

Sistemas Operacionais

Gerenciamento de Memória *Memória Virtual*

Sistemas Operacionais

Eduardo Nicola F Zagari

SO - Gerenciamento de Memória

- ◆ Memória Virtual
 - ❖ Overlay
 - ❖ Paginação por Demanda
- ◆ Substituição de Páginas
 - ❖ Algoritmos de Substituição de Páginas
 - ❖ Procedimentos de Armazenamento de Páginas
- ◆ Política de Alocação Local *versus* Alocação Global
- ◆ *Thrashing*
- ◆ *Working Set* (Conjunto de Trabalho)
- ◆ Frequência de Interrupção de Página Ausente
- ◆ Tamanho da Página
- ◆ Bloqueio de Páginas na Memória
- ◆ Estrutura de Programas

Memória Virtual

- ◆ Todas as estratégias de gerência até agora mantêm processos inteiros na memória
- ◆ Em muitos casos, não é necessário o programa inteiro estar em memória:
 - ❖ código de tratamento de erro
 - ❖ vetores, listas e tabelas geralmente alocam mais memória do que usam
 - ❖ certas opções e características de programas raramente são usadas

Memória Virtual

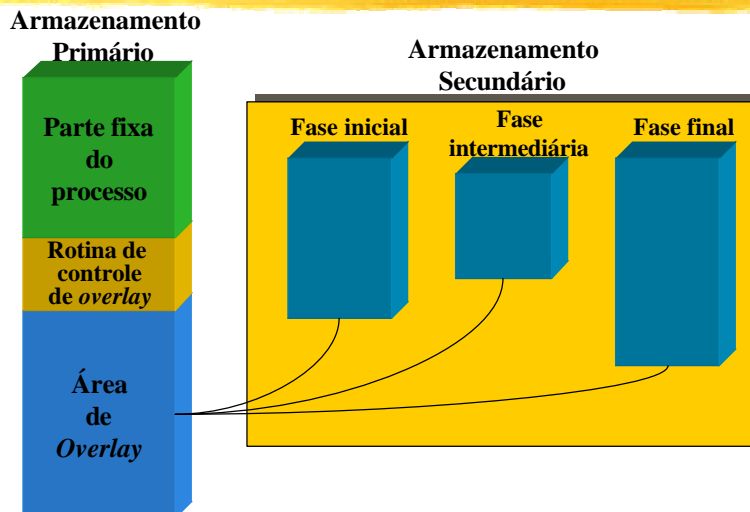
- ◆ Vantagens de executar programas parcialmente:
 - ❖ programa não é limitado pelo tamanho da memória
 - ❖ um maior número de processos podem ser carregados ao mesmo tempo → aumentando a utilização da CPU
 - ❖ menor necessidade E/S para carregar ou fazer *swapp* de processos

Overlay

◆ Sobreposição (*Overlay*) (usuário)

- ❖ O que fazer quando se tem um programa que ocupa mais memória do que a disponível?
- ❖ Verifica-se se o programa tem seções que não necessitem permanecer na memória durante toda a execução
- ❖ Essas seções, uma vez executadas, podem ser substituídas por outras
- ❖ A substituição ocorre pela transferência da seção do disco para a memória na área de *overlay*, cobrindo a seção anterior
- ❖ O programa é inicialmente carregado mais rapidamente, entretanto, a execução ficará um pouco mais lenta, devido ao tempo extra gasto com a sobreposição

(cont.) *Overlay*



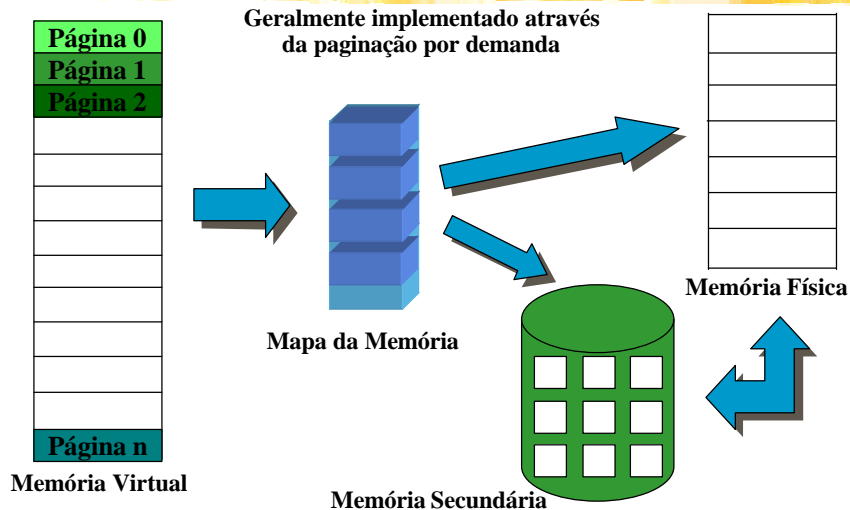
(cont.) *Overlay*

- ◆ O mecanismo de *overlay* é pouco transparente, pois a definição das áreas de *overlay* é função do programador (não é do SO), através de comandos específicos da linguagem utilizada. Dependendo da definição, pode ocasionar sérias implicações no desempenho das aplicações, devido à transferência excessiva dos módulos entre o disco e a memória

Memória Virtual

- ◆ Técnica de Memória Virtual
 - ❖ executar programas que não precisam estar completamente carregados na memória
 - ❖ separação da memória lógica de usuário e memória física

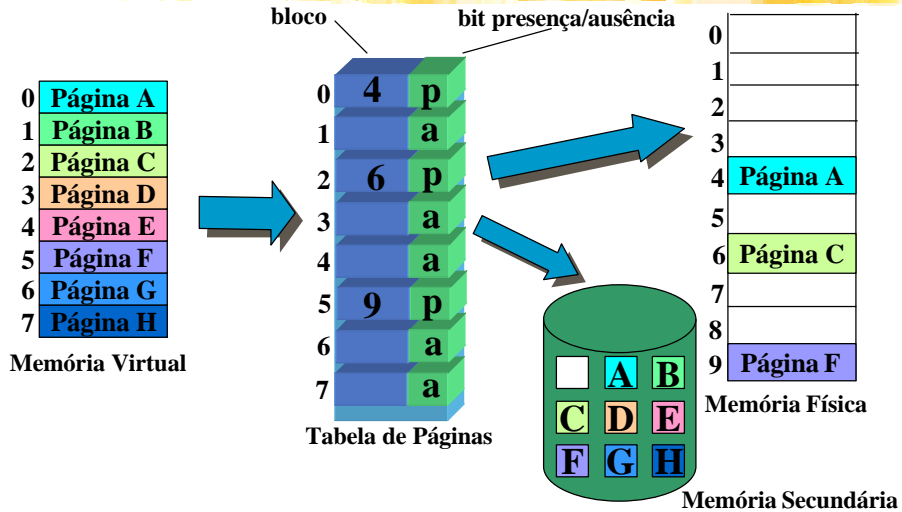
Memória Virtual



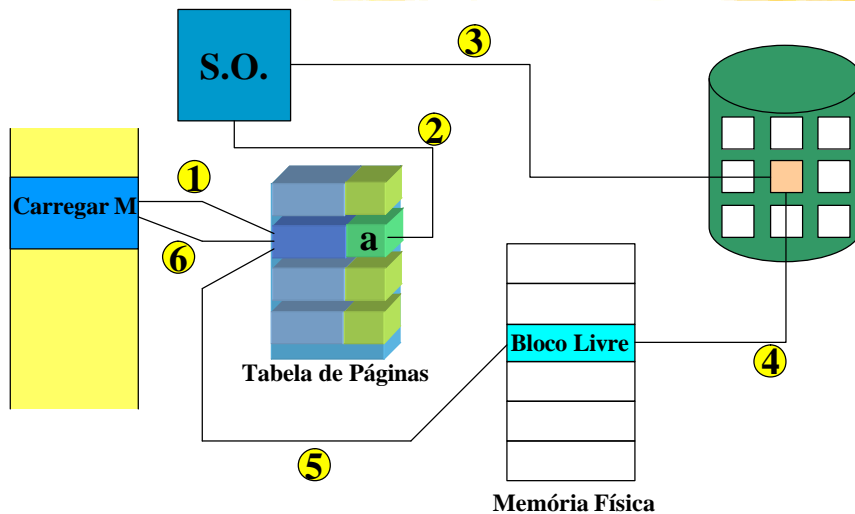
Paginação por Demanda

- ◆ Similar ao sistema de *swapping*, mas aplicado somente à página requisitada e não a todo o processo
- ◆ *Hardware* : bit de presença/ausência
- ◆ *Page-fault (trap)* : interrupção por falta de página

Paginação sob Demanda: *hardware*



Paginação sob Demanda: *page-fault*



Paginação sob Demanda: *page-fault*

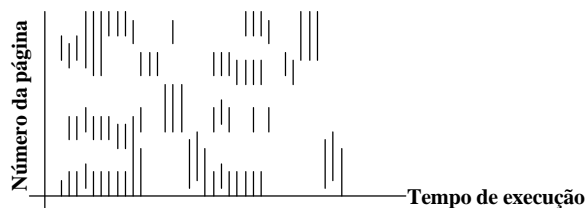
- ① Referência: verifica uma tabela do processo se referência é válida
- ② *Trap*: se a referência for inválida ERRO; se for válida, mas a página não está na memória, gera interrupção
- ③ Página na memória secundária: localiza um bloco livre
- ④ Carrega página: realiza uma operação no disco (E/S) para ler a pág. desejada e carregá-la p/ o bloco livre
- ⑤ Modifica tabela de páginas: terminada a op. de E/S, atualiza a tabela p/ indicar que a pág. está na memória
- ⑥ Reinicia execução da interrupção: como o estado do processo foi salvo, reinicia a interrupção que causou a falta de página (endereço de pág inexistente na mem.)

Paginação por Demanda

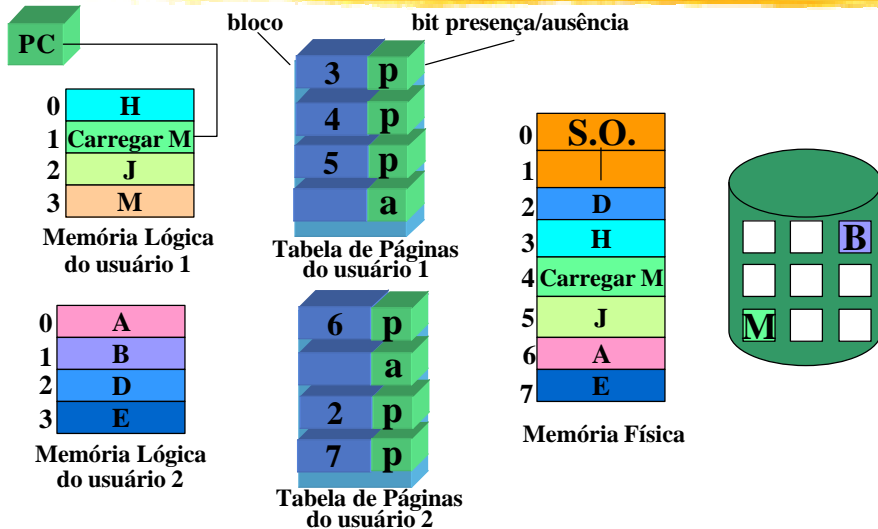
- ◆ Paginação por demanda pura:
 - ❖ não levar nenhuma página para memória até que ela seja requisitada

Queda de desempenho?

Padrão de localidade de referência



Substituição de Páginas: sobrealocação



Substituição de Páginas

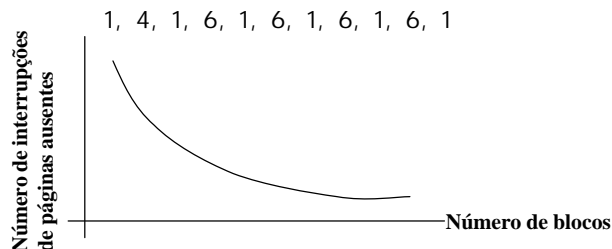
- ◆ Substituição de página: quando ocorre uma falta de página e não existe nenhum bloco disponível, um bloco que não está sendo usado no momento é selecionado e liberado. A rotina de tratamento de uma *page-fault*:
 - ❖ Localizar a página desejada no disco
 - ❖ Localizar um bloco livre:
 - 📄 se existir um bloco livre, usá-lo
 - 📄 caso contrário, usar um algoritmo de substituição de página para selecionar a que será substituída
 - 📄 Salvar a página a ser substituída no disco e atualizar as tabelas de blocos e páginas de acordo
 - ❖ Ler a página desejada no "novo" bloco e atualizar as tabelas
 - ❖ Reiniciar o processo do usuário

Substituição de Páginas

- ◆ Se não existe nenhum bloco livre → note que são necessárias 2 transferências de página
 - ❖ para reduzir esta sobrecarga, usa-se um bit de modificação
 - 📄 ele é ativado pelo *hardware* sempre que qualquer palavra ou byte desta página é alterado, de forma a indicar que a página foi modificada
 - 📄 se a página é selecionada para ser substituída:
 - bit de modificação ativado → deve ser gravada no disco
 - bit de modificação não ativado → não é necessário ser regravada no disco (o bloco é sobreposto)

Algoritmos de Substituição de Páginas

- ◆ Menor taxa de falta de página
- ◆ Ex: seqüência de referências a endereços de memória
0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103,
0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105
- ❖ Se cada página tiver 100 bytes, os acessos serão às pags.:



Algoritmos de Substituição de Páginas

- ◆ Algoritmo FIFO (*First-In, First-Out*) ou PCPS (Primeiro a Chegar, Primeiro a Sair)
- ◆ Algoritmo Ótimo
- ◆ Algoritmo NRU (*Not Recently Used*)
- ◆ Algoritmo LRU (*Least Recently Used*)
 - ❖ NFU (*Not Frequently Used*)
 - ❖ Aging
- ◆ Algoritmo da Segunda Chance
- ◆ Algoritmo do Relógio
- ◆ Algoritmo MFU

FIFO

- ◆ A página que foi primeiro utilizada (*first-in*) será a primeira a ser retirada (*first-out*)
 - ❖ Fila (lista encadeada) de páginas { Início: Página mais velha
Fim: Página mais nova
 - ❖ Falta de página { Retira página do início
Insere página nova no fim
- ◆ Simples
- ◆ A princípio: razoável que a página há mais tempo no *working set* deva ser retirada 1ª → mas isto nem sempre é verdade...

Algoritmo Ótimo

- ◆ Trocar a página que não será usada no período mais longo

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

9 faltas de páginas

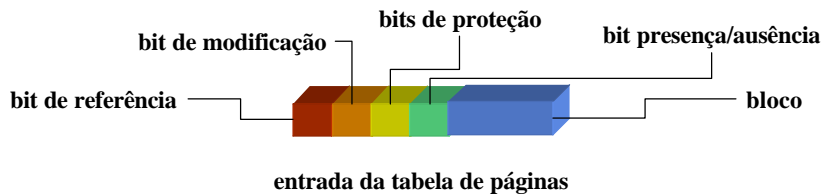
Algoritmo Ótimo

- ◆ Problema: saber quando cada página será referenciada
 - ➔ Impossível de se implementar
- ◆ Utilidade: Comparação com relação aos demais algoritmos

NRU (*Not Recently Used*)

◆ **Hardware:** 2 bits de estado associados a cada página

- ❖ Bit R (referência): ativado quando a página é referenciada
- ❖ Bit M (modificação): ativado quando ela é modificada



- ❖ Bits ativados por *hardware* e desativados por *software*

NRU (*Not Recently Used*)

◆ Se o *hardware* não dispõe dos bits R e M:

→ Simulados por *software*

- 📄 Processo iniciado → Ausente na memória (tabela de páginas zerada)
- 📄 Referência a uma página → falta de página
- 📄 SO ativa bit R (tabela interna), atualiza tabela de páginas, seta proteção para "Somente Leitura" e retoma instrução
- 📄 Se instrução de escrita → nova falta de página
- 📄 SO ativa bit M e muda modo da página para "Leitura/Escrita"

NRU (*Not Recently Used*)

- ◆ Escolhe a página que não foi recentemente utilizada

- ◆ Algoritmo:

- ☆ Processo iniciado → bits de página (R e M) iguais a zero
- 🕒 Periodicamente, bit R é zerado
- 🕒 Falta de página → SO inspeciona páginas e as classifica:
 - 🕒 Classe 0: não referenciadas, não modificadas
 - 🕒 Classe 1: não referenciadas, modificadas
 - 🕒 Classe 2: referenciadas, não modificadas
 - 🕒 Classe 3: referenciadas, modificadas
- 🕒 NRU remove página aleatória da classe não vazia de numeração mais baixa

Atenção



LRU (*Least Recently Used*)

- ◆ Seleciona a página usada há mais tempo, isto é, o objetivo é retirar aquela que está há mais tempo sem ser referenciada
 - ◆ Realizável → alto custo
 - ◆ Manter lista { Início: Página usada mais recentemente
Fim: Página usada menos recentemente
- dificuldade: atualizar a cada referência à memória

LRU (*Least Recently Used*)

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

12 faltas de páginas

LRU (*Least Recently Used*)

◆ Implementação por *hardware*: 2 maneiras

- ❖ Contador: um contador é adicionado à CPU e incrementado automaticamente a cada referência à memória. Toda vez que é feita uma referência à página, o conteúdo do contador é copiado para o campo instante-de-uso na entrada da tabela de páginas referente à página acessada.

→ seleciona-se a página com o menor contador

- ❖ Matriz $n \times n$ (onde n é o número de blocos): toda vez que a página k é referenciada

{ colocam-se todos os bits da linha k em 1
 { depois, zeram-se todos os bits da coluna k

→ em cada instante, a página LRU corresponde à linha com o menor valor binário armazenado

LRU (*Least Recently Used*)

- ◆ Considere a seguinte ordem de referências às páginas

0 1 2 3 2 1 0 3 2 3

	0	1	2	3		0	1	2	3		0	1	2	3		0	1	2	3		0	1	2	3
0	0	1	1	1		0	0	1	1		0	0	0	1		0	0	0	0		0	0	0	0
1	0	0	0	0		1	0	1	1		1	0	0	1		1	0	0	0		1	0	0	0
2	0	0	0	0		0	0	0	0		1	1	0	1		1	1	0	0		1	1	0	1
3	0	0	0	0		0	0	0	0		0	0	0	0		1	1	1	0		1	1	0	0

0	0	0	0	0		0	1	1	1		0	1	1	0		0	1	0	0		0	1	0	0
1	1	0	1	1		0	0	1	1		0	0	1	0		0	0	0	0		0	0	0	0
2	1	0	0	1		0	0	0	1		0	0	0	0		1	1	0	1		1	1	0	0
3	1	0	0	0		0	0	0	0		1	1	1	0		1	1	0	0		1	1	1	0

LRU (*Least Recently Used*)

- ◆ Primeira implementação por *software*:
- ❖ Algoritmo NFU (Não Frequentemente Usada)
 - 📖 requer um contador em *software* associado a cada página
 - 📖 a cada interrupção de tempo, SO faz

$$\text{Cont} \leftarrow \text{Cont} + \text{bit R}$$
 - 📖 Falta de página → SO escolhe página com menor Cont
 - ❖ Problema: nunca esquece referências anteriores

LRU (*Least Recently Used*)

◆ Segunda implementação por *software* :

❖ Algoritmo *Aging* (Envelhecimento): modificação do NFU

- 📄 Contadores são sempre deslocados um bit antes de serem somados com o bit R
- 📄 Bit R é adicionado ao bit mais à esquerda do contador, ao invés de ser somado ao mais à direita
- 📄 Falta de página → SO escolhe página com menor Cont

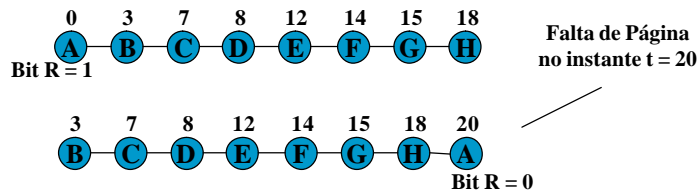
LRU (*Least Recently Used*) - *Aging*

Bits R	Bits R	Bits R	Bits R	Bits R
101011	110010	110101	100010	011000
10000000	11000000	11100000	11110000	01111000
00000000	10000000	11000000	01100000	10110000
10000000	01000000	00100000	00010000	10001000
00000000	00000000	10000000	01000000	00100000
10000000	11000000	01100000	10110000	01011000
10000000	01000000	10100000	01010000	00101000

Algoritmo da Segunda Chance

- ◆ É um algoritmo FI FO modificado
- ◆ Possui um bit de referência (bit R) que indica se página foi referenciada
 - ❖ Se página mais antiga possui $R = 0$ (não foi referenciada)
 - ela é escolhida
 - ❖ Caso contrário,
 - R é feito zero ($R \leftarrow 0$) e ela vai para o final da fila
 - A busca continua

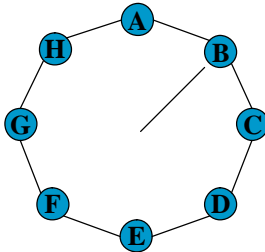
Algoritmo da Segunda Chance



- ◆ Retira página velha não referenciada no último intervalo de tempo
- ◆ Problema: Se todas as páginas foram referenciadas, então algoritmo degenera para um FI FO puro

Algoritmo do Relógio

- ◆ Semelhante ao da Segunda Chance, mas mais eficiente
 - ❖ Falta de Página → inspeciona página para a qual o ponteiro está apontando:
 - 📄 Se bit $R = 0$ → retire a página
 - 📄 Se bit $R = 1$ → faça $R \leftarrow 0$ e avance o ponteiro



MFU (*Most Frequently Used*)

- ◆ Escolhe a página mais freqüentemente usada
- ◆ Argumento: uma página com contador de valor baixo, foi, provavelmente, trazida para a memória recentemente e ainda será usada

Procedimentos de Armazenamento de Páginas

- ◆ Reserva de blocos
 - ❖ não é necessário esperar que a página selecionada seja transferida para o disco
- ◆ Lista de páginas modificadas
 - ❖ automação através do dispositivo de transferência de páginas
 - ❖ minimiza probabilidade da página selecionada não estar atualizada no disco
- ◆ Manter registros de qual página está no conjunto de blocos reservados
 - ❖ pode-se evitar E/S, aproveitando-se a antiga página contida na reserva de blocos

Política de Alocação Local *versus* Alocação Global

- ◆ Substituição Global:
 - ❖ permite alocar para um determinado processo qualquer bloco da memória, mesmo que já esteja alocado a outro processo
 - ❖ Desempenho do processo depende do comportamento dos demais
- ◆ Substituição Local:
 - ❖ o bloco alocado a um processo deve pertencer ao conjunto de blocos reservado para este processo
 - ❖ Poucos blocos: pode provocar queda de desempenho, mesmo havendo blocos livres ou usados com menos frequência
 - ❖ Muitos blocos: pode provocar desperdício
- ◆ Na prática os algoritmos globais funcionam melhor

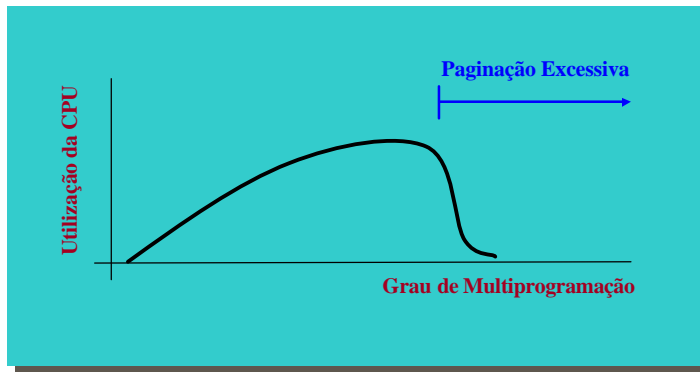
Política de Alocação Local *versus* Alocação Global

		Idade			
0	A0	10	A0	A0	A0
1	A1	7	A1	A1	A1
2	A2	5	A2	A2	A2
3	A3	4	A3	A3	A3
4	A4	3	A6	A4	A4
5	B0	9	B0	B0	B0
6	B1	4	B1	B1	B1
7	B2	6	B2	B2	B2
8	B3	2	B3	B3	A6
9	B4	5	B4	B4	B4
10	B5	12	B5	B5	B5
11	C1	5	C1	C1	C1
12	C2	6	C2	C2	C2
	Memória Física		Substituição Local		Substituição Global

Thrashing

- ◆ Considere que o sistema monitore a utilização da CPU
 - ❖ Se for baixa → aumenta o grau de multiprogramação
- ◆ Considere também que é usado um algoritmo de trocas de páginas global
- ◆ Agora suponha que um processo entre em uma nova fase, que precise de mais páginas
 - ❖ Ele "rouba" blocos dos outros processos, aumentando a taxa de falta de páginas, conseqüentemente, diminuindo a utilização da CPU → realimentando o ciclo
- ◆ Dizemos que há *thrashing* (paginação excessiva) q^{do} ele está consumindo mais tempo em substituição de páginas do que na execução do seu código

Thrashing



Conjunto de Trabalho (*Working Set*)

- ◆ Para prevenir *thrashing* deve-se fornecer aos processos o número de blocos que ele precisa
- ◆ Inicialização de um processo
 - ➔ Paginação sob Demanda
 - ➔ Funciona devido à localidade de referências

Mas o que fazer quando um processo sofre *swap in*?

- ◆ Conjunto de Trabalho
 - ❖ É o conjunto de páginas referenciadas por um processo durante um determinado intervalo de tempo
 - ❖ É o conjunto de páginas constantemente referenciadas pelo processo e que deve permanecer na memória para que ele execute de forma eficiente

Conjunto de Trabalho (*Working Set*)

◆ Modelo do Conjunto de Trabalho:

- ❖ SO mantém controle sobre o conjunto de trabalho dos processos
- ❖ SO carrega *antes* as páginas do CT de um processo, ao colocá-lo em execução, reduzindo o nº de falta de páginas (pré-paginação)

Quais páginas devem fazer parte do CT de um processo?

- ◆ Uma forma de implem/ é através do algoritmo *aging*:
 - ❖ qualquer página contendo um bit 1 entre os n bits mais significativos do contador será considerada membro do CT do processo à qual ela pertence

Conjunto de Trabalho (*Working Set*)

- ◆ SO monitora a quantidade de páginas (D_i) nos conjuntos de trabalho de todos os processos
 - ❖ Se $\sum_i (CT_i)$ for maior que o nº de blocos livres \rightarrow *thrashing*
 - ❖ Neste caso, SO seleciona um processo para sofrer *swapping*
- ◆ *Working Set* grande $\left\{ \begin{array}{l} \text{Menor taxa de falta de página} \\ \text{Menor número de processos na memória} \end{array} \right.$

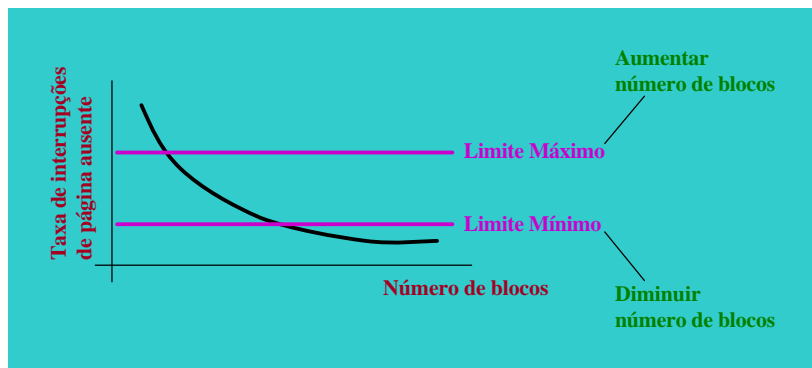
Alocação local: número de páginas iguais ou número de páginas proporcionais (nem considera *thrashing*)

**Alocação global: tamanho do CT pode variar muito mais rápido do que os bits de envelhecimento
Problema de *thrashing* persiste...**

Freqüência de Interrupção de Página Ausente

- ◆ Para evitar o *thrashing*: Alg. PFF (*Page Fault Frequency*)
 - ❖ Controla diretamente a taxa de interrupções por F.P.
 - 📄 F.P. muito alta → processo necessita de mais blocos
 - 📄 F.P. muito baixa → processo está com blocos em excesso
 - ❖ Estipula-se um limite máximo e um limite mínimo para a taxa de interrupções por F.P.
 - 📄 Processos acima do máx → recebem mais blocos
 - 📄 Processos abaixo do mín → removem-se blocos
 - ❖ Se a taxa de interrupções cresce e não há blocos disponíveis → seleciona-se um processo e o suspende → blocos liberados são redistribuídos ou colocados à disposição
- ◆ *Swapping* usado para redução de demanda potencial por memória

Freqüência de Interrupção de Página Ausente



Tamanho de Página

- ◆ Tamanho é sempre potência de 2 ($2^9 - 2^{14}$ bytes)
- ◆ SO pode escolher tamanho:
 - ❖ *hardware* projetado para pags. de 512 bytes
 - ❖ SO enxerga pag. 0 e 1, 2 e 3, ... como páginas de 1k, alocando sempre 2 blocos consecutivos na memória para elas
- ◆ Tamanho ótimo:
 - ❖ Páginas pequenas:
 - 📄 Menor desperdício: em média 1/2 página é perdida com fragmentação interna no fim de um segmento
 - 📄 Reduz a possibilidade de espaços inativos dentro de uma página

Tamanho de Página

- ❖ Páginas grandes:
 - 📄 Menor número de páginas → menor tabela de páginas
 - 📄 Transferir páginas pequenas toma quase tanto tempo quanto transferir páginas grandes para o disco (movimento da cabeça e latência rotacional)
 - 📄 Menor a taxa de falta de páginas
- ◆ Tamanho ótimo: 1448 bytes (valor teórico)
- ◆ A maioria dos computadores comerciais têm páginas de 512 a 8k bytes
- ◆ Historicamente, este valor tem aumentado...

Bloqueio de Páginas na Memória

◆ Considere o exemplo:

- ❖ Processo emite chamada de sistema de E/S
- ❖ Processo é suspenso
- ❖ Novo processo gera falta de página
- ❖ Se alocação global → existe chance da página escolhida para ser removida ser a página contendo o *buffer* de E/S
- ❖ Como o DMA (acesso direto) não tem como identificar em que página está escrevendo → inconsistência

◆ Primeira Solução:

- ❖ Operações de E/S usem um *buffer* dentro do *kernel* → a cópia pode resultar em uma sobrecarga inaceitável

Bloqueio de Páginas na Memória

◆ Segunda Solução:

- ❖ Bloquear páginas envolvidas em E/S, não permitindo que sejam removidas durante a operação de E/S → bit de bloqueio

◆ Outro uso do bit de bloqueio:

- ❖ Processo provoca interrupção por falta de página
- ❖ Página é carregada
- ❖ Processo volta para a fila de pronto
- ❖ Outro processo (de maior prioridade) ganha a CPU e provoca falta de página
- ❖ Escolhe página não referenciada e nem modificada: A candidata perfeita → a própria página do processo de baixa prioridade

Estrutura de Programas

- ◆ Paginação por demanda foi projetada para ser transparente aos programas de usuário, mas...
- ◆ Considere páginas de 128 bytes. Para iniciar com 'a' cada elemento de uma matriz de 128 x 128:

```
char A[128][128];
for (j=0; j<128; j++) {
    for (i=0; i<128; i++) {
        A[i][j] = 'a';
    }
}
```

```
char A[128][128];
for (i=0; i<128; i++) {
    for (j=0; j<128; j++) {
        A[i][j] = 'a';
    }
}
```

Estrutura de Programas

- ◆ A matriz é armazenada por linhas: A[0][0], A[0][1], A[0][2], ... , A[0][127], A[1][0], A[1][1], ... , A[127][127]
- ◆ Se a página tem tamanho 128 bytes → cada linha ocupa uma página

Taxa de faltas =

128 linhas X 128 colunas

16.384 faltas

Taxa de faltas =

128 colunas

128 faltas