Universidade Federal do Ceará
Sistemas Operacionais 2025.2
Professor: Fernando Antonio Mota Trinta

Ramon Nicolas Gomes Luna: 536534
Ernesto Dalva de medeiros: 541803
Diogo Santiago Oliveira 536808
July Santiago coelho: 555769

Paginação em Disco

Este documento descreve a estrutura, as decisões de projeto e os detalhes de implementação de um simulador de algoritmos de substituição de páginas. O objetivo do software é permitir a análise comparativa de desempenho entre diferentes algoritmos de gerenciamento de memória (FIFO, LRU, Ótimo, Clock, Segunda Chance, NRU, LFU e MFU) calculando valores como faltas de página (page faults) e evicções.

2. Estrutura do Software

O projeto foi desenvolvido em **Python** e segue o paradigma de **Orientação a Objetos**

2.1 Padrão de Projeto Strategy (Herança)

Para gerenciar a complexidade de múltiplos algoritmos, utilizou-se uma variação do padrão **Strategy**.

- **Classe Abstrata (PageReplacementAlgorithm):** Define a interface comum e o estado compartilhado.
- **Classes Concretas (FIFO, LRU, etc.):** Herdam da classe base e implementam a lógica específica de decisão no método access().
- **Polimorfismo:** O módulo principal (pager.py) trata todos os algoritmos de forma uniforme. Ele invoca o método simulator.access(page) sem precisar conhecer a implementação interna da estratégia escolhida.

3. Estruturas de Dados e Representação da Memória

Para atender ao requisito de mapeamento de informações de controle, o sistema utiliza **Estrutura de Arrays (Structure of Arrays)**. Em vez de criar objetos complexos para cada frame/página, utilizamos listas paralelas indexadas pelo ID da moldura (0 a N-1).

3.1 Mapeamento de Controle

1. **Memória Física (self.memory):** Uma lista de inteiros de tamanho fixo.
    - Armazena o Page ID presente na moldura.
    - O valor -1 indica que a moldura está livre (*Free Frame*).
2. **Bits de Referência (self.reference_bits):** Lista paralela usada pelos algoritmos *Clock* e *NRU*. Simula o bit de hardware R que é setado quando uma página é acessada.
3. **Bits de Modificação (self.m_bits):** Lista paralela exclusiva do *NRU*. Simula o bit M (Dirty Bit).
4. **Contadores de Frequência (self.counts):** Dicionário (Hash Map) utilizado por *LFU* e *MFU* para mapear Page ID -> Número de Acessos.

3.2 Lista Circular Virtual

Para algoritmos como FIFO, Clock e NRU, não utilizamos estruturas de dados encadeadas. A circularidade é implementada matematicamente através de aritmética modular no índice do ponteiro:

self.pointer = (self.pointer + 1) % self.capacity

4. Detalhes de Implementação dos Algoritmos

4.1 Algoritmos Básicos

- **FIFO (First-In, First-Out):** Utiliza um ponteiro circular simples. Substitui a página na posição atual do ponteiro e avança. É o algoritmo mais simples e base para o Clock.
- **LRU (Least Recently Used):** Implementado com uma lista auxiliar (usage history) que funciona como uma pilha temporal.
    - *Hit:* Remove a página da posição atual e a reinsere no final da lista (topo).
    - *Evicção:* Remove a página do início da lista (base), que representa o elemento menos recentemente usado.
- **Ótimo (OPT):** Implementado com "lookahead". O algoritmo recebe o *trace* completo de referências futuras. Na evicção, calcula a distância temporal de cada página na memória até sua próxima ocorrência e remove a que estiver mais distante (ou que não ocorrer mais). Serve como *baseline* teórico.

4.2 Algoritmos baseados em Relógio/Segunda Chance

A implementação destacou a diferença entre lógica física e lógica lógica:

- **Segunda Chance (SC):** Implementado usando uma **Fila Física**. Se a página na cabeça da fila tem Bit R=1, ela é removida e reinserida no final (resetando R=0). Isso envolve movimentação de dados na memória.
- **Clock:** Implementado usando **Lista Circular Estática**. As páginas não mudam de lugar; apenas o ponteiro se move e altera os bits de referência. É computacionalmente mais eficiente que o SC, embora a lógica de decisão seja idêntica.

4.3 Algoritmo NRU (Not Recently Used)

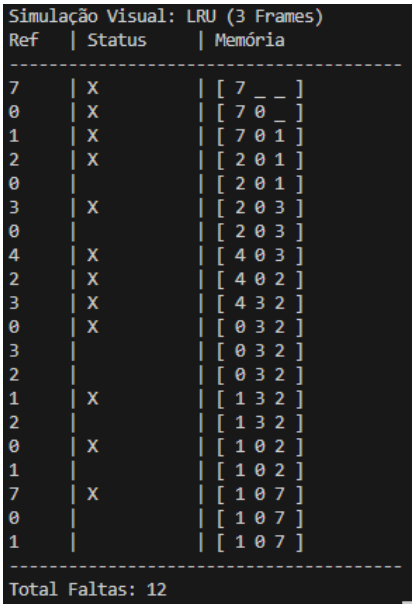
Esta implementação simula o comportamento do hardware e do SO:

1. **Simulação de Interrupção de Relógio:** A cada `RESET_INTERVAL` (10 acessos), todos os bits `R` são zerados para renovar a definição de "recente".
2. **Classes de Prioridade:** As páginas são divididas em 4 classes baseadas em (R, M):
    - Classe 0: R=0, M=0 (Melhor para remover)
    - Classe 1: R=0, M=1
    - Classe 2: R=1, M=0
    - Classe 3: R=1, M=1 (Pior para remover)
3. **Evicção:** O algoritmo realiza uma busca circular hierárquica. Ele percorre a memória procurando pela primeira página de Classe 0. Se não encontrar, percorre novamente buscando Classe 1, e assim sucessivamente.

4.4 Algoritmos de Contagem (LFU e MFU)

- **LFU:** Substitui a página com menor contador de acessos.
- **MFU:** Substitui a página com maior contador de acessos (assumindo que páginas com poucos acessos são novas e precisam de tempo para estabelecer seu conjunto de trabalho).
- **Decisão:** Os contadores são reiniciados quando uma página é removida da memória, evitando que páginas antigas "poluam" a decisão futura (problema de *aging*).

5. Ferramentas e Funcionalidades Extras



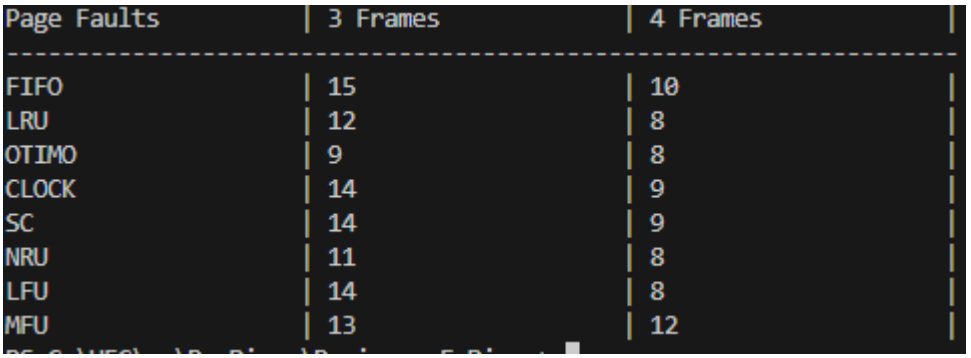
O simulador (`pager.py`) inclui funcionalidades avançadas para análise:

5.1 Modo Visual (`--visual`)

Permite a execução passo a passo, exibindo o estado da memória a cada referência.

5.2 Tabela Comparativa (`--algo ALL`)

Automatiza a execução de todos os 8 algoritmos para configurações de 3 e 4 molduras



Page Faults	3 Frames	4 Frames
FIFO	15	10
LRU	12	8
OTIMO	9	8
CLOCK	14	9
SC	14	9
NRU	11	8
LFU	14	8
MFU	13	12