

Sistemas Operacionais

Prof. Rafael Obelheiro
rafael.obelheiro@udesc.br



Gerência de Memória

Introdução

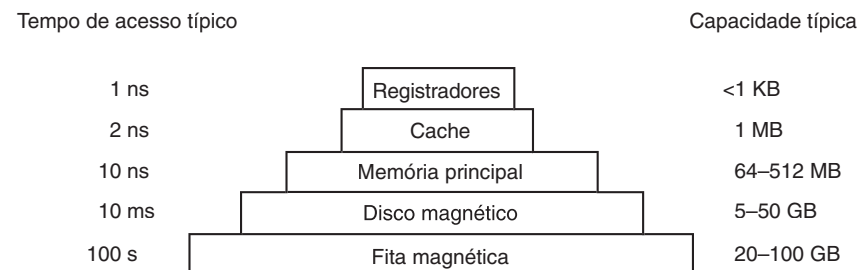
- A maioria dos sistemas atuais é multiprogramada
 - ▶ vários processos concorrem pelo processador
 - ▶ para serem executados, os processos precisam estar residentes na memória
- Para ter eficiência na multiprogramação, é necessário gerenciar a memória de forma adequada
 - ▶ a gerência de memória influi no grau de multiprogramação atingível em um sistema, e por consequência no seu desempenho global
- A gerência de memória atinge importância maior na medida em que as aplicações aumentam o seu tamanho

Sumário

- 1 Conceitos básicos
- 2 Gerenciamento sem abstração de memória
- 3 Gerenciamento com espaços de endereçamento
- 4 Memória virtual
- 5 Paginação
- 6 Paginação: questões de projeto e implementação
- 7 Segmentação

Hierarquia de memória

- Um sistema típico possui várias memórias, diferindo em capacidade, tempo de acesso e custo
- A gerência de memória em um SO envolve memória principal e disco
 - ▶ sem o auxílio do disco
 - ▶ com o auxílio do disco: swapping e memória virtual

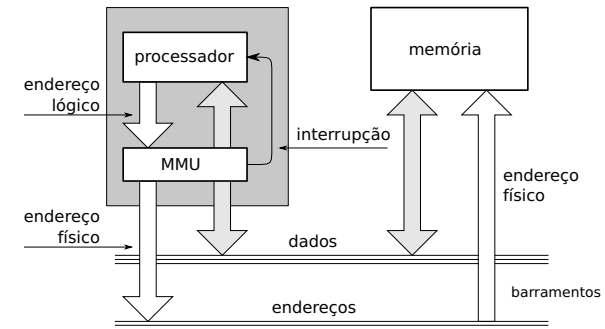


Memória lógica vs memória física (1)

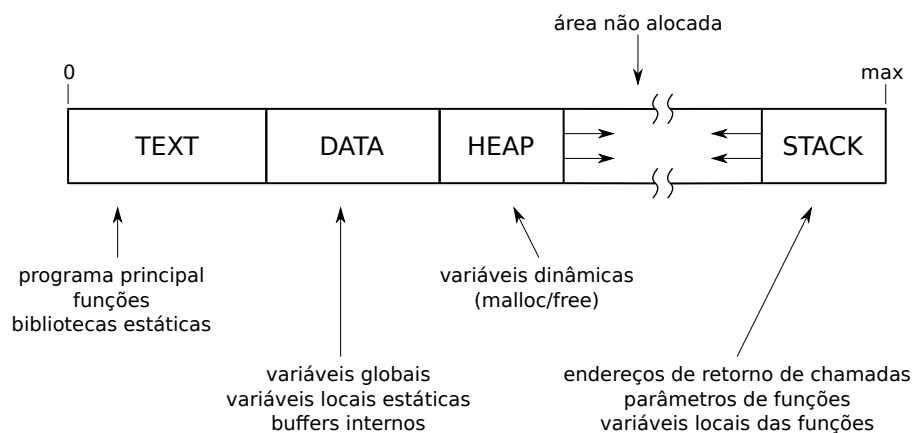
- Do ponto de vista do programador, a memória é um vetor de bytes endereçados individualmente
 - o tamanho máximo da memória é determinado pela largura do endereço em bits
 - b bits de endereço geram 2^b endereços diferentes ($0 \dots 2^b - 1$)
 $\Rightarrow 2^b$ bytes de memória
- Memória lógica: visão que um processo tem da memória
 - os endereços gerados por um processo são endereços lógicos
 - pertencentes à memória lógica
- Memória física: implementada pelos chips de memória
 - os endereços físicos correspondem a posições reais na memória
- Podem ser iguais ou diferentes
 - quando são diferentes, o mapeamento é feito em hardware com assistência do SO

Memória lógica vs memória física (2)

- O mapeamento entre endereços físicos e lógicos é realizado pela MMU (*Memory Management Unit*)
- O SO carrega tabelas de tradução na MMU



Layout típico da memória lógica

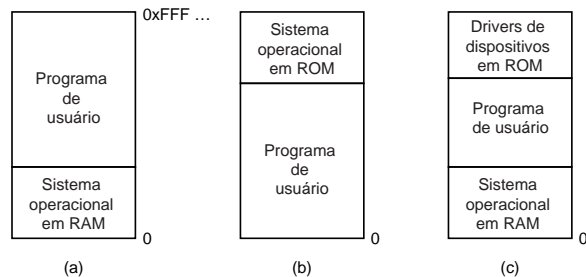


Sumário

- 1 Conceitos básicos
- 2 Gerenciamento sem abstração de memória
- 3 Gerenciamento com espaços de endereçamento
- 4 Memória virtual
- 5 Paginação
- 6 Paginação: questões de projeto e implementação
- 7 Segmentação

Gerenciamento sem abstração de memória

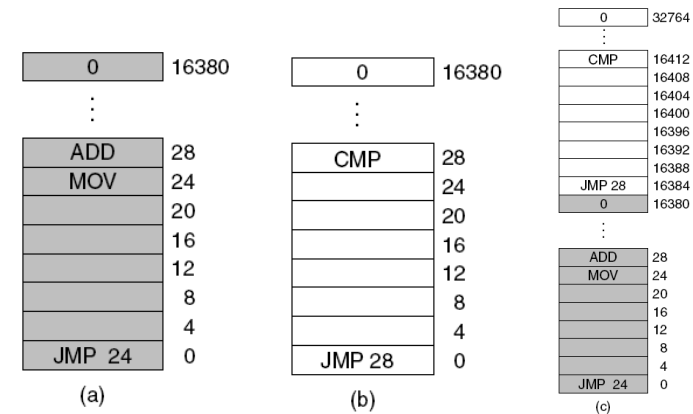
- Memória lógica = memória física
 - **endereçamento absoluto**
- Geralmente apenas um processo na memória por vez
 - **monoprogramação**
- Mecanismo **sem troca de processos** ou **paginação**
- Três maneiras simples de organizar a memória
 - um sistema operacional e um processo de usuário



© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 9/98

Dificuldades com multiprogramação (1)

- Dois programas de 16 KB (a) e (b) são carregados em regiões adjacentes na memória (c)
 - quando (b) começa a executar, desvia para um endereço em (a)



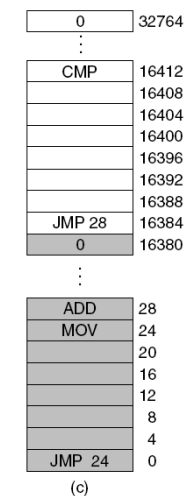
© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 10/98

Dificuldades com multiprogramação (2)

- Multiprogramação demanda proteção e relocação de memória
- IBM 360: memória dividida em blocos de 2 KB
 - cada bloco possuía uma chave de 4 bits, armazenada em um registrador específico
 - a PSW continha a chave do processo em execução
 - ★ um processo que tentava acessar um bloco de memória com uma chave diferente era abortado → **proteção**
- Relocação precisava ser feita durante a carga → **relocação estática**
 - quando um processo era carregado na memória, todas as referências a endereços eram substituídas pelos endereços efetivamente usados
 - ★ era preciso saber quais bytes no executável eram endereços e quais eram constantes

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 11/98

Dificuldades com multiprogramação (3)



- jmp 28 seria relocado para jmp 16412
- mov \$28, %eax não seria alterado

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 12/98

Sumário

- 1 Conceitos básicos
- 2 Gerenciamento sem abstração de memória
- 3 Gerenciamento com espaços de endereçamento
- 4 Memória virtual
- 5 Paginação
- 6 Paginação: questões de projeto e implementação
- 7 Segmentação

Registradores de base e limite (1)

- Mecanismo simples para implementar proteção e relocação
- Um par de registradores, que só podem ser manipulados pelo SO
 - base = início do processo na memória física
 - limite = tamanho do processo na memória física
- Princípio de funcionamento

se $EL < \text{limite}$
 $EF \leftarrow \text{base} + EL$ /* relocação */
senão
 aborte /* proteção */

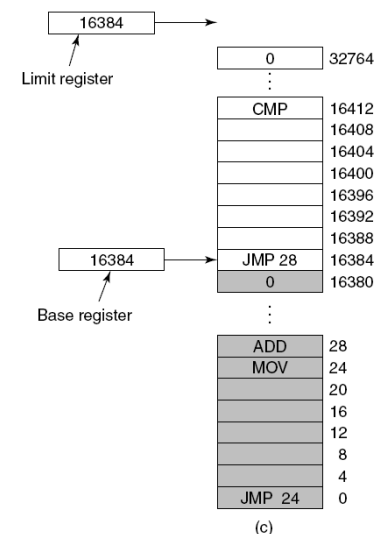
- Cada referência à memória exige uma adição e uma comparação

Espaços de endereçamento

- Endereçamento absoluto tem uma série de desvantagens
 - se não houver proteção, um processo pode corromper o SO
 - complica multiprogramação
- Uma solução melhor é recorrer à abstração de **espaço de endereçamento**
 - conjunto de endereços de memória que um processo pode usar
- Os programas são escritos considerando espaços de endereçamento privados, a menos que haja compartilhamento explícito
 - necessidade de proteção e relocação

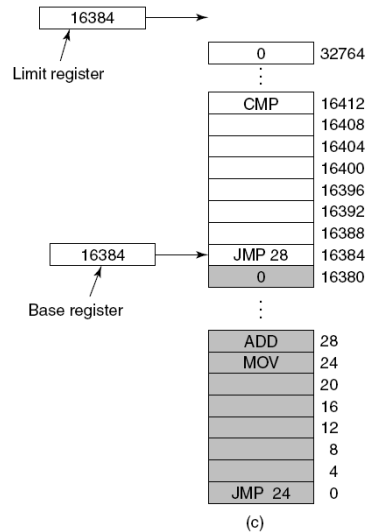
Registradores de base e limite (2)

- (a) base = 0, limite = 16384
- (b) base = 16384, limite = 16384
- (c) se um programa de 8 KB fosse carregado logo após o prog. (b)?
base = ?
limite = ?



Registradores de base e limite (2)

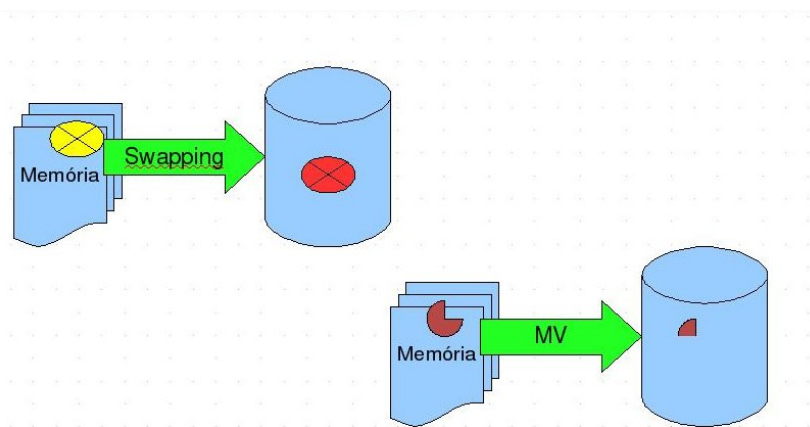
- (a) base = 0,
limite = 16384
- (b) base = 16384,
limite = 16384
- (c) se um programa de 8
KB fosse carregado
logo após o prog. (b)?
base = **32768**
limite = **8192**



Swapping vs memória virtual (1)

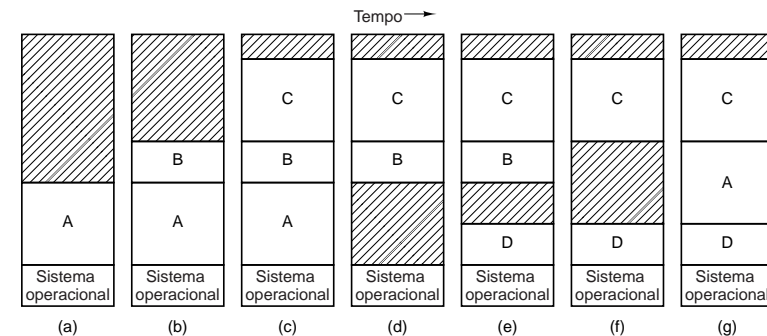
- Nem sempre há memória física suficiente para acomodar todos os processos ativos
- Uma solução é manter parte desses processos em disco
- Existem dois métodos básicos
 - swapping (troca de processos)**: processos inteiros são trazidos da memória para o disco e vice-versa
 - memória virtual**: os processos ativos estão parte na memória principal e parte no disco
 - particularmente útil se existem trechos de memória que não são efetivamente usados

Swapping vs memória virtual (2)



Exemplo de swapping

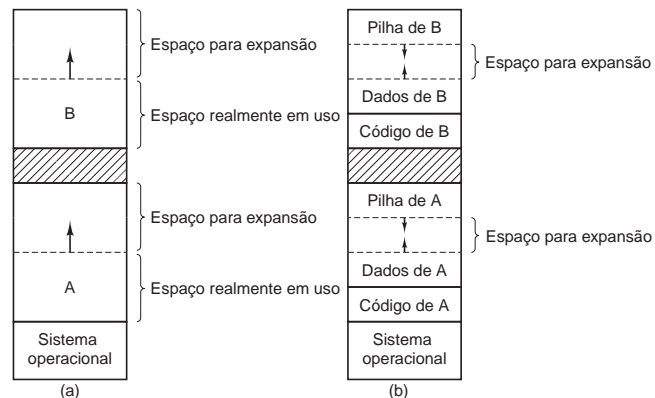
- Swapping com quatro processos A, B, C e D



áreas hachuradas = áreas livres (lacunas)

Dimensionamento de memória

- Normalmente é prudente alocar áreas de memória com uma certa folga para permitir alocação dinâmica
 - áreas de heap e pilha



© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 21/98

Problemas com swapping

- Como nem todos os processos usam a mesma quantidade de memória, ao longo do tempo ocorre **fragmentação externa**
 - grande número de pequenas lacunas que não podem ser usadas para alocar processos
 - uma solução é a **compactação de memória**
 - deslocar os processos e combinar as lacunas
 - overhead é grande
- Caso um processo precise alocar mais memória do que já possui, pode ser necessário movê-lo para outra região na memória física
 - se não houver uma lacuna grande o suficiente, outros processos também terão de ser movidos
- Se um processo usa apenas uma fração do seu espaço de endereçamento, muito tempo é gasto com transferências desnecessárias entre disco e memória

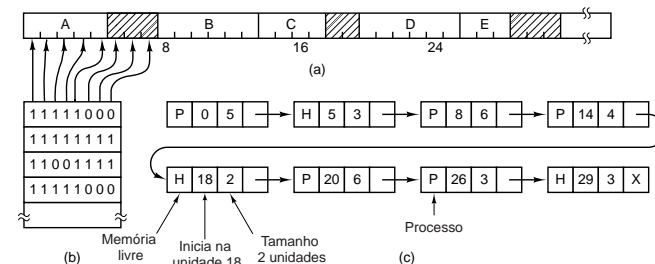
© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 22/98

Gerência de espaço livre (1)

- O SO precisa controlar quais áreas da memória física estão ocupadas e quais estão livres
- Existem duas maneiras básicas de fazer isso
 - usando mapas de bits
 - problema: busca de 0s consecutivos no mapa para encontrar uma área disponível
 - usando listas encadeadas
 - ordenada por endereços de memória (atualização rápida e simples)

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 23/98

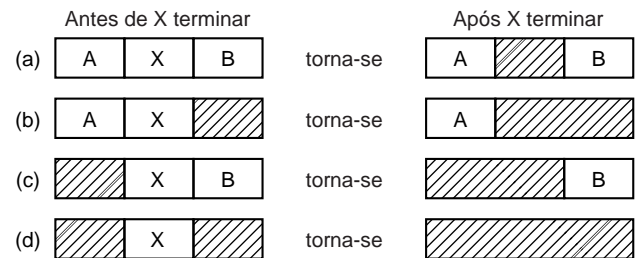
Gerência de espaço livre (2)



- (a) Parte da memória com 5 segmentos de processos e 3 segmentos de memória livre
 - riscos simétricos denotam as unidades de alocação
 - regiões sombreadas denotam segmentos livres
- (b) Mapa de bits correspondente
- (c) Mesmas informações em uma lista encadeada

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 24/98

Gerência com listas encadeadas

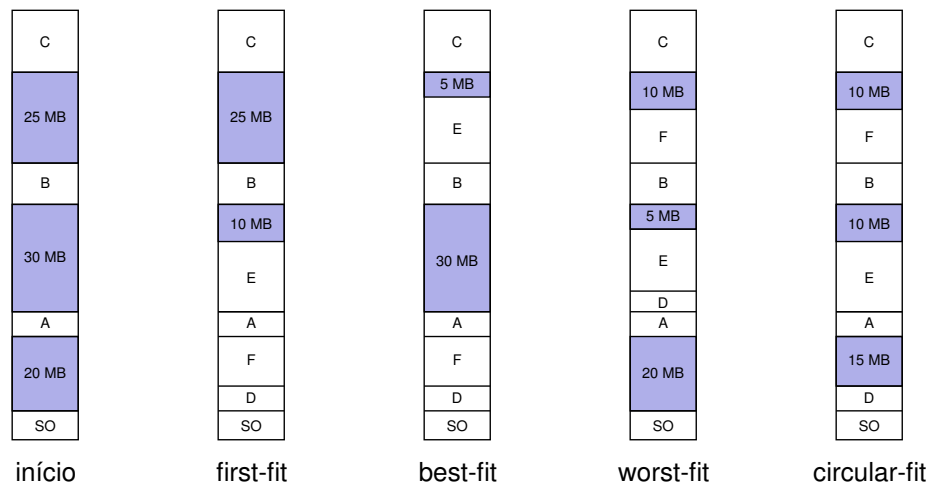


- Listas encadeadas de processos (P) e lacunas (H)
- Quando um processo termina, lacunas vizinhas são combinadas

Algoritmos para alocação de memória contígua (1)

- Como escolher onde alocar uma área contígua de memória?
- **First fit:** a primeira lacuna disponível é usada
- **Best fit:** a lacuna que melhor se ajusta é usada
 - tende a deixar lacunas muito pequenas
- **Worst fit:** a maior lacuna é usada
 - gera lacunas maiores, mas prejudica processos grandes
- **Next fit (circular fit):** como o first fit, mas guarda a última posição na lista de lacunas
- **Quick fit:** mantém listas de tamanhos comuns de áreas de memória
 - pode ser dispendioso descobrir quais são os segmentos de memória disponíveis para poder concatená-los

Algoritmos para alocação de memória contígua (2)



novos processos: D (5 MB), E (20 MB), F (15 MB)

Problemas de alocação contígua

- Um dos problemas da alocação contígua de memória é a fragmentação
 - pode haver **memória livre indisponível** para processos
- Um segundo problema é quando os programas usados extrapolam a memória física disponível
 - hoje em dia, existem programas enormes com muitas funcionalidades intocadas durante boa parte do tempo
 - pode afetar processos individuais ou o seu conjunto
 - problema é agravado pela fragmentação

Sumário

- 1 Conceitos básicos
- 2 Gerenciamento sem abstração de memória
- 3 Gerenciamento com espaços de endereçamento
- 4 Memória virtual
- 5 Paginação
- 6 Paginação: questões de projeto e implementação
- 7 Segmentação

Variações de memória virtual

- Paginação
- Segmentação
- Segmentação paginada

Memória virtual

- Como lidar com programas maiores do que a memória física disponível?
- A primeira solução foi a introdução de **overlays**
 - ▶ o programador dividia o programa em módulos que eram carregados e descarregados da memória semi-manualmente
- A solução mais definitiva veio com a memória virtual
 - ▶ espaço de endereçamento lógico (dos processos) é mapeado em um espaço de endereçamento físico
 - ▶ mapeamento permite usar regiões não contíguas na memória física
 - ▶ nem todo o espaço de endereçamento lógico precisa estar mapeado na memória física em um dado instante
 - ★ apenas as partes efetivamente usadas são mantidas na memória física, as demais são mantidas no disco e carregadas quando necessário

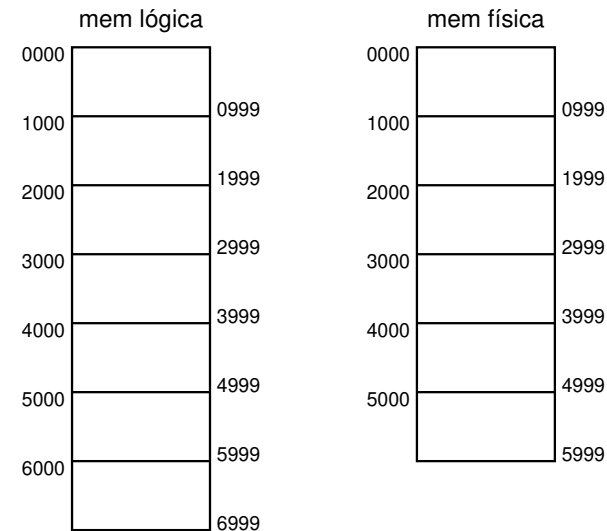
Sumário

- 1 Conceitos básicos
- 2 Gerenciamento sem abstração de memória
- 3 Gerenciamento com espaços de endereçamento
- 4 Memória virtual
- 5 Paginação
- 6 Paginação: questões de projeto e implementação
- 7 Segmentação

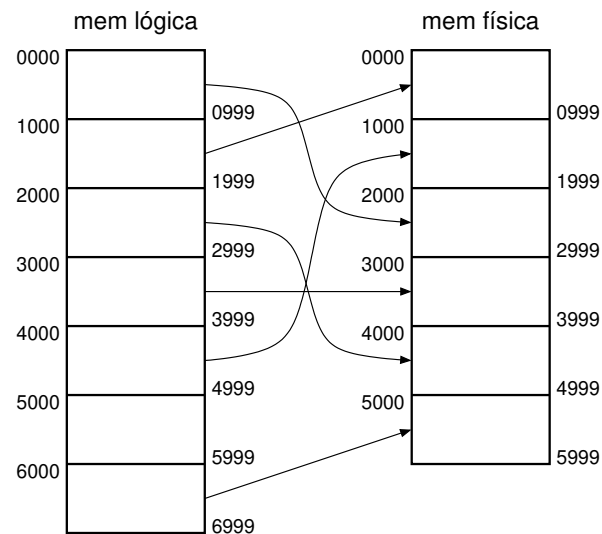
Memória virtual com paginação

- A memória é dividida em blocos de tamanho fixo
 - memória lógica: páginas [virtuais | lógicas]
 - memória física: molduras de página (*page frames*) ou páginas físicas
 - páginas têm tamanhos idênticos
- A paginação elimina a fragmentação externa e reduz a fragmentação interna

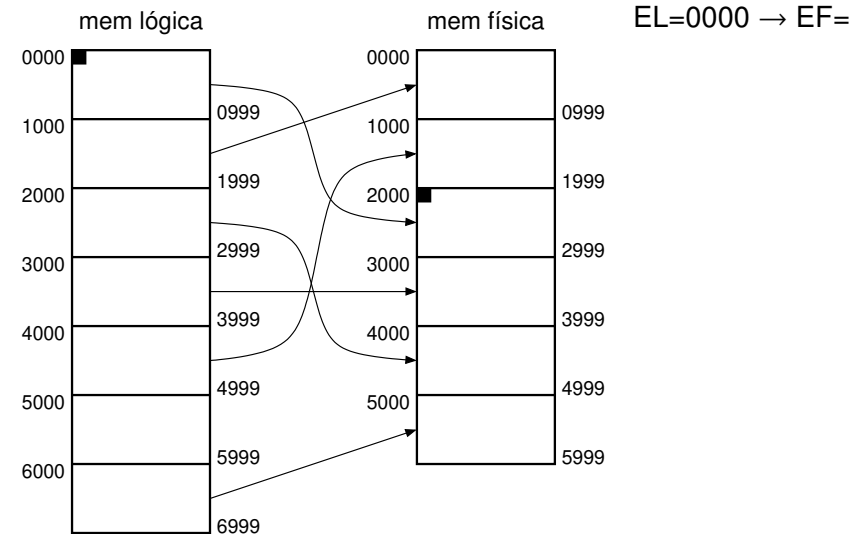
Paginação decimal



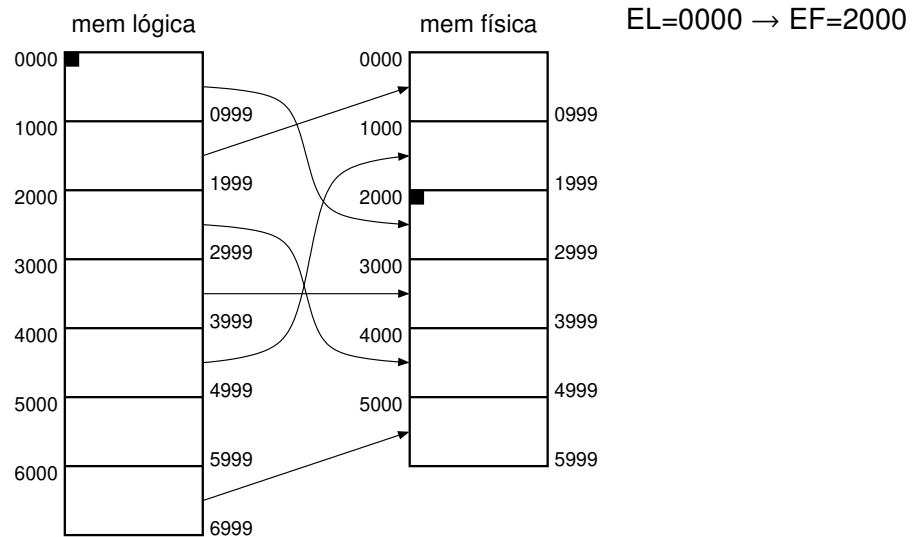
Paginação decimal



Paginação decimal

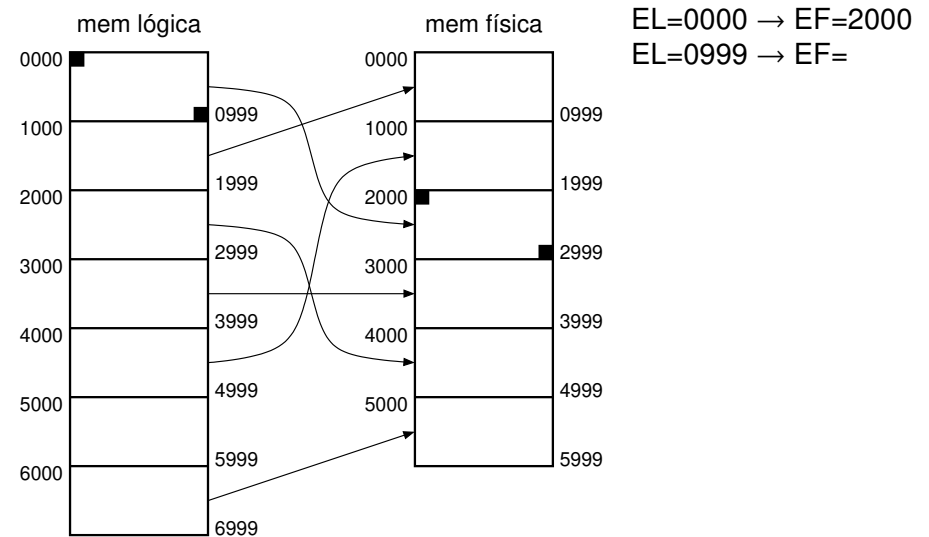


Paginação decimal



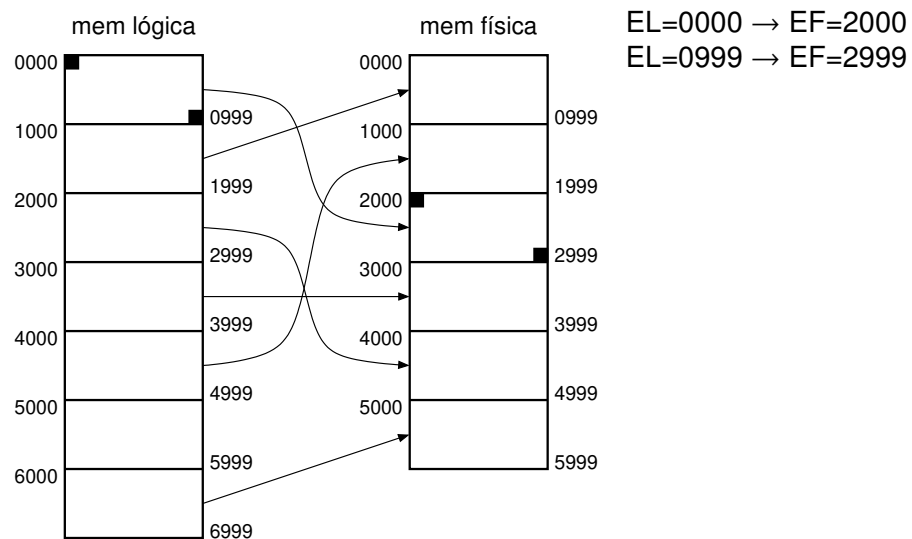
Navigation icons

Paginação decimal



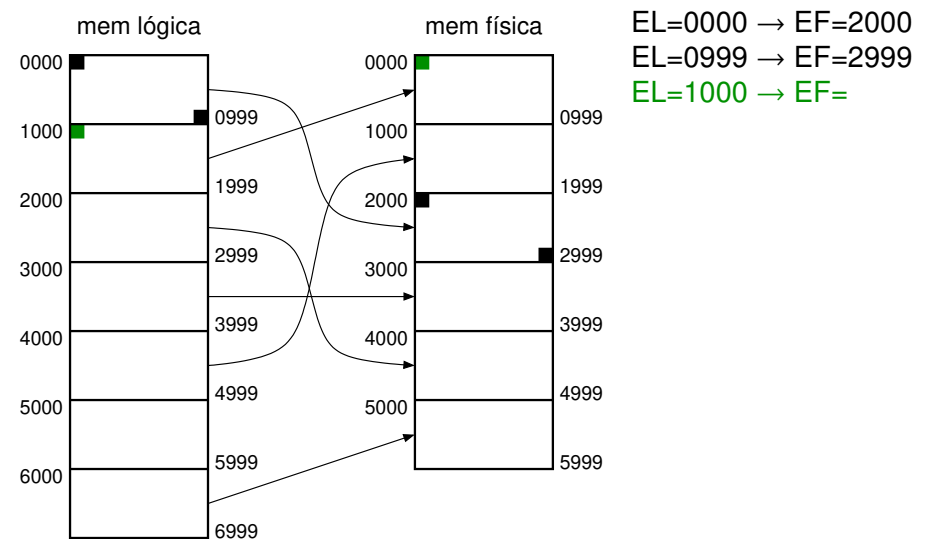
Navigation icons

Paginação decimal



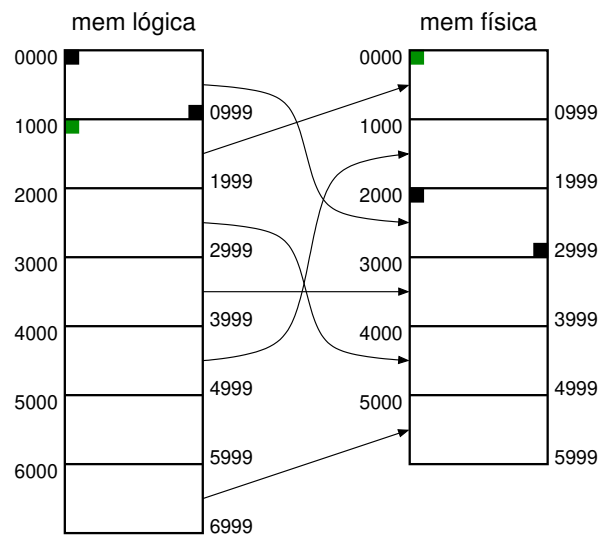
Navigation icons

Paginação decimal



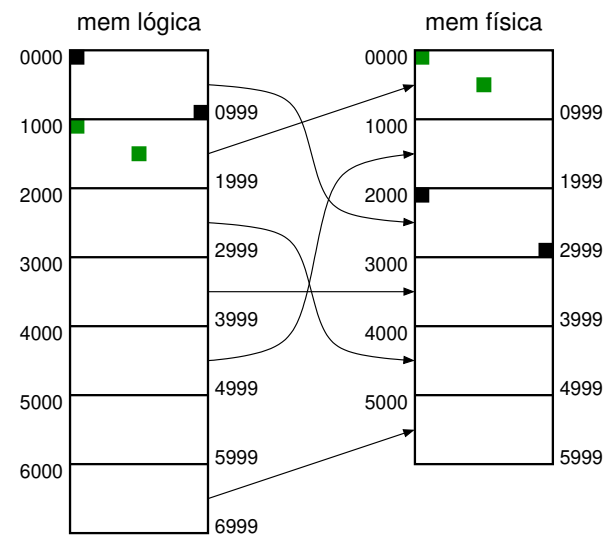
Navigation icons

Paginação decimal



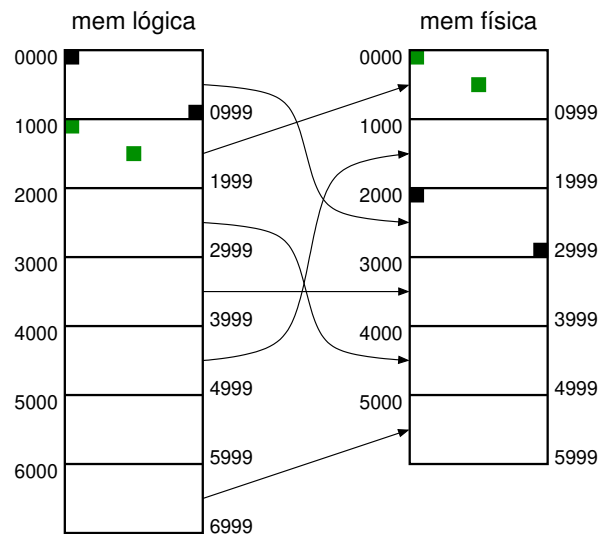
EL=0000 → EF=2000
EL=0999 → EF=2999
EL=1000 → EF=0000

Paginação decimal



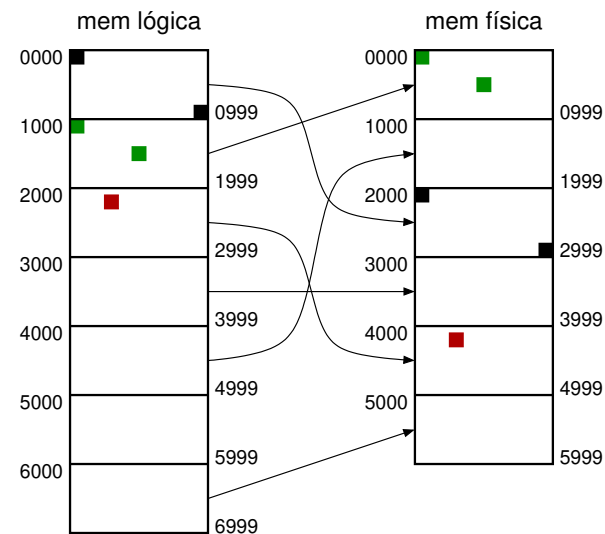
EL=0000 → EF=2000
EL=0999 → EF=2999
EL=1000 → EF=0000
EL=1500 → EF=

Paginação decimal



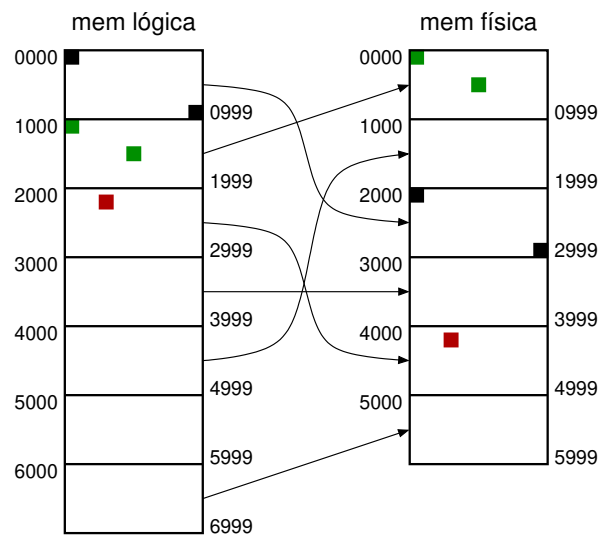
EL=0000 → EF=2000
EL=0999 → EF=2999
EL=1000 → EF=0000
EL=1500 → EF=0500

Paginação decimal



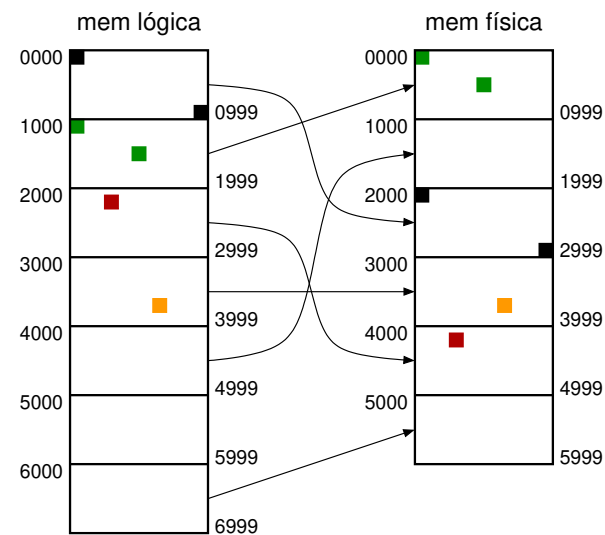
EL=0000 → EF=2000
EL=0999 → EF=2999
EL=1000 → EF=0000
EL=1500 → EF=0500
EL=2200 → EF=

Paginação decimal



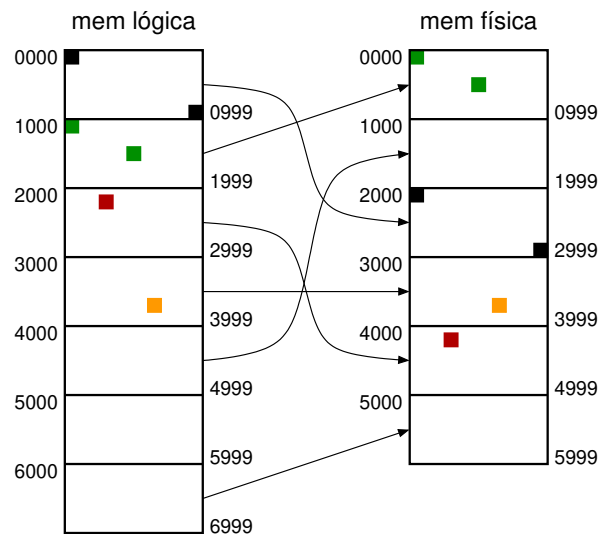
EL=0000 → EF=2000
EL=0999 → EF=2999
EL=1000 → EF=0000
EL=1500 → EF=0500
EL=2200 → EF=4200

Paginação decimal



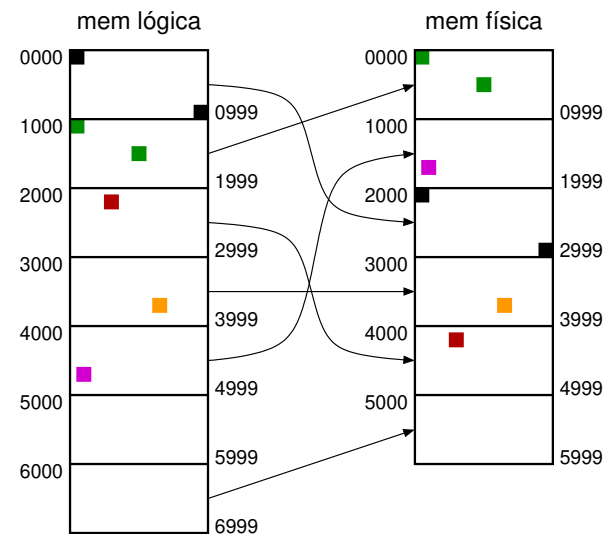
EL=0000 → EF=2000
EL=0999 → EF=2999
EL=1000 → EF=0000
EL=1500 → EF=0500
EL=2200 → EF=4200
EL=3846 → EF=

Paginação decimal



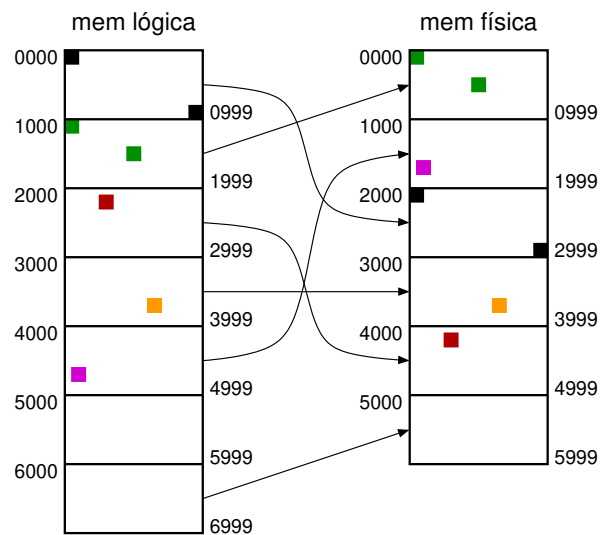
EL=0000 → EF=2000
EL=0999 → EF=2999
EL=1000 → EF=0000
EL=1500 → EF=0500
EL=2200 → EF=4200
EL=3846 → EF=3846

Paginação decimal



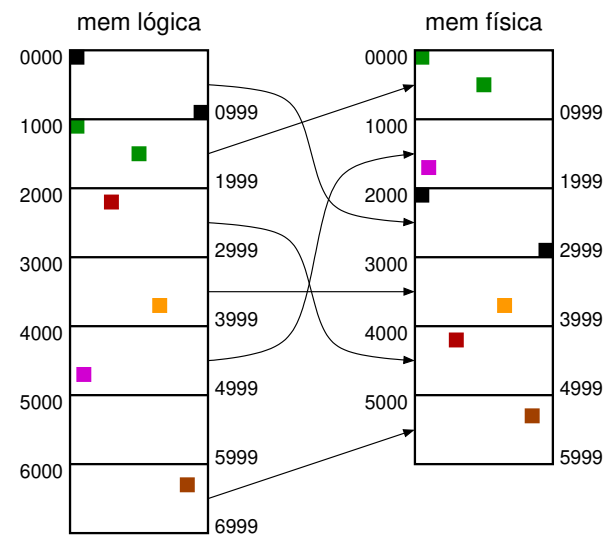
EL=0000 → EF=2000
EL=0999 → EF=2999
EL=1000 → EF=0000
EL=1500 → EF=0500
EL=2200 → EF=4200
EL=3846 → EF=3846
EL=4721 → EF=

Paginação decimal



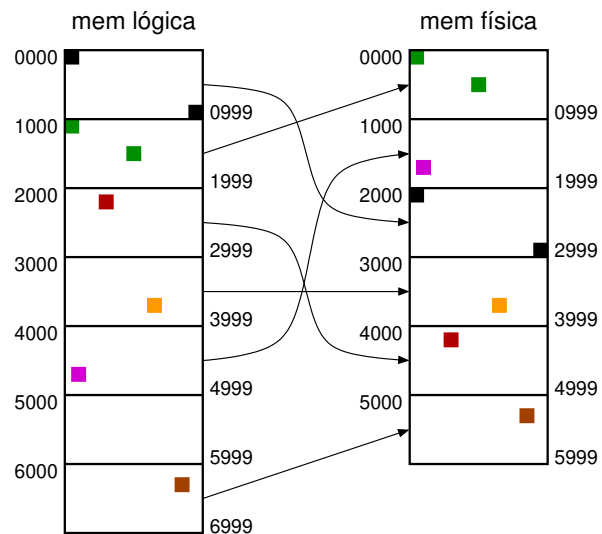
EL=0000 → EF=2000
 EL=0999 → EF=2999
 EL=1000 → EF=0000
 EL=1500 → EF=0500
 EL=2200 → EF=4200
 EL=3846 → EF=3846
 EL=4721 → EF=1721

Paginação decimal



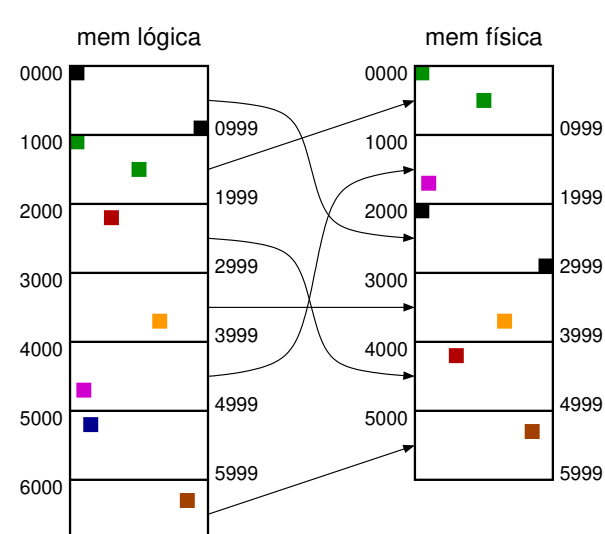
EL=0000 → EF=2000
 EL=0999 → EF=2999
 EL=1000 → EF=0000
 EL=1500 → EF=0500
 EL=2200 → EF=4200
 EL=3846 → EF=3846
 EL=4721 → EF=1721
 EL=6335 → EF=

Paginação decimal



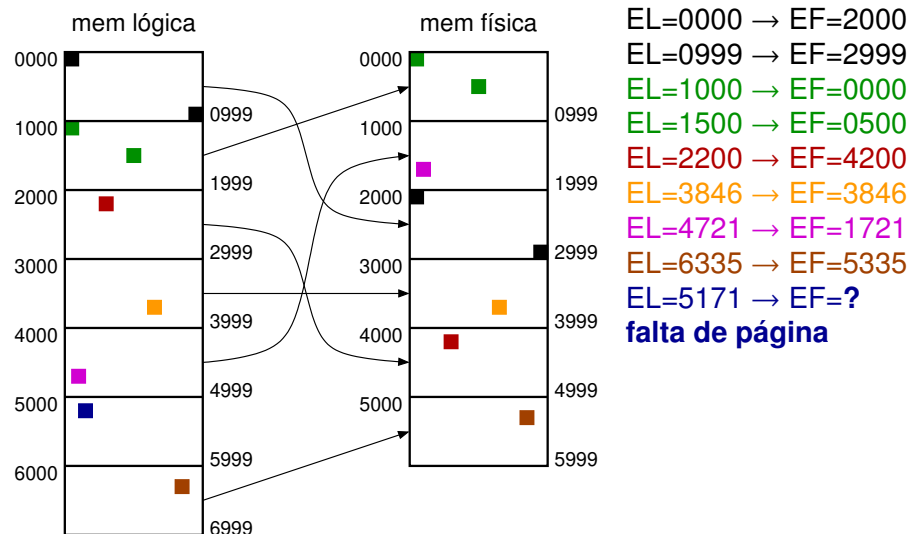
EL=0000 → EF=2000
 EL=0999 → EF=2999
 EL=1000 → EF=0000
 EL=1500 → EF=0500
 EL=2200 → EF=4200
 EL=3846 → EF=3846
 EL=4721 → EF=1721
 EL=6335 → EF=5335

Paginação decimal



EL=0000 → EF=2000
 EL=0999 → EF=2999
 EL=1000 → EF=0000
 EL=1500 → EF=0500
 EL=2200 → EF=4200
 EL=3846 → EF=3846
 EL=4721 → EF=1721
 EL=6335 → EF=5335
 EL=5171 → EF=

Paginação decimal



© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 34/98

Tradução de endereços

$$EL=2345 \rightarrow EF=4345$$

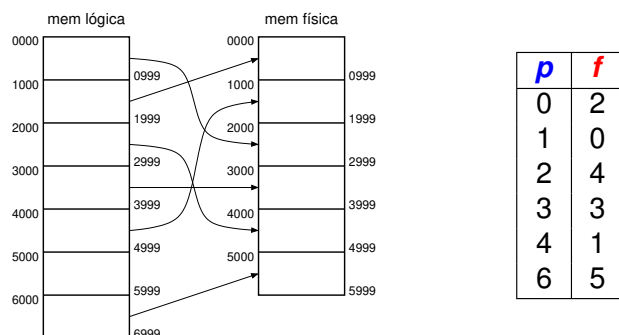
- Como as páginas têm 1000 bytes, os últimos 3 dígitos são iguais
- A diferença entre o EL e o EF é apenas o primeiro dígito
- A tradução de endereços lógicos em físicos consiste em
 - Traduzir o primeiro dígito
 - ★ faltas de página são tratadas aqui
 - Juntar com os últimos dígitos
- Mais formalmente, é preciso mapear p para f para traduzir

$$EL=pddd \rightarrow EF=fddd$$
 - p é o número de página [virtual | lógica]
 - f é o número de página física ou moldura de página
 - ddd é o deslocamento (\rightarrow distância desde o início da página)

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 35/98

Mapeando páginas lógicas em páginas físicas (1)

- Como mapear p em f ?
- Solução intuitiva: usar uma tabela

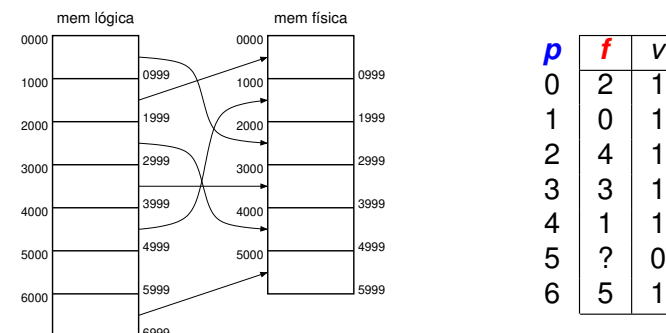


- Problema: busca na tabela é muito custosa, especialmente na falta de página
 - cada instrução de máquina envolve pelo menos uma tradução de endereço
 - nem busca binária salva (fora o custo da ordenação)

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 36/98

Mapeando páginas lógicas em páginas físicas (2)

- Se a tabela tiver uma entrada para cada possível número de página lógica, p pode virar um índice para a posição apropriada da tabela
 - acesso à tabela requer tempo constante, mesmo na falta de página
 - um bit v (de *válido*) indica se a página está presente ($v=1$) ou ausente ($v=0$) na memória física



© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 37/98

Um pseudocódigo para tradução de endereços

```
unsigned int traduz(unsigned int EL) {  
    unsigned int p = EL / 1000;  
    unsigned int ddd = EL % 1000;  
    if (tabpag[p].v)  
        return ((tabpag[p].f * 1000) + ddd);  
    else  
        /* falta de página */  
}
```

- Quando ocorre uma falta de página, uma página física livre deve ser alocada para o processo
 - ▶ caso não haja nenhuma PF livre, é necessário substituir uma das páginas atualmente alocadas
 - ★ algoritmos de substituição de páginas
 - ▶ em ambos os casos, a tabela de páginas deve ser atualizada com a nova situação

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 38/98

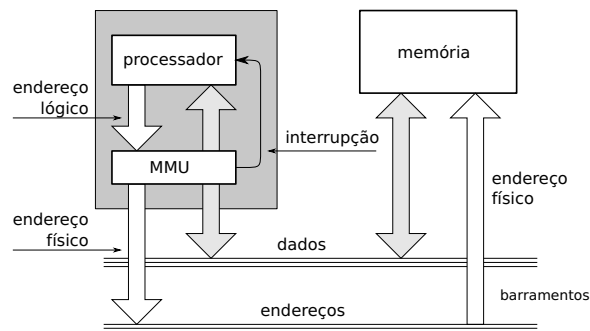
De paginação decimal para paginação binária

- Eficiência na tradução de endereços depende de minimizar o uso de instruções
 - ▶ páginas de mesmo tamanho são essenciais
- Computadores representam endereços em binário, não em decimal
 - ▶ **na prática só existe paginação binária, e não decimal**
- Um endereço de N bits é subdividido em D bits para o deslocamento e $P = N - D$ bits para o número de página
 - ▶ cada página tem 2^D bytes ($d = [0 \dots 2^D - 1]$)
 - ▶ existem 2^P páginas possíveis ($p = [0 \dots 2^P - 1]$)

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 39/98

Localização e função da MMU

- A tradução de endereços lógicos em físicos é realizada pela MMU (*memory management unit*), que usa a tabela de páginas



© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 40/98

Endereçamento virtual vs físico

- Seja um sistema com as seguintes características
 - ▶ espaço de endereçamento virtual de 64 KB
 - ▶ espaço de endereçamento físico de 32 KB
 - ▶ páginas de 4 KB
 - ★ tamanhos típicos de página variam entre 512 bytes e 64 KB
- Para esse sistema, são necessárias
 - ▶ $64 \text{ KB} / 4 \text{ KB} = 16$ páginas [virtuais]
 - ▶ $32 \text{ KB} / 4 \text{ KB} = 8$ páginas físicas

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 41/98

Endereçamento virtual vs físico

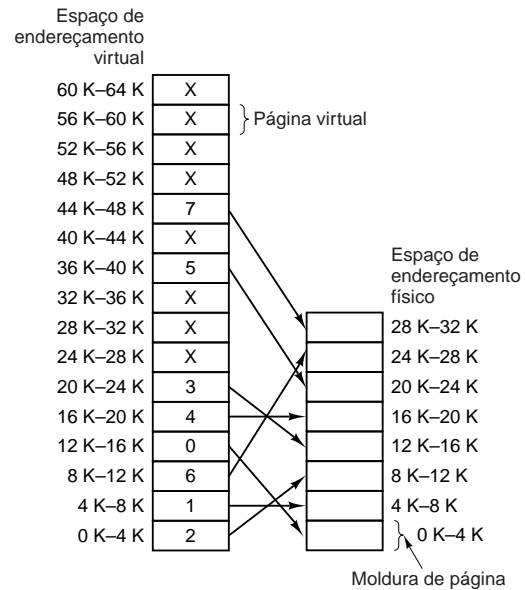
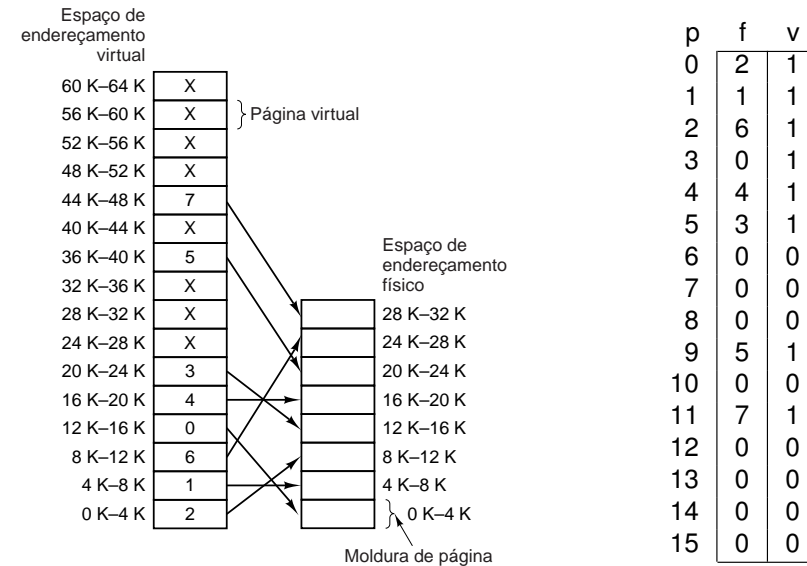


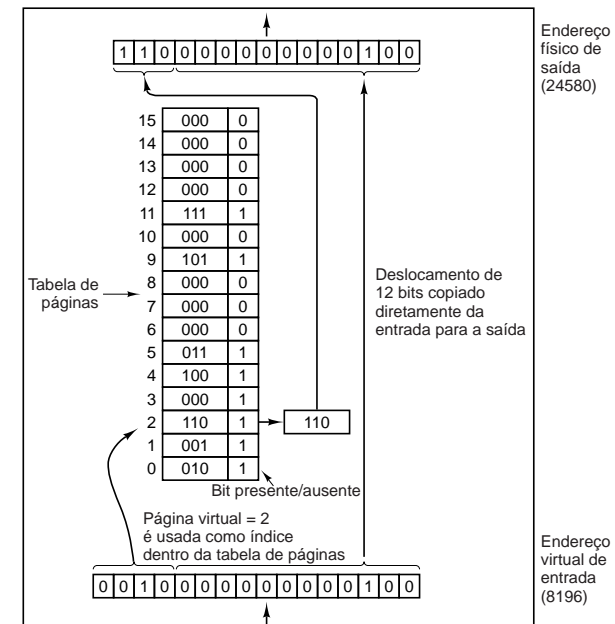
Tabela de páginas para o exemplo



Falta de página

- Um bit de presente/ausente indica quais páginas lógicas estão presentes na memória física
- Quando um endereço referenciado pertence a uma página lógica que não está na memória física, ocorre uma **falta de página** (*page fault*)
 - MMU gera uma interrupção
 - SO carrega a página para a memória física
 - se memória cheia, escolhe uma página para remoção
 - algoritmos de substituição de páginas serão vistos mais adiante
 - SO atualiza a tabela de páginas e retorna ao processo
 - instrução interrompida é reiniciada

Operação interna da MMU



Traduzindo o endereço em decimal

$$EL=8196 \rightarrow EF=?$$

- Tamanho de página = 4 KB = 4096 bytes
- $p = EL \div TamPag = 8196 \div 4096 = 2$
- $d = EL \bmod TamPag = 8196 \bmod 4096 = 4$
- $p = 2$ é mapeado em $f = 6$
- $EF = (f \times TamPag) + d = (6 \times 4096) + 4 = 24580$

Tabelas de páginas

- Endereços virtuais = número de página + deslocamento
 - deslocamento é posição dentro da página
- Endereços físicos = número de página física + deslocamento
 - deslocamento é o mesmo
- A tabela de páginas mapeia páginas lógicas em páginas físicas
- Para o exemplo anterior:
 - 12 bits para o deslocamento (páginas de 2^{12} bytes=4 KB)
 - 4 bits para o número de página ($2^4=16$ páginas)
 - 3 bits para o número da página física ($2^3=8$ páginas)
 - endereços lógicos têm $12 + 4 = 16$ bits (EEL= 2^{16} bytes=64 KB)
 - endereços físicos têm $12 + 3 = 15$ bits (EEF= 2^{15} bytes=32 KB)

Tabelas de páginas: desafios (1)

- Um desafio para sistemas de paginação é o tamanho da tabela de páginas
 - quantas entradas são requeridas para um sistema com endereços de 32 bits e páginas de 4 KB?
 - ★ memória lógica = 2^{32} bytes = 4 GB
 - ★ n° de páginas = $2^{32}/4096 = 2^{32}/2^{12} = 2^{20} = 1.048.576$ páginas
 - qual o espaço ocupado por essa tabela?
 - ★ espaço = n° de páginas \times tamanho de entrada
 - ★ com entradas de 4 bytes (caso típico), a tabela ocuparia $2^{20} \times 4 = 2^{22}$ bytes = 4 MB
- Cada processo tem sua própria tabela de páginas
 - com n processos, $n \times 4$ MB ocupados por tabelas de páginas
 - é necessário trocar a tabela a cada chaveamento de contexto

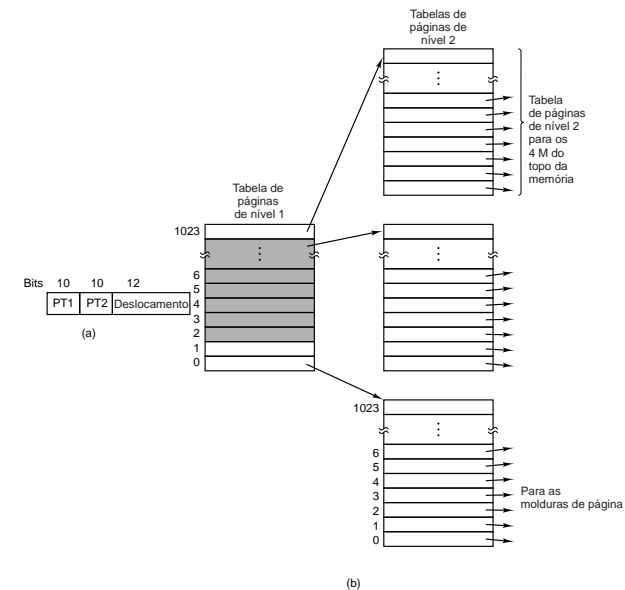
Tabelas de páginas: desafios (2)

- Outro desafio: o acesso à tabela deve ser rápido
 - usada em todo acesso à memória
- Soluções simplistas
 - tabela de páginas em memória
 - ★ um registrador da MMU armazena o endereço da tabela de páginas \Rightarrow PTBR (*page table base register*)
 - ★ problema: desempenho
 - tabela de páginas em registradores
 - ★ problemas: custo, desempenho nos chaveamentos de contexto

Tabelas de páginas multiníveis (1)

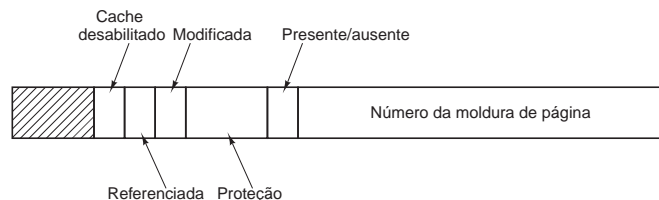
- Uma solução para tratar o tamanho de tabelas de páginas é o uso de tabelas de páginas multiníveis
- O número de página é subdividido em dois ou mais índices em tabelas de páginas distintas
 - ▶ ex: sistema de 32 bits com páginas de 4 KB
 - ▶ 20 bits estão disponíveis para o número de página
 - ▶ esses bits podem ser divididos em 10 + 10 bits
 - ★ tabela de páginas principal com $2^{10} = 1024$ entradas
 - ★ cada entrada aponta para outra tabela com $2^{10} = 1024$ entradas
 - ★ as entradas da 2ª tabela apontam para molduras de página
 - ★ esquema do 80386
- TP principal também é chamada de diretório de páginas
- Apenas as tabelas em uso precisam ser alocadas na memória
- Core i7: 4-5 níveis, cada tabela com 512 entradas de 8 bytes
 - ▶ cada subtabela ocupa exatamente uma página de 4 KB

Tabelas de páginas multiníveis (2)



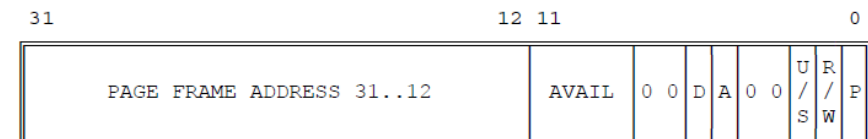
Entradas na tabela de páginas

- Tamanho comum de uma entrada: 32/64 bits
- Campos de uma entrada
 - ▶ número da moldura de página
 - ▶ bit presente (1) / ausente (0)
 - ★ acesso a página com este bit em 0 causa uma falta de página
 - ▶ bit de proteção: leitura-escrita (0) / leitura (1)
 - ★ esquemas mais sofisticados existem
 - ▶ modificada (sujo): (1) indica que a página foi escrita
 - ▶ referenciada: (1) indica que a página foi referenciada
 - ▶ cache desabilitado: (1) indica que a página não pode ser mantida em cache → E/S mapeada em memória



Exemplo: PTEs no 80386

Figure 5-10. Format of a Page Table Entry



P - PRESENT
 R/W - READ/WRITE
 U/S - USER/SUPERVISOR
 D - DIRTY
 AVAIL - AVAILABLE FOR SYSTEMS PROGRAMMER USE

NOTE: 0 INDICATES INTEL RESERVED. DO NOT DEFINE.

Memória associativa (TLB)

- Tabelas de páginas invariavelmente são mantidas em memória devido a seu tamanho
- Isso, porém, tem impacto no desempenho
 - a cada referência a memória, dois acessos são necessários
 - a própria busca de uma instrução é afetada
- Felizmente, na prática os programas tendem a referenciar um conjunto reduzido de páginas em um dado período
- Isso permite armazenar as entradas mais usadas em um conjunto de registradores na MMU
 - TLB (*translation lookaside buffer*)

Funcionamento da TLB

- Uma página primeiro é buscada na TLB
 - se ocorre um acerto (*hit*), só é feito um acesso à memória
 - se ocorre um erro (*miss*), dois acessos são efetuados
 - ★ nesse caso, a TLB é atualizada com os dados obtidos na tabela de páginas em memória
 - taxas de acerto típicas ultrapassam 99%
- TLB é invalidada a cada chaveamento de contexto
 - chaveamentos muito frequentes reduzem a taxa de acerto e consequentemente o desempenho

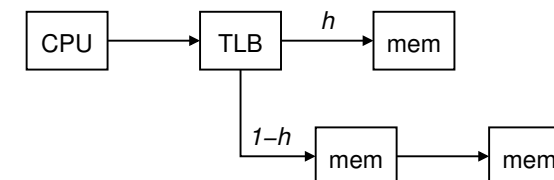
Exemplo de TLB

Válida	Página virtual	Bit modificada	Bits de proteção	Moldura de página
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Impacto de TLB no desempenho (1)

- Seja t_{tlb} o tempo de acesso à TLB e t_{mem} o tempo de acesso à memória
- $t_{hit} = t_{tlb} + t_{mem}$
- $t_{miss} = t_{tlb} + t_{mem} + t_{mem} = t_{tlb} + 2 t_{mem}$
- Para uma taxa de acerto h , o tempo médio de acesso à memória é

$$t_{ac} = h \cdot t_{hit} + (1 - h) \cdot t_{miss}$$



Impacto de TLB no desempenho (2)

- Seja $t_{tlb} = 20$ ns e $t_{mem} = 100$ ns

$$t_{hit} = 20 + 100 = 120 \text{ ns}$$

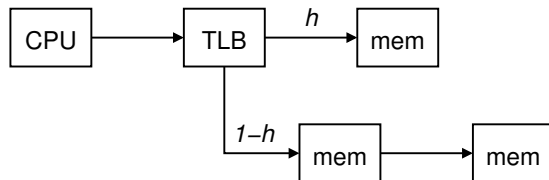
$$t_{miss} = 20 + 2 \times 100 = 220 \text{ ns}$$

- Se a taxa de acerto for de 85% ($h = 0,85$):

$$t_{ac} = 0,85 \times 120 + (1 - 0,85) \times 220 = 135 \text{ ns}$$

- Se a taxa de acerto for de 99% ($h = 0,99$):

$$t_{ac} = 0,99 \times 120 + (1 - 0,99) \times 220 = 121 \text{ ns}$$



Algoritmos de substituição de páginas

- A falta de página força uma escolha
 - qual página deve ser removida
 - alocação de espaço para a página a ser trazida para a memória
- A página modificada deve primeiro ser salva
 - se não tiver sido modificada é apenas sobreposta
- Melhor não escolher uma página que está sendo muito usada
 - provavelmente precisará ser trazida de volta logo

Algoritmo Ótimo

- Substitui a página necessária o mais à frente possível
 - ótimo mas não realizável
- Estimada através de
 - registro do uso da página em execuções anteriores do processo
 - apesar disto ser impraticável

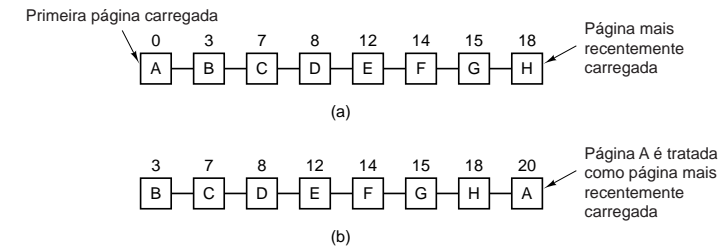
Algoritmo Não Usada Recentemente (NUR)

- Cada página tem os bits Referenciada (R) e Modificada (M)
 - bits são colocados em 1 quando a página é referenciada ou modificada
- As páginas são classificadas
 - classe 0: não referenciada, não modificada
 - classe 1: não referenciada, modificada
 - classe 2: referenciada, não modificada
 - classe 3: referenciada, modificada
- NUR remove página aleatoriamente
 - da classe de ordem mais baixa que não esteja vazia

Algoritmo FIFO

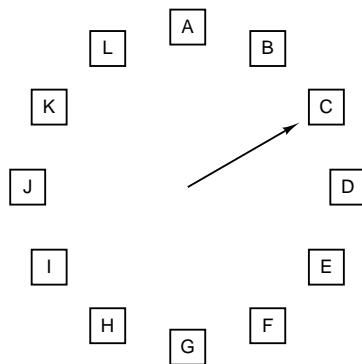
- Mantém uma lista encadeada de todas as páginas
 - ▶ página mais antiga na cabeça da lista
 - ▶ página que chegou por último na memória no final da lista
- Na ocorrência de falta de página
 - ▶ página na cabeça da lista é removida
 - ▶ nova página adicionada no final da lista
- Desvantagem
 - ▶ página há mais tempo na memória pode ser usada com muita frequência

Algoritmo Segunda Chance (SC)



- Operação do algoritmo segunda chance
 - ▶ lista de páginas em ordem FIFO
 - ▶ estado da lista em situação de falta de página no instante 20, com o bit R da página A em 1 (números representam instantes de carregamento das páginas na memória)
 - ▶ a referência considerada é aquela do intervalo de relógio anterior

Algoritmo do Relógio



Quando ocorre uma falta de página, a página apontada é examinada. A atitude a ser tomada depende do bit R:
R = 0: Retira a página,
R = 1: Faz R = 0 e avança o ponteiro.

Menos Recentemente Usada (MRU)

- Premissa do algoritmo ótimo (impraticável)
 - ▶ páginas referenciadas nas últimas instruções serão novamente referenciadas nas próximas instruções
- Reflexão (de modo oposto)
 - ▶ páginas que não foram referenciadas nas últimas instruções provavelmente não o sejam nas próximas

Menos Recentemente Usada (MRU)

- Assume que páginas usadas recentemente logo serão usadas novamente
 - retira da memória página que há mais tempo não é usada
- Uma lista encadeada de páginas deve ser mantida
 - página mais recentemente usada no início da lista, menos usada no final da lista
 - atualização da lista a cada referência à memória
- Alternativamente, manter contador em cada entrada da tabela de página
 - escolhe página com contador de menor valor
 - zera o contador periodicamente

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 66/98

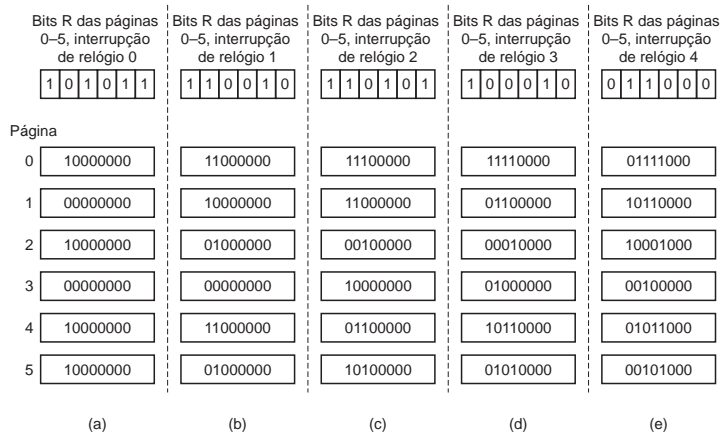
Simulação do MRU em software (1)

- A primeira aproximação seria o algoritmo NFU (não frequentemente usado)
 - associa um contador a cada página, inicialmente 0
 - a cada interrupção de relógio, percorre a lista de páginas e adiciona o bit R ao contador da página
 - página com o menor contador é a mais antiga
- O problema do NFU é que ele tem longa memória
 - páginas intensamente referenciadas no passado têm contadores com valores elevados, mesmo que não sejam mais necessárias
 - páginas recém carregadas, mesmo que estejam sendo muito usadas, têm contadores mais baixos e acabam sendo escolhidas

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 67/98

Simulação do MRU em software (2)

- Uma solução é o algoritmo de envelhecimento (*aging*)
 - o contador é deslocado 1 bit para a direita
 - o bit R é copiado para o bit mais à esquerda



© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 68/98

Revisão dos algoritmos

Algoritmo	Comentário
Ótimo	Não implementável, mas útil como um padrão de desempenho
NUR (não usada recentemente)	Muito rudimentar
FIFO (primeira a entrar, primeira a sair)	Pode descartar páginas importantes
Segunda chance	Algoritmo FIFO bastante melhorado
Relógio	Realista
MRU (menos recentemente usada)	Excelente algoritmo, porém difícil de ser implementado de maneira exata
NFU (não frequentemente usada)	Aproximação bastante rudimentar do MRU
Envelhecimento (<i>aging</i>)	Algoritmo bastante eficiente que se aproxima bem do MRU
Conjunto de trabalho	Implementação um tanto cara
WSClock	Algoritmo bom e eficiente

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 69/98

Sumário

- 1 Conceitos básicos
- 2 Gerenciamento sem abstração de memória
- 3 Gerenciamento com espaços de endereçamento
- 4 Memória virtual
- 5 Paginação
- 6 Paginação: questões de projeto e implementação
- 7 Segmentação

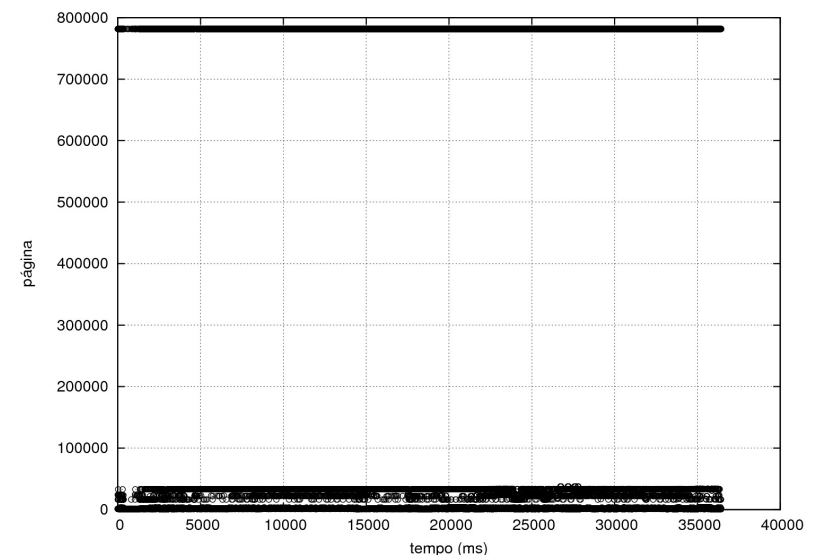
Paginação por demanda

- Na forma mais pura de paginação, um processo inicia sem nenhuma de suas páginas presentes na memória
- Quando a CPU vai buscar a primeira instrução, isso gera uma falta de página
 - ▶ outras faltas de página, para dados e pilha, geralmente se sucedem
- Depois de um certo tempo, todas as páginas necessárias à execução do processo estão na memória, e o número de faltas de página é bastante reduzido
- Essa estratégia é chamada de **paginação por demanda**

Localidade e conjunto de trabalho

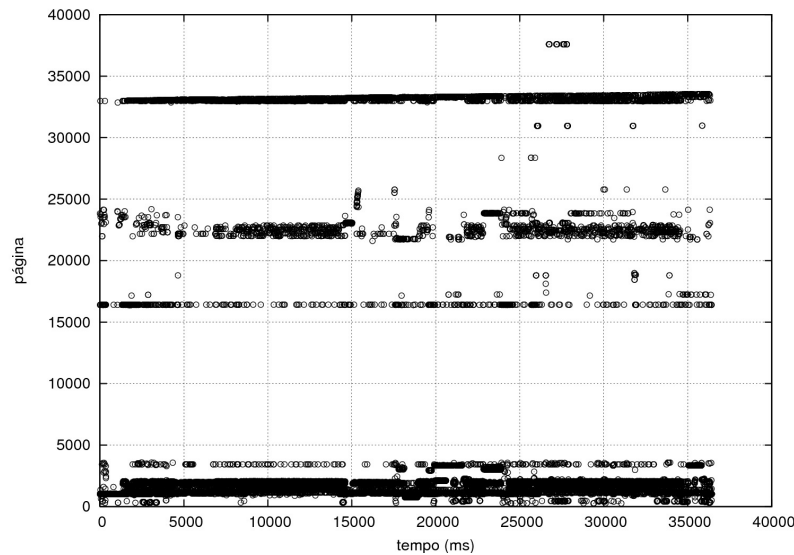
- A maioria dos processos apresenta **localidade de referências**: em um dado instante, as referências à memória se concentram em um conjunto reduzido de páginas
- O conjunto de páginas acessadas na história recente de um processo é o seu **conjunto de trabalho** (*working set*)
- O conjunto de trabalho evolui dinamicamente à medida em que o processo executa
- Se todas as páginas do conjunto de trabalho estão na memória, o processo executa com poucas faltas de página
 - ▶ apenas acessos a novas páginas geram faltas

Localidade de referências no gThumb (1)



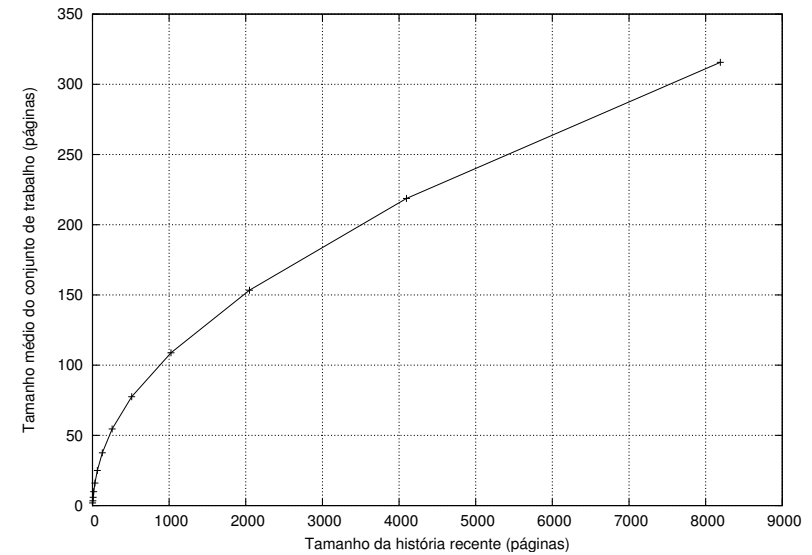
visão geral

Localidade de referências no gThumb (2)



visão da parte inferior (código, dados, heap)

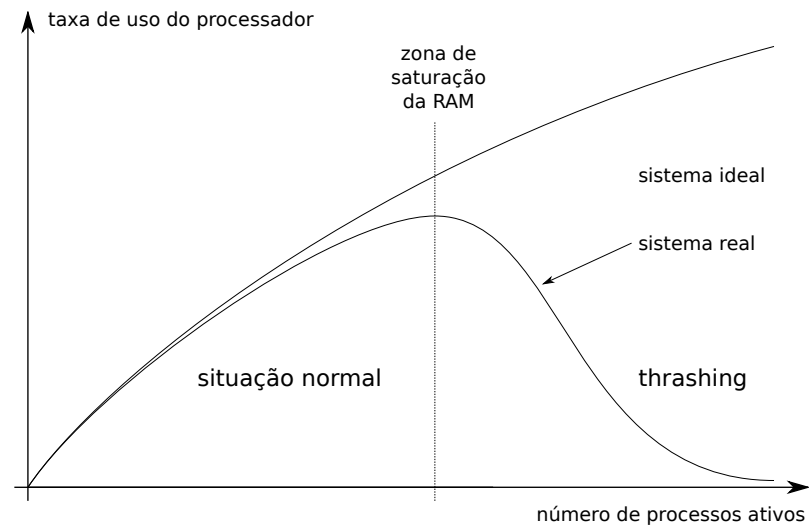
Conjunto de trabalho para o gThumb



Thrashing

- Caso a memória física não seja suficiente para armazenar o agregado do conjunto de trabalho de todos os processos, o número de faltas de página cresce rapidamente
 - ▶ o processo escalonado gera faltas de página para completar seu conjunto de trabalho
 - ▶ como não há memória física, é necessário usar páginas físicas de outros processos
 - ▶ quando um desses processos for escalonado, seu conjunto de trabalho também estará incompleto
- **Thrashing:** sistema passa boa parte do tempo (ou mesmo a maior parte do tempo) paginando em vez de fazer algo útil
 - ▶ solução é reduzir o número de processos no sistema

Thrashing



Políticas de alocação de páginas (1)

- Quando se escolhe uma página vítima, deve-se considerar todas as páginas na memória ou apenas as páginas do processo que causou a falta de página?
 - em outras palavras, como dividir a memória física entre os processos?
- **Alocação local:** considera apenas as páginas do processo
- **Alocação global:** considera as páginas de todos os processos

Políticas de alocação de páginas (3)

- Alocação global usa melhor a memória disponível
 - ▶ se um processo usar toda a memória disponível para ele e estiver sendo usada alocação local, o processo pode entrar em thrashing, mesmo que haja memória livre
 - ▶ caso o conjunto de trabalho diminua, alocação local causa desperdício de memória
- Outra estratégia é alocar uma fração das páginas disponíveis para cada processo
 - ▶ geralmente proporcional ao tamanho do processo

Políticas de alocação de páginas (2)

A0	10
A1	7
A2	5
A3	4
A4	6
A5	3
B0	9
B1	4
B2	6
B3	2
B4	5
B5	6
B6	12
C1	3
C2	5
C3	6

(a)

A0
A1
A2
A3
A4
A6
B0
B1
B2
B3
B4
B5
B6
C1
C2
C3

(b)

A0
A1
A2
A3
A4
A5
B0
B1
B2
A6
B4
B5
B6
C1
C2
C3

(c)

(a) original

(b) alocação local

(c) alocação global

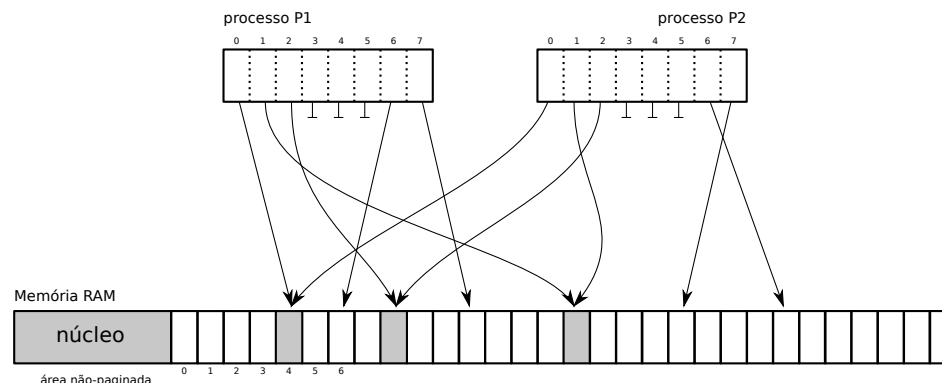
Tamanho de página

- Uma decisão de projeto importante é o tamanho da página
 - ▶ escolha de hardware ou do SO, dependendo da arquitetura
- Por que preferir páginas pequenas
 - ▶ menos desperdício de memória
 - ★ fragmentação interna: 1/2 página por processo (média)
 - ★ memória alocada porém não efetivamente usada
 - ▶ mais fácil manter o conjunto de trabalho de todos os processos na memória
- Por que preferir páginas grandes
 - ▶ tabelas de páginas menores
 - ▶ transferência mais rápida do disco
 - ▶ TLB cobre maiores áreas de memória
- Tendência tem sido o uso de páginas maiores

Compartilhamento de memória

- Em sistemas multiusuários e servidores, é comum ter várias instâncias do mesmo programa sendo executadas ao mesmo tempo
- Cada processo tem suas próprias áreas de código, dados, heap e pilha
- O código e os dados constantes podem ser compartilhados entre as várias instâncias → economia de memória
 - ▶ exemplo: programa com 200 MB (100 MB código + 100 MB dados)
 - ★ 10 instâncias **sem** compartilhamento: 2000 MB
 - ★ 10 instâncias **com** compartilhamento: 1100 MB
- O compartilhamento pode ser implementado mapeando páginas lógicas de processos distintos nas mesmas páginas físicas

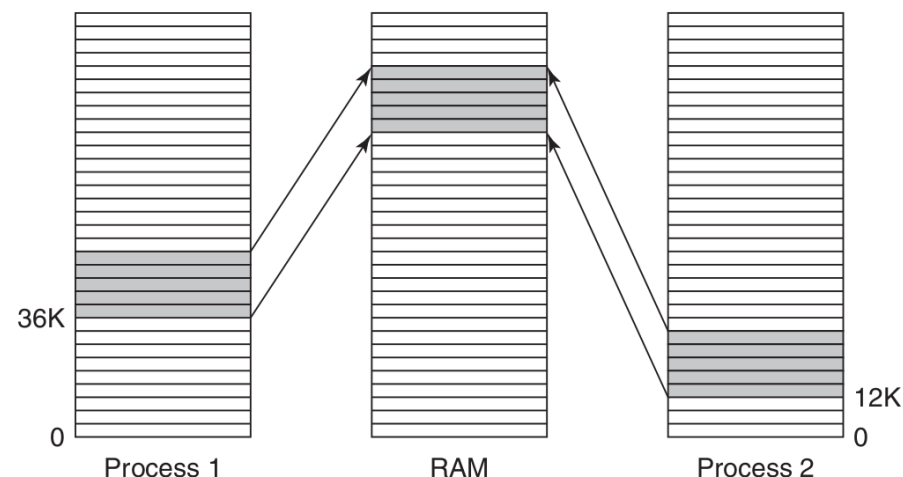
Compartilhamento de memória paginada



Bibliotecas compartilhadas (1)

- Programas executáveis precisam ter acesso a funções de biblioteca
 - ▶ `printf()`, `scanf()`, `qsort()`, ...
- Providenciar acesso às bibliotecas é tarefa do ligador (*linker*)
 - ▶ ligação estática: o executável incorpora o código e os dados necessários das bibliotecas
 - ▶ ligação dinâmica: o executável contém ponteiros para as bibliotecas que precisa, que são resolvidos durante a carga
 - ★ bibliotecas compartilhadas no Unix, *dynamic-link libraries* (DLLs) no Windows
- Ligação estática consome mais espaço em disco e na memória, e exige recompilar/religar todos os executáveis em caso de alterações na biblioteca
- Com paginação sob demanda, as bibliotecas compartilhadas podem residir apenas parcialmente na memória

Bibliotecas compartilhadas (2)



Copiar ao escrever – *copy-on-write* (COW)

- Técnica usada para implementar `fork()` em sistemas modernos
- Em vez de duplicar o espaço de endereçamento do processo pai para criar o processo filho, apenas as entradas da tabela de páginas são copiadas
 - ▶ páginas protegidas contra escrita no pai e no filho
 - ▶ por default, todas as páginas são compartilhadas
- Quando um processo (pai ou filho) tenta escrever em uma página, a MMU gera uma exceção
 - ▶ SO aloca outra página física, copia o conteúdo para a nova página e ajusta a tabela de páginas → apenas no processo que escreve
 - ★ página passa a ter permissão de escrita

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 86/98

Travamento de páginas

- Determinadas páginas não podem ser retiradas da memória
 - ▶ buffers de E/S (especialmente com DMA), estruturas de dados usadas pelo núcleo, pilha do núcleo
- Essas páginas são travadas na memória
- O algoritmo de substituição de páginas precisa levar em conta quais páginas estão travadas

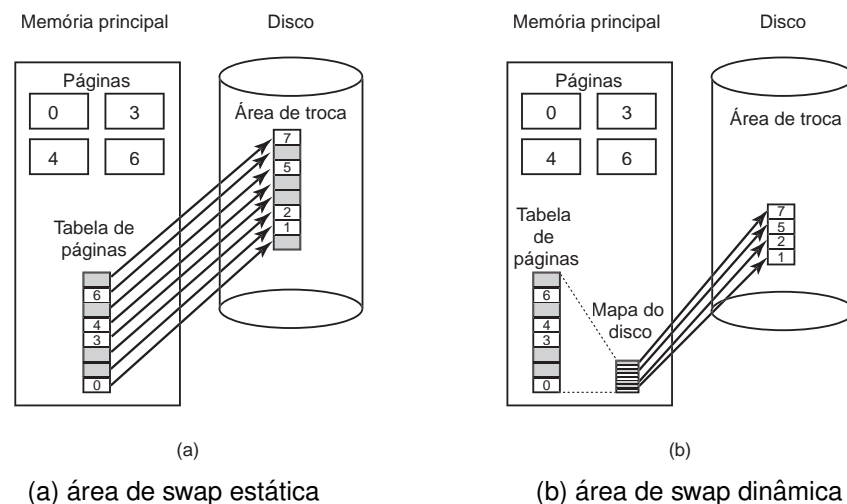
© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 87/98

Memória secundária (1)

- Como gerenciar o armazenamento de páginas no disco?
- **Área de swap (troca):** partição/disco dedicada para páginas
 - ▶ um espaço reservado na área de swap para cada processo criado
 - ★ endereço da área do processo é mantido na tabela de páginas
 - ★ processo deve ser copiado para área de swap na carga, ou páginas copiadas da memória quando necessário
 - ▶ para lidar com crescimento da memória, pode-se ter áreas de swap separadas para código, dados e pilha (possivelmente com múltiplas partes)
 - ▶ outra forma é alocar espaço apenas quando necessário
 - ★ desvantagem é precisar mapear páginas nos seus endereços no disco
 - ★ própria tabela de páginas pode armazenar endereços de disco nas entradas marcadas como inválidas
- Alguns sistemas (ex: Windows) usam um **arquivo de paginação**, com tamanho prealocado
 - ▶ para código pode-se usar o próprio executável como memória secundária

© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 88/98

Memória secundária (2)



© 2022 Rafael Obelheiro (DCC/UDESC) Gerência de Memória SOP 89/98

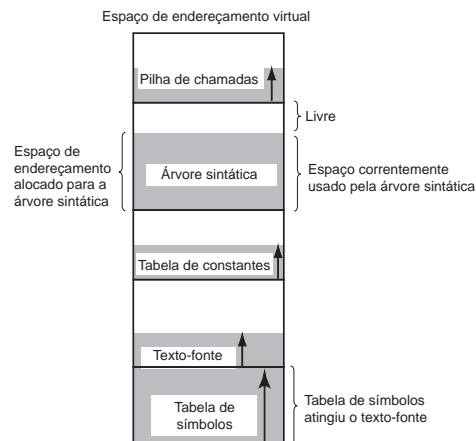
Sumário

- 1 Conceitos básicos
- 2 Gerenciamento sem abstração de memória
- 3 Gerenciamento com espaços de endereçamento
- 4 Memória virtual
- 5 Paginação
- 6 Paginação: questões de projeto e implementação
- 7 Segmentação

Segmentação

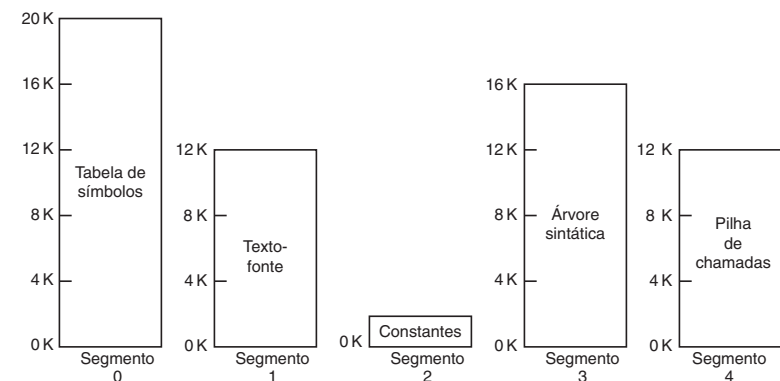
- Para alguns problemas, ter dois ou mais **espaços de endereçamento** é melhor do que um
- Em um compilador, o crescimento de uma **tabela de símbolos** é diferente do **texto fonte** (código)
- Fornecer espaços de endereçamento completamente independentes, **segmentos**
- Segmentos podem ter tamanhos diferentes
 - pode causar fragmentação externa
- Solução: segmentação paginada

Segmentação



- Espaço de endereçamento unidimensional com tabelas crescentes
- Uma tabela pode atingir outra

Segmentação



Permite que cada tabela cresça ou encolha, independentemente

Implementação de segmentação

- O espaço de endereçamento de um processo é composto por um conjunto de segmentos, que podem ser de tamanhos diferentes
- Cada segmento pode ser descrito por um par de endereços base e limite
 - análogo ao usado com alocação contígua
- A descrição dos segmentos é armazenada em uma tabela de segmentos
- Um endereço lógico é um par (*segmento, deslocamento*)

Exemplo de tabela de segmentos

- Tabela de segmentos (*limite* indica o final do segmento):

segmento	base	limite
0	3400	5000
1	7000	11476
2	0	3200
3	5500	6500

- EL: (0, 1400) \rightarrow EF: $3400 + 1400 = 4800$
- EL: (1, 1400) \rightarrow EF: $7000 + 1400 = 8400$
- EL: (2, 1400) \rightarrow EF: $0 + 1400 = 1400$
- EL: (3, 1400) $\rightarrow ?$

Exemplo de tabela de segmentos

- Tabela de segmentos (*limite* indica o final do segmento):

segmento	base	limite
0	3400	5000
1	7000	11476
2	0	3200
3	5500	6500

- EL: $(0, 1400) \rightarrow$ EF: $3400 + 1400 = 4800$
- EL: $(1, 1400) \rightarrow$ EF: $7000 + 1400 = 8400$
- EL: $(2, 1400) \rightarrow ?$
- EL: $(3, 1400) \rightarrow ?$

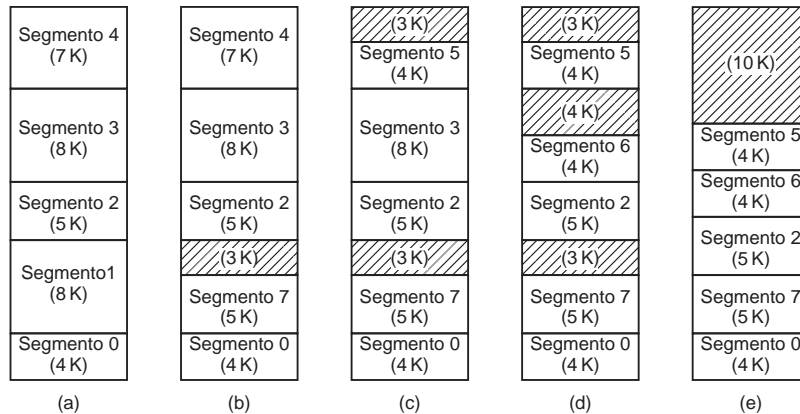
Exemplo de tabela de segmentos

- Tabela de segmentos (*limite* indica o final do segmento):

segmento	base	limite
0	3400	5000
1	7000	11476
2	0	3200
3	5500	6500

- EL: (0, 1400) → EF: 3400 + 1400 = 4800
- EL: (1, 1400) → EF: 7000 + 1400 = 8400
- EL: (2, 1400) → EF: 0 + 1400 = 1400
- EL: (3, 1400) → EF: 5500 + 1400 = 6900
6900 > 6500 (limite) ⇒ **falha de segmentação**

Fragmentação externa na segmentação





- (a)–(d) Desenvolvimento de fragmentação externa
- (e) Remoção da fragmentação via compactação

Segmentação com paginação

- Combina segmentação e paginação
 - segmentação paginada
- Um segmento não é alocado contigualmente, mas é paginado
 - cada segmento possui uma tabela de páginas
- Um endereço lógico é dividido em três partes
 - número do segmento
 - número da página (referente ao segmento)
 - deslocamento dentro da página

⇒ estrutura análoga à paginação multinível
- Elimina fragmentação externa na segmentação

Bibliografia Básica

-  **Andrew S. Tanenbaum.**
Sistemas Operacionais Modernos, 4ª Edição. Capítulo 3.
Pearson Prentice Hall, 2016.
-  **Carlos A. Maziero.**
Sistemas Operacionais: Conceitos e Mecanismos. Capítulos 14–18.
Editora da UFPR, 2019.
<http://wiki.inf.ufpr.br/maziero/doku.php?id=socm:start>