

Sistemas Operacionais

Memória virtual



2ª edição

Capítulo 7

Revisão: Fev/2003

Motivação

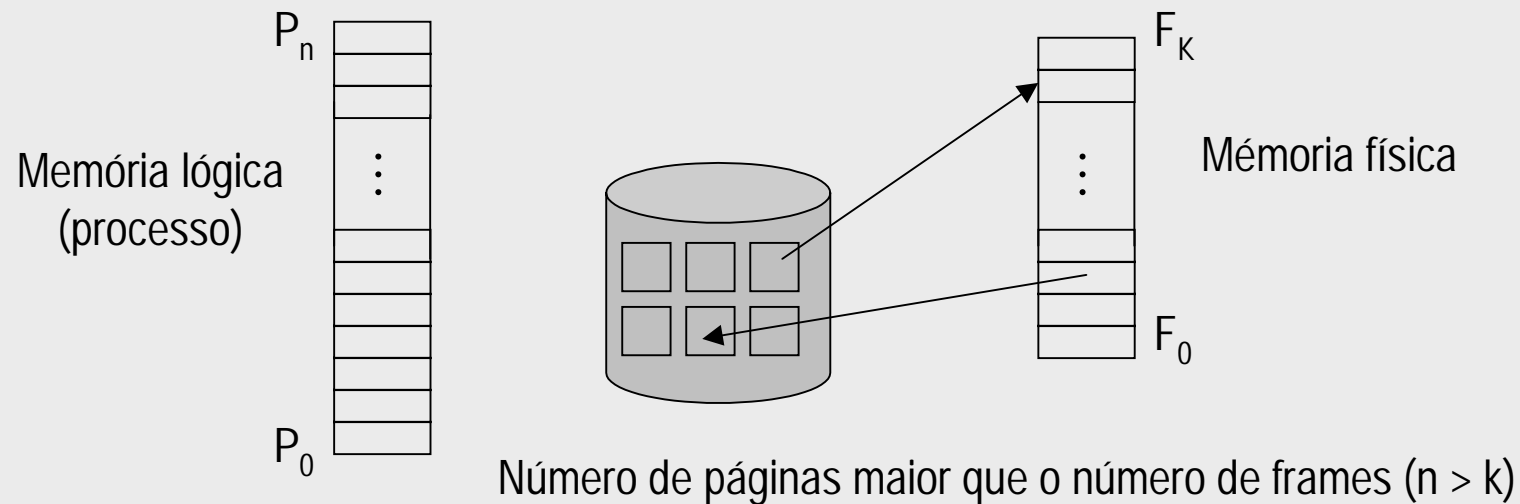
- Problemas da gerência de memória (modelo visto até o momento)
 - Todo o espaço lógico mapeado no espaço físico
 - O tamanho do programa é limitado pelo tamanho da memória
 - Desperdício de memória por manter armazenado código não utilizado frequentemente
 - e.g.; rotinas de tratamento de erro
- Um programa ocupando apenas um espaço de memória que realmente necessária permite uma economia substancial de espaço de memória
 - Espaço liberado permitiria a carga e execução simultânea de mais programas

Memória virtual: conceito

- Memória virtual é a técnica que permite a execução de um processo sem que ele esteja completamente em memória
 - Separação do vínculo de endereço lógico do endereço físico
- Princípios básicos:
 - Carregar uma página/segmento na memória principal apenas quando ela for necessária
 - Paginação por demanda
 - Segmentação por demanda
 - Manter em memória apenas as página/segmentos necessários

Memória virtual: vantagens

- Aumento do grau de multiprogramação
- Reduz o número de operações de E/S para carga/*swap* do programa
- Capacidade de executar programas maiores que a capacidade disponível de memória



Princípio da localidade

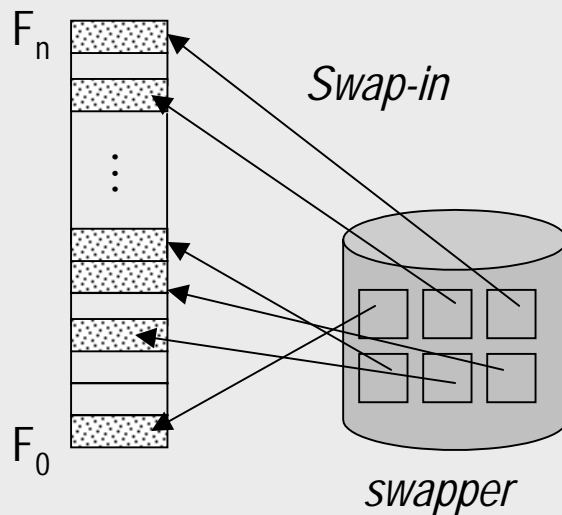
- Base de funcionamento da memória virtual
- Na execução de um processo existe uma probabilidade que acessos a instruções e a dados sejam limitados a um trecho
 - Exemplos:
 - Execução da instrução $i+1$ segue a execução da instrução i
 - Laços *for*, *while*, *do-while*
 - Acessos a elementos de vetores
- Em um determinado instante de tempo apenas esses trechos do processo necessitam estar em memória
- Possibilidade de “adivinhar” os trechos necessários a seqüência de execução

Necessidades para implementação de memória virtual

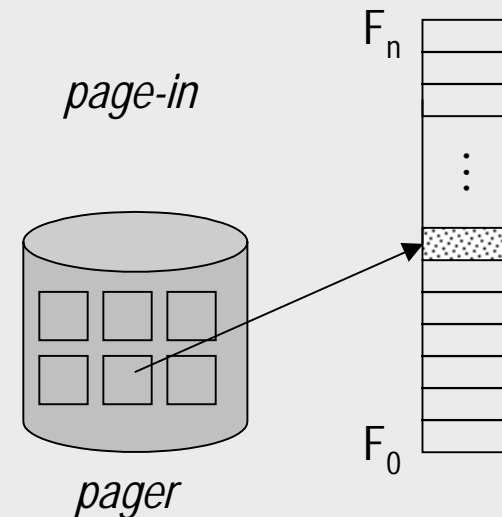
- Hardware deve suportar paginação e/ou segmentação
- Sistema operacional deve controlar o fluxo de páginas/segmentos entre a memória secundária (disco) e a memória principal
- Necessidade de gerenciar
 - Areas livres e ocupadas
 - Mapeamento da memória lógica em memória física
 - Substituição de páginas/segmentos

Paginação por demanda (1)

- Forma mais comum de implementação de memória virtual
- Similar a paginação com *swapping*
 - Invés de ser realizado o *swap-in/out* de um processo, se realiza o *swap-in/out* de apenas uma página do processo (*page-in/out*)



Paginação simples



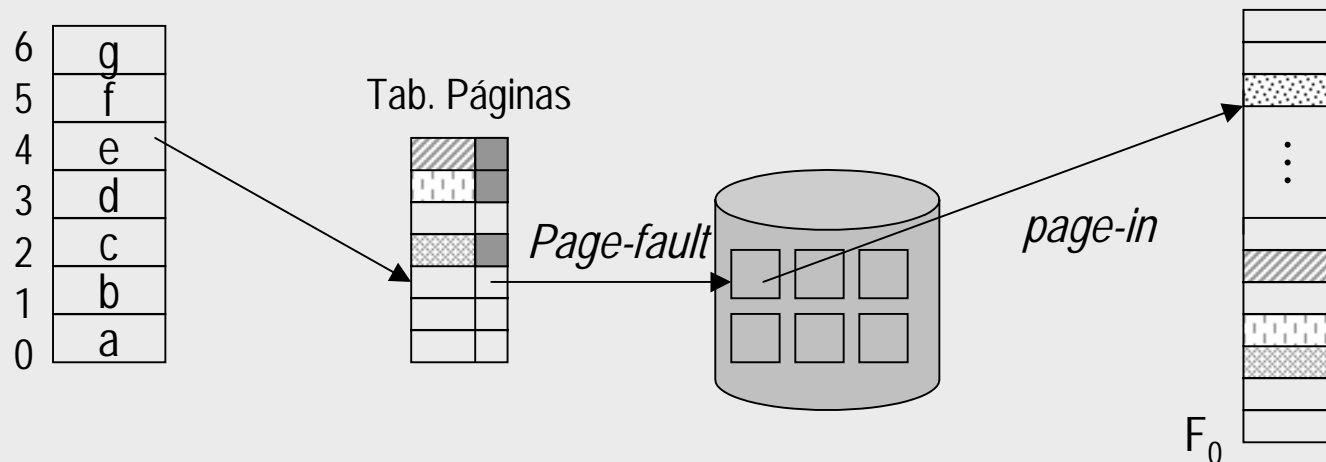
Paginação por demanda

Paginação por demanda (2)

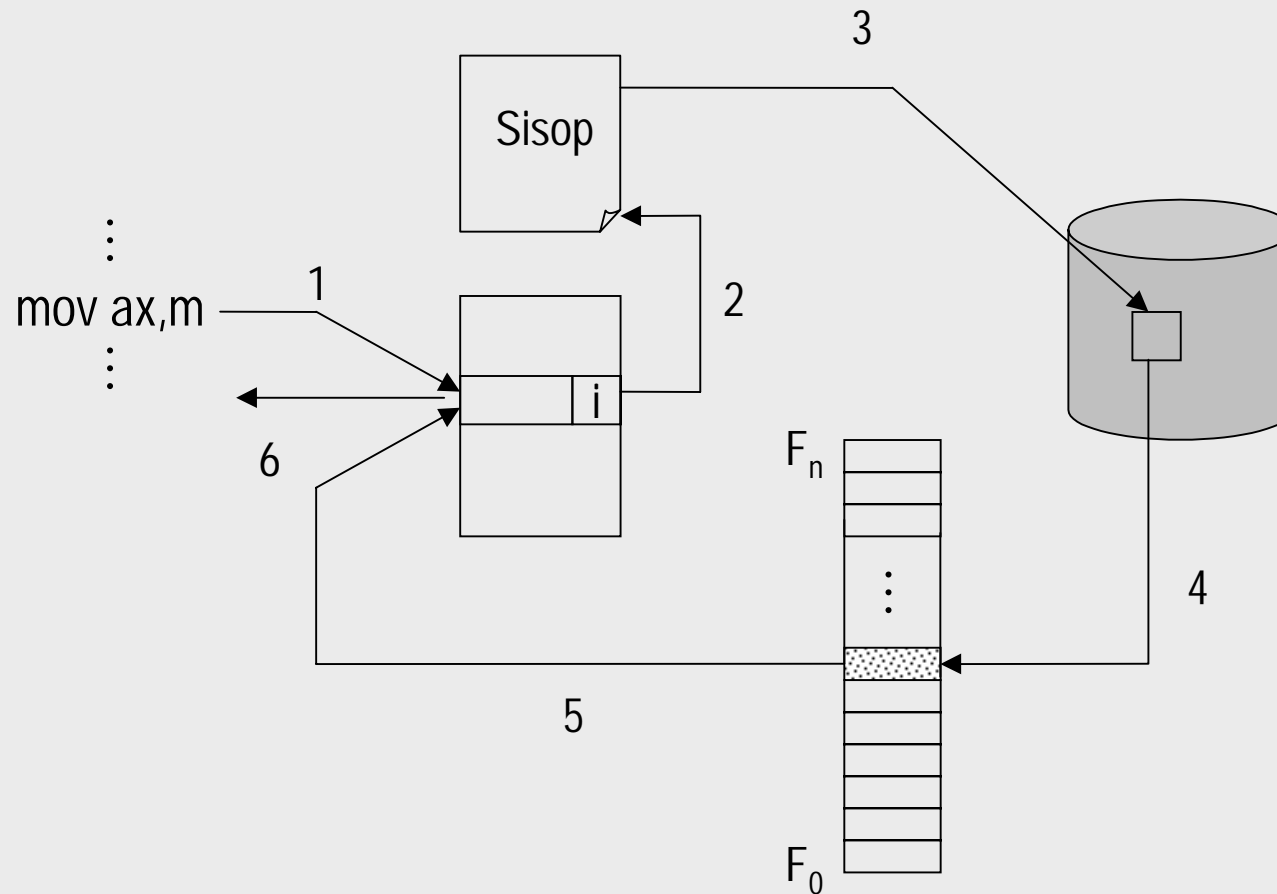
- Carrega uma página para a memória principal somente quando ela é necessária
- Vem de encontro com a filosofia de memória virtual
 - Reduz operações de E/S
 - Carrega apenas a(s) página(s) necessária(s)
 - Reduz quantidade de memória utilizada por processo
 - Aumenta grau de multiprogramação

Paginação por demanda (3)

- Uma referência está sempre associada a uma página
 - Inválida
 - Página presente na memória/não presente na memória (*page-fault*)
- Necessita conhecimento de quais páginas estão na memória secundária e quais páginas estão na memória principal
 - Bit válido/inválido associado a cada entrada na tabela de páginas



Tratamento de *page-fault* (1)



Tratamento de falta de página (*page-fault*) (2)

■ Passos realizados (visão do sistema)

1. *Trap* para sistema operacional
2. Salvamento de contexto
3. Detecção que a interrupção é por *page-fault*
4. Verifica se referência é válida e determina localização da página no disco
5. Solicita operação de leitura da página "faltante" do disco para o frame
6. Passa processo da página "faltante" para estado suspenso e escalona outro processo
7. Interrupção do disco (final de transferência da página)
8. Atualiza tabela de página
9. Passa processo do estado suspenso para estado pronto

Desempenho da paginação por demanda (1)

- Taxa de falta de páginas $\rightarrow 0 \leq p \leq 1.0$
 - $p = 0$; qualquer referência não gera falta de páginas
 - $p = 1$, qualquer referência provoca uma falta de página
- Tempo efetivo de acesso:

$$t_{efetivo} = (1 - p) \times t_{acesso} + p \times t_{page_fault}$$

Ex.: tempo acesso: 100 ns
tempo *page_fault*: 25 ms
 $t_e = (1-p) \times 100 + p \times 25000000$
 $t_e = 100 + 24999900 \times p \quad \rightarrow \quad p = 1/1000$
 $t_e \cong 25 \text{ us } (250 \times t_{acesso})$

Desempenho da paginação por demanda (2)

■ Exemplo:

- Objetivo de aumentar no máximo em 10% o tempo de acesso

$$110 \cong 100 + p \times 25000000$$

$$10 \cong 25000000p$$

$$p = 0.0000004 \text{ (1 } page\ fault \text{ a cada 250000 acessos)}$$

■ Como melhorar o desempenho?

- Manter a taxa de falta de páginas (*page fault*) pequena
- Acelerar procedimentos de leitura em disco
 - Procedimento de *swap* (*pager*)

Implementação da memória virtual (1)

- A memória virtual pode ser implementada por:
 - Paginação por demanda
 - Segmentação por demanda
 - Segmentação com paginação (recai em paginação)
- O princípio de funcionamento é o mesmo porém é mais simples de gerenciar paginação por demanda

Implementação da memória virtual (2)

- Memória física é um recursos limitado, é necessário então:
 - Política de carga de página
 - Política de localização da página
 - Política de substituição de página
- Partição de *swap*
 - Área específica do disco destinada a armazenar páginas (segmentos)
 - Organizado de forma diferente do sistema de arquivos para otimizar o acesso

Política de carga de páginas

- Carrega uma página para um frame
- Duas situações:
 - Frame livre: carrega página no frame
 - Não há frame disponível:
 - Libera espaço transferindo página(s) da memória (*pager-out*) para o disco (área de *swap*)
 - Política de substituição para seleção da página “vítima”
- Otimizações:
 - Carregar mais de uma página na memória
 - Nem toda página necessita *pager-out*:
 - Páginas não modificadas
 - Páginas *read-only* (código)

Política de localização

- Determina na memória real a localização das páginas de um processo
- Realizado pelo hardware de paginação/segmentação do processador
 - Transparente sob o ponto de vista do sistema operacional

Substituição de páginas na memória

- As páginas físicas (frames) podem ficar totalmente ocupada a medida que páginas lógicas de processos são carregadas
- Necessidade de liberar uma página física (frame) para atender à falta de página
- O algoritmo de substituição de páginas é responsável pela escolha de uma página "vítima"

Bits auxiliares

- Objetivo é auxiliar a implementação do mecanismo de substituição de páginas
 - Não são absolutamente necessários porém simplificam e tornam mais eficaz o mecanismo de substituição de páginas
- Bit de sujeira (*dirty bit*)
 - Indica quando uma página foi alterada durante a execução do processo
 - Se página não foi alterada não é necessário salvar seu conteúdo no disco
- Bit de referência (*reference bit*)
 - Indica se uma página foi acessada dentro de um intervalo de tempo
- Bit de tranca (*lock bit*)
 - Evita que uma página seja selecionada como "vítima"

Política de substituição de páginas (1)

- Utilizado quando não há mais frames livres
- Seleciona na memória uma página a ser substituída quando outra página necessita ser carregada na memória
 - Problema é determinar a página menos necessária
- Certas páginas que não devem ser substituídas podem ser "grampeadas" a frames (*frame locking*), e.g.;
 - Código e estruturas de dados do sistema operacional
 - Buffers de E/S

Política de substituição de páginas (2)

- Selecionar para substituição uma página que será referenciada dentro do maior intervalo de tempo (algoritmo ótimo)
 - Impossível de se ter conhecimento do “futuro”
 - Emprego de aproximações
- Algoritmos para substituição de páginas
 - *First-come-first-served* (FCFS)
 - *Least Recently Used* (LRU)
 - Baseado em contadores
- *String* de referência:
 - Método para avaliar algoritmos de substituição de página
 - Geração randômica de endereços

First-come-first-served (FCFS)

- Também denominado de *First-In, First-Out* (FIFO)
- Frames na memória principal são alocados a páginas na forma de um buffer circular
- Simples implementação
- Substitui a página que está a mais tempo na memória
- Desvantagem é que a página substituída pode ser necessária logo a seguir

Exemplo: *First-come-first-serverd* (FCFS)

- Características do sistema
 - Memória física composta por 3 frames
 - Processo composto por uma memória lógica de 8 páginas

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

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

15 Page-faults

Página 0 foi substituída e em seguida referenciada!!


Least Recently Used (LRU)

- Premissa básica é que páginas acessadas recentemente por um processo serão novamente acessadas em um futuro “próximo”
- Página a ser substituída é a página referenciada a mais tempo
 - Pelo princípio da localidade, esta página deve ser a de menor probabilidade de ser referenciada em futuro próximo
- Desvantagem:
 - Cada página deve possuir a “data” da última referência
 - Suporte raramente suportado pelas MMUs

Exemplo: *Least Recently Used* (LRU)

- Características do sistema
 - Memória física composta por 3 frames
 - Processo composto por uma memória lógica de 8 páginas

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



12 Page-faults

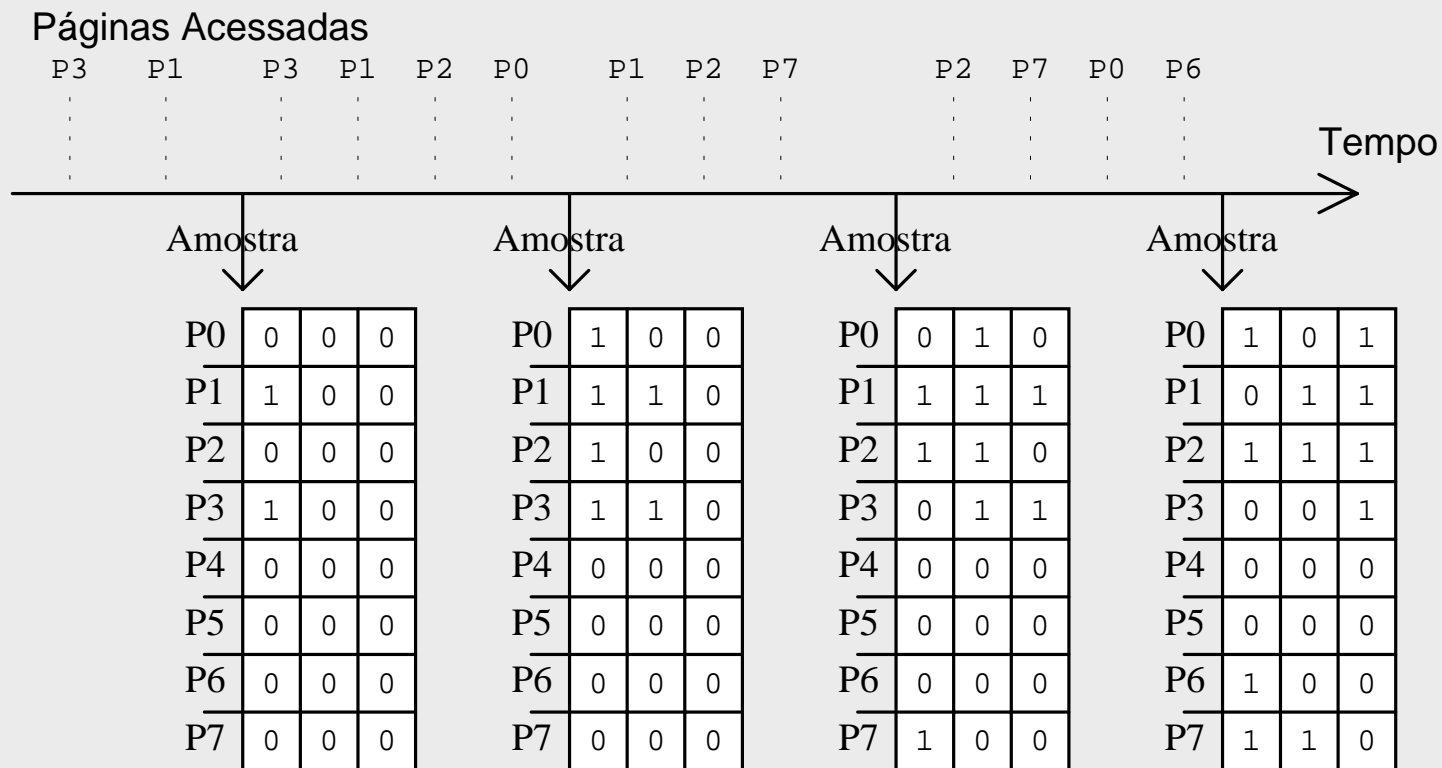
Implementação de *Least Recently Used* (LRU)

- Contadores na tabela de páginas
 - Contador
 - Tempo da última referência (*time stamp*)
 - Desvantagem é o *overhead*
 - Atualização da entrada na tabela de páginas
 - Seleção da página vítima
- Pilha
 - Página referenciada é inserida no topo da pilha
 - Topo da pilha está a página referenciada mais recentemente
 - Base da pilha está a página referenciada menos recentemente
 - Lista duplamente encadeada

Aproximações para algoritmo LRU

- Hardware (MMU) deve suportar bit(s) de referência
- Bit(s) de referência podem ser utilizados para aproximar LRU
 - Bit de referência = 1 (página acessada);
 - Bit de referência = 0 (página não acessada)
 - Substitui página cujo bit de referência é igual a zero
 - Desvantagem: não se sabe o “uso passado” e a ordem
- Melhoria:
 - Incluir vários bits de referência adicionais (registrador)
 - A cada Δt consulta o bit de referência e atualiza registrador (*shift register*)

Construção de histórico de bits de referência

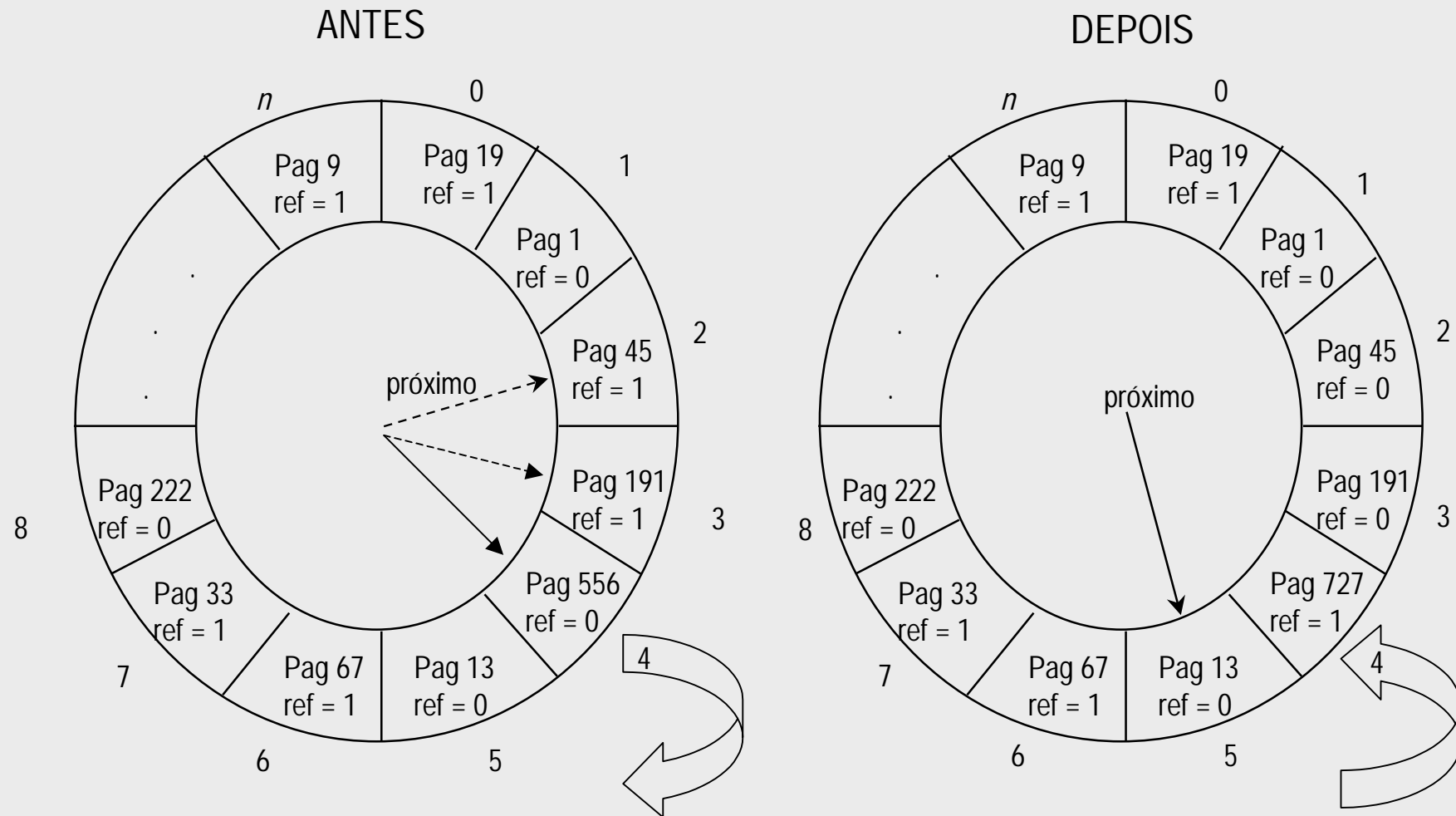


Histórico dos Bits de Referência após cada amostragem

Algoritmo de segunda chance (1)

- Denominado também de algoritmo do relógio
- Também baseado em bit de referência
- Considera que páginas lógicas formam uma lista circular
 - Apontador percorre a lista circular informando qual será a próxima “vítima”
 - Se página apontada (sentido horário) tem o bit de referência=1, então:
 - Posiciona bit de referência em zero e mantém página na memória
 - Substitui próxima página que tem bit de referência=0

Esquematização do algoritmo de segunda chance



Algoritmo de segunda chance melhorado

- Considera além do bit de referência um bit de modificação (bit uso)
 - bit uso=0: quando a página é carregado na memória
 - bit uso=1: quando a página é modificada
- Durante pesquisa por página a ser substituída modifica bit de referência de 1 para 0 (algoritmo do relógio)
 - Privilegia a substituição de frames com bit uso=0
 - 4 Combinações:
 - 00: não referenciada, nem modificada
 - 01: não referenciada recentemente, porém modificada
 - 10: recentemente referenciada, mas não modificada
 - 11: recentemente referenciada, modificada

Variação do algoritmo de segunda chance

- Variação destinada a MMUs que não implementam bit de referência
- Baseada em uma lista circular de páginas lógicas e um apontador (algoritmo relógio) e uma lista de páginas físicas livres
- Em caso de falta de página
 - Remove uma página física da lista livres para a qual será lida a página faltosa
 - Página “vítima” é marcada como inválida e incluída na lista de livres
 - Se essa página “vítima” for acessada logo em seguida ela simplesmente é retirada da lista de livres e inserida na lista de páginas lógicas válidas
- O tempo de permanência da página na lista de livres oferece a ela uma segunda chance de permanecer na memória

Algoritmos de substituição baseado em contadores

- A cada página é associado um contador de número de referências
- Duas políticas básicas:
 - *Least Frequently Used* (LFU)
 - Substitui a página que possui o menor valor
 - *Most Frequently Used* (MFU)
 - Não substitui a página que possui o menor valor
- Algoritmos não utilizados
 - overhead
 - Comportamento não aproxima algoritmo ótimo

Alocação de páginas físicas (frames)

- Questão:

- Como se deve alocar frames na presença de "n" processos ?

- Problemas:

- Qual o mínimo necessário por processo ?
 - Quantas páginas físicas alocar para cada processo ?
 - De onde alocar os frames ?

Número mínimo de frames

- Um processo necessita um número mínimo de frames para executar
 - Definido pelo conjunto de instruções da máquina (modos de endereçamento)
 - e.g;
 - INC MEM necessita duas páginas (código, dados)
 - MOV MEM,POS necessita três páginas (código, 2 de dados)
- Quanto menor o número de frames alocado a um processo maior a taxa de falta de páginas (*page fault*)
 - Queda de desempenho do sistema
 - Tentar manter o máximo de páginas em memória
- Número máximo de páginas mantidas em memória depende da capacidade física da máquina

Algoritmos de alocação (quanto alocar ?)

- O problema consiste em determinar quantos frames serão alocados para cada processo
- Dois algoritmos:
 - Alocação igualitária
 - Alocação proporcional

Alocação igualitária

- Princípio é dividir os m frames da memória física entre os n processos aptos a executar
 - Cada processo recebe m/n frames
 - A "sobra" pode compor um *pool* de frames livres
 - Número de frames é ajustado dinamicamente em função do grau de multiprogramação
- Desvantagem:
 - Provoca distorções já que processo possuem diferentes necessidades de memória

Alocação proporcional (1)

- Princípio é alocar frames (f) em função do tamanho do processo

$$f_i = \frac{s_i}{\sum_{j=1}^n s_j} \times m$$

s_i : memória virtual do processo p_i

n : número de processos em estado apto

m : número de frames

- A alocação deve ser reajustada dinamicamente em função do grau de multiprogramação

Alocação proporcional (2)

- Empregar a prioridade de um processo, invés de seu tamanho, como critério de “peso”
- Se um processo P_i provoca uma falta de página
 - Seleciona como “vítima” um de seus frames
 - Seleciona como “vítima” um frame de um processo de prioridade mais baixa

De onde alocar os frames ?

- Duas soluções possíveis:
 - Alocação local
 - Alocação global

Alocação local

- A gerência de memória define quantas páginas físicas cada processo deve dispor
- Em caso de falta de páginas a substituição ocorre entre as próprias páginas do processo que gerou a falta
- Desvantagens:
 - Definição do número de páginas físicas para cada processo
 - Impede que um processo utilize páginas físicas disponíveis pertencentes a outros processos

Alocação global

- Uma lista única de gerência de páginas físicas compartilhada por todos processos
- Um processo pode receber uma página física de outro processo
- Desvantagens:
 - O conjunto de páginas físicas ocupado por um processo depende do comportamento dos outros processos
 - Um processo de maior prioridade pode recuperar as páginas físicas de um processo de menor prioridade
- Apesar das desvantagens é o método mais comumente empregado
 - Privilegia o uso “inteligente” da memória física disponível da máquina

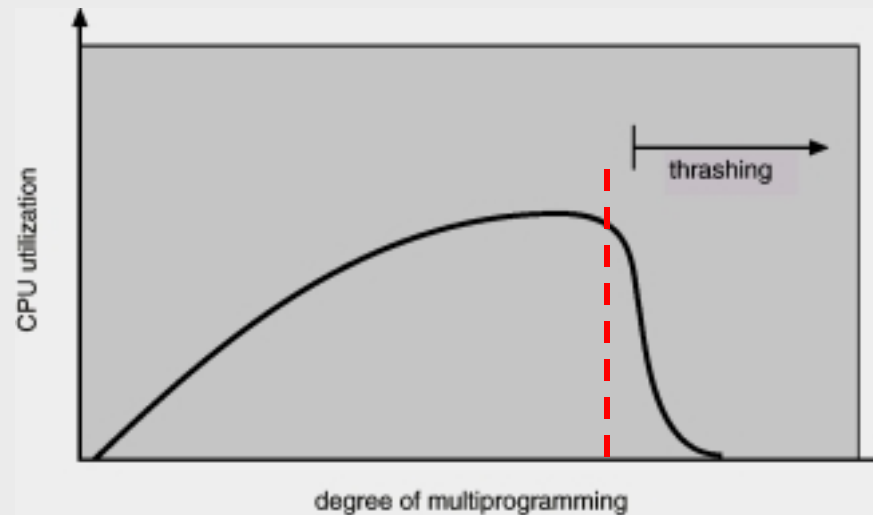
Thrashing

- Tratamento de uma falta de páginas é “caro” em tempo
 - Impacto de falta de páginas é grande para o desempenho do processo
- A medida que o número de páginas físicas alocadas a um processo a taxa de falta de páginas aumenta
- Um processo está em *thrashing* quando ele passa a maior parte do seu tempo de execução no processo de paginação
- Efeito “congelamento” diferente em função da política de alocação de páginas físicas
 - Alocação local implica no “congelamento” ao processo
 - Alocação global implica no “congelamento” do sistema

Conseqüências do *thrashing*

- Baixa taxa de uso da CPU para execução de processos de usuários
 - Sistema operacional pode “pensar” que está faltando processos aptos para execução e permite a criação/adição de novos processos
 - Escalonador de médio e longo prazo
- Adição de processos implica em maior necessidade de frames
 - Agrava a situação
- Conclusão:
 - Existe um ponto onde o grau de multiprogramação compromete o desempenho do sistema

Multiprogramação e *Thrashing*



- Para retirar um sistema do estado de *thrashing* é necessário suspender alguns processos temporariamente
- Mecanismo natural é o *swapping*
 - Não desejável por aumentar o tempo de resposta do processos
- Conclusão: "Mais vale prevenir que remediar !"

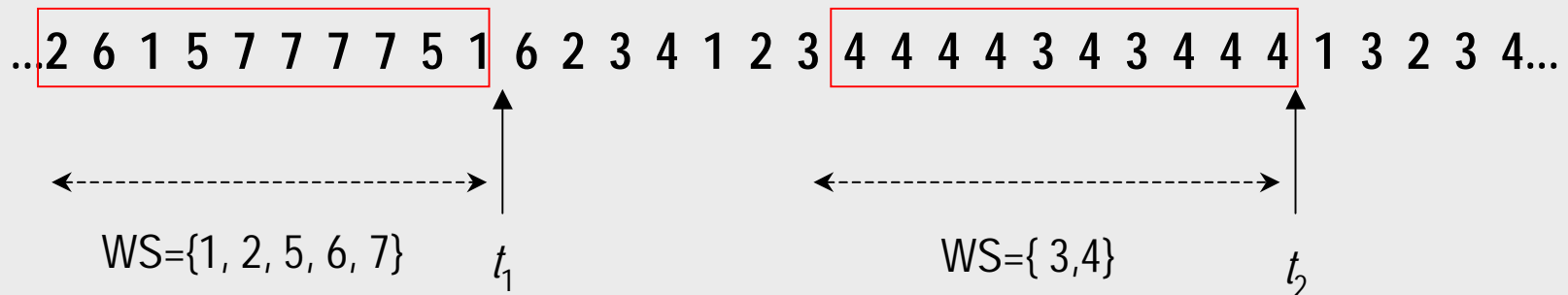
Prevenção do *thrashing*

- Memória virtual é baseada no princípio da localidade
- *Thrashing* sempre que:
 - Σ da memória necessária a localidade > memória física disponível (frames)
- Solução é providenciar os frames necessários a execução do processo. Duas abordagens:
 - Modelo de *working-set*
 - Método frequência de falta de página

Modelo de *working-set*

- Baseado no princípio da localidade
- Um parâmetro Δ define a largura da janela do *working-set*
- Princípio de base é examinar as últimas Δ referências a páginas
 - Se página está em uso: pertence ao *working set*
 - Se página não está em uso: eliminada do *working set* em Δ unidades

$\Delta = 10$ unidades



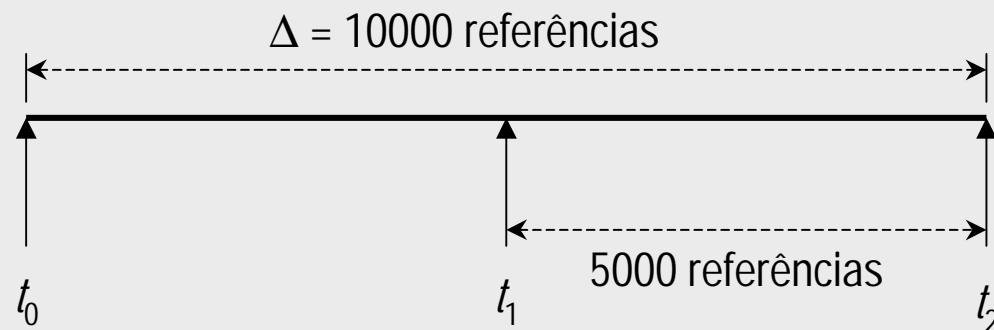
Definição e manutenção do *working-set* (1)

- Define-se WSS_i como sendo o *working-set size* do processo P_i
- Problema é definir o Δ :
 - Δ = valor pequeno: não abrange toda a localidade.
 - Δ = valor grande: abrange várias
 - $\Delta = \infty$: abrange todo o programa.
- $D = \sum WSS_i$ é a quantidade total de frames necessários no sistema
 - Se $D > m \Rightarrow \textit{Thrashing}$
 - Para evitar que $D > m$, o sistema operacional monitora o uso de frames e, em caso de necessidade, suspende processo(s)

Definição e manutenção do *working set* (2)

- Problema é calcular o *working set*
 - Aproximação é feita com interrupção de tempo mais bit(s) de referência
 - aumento da frequência de interrupção e do número de bits de referência melhora a precisão da aproximação

■ Exemplo:



Interrupção:

- copia bit de referência para memória
- zera bit de referência

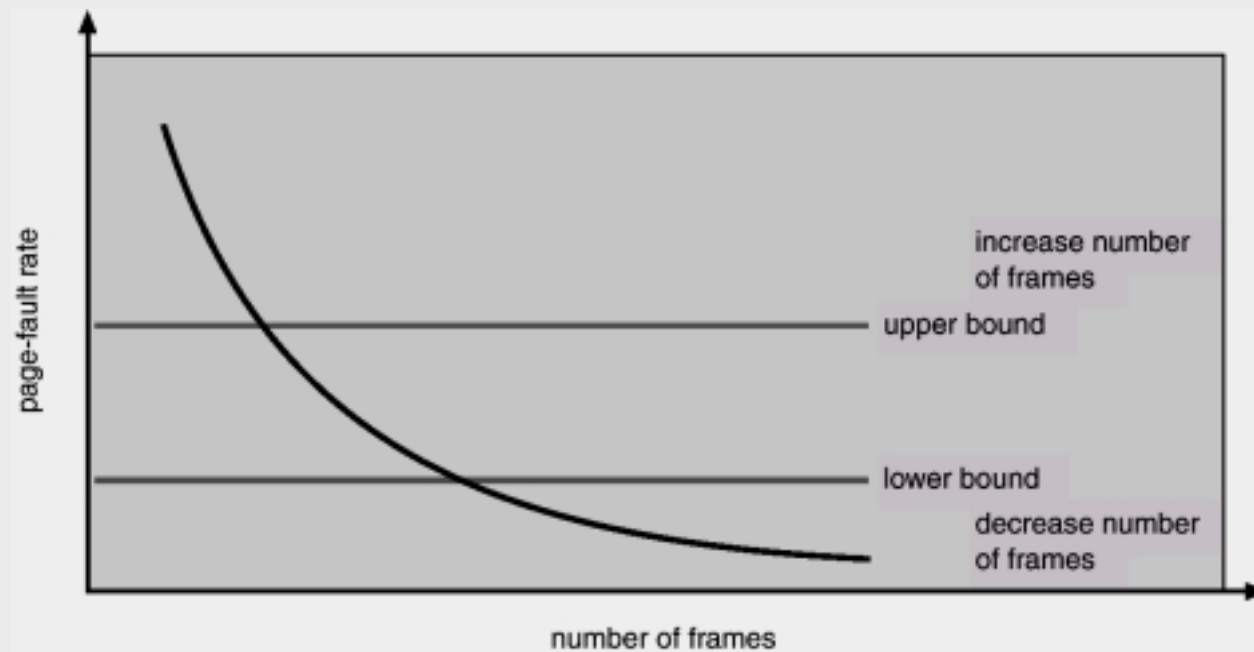
Histórico:

00:
01: }
10: } *working set*
11: }

Método frequência de falta de página

- Objetivo é controlar a taxa de faltas de páginas para manter dentro de um limite razoável
 - Taxa maior que o máximo aceitável:
 - Há processos que estão necessitando páginas físicas
 - Realiza o *swap-out* de alguns processos
 - Libera páginas físicas para processos que necessitam
 - Taxa menor que o valor mínimo:
 - Desligar o mecanismo de *swapping*
 - Processos tem páginas física demais alocadas
 - Pode liberar algumas

Método frequência de falta de página (2)



Fatores adicionais

- Além dos algoritmos de substituição de página e da política de alocação propriamente ditos existem outros fatores a serem considerados no projeto de um sistema de memória
 - Pré-paginação
 - Seleção do tamanho da página

Pré-paginação

- Consiste em trazer para a memória todo o *working-set* de um processo
 - Possível para caso em que processos que estão realizando transições dos estados bloqueado/suspenso para apto
- Custo da pré-paginação deve ser menor que custo de tratamento de falta de páginas
 - Qual fração (α) de páginas carregadas pela pré-paginação são efetivamente utilizadas?
 - Vale a pena: se o custo de trazer $(1 - \alpha)$ páginas é menor que o tratamento de falta de α páginas.

Tamanho da página (1)

- Fatores a serem considerados na definição do tamanho de página
 - Fragmentação
 - Tamanho de estruturas internas do sistema operacional (tabelas de páginas)
 - Overhead das operações de E/S
 - Localidade

Tamanho da página (2)

- Menor a página, menor a quantidade de fragmentação interna
- Menor a página, maior a quantidade de páginas por processo
- Mais páginas por processo, maior tabelas de páginas
- Maior a tabela de página, maior a necessidade de uso de memória

Tamanho da página (3)

- Menor é o tamanho da página, maior é o número de páginas que podem ser mantidas na memória principal
- Execução tende a deixar em memória apenas as páginas que são necessárias (localidade).
 - Com muitas páginas, tendência é reduzir a taxa de *page-fault*
- Página grande tende a desperdiçar a memória pois mantém além das referências necessárias, outras não utilizadas
 - Tendência a aumentar a taxa de *page-fault*

Estrutura do programa

- Exemplo

- Vetor A[1024, 1024] de inteiros
- Cada linha é armazenada em uma página

- Compilador re-organiza o código para evitar *page-faults*

- Estrutura de dados
- Pilha

- Programa 1 (1024 x 1024 *page faults*)

```
for j:= 1 to 1024 do  
  for i:= 1 to 1024 do  
    A[i,j]=0;
```

- Programa 2 (1024 *page faults*)

```
for i:= 1 to 1024 do  
  for j:= 1 to 1024 do  
    A[i,j]=0;
```

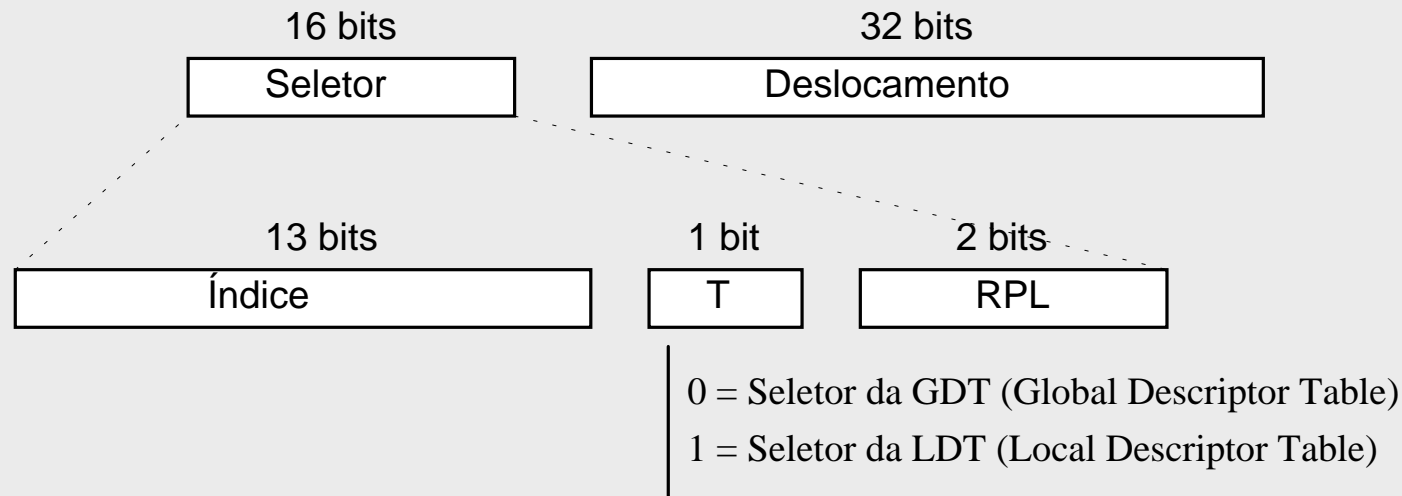
Estudo de caso: arquitetura Intel 386

- Esquema básico é o mesmo para a linha Pentium
- Arquitetura do 80386 oferece suporte à segmentação com paginação
- Capacidade de utilizar:
 - Segmentação pura
 - Paginação pura (2 níveis)

Endereço lógico arquitetura Intel

- Endereço lógico é formado por 48 bits divididos em:
 - Seletor (16 bits)
 - Indica qual segmento está sendo acessado
 - Deslocamento (32 bits)
 - Indica uma posição relativa dentro de um segmento
- Instruções de máquinas são em 32 bits
 - O valor do seletor é carregado em registradores específicos
 - Registradores de segmento (CS, DS, ES, SS, FS e GS)
 - Instruções consideram sempre como referência um registrador de segmento

Formato do endereço lógico no Intel 386



- Índice:
 - Aponta para uma entrada específica na LDT ou na GDT
- Bit T
 - Indica se acesso é na LDT ou na GDT
- Bits RPL
 - Associado ao mecanismo de proteção

Tradução de endereço lógico em físico

- Variação do procedimento estudado
- Endereço lógico é transformado em um endereço linear
- Interpretação do endereço linear depende se está sendo utilizado segmentação ou paginação
 - Com segmentação o endereço linear é o endereço físico utilizado para acessar uma posição na memória
 - Com paginação o endereço linear sofre um procedimento adicional de transformação em endereço físico

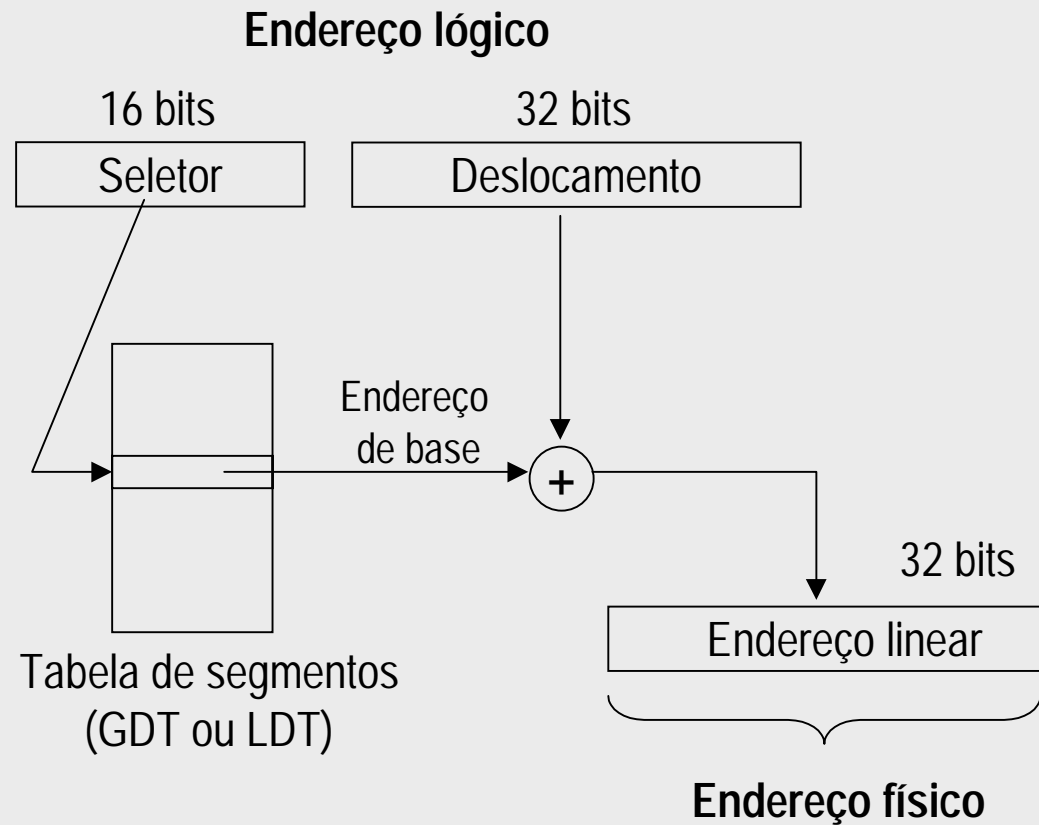
Segmentação no Intel 80386

- Forma nativa de operação
- Espaço lógico de endereçamento dividido em dois cada um contendo uma tabela de segmentos a parte
 - LDT: Local Descriptor Table
 - GDT: Global Descriptor Table
- Cada tabela de segmentos possui até 8K entradas, cada uma contendo um descritor de segmento com 8 bytes
 - Tabela de segmento ocupa no máximo 64 Kbytes

Formato do descritor de segmento

- Endereço de base do segmento (32 bits)
- Limite do segmento (20 bits)
 - Interpretação desse valor depende da granularidade
 - Granularidade=0: valor é expresso em bytes
 - Tamanho máximo de um segmento é 1 Megabyte
 - Granularidade=1: valor é expresso em páginas de 4 Kbytes
 - Tamanho máximo de um segmento é 4 Gbytes (1 Mega páginas)
- Controle de acesso (8 bits)
- Bit de granularidade (1 bit)
- Reservado para uso futuro (3 bits)

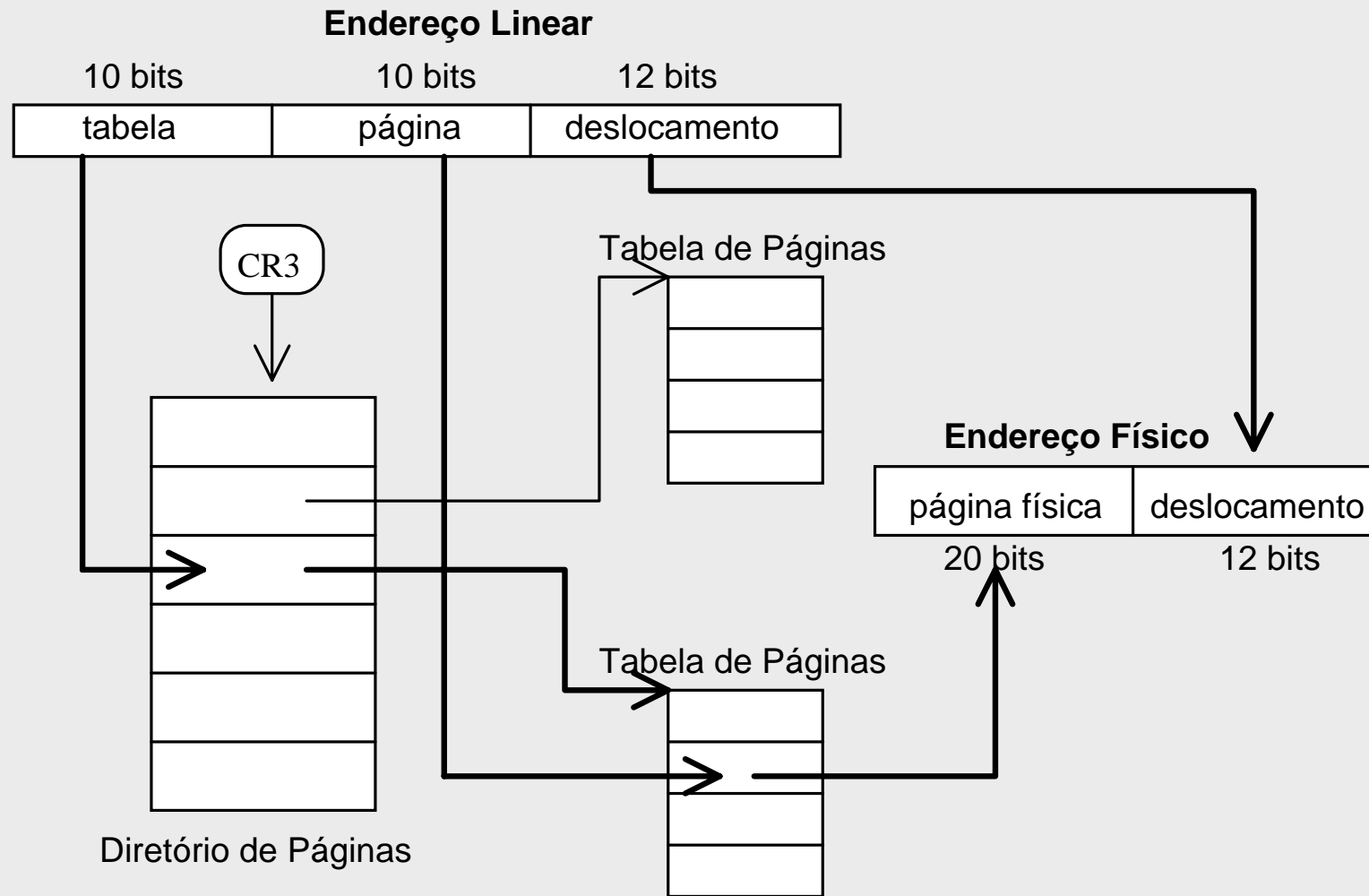
Esquema de segmentação no 80386



Paginação no Intel 80386

- Uso opcional no 80386
 - Desativada no momento do *reset* do processador
- Características gerais:
 - Páginas de 4 Kbytes
 - Espaço de endereçamento linear como o espaço de endereçamento físico possuem 4 Gbytes (2^{32})
 - Máximo 1 Mega páginas
 - Tabela de páginas considera dois níveis
 - Registrador (CR3) aponta para o diretório de tabela de páginas
 - Diretório de tabela de páginas
 - 1 k entradas de 4 bytes
 - Cada entrada aponta para uma tabela de páginas

Esquema de paginação no 80386



Leituras complementares

- R. Oliveira, A. Carissimi, S. Toscani; Sistemas Operacionais. Editora Sagra-Luzzato, 2001.
 - Capítulo 7
- A. Silberchatz, P. Galvin, G. Gagne; Applied Operating System Concepts. Addison-Wesley, 2000, (1st edition).
 - Capítulo 10
- W. Stallings; Operating Systems. (4th edition). Prentice Hall, 2001.
 - Capítulo 8