# Unidade 3 - Análise Comparativa de Algoritmos de Substituição de Páginas em um Simulador de Memória Virtual

**Emerson da S. Santos[1], João Pedro dos S. Medeiros[1], João Batista B. Neto[2]**

[1]Bacharelado em Sistemas de Informação(BSI)
Universidade Federal do Rio Grande do Norte (UFRN) – Campus CERES
Rua Joaquim Gregório, 296, Penedo, 59.300-000 – Caicó – RN – Brasil

[2]Departamento de Computação e Tecnologia (DCT)
Universidade Federal do Rio Grande do Norte (UFRN) – Campus CERES
Rua Joaquim Gregório, 296, Penedo, 59.300-000 – Caicó – RN – Brasil

emerson.santos.125@ufrn.edu.br, jopesame@gmail.com, joao.borges@ufrn.br

***Abstract.*** *This paper presents the implementation and a comparative performance analysis of six page replacement algorithms within a virtual memory simulator. The work was based on an existing simulator, extending it to support First-In, First-Out (FIFO), Least Recently Used (LRU), Second Chance (SC), Least Frequently Used (LFU), and Most Frequently Used (MFU), in addition to the pre-existing Random algorithm. The performance analysis was conducted through automated simulations using four different memory access traces and varying the number of available memory frames from 2 to 64. The primary metric for evaluation was the total number of Page Faults. The results are presented in line graphs, allowing for a clear visual assessment of the performance and behavior of each algorithm under different workloads.*

***Resumo.*** *Este trabalho apresenta a implementação e a análise de performance comparativa de seis algoritmos de substituição de páginas em um simulador de memória virtual. O trabalho foi baseado em um simulador existente, estendendo-o para suportar os algoritmos First-In, First-Out (FIFO), Least Recently Used (LRU), Second Chance (SC), Least Frequently Used (LFU) e Most Frequently Used (MFU), além do algoritmo Aleatório (Random) pré-existente. A análise de desempenho foi realizada por meio de simulações automatizadas, utilizando quatro diferentes traces de acesso à memória e variando a quantidade de frames de memória disponíveis de 2 a 64. A métrica principal de avaliação foi o número total de Faltas de Página (Page Faults). Os resultados são apresentados em gráficos de linha, permitindo a análise visual do desempenho e do comportamento de cada algoritmo sob diferentes cargas de trabalho.*

## 1. Introdução

O Sistema Operacional (SO) atua como um gerenciador de recursos complexo, sendo uma de suas responsabilidades mais críticas o *gerenciamento de memória*. Em sistemas modernos, a memória virtual é uma técnica de abstração que permite a um processo utilizar um espaço de endereçamento muito maior do que a memória física (RAM) realmente disponível. Isso é alcançado mantendo na memória principal apenas as partes do processo que estão em uso ativo, enquanto o restante é armazenado em disco.

Quando um processo tenta aceder uma porção de seu espaço de endereçamento que não está na memória física, ocorre uma interrupção de hardware conhecida como *falta de página (page fault)*. O SO deve então tratar essa falta, buscando a página necessária no disco e carregando-a na memória. Se não houver frames de memória livres, o SO precisa selecionar uma página que está na memória para ser removida (a "página vítima") e dar lugar à nova. A escolha da página vítima é determinada por um *algoritmo de substituição de páginas*. A eficiência deste algoritmo é crucial, pois uma má escolha pode levar a um número excessivo de page faults, um fenômeno conhecido como *thrashing*, que degrada severamente a performance do sistema.

Não existe um único algoritmo de substituição ideal para todas as situações; a escolha mais apropriada depende do padrão de acesso à memória dos programas em execução. Diante disso, a análise comparativa de diferentes estratégias torna-se essencial para compreender seus respectivos pontos fortes e limitações.

Este trabalho tem como objetivo a implementação e análise comparativa de desempenho de cinco algoritmos de substituição de páginas. Utilizando o simulador `virtmem_sim` como base, foram implementados os algoritmos: *First-In, First-Out (FIFO)*, *Least Recently Used (LRU)*, *Second-Chance (SC)*, *Least Frequently Used (LFU)* e *Most Frequently Used (MFU)*. A metodologia de análise envolve a execução automatizada de cada algoritmo sob diferentes cargas de trabalho (traces) e com um número crescente de frames de memória, de 2 a 64. Por fim, os resultados são apresentados em gráficos comparativos que permitem uma discussão aprofundada sobre qual algoritmo se mostrou mais eficiente para cada cenário.

## 2. Metodologia

A implementação dos algoritmos de substituição de páginas foi desenvolvida na linguagem C, utilizando como base a estrutura do simulador `virtmem_sim`. O simulador processa ficheiros de *trace* que contêm sequências de acessos a endereços de memória. Quando uma falta de página ocorre e não há frames de memória livres, uma função de seleção de vítima é chamada. Para cada um dos cinco algoritmos, uma função `selectVictim<NomeDoAlgoritmo>()` foi implementada.

### 2.1. Algoritmo FIFO (First-In, First-Out)

A implementação do FIFO visa selecionar a página que está na memória há mais tempo. A lógica de seleção foi implementada iterando pela tabela de frames (`frameTable`) para encontrar a entrada que possui o menor valor no campo `load_time`, que armazena o "instante"(passo da simulação) em que a página foi carregada.

### 2.2. Algoritmo LRU (Least Recently Used)

O LRU seleciona como vítima a página que não é acedida há mais tempo. De forma análoga ao FIFO, sua implementação percorre a tabela de frames para encontrar a entrada com o menor valor no campo `last_reference_time`, que é atualizado a cada acesso à página.

### 2.3. Algoritmo SC (Second Chance)

Este algoritmo funciona como uma melhoria do FIFO. Ele utiliza um ponteiro circular (`sc_victim_ptr`) que percorre os frames. Ao inspecionar um frame, ele verifica o *bit*

*de referência* (`referenced`). Se o bit for 0, a página é selecionada como vítima. Se for 1, o bit é zerado (dando-lhe uma ”segunda chance”) e o ponteiro avança para o próximo frame. Este processo continua até que uma vítima seja encontrada.

### 2.4. Algoritmo LFU (Least Frequently Used)

O LFU seleciona a página que foi acedida o menor número de vezes. A implementação itera pela tabela de frames e identifica a entrada com o menor valor no campo `reference_num`, um contador que é incrementado a cada acesso à página.

### 2.5. Algoritmo MFU (Most Frequently Used)

Implementado como o inverso conceitual do LFU, o MFU seleciona a página que foi acedida o maior número de vezes. A sua lógica percorre a tabela de frames para encontrar o processo com o valor máximo de `reference_num`. A premissa (geralmente incorreta) é que a página mais usada provavelmente não será mais necessária em breve.

## 3. Resultados e Discussão

Nesta seção, são apresentados e analisados os resultados obtidos a partir da execução dos algoritmos com os quatro ficheiros de trace fornecidos. A métrica de avaliação é o número total de Faltas de Página (Page Faults) em função do número de frames disponíveis na memória.

### 3.1. Análise do Trace 1

O comportamento dos algoritmos para o `trace1.trace` é apresentado na Figura 1. Este trace demonstra um caso em que o ”working set”(conjunto de páginas ativas) do programa é relativamente pequeno. Com poucos frames (menos de 16), o número de page faults é alto, mas à medida que o número de frames aumenta, a maioria dos algoritmos (LRU, LFU, SC, FIFO, Random) converge rapidamente para um número muito baixo de falhas. Isto indica que, com memória suficiente, quase todas as páginas necessárias cabem na RAM. O algoritmo MFU, no entanto, exibe um comportamento anómalo e ineficiente, mantendo um número elevado de falhas, evidenciando sua inadequação para este padrão de acesso.

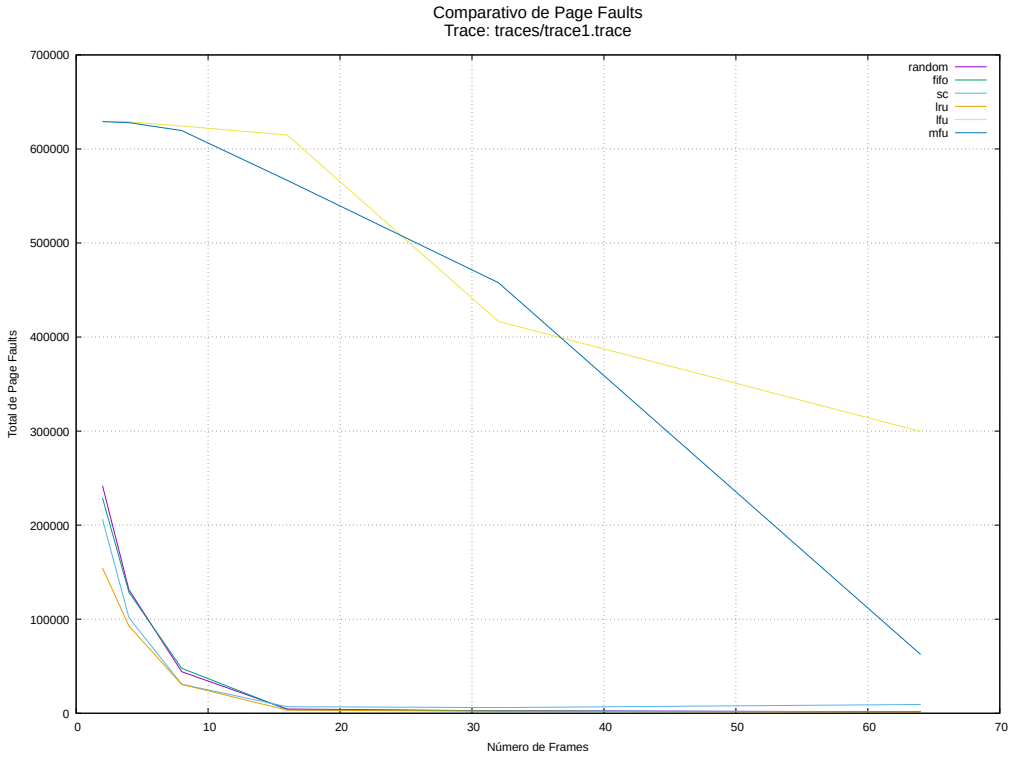


**Figura 1. Desempenho dos algoritmos para o trace1.trace.**

## 3.2. Análise do Trace 2

A Figura 2 ilustra o desempenho para o `trace2.trace`. Neste cenário, o número geral de page faults é maior, sugerindo um ”working set”mais amplo ou um padrão de acesso mais disperso. Fica clara a hierarquia de desempenho: LRU e LFU são consistentemente os mais eficientes, com o menor número de falhas. FIFO, SC e Random formam um grupo intermediário, com o SC mostrando uma ligeira vantagem sobre o FIFO. Novamente, o MFU apresenta o pior desempenho, com um número de falhas drasticamente superior aos demais.

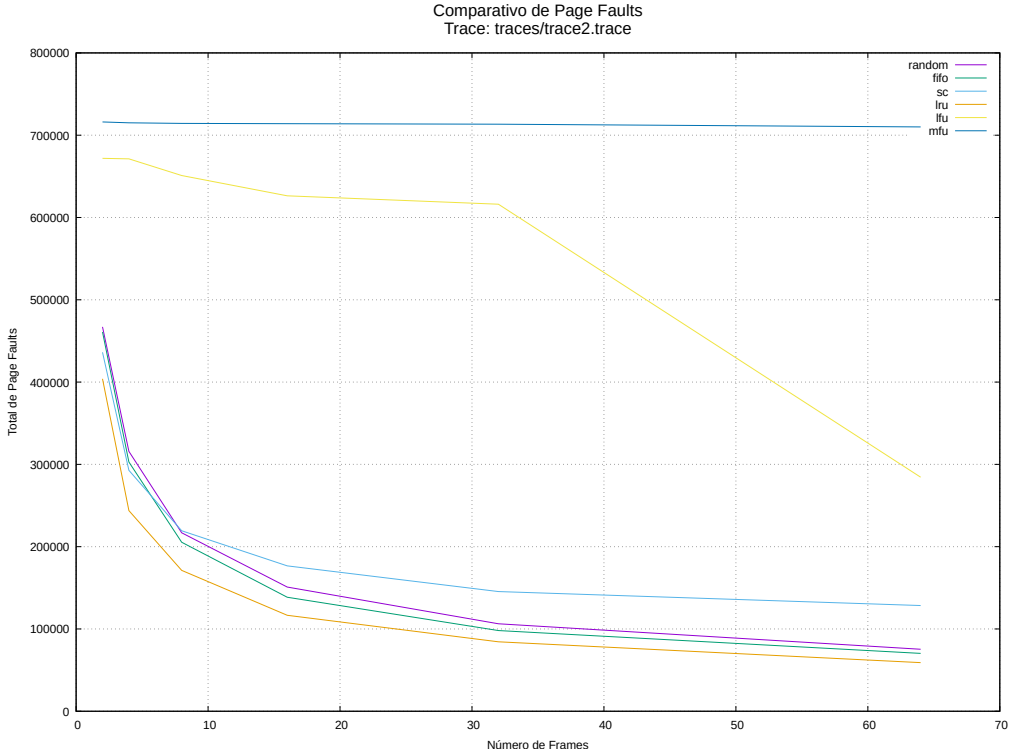


**Figura 2. Desempenho dos algoritmos para o trace2.trace.**

### 3.3. Análise do Trace 3

O resultado para o `trace3.trace`, exibido na Figura 3, reforça as observações do trace anterior. A superioridade dos algoritmos LRU e LFU é evidente, pois eles conseguem adaptar-se melhor à localidade de referência do programa, mantendo as páginas mais relevantes na memória. O MFU continua a ser uma escolha contraproducente, removendo precisamente as páginas que têm maior probabilidade de serem reutilizadas. O desempenho do FIFO, SC e Random permanece intermediário.

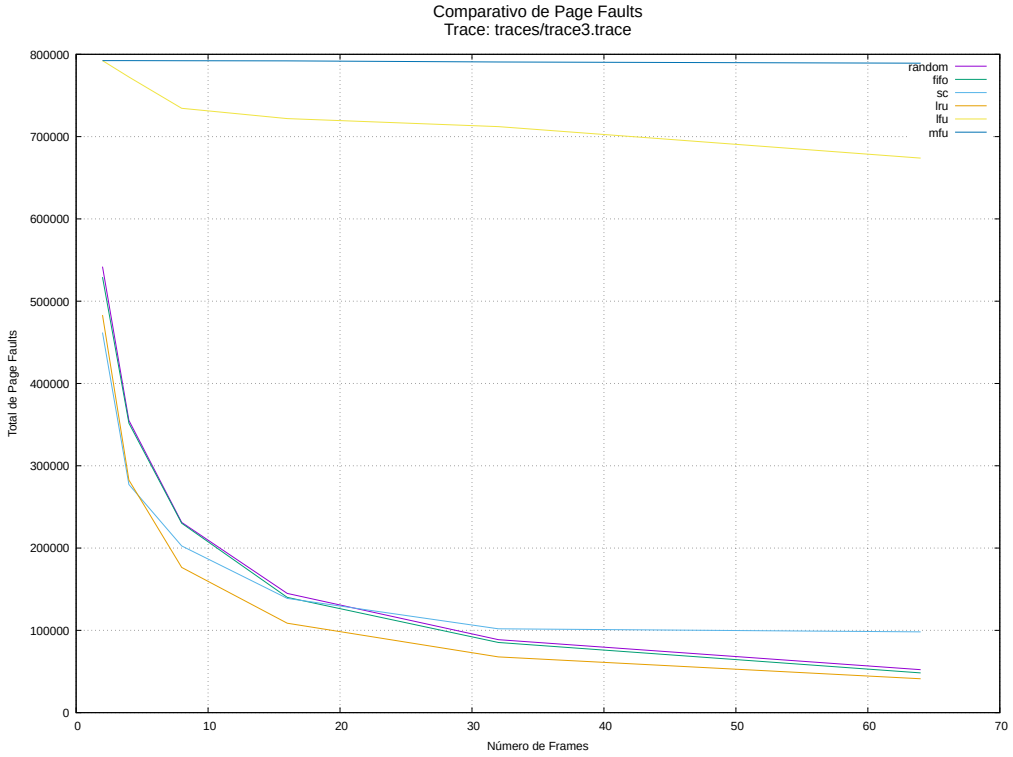


**Figura 3. Desempenho dos algoritmos para o trace3.trace.**

### 3.4. Análise do Trace 4

Finalmente, a Figura 4 apresenta os resultados para o `trace4.trace`. Este gráfico consolida a tendência geral. Todos os algoritmos (exceto o MFU) beneficiam do aumento de frames, mas a eficiência varia significativamente. LRU e LFU demonstram sua robustez, minimizando as faltas de página de forma eficaz. O MFU, por sua lógica falha, mantém um patamar de falhas muito elevado, provando ser um algoritmo inadequado para cenários de uso geral. A análise conjunta dos quatro gráficos demonstra que não há um ”pior”algoritmo entre FIFO, SC e Random, mas sim que sua eficiência relativa depende do padrão de acesso específico de cada trace.

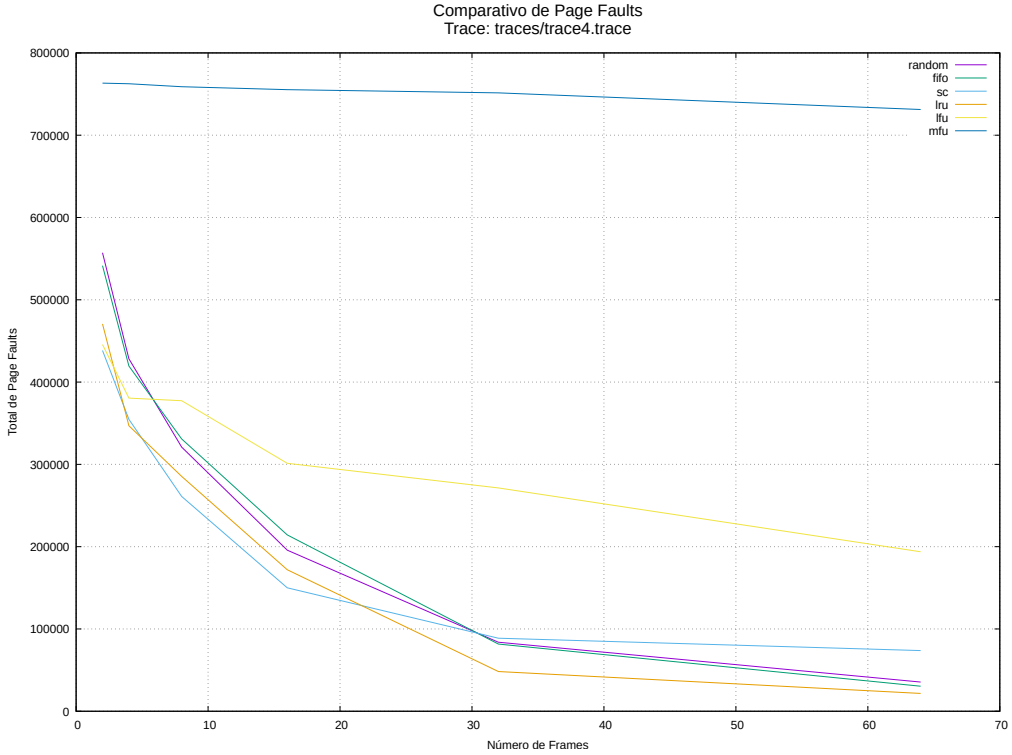


**Figura 4. Desempenho dos algoritmos para o trace4.trace.**

## 4. Conclusão

Este trabalho abordou a implementação e a análise de desempenho de cinco algoritmos de substituição de páginas, adicionando-os a um simulador de memória virtual. A avaliação foi realizada por meio de uma metodologia de simulação automatizada, utilizando o número total de Faltas de Página como métrica principal de performance.

Os resultados obtidos confirmaram de forma prática as expectativas teóricas da literatura de sistemas operativos. Os algoritmos LRU e LFU, que exploram a localidade de referência, demonstraram ser consistentemente os mais eficientes na redução de page faults para todos os traces testados. Em contrapartida, o algoritmo MFU mostrou-se contraproducente, resultando no pior desempenho em todos os cenários, como previsto. Os algoritmos FIFO, SC e Random apresentaram um desempenho intermediário, com o SC a confirmar-se como uma melhoria modesta, mas consistente, sobre o FIFO.

Este projeto cumpriu com sucesso seus objetivos, não apenas ao implementar funcionalmente os algoritmos propostos, mas também ao fornecer uma visualização clara e comparativa dos trade-offs inerentes à gestão de memória virtual. O trabalho serve como uma demonstração prática e valiosa dos conceitos fundamentais que governam a eficiência dos sistemas operativos modernos.

## Referências

Silberschatz, A., Galvin, P. B., and Gagne, G. (2015). *Fundamentos de Sistemas Operacionais*. LTC, Rio de Janeiro, 9ª edition.