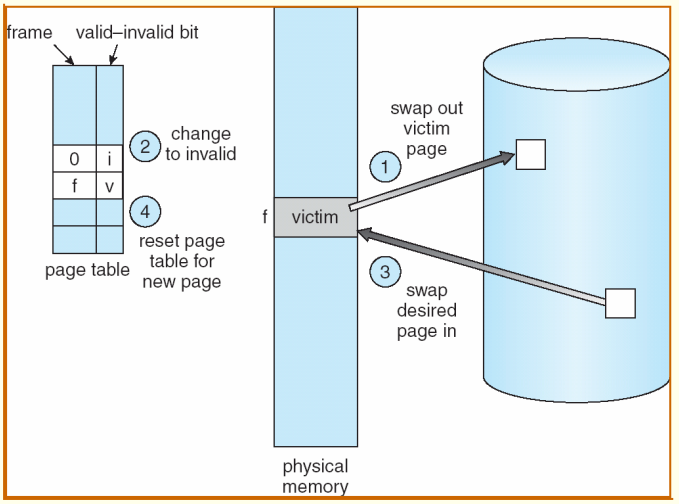
Need For Page Replacement



Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim** frame
3. Read the desired page into the (newly) free frame. Update the page and frame tables.
4. Restart the process

Page Replacement

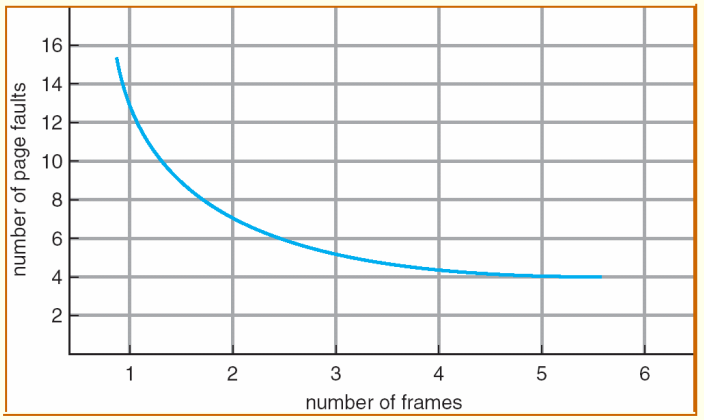


Page Replacement Algorithms

- Want lowest page-fault rate
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
- In all our examples, the reference string is

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

Graph of Page Faults Versus The Number of Frames



Políticas de Realocação de Paginas

· ***Aleatória -*** *Não utiliza critério algum de seleção. Todas as páginas do ws tem igual chance de ser selecionadas inclusive as páginas que são freqüentemente referenciadas. Apesar de ser implementada facilmente é muito ineficiente;*

Políticas de Realocação de Paginas

FIFO - First In First Out - *A página que primeiro foi utilizada será a primeira a ser escolhida para ser substituída. Sua implementação é simples, sendo necessária apenas o uso de uma fila onde as páginas mais antigas estão no início e as mais recentes no final. O problema acontece quando páginas que são constantemente referenciadas, como é o caso das páginas de código dos utilitários do sistema, são substituídas devido ao fator tempo e o SO tem que fazer retorna-las novamente para a memória várias vezes;*

First-In-First-Out (FIFO) Algorithm

- Reference string: 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)

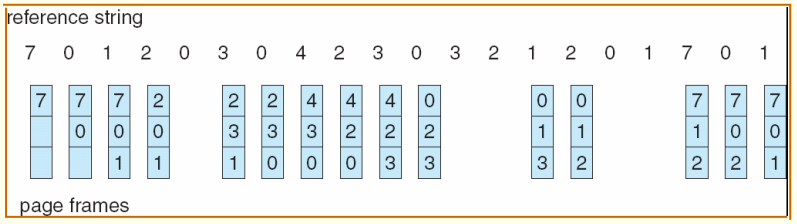
1	1	4	5		
2	2	1	3	9 page faults	
3	3	2	4		

  - 4 frames

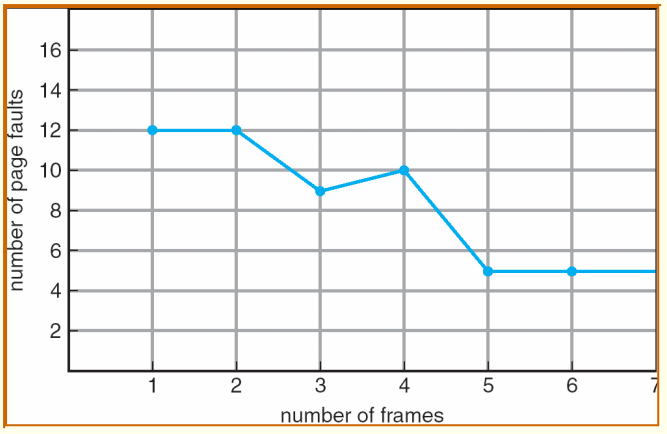
1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

- FIFO Replacement – **Belady's Anomaly**
  - more frames ⇒ more page faults

FIFO Page Replacement

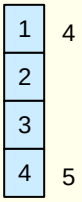


FIFO Illustrating Belady's Anomaly



Optimal Algorithm

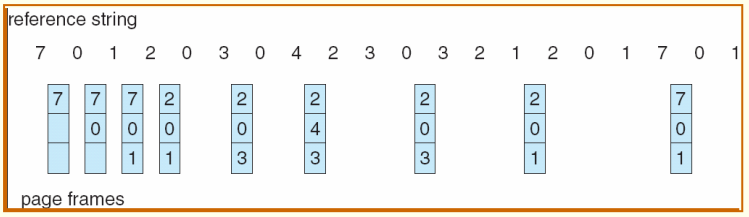
- Replace page that will not be used for longest period of time
- 4 frames: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



6 page faults !!!

- How do you know this?
- Used for measuring how well your algorithm performs

Optimal Page Replacement

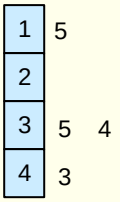


Políticas de Realocação de Paginas

· ***LRU - Least Recently Used -*** *Seleciona a página menos recentemente utilizada. Apesar de ser uma boa estratégia é difícil de ser implementada devido ao grande overhead causado pela atualização, em cada página referenciada, do momento do ultimo acesso, além do algoritmo de busca dessas páginas;*

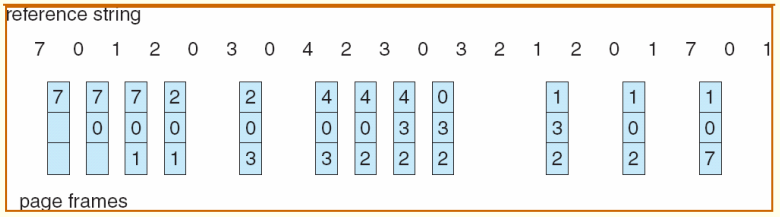
Least Recently Used (LRU) Algorithm

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



- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change

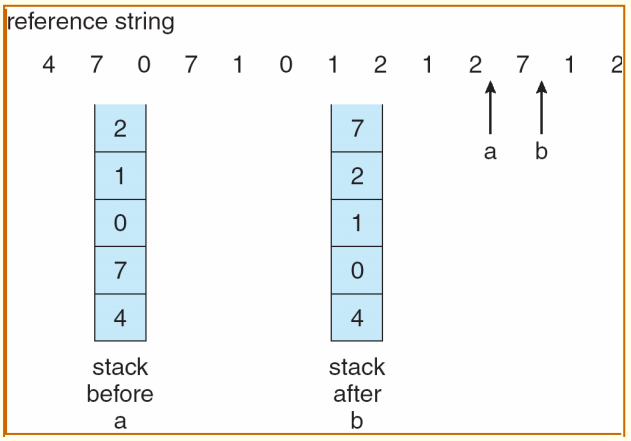
LRU Page Replacement



LRU Algorithm (Cont.)

- Stack implementation – keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement

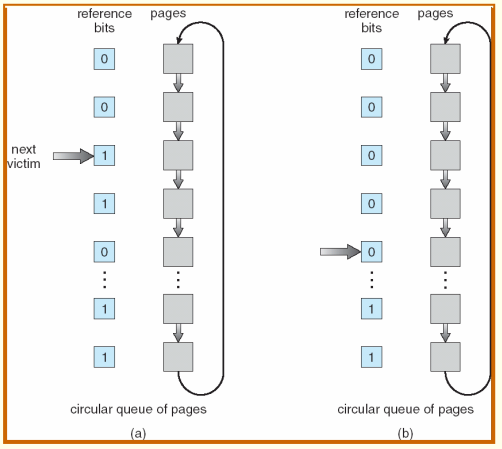
Use Of A Stack to Record The Most Recent Page References



LRU Approximation Algorithms

- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace the one which is 0 (if one exists). We do not know the order, however.
- Second chance
  - Need reference bit
  - Clock replacement
  - If page to be replaced (in clock order) has reference bit = 1 then:
    - set reference bit 0
    - leave page in memory
    - replace next page (in clock order), subject to same rules

Second-Chance (clock) Page-Replacement Algorithm



Políticas de Realocação de Paginas

· ***NUR - Not recently Used -*** *Escolhe-se a página que não foi recentemente utilizada. Nessa estratégia existe um bit, que permite ao SO a implementação do algoritmo. O* ***bit de referência*** *indica quando a página foi referenciada ou não, e está associado a cada entrada da tabela de páginas. No início todas as páginas estão com o bit zerado indicando que não foram referenciadas. À medida que as páginas são referenciadas, o* ***flag*** *associado a cada página é modificado pelo HW. Depois de um certo tempo, é possível saber quais páginas foram referenciadas ou não;*

Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- **LFU Algorithm**: replaces page with smallest count
- **MFU Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Gerência de Memória
Memória Virtual

· ***LFU - Least Frequently Used -*** *Neste esquema, a página menos freqüentemente utilizada será escolhida. Para isso é mantido um contador do número de referências feitas às paginas. A página que tiver o contador com o menor número de referências será a página escolhida. O algoritmo privilegia as páginas que são bastante utilizadas. Essa parecer uma boa estratégia, porém, as páginas que entrarem mais recentemente no* ***ws*** *serão, justamente, aquelas que estarão com os contadores com o menor valor;*

Allocation of Frames

- Each process needs *minimum* number of pages
- Example: IBM 370 – 6 pages to handle SS MOVE instruction:
  - instruction is 6 bytes, might span 2 pages
  - 2 pages to handle *from*
  - 2 pages to handle *to*
- Two major allocation schemes
  - fixed allocation
  - priority allocation

Fixed Allocation

- Equal allocation – For example, if there are 100 frames and 5 processes, give each process 20 frames.
- Proportional allocation – Allocate according to the size of process

s_i = size of process p_i

$S = \sum s_i$

m = total number of frames

a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 64$

$s_i = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 64 \approx 5$

$a_2 = \frac{127}{137} \times 64 \approx 59$

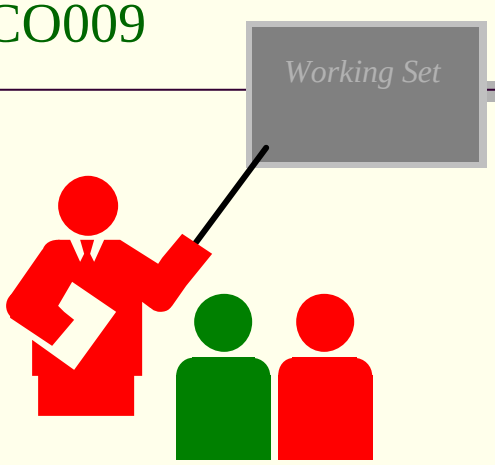
Priority Allocation

- Use a proportional allocation scheme using priorities rather than size
- If process P_i generates a page fault,
  - select for replacement one of its frames
  - select for replacement a frame from a process with lower priority number

Global vs. Local Allocation

- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
- **Local replacement** – each process selects from only its own set of allocated frames

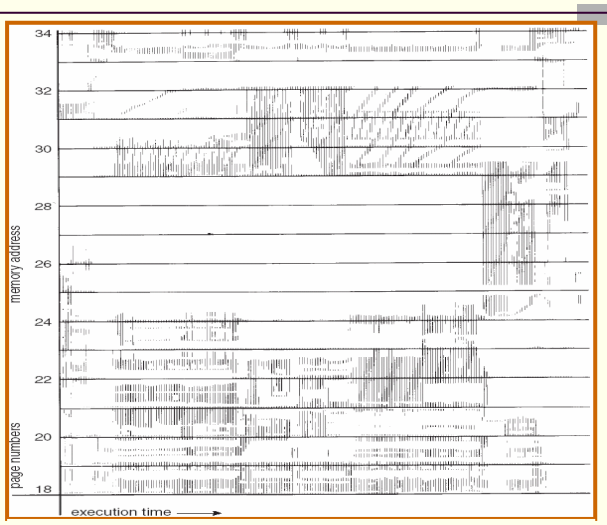
SOP – CO009



Demand Paging and Thrashing

- Why does demand paging work?
  Locality model
  - Process migrates from one locality to another
  - Localities may overlap
- Why does thrashing occur?
  Σ size of locality > total memory size

Locality In A Memory-Reference Pattern



Working Set

- **Working-Set**

  · *O conceito de **working set** surgiu a partir da análise da taxa de paginação dos processos. Quando um programa iniciava a sua execução, percebia-se uma elevada taxa de **page-faults**, que se estabilizava com o decorrer de sua execução.*

  · *Este fato está ligado diretamente a um outro conceito-chave na estratégia de gerência de memória **chamado princípio de localidade:***

Working Set

"Programas tendem a reutilizar dados e instruções que foram utilizados recentemente. Uma heurística aplicada a quase todos os programas é que geralmente um programa gaste 90% do seu tempo de execução em somente 10% do código".

Donald Knuth

· *Uma implicação deste princípio é que baseado no passado recente da execução de um programa, alguém pode predizer com uma precisão razoável quais instruções e dados serão referenciados em um futuro próximo. É nesta informação que se baseiam as políticas de gerenciamento de cache e gerenciamento de política de realocação de páginas.*

Working Set

Existem dois tipos de localidade:

· ***Localidade temporal*** *- itens recentemente acessados têm alta probabilidade de serem acessados em um futuro próximo.*

· ***Localidade espacial*** *- itens cujos endereços são próximos tendem a ser acessados em um futuro próximo.*

A localidade tem muito a ver com a forma que a aplicação foi escrita. Normalmente, se um programa foi desenvolvido utilizando técnicas estruturadas, o conceito de localidade quase sempre é válido.

Working Set

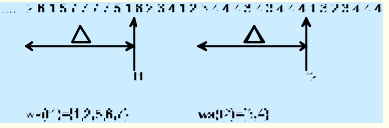
· *A partir da observação do princípio da localidade, formulou-se a teoria do **working set (ws)**. O ws é o conjunto de páginas que um processo referencia constantemente, e, por isso mesmo, deve permanecer na MP. Caso contrário, o sistema poderá sofrer com a elevada taxa de paginação, comprometendo sua performance.*

· *Quando um processo é criado, todas as suas páginas estão na MS. À medida que acontecem referências às páginas virtuais, elas são transferidas para o ws do processo na MP (page in). O ws usa um parâmetro, D, para definir a janela de working set (ws window). O conjunto das últimas D páginas referenciadas forma o ws. Se uma página está sendo acessada, então ela está no working set.*

Working Set

· *Se a página não está sendo acessada ela deixará o ws após D referências a outras páginas distintas. Portanto o ws é uma aproximação da localidade de um programa. A acurácia do ws depende da seleção do parâmetro D. Se D for muito pequeno, ele não abrigará todo ws; se for muito grande, ele pode conter diversas localidades de um programa.*

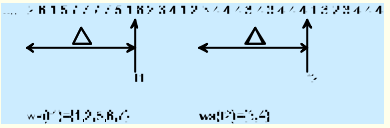
Exemplo: Apostila pagina 70 !



· *A propriedade mais importante no ws é o seu tamanho. Se nós computarmos o tamanho do working set de cada processo (wssi) no sistema então D=åwssi é o número total de páginas sendo requeridas pelos processos.*

Working Set

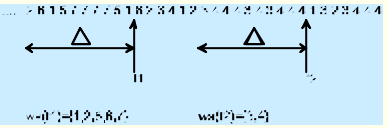
Se D>nº. de frames disponíveis nós começaremos a ter problemas (trashing) uma vez que alguns processos não poderão ser executados por falta de frames suficientes.



· O SO monitora o ws de cada processo e aloca para ele o número de páginas necessárias (wss) para completar o ws do processo. Se existem frames suficientes um novo processo pode ser iniciado. Se a soma dos ws's dos processos (D) aumenta ultrapassando o número de frames disponíveis, o SO escolhe um processo para suspender a sua execução.

Working Set

· As suas páginas são gravadas em disco (swapped out) e os seus frames são realocados para outros processos. O processo suspenso pode ser reescalonado mais tarde.

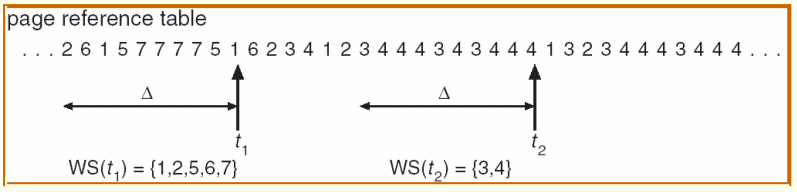


· A estratégia do ws serve para prevenir o trashing e elevar ao máximo possível o grau de multiprogramação, o que acarreta uma otimização no uso da UCP.

Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references.
- Example: 10,000 instruction
  - WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ (varies in time)
    - if Δ too small will not encompass entire locality
    - if Δ too large will encompass several localities
    - if $\Delta = \infty \Rightarrow$ will encompass entire program
  - $D = \Sigma\ WSS_i \equiv$ total demand frames
  - if $D > m \Rightarrow$ Thrashing
    - Policy if $D >$ m, then suspend one of the processes

Working-set model

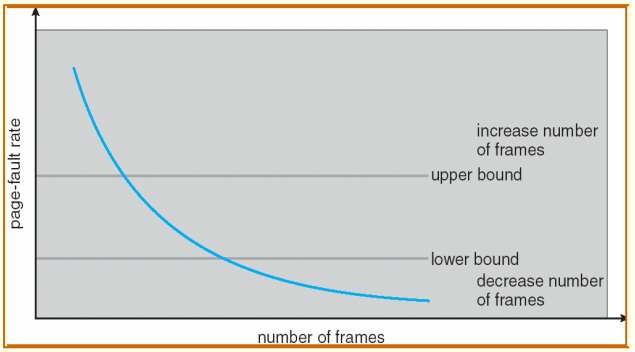


Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: Δ = 10,000
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units

Page-Fault Frequency Scheme

- Establish “acceptable” page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



Gerência de Memória
Memória Virtual

- **Trashing**
  - *Trashing pode ser definido como sendo a excessiva transferência de páginas/segmentos entre a MP e a MS, presentes em sistemas paginados ou segmentados (veja próximo item). Na paginação, o trashing pode ocorrer em dois níveis:* ***processo e sistema.***
  - ***processo*** *- elevado número de page-faults gerado pelo programa. Os principais motivos são:*

    (a) mau dimensionamento do tamanho do ws (wss) sendo pequeno demais para acomodar as páginas constantemente referenciadas.

    (b) programas mal-estruturados onde o princípio da localidade não se aplica.

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system
- **Thrashing** ≡ a process is busy swapping pages in and out

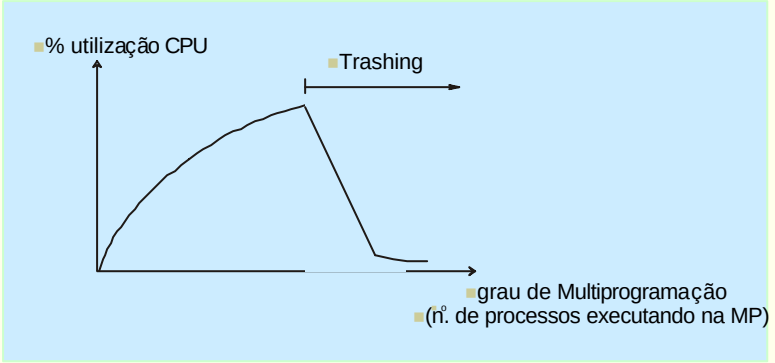
Thrashing

- **Trashing**
  - ***sistema*** *- quando existem mais processos competindo pela MP que espaço disponível. O SO tenta administrar a memória de forma que todos os processos sejam atendidos. O primeiro passo é a redução do ws dos processos, o que acarreta o problema de trashing em nível de processo. Caso a redução do ws não resolva, o SO começa o trabalho de swapping para liberar espaço na MP.*

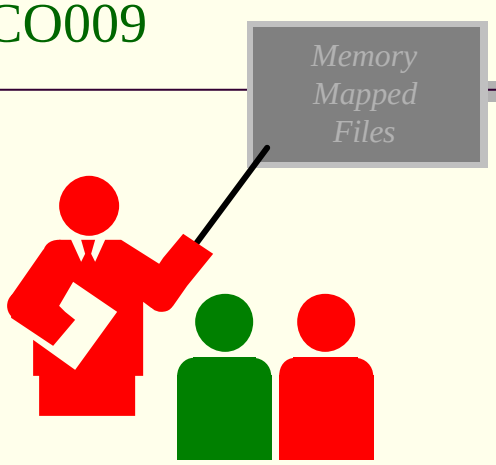
  - *O fenômeno de Trashing pode ser visualizado em um gráfico que indique o comportamento da UCP (%utilização de UCP) contra o número de processos executando em memória (Grau de Multiprogramação).*

Thrashing

- **Trashing**



SOP – CO009



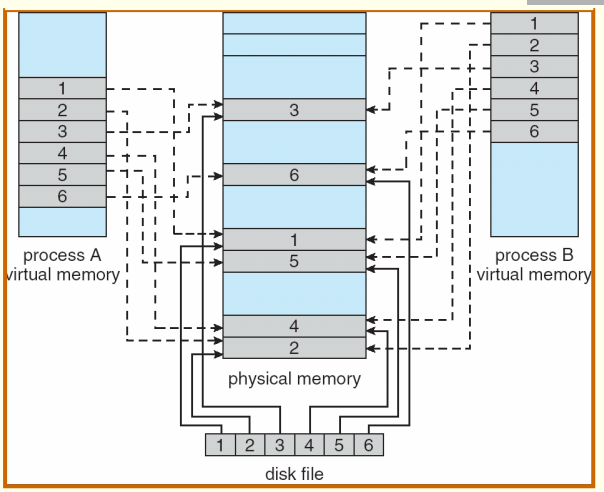
Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by **mapping** a disk block to a page in memory
- A file is initially read using demand paging. A page-sized portion of the file is read from the file system into a physical page. Subsequent reads/writes to/from the file are treated as ordinary memory accesses.

Memory-Mapped Files (cont.)

- Simplifies file access by treating file I/O through memory rather than **read() write()** system calls
- Also allows several processes to map the same file allowing the pages in memory to be shared

Memory Mapped Files



Memory-Mapped Files in Java

```
import java.io.*;
import java.nio.*;
import java.nio.channels.*;

public class MemoryMapReadOnly {
   // Assume the page size is 4 KB
   public static final int PAGE SIZE = 4096;

   public static void main(String args[]) throws IOException {
     RandomAccessFile inFile = new RandomAccessFile(args[0],"r");
     FileChannel in = inFile.getChannel();
     MappedByteBuffer mappedBuffer =
          in.map(FileChannel.MapMode.READ ONLY, 0, in.size());

     long numPages = in.size() / (long)PAGE SIZE;
     if (in.size() % PAGE SIZE > 0)
        ++numPages;
```

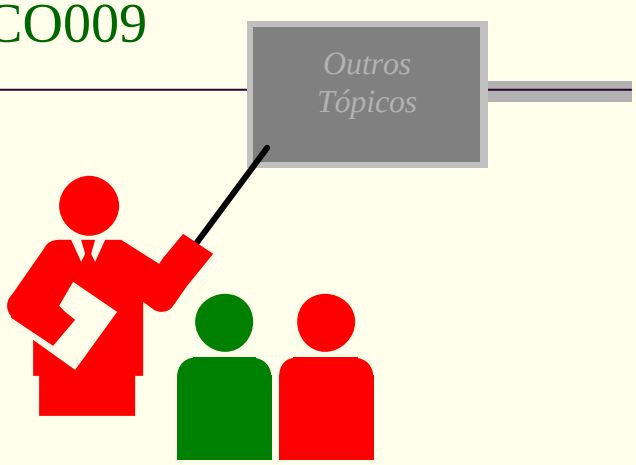

Memory-Mapped Files in Java (cont)

```
       // we will "touch" the first byte of every page
       int position = 0;
       for (long i = 0; i < numPages; i++) {
         byte item = mappedBuffer.get(position);
         position += PAGE SIZE;
       }
       in.close();
       inFile.close();
   }
}
```

- The API for the map() method is as follows:

map(mode, position, size)

SOP – CO009



Other Issues - Prepaging

- Prepaging
  - To reduce the large number of page faults that occurs at process startup
  - Prepage all or some of the pages a process will need, before they are referenced
  - But if prepaged pages are unused, I/O and memory was wasted
  - Assume *s* pages are prepaged and *a* of the pages is used
    - Is cost of *s*a* save pages faults > or < than the cost of prepaging *s * (1- a)* unnecessary pages?
    - *a* near zero ⇒ prepaging loses

Other Issues – Page Size

- Page size selection must take into consideration:
  - fragmentation
  - table size
  - I/O overhead
  - locality

Other Issues – TLB Reach

- TLB Reach - The amount of memory accessible from the TLB
- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB. Otherwise there is a high degree of page faults.
- Increase the Page Size. This may lead to an increase in fragmentation as not all applications require a large page size
- Provide Multiple Page Sizes. This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation.

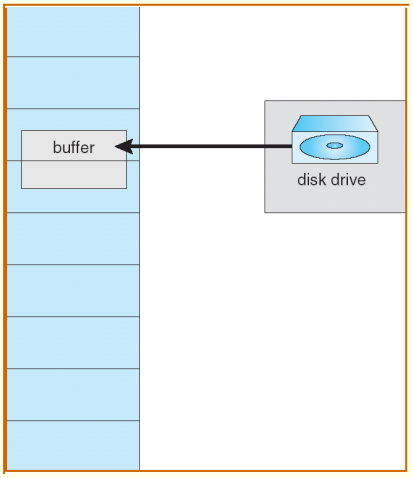
Other Issues – Program Structure

- Program structure
  - Int[128,128] data;
  - Each row is stored in one page
  - Program 1
```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```
128x128 = 16,384 page faults !
  - Program 2
```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```
128 page faults !

Other Issues – I/O interlock

- **I/O Interlock** – Pages must sometimes be locked into memory
- Consider I/O. Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm.

Reason Why Frames Used For I/O Must Be In Memory



Operating System Examples

- Windows XP
- Solaris

Windows XP

- 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

Windows XP (cont.)

- 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

Solaris

- Maintains a list of free pages to assign faulting processes
- *Lotsfree* – threshold parameter (amount of free memory) to begin paging
- *Desfree* – threshold parameter to increasing paging
- *Minfree* – threshold parameter to being swapping

Solaris

- Paging is performed by *pageout* process
- Pageout scans pages using modified clock algorithm
- *Scanrate* is the rate at which pages are scanned. This ranges from *slowscan* to *fastscan*
- Pageout is called more frequently depending upon the amount of free memory available

Solaris 2 Page Scanner