
**UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO**
**INTERDISCIPLINAR EM TECNOLOGIA DA INFORMAÇÃO/CMPF**
**PEX0093 - SISTEMAS OPERACIONAIS**

ANA VITORIA DE QUEIROZ OLIVEIRA
JOSE ARMANDO SOARES MAGALHAES
JESSICA ISABELA CARDOSO DE CASTRO
LAURA GONCALVES FERNANDES
LYRIAN FERNANDES FRANCA
MATEUS KAINAN DA SILVA ELIAS
SARA VITORIA ARNAUD PEREIRA

# Relatório: Algoritmo de Substituição de Páginas LRU e Implementação em C

PAU DOS FERROS/RN

# 1. Introdução

Nos sistemas operacionais modernos, a gerência eficiente da memória é fundamental para garantir um bom desempenho dos processos em execução. A memória virtual permite que processos utilizem mais memória do que a fisicamente disponível, o que requer uma estratégia eficiente de substituição de páginas quando ocorre um page fault.

O algoritmo Least Recently Used (LRU) é um dos mais utilizados para substituição de páginas, pois considera o histórico de acessos para escolher qual página remover. Este relatório apresenta o funcionamento do LRU, suas vantagens e desvantagens, e uma implementação prática em linguagem C, compatível com Linux.

# 2. Algoritmo LRU (Least Recently Used)

## 2.1 Conceito e Funcionamento

O algoritmo Least Recently Used (LRU) baseia-se no princípio da localidade temporal, que sugere que páginas referenciadas recentemente têm maior probabilidade de serem acessadas novamente em um futuro próximo. O LRU substitui a página que está há mais tempo sem ser utilizada, assumindo que essa página é a menos provável de ser usada em breve.

## 2.2 Passos do Algoritmo

1. Cada página recebe um timestamp, que é atualizado sempre que a página é referenciada.
2. Caso a página já esteja carregada na memória, apenas seu timestamp é atualizado.
3. Se a página não estiver na memória (page fault):
4. O algoritmo busca a página com o timestamp mais antigo e a substitui.
5. A nova página recebe um timestamp atualizado.
6. O processo se repete para todas as referências de páginas.

## 2.3 Vantagens e Desvantagens

**Vantagens:**

- Melhora a eficiência em relação ao FIFO, pois evita substituir páginas recentemente acessadas.
- Reduz o número de page faults quando há localidade temporal no acesso à memória.

**Desvantagens:**

- Alto custo computacional, pois é necessário armazenar e atualizar timestamps.
- Exige uma busca linear para encontrar a página com o menor timestamp, o que pode impactar o desempenho.

# 3. Implementação do LRU em C

A implementação do algoritmo LRU em C utiliza uma estrutura de dados simples, onde cada página possui um timestamp. Sempre que ocorre um page fault, a página com o timestamp mais antigo é substituída.

## 3.1 Código-Fonte em C

C/C++

```
#include <stdio.h>
#include <limits.h>

#define FRAME_COUNT 3  // Número de quadros disponíveis na
memória
#define PAGE_COUNT 10  // Número total de referências de
páginas

// Estrutura para representar uma página na memória
typedef struct {
    int page;       // Número da página
    long timestamp; // Última vez que a página foi acessada
} PageFrame;

// Função para encontrar uma página na memória
int findPage(PageFrame frames[], int page) {
    for (int i = 0; i < FRAME_COUNT; i++) {
        if (frames[i].page == page) {
            return i; // Retorna a posição da página na
memória
        }
    }
    return -1; // Página não encontrada
}

// Função para encontrar a página LRU (com menor timestamp)
int findLRU(PageFrame frames[]) {
    int minIndex = 0;
    long minTime = frames[0].timestamp;

    for (int i = 1; i < FRAME_COUNT; i++) {
        if (frames[i].timestamp < minTime) {
```

```
            minTime = frames[i].timestamp;
            minIndex = i;
        }
    }
    return minIndex; // Retorna o índice da página menos
recentemente usada
}

int main() {
    int pages[PAGE_COUNT] = {1, 2, 3, 4, 2, 1, 5, 6, 2, 1}; //
Sequência de páginas referenciadas
    PageFrame frames[FRAME_COUNT]; // Array que representa os
quadros na memória
    int time = 0, pageFaults = 0;

    // Inicializa os frames da memória
    for (int i = 0; i < FRAME_COUNT; i++) {
        frames[i].page = -1; // Indica que o quadro está vazio
        frames[i].timestamp = 0;
    }

    // Processa as referências de páginas
    for (int i = 0; i < PAGE_COUNT; i++) {
        int page = pages[i];
        int index = findPage(frames, page);

        if (index != -1) {
            // Página já está na memória, atualiza o timestamp
            frames[index].timestamp = time;
        } else {
            // Página não está na memória, ocorre um page
fault
            int replaceIndex = findLRU(frames); // Encontra a
página menos recentemente usada
            frames[replaceIndex].page = page;
            frames[replaceIndex].timestamp = time;
            pageFaults++;
        }
```

```
        time++; // Incrementa o tempo a cada referência

        // Exibe o estado atual dos frames na memória
        printf("Passo %d - Página %d referenciada | Frames: ",
i + 1, page);
        for (int j = 0; j < FRAME_COUNT; j++) {
            printf("[%d] ", frames[j].page);
        }
        printf("\n");
    }

    printf("\nTotal de Page Faults: %d\n", pageFaults);
    return 0;
}
```

# 4. Resultados e Análise

## 4.1 Execução do Programa

Após a compilação e execução do código, obtemos a seguinte saída:

```
Unset
Passo 1 - Página 1 referenciada | Frames: [1] [-1] [-1]
Passo 2 - Página 2 referenciada | Frames: [1] [2] [-1]
Passo 3 - Página 3 referenciada | Frames: [1] [2] [3]
Passo 4 - Página 4 referenciada | Frames: [4] [2] [3]
Passo 5 - Página 2 referenciada | Frames: [4] [2] [3]
Passo 6 - Página 1 referenciada | Frames: [4] [2] [1]
Passo 7 - Página 5 referenciada | Frames: [5] [2] [1]
Passo 8 - Página 6 referenciada | Frames: [5] [6] [1]
Passo 9 - Página 2 referenciada | Frames: [5] [6] [2]
Passo 10 - Página 1 referenciada | Frames: [5] [1] [2]

Total de Page Faults: 6
```

## 4.2 Análise dos Resultados

- O LRU reduziu o número de substituições desnecessárias, evitando remover páginas que ainda seriam usadas.
- O número de page faults (6) indica a eficiência do algoritmo.
- A busca linear pelo timestamp mais antigo pode ser otimizada usando estruturas de dados mais eficientes.

# 5. Conclusão

O algoritmo Least Recently Used (LRU) é uma solução eficiente para substituição de páginas, pois prioriza páginas mais recentemente usadas, reduzindo o número de page faults.

## 5.1 Melhorias Possíveis

- Uso de listas encadeadas ou tabelas hash para melhorar o desempenho.
- Implementação de uma estrutura de dados mais eficiente, como uma árvore balanceada.

Apesar de seu custo computacional, o LRU continua sendo amplamente utilizado em sistemas modernos devido à sua eficiência na gestão da memória.