Análise de Algoritmos de Substituição de Páginas a partir de Traces de Aplicações

Igor Correa
Unioeste, Campus Cascavel
Cascavel, Brazil
igor.almeida6@unioeste.br

Guilherme Altemeyer
Unioeste, Campus Cascavel
Cascavel, Brazil
altmeyergui@gmail.com

Maria Quevedo
Unioeste, Campus Cascavel
Cascavel, Brazil
maria.quevedo2@unioeste

I. Introdução

A gerência de memória virtual é um pilar dos sistemas operacionais modernos, permitindo a execução de processos em um espaço de endereçamento superior à memória física disponível[1]. Contudo, o desempenho desses sistemas é diretamente impactado pela frequência de falhas de página (*page faults*), que ocorrem quando um dado necessário não está na memória principal. A política de substituição de páginas é crucial para minimizar essas falhas.

O presente artigo analisa a taxa de falhas de página em sistemas de memória virtual por meio da comparação de dois algoritmos de substituição: o Ótimo e o FIFO (First-In, First-Out). A análise empírica foi conduzida a partir de traces de acesso à memória, gerados com a ferramenta *Valgrind*[2], durante a execução de três aplicações distintas: o algoritmo de Dijkstra, um programa "Hello World" em C, e um algoritmo sequencial. Estes traces foram processados para gerar uma *reference string*, que serviu como entrada ao código desenvolvido para calcular o número de falhas de página, permitindo uma avaliação quantitativa do desempenho de cada algoritmo sob as mesmas condições de acesso.

II. Fundamentação Teórica

A memória virtual é uma técnica amplamente utilizada pelos sistemas operacionais modernos, permitindo que programas sejam executados mesmo quando a memória física disponível é insuficiente para armazenar todos os seus dados e instruções. Um dos métodos mais comuns de gerenciamento da memória virtual é a paginação por demanda, em que as páginas de um processo são carregadas na memória principal somente quando são efetivamente necessárias durante a execução. Dessa forma, evita-se trazer o processo inteiro para a memória, otimizando o uso dos recursos e possibilitando a execução de programas maiores do que a memória física instalada[1].

Quando um processo tenta acessar uma página que ainda não se encontra na memória, ocorre uma interrupção conhecida como falha ou erro de página. Nessa situação, o sistema operacional precisa buscar a página requerida no disco e carregá-la na memória, antes de retomar a execução da instrução interrompida. Esse processo, no entanto, acarreta um custo elevado em termos de desempenho, visto que a leitura em disco é significativamente mais lenta que o acesso direto à memória. O tempo de acesso efetivo pode ser estimado pela expressão:

$$EAT = (1 - p) \, . \, ma + p \, . \, tpf$$

onde *EAT* é o tempo médio de acesso, *ma* é o tempo de acesso à memória, p é a taxa de falhas de página e *tpf* é o tempo associado ao tratamento da falha de página, incluindo o tempo de E/S em disco[1]. Assim, quanto maior a taxa de falhas, mais comprometido estará o desempenho do sistema.

Para reduzir o impacto das falhas de página, torna-se necessário um bom algoritmo de substituição de páginas, responsável por decidir qual página será removida da memória quando não houver mais espaço disponível. O algoritmo mais simples é o FIFO (First-In, First-Out), que substitui a página mais antiga, isto é, a que está na memória há mais tempo. Embora seja de fácil implementação, esse algoritmo pode apresentar comportamentos inesperados, como a chamada anomalia de Belady, em que a taxa de falhas de página aumenta à medida que se aumenta o número de quadros de memória disponíveis[1].

A descoberta dessa anomalia levou ao estudo de algoritmos mais sofisticados, como o algoritmo ótimo de substituição de páginas, que substitui a página que não será utilizada por mais tempo no futuro. Esse algoritmo garante a menor taxa possível de falhas de página e nunca sofre da anomalia de *Belady*. Contudo, sua implementação prática é inviável, pois exige conhecimento prévio da sequência completa de referências de memória, sendo empregado principalmente como referência teórica em estudos comparativos [1].

III. Metodologia

A condução deste estudo envolveu três etapas principais: a geração dos traces de acesso à memória, a construção da *reference string* a partir dessas traces e a simulação dos algoritmos de substituição de páginas.

A. Geração dos traces

Para coletar os acessos à memória, utilizou-se a ferramenta *Valgrind*, especificamente o módulo *Lackey*, que permite registrar todas as instruções e acessos a dados realizados por um programa durante sua execução. O processo consistiu em compilar as aplicações desenvolvidas e em seguida, executá-las sob o *Valgrind* com o parâmetro `--trace-mem=yes`, redirecionando a saída para arquivos de trace.

As três aplicações selecionadas foram:

- *Hello World* em C: produz um trace reduzido (aproximadamente 200 mil acessos à memória).
- Algoritmo de *Dijkstra*, que envolve operações em grafos e gera um trace mais representativo (aproximadamente 2 milhões e 600 mil acessos).
- Simulação bancária sequencial, que realiza depósitos, saques e transações entre contas,

resultando em um trace extenso (Aproximadamente 12 milhões e 100 mil acessos).

Essa diversidade de aplicações permitiu analisar algoritmos de substituição em contextos de acesso simples, intermediário e intensivo à memória

B. Construção da reference string

Cada linha do trace representa um acesso a instruções ou dados em um endereço de memória. Considerando que o tamanho da página adotado foi de 4096 *bytes* e o espaço de endereçamento de 48 *bits*, foram utilizados 12 *bits* para o deslocamento e 36 *bits* para o número da página. Dessa forma, cada endereço foi convertido no número de página correspondente, gerando a *reference string* utilizada como entrada para o simulador.

C. Simulador de substituição de páginas

O simulador foi implementado em C++. Ele recebe como entrada o arquivo de *reference string* e o número de quadros de memória disponíveis.

Foram implementados dois algoritmos de substituição de páginas:

- Ótimo: substitui a página cuja próxima utilização ocorrerá mais distante no futuro, garantindo a menor taxa de falhas possível.
- FIFO (First-In, First-Out): substitui a página mais antiga na memória.

IV. Resultados e Análise

Os experimentos foram conduzidos variando o número de quadros de memória disponíveis em 4, 8, 16, 32, 64 e 128, aplicados sobre as três aplicações selecionadas. Para cada configuração, foram avaliados ambos algoritmos, FIFO e Ótimo, registrando-se o número total de falhas de página observadas.

D. Resultados Obtidos

As Tabelas I e II apresentam os resultados referentes ao programa Hello World, cuja *reference string* conta com 96149 entradas.

TABLE I. Número de falhas Hello World

Número de quadros	**Modelos de execução**	
	Ótimo	***FIFO***
	Número de falhas	
4	15997	24643
8	9920	16480
16	6339	11325
32	3456	7121
64	1568	3478
128	1189	1996

TABLE II. Porcentagem de falhas Hello World

Número de quadros	**Modelos de execução**	
	Ótimo	***FIFO***
	Porcentagem de falhas (%)	
4	16,64	25,63
8	10,32	17,14
16	6,59	11,78
32	3,59	7,41
64	1,63	3,61
128	1,24	2,08

As Tabela III e IV apresentam os resultados para a aplicação do algoritmo de Dijkstra, cuja *reference string* conta com 1429517 entradas.

TABLE III. Número de Falhas Dijkstra

Número de quadros	**Modelos de execução**	
	Ótimo	***FIFO***
	Número de falhas	
4	227572	350069
8	140828	226425
16	89747	154981
32	46937	109659
64	14269	42432
128	8577	19781

TABLE IV. Porcentagem de Falhas Dijkstra

Número de quadros	**Modelos de execução**	
	Ótimo	***FIFO***
	Porcentagem de falhas (%)	
4	15,92	24,49
8	9,85	15,84
16	6,28	10,84
32	3,28	7,67
64	0,99	2,97
128	0,59	1,38

As Tabelas V e VI apresentam os resultados da simulação bancária sequencial, cuja *reference string* conta com 4336557 entradas:

TABLE V. Número de Falhas Simulação Bancária

Número de quadros	**Modelos de execução**	
	Ótimo	***FIFO***
	Número de falhas	
4	774074	1737951
8	10365	16664
16	6797	11871
32	3751	7650
64	1721	3793
128	1306	2185

TABLE VI. Porcentagem de Falhas Simulação Bancária

Número de quadros	**Modelos de execução**	
	Ótimo	***FIFO***
	Número de falhas	
4	17,85	40,08
8	0,24	0,38
16	0,16	0,27
32	0,09	0,18
64	0,04	0,09
128	0,03	0,05

E. Análise dos Resultados

Em todas as aplicações, observa-se que quanto maior o número de quadros disponíveis, menor o número de falhas de página. Isso está de acordo com a teoria da memória virtual: aumentar a quantidade de quadros reduz a necessidade de substituições, já que mais páginas podem permanecer residentes em memória[1].

A diferença de desempenho entre Ótimo e FIFO é mais acentuada nos cenários com poucos quadros. À medida que o número de quadros aumenta, os dois algoritmos tendem a convergir, pois há menos substituições necessárias.

Na aplicação Hello World temos um programa simples com poucas referências às páginas,. Para 4 quadros, a diferença é marcante: 15997 (16,64% de falhas) no algoritmo Ótimo vs 24643 (25,63% de falhas) no FIFO. À medida que os quadros aumentam, as falhas reduzem bastante: em 128 quadros, caem para apenas 1,24% (Ótimo) vs 2,08% (FIFO). Isso mostra que o Hello World é pouco exigente em memória, convergindo rapidamente para números baixos de falhas.

No Algoritmo de Dijkstra temos uma aplicação bem mais intensiva em memória, como descrito anteriormente no artigo. Com 4 quadros, temos 227572 (Ótimo) vs 350069 (FIFO), valores enormes comparados ao Hello World, mas com porcentagem de falhas próximas às obtidas anteriormente: 15,92% (Ótimo) vs 24,49% (FIFO). Mesmo com 128 quadros, ainda há 8577 (Ótimo) vs 19781 (FIFO) falhas, ou seja, o ganho de memória ajuda, mas o programa continua exigente. A discrepância entre Ótimo e FIFO é evidente em todos os casos.

Na Simulação Bancária Sequencial para 4 quadros, temos 774074 (Ótimo) vs 1.737.951 (FIFO), mais que o dobro de falhas no FIFO, com mais de 40% das referências na memória gerando substituições. Com 128 quadros, os valores caem bastante: 1306 (Ótimo) vs 2185 (FIFO), que equivalem a uma porcentagem de falhas de 0,03% vs 0.05%. Aqui se observa que o FIFO é especialmente ineficiente em cenários críticos (pouca memória), mas se aproxima do Ótimo quando a memória cresce.

Ela sofre muito quando há pouquíssima memória (4 quadros), porque seu padrão de acesso não se adapta bem a uma janela tão pequena gerando um thrashing extremo. Mas, quando a memória é suficiente para cobrir sua janela de trabalho (~128 quadros), o número de falhas despenca. Diferente do Dijkstra que sofre relativamente menos no início (com 4 quadros), porque consegue explorar algum nível de localidade temporal. Mas sua demanda de memória continua alta mesmo com 128 quadros, impedindo que o número de falhas caia tanto quanto no caso bancário. Isso explica a diferença entre os dois quanto ao número de quadros

V. Conclusão

A análise realizada evidenciou a importância da política de substituição de páginas no desempenho dos sistemas de memória virtual. Os experimentos mostraram que, independentemente da aplicação analisada, o aumento do número de quadros disponíveis reduz significativamente a ocorrência de falhas de página, confirmando o princípio teórico da paginação por demanda.

Os resultados também destacaram que o algoritmo Ótimo, por definição, apresentou sempre o menor número de falhas, servindo como limite inferior para a avaliação do desempenho. O FIFO, embora simples de implementar, mostrou-se mais suscetível a falhas adicionais, especialmente em cenários com memória restrita, evidenciando suas limitações práticas.

Outro ponto relevante é que o comportamento das falhas de página variou de acordo com o perfil das aplicações. O programa *Hello World*, por ser simples e de baixa demanda, rapidamente convergiu para valores reduzidos de falhas conforme os quadros aumentaram. O algoritmo de Dijkstra, em contraste, manteve taxas de falhas elevadas mesmo com maior disponibilidade de memória, refletindo sua alta intensidade de acesso. Já a simulação bancária sequencial apresentou um padrão distinto: sofreu intensamente com pouquíssimos quadros (thrashing), mas com memória suficiente conseguiu reduzir drasticamente o número de falhas, superando inclusive o desempenho do Dijkstra em configurações mais amplas.

Portanto, conclui-se que a escolha do algoritmo de substituição de páginas exerce impacto direto na eficiência do sistema, sendo ainda mais determinante em cenários de limitação de recursos. Além disso, a natureza da aplicação desempenha papel central no padrão de falhas, reforçando que soluções de gerenciamento de memória devem considerar tanto a política de substituição quanto o perfil de uso da aplicação para otimizar o desempenho global do sistema.

Referências

[1] SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGNE, Greg. Fundamentos de Sistemas Operacionais. 9. ed. Rio de Janeiro: LTC, 2013.

[2] https://valgrind.org/