UNIVERSIDADE FEDERAL DE RORAIMA
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
CURSO BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

DCC403 - SISTEMAS OPERACIONAIS

ABRAHÃO PICANÇO NERES DE OLIVEIRA

ANDREZA OLIVEIRA GONÇALVES

PROJETO FINAL - SIMULADOR DE MEMÓRIA VIRTUAL E FÍSICA

Boa Vista/RR

2025

ABRAHÃO PICANÇO NERES DE OLIVEIRA

ANDREZA OLIVEIRA GONÇALVES

PROJETO FINAL - SIMULADOR DE MEMÓRIA VIRTUAL E FÍSICA

Trabalho Avaliativo para matéria de Sistemas Operacionais I em Ciências da Computação da Universidade de Roraima - UFRR.

Orientador: Herbert Oliveira Rocha

Boa Vista/RR

2025

SUMÁRIO

1. INTRODUÇÃO

Este projeto tem como objetivo desenvolver um simulador em Linguagem C para modelar o gerenciamento de memória virtual em um sistema operacional, utilizando paginação. O simulador processa um arquivo de log contendo uma sequência de endereços de memória (em formato hexadecimal, com acessos de leitura ou escrita) e simula o comportamento de três algoritmos de substituição de página: FIFO, LRU e Random.

O programa coleta estatísticas, como o número de page faults e páginas sujas gravadas em disco, para avaliar a eficiência de cada algoritmo sob diferentes configurações de tamanho de página (2 a 64 KB) e memória física (128 a 16384 KB). Este relatório apresenta a fundamentação teórica, a implementação do simulador, os resultados dos testes e as conclusões sobre o desempenho dos algoritmos..

2. FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta os conceitos fundamentais de gerenciamento de memória que embasam o desenvolvimento do simulador de memória virtual e física. São abordados os princípios de memória virtual e paginação, o fenômeno da falta de página (page fault), o conceito de páginas sujas e as estratégias dos algoritmos de substituição de página implementados: FIFO, LRU e Random.

2.1. MEMÓRIA VIRTUAL E PAGINAÇÃO

A memória virtual é uma técnica de gerenciamento de memória que permite a execução de processos cujas demandas excedem a memória física (RAM) disponível. Ela cria a ilusão de um espaço de endereçamento lógico contíguo e privado para cada processo, que pode ser muito maior que a memória física.

Na paginação, o espaço de endereçamento lógico do processo é dividido em blocos de tamanho fixo chamados páginas, enquanto a memória física é dividida em quadros de página de mesmo tamanho. O sistema operacional mantém uma tabela de páginas para mapear páginas lógicas aos quadros físicos, que não precisam ser contíguos. Essa abordagem permite carregar apenas partes do processo na

memória, otimizando o uso da RAM e possibilitando a execução de múltiplos processos simultaneamente. No simulador, o tamanho da página (2 a 64 KB) e da memória física (128 a 16384 KB) influencia diretamente o número de quadros disponíveis e o comportamento dos algoritmos de substituição.

2.2. A FALTA DE PÁGINA (PAGE FAULT) E PÁGINAS SUJAS

Uma falta de página (page fault) ocorre quando um processo tenta acessar uma página que não está carregada na memória física. Nesse caso, a Unidade de Gerenciamento de Memória (MMU) gera uma interrupção, e o sistema operacional executa os seguintes passos:

1. Salva o estado do processo.
2. Verifica a validade do acesso (acessos inválidos terminam o processo).
3. Localiza a página no armazenamento secundário (ex.: disco).
4. Seleciona um quadro livre ou utiliza um algoritmo de substituição para liberar um quadro.
5. Carrega a página do disco para o quadro.
6. Atualiza a tabela de páginas.
7. Restaura o processo e reexecuta a instrução.

Uma página é considerada "suja" (dirty) quando foi modificada na memória devido a acessos de escrita (indicados por 'W' no arquivo de log). Se uma página suja for selecionada para substituição, ela deve ser gravada no disco antes de ser removida, aumentando o custo do processo. O simulador contabiliza essas gravações, exceto para páginas sujas remanescentes ao final da execução, conforme especificado.

2.3. ALGORITMOS DE SUBSTITUIÇÃO DE PÁGINA

Quando não há quadros livres na memória física, um algoritmo de substituição de página decide qual página será removida para dar lugar à nova. O simulador implementa três algoritmos: FIFO, LRU e Random, descritos a seguir.

2.3.1. FIFO (FIRST-IN, FIRST-OUT)

O algoritmo FIFO substitui a página que está na memória há mais tempo, tratando os quadros como uma fila circular.

- **Vantagens**: Implementação simples, com baixo custo computacional, usando um ponteiro para a página mais antiga.
- **Desvantagens**: Ignora a frequência ou recenticidade de uso, podendo remover páginas mais acessadas. Pode sofrer da Anomalia de Belady, onde aumentar a memória física aumenta os page faults.

No simulador, o FIFO é implementado com uma lista que rastreia a ordem de chegada das páginas..

2.3.2. LRU (LEAST RECENTLY USED)

O algoritmo LRU substitui a página menos recentemente usada, baseado no Princípio da Localidade, que assume que páginas acessadas recentemente têm maior probabilidade de serem acessadas novamente.

- **Vantagens**: Geralmente eficiente, aproximando-se do algoritmo ótimo, pois considera o padrão de acesso dos processos.
- **Desvantagens**: Exige rastreamento do histórico de acessos, o que aumenta a complexidade. No simulador, o LRU é aproximado usando uma lista ordenada que atualiza a posição das páginas a cada acesso.

2.3.3. ALEATÓRIO (RANDOM)

O algoritmo Random seleciona uma página para substituição de forma aleatória.

- **Vantagens**: Implementação extremamente simples, com overhead mínimo.
- **Desvantagens**: Desempenho imprevisível, pois não considera o comportamento do processo. Pode remover páginas críticas ou poupar páginas obsoletas por acaso. No simulador, é implementado usando uma função de geração de números aleatórios.

Os algoritmos são avaliados com base no número de page faults e gravações de páginas sujas, influenciados pelos parâmetros de tamanho de página e memória física, permitindo comparar sua eficiência em diferentes cenários.

3. DESENVOLVIMENTO E IMPLEMENTAÇÃO

Esta seção detalha a construção do simulador de memória virtual, abordando as ferramentas utilizadas, as estruturas de dados projetadas, a arquitetura do software e a implementação dos algoritmos de substituição de página (FIFO, LRU e Random), conforme os requisitos do projeto.

3.1 FERRAMENTAS E AMBIENTE DE DESENVOLVIMENTO

O simulador foi desenvolvido em Linguagem C, utilizando o padrão C11 para garantir portabilidade e eficiência no gerenciamento de memória. A escolha do C atende aos requisitos do projeto e permite controle de baixo nível, ideal para simulações de sistemas operacionais.

O ambiente de desenvolvimento foi baseado em linha de comando, com compilação realizada pelo GCC (GNU Compiler Collection). O comando de compilação incluiu a flag -Wall para habilitar avisos e -o progvirtual para gerar o executável, garantindo controle rigoroso sobre o processo de construção.

3.2 ESTRUTURA DE DADOS PRINCIPAL

A memória física foi modelada por um vetor dinâmico de estruturas QuadroPagina, alocado com base nos parâmetros sizeFis (tamanho da memória física, em KB) e sizePg (tamanho da página, em KB). Cada estrutura contém:

- bool ocupado: Indica se o quadro está em uso ou livre.
- unsigned int numero_pagina: Armazena o número da página lógica mapeada no quadro.
- bool suja: Bit de controle (dirty bit) que indica se a página foi modificada por uma operação de escrita ('W').
- unsigned long long contador_lru: Registra o "momento" do último acesso à página, usado pelo algoritmo LRU.

O número total de quadros é calculado como sizeFis / sizePg, e o vetor é inicializado com todos os quadros marcados como não ocupados.

3.3 ARQUITETURA GERAL DO SIMULADOR

O simulador segue um fluxo modular, descrito nos seguintes passos:

1. Inicialização: Lê e valida os argumentos da linha de comando (<polisub>, <arquivo.log>, <sizePg>, <sizeFis>). Calcula o número de quadros (sizeFis / sizePg) e o deslocamento de bits (s = log2(sizePg * 1024)) para converter endereços em números de página. Aloca dinamicamente o vetor de quadros e inicializa todos como não ocupados.
2. Processamento do Log: Lê o arquivo de log linha a linha com fscanf, extraindo o endereço hexadecimal e o tipo de operação ('R' ou 'W').
3. Cálculo da Página: Converte o endereço em número de página usando a operação page = addr >> s.
4. Simulação de Acesso:
    - Hit: Se a página está na memória, atualiza o contador_lru (para LRU) e, se a operação for 'W', marca o bit sujo como verdadeiro.
    - Miss: Incrementa o contador de page faults. Aloca um quadro livre, se disponível, ou chama o algoritmo de substituição. Se o quadro substituído contém uma página suja, incrementa o contador de gravações em disco.
5. Finalização: Após processar o log, imprime um relatório com:
    - Configuração utilizada (algoritmo, arquivo, sizePg, sizeFis).
    - Total de acessos à memória.
    - Número de page faults.
    - Número de páginas sujas gravadas no disco (excluindo páginas sujas remanescentes).

3.4 IMPLEMENTAÇÃO DOS ALGORITMOS DE SUBSTITUIÇÃO

Os algoritmos são selecionados na inicialização com base no argumento <polisub> e acionados quando ocorre um page fault sem quadros livres.

3.4.1 FIFO (FIRST-IN, FIRST-OUT)

O FIFO utiliza um ponteiro circular (ponteiro_fifo) que aponta para o próximo quadro a ser substituído. Após a substituição, o ponteiro é incrementado com ponteiro_fifo = (ponteiro_fifo + 1) % total_quadros. Se a página substituída for suja, o contador de gravações em disco é incrementado.

3.4.2 LRU (LEAST RECENTLY USED)

O LRU usa um contador global (contador_global_lru), incrementado a cada acesso à memória. Em um hit ou miss, o contador_lru do quadro é atualizado com o valor atual do contador global. Para substituição, o quadro com o menor contador_lru é escolhido. Se a página substituída for suja, o contador de gravações é incrementado.

3.4.3 ALEATÓRIO (RANDOM)

O Random seleciona um quadro vítima gerando um número aleatório com rand() % total_quadros, inicializado com srand(time(NULL)). Se a página substituída for suja, o contador de gravações é incrementado.

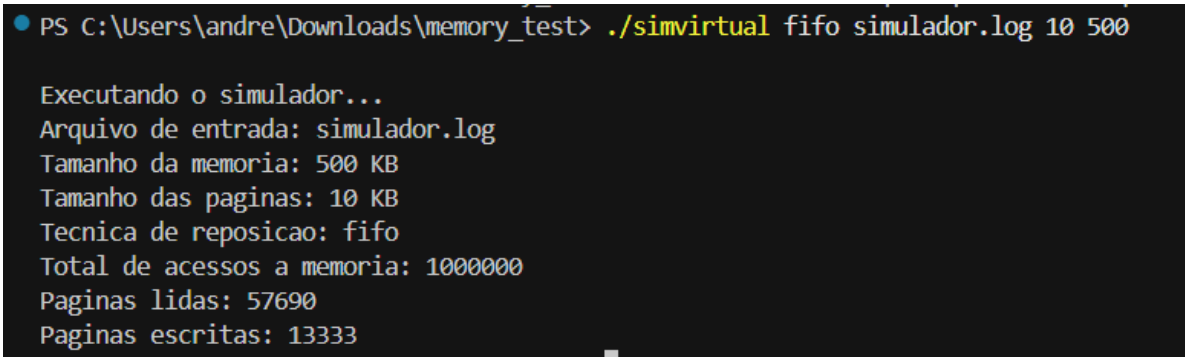
4. METODOLOGIA DE TESTES E RESULTADOS

Nosso simulador de memória recebe argumentos de entrada e deve gerar um relatório contendo os argumentos e número de acessos, leituras e escritas realizadas.

- Metodologia**:** Testamos nosso simulador ao inserir os quatro argumentos de entrada para que seja gerado o respectivo relatório.
- Resultados:

Considerando o modelo solicitado: simulador <lru|fifo|random> <arquivo.log> <tam_pagina_KB> <tam_memoria_KB> [debug]

Teste 1: simvirtual fifo simulador.log 10 500



Dentre os 1.000.000 de acessos, a memória apresentou 57.690 page faults e 13.000 dirty page. O FIFO remove a página mais antiga na memória, sem considerar se ela foi usada recentemente ou não. Apresenta um número intermediário de *page faults* e *dirty pages*. Pode funcionar mal em padrões de acesso onde as páginas mais antigas ainda estão sendo utilizadas.

Teste 2: simvirtual lru simulador.log 10 500

```
PS C:\Users\andre\Documents\simulador_memoria> ./simvirtual lru simulador.log 10 500

Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho da memoria: 500 KB
Tamanho das paginas: 10 KB
Tecnica de reposicao: lru
Total de acessos a memoria: 1000000
Paginas lidas: 47057
Paginas escritas: 9972
PS C:\Users\andre\Documents\simulador_memoria>
```

Dentre os 1.000.000 de acessos, a memória apresentou 47.057 page faults e 9.972 dirty page. O LRU remove a página que não foi usada há mais tempo. Apresentou melhor desempenho geral, pois teve menos falhas de página e menos escritas no disco, o que significa que está fazendo substituições mais inteligentes.

Teste 3: simvirtual random simulador.log 10 500

```
PS C:\Users\andre\Documents\simulador_memoria> ./simvirtual random simulador.log 10 500

Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho da memoria: 500 KB
Tamanho das paginas: 10 KB
Tecnica de reposicao: random
Total de acessos a memoria: 1000000
Paginas lidas: 56413
Paginas escritas: 13851
```

Dentre os 1.000.000 de acessos, a memória apresentou 56.413 page faults e 13.851 dirty page. O random remove uma página aleatória. O resultado foi quase tão ruim quanto o FIFO, mas com mais dirty pages. A falta de lógica nas substituições causa mais trocas e escritas desnecessárias.

5. ANÁLISE DOS RESULTADOS

- O tamanho da memória física é tão importante quanto a política de substituição. Mais RAM significa menos faults em geral. Conforme analisamos os resultados descritos acima, constatamos que:

- **LRU** é robusto em cenários onde o padrão de acesso tem “localidade temporal” (as mesmas páginas são acessadas repetidamente).

- **FIFO** pode ser competitivo quando há memória suficiente e não há forte localidade temporal.

- **RANDOM** costuma ser o pior em fault rate, mas requer zero manutenção de contadores ou listas.

6. CONCLUSÃO

O presente trabalho teve como objetivo principal o desenvolvimento, a implementação e a análise detalhada de um simulador de memória virtual desenvolvido na Linguagem C, com foco específico na avaliação comparativa do desempenho de diferentes algoritmos de substituição de páginas, nomeadamente FIFO, LRU e RANDOM. Por meio da execução de simulações baseadas em traços reais de acesso à memória, foi possível observar, de forma prática e quantitativa, o impacto direto que as políticas de gerenciamento de memória e as configurações específicas do sistema exercem sobre a eficiência geral do processo de substituição de páginas.

A implementação do simulador não apenas permitiu uma análise empírica dos algoritmos mencionados, mas também proporcionou uma compreensão mais prática, aprofundada e contextualizada dos conceitos teóricos relacionados ao gerenciamento de memória virtual, reforçando e solidificando de maneira significativa o conhecimento teórico previamente adquirido em sala de aula. Além disso, o desenvolvimento do simulador possibilitou a exploração de cenários variados, contribuindo para uma visão mais clara das vantagens, limitações e particularidades de cada algoritmo em diferentes condições operacionais.

7. REFERÊNCIAS

TANENBAUM, Andrew S.; BOS, Herbert. *Modern Operating Systems*. 4. ed. Boston: Pearson, 2015.

GEEKSFORGEEKS. *Page Replacement Algorithms in Operating Systems*. Disponível em: https://www.geeksforgeeks.org/operating-systems/page-replacement-algorithms-in-operating-systems/. Acesso em: 7 ago. 2025.