

Capítulo 9: Memória virtual



Capítulo 9: Memória virtual

- Conceitos Básicos
- Paginação por demanda
- Cópia na escrita
- Substituição de página
- Alocação de frames
- Thrashing
- Arquivos mapeados na memória
- Alocando memória ao kernel
- Outras considerações
- Exemplos de sistema operacional



Objetivos

- ❑ Descrever os benefícios de um sistema de memória virtual
- ❑ Explicar os conceitos de paginação por demanda, algoritmos de substituição de página e alocação de frames de página
- ❑ Discutir os princípios do modelo de conjunto de trabalho

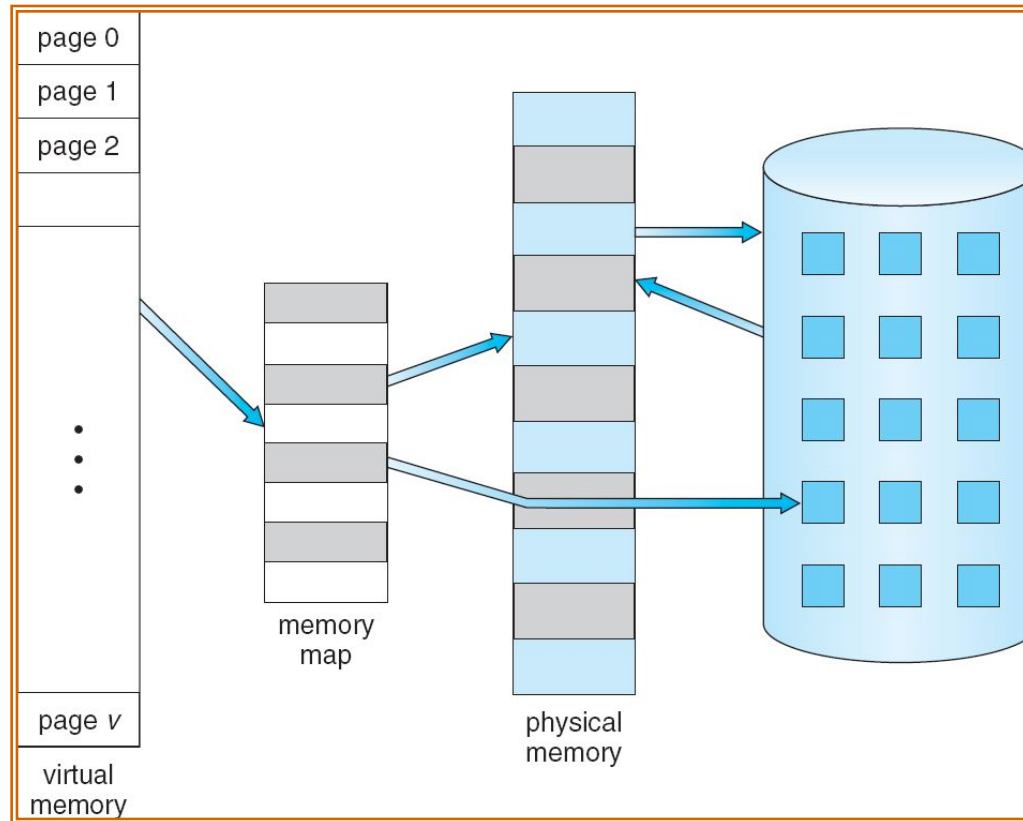


Base

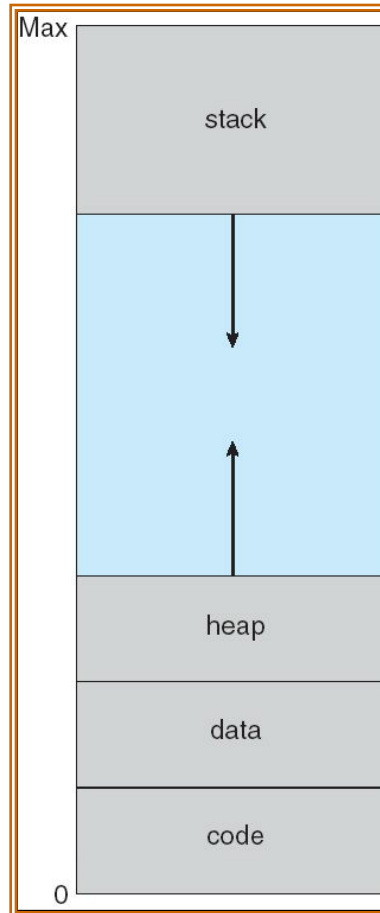
- **Memória virtual** – separação da memória lógica do usuário da memória física.
 - Somente parte do programa precisa estar na memória para execução
 - Logo, espaço de endereço lógico pode ser muito maior que o espaço de endereços físicos
 - Permite que espaços de endereço sejam compartilhados por vários processos
 - Permite criação de processo mais eficiente
- A memória virtual pode ser implementada por:
 - Paginação por demanda
 - Segmentação por demanda



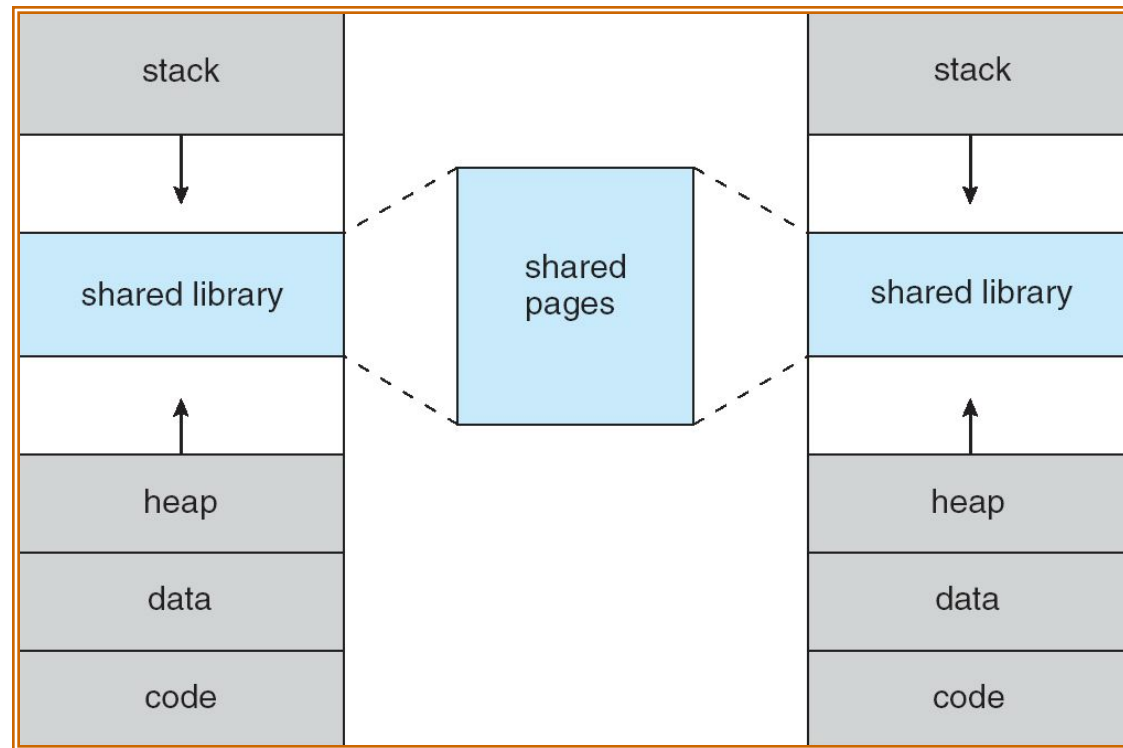
Memória virtual maior que a memória física



Espaço de endereço virtual



Biblioteca compartilhada usando memória virtual

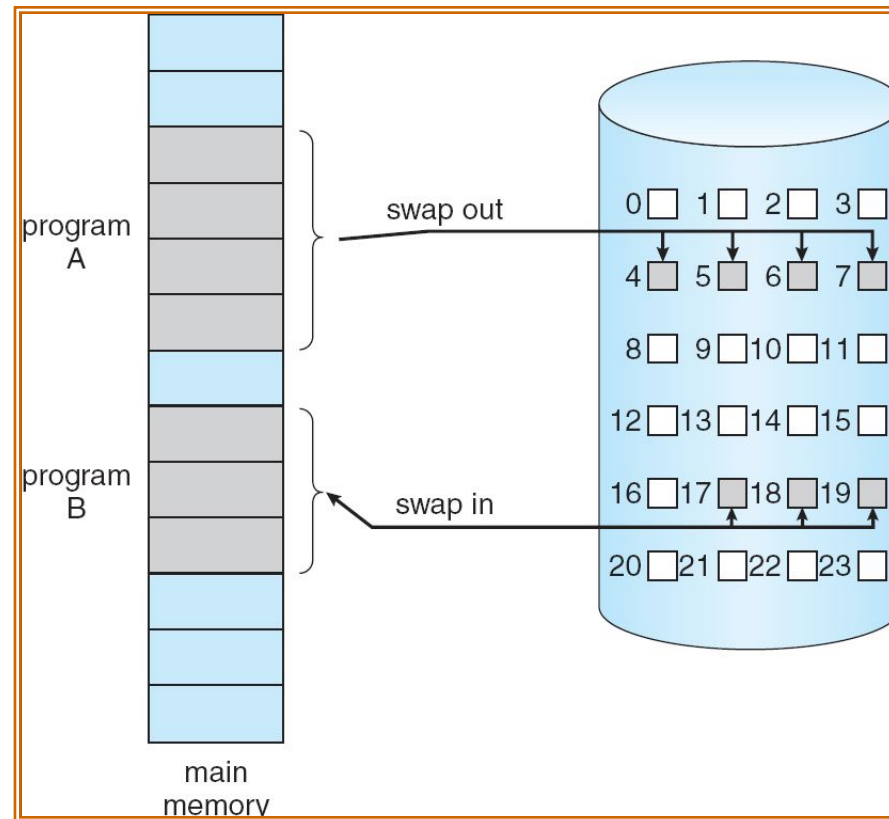


Paginação por demanda

- Traz a página para memória somente quando necessária
 - Menos E/S necessária
 - Menos memória necessária
 - Resposta mais rápida
 - Mais usuários
- Página é necessária \Rightarrow referencia ela
 - referência inválida \Rightarrow aborta
 - não está na memória \Rightarrow traz para memória
- **Lazy swapper** – nunca troca uma página da memória, a menos que a página seja necessária
 - Swapper que lida com páginas em um **paginador**



Transferência de página em memória para espaço contíguo em disco



Bit válido-inválido

- A cada entrada de tabela de página, um bit de válido-inválido é associado (**v** \Rightarrow na memória, **i** \Rightarrow não-na-memória)
- Inicialmente, o bit válido–inválido é definido como **i** em todas as entradas
- Exemplo de um snapshot de tabela de página

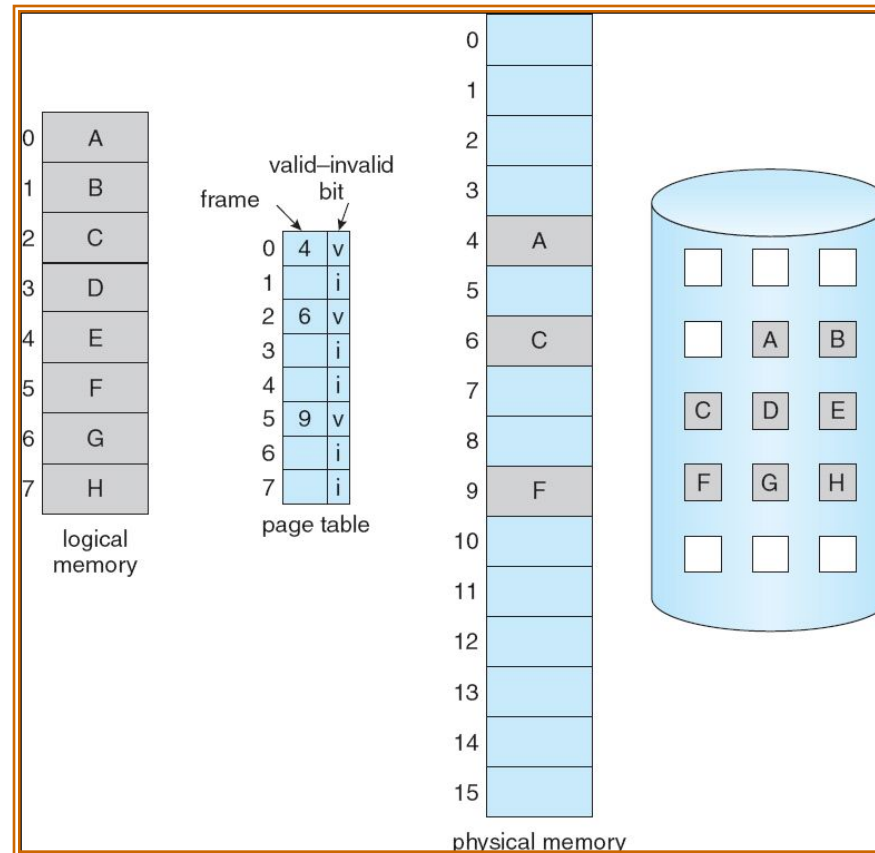
Quadro #	Bit válido-inválido
	v
	v
	v
	v
	i
....	
	i
	i

tabela de página

- Durante a tradução do endereço, se o bit válido–inválido na entrada da tabela de página for **i** \Rightarrow falta de página



Tabela de página quando algumas páginas não estão na memória



Falta de página

- Se houver referência a uma página, primeira referência a essa página causará um trap para o sistema operacional:
- **falha de página (page fault)**
- 1. Sistema operacional examina outra tabela para decidir:
 - Referência inválida \Rightarrow aborta
 - Apenas não está na memória
- 2. Apanha quadro vazio
- 3. Passa página para quadro
- 4. Reinicia tabelas
- 5. Define bit de validação = **v**
- 6. Reinicia a instrução que causou a falta de página



Exemplo de paginação por demanda

- Tempo de acesso à memória = 200 nanossegundos
- Tempo médio de serviço de falta de página = 8 milissegundos
- $EAT = (1 - p) \times 200 + p (8 \text{ milissegundos})$
 $= (1 - p) \times 200 + p \times 8.000.000$
 $= 200 + p \times 7.999.800$
- Se um acesso dentro de 1.000 causar uma falta de página, então
 $EAT = 8.2 \text{ microssegundos.}$
Isso é um atraso por um fator de 40 vezes !!



Criação de processo

- A memória virtual permite outros benefícios durante a criação do processo:
 - Cópia na escrita
 - Arquivos mapeados na memória



Cópia na escrita

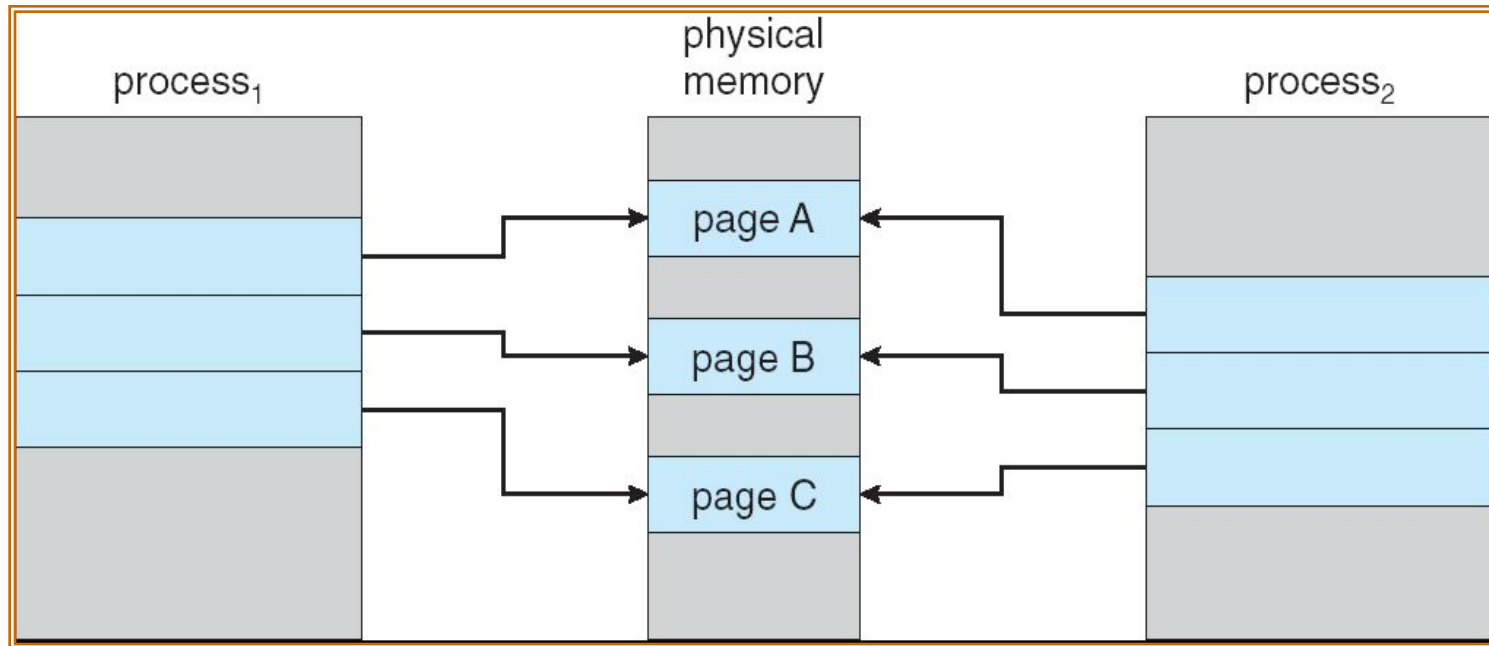
- Cópia na escrita permite que processos pai e filho inicialmente *compartilhem* as mesmas páginas na memória

Se qualquer processo modificar uma página compartilhada, somente então a página é copiada

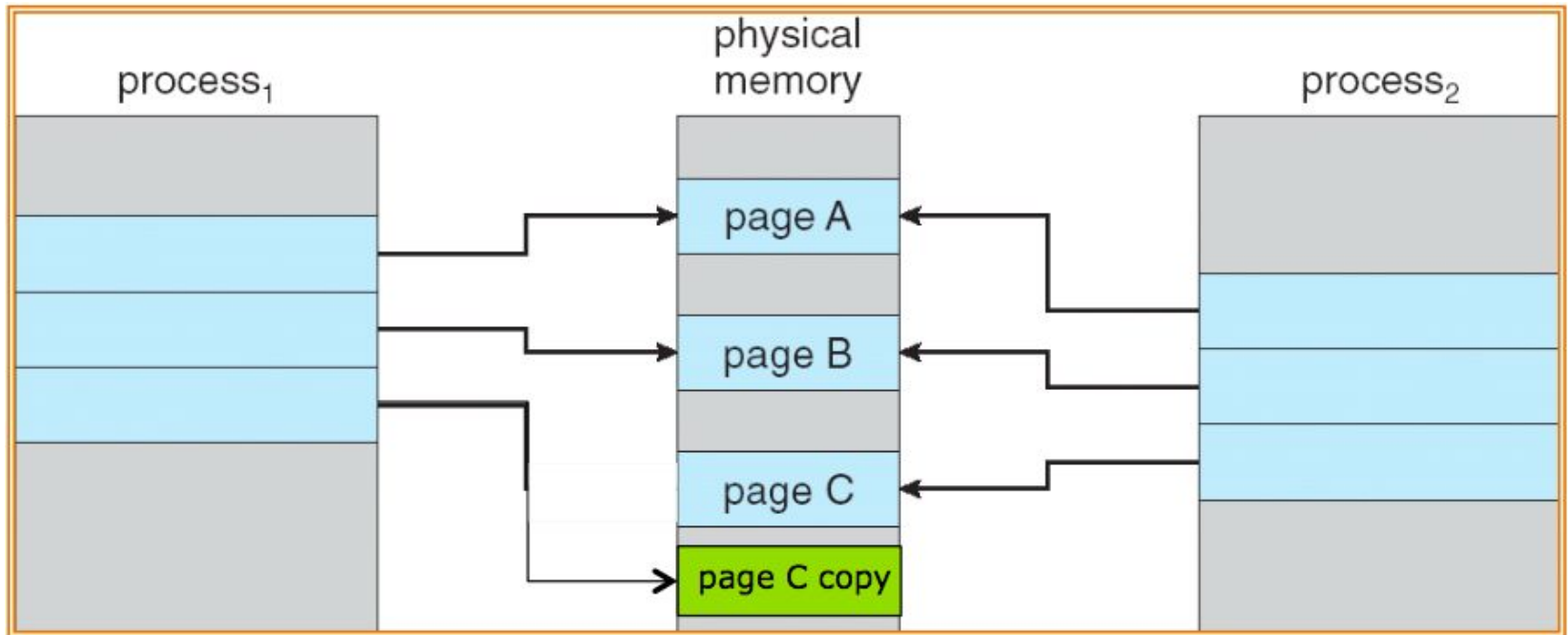
- A cópia na escrita permite criação de processo mais eficiente, pois somente páginas modificadas são copiadas
- Páginas livres são alocadas de um **pool** de páginas zeradas



Antes que processo 1 modifique página C



Depois que processo 1 modifica página C



O que acontece se não houver frame livre?

- Substituição de página – encontre alguma página na memória, mas se não estiver realmente em uso, move-a para fora (swap-out)
 - algoritmo
 - desempenho – deseja-se um algoritmo que resultará no mínimo de faltas de página
- Mesma página pode ser trazida para a memória várias vezes

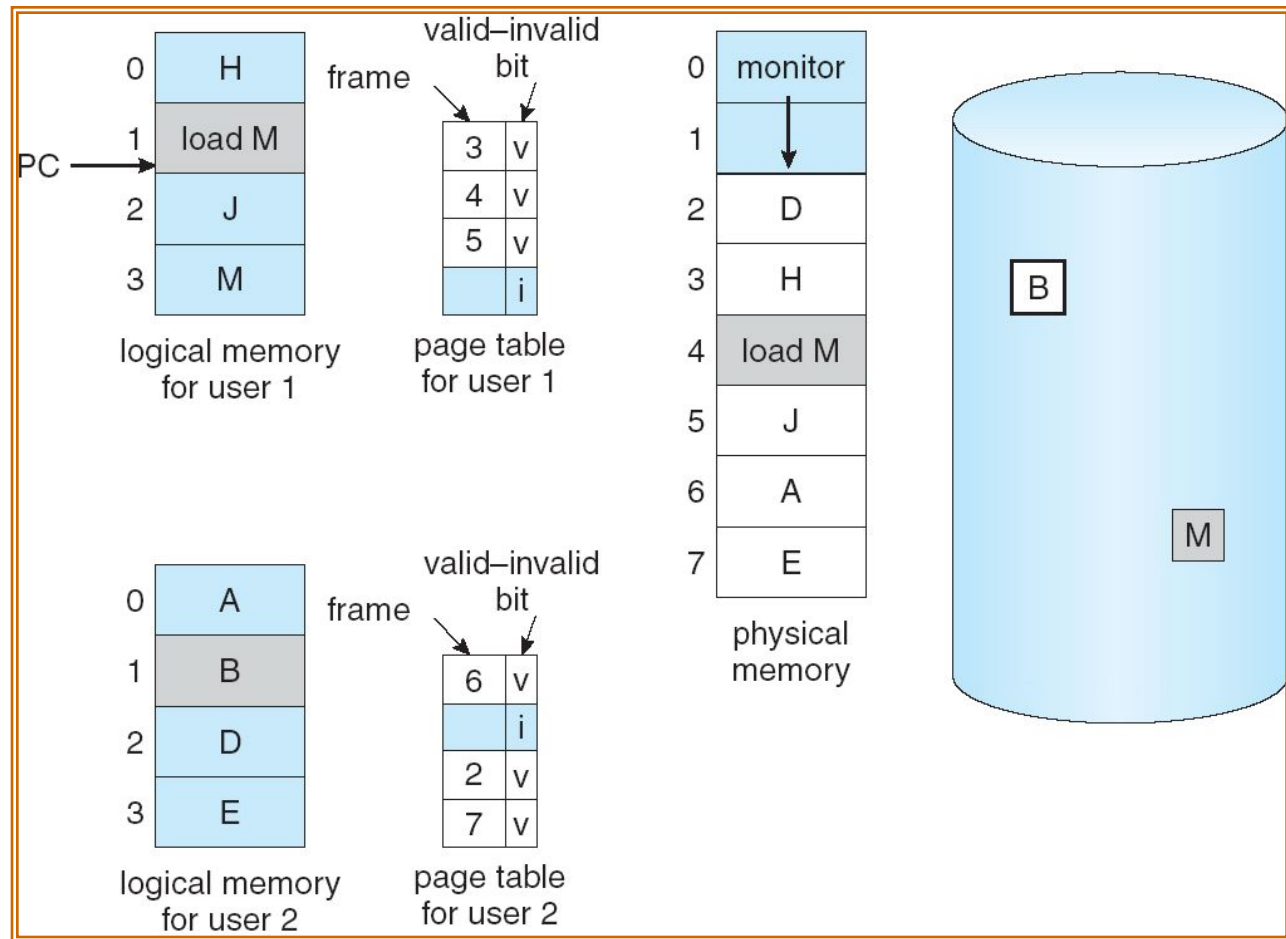


Substituição de página

- Impede superalocação de memória modificando rotina de serviço de falta de página para incluir substituição de página
- Use **bit de modificação (sujo)** para reduzir overhead das transferências de página – somente páginas modificadas são gravadas em disco
- A substituição de página completa a separação entre memória lógica e memória física – memória virtual grande pode ser fornecida em uma memória física menor



Necessidade de substituição de página

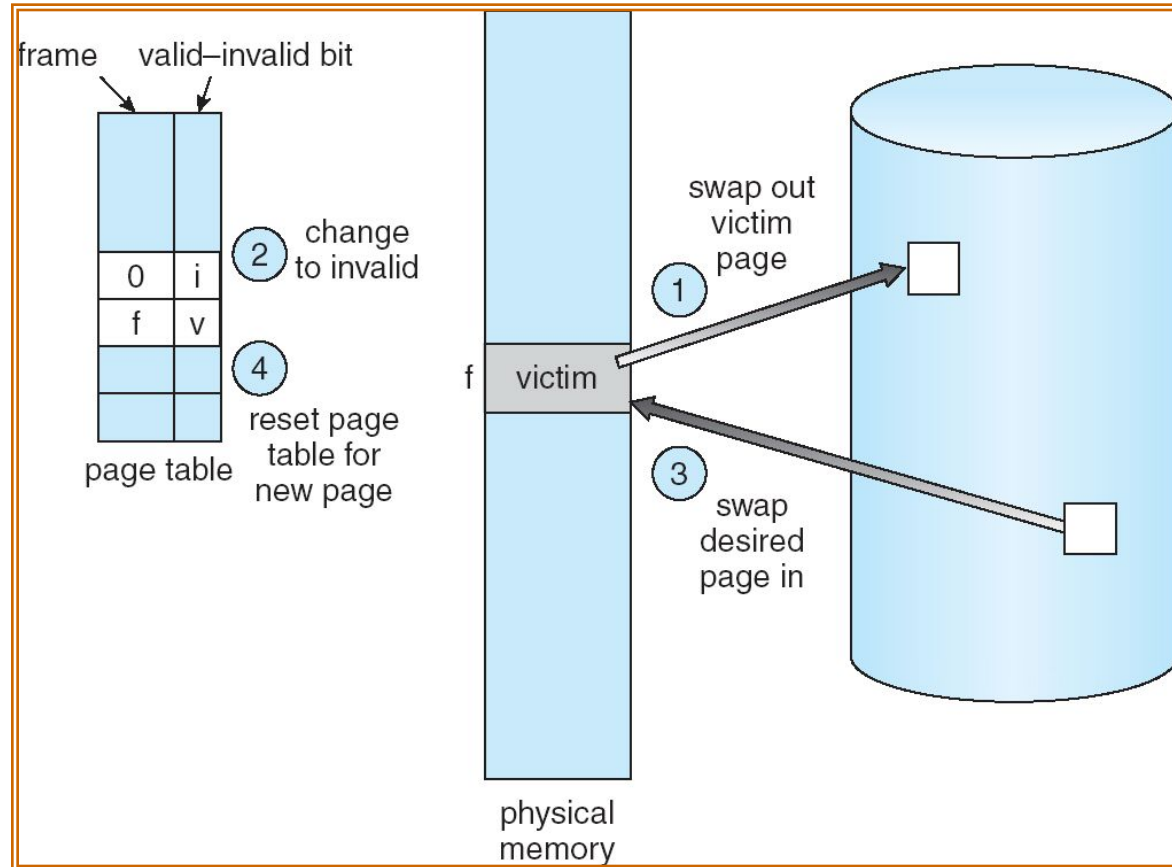


Substituição de página básica

1. Ache o local da página desejada no disco
2. Ache um quadro livre:
 - Se houver um quadro livre, use-o
 - Se não houver quadro livre, use um algoritmo de substituição de página para selecionar um quadro **vítima**
3. Traga a página desejada para o quadro (recém) livre; atualize a página e as tabelas de quadro
4. Reinicie o processo



Substituição de página



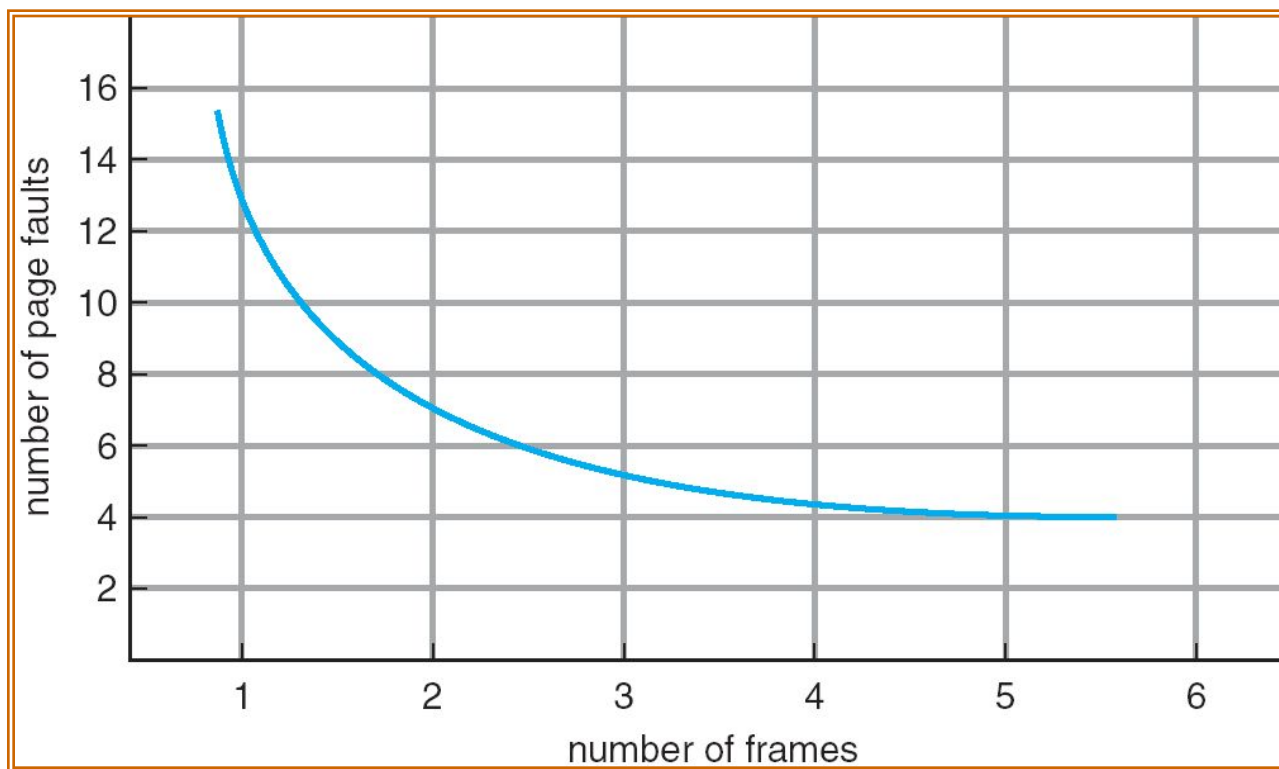
Algoritmos de substituição de página

- Deseja taxa de falta de página mais baixa
- Avalia algoritmo executando-o em uma string em particular de referências de memória (string de referência) e calculando o número de faltas de página nessa string
- Em todos os nossos exemplos, a string de referência é

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



Gráfico de faltas de página versus número de quadros



Algoritmo First-In-First-Out (FIFO)

- String de referência: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 quadros (3 páginas podem estar na memória ao mesmo tempo por processo)

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

9 faltas de página

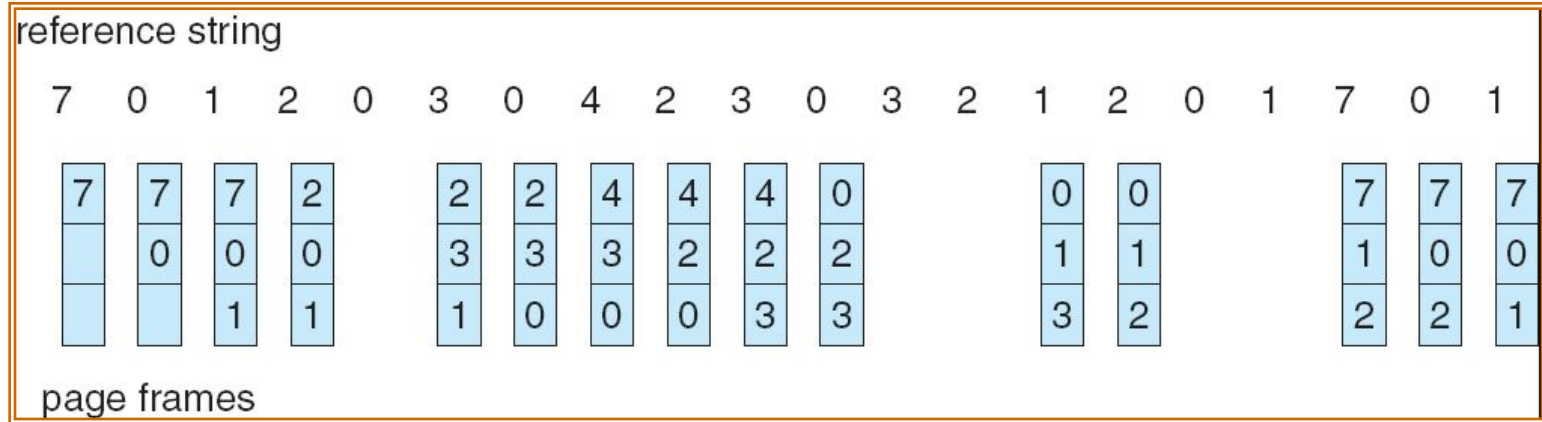
- 4 frames

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

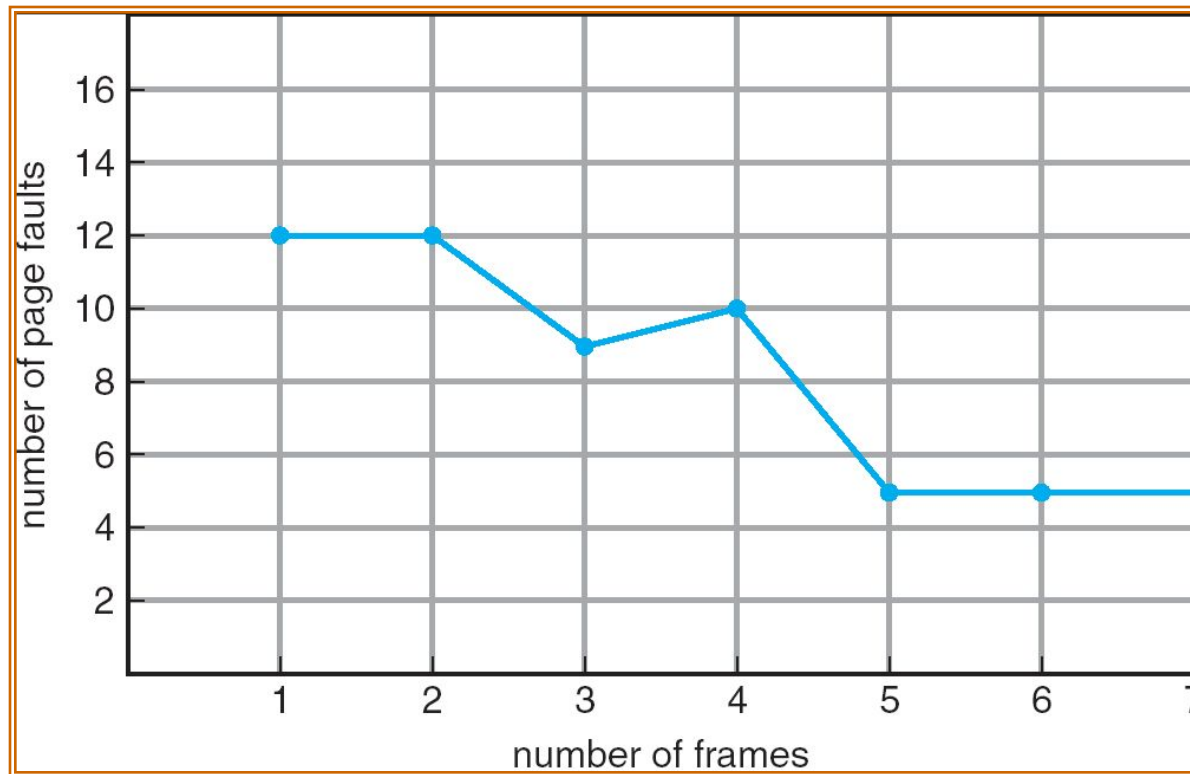
10 faltas de página



Substituição de página FIFO



FIFO ilustrando anomalia de Belady



Algoritmo ideal

- ❑ Substitua página que não será usada pelo maior período de tempo
- ❑ Exemplo de 4 frames
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1
2
3
4

4

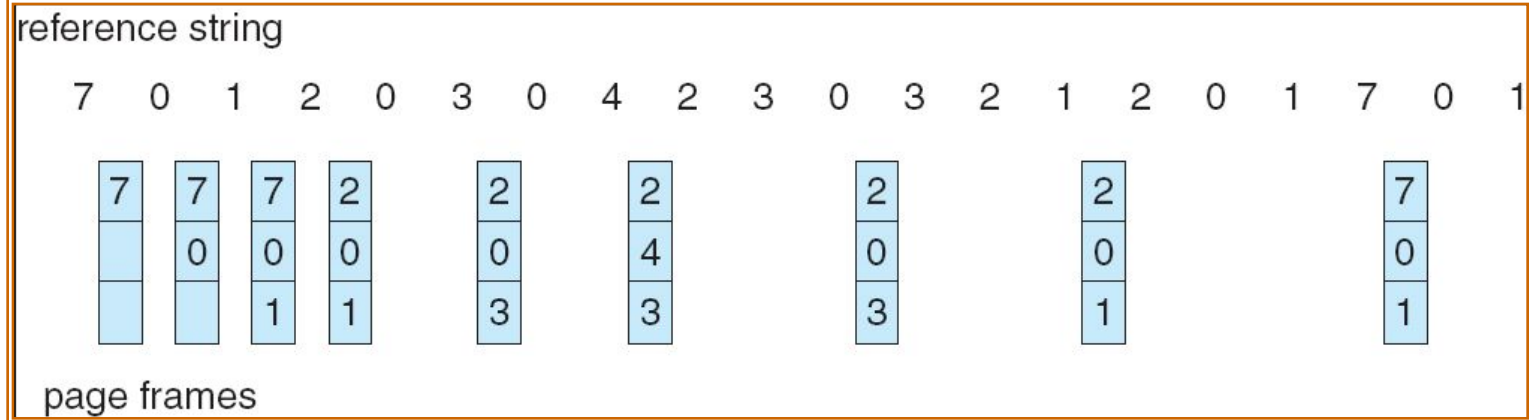
6 faltas de
página

5

- ❑ Como você sabe disto?
- ❑ Usado para medir como seu algoritmo funciona



Substituição de página ideal



Algoritmo Least Recently Used (LRU)

- String de referência: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

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

- Implementação do contador
 - Cada entrada de página tem um contador; toda vez que a página é referenciada por essa entrada, copia o clock para o contador
 - Quando uma página precisa ser mudada, veja os contadores para determinar quais devem mudar



Substituição de página LRU

reference string

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

page frames

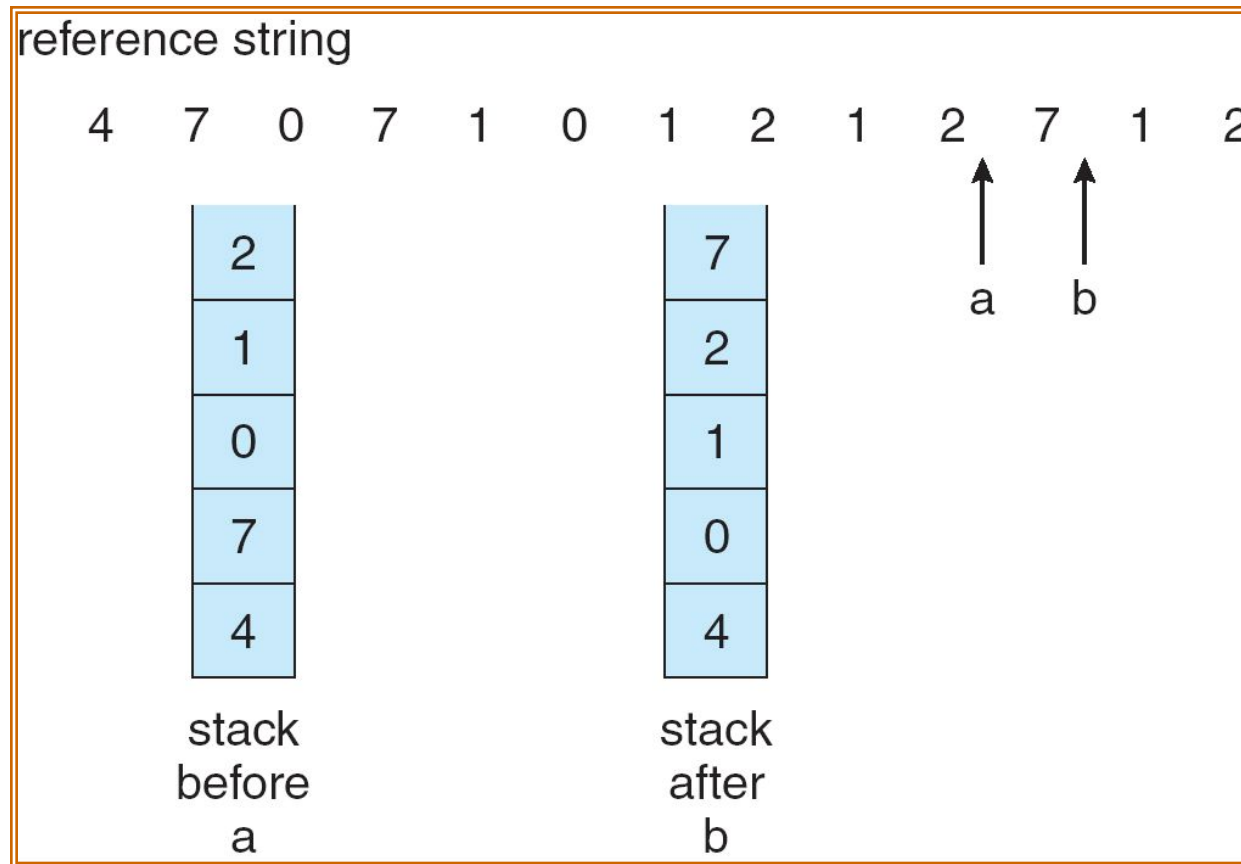


Algoritmo LRU (cont.)

- Implementação de pilha – mantenha uma pilha de números de página em um formato de link duplo:
 - Página referenciada:
 - mova-a para o topo
 - requer que 6 ponteiros sejam trocados
 - Nenhuma busca para substituição



Uso de uma pilha para registrar as referências de página mais recentes



Algoritmos de aproximação LRU

□ Bit de referência

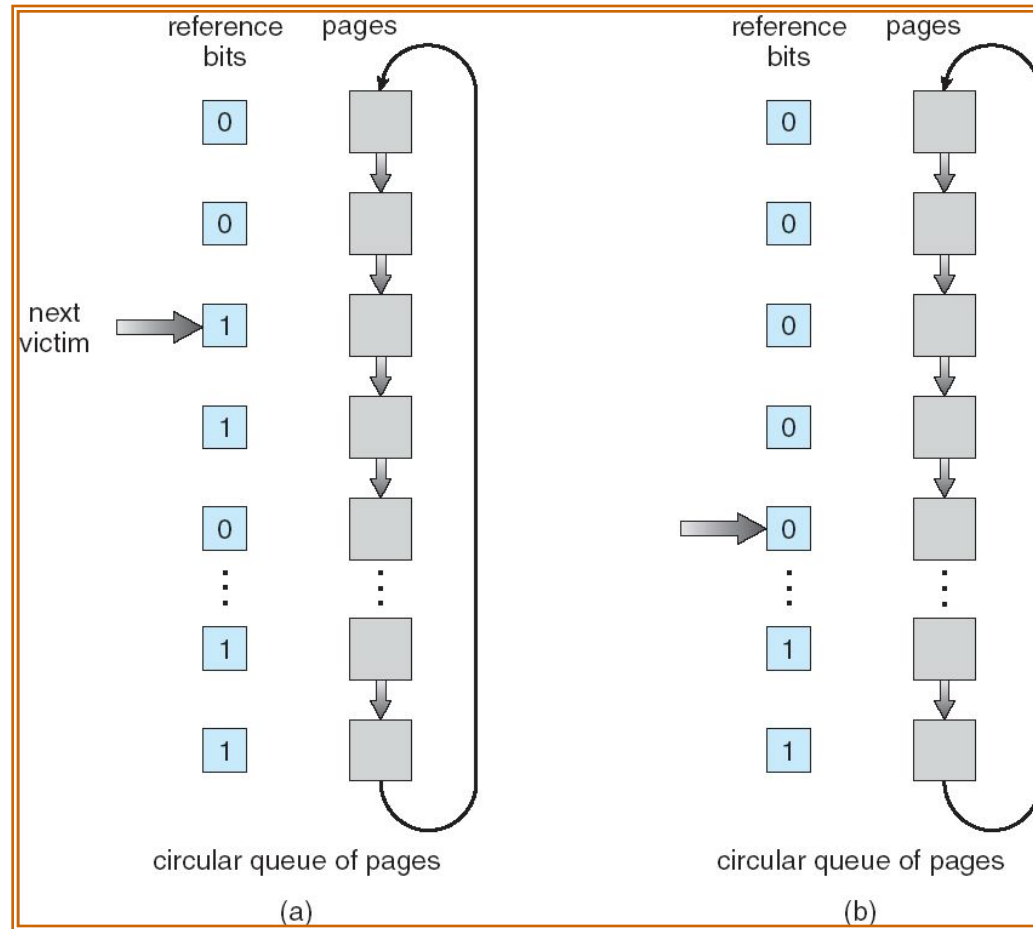
- A cada página associe um bit, inicialmente = 0
- Quando a página é referenciada, bit passa para 1
- Substitua a página que é 0 (se uma existir)
 - Não sabemos a ordem

□ Segunda chance

- Precisa de bit de referência
- Substituição relógio
- Se página a ser substituída (na ordem do relógio) tem bit de referência = 1 então:
 - define bit de referência 0
 - deixa página na memória
 - substitui próxima página (na ordem do relógio), sujeito às mesmas regras



Algoritmo de substituição de página da mesma chance (relógio)



Algoritmos de contagem

- ❑ Mantenha um contador do número de referências que foram feitas a cada página
- ❑ **Algoritmo LFU:** substitui página por menor contador
- ❑ **Algoritmo MFU:** baseado no argumento de que a página com a menor contagem provavelmente acabou de ser trazida e ainda está para ser usada



Alocação de quadros

- Cada processo precisa do número *mínimo* de páginas
- Exemplo: IBM 370 – 6 páginas para tratar da instrução SS MOVE:
 - instrução tem 6 bytes, pode espalhar por 2 páginas
 - 2 páginas para tratar *de*
 - 2 páginas para tratar *para*
- Dois esquemas de alocação principais
 - alocação fixa
 - alocação por prioridade



Alocação fixa

- Alocação igual – Por exemplo, se houver 100 quadros e 5 processos, dá a cada processo 20 quadros.
- Alocação proporcional – Aloca de acordo com o tamanho do processo

— s_i = tamanho do processo p_i
— $S = \sum s_i$
— m = número total de quadros
— a_i = alocação para $p_i = \frac{s_i}{S} \times m$

$$m = 64$$

$$s_1 = 10$$

$$s_2 = 127$$

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

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



Alocação por prioridade

- Usa um esquema de alocação proporcional ao invés de tamanho
- Se processo P_i gera uma falta de página,
 - seleciona para substituição um de seus quadros
 - seleciona para substituição um quadro de um processo com número de prioridade menor



Alocação global versus local

- ▣ **Substituição global** – processo seleciona um quadro de substituição do conjunto de todos os quadros; um processo pode apanhar um quadro de outro
- ▣ **Substituição local** – cada processo seleciona apenas do seu próprio conjunto de quadros alocados

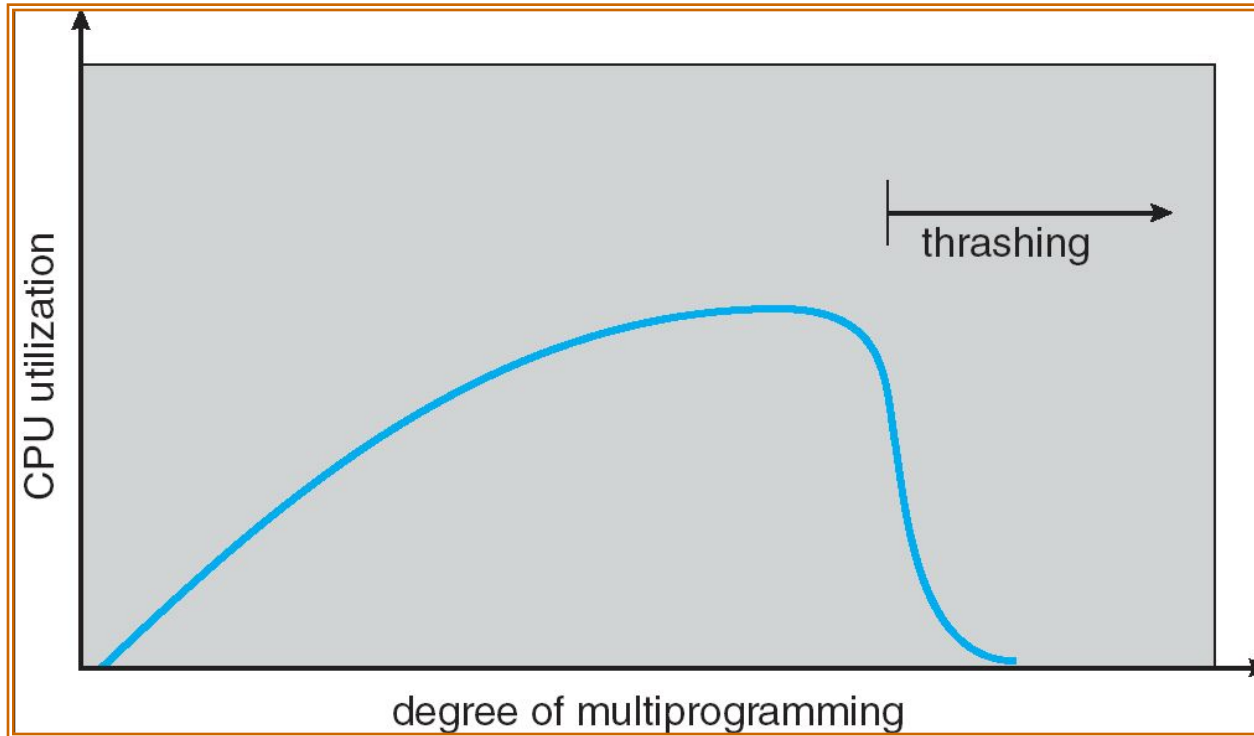


Thrashing

- Se um processo não tem páginas “suficientes”, a taxa de falta de página é muito alta. Isso leva a:
 - baixa utilização de CPU
 - sistema operacional pensa que precisa aumentar o grau de multiprogramação
 - outro processo acrescentado ao sistema
- **Thrashing** \equiv um processo está ocupado trocando páginas para dentro e pra fora



Thrashing (cont.)



Paginação por demanda e thrashing

- Por que a paginação por demanda funciona?

Modelo de localidade

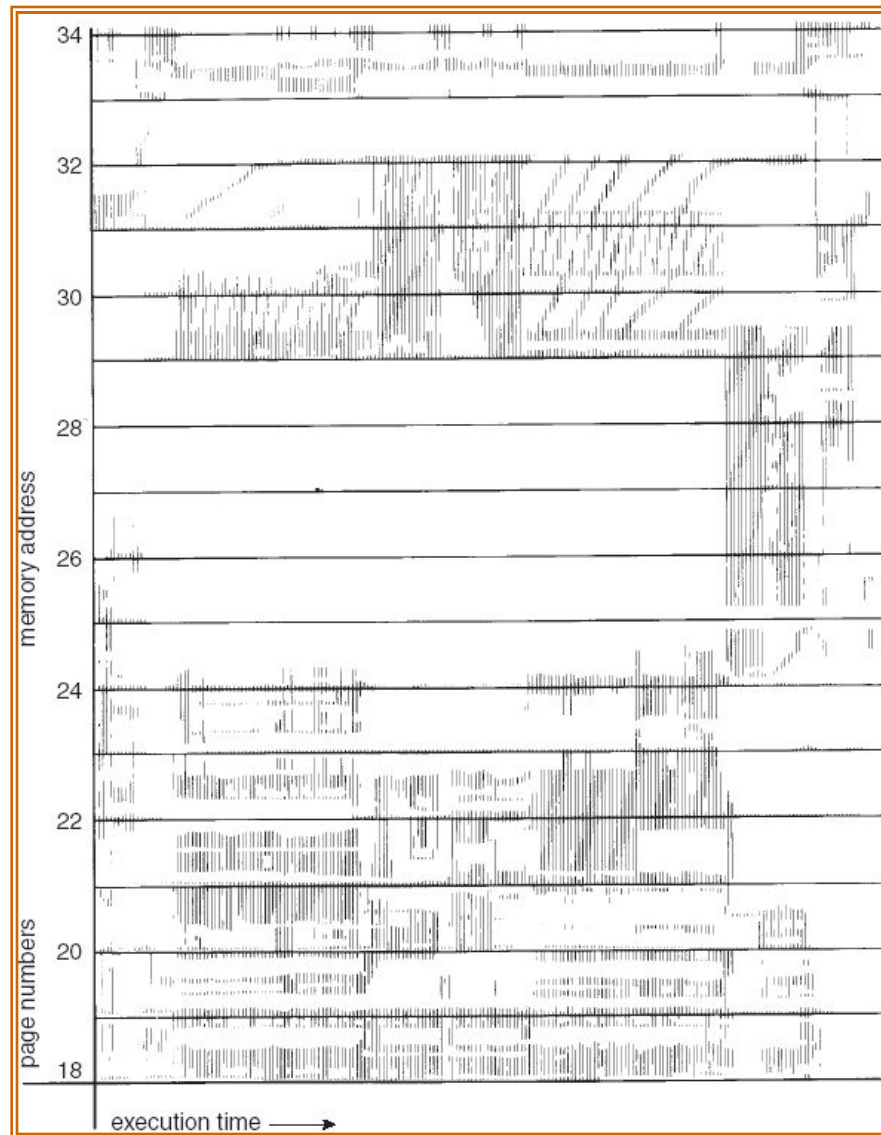
- Processo migra de uma localidade para outra
- Localidades podem se sobrepor

- Por que ocorre o thrashing?

Σ tamanho da localidade > tamanho total da memória



Localidade em um padrão de referência de memória



Modelo de conjunto de trabalho

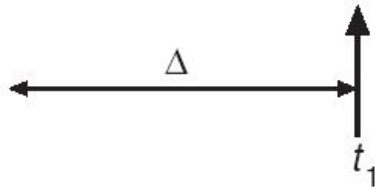
- $\Delta \equiv$ janela conjunto de trabalho \equiv um número fixo de referências de página
Exemplo: 10.000 instruções
- WSS_i (conjunto de trabalho do Processo P_i) = número total de páginas referenciadas no Δ mais recente (varia no tempo)
 - se Δ muito pequeno, não abrangerá localidade inteira
 - se Δ muito grande, abrangerá várias localidades
 - se $\Delta = \infty \Rightarrow$ abrangerá programa inteiro
- $D = \sum WSS_i \equiv$ total de quadros por demanda
- se $D > m \Rightarrow$ Thrashing
- Política se $D > m$, então suspenda um dos processos



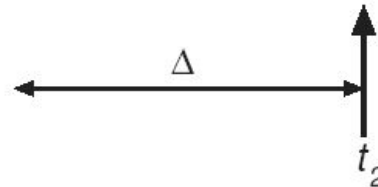
Modelo de conjunto de trabalho

page reference table

... 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...



$WS(t_1) = \{1, 2, 5, 6, 7\}$



$WS(t_2) = \{3, 4\}$



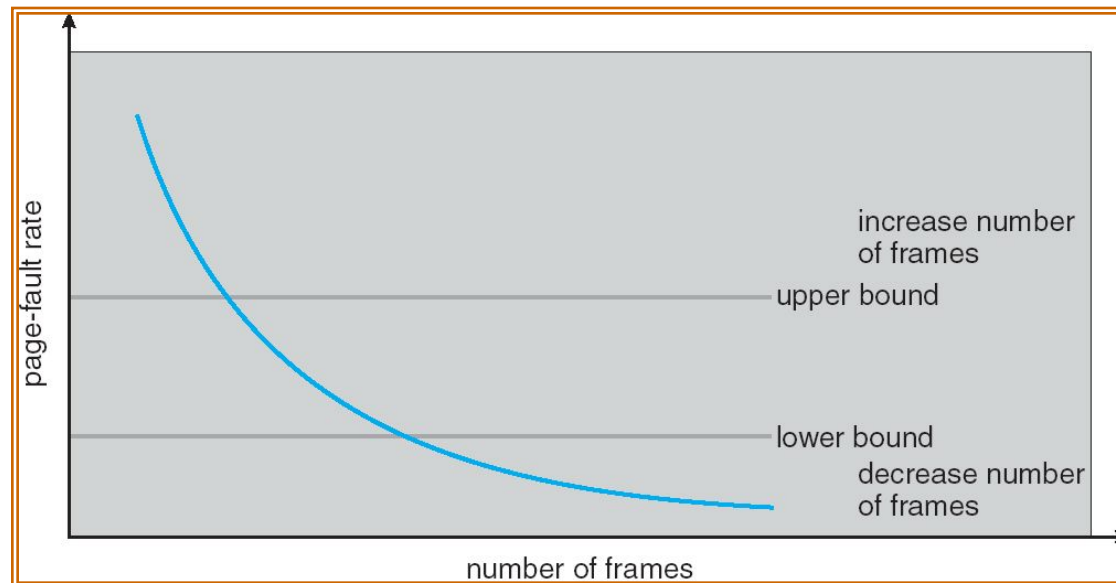
Acompanhando o conjunto de trabalho

- Aproximado com timer de intervalo + um bit de referência
- Exemplo: $\Delta = 10.000$
 - Interrupções de timer após cada 5000 unidades de tempo
 - Mantém na memória 2 bits para cada página
 - Sempre que um timer interrompe a cópia e define os valores de todos os bits de referência como 0
 - Se um dos bits na memória = 1 \Rightarrow página não conjunto de trabalho
- Por que isso não é completamente preciso?
- Melhoria = 10 bits e interrupção a cada 1000 unidades de tempo



Esquema de frequência de falta de página

- Estabelece taxa de falta de página “aceitável”
 - Se a taxa real for muito baixa, processo perde quadro
 - Se a taxa real for muito alta, processo ganha quadro

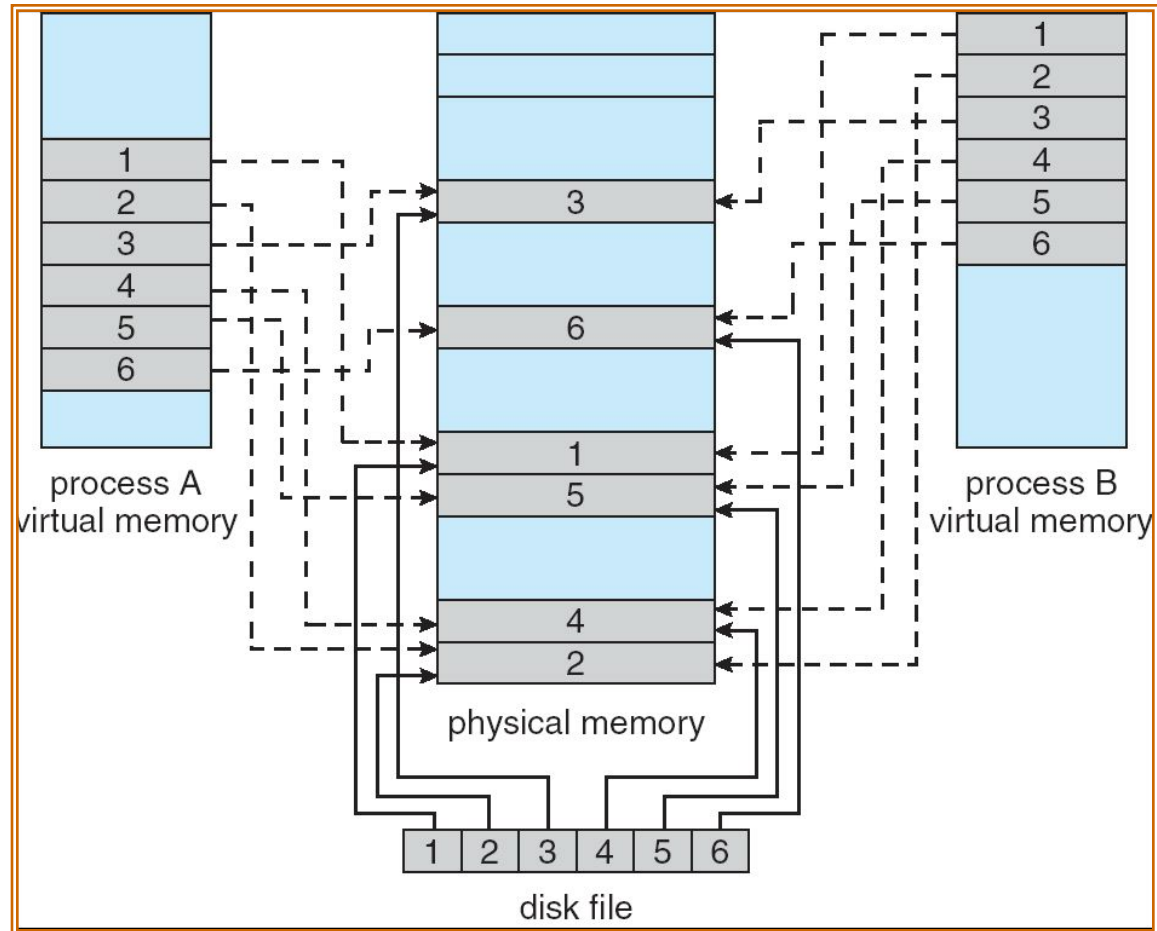


Arquivos mapeados na memória

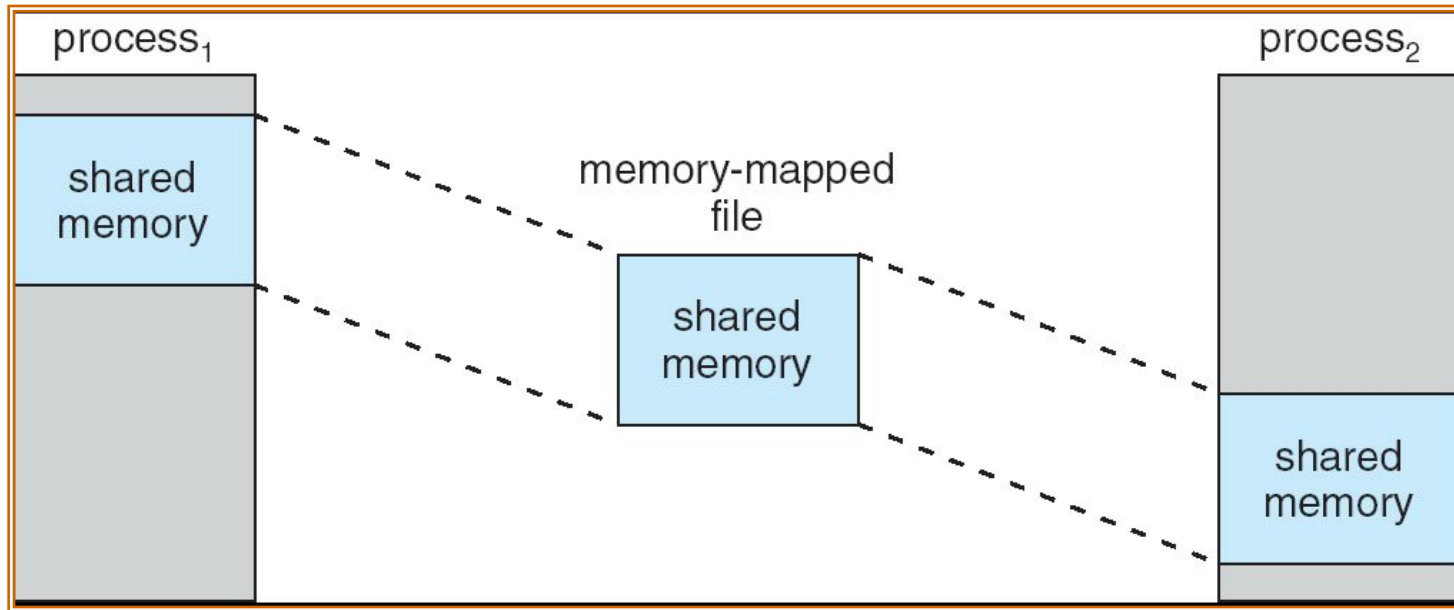
- E/S de arquivo mapeado na memória permite que a E/S de arquivo seja tratada como acesso de rotina à memória, **mapeando** um bloco de disco para uma página na memória
- Um arquivo é lido inicialmente usando paginação por demanda. Uma parte do arquivo com tamanho da página é lida do sistema para uma página física. Leituras/escritas subseqüentes de/para o arquivo são tratadas como acessos comuns à memória.
- Simplifica o acesso ao arquivo, tratando a E/S do arquivo por meio da memória, ao invés das chamadas do sistema **read()** e **write()**
- Também permite que vários processos sejam mapeados para o mesmo arquivo, permitindo que as páginas na memória sejam compartilhadas



Arquivos mapeados na memória



Memória compartilhada mapeada na memória no Windows



Arquivos mapeados na memória em 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;

        // 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();
    }
}
```



Alocando memória do kernel

- Tratado diferente da memória do usuário
- Normalmente alocado por um pool de memória livre
 - Kernel solicita memória para estruturas de tamanhos variáveis;
 - Alguns dispositivos de hardware interagem diretamente com a memória, sem benefício da interface da memória virtual, exigindo que a memória seja contígua.

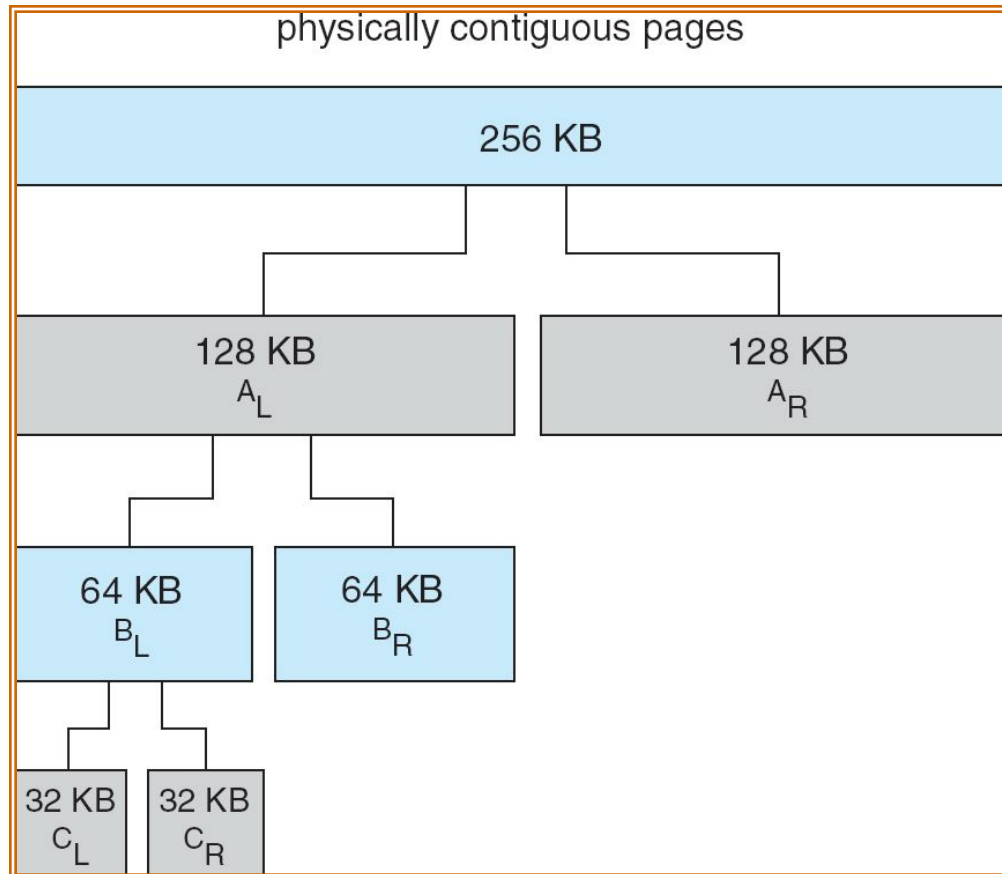


Sistema buddy

- Aloca memória a partir de segmento de tamanho fixo, consistindo em páginas de memória fisicamente contíguas
- Memória alocada usando **alocador de potência de 2 (4KB, 8KB, 16KB ...)**
 - Satisfaz requisitos em unidades com tamanho de potência de 2
 - Requisição arredondada para próxima potência de 2 mais alta
 - Quando precisa de alocação menor que o disponível, pedaço atual dividido em dois buddies da próxima potência de 2 menor
 - Continua até haver chunk de tamanho apropriado



Alocador de sistema buddy

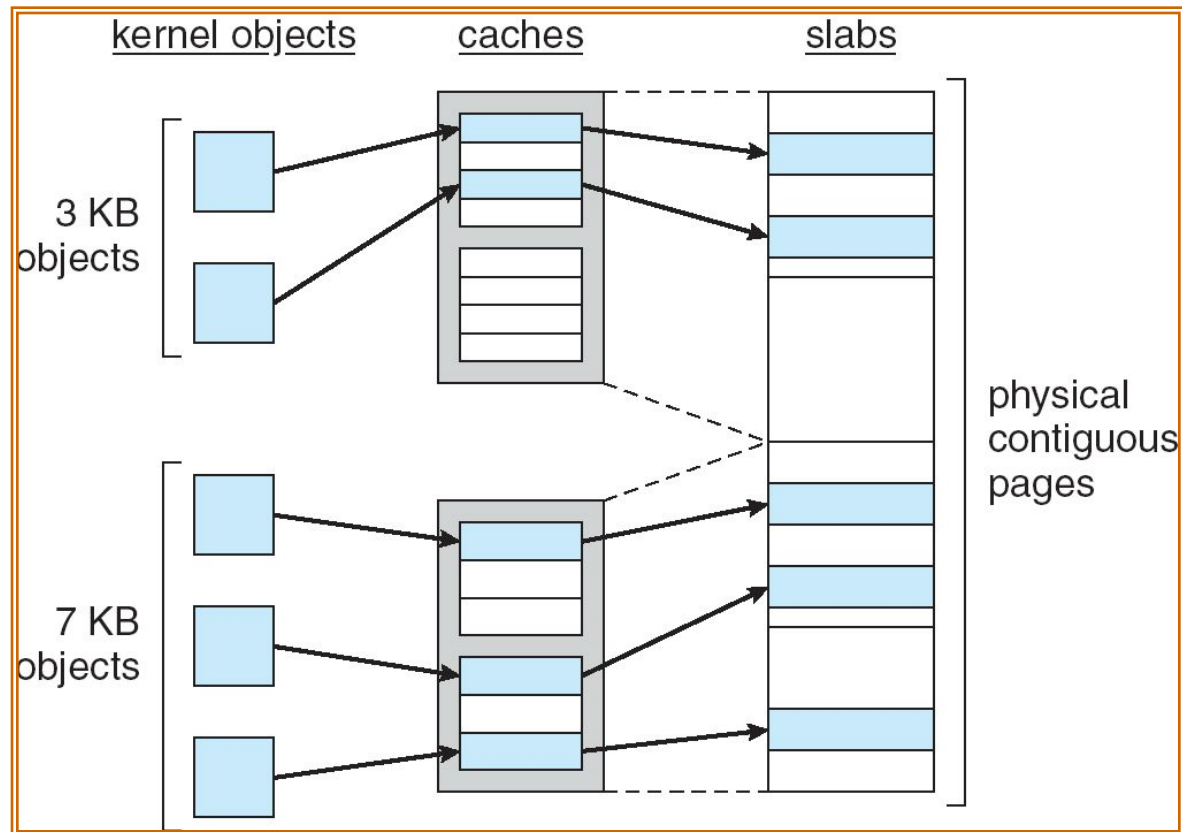


Alocador de slab

- Estratégia alternativa
- **Slab** é uma ou mais páginas fisicamente contíguas
- **Cache** consiste em uma ou mais slabs
- Cache único para cada estrutura de dados exclusiva do kernel
 - Cada cache preenchido com **objetos** – instâncias da estrutura de dados
- Quando cache criado, preenchido com objetos marcados com **free**
- Quando estruturas armazenadas, objetos marcados como **used**
- Se slab cheio de objetos usados, próximo objeto alocado a partir de slab vazia
 - Se nenhuma slab vazia, nova slab alocada
- Benefícios incluem nenhuma fragmentação, satisfação rápida da requisição de memória



Alocação de slab



Outras questões – pré-paginação

□ Pré-paginação

- Reduzir o grande número de faltas de página que ocorre no início do processo
- Pré-paginar todas ou algumas das páginas que um processo precisará.
- Mas se as páginas pré-paginadas não forem usadas, E/S e memória foram desperdiçadas
- Suponha que s páginas sejam preparadas e α das páginas seja usado ($0 \leq \alpha \leq 1$)
 - O custo de $s * \alpha$ faltas de página de salvamento $>$ ou $<$ que o custo de preparar $s * (1 - \alpha)$ páginas desnecessárias?
 - α perto de zero \Rightarrow pré-paginação perde



Outras questões – tamanho de página

- Coleção de tamanho de página deve levar em consideração:
 - fragmentação
 - tamanho da tabela
 - overhead de E/S
 - localidade



Outras questões – Alcance do TLB

- Alcance do TLB – A quantidade de memória acessível pelo TLB
- Alcance do TLB = (Tamanho TLB) X (Tamanho da página)
- Ideal que o conjunto de trabalho de cada processo seja armazenado no TLB
 - Caso contrário, há um alto grau de faltas de página
- Aumenta o tamanho da página
 - Isso pode levar a um aumento na fragmentação, pois nem todas as aplicações exigem um tamanho de página grande
- Oferece múltiplos tamanhos de página
 - Isso permite às aplicações que exigem tamanhos de página maiores oportunidade de usá-las sem aumento na fragmentação



Outras questões – Estrutura do programa

□ Estrutura do programa

- `int[128,128] data;`
- Cada linha é armazenada em uma página
- Programa 1

```
for (j = 0; j < 128; j++)  
    for (i = 0; i < 128; i++)  
        data[i,j] = 0;
```

128 x 128 = 16.384 faltas de página

■ Programa 2

```
for (i = 0; i < 128; i++)  
    for (j = 0; j < 128; j++)  
        data[i,j] = 0;
```

128 faltas de página

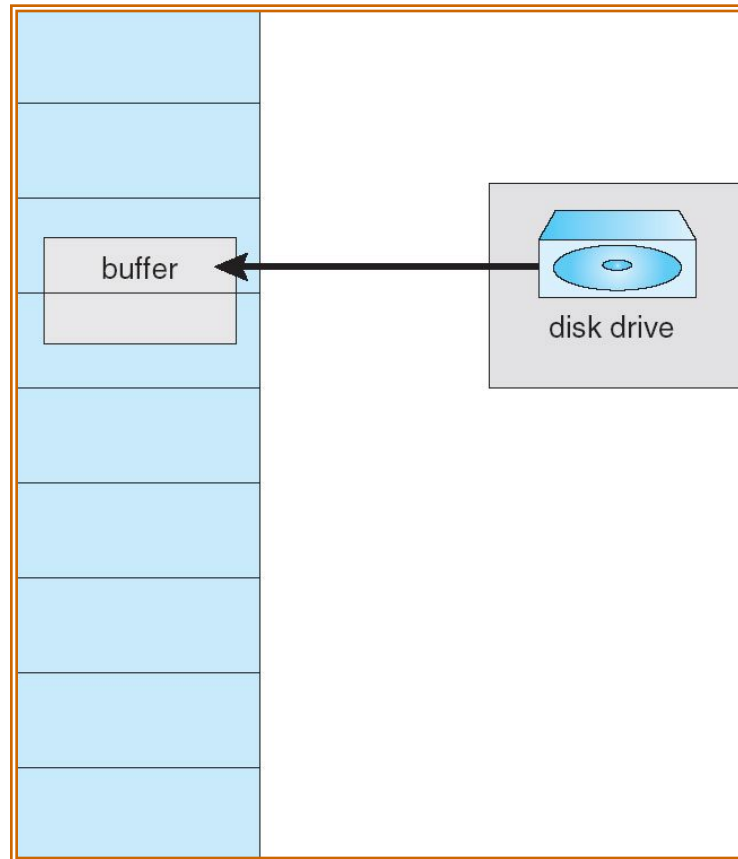


Outras questões – interlock de E/S

- ❑ **Interlock de E/S** – Páginas às vezes precisam ser bloqueadas na memória
- ❑ Considere E/S – Páginas que são usadas para copiar um arquivo de um dispositivo precisam evitar serem selecionadas para evicção por um algoritmo de substituição de página



Motivo para os quadros usados para E/S estarem na memória



Exemplos de sistema operacional

- Windows XP
- Solaris



Windows XP

- ❑ Usa paginação por demanda com **clustering**. Clustering traz páginas ao redor da página que falta.
- ❑ Os processos recebem **mínimo do conjunto de trabalho e máximo do conjunto de trabalho**
- ❑ Mínimo do conjunto de trabalho é o número mínimo de páginas que o processo tem garantia de ter na memória
- ❑ Um processo pode receber até o número máximo de páginas do seu conjunto de trabalho
- ❑ Quando a quantidade de memória livre no sistema ficar abaixo de um patamar, o **trimming automático do conjunto de trabalho** é realizado para restaurar a quantidade de memória livre
- ❑ O trimming do conjunto de trabalho remove páginas dos processos que possuem páginas em excesso do seu mínimo do conjunto de trabalho

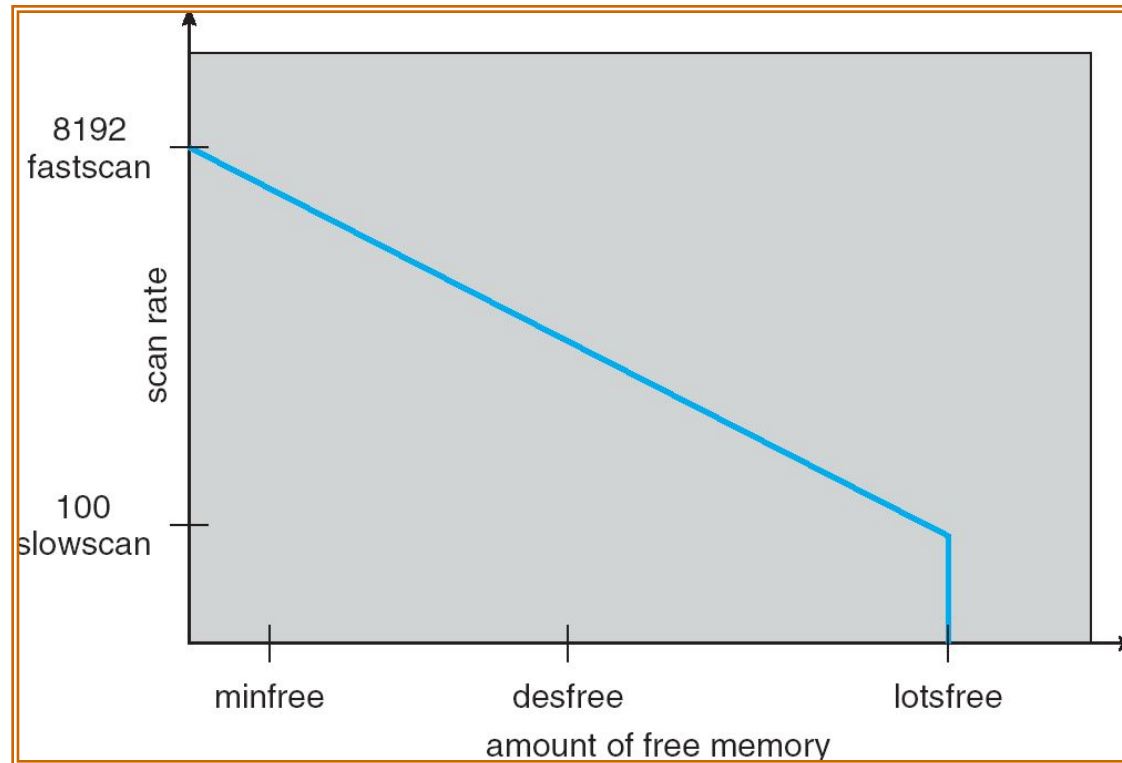


Solaris

- ❑ Mantém uma lista de páginas livres para atribuir processos com falta
- ❑ *Lotsfree* – parâmetro de patamar (quantidade de memória livre) para iniciar paginação
- ❑ *Desfree* – parâmetro de patamar para aumentar paginação
- ❑ *Minfree* – parâmetro de patamar para iniciar swapping
- ❑ Paginação é realização por processo *pageout*
- ❑ Pageout varre páginas usando algoritmo de relógio modificado
- ❑ *Scanrate* é a taxa em que as páginas são varridas. Isso varia de *slowscan* até *fastscan*
- ❑ Pageout é chamado com mais frequência, dependendo da quantidade de memória disponível



Scanner de página do Solaris 2



Final do Capítulo 9

