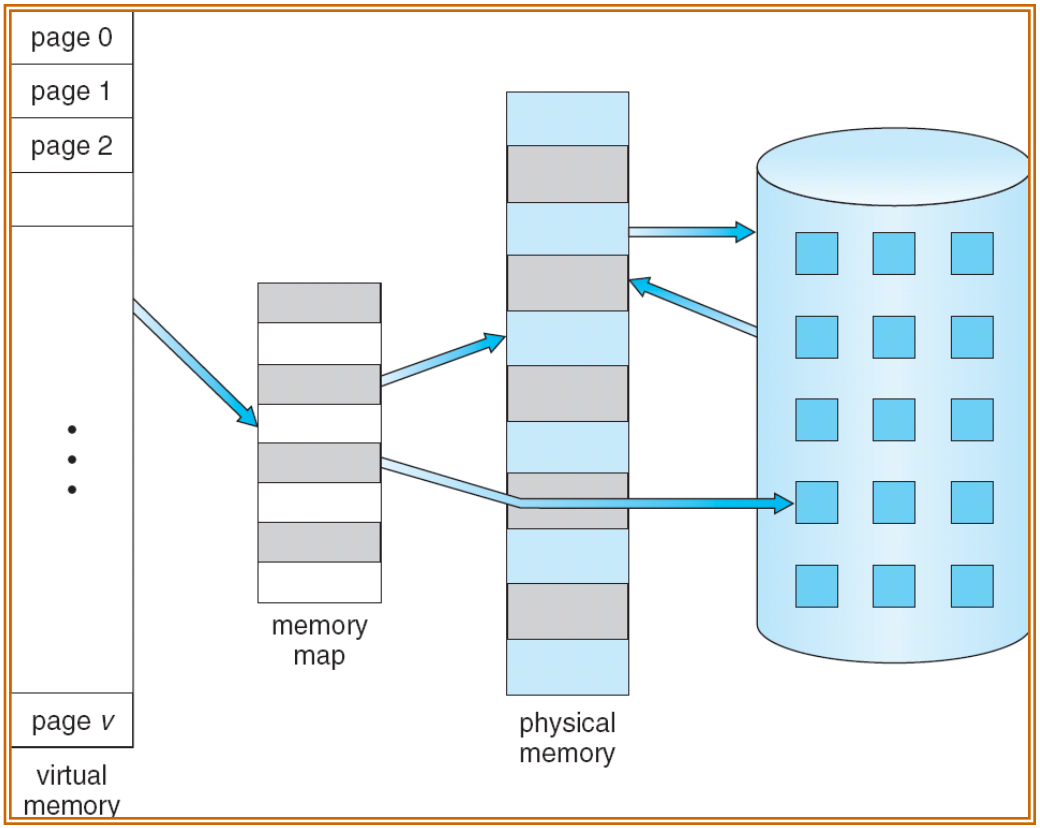
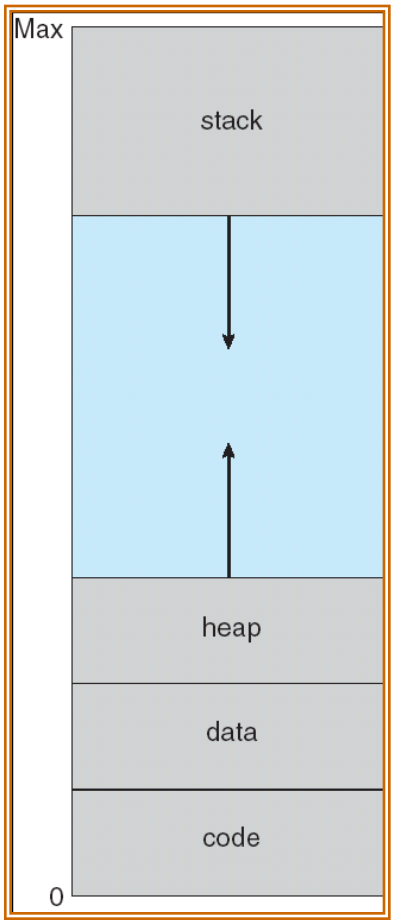
Sistemas Operacionais: Memória Virtual

Memória virtual

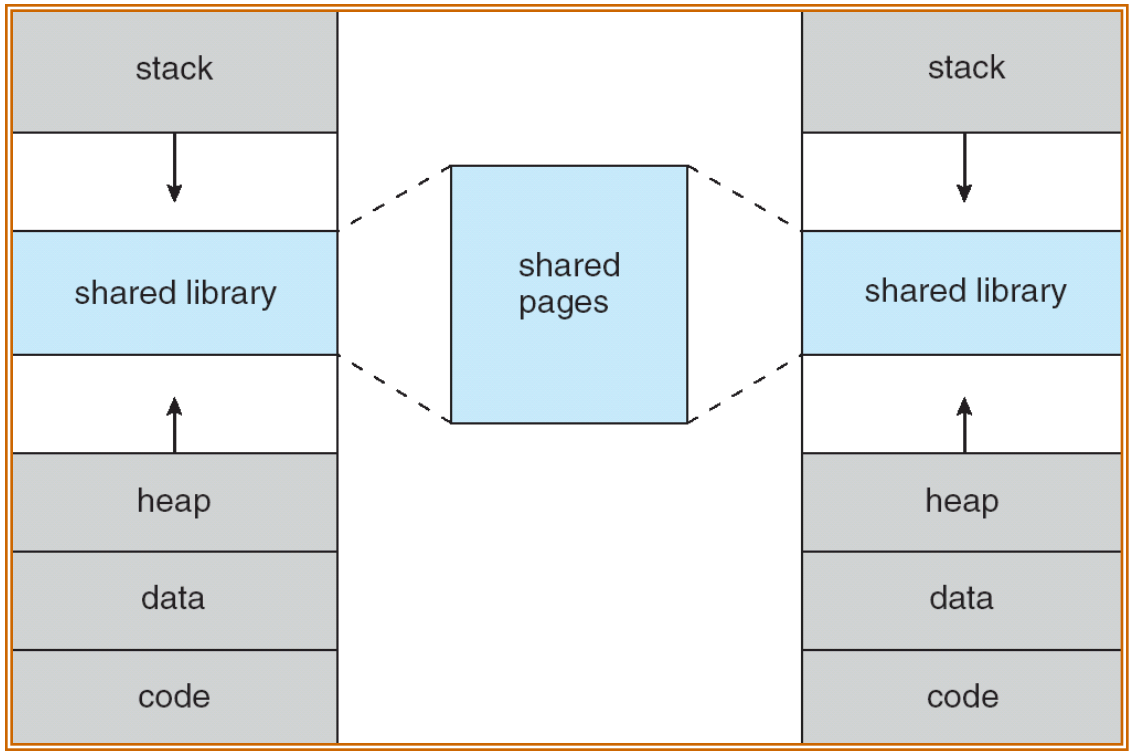
- Memória virtual: separação entre a visão lógica do usuário e a memória física
  - Somente uma parte do programa necessita estar na memória para executar
  - O espaço de endereçamento lógico pode ser maior que o espaço físico da memória
  - Permite que o mesmo espaço de endereçamento seja compartilhado por vários processos
- Memória virtual implementada por
  - Paginação por demanda
  - Segmentação por demanda

Memória virtual





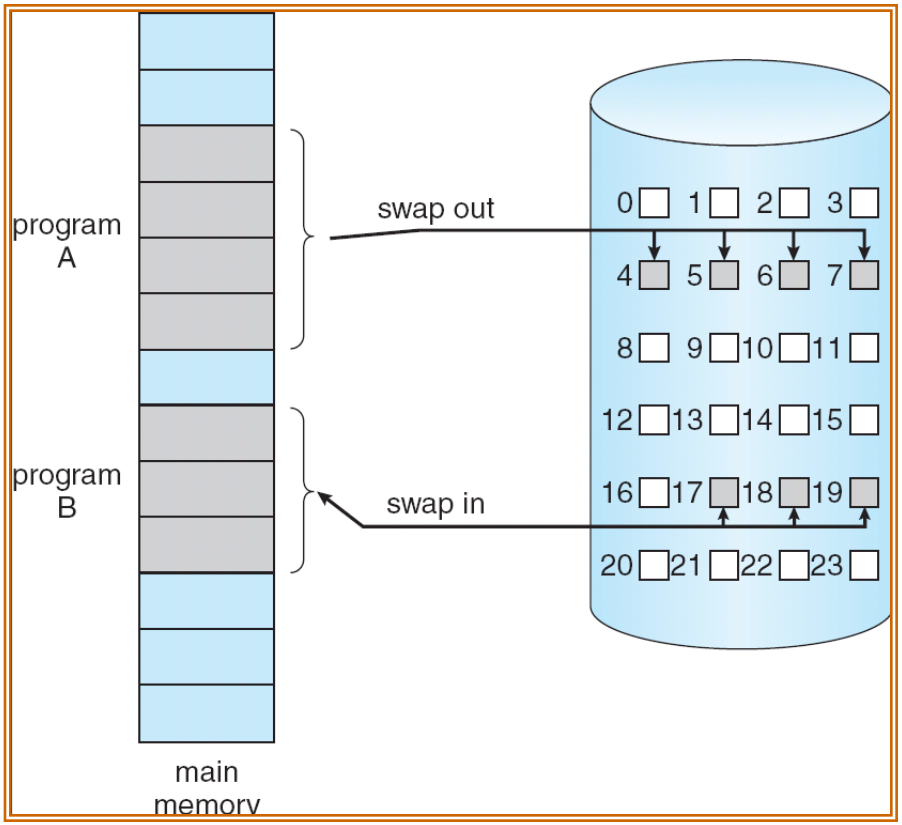
Bibliotecas compartilhadas



Paginação por demanda

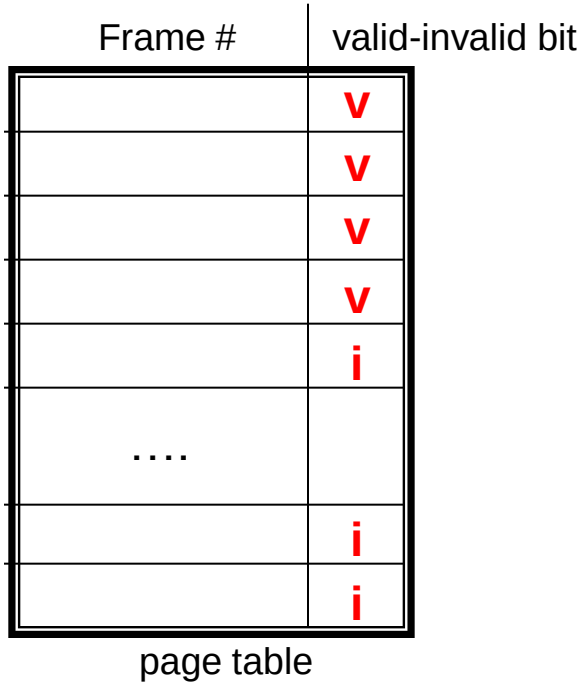
- Carregar uma página na memória somente quando ela é necessária
  - Menor número de operações de E/S
  - Menor ocupação da memória
  - Resposta mais rápida
  - Mais usuários
- Quando uma página é necessária
  - Referência inválida-> interrupção de erro
  - Armazenada no disco -> carregar na memória
- Paginador “preguiçoso”
  - Carrega uma memória somente quando necessário

Swap in- Swap out

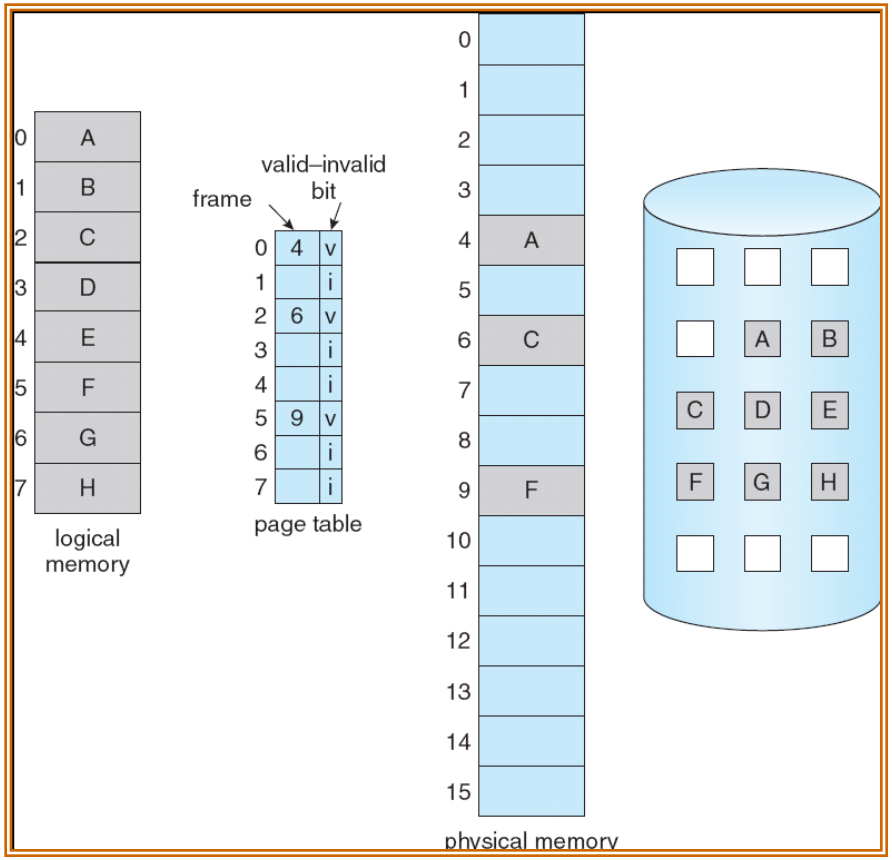


Bit válido-inválido

- Bit inválido-> residente no disco
- Inicialmente todos os bits inválidos
- Tradução de endereços -> falha de página (*page fault*)



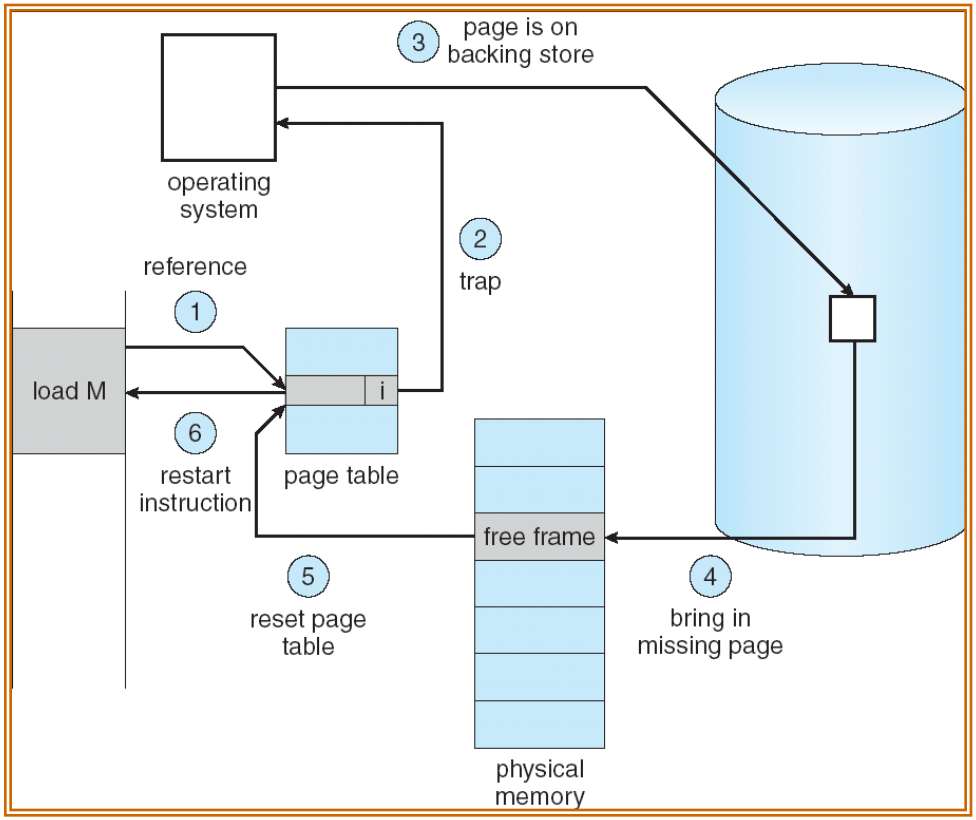
Páginas armazenadas no disco



Falha de página (*page fault*)

- Na primeira referência à página
  - *Trap* para o sistema operacional
- Se a referência:
  - Inválida: erro
  - Carregar a página para a memória
    - Obter um frame livre
    - Carregar a página no frame
    - Atualizar a tabela de página (bit válido)
    - Reinicializar a instrução que gerou o *trap*

Page Fault


3
page is on
backing store
operating
system
2
trap
reference
1
load M
i
6
restart
instruction
page table
free frame
5
reset page
table
4
bring in
missing page
physical
memory

Desempenho do sistema: paginação por demanda

- Taxa de falha de página: $0 \leq p \leq 1.0$
  - Se p=0 sem falhas de páginas
  - Se p=1 todo acesso gera uma falha de página
- Tempo efetivo de acesso

$$t_{ea}=(1-p)\ t_{mem}+p\ (t_{page\_fault}+t_{swap\_in}+t_{swap\_out}+t_{restart})$$

Exemplo

- t_{mem}=200 ns
- $t_{page\_fault} + t_{swap\_in} + t_{swap\_out} + t_{restart} = 8$ ms

- $t_{ea=} (1 – p)$ x 200 + p x 8,000,000
- t_{ea}= 200 + p x 7,999,800
  - Se p= 0.001 (1 falha de página a cada 1000 acessos)
- t_{ea}= 8.2 ms

Criação de processos

- Copy-on-write
  - Processos pai e filho compartilham a mesma página, somente quando algum dado é alterado uma nova cópia é criada
  - Eficiente: menor utilização da memória

9.3 Substituição de páginas

- “Uma página poderá causar somente uma falha de página”
  - A afirmativa acima não é verdadeira em todos os casos
  - Ex. 60 frames na memória física
  - 7 processos executando e ocupando 10 frames
- Um processo em execução causa uma falha de página
  - o sistema operacional verifica se o endereço é válido
  - busca um frame livre na memória

Substituição de páginas

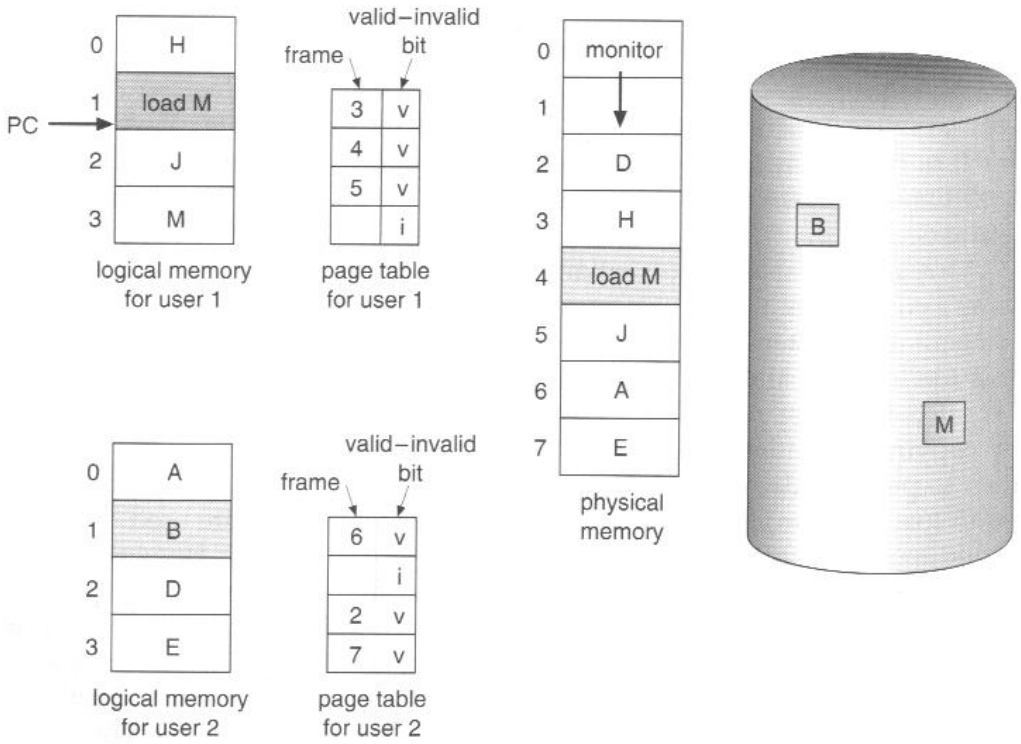
- Caso não exista um frame livre na memória:
  - terminar o processo
  - escolher um processo e colocá-lo na memória secundária
  - escolher uma página e substituí-la

Substituição de páginas

- Rotina de falha de página
  - Procurar a localização da página no disco
  - Procurar um frame livre
    - caso exista, utilize-o
    - caso contrário, utilize um algoritmo de troca de página para selecionar uma página vítima
    - colocar a página vítima no disco, e atualizar as tabelas (páginas, frames)
  - Carregar a página no frame livre e atualizar as tabelas (páginas, frames)
  - Reinicializar os processos

Substituição de páginas

- É necessário 2 transferências de páginas
- Utilização de bits modificação
  - suporte do HW
  - reduzir as operações de E/S



Algoritmos de substituição de páginas

- "Como selecionar a página vítima???"
- Avaliação de um algoritmo é realizado através de uma seqüência de acessos -> *"reference string"* ou *"reference trace"*
- Gerado randomicamente ou através da observação de um sistema real (*"trace"*)
  - kVMTrace (http://www.cs.amherst.edu/~sfkaplan/research/kVMTrace/index.html)
  - VMTrace (http://linux-mm.org/VmTrace)

Reference string

- Ex:
  - `0100, 0432, 0101, 0612, 0102,`
    `0103, 0104, 0101, 0611, 0102,`
    `0103, 0104, 0101, 0610, 0102,`
    `0103, 0104, 0101, 0609, 0102, 0105`
  - representação utilizando número de páginas (100 bytes)
  - `1, 4, 1, 6, 1, 6, 1, 6, 1`

Algoritmos de substituição de páginas

- Número de frames livres

- O número de falhas de páginas diminui a medida que o número de *frames* aumenta.
  - String
    - `7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0 ,1, 7, 0, 1`

FIFO

- *"First in- First out"*:
- cada página tem um tempo associado relativo ao instante em que foi carregado na memória (fila)
- `7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0 ,1, 7, 0, 1`

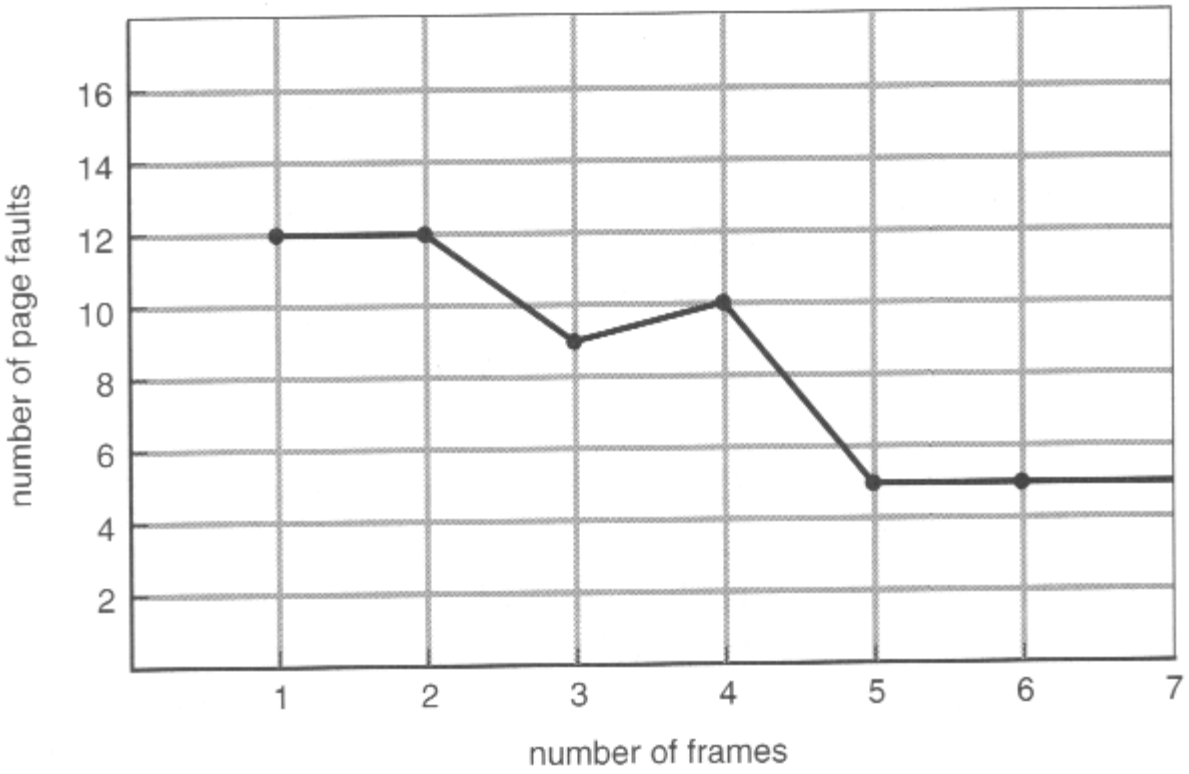
7	7	7	2	2	2	4	4	4	0	0	0	7	7	7
	0	0	0	3	3	3	2	2	2	1	1	1	0	0
		1	1	1	0	0	0	3	3	3	2	2	2	1

FIFO

- Fácil implementação
- Considerações:
  - a página substituída pode ser de um módulo carregado que não está sendo mais utilizado
  - variável inicializada no início da execução sendo acessada constantemente
- a execução é realizada de forma correta mesmo que a página escolhida seja uma que está sendo utilizada

FIFO

- 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- anomalia de Belady: “taxa de falhas pode aumentar quando o número de frames alocados aumenta” (3 e 4 frames livres)



Algoritmo ótimo

- Menor taxa de falhas entre todos os outros algoritmos
- “Substitua a página que não será utilizada pelo maior período de tempo”
- Este algoritmo garante que a menor taxa de falhas de páginas para um conjunto número de frames.
- `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	2	7
	0	0	0	0	4	0	0	0
		1	1	3	3	3	1	1

Algoritmo ótimo

- 9 falhas de página
- Necessita do conhecimento futuro sobre os acessos a memória
- Utilizado para comparar com outros algoritmos existentes.

Algoritmo LRU

- FIFO x OPT
- menos recentemente utilizado
- “Utilizar o passado recente como uma aproximação para o futuro próximo”
- A página substituída será a que está a mais tempo sem ser referenciada
- 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

Algoritmo LRU

- 12 falhas x 15 falhas FIFO
- Implementação
  - contadores: cada tabela tem um campo contendo o tempo lógico da última referência
    - substituir a página com o menor tempo associado a página
    - busca na tabela de páginas
    - estouro do relógio
    - toda referência a memória deve atualizar o campo do relógio

Algoritmo LRU

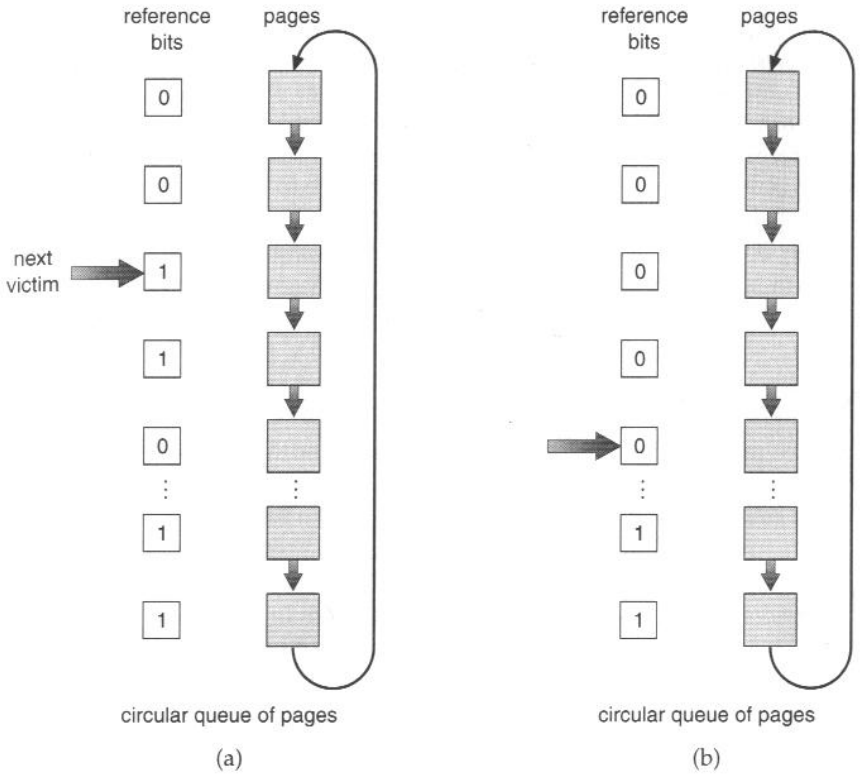
- pilha:
  - número de páginas
  - quando uma página é referenciada ela é removida da pilha e colocada no topo
  - o topo da pilha contém a página mais recentemente utilizada e a base da pilha a página LRU
  - Implementação através de listas duplamente encadeadas
  - não existe busca na substituição de páginas
- OPT, LRU não sofre a anomalia de Belady
- suporte do hardware

Algoritmos aproximados LRU

- Suporte através de bit de referência
  - o bit é “ligado” caso a página seja referenciada
  - um bit para cada entrada na tabela de páginas
- Utilização de bits adicionais de referência
  - armazenar para cada entrada da tabela de páginas um campo de 8-bit (byte)
  - a cada intervalo o sistema operacional é invocado
    - o SO desloca o bit para a direita
    - transfere o bit de referência na parte alta do byte
    - histórico da utilização dos últimos 8 períodos

Algoritmos aproximados de LRU

- Algoritmo da segunda chance (relógio)
- Similar a política FIFO
  - se o bit de referência é 1 a página não é escolhida, e o bit recebe o valor 0



Algoritmos aproximados de LRU

- Algoritmo da segunda chance melhorado
- considerar o bit de referência e bit de modificação
  - (0, 0): não foi recentemente utilizado nem modificado
  - (0, 1): não foi recentemente utilizado porém modificado
  - (1, 0): recentemente utilizado porém não modificado
  - (1, 1): recentemente utilizado e modificado
- Utilizar o mesmo algoritmo do relógio, porém dar prioridade para páginas não acessadas e não modificadas

Algoritmos de contagem

- Associar para cada página um contador, que armazena o número de acessos a página
  - Algoritmo menos freqüentemente utilizado: menor contador. Uma página pode ser utilizada um grande número de vezes na inicialização e depois nunca mais ser utilizada
    - Periodicamente, deslocar o contador para a direita em 1 bit para decair o número de acessos
  - Algoritmo mais freqüentemente utilizado: “uma página com um valor de contador pequeno, acabou de ser carregada não foi utilizada suficientemente”
- Custo alto de implementação e não se aproximam do algoritmo ótimo

Utilização de buffers de página

- Manter sempre um conjunto de frames livres, assim quando uma falha de pagina ocorre ela é carregada no frame livre, enquanto a página vítima é escrita no disco
  - o processo inicializa o mais rápido possível
- Manter uma lista de páginas modificadas
  - quando o disco estiver ocioso, as páginas são escritas no disco
  - aumenta a probabilidade de um frame com o bit modificado em 0

9.4 Alocação de frames

- A partir de um número **m** de frames livres, qual número de frames que serão alocados para os processos em execução
- Número mínimo de frames que devem ser alocados é determinado pelo conjunto de instruções
  - instruções que acessam somente um endereço por instrução: 2 páginas (1 instrução + 1 dado)
  - referências indiretas:load M -> 3 frames

Algoritmos de alocação

- **m** frames com **n** processos em execução
- **m/n**: alocação igualitária
- Ex:Memória com 62 frames de tamanho 1K
  - Estudante : 10 K
  - Banco de dados 127 K
  - 31 frames para cada processo !!!
- Alocação proporcional
  - seja s_i o tamanho da memória do processo p_i
  - $S = \Sigma s_i$

Algoritmos de alocação

- Número de frames disponíveis **m**. Alocação de a_i frames para o processo pi, onde ai é determinado por:
  - $a_i = (s_i / S) \times m$
- a_i deve ser ajustado para o número mínimo de frames necessário pela arquitetura
- Ex: Frames= 62
  - Processo 1= 10 páginas (~5 frames)
  - Processo 2=127 páginas (~57 frames)
- Novo processo entra em execução é recalculado a divisão de frames -> alguns processos perdem frames

Algoritmos de alocação

- Processos de alta e baixa prioridade são tratados da mesma forma
- Utilização de prioridade: dar mais tempo de CPU para processos com prioridade alta
- Considerar a prioridade também para calcular o número de frames alocados para cada processo

Alocação Global x Local

- Ambiente com múltiplos processos em execução: competindo por frames
- Global
  - seleção do frame entre todos os frames disponíveis na memória
  - processos de alta prioridade podem obter frames de processos de baixa prioridade
  - problema: processo não controla a sua taxa de falhas de página
  - sua execução vária de acordo com as condições do sistema

Alocação Global x Local

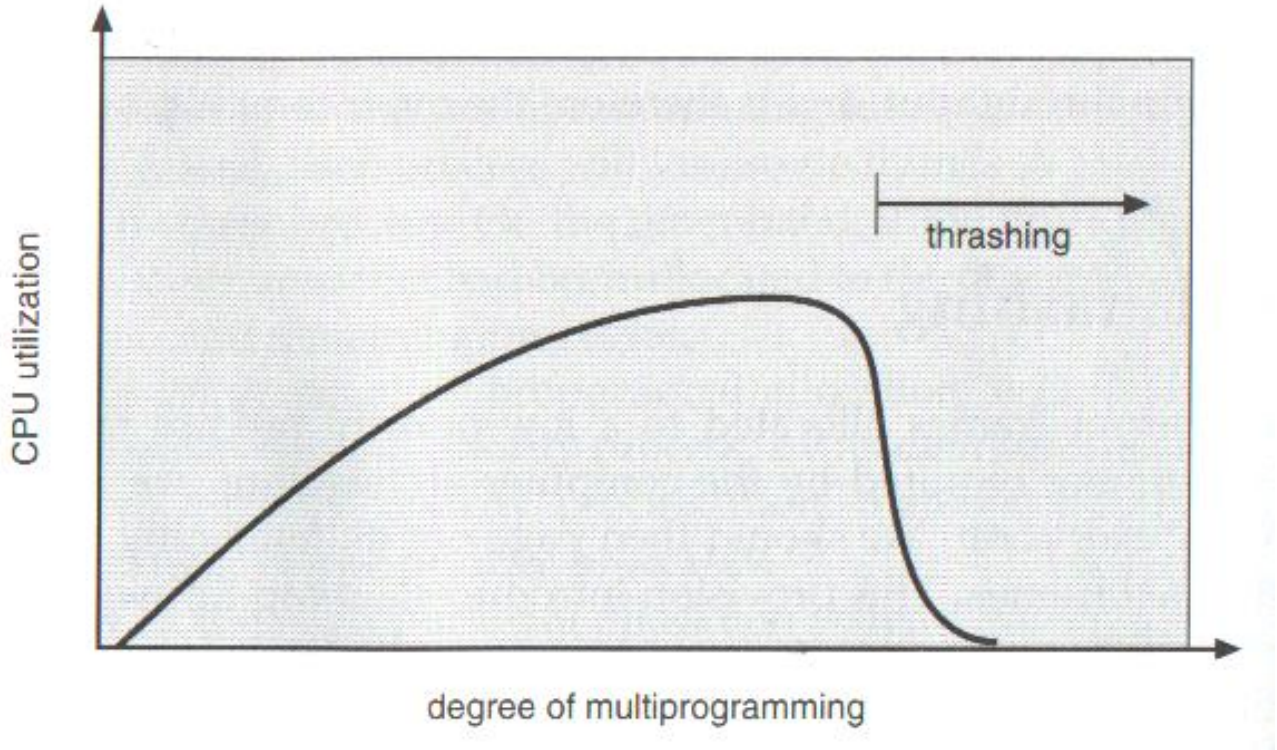
- Local
  - o frame a ser substituído é escolhido dentre o conjunto de frames alocados para o processo
  - o processo é afetado somente pelo comportamento da taxa de falhas do processo
  - os frames alocados ficam indisponíveis para outros processos

“Thrashing”

- Thrashing: alta taxa da atividade de paginação
- o tempo gasto para paginação é maior que a execução
- SO monitora o estado do sistema
  - caso a utilização seja baixa, aumenta o grau de multiprogramação do sistema
  - mais processos executando, maior necessidade de memória -> aumento do número de falhas de páginas
  - acesso ao disco para busca da página
  - a taxa de utilização da CPU diminui

Thrasing

- Utilização de algoritmos de substituição local
- Modelo de localidade



Modelo de conjunto de trabalho

- *"Working set model"*
- Δ -= janela do conjunto de trabalho

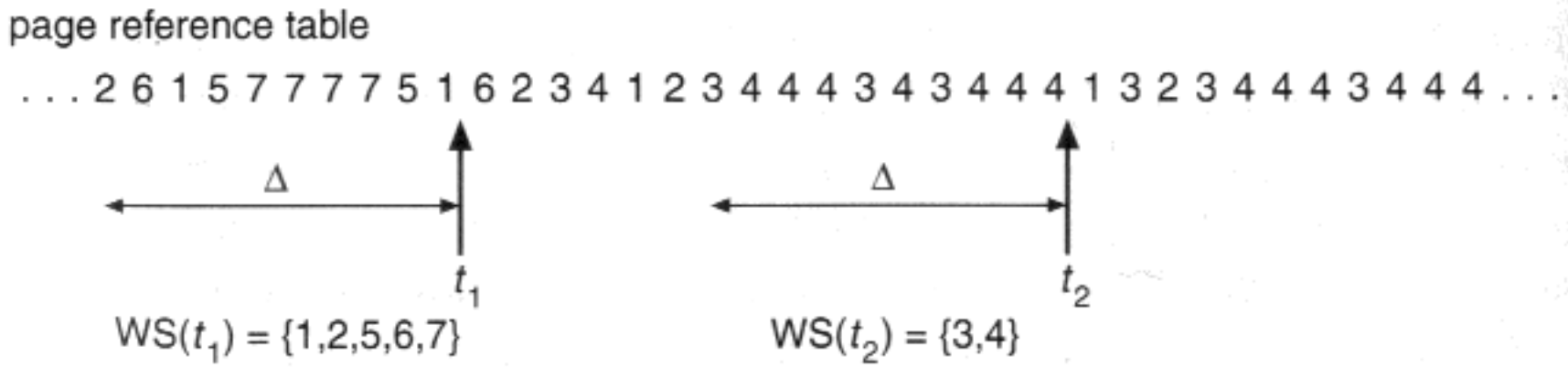


Figure 9.16 Working-set model.

Modelo de conjunto de trabalho

- Δ referências a páginas
  - valor pequeno:não captura toda a localidade exibida pelo programa
  - valor grande: vai armazenar páginas que não são utilizadas
  - infinito: todas as páginas referenciadas durante a execução
- D= ΣWSSi (demanda de frames)
- D > m (m número de frames disponíveis)
- SO monitora para verificar se a demanda é menor que o número de frames disponíveis

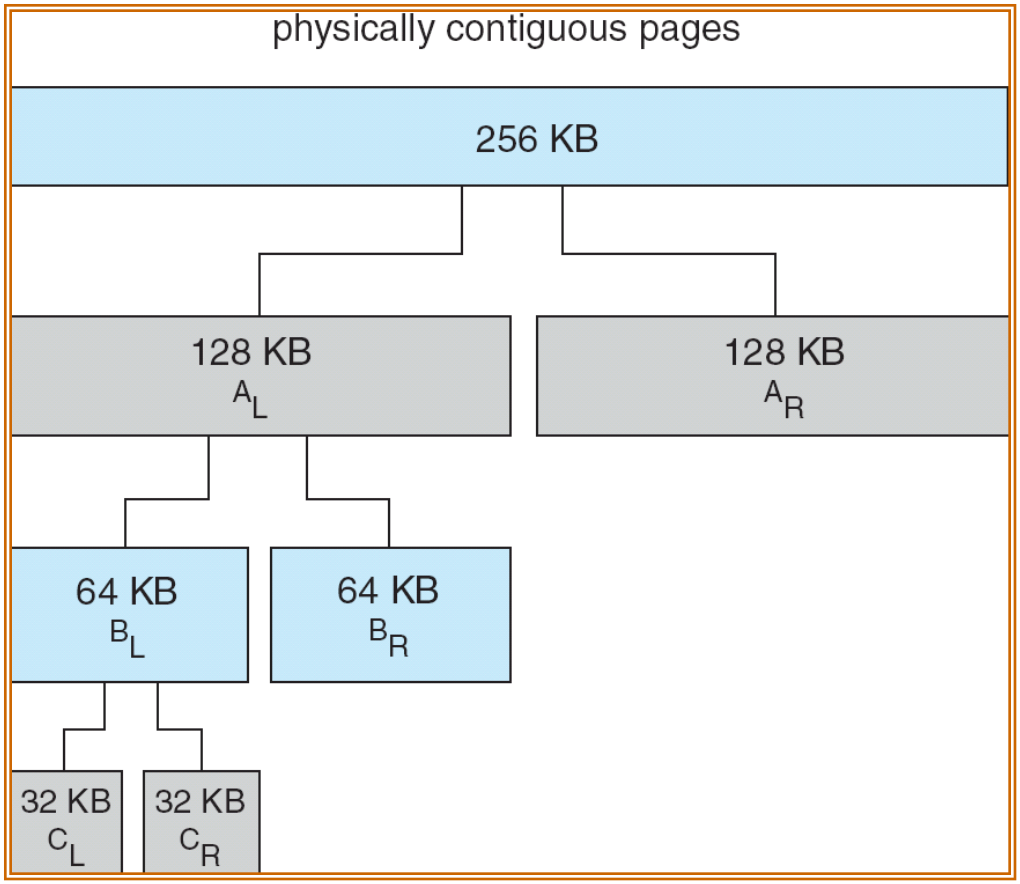
Modelo de conjunto de trabalho

- Como obter a janela de conjunto de trabalho??
  - Utilizar os bits de referência
  - Ex: Δ = 10000 referências
  - *timer* de interrupção a cada 5000 referências
  - copia e limpa os bits de referência
- **Freqüência de falhas de página**
  - baixa: o número de frames do processo está acima do necessário
  - alta: o número de frames alocados para o processo é insuficiente

Buddy System

- Alocação da memória a partir de segmentos de tamanho fixo
- Memória alocada de tamanho 2^n
  - Uma requisição é arredondada para um tamanho 2^n
  - Quando um tamanho menor deve ser alocado, um segmento maior é dividido na metade.
    - Continua até que o tamanho adequado seja obtido

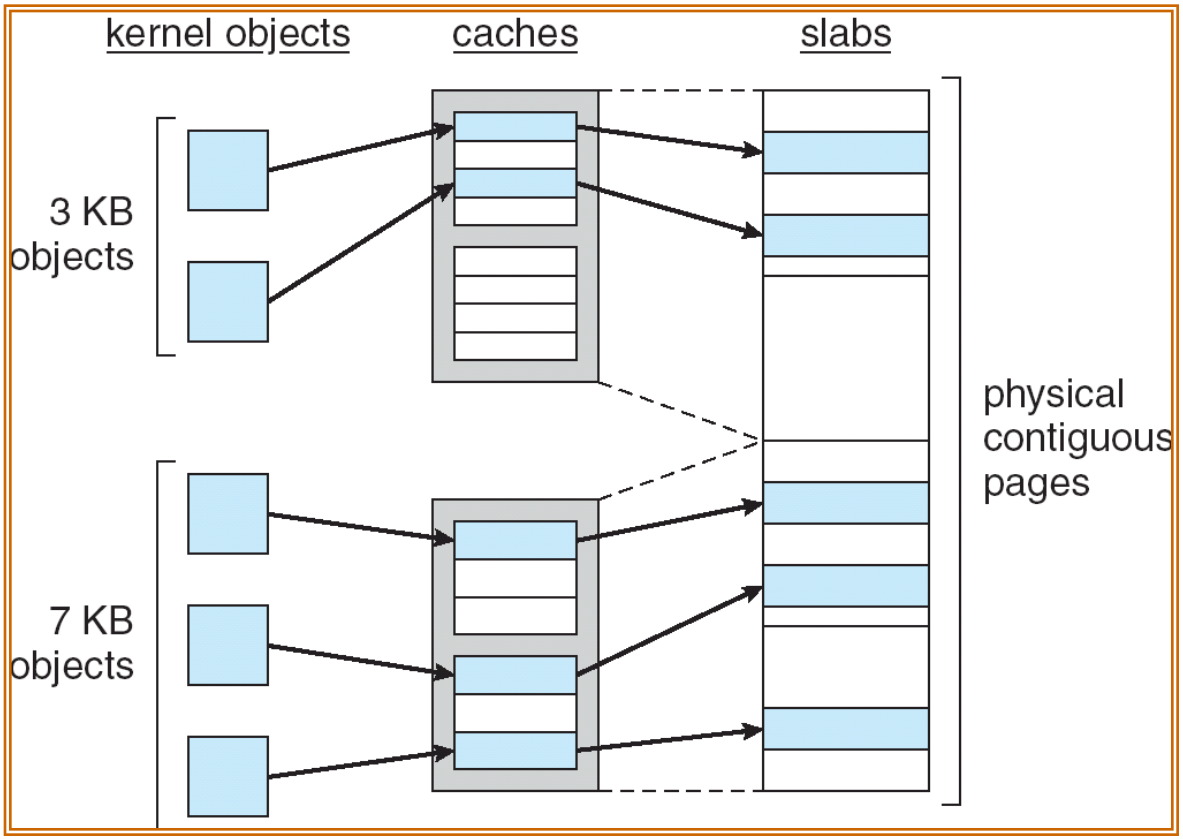
Buddy System Allocator



Slab Allocator

- Estratégia alternativa para alocação de memória do kernel
- **Slab**: uma ou mais páginas físicamente contíguas
- **Cache**: um ou mais Slabs
- Cache única para cada estrutura de dados do kernel
  - Cada cache é preenchida com dados do kernel – instâncias das estruturas de dados
- Quando uma cache é criada, os objetos preenchidos são marcados como livres
- Quando as estruturas são utilizadas, os objetos são marcados como usados
- Se uma slab está cheio, um novo slab é utilizado
  - Se nenhum slab está disponível, criar um novo slab
- Benéficios: sem fragmentação, requisição de memória é satisfeita rapidamente

Slab Allocation



9.6 Pré-paginação

- Paginação por demanda: alta taxa de falhas de paginas no início da execução do processo, ou o retorno da execução de um processo que foi transferido para o disco (*swap-out*)
- Pré-paginação das páginas necessárias
- Utilização com o modelo de conjunto de trabalho
- s= número de páginas pré-paginadas
- α= a fração de página efetivamente utilizadas
- s * α= o número de falhas de páginas que foram evitadas
- s*(1- α)= número de páginas carregadas sem necessidade

Tamanho de página

- Fortemente influenciada pela arquitetura
  - ex: Intel 386- 4 K
  - Alpha 20164: 4 - 16K
- páginas pequenas: diminuem a fragmentação interna
- páginas grandes: diminuem o tempo de E/S necessário. Ex: buscar 1 página de 512 bytes é mais rápido que buscar 2 páginas de 256 bytes (seek + latência + transferência)
- tamanho da tabela de páginas

Tabela de páginas invertidas

- Armazenamento da informação do processo que está utilizando o frame < processo, número da página>
- Não contém todas as informações do endereço lógico do processo
- Tabela de páginas necessário para localizar a página no disco
- Por que utilizar tabelas invertidas ??
  - A taxa de falhas pequena, logo a tabela de páginas podem ser armazenadas no disco

Programação

- Transparente para o usuário
- Cuidados durante a programação

```
var a: array[1..128, 1..128] of integer;
for j:= 1 to 128 do
    for i:= 1 to 128 do
      A[i][j]:= 0
```

- A[1][1], A[1][2], A[1][3]...A[128][128]
- páginas de 128 palavras

```
var a: array[1..128, 1..128] of integer;
for i:= 1 to 128 do
    for j:= 1 to 128 do
      A[i][j]:= 0
```

“I/O interlock”

- Páginas utilizadas como *buffer* de E/S
  - processo executa uma instrução de E/S, passando para o estado bloqueado
  - outro processo é escalonado e causa uma falha de página
  - a página onde estão sendo copiados os dados de E/S é escolhida para ser substituída...
- Duas soluções
  - bloquear páginas na memória
  - utilizar buffers do sistema operacional

Tempo real

- Memória virtual: acrescenta a incerteza no processamento (inaceitável em processamento de sistemas de tempo real)
- Solaris 2: páginas podem ser bloqueadas na memória por processos privilegiados