

Sistemas Operacionais

Gestão de memória - Paginação em disco

Prof. Carlos Maziero

DInf UFPR, Curitiba PR

Agosto de 2020

Conteúdo

- 1 Escassez de memória RAM
- 2 Mecanismo de paginação em disco
- 3 Algoritmos de substituição de páginas
- 4 Aproximações do algoritmo LRU
- 5 Anomalia de Belady
- 6 Thrashing

Uso da memória RAM

Memória é um recurso precioso:

- Programas cada vez maiores
- Mais processos na memória
- Memória RAM é cara (\sim U\$10 / GByte)

Contudo:

- Nem todos os processos estão ativos
- Nem todas as áreas de memória são usadas
- Discos são baratos (\sim U\$0,05 - 0,30 / GByte)

Ideia: usar um disco como **extensão** da memória

Estendendo a memória RAM

Conceito:

- Transferir partes da RAM para um disco
- Buscar o conteúdo no disco quando necessário
- Usar a MMU para tornar tudo transparente

Abordagens:

- **Overlays**
- **Swapping**
- **Paging**

Abordagens

■ **Overlays:**

- Programador divide o executável em módulos
- Somente um módulo é carregado na memória
- Popular nos anos 1970-90 (Turbo Pascal, Delphi)

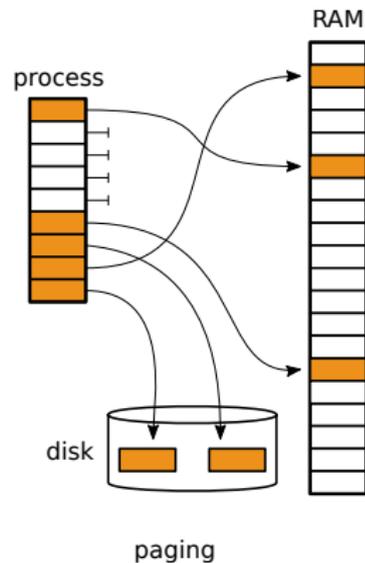
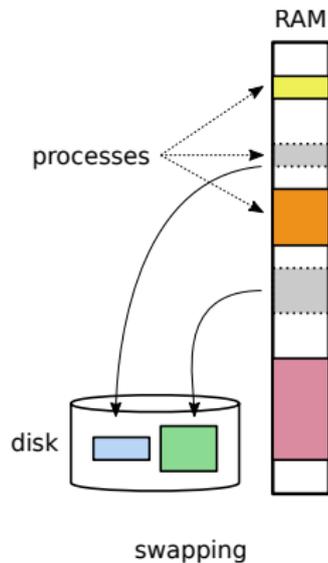
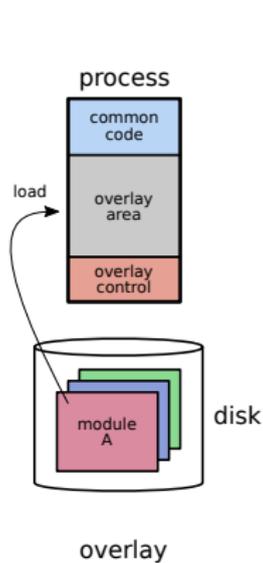
■ **Swapping:**

- Processos suspensos são transferidos para o disco
- Voltam para a memória ao acordar

■ **Paging:**

- Páginas pouco usadas saem da memória
- *Page faults* trazem as páginas de volta
- É a técnica mais usada hoje em dia

Abordagens



Paginação em disco

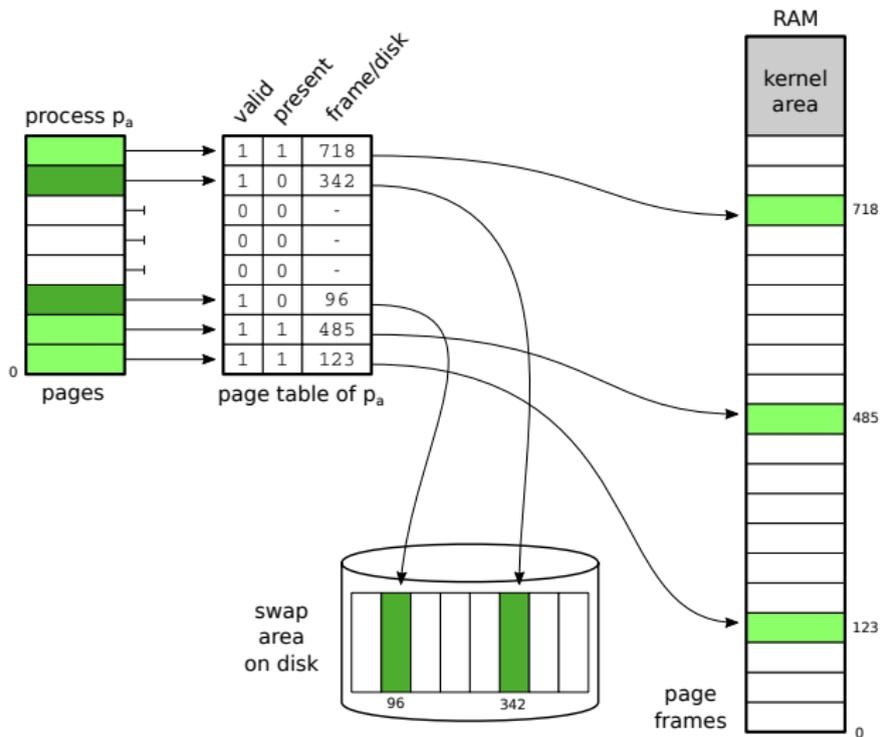
Princípio:

- Mover páginas ociosas da RAM para o disco
- Ajustar tabela de páginas para indicar isso
- Se página for acessada, gerar *page fault*
- Trazer página de volta para a RAM

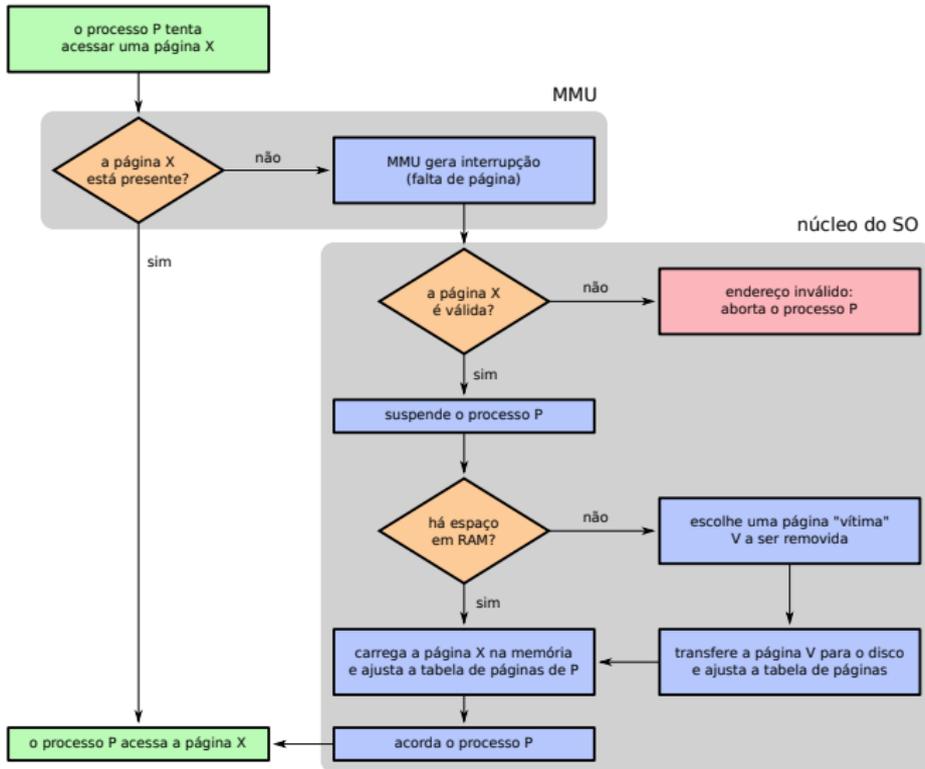
Implementação:

- Núcleo escolhe páginas ociosas
- Núcleo move as páginas entre RAM e disco
- MMU detecta acessos a páginas no disco

Paginação em disco



Dinâmica da paginação



Desempenho da memória virtual

Problema: discos são **lentos**: $t_{disk} \gg t_{ram}$

Considerando $t_{disk} = 6ms$ e $t_{ram} = 60ns$

- uma falta de página em 1.000.000 acessos à RAM:

$$\begin{aligned}
 t_{medio} &= \frac{(999.999 \times 60ns) + 6ms + 60ns}{1.000.000} \\
 &= 66ns
 \end{aligned}$$

- uma *fp* em 100.000 acessos $\rightarrow t_{medio} = 120ns$

Frequência de faltas de páginas

Fatores que a influenciam:

- A **quantidade de RAM** em relação à demanda
- A forma como os processos **acessam** a memória
 - Localidade de referências
- A **escolha das páginas** a tirar da memória

Escolha das páginas:

- Escolhas ruins podem prejudicar o desempenho
- Algoritmos de **substituição** de páginas

Escolha de páginas “vítimas”

Fatores a considerar na escolha:

- Idade da página na memória
- Frequência de acessos à página
- Data do último acesso
- Prioridade do processo dono
- Conteúdo da página (código ou dados)
- Páginas especiais (*buffers*, conteúdos sensíveis)
- ...

Cadeia de referências

- Sequência de páginas acessadas durante uma execução
- Usada para estudar algoritmos de substituição
- Cadeias reais são muito longas (milhões refs/segundo)

Cadeia usada neste texto:

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

Algoritmos clássicos

FIFO - *First-In, First-Out*

- Solução mais simples
- Desempenho fraco

ÓTIMO

- Melhor solução teórica
- Impossível de implementar

LRU - *Least Recently Used*

- Melhor solução implementável

Algoritmo FIFO

Ideia básica:

- Considerar o tempo em que as páginas estão na memória
- Retirar da RAM as páginas mais antigas

Implementação: uma **fila de números** de páginas em RAM

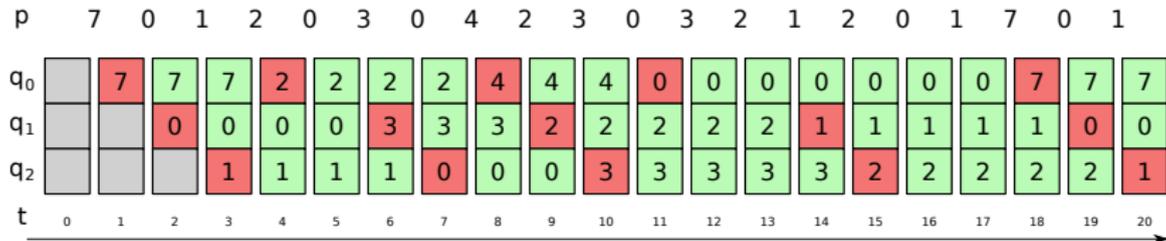
Problemas:

- Não considera o **uso** das páginas
- Processos antigos podem sofrer mais
- Não oferece bons resultados

Algoritmo FIFO

t	página	quadros			fp	ação realizada
		q_0	q_1	q_2		
0						situação inicial, quadros vazios
1	7	7			✓	p_7 é carregada em q_0
2	0	7	0		✓	p_0 é carregada em q_1
3	1	7	0	1	✓	p_1 é carregada em q_2
4	2	2	0	1	✓	p_2 substitui p_7 (carregada em $t = 1$)
5	0	2	0	1		p_0 já está na memória
6	3	2	3	1	✓	p_3 substitui p_0
7	0	2	3	0	✓	p_0 substitui p_1
8	4	4	3	0	✓	p_4 substitui p_2
						...

Algoritmo FIFO



O algoritmo FIFO gera 15 faltas de página.

Algoritmo Ótimo

Critério

Retirar a página que ficará **mais tempo** sem ser acessada.

Problema

Como prever o **comportamento futuro** dos processos?

→ o algoritmo ótimo **não é implementável !**

Mas ele serve como limite conceitual:

Se o algoritmo ótimo gera N faltas de página, nenhum algoritmo irá gerar $M < N$ faltas na mesma situação.

Algoritmo Ótimo

t	página	quadros			fp	ação realizada
		q_0	q_1	q_2		
0						situação inicial, quadros vazios
1	7	7			✓	p_7 é carregada em q_0
2	0	7	0		✓	p_0 é carregada em q_1
3	1	7	0	1	✓	p_1 é carregada em q_2
4	2	2	0	1	✓	p_2 substitui p_7 (só é usada em $t = 18$)
5	0	2	0	1		p_0 já está na memória
6	3	2	0	3	✓	p_3 substitui p_1
7	0	2	0	3		p_0 já está na memória
8	4	2	4	3	✓	p_4 substitui p_0
						...

Algoritmo Ótimo

p	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	
q ₀	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7	
q ₁	0	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0	
q ₂	1	1	1	3	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1	1	
t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

O algoritmo ótimo gera 9 faltas de página.

Algoritmo LRU

Localidade de referências:

- páginas com uso recente podem ser usadas em breve

Ideia:

- Retirar da RAM as páginas há mais tempo **sem uso**
- *Least Recently used*

O LRU é simétrico ao OPT em relação ao tempo:

enquanto o OPT busca as páginas que serão acessadas “mais longe” no futuro, o algoritmo LRU busca as páginas que foram acessadas “mais longe” no passado.

Algoritmo LRU

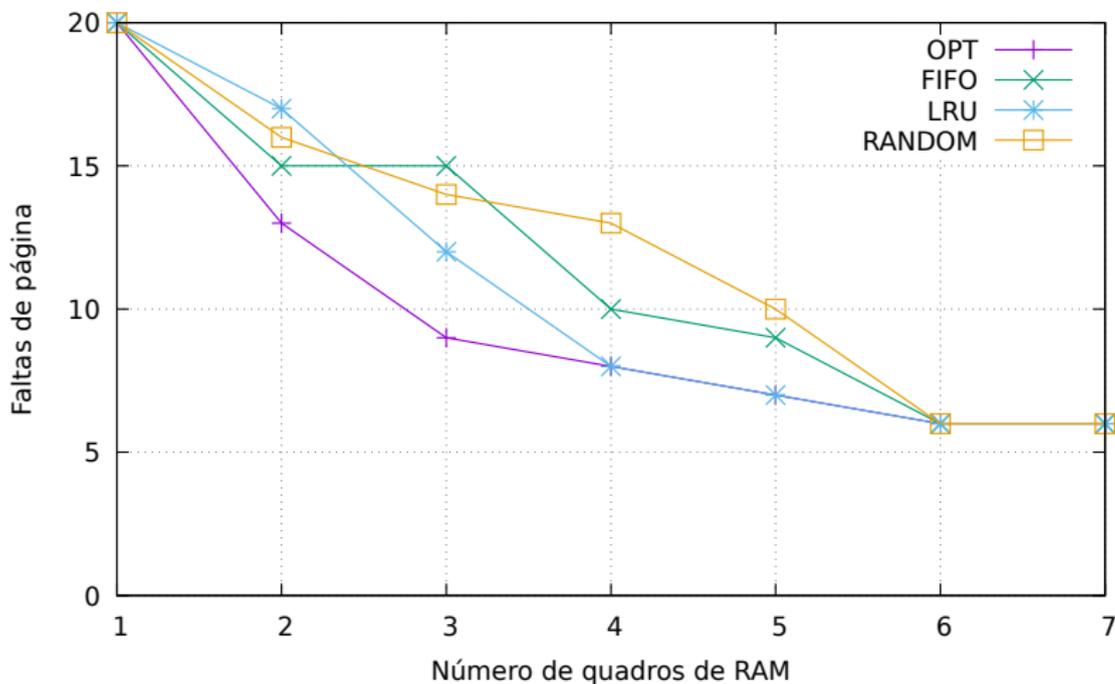
t	página	quadros			fp	ação realizada
		q_0	q_1	q_2		
0						situação inicial, quadros vazios
1	7	7			✓	p_7 é carregada em q_0
2	0	7	0		✓	p_0 é carregada em q_1
3	1	7	0	1	✓	p_1 é carregada em q_2
4	2	2	0	1	✓	p_2 substitui p_7 (mais tempo sem uso)
5	0	2	0	1		p_0 já está na memória
6	3	2	0	3	✓	p_3 substitui p_1
7	0	2	0	3		p_0 já está na memória
8	4	4	0	3	✓	p_4 substitui p_2
						...

Algoritmo LRU

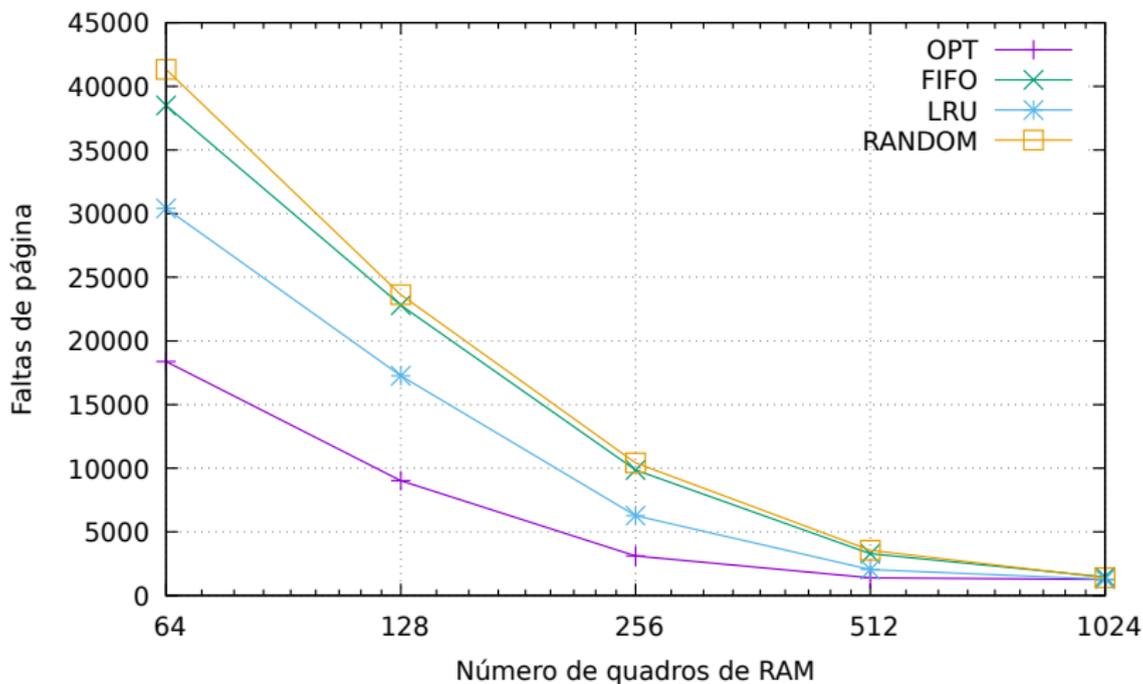
p	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	
q ₀	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1	
q ₁	0	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0	
q ₂	1	1	1	3	3	3	3	2	2	2	2	2	2	2	2	2	2	7	7	7	
t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

O algoritmo LRU gera 12 faltas de página.

Comparativo: string de referência



Comparativo: compilador GCC



Aproximações do algoritmo LRU

LRU completo é inviável:

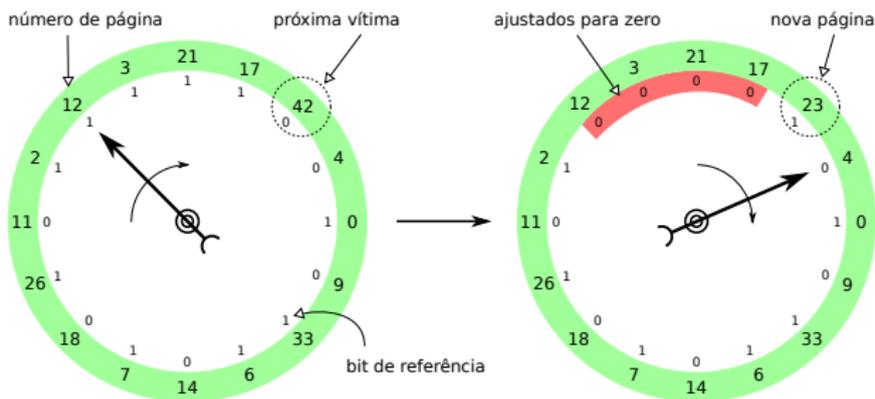
- registrar as datas de acesso às páginas a cada acesso
- encontrar a página com data de acesso mais antiga

Aproximações do LRU usadas na prática:

- Algoritmo do relógio (ou da 2ª chance)
- *Not-Recently Used* (NRU)
- Envelhecimento (ou dos contadores)
- *Working sets*

Algoritmo do relógio (ou da 2ª chance)

- Baseado no algoritmo FIFO
- Analisa o bit de referência de cada página candidata
- Dá “Segunda chance” às páginas usadas recentemente
- Evita substituir páginas antigas mas muito acessadas



Algoritmo *Not Recently Used* - NRU

Usa flags de referência (R) e modificação (M) das páginas.

Bits R e M definem níveis de importância:

R	M	status
0	0	nem usada nem modificada; melhor escolha!
0	1	sem uso mas alterada (salvar antes de remover)
1	0	usada recentemente
1	1	usada recentemente e alterada, pior escolha!

Algoritmo: substituir primeiro páginas do nível mais baixo (00).

Algoritmo do Envelhecimento

Ideia: usar o bit R para construir *contadores de acesso*:

- Um contador inteiro com N bits para cada página
- O bit R é usado para atualizar o contador
- Periodicamente, para cada página:
 - 1 os bits do contador são deslocados para a direita
 - 2 valor do bit R é usado como novo *Most Significant Bit*
 - 3 O bit R é ajustado para zero
- Acessos mais recentes têm maior peso

Algoritmo do Envelhecimento

$$R + \overbrace{(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)}^{\text{contador}} \rightarrow \overbrace{(R, b_7, b_6, b_5, b_4, b_3, b_2, b_1)}^{\text{novo contador}}$$

$$\begin{array}{l}
 p_1 \\
 p_2 \\
 p_3 \\
 p_4
 \end{array}
 \begin{bmatrix}
 R \\
 0 \\
 1 \\
 0 \\
 1
 \end{bmatrix}
 \begin{bmatrix}
 \text{contadores} \\
 0000 \ 0011 \ (3) \\
 0011 \ 1101 \ (61) \\
 1010 \ 1000 \ (168) \\
 1110 \ 0011 \ (227)
 \end{bmatrix}
 \Rightarrow
 \begin{bmatrix}
 R \\
 0 \\
 0 \\
 0 \\
 0
 \end{bmatrix}
 \begin{bmatrix}
 \text{contadores} \\
 0000 \ 0001 \ (1) \\
 1001 \ 1110 \ (158) \\
 0101 \ 0100 \ (84) \\
 1111 \ 0001 \ (241)
 \end{bmatrix}$$

Conjunto de trabalho

Definição:

Conjunto de páginas acessadas recentemente pelo processo.

Propriedades:

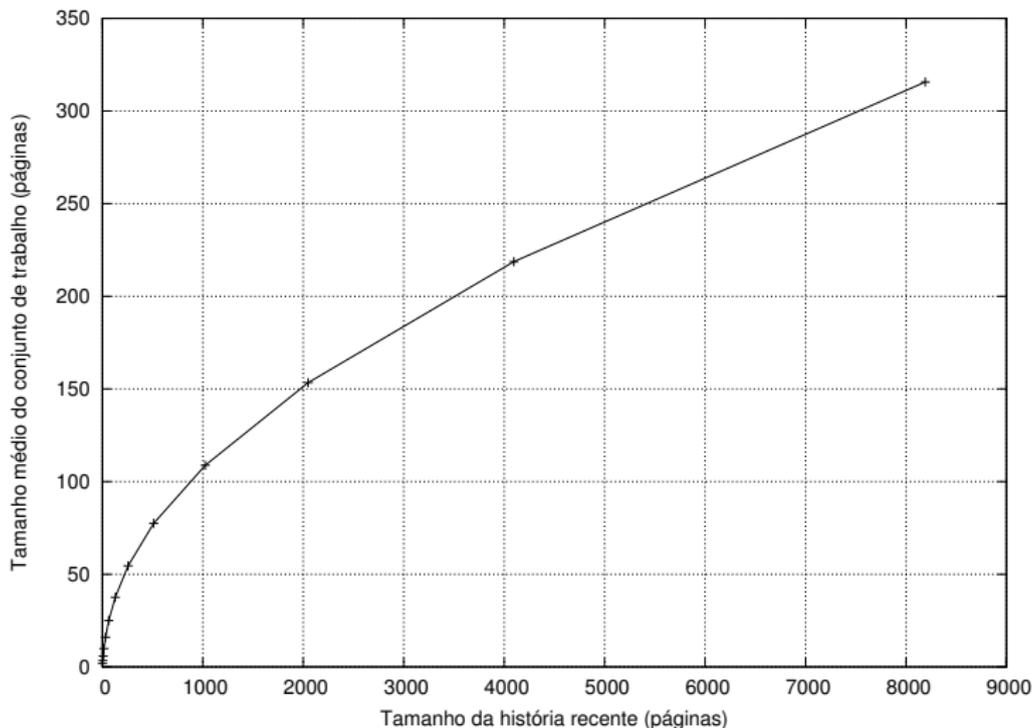
- Geralmente pequeno, devido à localidade de referências
- Dinâmico, varia com a evolução do processo
- Se o processo tiver seu conjunto de trabalho na memória, terá poucas faltas de página

Algoritmo: substituir páginas que não pertençam ao conjunto de trabalho de nenhum processo ativo (difícil de implementar)

Conjuntos de trabalho

t	página	history = 3	history = 4	history = 5
1	7	7	7	7
2	0	0, 7	0, 7	0, 7
3	1	0, 1, 7	0, 1, 7	0, 1, 7
4	2	0, 1, 2	0, 1, 2, 7	0, 1, 2, 7
5	0	0, 1, 2	0, 1, 2	0, 1, 2, 7
6	3	0, 2, 3	0, 1, 2, 3	0, 1, 2, 3
7	0	0, 3	0, 2, 3	0, 1, 2, 3
8	4	0, 3, 4	0, 3, 4	0, 2, 3, 4
9	2	0, 2, 4	0, 2, 3, 4	0, 2, 3, 4
10	3	2, 3, 4	0, 2, 3, 4	0, 2, 3, 4
11	0	0, 2, 3	0, 2, 3, 4	0, 2, 3, 4
12	3	0, 3	0, 2, 3	0, 2, 3, 4

Conjunto de trabalho do Gnome GThumb



Algoritmo Working Set Clock (WSClock)

Determinar os *Working Sets* diretamente é caro.

WSClock: aproximação de WS usando o algoritmo do relógio:

- Cada página tem uma data de último acesso $t_a(p)$
- Cada página tem uma idade $i(p) = t_{now} - t_a(p)$
- $t_a(p)$ é atualizada quando o ponteiro do relógio a visitar
- As páginas têm um prazo de validade τ

Princípio do algoritmo:

- Visitar as páginas usando o algoritmo do relógio
- Remover as páginas não-acessadas e fora da validade

Algoritmo Working Set Clock (WSClock)

O ponteiro percorre a fila e analisa cada página p :

- 1 Se $R(p) = 1$ (a página foi acessada):
 - $t_a(p) = t_{now}$ (atualiza a data de acesso)
 - $R(p) = 0$ (limpa o bit de referência)
- 2 Se $R(p) = 0$:
 - se $i(p) \leq \tau$, a página está no conjunto de trabalho
 - se não estiver:
 - Se $M(p) = 0$ a página pode ser removida
 - Se $M(p) = 1$ agenda-se a escrita da página
- 3 Se não achar página com $i(p) > \tau$, $R(p) = 0$ e $M(p) = 0$:
 - 1 remover a página mais antiga com $M(p) = 0$ e $R(p) = 0$
 - 2 ... (usando NRU)

A anomalia de Belady

Intuição: quanto mais RAM, menos faltas de páginas

Problema:

- Esse comportamento **não se verifica** sempre
- Alguns algoritmos têm comportamento anômalo

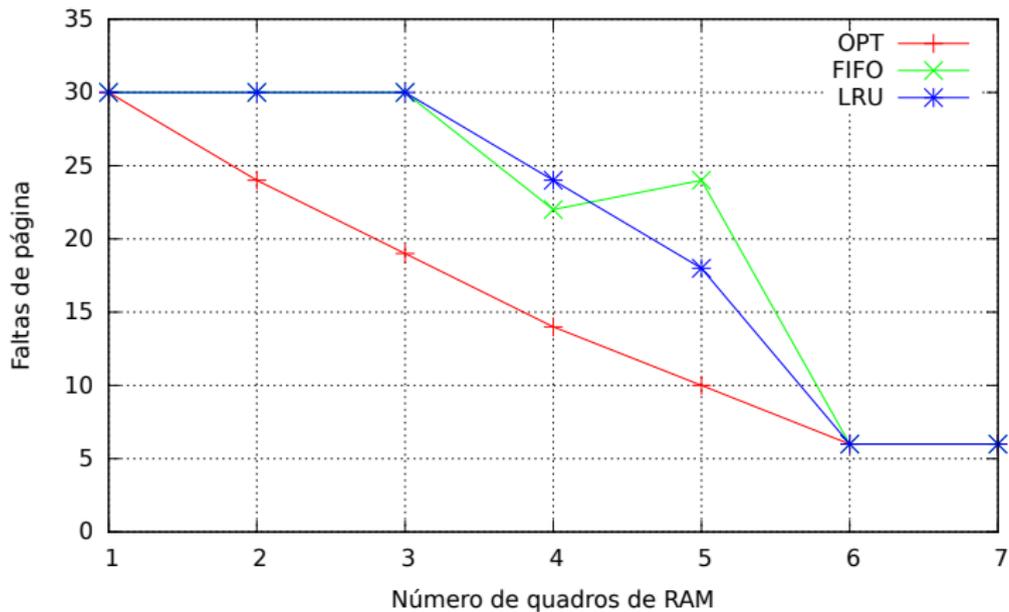
Anomalia de Belady

Ao aumentar o número de quadros de RAM, o número de faltas de página **umenta**, ao invés de diminuir.

Exemplo de cadeia de referências:

0, 1, 2, 3, 4, 0, 1, 2, 5, 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 0, 1, 2, 5, 0, 1, 2, 3, 4, 5

A anomalia de Belady



Thrashing

Saturação no uso da RAM:

- Muitos processos ativos
- Os processos não têm RAM para seus WSets
- Cada processo gera muitas faltas de página

Consequências:

- Processos ficam suspensos a maior parte do tempo
- A paginação toma a maior parte do tempo do sistema

Solução: diminuir o número de processos ativos

Thrashing

