UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA
BACHARELADO INTERDISCIPLINAR EM TECNOLOGIA DA INFORMAÇÃO

Francisco A. V. C. Sampaio,
Gabriel D. M. Sousa,
Gustavo K. F. Nunes,
Jackson R. O. Santos,
Francisco D. S. Bruno,
Bruno G. S. Silveira

Implementação de um algoritmo de LRU para page-in de memória virtual

PAU DOS FERROS
2025

1 Introdução

O gerenciamento de memória é uma parte essencial de qualquer sistema operacional, permitindo que vários processos sejam executados simultaneamente sem sobrecarregar a memória principal (RAM). Para realizar esse gerenciamento de forma eficiente, os sistemas operacionais utilizam algoritmos de substituição de páginas, os quais determinam qual página deve ser removida da memória quando é necessário liberar espaço para novas páginas.

Neste relatório, abordamos a implementação de um dos algoritmos de substituição de páginas mais populares, o algoritmo *Least Recently Used* (LRU), que substitui a página menos recentemente utilizada. O algoritmo LRU baseia-se na suposição de que páginas que não são acessadas frequentemente têm menor probabilidade de serem usadas novamente no futuro.

A implementação deste algoritmo foi realizada em Java, simulando o comportamento de um gerenciador de memória com um disco rígido (HD) e uma memória RAM. O código apresentado neste relatório inclui a estrutura de dados para armazenar as páginas, a lógica do algoritmo LRU e as interações com o usuário para realizar operações como *Page In* e *referenciamento* de páginas.

2 Objetivos

O objetivo deste projeto é implementar o algoritmo LRU para gerenciar a substituição de páginas em um sistema de memória virtual simulada. As operações realizadas pelo sistema incluem:

- Carregar páginas do HD para a RAM (Page In).
- Referenciar uma página na RAM, atualizando seu tempo de acesso.
- Visualizar o estado atual da RAM e do HD.

3 Ferramentas Utilizadas

Para a implementação do algoritmo LRU, utilizamos as seguintes ferramentas:

- **Linguagem de Programação**: Java
- **Estruturas de Dados**:
  - **Classe** `Page`: Representa as páginas de memória com os atributos `nome` (identificador da página) e `tempo` (marca de tempo que rastreia o último acesso da página).
  - **Array de objetos** `Page` **para a RAM**: Um array fixo de tamanho 4 (`Page[] ram`), usado para armazenar as páginas ativas na memória principal.
  - **Array de objetos** `Page` **para o HD**: Um array fixo de tamanho 6 (`Page[] hd`), usado para armazenar as páginas que estão em memória secundária (HD).
- **IDE**: Visual Studio Code

4 Estrutura de Dados

A estrutura central da implementação é a classe `Page`, que representa uma página de memória, com os seguintes atributos:

- `nome`: Identificador da página (string).
- `tempo`: O tempo de acesso à página, usado para determinar a substituição no algoritmo LRU.

Além disso, o sistema simula duas áreas de memória:

- **RAM**: Memória principal com um número fixo de 4 páginas.
- **HD**: Memória secundária com 6 páginas.

5 Funcionamento do Algoritmo LRU

O algoritmo LRU funciona mantendo um registro do tempo de acesso de cada página na RAM e sempre substituindo a página com o menor tempo de acesso, ou seja, a menos recentemente utilizada.

5.1 Etapas do Algoritmo LRU:

1. **Page In**: Quando uma nova página precisa ser carregada na RAM:
   - Se a página já estiver na RAM, seu tempo de acesso é atualizado.
   - Caso contrário, a página menos recentemente usada (determinada pelo menor valor do atributo `tempo`) é removida da RAM e substituída pela nova página.

2. **Referenciamento**: Quando o usuário referencia uma página na RAM, o tempo de acesso da página é atualizado para o tempo atual, indicando que a página foi utilizada mais recentemente.

3. **Visualização**: O sistema permite que o usuário visualize o estado atual da RAM e do HD, exibindo as páginas presentes em cada área de memória.

A seguir, detalhamos as operações implementadas:

6 Operações Implementadas

O programa oferece um menu interativo para o usuário realizar as seguintes operações:

6.1 1. Realizar Page In

Essa operação permite carregar uma página do HD para a RAM. Caso a RAM esteja cheia, o algoritmo LRU é executado para substituir a página menos recentemente utilizada.

6.2 2. Referenciar Quadro da RAM

Nesta operação, o usuário escolhe uma página da RAM para referenciá-la. O tempo de acesso dessa página é atualizado, conforme o algoritmo LRU.

6.3 3. Visualizar RAM e Disco Atualmente

O sistema exibe o estado atual da RAM e do HD, mostrando as páginas presentes em cada área de memória.

7 Exemplo de Execução

Considerando o seguinte estado inicial:

- RAM: {PARTE 3 DE A, PARTE 1 DE C, PARTE 2 DE C, PARTE 3 DE C}
- HD: {PARTE 1 DE A, PARTE 2 DE A, PARTE 1 DE B, PARTE 2 DE B, PARTE 3 DE B, PARTE 4 DE B}

Se o usuário escolher a opção 1 (Realizar Page In) e selecionar a página `PARTE 1 DE A` no HD, a página menos recentemente usada será removida da RAM e substituída por essa página. A operação de substituição é realizada de acordo com o algoritmo LRU.

8 Vantagens e Desvantagens

8.1 Vantagens do LRU

- Aproxima-se do comportamento ideal de substituição, minimizando o número de *page faults*.
- Melhor desempenho que algoritmos como FIFO (*First-In, First-Out*), pois evita remover páginas que ainda estão sendo frequentemente usadas.
- Pode ser implementado de maneira relativamente eficiente utilizando estruturas como listas duplamente encadeadas e tabelas de dispersão.

8.2 Desvantagens do LRU

- O custo computacional pode ser alto, pois a busca pela página menos recentemente usada exige percorrer toda a RAM para encontrar o menor valor de `tempo`.
- A implementação com listas encadeadas pode reduzir esse custo, mas adiciona complexidade ao código.
- Pode ser ineficiente para sistemas com um grande número de páginas, pois a atualização constante do tempo de referência pode impactar o desempenho.

8.3 Possíveis Melhorias

Para otimizar a implementação do LRU no código, algumas estratégias poderiam ser aplicadas:

- Utilização de uma lista duplamente encadeada em conjunto com um *hash map* para garantir que a busca e a atualização das páginas ocorram em tempo $O(1)$.

- Implementação do algoritmo *LRU Approximation* (*Clock Algorithm*), que reduz a necessidade de atualizações constantes dos tempos de referência.
- Substituir a busca linear pela página menos recentemente usada por uma estrutura de dados mais eficiente, como uma fila de prioridade baseada em *heaps*.

9 Considerações Finais

A implementação do algoritmo LRU demonstrou ser eficaz para gerenciar o uso da memória, mantendo as páginas mais recentemente acessadas na RAM. A utilização de estruturas de dados como listas e arrays foi adequada para armazenar as páginas e realizar a substituição com eficiência.

Para futuras melhorias, seria interessante explorar variações do algoritmo LRU, como o *Clock Algorithm*, que oferece uma solução mais simples e com menor sobrecarga computacional. Além disso, a implementação de mais operações de manipulação de memória poderia enriquecer o simulador, tornando-o mais robusto e capaz de simular situações mais complexas.