Simulador de Memória Virtual

Gabriel F. C. da Silva, Jean E. W. Meier

Centro de Ciências Tecnológicas – Universidade do Estado de Santa Catarina (UDESC)
89.219-710 – Joinville – SC – Brazil

{gabriel.silva1110,jean.meier45}@edu.udesc.br

1. Resumo

O programa *simula_memoria_virtual.c* implementa um simulador dos três principais algoritmos de substituição de páginas utilizados no gerenciamento de memória virtual: FIFO (*First In, First Out*), LRU (*Least Recently Used*) e OPT (Algoritmo Ótimo). O simulador permite comparar a eficiência destes algoritmos através da contagem de *page faults* gerados para diferentes configurações de memória.

1. Abstract

The program *simula_memoria_virtual.c* implements a simulator of the three main page replacement algorithms used in virtual memory management: FIFO (First In, First Out), LRU (Least Recently Used) and OPT (Optimal Algorithm). The simulator allows comparing the efficiency of these algorithms by counting the number of page faults generated for different memory configurations.

2. Introdução

A paginação de memória é uma técnica de gerenciamento de memória virtual que divide a memória física em blocos de tamanho fixo chamados quadros, e a memória virtual do processo em blocos chamados páginas. Quando uma página referenciada não está na memória física (ocorrendo um *page fault*), o sistema operacional precisa carregá-la do disco, substituindo uma página existente. Os algoritmos de substituição de página definem qual página será removida.

Há três algoritmos principais para a realização da paginação de memória (FIFO, LRU e Ótimo), mas a escolha do algoritmo depende do equilíbrio entre overhead computacional, padrões de acesso à memória e requisitos de desempenho do sistema.

Acerca dos algoritmos, de forma resumida, FIFO é simples, mas ineficiente em cenários com padrões repetitivos. LRU equilibra eficiência e viabilidade, sendo o mais usado na prática. O algoritmo Ótimo serve como benchmark para avaliar outros algoritmos, mas não é implementável em sistemas reais. Veremos estes algoritmos de forma mais aprofundada posteriormente.

3. Estrutura do código

3.1. Bibliotecas e Definições

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_REFS 100000
```

- stdio.h: Funções de entrada/saída (scanf, printf, fprintf);
- stdlib.h: Funções utilitárias (atoi);
- MAX_REFS: Limite máximo de referências que podem ser processadas.

2.2. Função Principal (main)

A função main realiza:

1. Validação de argumentos: Verifica se o número de quadros foi fornecido;
2. Leitura de entrada: Lê as referências de páginas da entrada padrão (stdin);
3. Execução dos algoritmos: Chama as três funções de simulação;
4. Exibição de resultados: Apresenta os *page faults* de cada algoritmo formatados.

3. Implementação dos algoritmos

3.1. Algoritmo FIFO (*First In, First Out*)

```
int simula_fifo(int frames, int* refs, int n_refs)
```

Princípio: Remove a página mais antiga na memória quando necessário substituir.

Estruturas de dados:

- memoria[frames]: Vetor representando os quadros de memória;
- pos: Índice circular para controle da ordem de inserção.

Funcionamento:

1. Inicializa todos os quadros com -1 (vazio);
2. Para cada referência:
    a. Verifica se a página já está na memória;
    b. Se não estiver: substitui a página na posição pos e incrementa page_faults;
    c. Atualiza *pos* de forma circular: pos = (pos + 1) % frames.

Complexidade: O(n_refs × frames) para verificação de presença na memória.

3.2. Algoritmo LRU (Least Recently Used)

```
int simula_lru(int frames, int* refs, int n_refs)
```

Princípio: Remove a página menos recentemente utilizada quando necessário substituir.

Estruturas de dados:

- memoria[frames]: Vetor representando os quadros de memória;
- ultima_vez[frames]: Vetor que armazena o *timestamp* do último acesso de cada quadro.

Funcionamento:

1. Inicializa memória e *timestamps* com -1;
2. Para cada referência:
    a. Se a página está na memória: atualiza seu *timestamp*;
    b. Se não estiver: encontra o quadro com menor *timestamp* (LRU) e o substitui;
    c. Incrementa *page_faults* quando há substituição.

Complexidade: O(n_refs × frames) para busca e determinação do LRU.

3.3. Algoritmo OPT (Ótimo)

```
int simula_opt(int frames, int* refs, int n_refs)
```

Princípio: Remove a página que será referenciada mais distante no futuro (ou nunca mais).

Estruturas de dados:

- memoria[frames]: Vetor representando os quadros de memória;
- distancia: Posição da próxima ocorrência de cada página.

Funcionamento:

1. Para cada referência que causa *page fault*:
    a. Examina cada página na memória;
    b. Procura a próxima ocorrência de cada página nas referências futuras;
    c. Se uma página nunca mais será usada: a escolhe imediatamente;
    d. Caso contrário: escolhe a página com maior distância futura.

Complexidade: O(n_refs × frames × n_refs) devido à busca futura para cada página.

4. Análise comparativa dos algoritmos

4.1. FIFO

- Vantagens: simples implementação, baixo *overhead*;
- Desvantagens: não considera frequência de uso, pode apresentar anomalia de Belady;
- Uso recomendado: sistemas com restrições de memória e processamento.

4.2. LRU

- Vantagens: boa aproximação do comportamento ideal, considera localidade temporal;
- Desvantagens: *overhead* para manter *timestamps*;

- Uso recomendado: sistemas com boa localidade temporal de referência.

4.3. OPT

- Vantagens: teoricamente ótimo, menor número de *page faults* possível;
- Desvantagens: impraticável (requer conhecimento do futuro);
- Uso recomendado: apenas para comparação e análise teórica.

5. Implementação dos algoritmos

5.1. Entrada

- Linha de comando: `./programa <numero_quadros>`
- Stdin: Uma referência de página por linha (números inteiros)

5.2. Saída

```
X quadros, Y refs: FIFO: Z PFs, LRU: W PFs, OPT: V PFs
```

Onde:

- X = número de quadros de memória;
- Y = total de referências processadas;
- Z, W, V = *page faults* para FIFO, LRU e OPT, respectivamente.

6. Casos de teste e validação

O programa foi testado com três cenários distintos:

1. arquivo1.txt (24 referências);
2. arquivo2.txt (30 referências);
3. arquivo3.txt (100.000 referências).

6.1. Resultados Destacados

Com o padrão sequencial cíclico, observou-se que:

- FIFO e LRU apresentaram performance crítica (quase 100% de *page faults*) em número pequeno de quadros e referências;
- OPT demonstrou superioridade significativa mesmo em testes pesados;
- A escolha do algoritmo é fortemente dependente do padrão de acesso.

7. Exemplos de execução

Compilação:

```
gcc simula_memoria_virtual.c -o simula_memoria_virtual
```

Execução (Linux/Unix):

```
./simula_memoria_virtual 4 < referencias.txt
```

Execução (Windows PowerShell):

```
Get-Content referencias.txt | .\simula_memoria_virtual.exe 4
```

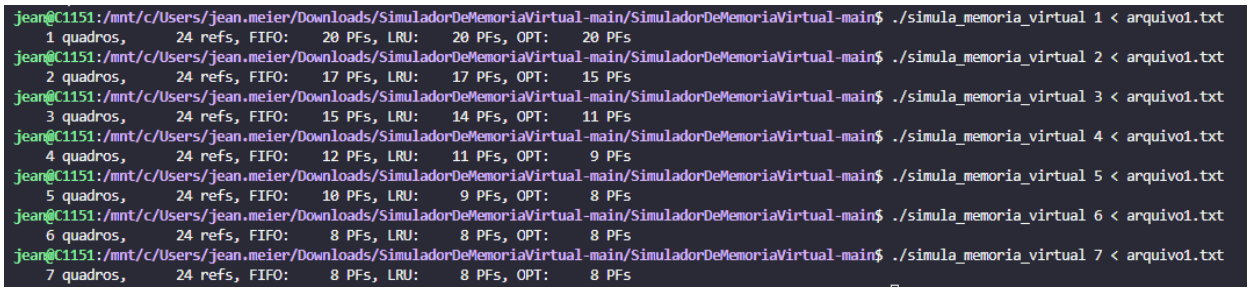
Exemplo de saída:

```
4 quadros,     24 refs: FIFO:     12 PFs, LRU:     11 PFs, OPT:     9 PFs
```

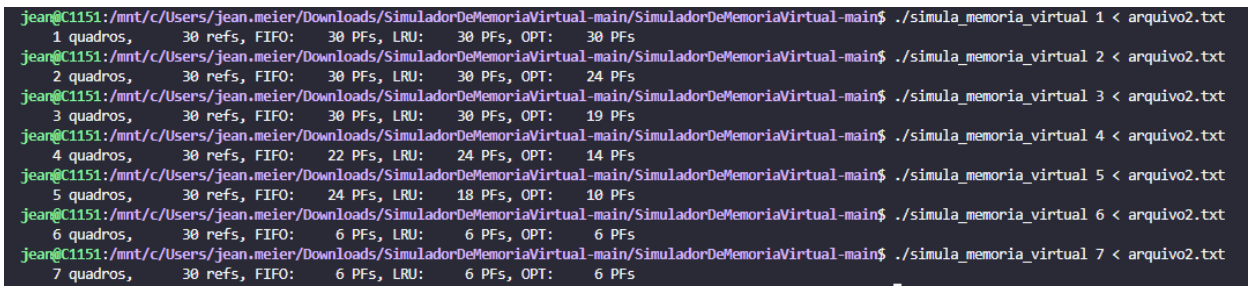
8. Resultados experimentais

Resultados obtidos rodando o compilador GCC pelo terminal do Visual Studio Code.

8.1. arquivo1.txt (24 referências)



8.2. Referencias2.txt (30 referências)

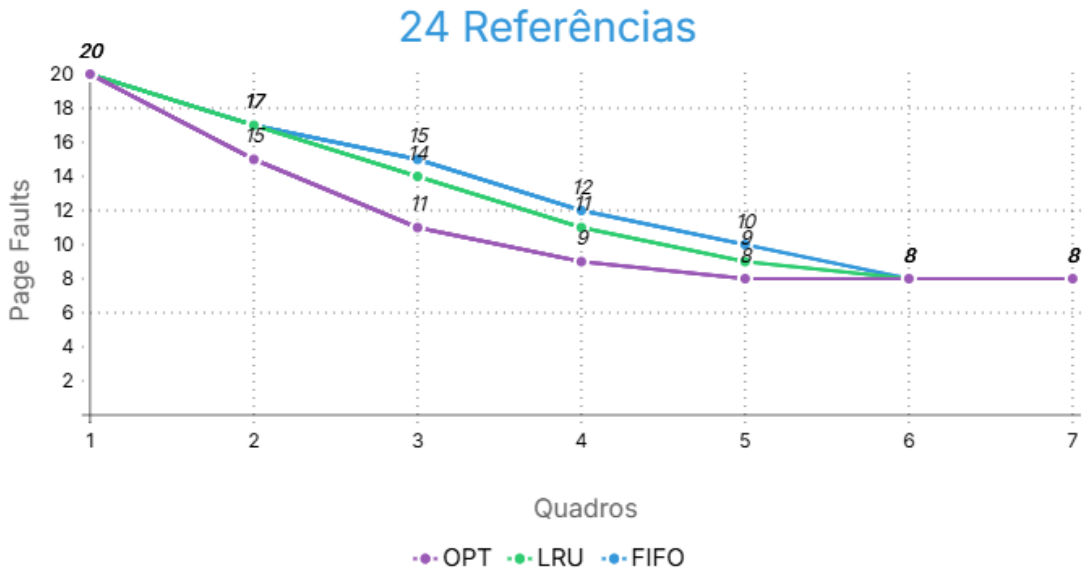


8.3. Referencias3.txt (10.000 referências)

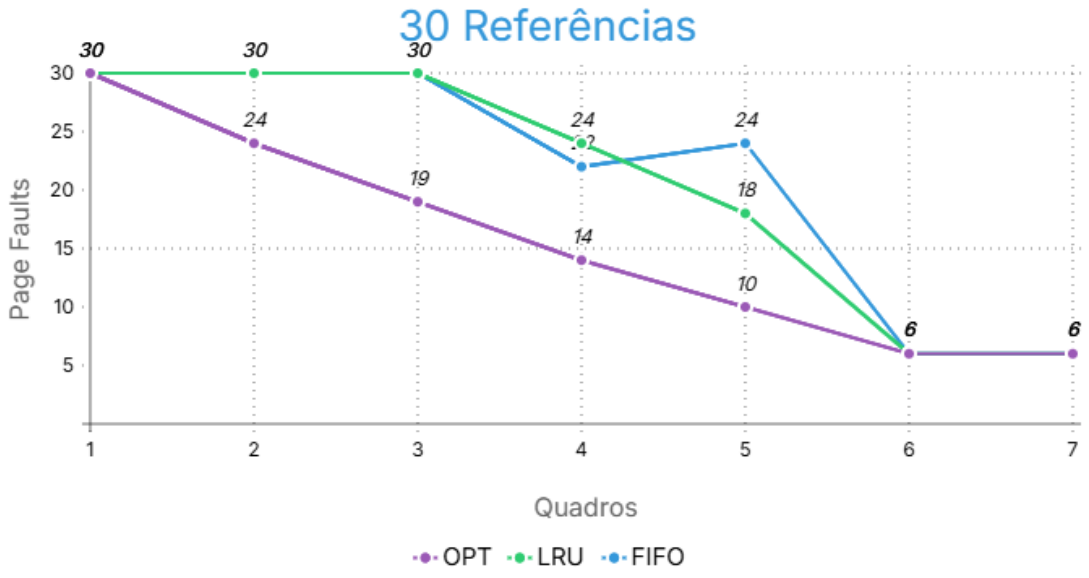
```
jean@C1151:/mnt/c/Users/jean.meier/Downloads/SimuladorDeMemoriaVirtual-main/SimuladorDeMemoriaVirtual-main$ ./simula_memoria_virtual 64 < arquivo3.txt
   64 quadros,  100000 refs, FIFO:  3219 PFs, LRU:  2535 PFs, OPT:  1629 PFs
jean@C1151:/mnt/c/Users/jean.meier/Downloads/SimuladorDeMemoriaVirtual-main/SimuladorDeMemoriaVirtual-main$ ./simula_memoria_virtual 256 < arquivo3.txt
  256 quadros,  100000 refs, FIFO:  1172 PFs, LRU:   986 PFs, OPT:   695 PFs
jean@C1151:/mnt/c/Users/jean.meier/Downloads/SimuladorDeMemoriaVirtual-main/SimuladorDeMemoriaVirtual-main$ ./simula_memoria_virtual 1024 < arquivo3.txt
 1024 quadros,  100000 refs, FIFO:   684 PFs, LRU:   684 PFs, OPT:   684 PFs
jean@C1151:/mnt/c/Users/jean.meier/Downloads/SimuladorDeMemoriaVirtual-main/SimuladorDeMemoriaVirtual-main$ ./simula_memoria_virtual 4096 < arquivo3.txt
 4096 quadros,  100000 refs, FIFO:   684 PFs, LRU:   684 PFs, OPT:   684 PFs
```

9. Gráficos comparativos

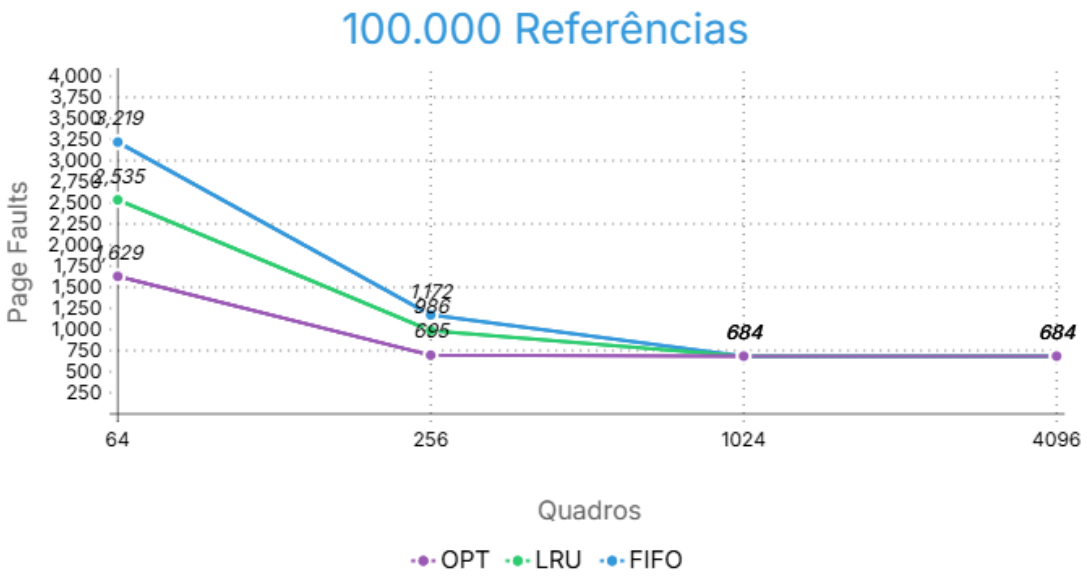
9.1. arquivo1.txt (24 referências)



9.2. arquivo2.txt (30 referências)



9.3. arquivo3.txt (100.000 referências)



10. Conclusões técnicas

O simulador demonstra que:

1. Não existe algoritmo universalmente superior. A eficiência depende do *workload*, ou se o algoritmo é aplicável na vida real;
2. Padrões sequenciais podem ser problemáticos para FIFO e LRU com memória limitada;
3. OPT serve como *baseline* teórico para avaliar algoritmos práticos;
4. A quantidade de memória influencia dramaticamente a performance de todos os algoritmos;
5. O padrão de acesso às páginas é um fator crítico na escolha do algoritmo.

11. Ambiente de execução

- Linguagem: C (padrão ANSI C);
- Compilador usado: GCC 11.4.0;
- Plataformas: WSL2 (Ubuntu 22.04);
- Processador: Intel(R) Core(TM) i5-10500T CPU @ 2.30GHz, 2301 Mhz, 6 Núcleo(s), 12 Processador(es) Lógico(s)
- Memória: 16 GB (15,7 GB utilizável);
- Entrada: Leitura via stdin (redirecionamento de arquivo);
- Saída: Formato padronizado com alinhamento de colunas;
- Validação: Verificação de argumentos de linha de comando.

Referências

MAZIERO, Carlos. **Sistemas Operacionais:** Gestão de memória - Paginação em disco. 2020. Disponível em: https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-slides-17.pdf. Acesso em 01 jul. 2025.

Paging. **OSDev**, 2025. Disponível em: https://wiki.osdev.org/Paging. Acesso em 03 jul. 2025.