Laboratório de Gerenciamento de Memória Virtual, Uma Análise Detalhada dos Algoritmos de Alocação *

Fladson Thiago Oliveira Gomes[1], Kelvin Kirk Miná Cavalcante[1]

[1]Departamento de Ciências Exatas e Aplicadas
Universidade Federal do Rio Grande do Norte (UFRN) – Caicó, RN – Brasil

fladsonthiago@gmail.com, kelvin.kirk@gmail.com

Abstract. *This meta-paper describes a comparative analysis of page replacements algorithms in virtual memory. There are several algorithms for choosing to replace pages in virtual memory, the main test will be covered in a style 'benchmark' by comparing and analyzing their efficiency and main aspects.*

Resumo. *Este meta artigo descreve a análise comparativa de algoritmos de substituição de páginas em memória virtual. Existem vários algoritmos para a escolha de páginas a substituir na memória virtual, os principais serão abordados em um teste estilo 'benchmark', comparando sua eficiência e analisando os seus principais aspectos.*

1.Introdução

A falta de memória física é um problema conhecido e constante nos sistemas operacionais modernos, e como a memória RAM é um recurso caro e que consome uma quantidade considerável de energia, nem sempre aumentar a sua capacidade é uma opção viável. A memória virtual surgiu deste empecilho e também da necessidade de ter mais processos do que o limite da memória física rodando, pois é constatado que nem sempre todas as áreas de memória estão constantemente sendo usadas e que nem os processos estão constantemente ativos, a análise demonstra como os algoritmos de alocação, Random, FIFO (First In Firts Out) e o LRU (Least Recently Used), operam visando um maior desempenho na alocação de páginas a substituir na memória, visando reduzir a frequência de falta de páginas.

2.Memória Virtual

Ao observar o comportamento de um sistema, constatamos que boa parte dos processos não está constantemente ativa, consequentemente nem todas as áreas da memória estão sendo usadas. Essas áreas pouco usadas poderiam ser transferidas para meios de armazenamento mais barato como um Disco Rígido ou uma Memória Flash, quando for necessário utiliza-las o sistema a transfere novamente para a memoria RAM. Essa estratégia é denominada Memoria Virtual.

---

* Adaptação para OpenOffice.org 1.1 feita por Roland Teodorowitsch (roland@ulbra.tche.br) em 29 mar. 2005.

Nos Primeiros usos da memoria virtual utilizava-se swapping que transfere todo o processo da memoria para a unidade de armazenamento auxiliar, atualmente os sistemas não transferem processos inteiros, geralmente utilizam a transferência por pagina denominado paginação. As paginas, ao contrario dos segmentos tem tamanho fixo o que simplifica os algoritmos para escolhas de paginas a remover. O mecanismo de memoria virtual consiste em retirar quadros pouco usados da memoria principal e salva-los em uma área do disco rígido, as entradas das tabelas de paginas relativas devem ser ajustadas de forma a referenciar os conteúdos correspondentes no disco rígido. Podemos visualizar essa situação na figura 1.

3.Simulador de Memória Virtual

O Simulador de Memória Virtual utilizado para a análise é implementado na linguagem C e foi projetado para simular ações do hardware de paginação e da rotina de tratamento de interrupção por falta de páginas. O simulador usa como argumento um arquivo que contém sequências de referências à memória, são linhas que representam endereços virtuais(32439800) e a sua operação (R para Operações de Leitura e W para Operações de Escrita). Exemplo de uma referência à memória contida no arquivo, 32439800 R.

Para um melhor entendimento sobre a execução do simulador é necessário ter conhecimento de algumas condições:

- O processo poderá ter no máximo 2^20 páginas virtuais;
- Cada página terá 2^12 bytes, onde a tabela de páginas é simulada por um vetor de registros que para cada página contém o bit de validade e o número do frame em que a página está carregada;
- A memória está dividida em frames de mesma dimensão;
- O número de frames que o processo pode ocupar é um dos parâmetro que o simulador recebe em linha de comando;
- A tabela de frames é simulada por um vetor de registros que para cada frame contém o número da página armazenada, o bit de referência, o bit de modificação, a data de carregamento da página e a data da última referência ao frame;
- As páginas vão sendo carregadas a medida que vão sendo referenciadas;
- A substituição da página vítima é feita por um algoritmo de substituição de páginas que deve ser definido na execução do simulador. Os algoritmos disponíveis são o Random o FIFO e o LRU.

4.Implementação dos Algoritmos

Os algoritmos de substituição são muito importantes para que a memória virtual tenha um desempenho satisfatório. Para esta análise foram implementados os algortimos FIFO, LRU e Random. Abaixo uma explanação de como eles foram implementados:

A implementação do FIFO (*First In First Out*) se baseia na premissa de que as páginas que estão há mais tempo na memória não serão utilizadas em um futuro próximo, por isso o algoritmo é implementado para vasculhar a tabela de páginas em busca dessas páginas 'velhas' para que sejam substituídas pela nova página, isso é possível graças ao campo 'load_time', presente na estrutura de cada página. O algoritmo percorre a tabela de páginas que estão carregadas na memória e vai comparando o 'load_time' de cada página até encontrar a que possuí o maior valor de 'load_time', o passo seguinte é atribuir a página vítima o índice de página que teve o maior 'load_time', feito isso a página vítima a ser substituída pela alocação será a que estava a mais tempo na memória.

```
149 int
150 selectVictimFifo (void)
151 {
152   /* estrategia de substituicao que escolhe como vitima a
153      frame que esta ha mais tempo carregada em memoria
154    */
155
156   int victim = 0;
157
158
159   int i =1;
160
161   while(i < n_frames){
162       if(frameTable[i].load_time < frameTable[victim].load_time){
163           victim = i;
164       }
165       i++;
166   }
167
168   return victim;
169 }
```

Figura 2. Trecho de código FIFO

Na implementação do LRU (*Least Recently Used*) a escolha da dita página vítima recai sobre as páginas que estão na memória há mais tempo sem ser acessadas. As páginas antigas e não muito utilizadas serão o alvo deste algoritmo, ou seja, páginas que tenham o 'load_time' alto mas que possuam acesso frequente não serão escolhidas para alocação. Aqui a tabela de páginas também será percorrida, mas o critério de comparação será o 'last_reference_time' que nada mais é que um campo do tipo inteiro que acumula a quantidade de referências a uma página.

```
172 int
173 selectVictimLRU (void)
174 {
175   /* estrategia de substituicao que escolhe como vitima a
176      frame que esta ha mais tempo sem ser referenciada
177    */
178
179   int victim = 0;
180
181   int i =1;
182   while(i < n_frames){
183       if(frameTable[i].last_reference_time < frameTable[victim].last_reference_time){
184             victim = i;
185       }
186       i++;
187   }
188
189   return victim;
190 }
```

Figura 3. Trecho de código LRU

O Random não escolhe critério nenhum para determinar qual será a página vítima, o índice da página escolhida é obtido de uma função randômica usual, apenas atentando para o fato de ser necessário dividir esse número obtido pelo número de páginas, para não ter o problema de o índice estar fora dos limites da tabela.

```
136 int
137 selectVictimRandom (void)
138 {
139   /* estrategia de substituicao que escolhe como vitima
140      uma frame ao acaso
141    */
142   int victim;
143
144   victim = random () % n_frames;
145
146   return victim;
147 }
```

Figura 4. Trecho de código Random

5.Análise Comparativa

Para possibilitar a comparação dos algoritmos de substituição de páginas apresentados na sequência, serão usados arquivos com 1 milhão de referências a memória cada.

Deve-se atentar para os números de pages fault e pages write que cada algoritmo gerou, pois esses fatores é quem irão determinar a eficiência de um dado algoritmo.

5.1. FIFO

Apesar da implementação ser simples o FIFO na prática não oferece bons resultados, sua principal fraqueza é o fato de ele considerar apenas a idade da página, sem levar em consideração a sua importância. Páginas carregadas na memória há muito tempo podem estar sendo frequentemente acessadas, e o FIFO simplesmente ignora esse fato.

bzip.trace		
Frames	Page Faults	Pages Written
8	47828	18797
16	3820	1335
32	2497	851
64	1467	514

sixpack.trace		
Frames	Page Faults	Pages Written
8	230168	57121
16	140083	31314
32	85283	18805
64	48301	11936

gcc.trace		
Frames	Page Faults	Pages Written
8	205368	18797
16	138539	24043
32	98067	16759
64	70315	12053

swin.trace		
Frames	Page Faults	Pages Written
8	330893	55488
16	214295	47434
32	81638	18172
64	30422	7585

Figura 5. Tabela de Resultados FIFO

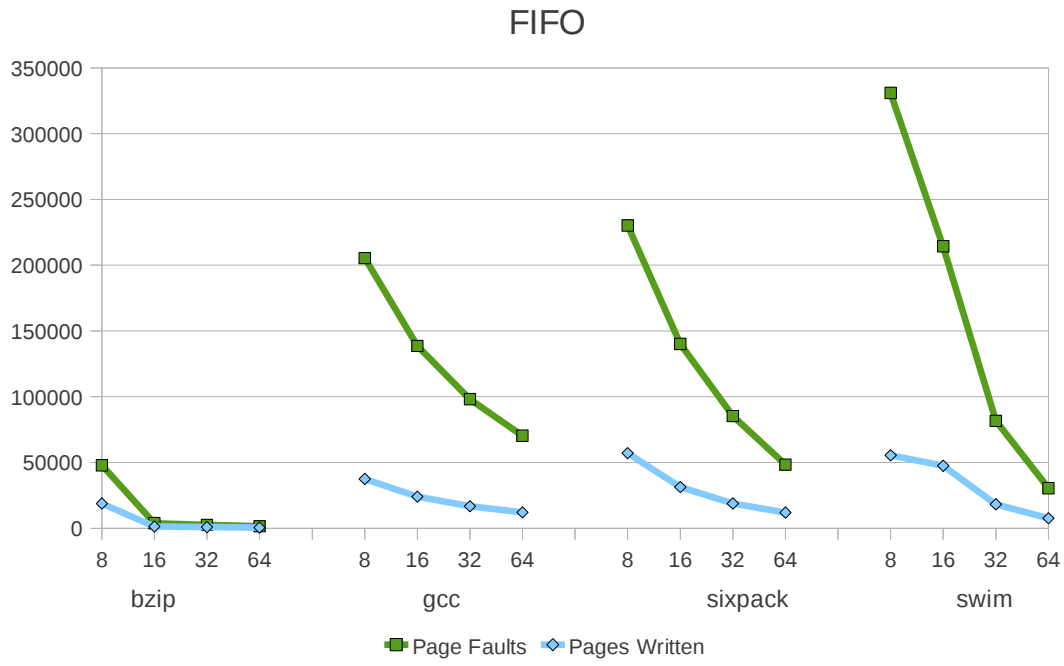

Figura 6. Gráfico de Resultados FIFO

5.2. LRU

O LRU foi o algoritmo que se saiu melhor entre os três aqui analisados, isso se deve ao pressuposto de manter páginas de uso frequente na memória, diminuindo muito a quantidade de pages fault e pages writen.

> *"Embora possa ser implementado, o algoritmo LRU básico é pouco usado na prática, porque sua implementação exigiria registrar as datas de acesso às páginas a cada leitura ou escrita na memória, o que é difícil de implementar de forma eficiente em software e com custo proibitivo para implementar em hardware. Além disso, sua implementação exigiria varrer as datas de acesso de todas as páginas para buscar a página com acesso mais antigo (ou manter uma lista de páginas ordenadas por data de acesso), o que exigiria muito processamento. Assim, na prática a maioria dos sistemas operacionais implementam aproximações do LRU, como as apresentadas na seqüência. " Carlos Alberto Maziero, pág 41.*

bzip.trace		
Frames	Page Faults	Pages Written
8	30691	11092
16	3344	1069
32	2133	702
64	1264	420

sixpack.trace		
Frames	Page Faults	Pages Written
8	176496	32717
16	108682	19342
32	67747	13730
64	41186	9672

gcc.trace		
Frames	Page Faults	Pages Written
8	171186	23983
16	116604	14749
32	84401	11737
64	59089	8863

swin.trace		
Frames	Page Faults	Pages Written
8	285375	53614
16	171961	46293
32	48254	9674
64	21656	5687

Figura 7. Tabela de Resultados LRU

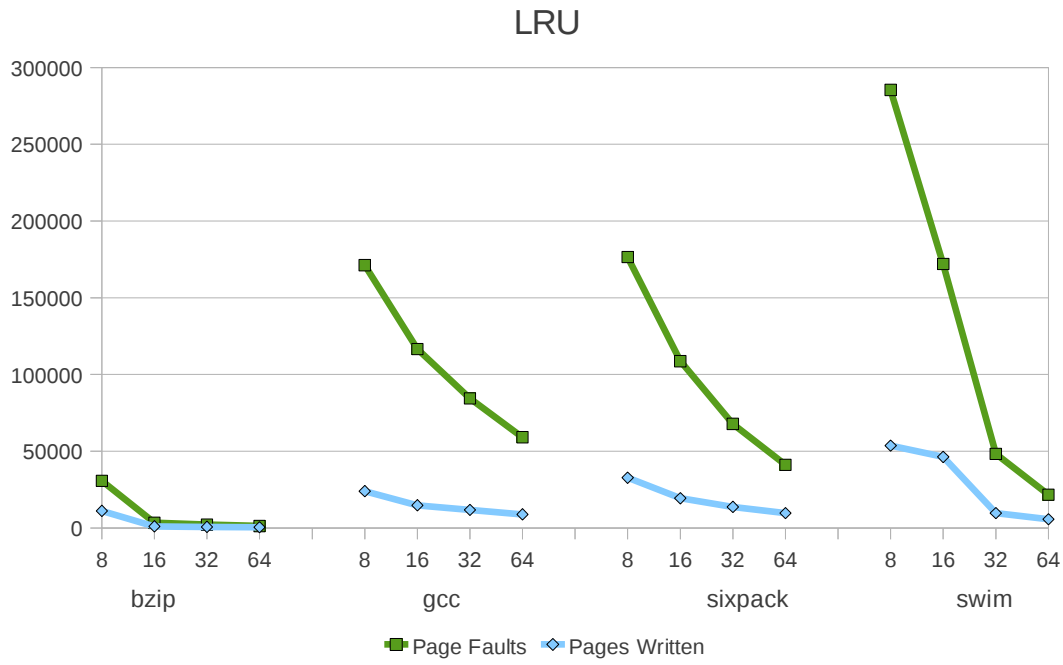

Figura 8. Tabela de Resultados LRU

5.3. Random

O random não pode ser analisado a fundo pois não há critérios para substiruição das páginas, mas em alguns momentos o algoritmo se saiu melhor que o FIFO, mostrando que é bastante estável mesmo que os índices sejam escolhidos ao acaso.

bzip.trace		
Frames	Page Faults	Pages Written
8	44085	16473
16	4524	1500
32	2603	880
64	1585	554

sixpack.trace		
Frames	Page Faults	Pages Written
8	231149	54751
16	145110	32036
32	88549	19554
64	52024	12652

gcc.trace		
Frames	Page Faults	Pages Written
8	216505	37350
16	150285	24727
32	106171	17112
64	75155	12247

swin.trace		
Frames	Page Faults	Pages Written
8	321824	54510
16	195456	40081
32	83815	18323
64	35558	8562

Figura 9. Tabela de Resultados Random

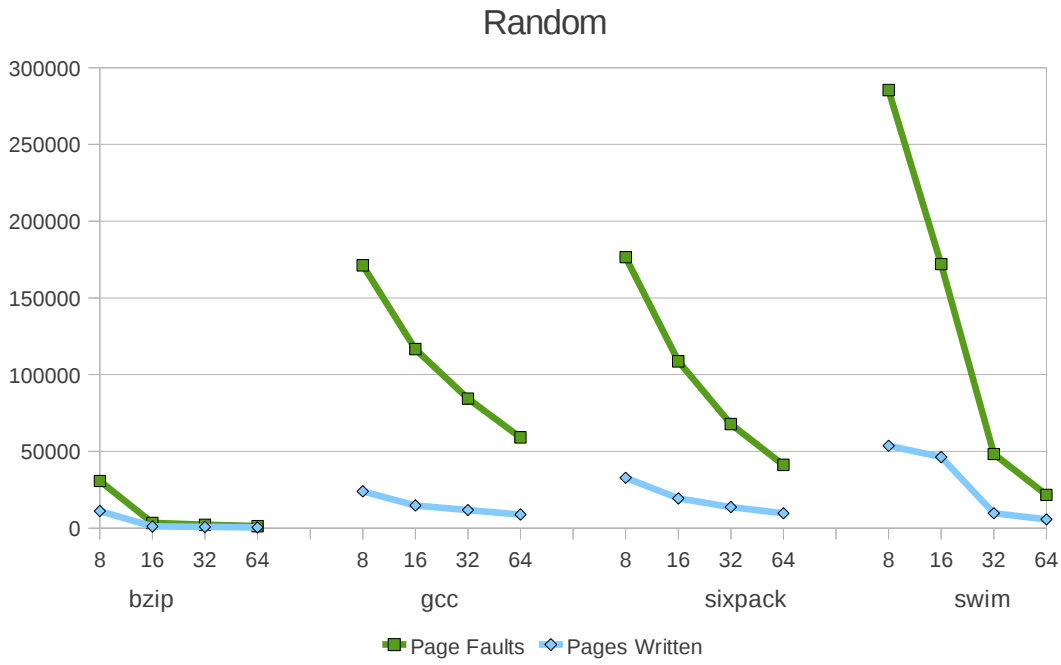

Figura 10. Gráfico de Resultados Random

6.Conclusão

A escolha correta das páginas a remover da memória física é um fator essencial para a eficiência do mecanismo de memória virtual. Más escolhas poderão remover da memória páginas muito usadas, aumentando a taxa de faltas de página e diminuindo o desempenho do sistema.

7.Figuras

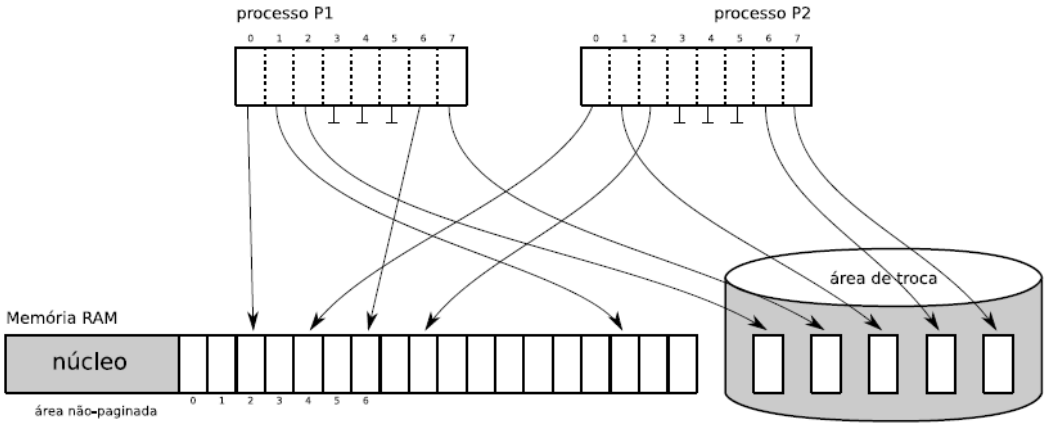

Figura 1. Memória Virtual Paginada

References

Maziero, Carlos Alberto (2009) “Sistemas Operacionais V - Gerência de Memória ”, http://www.ppgia.pucpr.br/∼maziero

Oliveira, Rômulo Silva de; Carissimi, Alexandre da Silva; Toscani, Simão Sirineo (2010) “Sistemas Operacionais” 4ª Edição editora Bookman

[Tanenbaum, 2003] Tanenbaum, A. (2003). Sistemas Operacionais Modernos, 2a edição. Pearson – Prentice-Hall.