

Análise Comparativa dos Algoritmos de Substituição de Página FIFO, LRU e RANDOM em um Simulador de Memória Virtual e Física

Marcia Gabrielle B. Oliveira¹, Shelly da C. Leal¹, Wandressa da S. Reis¹

¹Departamento de Ciência da Computação – Universidade Federal de Roraima (UFRR)
Boa Vista - RR - Brasil

{marciagaby.oliveira, contatoshellyleal05, reiswandressa}@gmail.com

Abstract. This article presents a comparative analysis of the page replacement algorithms LRU (Least Recently Used), FIFO (First-In, First-Out), and RANDOM (random) in a virtual and physical memory simulator. Experiments were conducted on different files containing memory addresses from a real program. The execution results of the program include the number of pages read and written for each page replacement algorithm (FIFO, LRU, and RANDOM). The obtained results provide important information for the appropriate choice of the page replacement algorithm.

Resumo. Este artigo apresenta uma análise comparativa dos algoritmos de substituição de página LRU (Least Recently Used), FIFO (First-In, First-Out) e RANDOM (aleatório) em um simulador de memória virtual e física. Foram realizados experimentos em diferentes arquivos contendo endereços de memória de um programa real. Os resultados da execução do programa incluem a quantidade de páginas lidas e de páginas escritas em cada algoritmo de técnica de reposição de páginas (FIFO, LRU e RANDOM). Os resultados obtidos fornecem informações importantes para a escolha adequada do algoritmo de substituição de página.

1. Introdução

A gestão eficiente da memória é um aspecto crítico em sistemas operacionais e arquiteturas de computadores modernos. Com o crescimento contínuo das demandas por recursos de memória, torna-se essencial implementar estratégias eficazes para gerenciar a memória física e a memória virtual. Nesse contexto, os algoritmos de substituição de página desempenham um papel fundamental, pois são responsáveis por decidir quais páginas devem ser mantidas na memória física e quais devem ser transferidas para a memória virtual.

O objetivo deste trabalho é realizar uma análise comparativa de diferentes algoritmos de substituição de página em um simulador de memória virtual e física.

Os algoritmos de substituição de página são projetados para otimizar o desempenho do sistema, minimizando a taxa de falhas de página e maximizando a utilização da memória física disponível. Compreender como esses algoritmos funcionam e como eles se comportam em diferentes cenários pode auxiliar na escolha da estratégia mais adequada para um determinado ambiente de aplicação.

Nesta análise comparativa, serão considerados três algoritmos de substituição de página, o FIFO (First-In, First-Out), LRU (Least Recently Used), Random (Aleatório). Cada algoritmo tem suas próprias características e critérios para decidir quais páginas devem ser substituídas, e é importante examinar seu desempenho em termos de eficiência e impacto na taxa de falhas de página.

Ao simular os algoritmos, poderemos analisar as métricas de desempenho, como o número de páginas lidas e número de páginas escritas. Essas métricas nos permitirão identificar a capacidade de leitura e escrita de cada um dos três algoritmos, determinados os tamanhos razoáveis de página/quadro de memória e o tamanho da memória física disponível dentro de um escopo razoável.

No decorrer deste trabalho, serão apresentados os princípios básicos de cada algoritmo de substituição de página, suas vantagens e limitações, além dos resultados obtidos por meio da simulação. Com base nessa análise comparativa, será possível fazer recomendações sobre quais algoritmos são mais adequados para determinados cenários e aplicações específicas.

Em suma, a análise comparativa de algoritmos de substituição de página em um simulador de memória virtual e física tem como objetivo aprofundar o entendimento sobre o desempenho e eficiência desses algoritmos, fornecendo informações valiosas para o desenvolvimento de estratégias de gerenciamento de memória mais eficazes e otimizadas.

2. Fundamentação Teórica

2.1. Gerência de Memória

Nos primeiros sistemas, somente um processo poderia ser executado por vez e ele permanecia na memória até terminar a sua execução. Uma parte da memória era dedicada ao sistema operacional e o restante era todo disponível para o processo que estivesse em execução. O programador era totalmente responsável por gerenciar a alocação da memória utilizada pelos processos. Se um processo fosse grande demais

para ser carregado, a memória principal deveria ser aumentada ou então o programa deveria ser modificado de maneira que ele coubesse na memória [Deitel et al. 2005, Flynn and McHoes 2002, de Oliveira et al. 2010, Tanenbaum 2010].

Para resolver a restrição de permitir apenas a execução de um processo por vez, foram desenvolvidas estratégias, como a abordagem de partição fixa, que divide a memória logicamente em várias partes de tamanhos iguais ou variados. Essa abordagem apresenta desafios, como proteger o espaço de memória de um processo, lidar com a fragmentação interna e alocar processos em partições dinâmicas [Deitel et al., 2005; Flynn e McHoes, 2002; de Oliveira et al., 2010].

Independentemente de o esquema adotado ser de partições fixas ou dinâmicas, o sistema operacional precisa manter uma lista que registra quais partições estão desocupadas. Quando um novo processo precisa ser carregado, o gerenciador deve encontrar uma partição livre na qual possa alocar o processo. Existem vários algoritmos para percorrer a lista e encontrar uma partição com tamanho adequado para o processo [Flynn e McHoes, 2002].

A capacidade da memória sozinha não é suficiente para suportar todos os programas. Se todas as partições de memória estiverem ocupadas e um processo precisa ser executado, ele só poderá ser executado quando alguma partição for liberada e, além disso, tiver tamanho suficiente para acomodar o novo processo. Para resolver esse problema, foi criada a técnica de swapping, que realiza a transferência temporária do processo entre o disco e a memória [Stuart, 2011; Deitel et al., 2005; de Oliveira et al., 2010; Sílberschatz et al., 2004; Tanenbaum, 2010]. Essa técnica é fundamental para a implementação da Memória Virtual.

2.2. Conceitos Básicos de Memória Virtual e Física

Para compreender a análise comparativa de algoritmos de substituição de página, é importante ter um entendimento dos conceitos básicos de memória virtual e física. A memória virtual é uma extensão da memória física disponível em um sistema computacional. Ela permite que um programa acesse uma quantidade maior de memória do que está fisicamente presente no hardware. A memória virtual é dividida em unidades chamadas de páginas, que são blocos de tamanho fixo.

A memória física, por sua vez, refere-se à memória real do sistema, que é composta pelos chips de memória instalados no computador. Ela armazena as páginas que estão atualmente sendo utilizadas pelos processos em execução.

2.3. Paginação

A técnica de paginação permite que um processo seja alocado em espaços de memória não contíguos. Não é mais necessário ter partições de tamanhos diversos, pois as páginas têm um tamanho fixo. Cada página é armazenada em um bloco da memória principal chamado de moldura de página, que possui exatamente o tamanho de uma página. Quando um processo está pronto para ser carregado, as páginas necessárias são carregadas em qualquer espaço disponível na memória. Nesse esquema, não há fragmentação externa, mas pode ocorrer fragmentação interna, embora em menor escala [Sílberschatz et al., 2004].

Para determinar se uma determinada página está carregada na memória principal ou no disco, é utilizado um bit válido-inválido. O gerenciador de memória mantém uma tabela de páginas que mapeia todas as páginas do processo e indica se cada página está carregada na memória principal ou não.

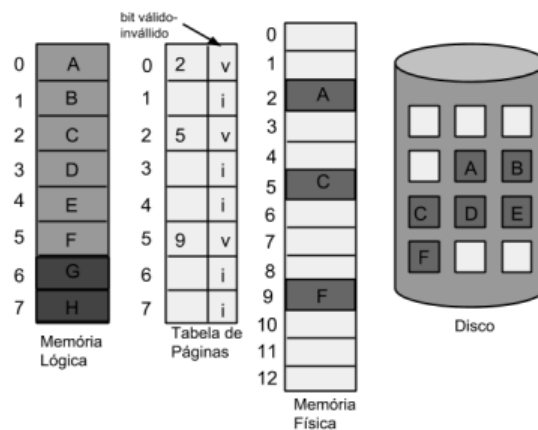


Figura 1. Tabela de página quando algumas páginas não estão na memória principal [Sílborschatz et al. 2004]

(A Figura 1) apresenta a memória física com seus endereços (variando de 0 à 12) e a memória virtual, modelada no disco e a Tabela de Páginas, que apresenta a correlação dos endereços e a coluna que apresenta o bit de válido / inválido (“i” e “v”, respectivamente) [Sílborschatz et al. 2004].

2.4. Troca de Páginas

Quando um processo faz referência a uma página e todas as molduras de página na memória principal estão ocupadas, ocorre o que é chamado de page-fault ou falta de página [Sílborschatz et al., 2004]. A memória principal não é utilizada apenas para carregar dados e instruções de programas, mas também para outros fins, como buffers de operações de entrada/saída, que ocupam uma quantidade considerável de memória.

Quando ocorre uma falta de página, o gerenciador de memória precisa transferir uma página carregada na memória para o disco e, em seguida, carregar a página referenciada. Esse processo é conhecido como substituição de páginas. A seguir, descreveremos como esse processo funciona [Sílborschatz et al., 2004; Flynn e McHoes, 2002]:

1. Localizar a posição da página desejada no disco.
2. Encontrar uma moldura de página livre. (a) Se houver uma moldura livre, utilizá-la. (b) Se não houver molduras livres, utilizar um algoritmo para selecionar uma moldura. (c) Gravar a página vítima no disco; alterar as tabelas de página e moldura.
3. Ler a página vítima do disco; atualizar as tabelas de páginas e moldura conforme necessário.
4. Retornar ao processo do usuário.

Existem diversas estratégias de substituição de páginas. Para análise, consideramos os algoritmos LRU (Least Recently Used), FIFO (First-In, First-Out) e RANDOM (aleatório).

2.5. Descrições dos Algoritmos de Substituição de Página: FIFO, LRU e RANDOM

Nesta seção, serão apresentadas as descrições dos algoritmos de substituição de página que serão comparados na análise.

2.5.1. FIFO (First In, First Out)

O algoritmo FIFO substitui a página mais antiga, ou seja, a página que foi carregada primeiro na memória física. Ele mantém uma fila de páginas e remove a página que está na frente da fila quando é necessário realizar uma substituição. [Tanenbaum, 2010]

Vantagens: A implementação do algoritmo FIFO é simples e não requer estruturas de dados complexas adicionais e é fácil de entender e implementar.

Desvantagens: O algoritmo FIFO não leva em consideração padrões de acesso à memória. Mesmo que uma página tenha sido recentemente acessada e seja muito provável que seja acessada novamente em breve, ela ainda será substituída se estiver no início da fila. Isso pode resultar em um desempenho pior em comparação com o LRU em muitos cenários de uso.

2.5.2. LRU (Least Recently Used)

O algoritmo LRU é baseado no princípio de que as páginas que não foram acessadas há mais tempo têm uma menor probabilidade de serem acessadas novamente no futuro próximo. Portanto, o LRU substitui a página que não foi usada por mais tempo. Para implementar esse algoritmo, é necessário manter um registro do histórico de acesso a cada página para determinar qual foi a menos recentemente utilizada.

Vantagens: O algoritmo LRU tem o potencial de apresentar um bom desempenho em muitos casos, especialmente em padrões de acesso em que há uma alta taxa de localidade temporal, ou seja, as páginas que foram recentemente acessadas têm maior probabilidade de serem acessadas novamente no futuro próximo. Ele também minimiza o número de faltas de página quando comparado ao FIFO e RANDOM.

Desvantagens: A implementação eficiente do LRU requer uma estrutura de dados adicional, como uma lista ligada ou uma fila de prioridade, para manter o rastreamento das páginas usadas recentemente. Isso pode levar a um aumento na sobrecarga de memória e complexidade de implementação.

2.5.3. RANDOM (Aleatório)

O algoritmo de substituição de página aleatória, como o próprio nome sugere, escolhe aleatoriamente uma página para substituir. Não leva em consideração o histórico de acesso ou a frequência de uso das páginas. A escolha é completamente aleatória, o que pode levar a um desempenho imprevisível e variável.

Vantagens: O algoritmo RANDOM é simples de implementar. E em algumas situações específicas, ele pode ter um desempenho semelhante ao LRU ou até mesmo superá-lo.

Desvantagens: Devido à sua natureza aleatória, o algoritmo RANDOM pode ter um desempenho inconsistente. Em alguns casos, pode escolher uma página que foi acessada recentemente ou uma página que é frequentemente acessada, resultando em um maior número de faltas de página.

2.6 Métricas utilizadas na avaliação de desempenho

Ao realizar a análise comparativa dos algoritmos de substituição de página, serão utilizadas algumas métricas para avaliar o desempenho de cada algoritmo. As métricas utilizadas incluem:

1. Número de leituras de página:

Essa métrica mede o total de leituras de página que ocorrem durante a execução do simulador de memória. Uma leitura de página refere-se à recuperação de uma página da memória física ou virtual.

2. Número de escritas de página:

O número de escritas de página representa a contagem total de páginas que são escritas de volta no subsistema de memória, seja na memória física ou virtual, durante a simulação.

Ao final da execução, um relatório será gerado, resumindo as estatísticas coletadas. Esse relatório fornecerá insights sobre o comportamento e a eficiência de cada um dos três algoritmos de substituição de página, auxiliando na análise comparativa.

Nas seções seguintes, iremos descrever a metodologia utilizada para coletar essas métricas de desempenho e discutir sua importância na avaliação da eficácia dos algoritmos de substituição de página.

3. Metodologia

3.1. Descrição do simulador de memória virtual e física

A metodologia utilizada neste estudo envolve a implementação de um simulador de memória virtual e física. O simulador é desenvolvido em linguagem de programação C e possui a finalidade de analisar o desempenho de diferentes algoritmos de substituição de páginas em sistemas de memória.

O simulador é composto por um conjunto de funções e estruturas de dados para realizar as operações de leitura e escrita em memória, além de gerenciar as páginas armazenadas na memória física. O código foi dividido em várias funções para facilitar o entendimento e a manutenção do simulador.

Inicialmente, o simulador recebe os parâmetros necessários através da linha de comando, incluindo o algoritmo de substituição de páginas a ser utilizado (LRU, FIFO ou aleatório), o caminho do arquivo de entrada contendo as operações a serem executadas, o tamanho da página e o tamanho da memória física.

O arquivo de entrada contém uma sequência de operações de leitura (R) e escrita (W), juntamente com os endereços de memória correspondentes. O simulador lê o arquivo linha por linha e executa as operações especificadas. Para cada operação de leitura, o simulador verifica se o endereço já está presente na memória física. Se estiver presente, ocorre um "acerto" (hit); caso contrário, ocorre um "erro de página" (miss). Nas operações de escrita, o simulador atualiza o conteúdo da página correspondente.

O simulador utiliza uma lista encadeada para representar as páginas armazenadas na memória física. Cada página é representada por uma estrutura que contém um endereço e um ponteiro para a próxima página. As funções de substituição de páginas são implementadas de acordo com o algoritmo especificado. Para o algoritmo LRU, a página menos recentemente utilizada é substituída. Para o algoritmo FIFO, a página mais antiga é substituída. No algoritmo RANDOM, uma página aleatória é escolhida para ser substituída.

3.2 Coleta de dados

Para a coleta de dados utilizamos um arquivo de log contendo os endereços de memória (leitura e escrita) de um programa real. Os endereços são lidos e é verificado se eles estão presentes na memória física. Caso não estejam, é considerado um page fault, e o endereço requisitado é buscado no disco de memória e é alocado para a memória física através do método de algum dos algoritmos de substituição de página. Claro que como se trata de uma simulação, a memória não é devidamente alocada a nível de hardware.

Ao final da execução do simulador, são apresentados os resultados obtidos, incluindo o número de páginas lidas, o número de páginas escritas, o número de acertos, o número de erros de página e outras informações relevantes para a análise do desempenho dos algoritmos de substituição de páginas.

4. Resultados e Discussão

4.1. Apresentação e análise dos resultados obtidos para cada algoritmo

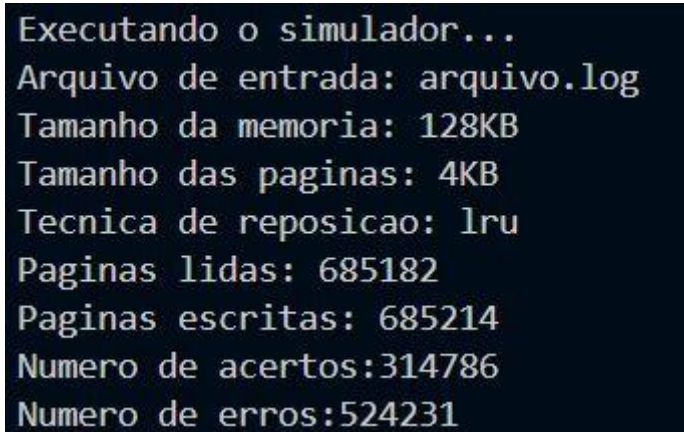
FIFO (First-In-First-Out):

```
Executando o simulador...
Arquivo de entrada: arquivo.log
Tamanho da memoria: 128KB
Tamanho das paginas: 4KB
Tecnica de reposicao: fifo
Paginas lidas: 689877
Paginas escritas: 689909
Numero de acertos:310091
Numero de erros:528926
```

Figura 1. Resultado do simulador após a execução utilizando o algoritmo de substituição de página FIFO.

O algoritmo FIFO (First-In-First-Out) substitui a página mais antiga sempre que ocorre um page fault. Nesse caso, observamos que o número de erros (528.926) é bastante elevado, o que indica que muitas páginas tiveram que ser substituídas durante a execução do simulador. Isso sugere que o algoritmo FIFO não conseguiu manter na memória as páginas mais relevantes e frequentemente acessadas. O número de acertos (310.091) é menor em comparação com os erros, o que reforça a baixa eficiência deste algoritmo.

LRU (Least Recently Used)

A screenshot of a terminal window with a dark background and light-colored text. The text displays the results of a simulation using the LRU algorithm. The output is as follows:

```
Executando o simulador...
Arquivo de entrada: arquivo.log
Tamanho da memoria: 128KB
Tamanho das paginas: 4KB
Tecnica de reposicao: lru
Paginas lidas: 685182
Paginas escritas: 685214
Numero de acertos:314786
Numero de erros:524231
```

Figura 2. Resultado do simulador após a execução utilizando o algoritmo de substituição de página LRU.

O algoritmo LRU (Least Recently Used) substitui a página que não foi acessada por um período mais longo de tempo quando ocorre um page fault. Observamos que o número de erros (524.231) é menor em comparação com o FIFO, indicando uma melhoria no desempenho. O algoritmo LRU conseguiu manter mais páginas relevantes na memória, resultando em um número de acertos (314.786) relativamente maior. Isso sugere que o LRU é capaz de aproveitar melhor as informações de acesso passadas e tomar decisões mais eficientes de substituição de páginas.

RANDOM (Aleatório)


```
Executando o simulador...
Arquivo de entrada: arquivo.log
Tamanho da memoria: 128KB
Tamanho das paginas: 4KB
Tecnica de reposicao: random
Paginas lidas: 881891
Paginas escritas: 881923
Numero de acertos:118077
Numero de erros:720940
```

Figura 3. Resultado do simulador após a execução utilizando o algoritmo de substituição de página RANDOM.

O algoritmo Random (Aleatório) seleciona aleatoriamente uma página para substituição quando ocorre um page fault. Observamos que o número de erros (720.940) é significativamente maior em comparação com os outros dois algoritmos, indicando um desempenho menos eficiente. O algoritmo Random não leva em consideração a frequência ou o histórico de acesso às páginas, o que pode resultar em uma escolha inadequada de páginas para substituição. Como resultado, o número de acertos (118.077) é consideravelmente menor, sugerindo que o Random não consegue manter na memória as páginas mais relevantes de forma consistente.

5. Trabalhos Futuros

O simulador proposto serve como um ponto de partida para desenvolver uma ferramenta que simula o funcionamento de um sistema operacional completo. Compreender completamente o funcionamento de um sistema operacional demanda muito tempo de estudo, devido à sua complexidade e às diversas formas de implementação existentes. Portanto, o escopo deste trabalho se limita a simular apenas uma pequena parte de um sistema operacional, deixando espaço para futuros projetos que possam se basear no que já foi realizado. Algumas sugestões para esses projetos futuros incluem: (a) simulação da tradução dinâmica de endereço virtual para endereço físico, (b) simulação da gerência de processos.

6. Conclusão

Após a análise dos resultados obtidos nos testes utilizando os algoritmos de substituição FIFO, LRU e Random, pode-se concluir que o desempenho dos algoritmos varia significativamente em termos de eficiência na utilização da memória virtual.

O algoritmo FIFO mostrou-se menos eficiente, apresentando um alto número de erros (page faults) e um número relativamente baixo de acertos. Isso sugere que o algoritmo FIFO não é capaz de priorizar adequadamente as páginas mais relevantes e frequentemente acessadas, resultando em uma taxa maior de substituição de páginas, o que impacta negativamente no desempenho geral do sistema.

Por outro lado, o algoritmo LRU demonstrou um desempenho superior em relação ao FIFO. Com um número menor de erros e um número relativamente maior de

acertos, o LRU mostra-se mais eficiente em manter as páginas relevantes na memória, levando em consideração o histórico de acesso. Essa capacidade de tomar decisões de substituição com base no acesso mais recente contribui para uma melhor utilização da memória e, consequentemente, para uma redução na ocorrência de page faults.

Já o algoritmo Random apresentou o pior desempenho entre os três. Com um número elevado de erros e um número de acertos significativamente menor em relação aos demais algoritmos, o Random revela-se ineficiente em manter as páginas relevantes na memória. A escolha aleatória de páginas para substituição não leva em consideração o histórico de acesso, resultando em uma alocação de memória menos otimizada e em um aumento na ocorrência de page faults.

Diante desses resultados, concluímos que o algoritmo LRU é a opção mais recomendada para otimizar o desempenho do sistema de gerenciamento de memória. Sua capacidade de tomar decisões com base no acesso mais recente permite uma melhor utilização da memória virtual, reduzindo o número de page faults e melhorando a eficiência geral do sistema.

No entanto, é importante ressaltar que a escolha do algoritmo de substituição de páginas ideal depende das características específicas do sistema, do padrão de acesso às páginas e do tamanho da memória disponível. É necessário considerar esses aspectos e realizar testes mais abrangentes para determinar a melhor estratégia de gerenciamento de memória em um contexto específico.

7. Referências

- Deitel, H. M., Deitel, P. J., & Choffnes, D. R. (2005). *Operating Systems*. Pearson Education.
- Flynn, C., & McHoes, J. (2002). *Understanding Operating Systems*. Cengage Learning.
- Oliveira, R. B., Silva, A. C., & Valente, J. L. (2010). *Sistemas Operacionais: Uma Abordagem Prática*. LTC.
- Sílbberschatz, A., Galvin, P. B., & Gagne, G. (2004). *Operating System Concepts*. John Wiley & Sons.
- Stuart, B. R. (2011). *Principles of Operating Systems: Design and Applications*. Cengage Learning.
- Tanenbaum, A. S. (2010). *Modern Operating Systems*. Pearson Education.