Universidade LaSalle

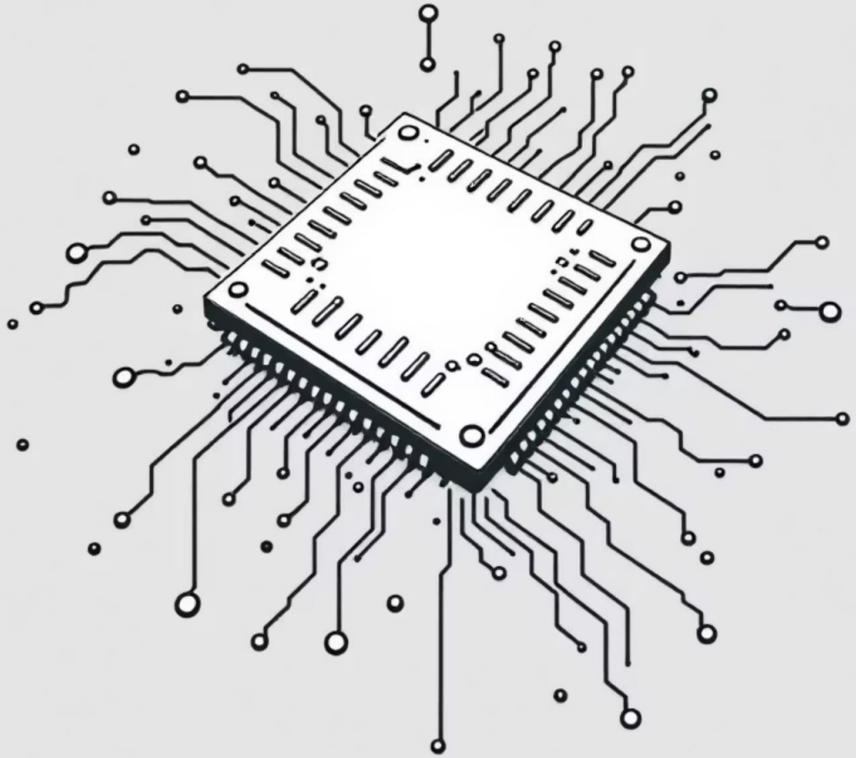
Bacharelado em Ciência da Computação

Sistemas Operacionais

Aula começa as
10h30min

Prof. Me. Filipo Novo Mór

filipo.mor *at* unilasalle *dot* edu *dot* br



Capítulo 3 – Gerenciamento de Memória

Explorando os conceitos fundamentais, técnicas e estratégias utilizadas pelos sistemas operacionais para gerenciar um dos recursos mais críticos: a memória.

Introdução ao Gerenciamento de Memória

Todo programa precisa estar na memória principal para ser executado. A memória é um recurso limitado e caro, especialmente quando comparado à memória secundária.

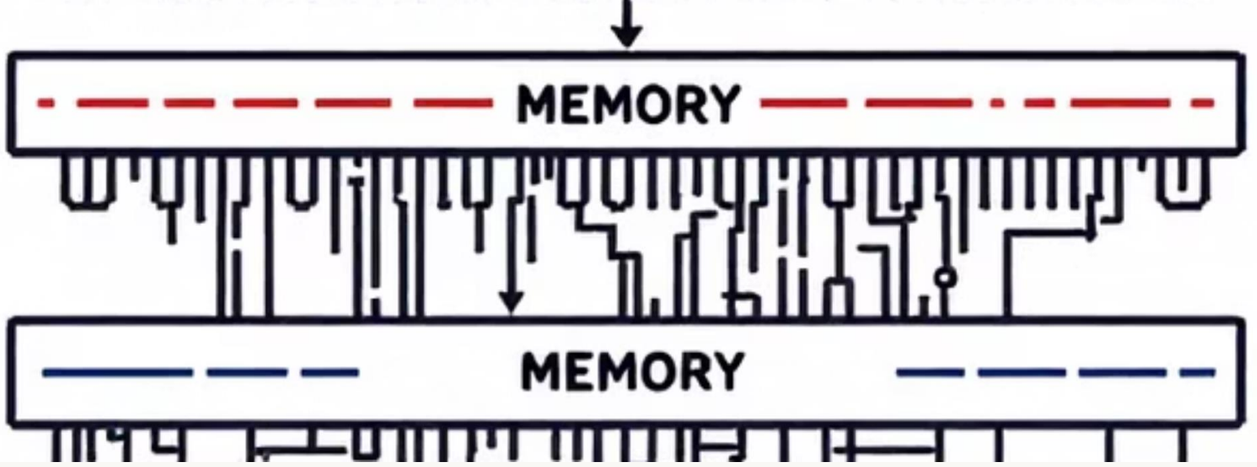
O gerenciador de memória é o componente crítico do sistema operacional responsável por controlar o uso da memória, alocar e desalocar espaço para os processos, e gerenciar as trocas entre memória principal e secundária.

Sistemas que movem processos

Paginação e swapping

Sistemas que não movem

Alocação estática



Conceitos Básicos de Memória

Vetor de Endereços

A memória é organizada como um vetor de endereços que a CPU acessa diretamente

Execução de Instruções

Instruções são carregadas, decodificadas e executadas sequencialmente

Contador de Programa

O PC/IP indica sempre a próxima instrução a ser executada

A UCP não distingue entre dados e instruções durante o processamento; ela simplesmente manipula endereços. Os tipos de memória incluem RAM, ROM, cache, registradores e memória secundária.

Ligação de Endereços

O programa passa por diferentes fases de ligação ao longo do seu ciclo de vida, cada uma definindo quando e como os endereços são convertidos em físicos.

Tempo de Compilação

1

Endereço fixo definido durante a compilação, gerando código absoluto. Processo deve sempre ser carregado no mesmo endereço.

Tempo de Carregamento

2

Endereços relocáveis são definidos apenas quando o programa é carregado na memória, permitindo flexibilidade.

Tempo de Execução

3

Endereços dinâmicos são calculados pela MMU durante a execução, oferecendo máxima flexibilidade e suporte à multiprogramação.

Carregamento e Ligação Dinâmica

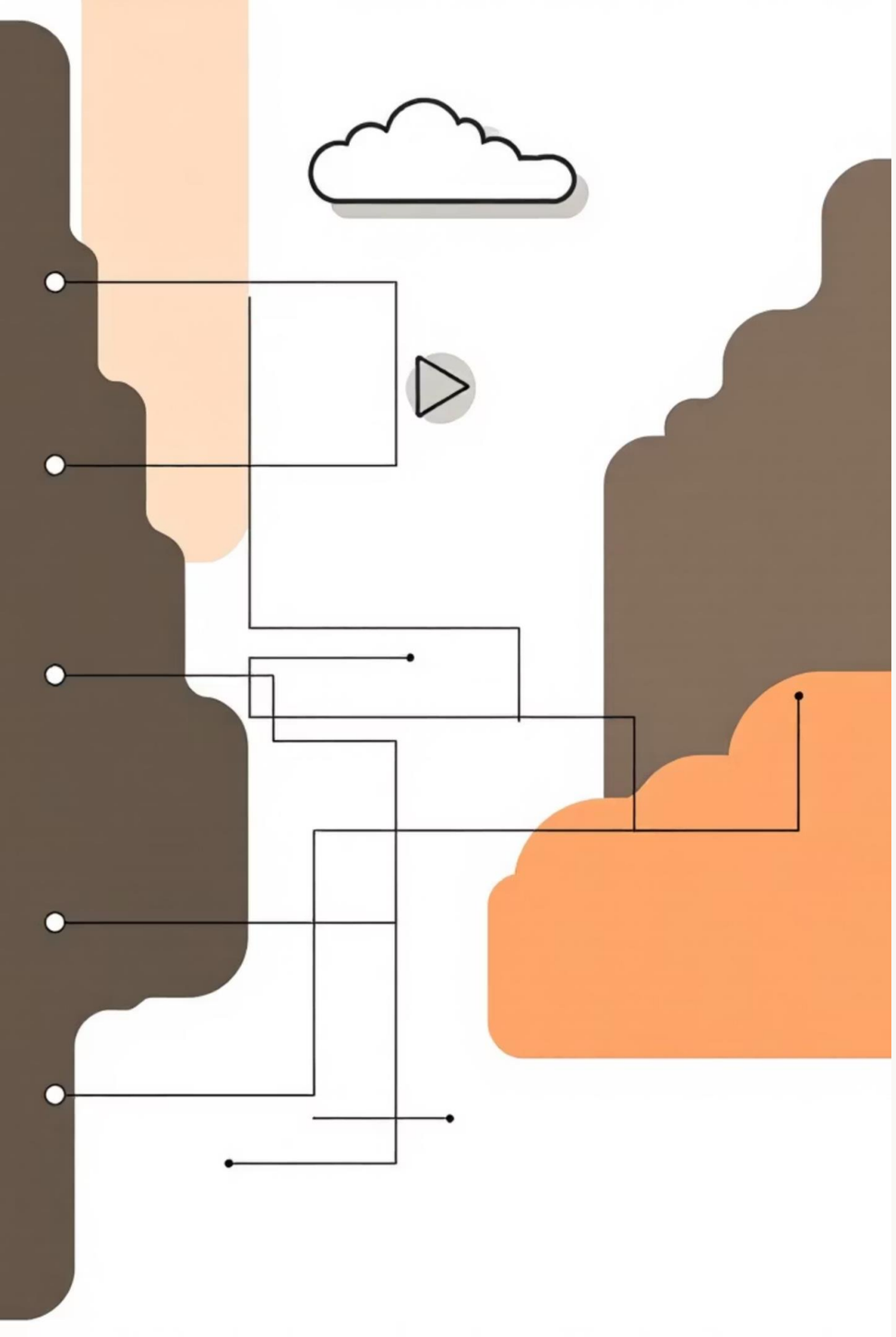
Carregamento Dinâmico

Rotinas são carregadas apenas quando chamadas, economizando memória e acelerando a inicialização. Exemplo prático: Microsoft Word.

Ligação Dinâmica

Adia a associação de bibliotecas até o momento da execução, usando stubs para localizar e carregar rotinas quando necessário.

Esta técnica permite atualização de bibliotecas compartilhadas sem necessidade de recompilar os programas, facilitando manutenção e atualizações de segurança.



Endereçamento Lógico x Físico

Endereço Lógico

Também chamado de endereço virtual, é gerado pela CPU durante a execução do programa. Independente da localização física real.

Endereço Físico

Endereço real na memória RAM onde os dados estão efetivamente armazenados.

A MMU (Memory Management Unit) realiza o mapeamento dinâmico entre endereços lógicos e físicos, protegendo o SO e outros processos, além de suportar relocação dinâmica e multiprogramação segura.

Swapping: Troca de Processos

Memória Principal

Processos ativos em execução

Swapping

Movimentação bidirecional

Memória Secundária

Processos em espera

Técnica que move processos entre memória principal e secundária quando a memória física não comporta todos os processos simultaneamente. Permite multiprogramação e melhor aproveitamento da CPU.

Vantagens

- Suporta mais processos que a capacidade física
- Implementa prioridades (Roll out/Roll in)
- Melhora utilização da CPU

Desvantagens

- Tempo de troca elevado devido à E/S lenta
- Overhead de gerenciamento
- Hoje usado em versões modificadas (UNIX)

Alocação Contígua e Proteção

A memória principal é estrategicamente dividida entre o sistema operacional e os processos do usuário, garantindo isolamento e proteção.

Partição Única

Modelo mais simples onde apenas um processo de usuário executa por vez na memória

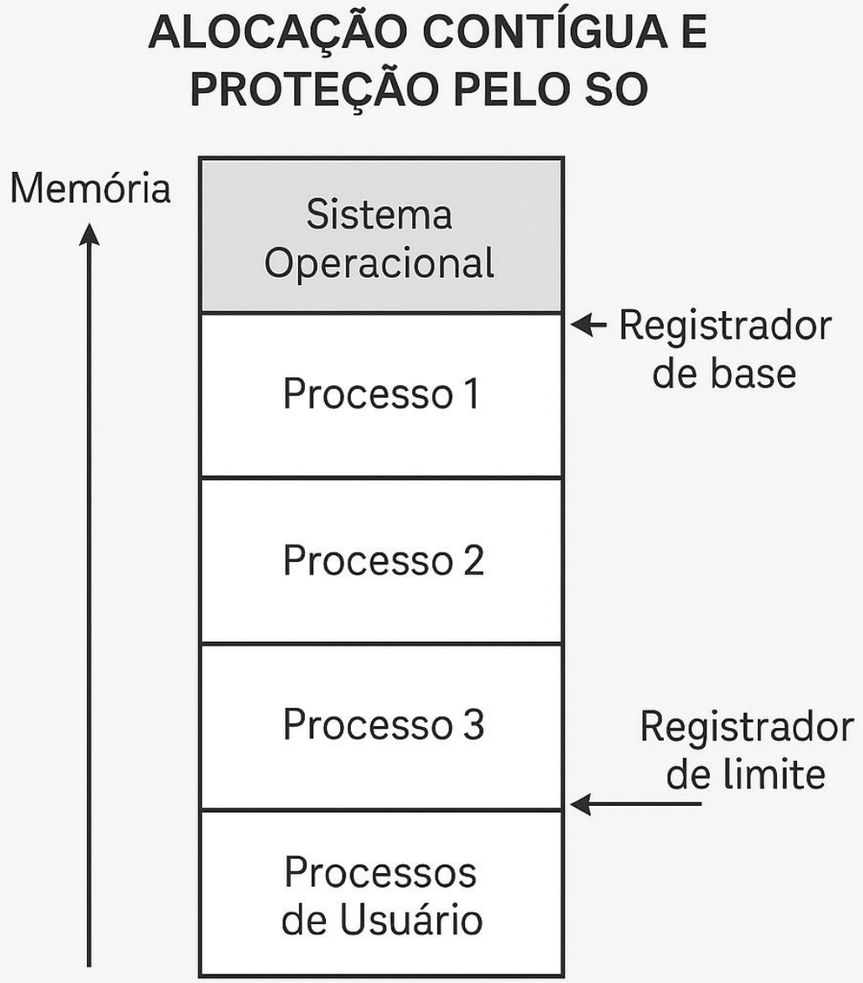
Partições Múltiplas

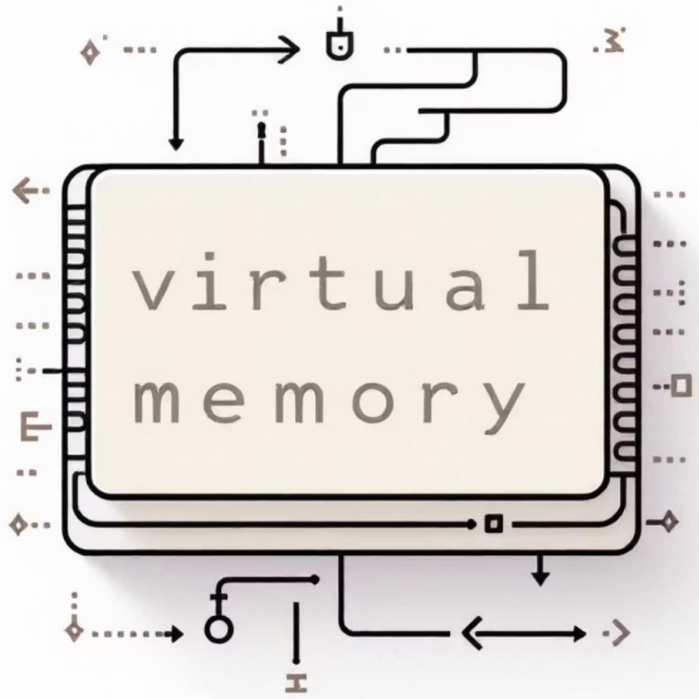
SO e múltiplos processos de usuários compartilham a RAM simultaneamente

Proteção por Registradores

Uso de registradores de limite e relocação para impedir acessos indevidos

O sistema operacional deve garantir que processos não acessem áreas de memória alheias ou o próprio kernel. Sistemas modernos como Linux e Solaris utilizam drivers carregáveis dinamicamente para maior flexibilidade.





Memória Virtual: Expandindo os Limites

> A memória virtual combina memória principal e secundária, criando a ilusão de uma capacidade muito maior do que a fisicamente disponível.

Maior Eficiência da CPU

Mais processos podem competir pelo processador, reduzindo tempo ocioso

Redução da Fragmentação

Melhor aproveitamento do espaço disponível através de técnicas inteligentes

Programas Maiores

Permite executar programas maiores que a memória física disponível

O conceito baseia-se no mapeamento entre endereços virtuais e reais, com apenas partes dos processos residindo na RAM a cada momento.

Paginação e Substituição de Páginas

A paginação implementa a memória virtual dividindo o espaço de endereçamento em páginas (lógicas) e a memória física em frames (físicos) de mesmo tamanho.

Falha de Página

Ocorre quando a página solicitada não está em RAM, exigindo carregamento da memória secundária.

Políticas de Carregamento:

- Por demanda: carrega quando necessário
- Antecipada: carrega páginas previsivelmente úteis
- Fixa/Variável: número de frames por processo

Algoritmos de Substituição

- Ótimo

  Teórico, substitui página que será usada mais tarde

- FIFO

  First In, First Out - remove página mais antiga

- LRU

  Least Recently Used - remove menos recentemente usada

- LFU e NRU

  Baseados em frequência e bits de referência

Objetivo: reduzir a taxa de falhas de página e otimizar o desempenho geral do sistema.

Algoritmos de Substituição: F.I.F.O.

https://github.com/ProfessorFilipo/Aulas_SisOp/blob/main/fifo_pages.c

```c
/*****************************************************************************
***           D E M O N S T R A C A O   A L G O R I T M O   F I F O       ***
*****************************************************************************
*** Prof. Filipo Mor - github.com/ProfessorFilipo                           ***
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int page_refs[] = {1, 2, 3, 4, 1, 5};   /* sequência de referência */
    int n = sizeof(page_refs) / sizeof(page_refs[0]);
    int capacity = 3;                       /* número de frames */
    int frames[10];                         /* alocar um pouco a mais */
    int i, j;

    /* inicializar frames com -1 (vazio) */
    for (i = 0; i < capacity; ++i) frames[i] = -1;

    int next = 0;        /* índice FIFO: próximo a ser substituído */
    int faults = 0;

    printf("FIFO troca de páginas de memória (capacidade=%d)\n\n", capacity);
    for (i = 0; i < n; ++i) {
        int ref = page_refs[i];
        int found = 0;
        /* verificar se ref já está em frames (hit) */
        for (j = 0; j < capacity; ++j) {
            if (frames[j] == ref) { found = 1; break; }
        }
        if (!found) {
            /* page fault: substituir o frame apontado por 'next' */
            frames[next] = ref;
            next = (next + 1) % capacity; /* circular */
            faults++;
            printf("Ref %d -> FAULT, frames: ", ref);
        } else {
            printf("Ref %d -> HIT   , frames: ", ref);
        }
        /* imprimir estado de frames */
        for (j = 0; j < capacity; ++j) {
            if (frames[j] == -1) printf(" - ");
            else printf("%2d ", frames[j]);
        }
        printf("\n");
    }

    printf("\nTotal de falhas de páginas (page faults): %d\n", faults);
    return 0;
}
```

```
FIFO troca de páginas de memória (capacidade=3)

Ref 1 -> FAULT, frames:  1  -  -
Ref 2 -> FAULT, frames:  1  2  -
Ref 3 -> FAULT, frames:  1  2  3
Ref 4 -> FAULT, frames:  4  2  3
Ref 1 -> FAULT, frames:  4  1  3
Ref 5 -> FAULT, frames:  4  1  5

Total de falhas de páginas (page faults): 6
```

Análise passo a passo

Referência	Situação	Memória após operação	Observação
1	Falha	1 – –	Primeiro carregamento
2	Falha	1 2 –	Segunda página
3	Falha	1 2 3	Memória cheia pela primeira vez
4	Falha	4 2 3	Página 1 (mais antiga) foi substituída
1	Falha	4 1 3	Página 2 era a mais antiga agora
5	Falha	4 1 5	Página 3 foi substituída

Algoritmos de Substituição: F.I.F.O. versus L.R.U.

Casos onde o FIFO é mais eficiente (menos page faults)

🔶 Exemplo 1:

Sequência: `{1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5}`

Capacidade: 3 quadros

- **FIFO → 9 falhas**
- **LRU → 10 falhas**

📘 *Explicação:*

O padrão de repetição faz o LRU "descartar" páginas que logo voltam a ser usadas.

O FIFO mantém algumas delas por mais tempo e evita algumas falhas.

🔶 Exemplo 2:

Sequência: `{1, 2, 3, 4, 5, 1, 2, 3}`

Capacidade: 3 quadros

- **FIFO → 8 falhas**
- **LRU → 9 falhas**

📘 *Explicação:*

A sequência cresce linearmente e repete no final.

O LRU tenta se ajustar dinamicamente, mas o FIFO mantém páginas antigas (1,2,3) que logo são reutilizadas — e acaba tendo menos falhas.

Casos onde o LRU é mais eficiente (menos page faults)

🔶 Exemplo 3:

Sequência: `{1, 2, 3, 1, 4, 2, 5, 1, 2, 3, 4, 5}`

Capacidade: 3 quadros

- **FIFO → 10 falhas**
- **LRU → 8 falhas**

📘 *Explicação:*

O LRU reconhece padrões de reutilização (1 e 2 voltam com frequência).

O FIFO descarta essas páginas antigas cedo demais, provocando mais *page faults*.

🔶 Exemplo 4:

Sequência: `{1, 2, 3, 4, 1, 2, 5, 6, 2, 1, 2, 3}`

Capacidade: 3 quadros

- **FIFO → 11 falhas**
- **LRU → 9 falhas**

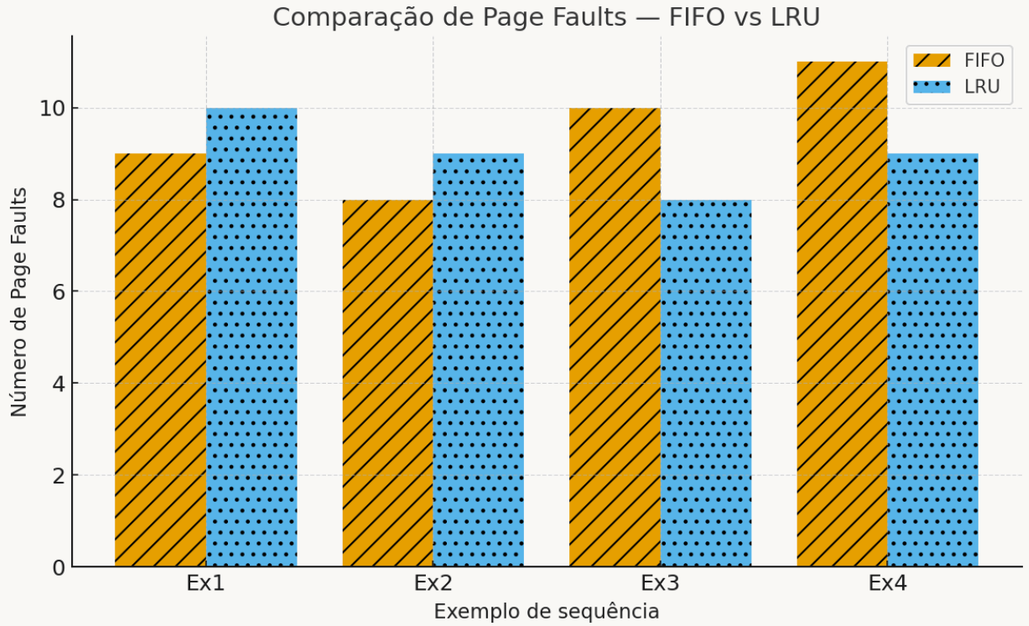
📘 *Explicação:*

O LRU se adapta melhor ao retorno de páginas "recentemente usadas" (como 1 e 2).

O FIFO substitui de forma cega, removendo páginas que ainda seriam úteis.

https://github.com/ProfessorFilipo/Aulas_SisOp/blob/main/comparacao_FIFO_LRU.c

Algoritmos de Substituição: F.I.F.O. versus L.R.U.



Situação	Sequência	Capacidade	Melhor algoritmo	Diferença esperada
FIFO melhor	{1,2,3,4,1,2,5,1,2,3,4,5}	3	FIFO	9 × 10
FIFO melhor	{1,2,3,4,5,1,2,3}	3	FIFO	8 × 9
LRU melhor	{1,2,3,1,4,2,5,1,2,3,4,5}	3	LRU	8 × 10
LRU melhor	{1,2,3,4,1,2,5,6,2,1,2,3}	3	LRU	9 × 11

https://github.com/ProfessorFilipo/Aulas_SisOp/blob/main/comparacao_FIFO_LRU.c

Universidade LaSalle

Bacharelado em Ciência da Computação

Sistemas Operacionais

Muito obrigado!

Prof. Me. Filipo Novo Mór

filipo.mor *at* unilasalle *dot* edu *dot* br