Faculdade de Computação e Informática FCI

Sistemas Operacionais

Prof. Lucas Cerqueira Figueiredo

GABRIEL DE SANTANA SANTOS

SIMULADOR DE PAGINAÇÃO DE MEMÓRIA

SÃO PAULO

05/2025

1. Introdução

1.1 Contextualização

A paginação é uma técnica fundamental de gerenciamento de memória em sistemas operacionais modernos que permite a alocação não contígua da memória física para processos. Esta técnica elimina a fragmentação externa e facilita o compartilhamento de recursos entre diferentes processos, sendo essencial para o funcionamento eficiente de sistemas multiprogramados.

O gerenciamento eficaz da memória virtual através da paginação envolve diversos conceitos importantes, incluindo a tradução de endereços virtuais para físicos, o tratamento de page faults e a implementação de algoritmos de substituição de páginas quando a memória física se torna insuficiente.

1.2 Objetivos do Projeto

Este projeto tem como objetivo desenvolver um simulador de paginação de memória que demonstre na prática os conceitos teóricos estudados em sala de aula. Os principais objetivos incluem:

- Implementar um sistema de tradução de endereços virtuais para físicos
- Simular o comportamento de múltiplos processos com seus respectivos espaços de endereçamento
- Implementar e comparar diferentes algoritmos de substituição de páginas
- Analisar o desempenho dos algoritmos através de métricas como taxa de page faults
- Visualizar o estado da memória física durante a execução da simulação

2. Descrição da Implementação

2.1 Estruturas de Dados Utilizadas

2.1.1 Estrutura de Página

```
typedef struct {
    int frame;          // Número do frame físico onde a página está alocada
    int presente;       // Flag indicando presença na memória física
    int tempo_carga;    // Timestamp do carregamento (para FIFO)
    int ultimo_acesso;  // Timestamp do último acesso (para LRU)
} Pagina;
```

Esta estrutura representa uma entrada na tabela de páginas de um processo, contendo todas as informações necessárias para o gerenciamento da página e implementação dos algoritmos de substituição.

2.1.2 Estrutura de Processo

```
typedef struct {
    int pid;                // Identificador único do processo
    int tamanho;            // Tamanho total em bytes
    int num_paginas;        // Número de páginas necessárias
    Pagina *tabela_paginas;  // Tabela de páginas do processo
} Processo;
```

Representa um processo no sistema, incluindo sua tabela de páginas individual e metadados essenciais.

2.1.3 Estrutura da Memória Física

```
typedef struct {
    EntradaMemoria *frames;  // Array de frames da memória
    int *tempo_carga;       // Timestamps de carregamento
    int num_frames;         // Número total de frames
} MemoriaFisica;
```

Modela a memória física do sistema, mantendo informações sobre cada frame e facilitando a implementação dos algoritmos de substituição.

2.1.4 Estrutura Principal do Simulador

```
typedef struct {
    Processo *processos;      // Array de processos
    int num_processos;        // Contador de processos
    MemoriaFisica memoria;    // Estrutura da memória física
    int tamanho_pagina;       // Tamanho das páginas
    int algoritmo;            // Algoritmo de substituição atual
    int tempo_atual;          // Contador de tempo da simulação
    int page_faults;          // Contador de page faults
    int total_acessos;        // Contador de acessos totais
} Simulador;
```

2.2 Algoritmos Implementados

2.2.1 Algoritmo FIFO (First In, First Out)

O algoritmo FIFO implementado utiliza o timestamp de carregamento de cada página para determinar qual deve ser substituída. A página que foi carregada há mais tempo (menor timestamp) é escolhida para substituição.

Implementação:

- Cada página recebe um timestamp no momento do carregamento
- Durante a substituição, percorre-se todos os frames buscando o menor timestamp
- A página com menor timestamp é removida para dar lugar à nova página

2.2.2 Algoritmo LRU (Least Recently Used)

O algoritmo LRU utiliza o timestamp do último acesso para determinar a página menos recentemente usada. A página que não foi acessada há mais tempo é escolhida para substituição.

Implementação:

- A cada acesso à página, o timestamp de último acesso é atualizado
- Durante a substituição, busca-se a página com menor timestamp de último acesso
- Esta estratégia tenta manter na memória as páginas mais utilizadas recentemente

2.3 Decisões de Projeto

2.3.1 Sistema de Timestamps

Optou-se por utilizar um contador global de tempo (tempo_atual) que é incrementado a cada acesso à memória. Este sistema simples e eficaz permite a implementação tanto do FIFO quanto do LRU sem complexidade adicional.

2.3.2 Estrutura Unificada para Algoritmos

Em vez de implementar estruturas separadas para cada algoritmo, utilizou-se uma abordagem unificada onde a mesma estrutura Página contém campos para ambos os algoritmos. Isso simplifica o código e permite fácil alternância entre algoritmos.

2.3.3 Visualização da Memória

Implementou-se uma função de visualização que mostra o estado da memória física após cada acesso, facilitando o entendimento do funcionamento dos algoritmos e a depuração do código.

2.4 Limitações da Implementação

1. **Número Fixo de Processos:** O simulador está limitado a 10 processos simultâneos
2. **Algoritmos Básicos:** Apenas FIFO e LRU foram implementados, sem variações mais sofisticadas

3. **Sem Persistência:** Não há salvamento do estado da simulação
4. **Interface Simples:** A interface é baseada em linha de comando sem recursos gráficos
5. **Sem TLB:** Não foi implementado Translation Lookaside Buffer para acelerar traduções

3. Análise Comparativa dos Algoritmos

3.1 Metodologia de Comparação

Para comparar os algoritmos FIFO e LRU, utilizou-se uma sequência fixa de acessos à memória com dois processos de tamanhos diferentes:

- Processo 1: 10KB (3 páginas)
- Processo 2: 15KB (4 páginas)
- Memória física: 16KB (4 frames)
- Tamanho da página: 4KB

3.2 Sequência de Teste

A sequência de acessos utilizada foi:

P1:1111, P1:5000, P2:2000, P2:10000, P1:3000,
P2:14000, P1:9000, P2:500, P1:1000, P2:8000

3.3 Resultados Observados

3.3.1 Algoritmo FIFO

- **Total de acessos:** 10
- **Page faults:** Variável conforme padrão de acesso
- **Comportamento:** Remove sempre a página carregada há mais tempo, independentemente de seu uso recente

3.3.2 Algoritmo LRU

- **Total de acessos:** 10
- **Page faults:** Geralmente menor que FIFO para padrões com localidade temporal
- **Comportamento:** Remove a página menos recentemente usada, mantendo páginas ativas na memória

3.4 Análise Comparativa

3.4.1 Vantagens do FIFO

- **Simplicidade:** Implementação mais simples e direta
- **Baixo overhead:** Requer apenas timestamp de carregamento
- **Previsibilidade:** Comportamento determinístico e fácil de entender

3.4.2 Vantagens do LRU

- **Eficiência:** Melhor desempenho com padrões que apresentam localidade temporal
- **Adaptabilidade:** Se adapta ao padrão de uso das páginas
- **Realismo:** Mais próximo do comportamento ideal de substituição

3.4.3 Desvantagens

- **FIFO:** Pode remover páginas frequentemente usadas.
- **LRU:** Maior complexidade de implementação e overhead de atualização

4. Funcionamento do Sistema

4.1 Processo de Tradução de Endereços

1. **Recepção do endereço virtual:** O sistema recebe um endereço virtual e o PID do processo
2. **Extração de página e deslocamento:** O endereço é dividido pelo tamanho da página
3. **Consulta à tabela de páginas:** Verifica se a página está presente na memória
4. **Tratamento de page fault:** Se não presente, carrega a página na memória
5. **Tradução final:** Calcula o endereço físico usando frame + deslocamento

4.2 Gerenciamento de Page Faults

1. **Detecção:** Identifica quando uma página não está na memória física
2. **Busca por frame vazio:** Procura frame disponível na memória
3. **Substituição:** Se não há frames vazios, aplica algoritmo de substituição
4. **Carregamento:** Simula o carregamento da nova página
5. **Atualização:** Atualiza estruturas de controle e timestamps

4.3 Coleta de Estatísticas

O simulador coleta automaticamente:

- Número total de acessos à memória
- Número de page faults ocorridos
- Taxa de page faults (percentual)
- Informações sobre substituições realizadas

---

5. Resultados e Validação

5.1 Testes Realizados

O simulador foi testado com diferentes cenários:

- **Teste básico:** Sequência simples com poucos processos
- **Teste de sobrecarga:** Mais páginas que frames disponíveis
- **Teste de localidade:** Acessos concentrados em poucas páginas
- **Teste aleatório:** Padrão de acesso sem localidade aparente

5.2 Validação dos Resultados

A validação foi realizada através de:

- **Verificação manual:** Cálculo manual dos page faults esperados
- **Logs detalhados:** Acompanhamento passo a passo da execução
- **Consistência:** Verificação da consistência das estruturas de dados
- **Conformidade:** Comparação com especificações teóricas dos algoritmos

5.3 Observações Importantes

1. O simulador demonstra corretamente o funcionamento dos algoritmos de paginação
2. A visualização do estado da memória facilita o entendimento do processo
3. As estatísticas geradas são precisas e úteis para análise de desempenho
4. O código é robusto e trata adequadamente casos extremos

6. Conclusões

6.1 Aprendizados Obtidos

Este projeto proporcionou uma compreensão prática aprofundada dos conceitos de paginação de memória, incluindo:

- **Tradução de endereços:** Entendimento do processo de conversão de endereços virtuais para físicos
- **Algoritmos de substituição:** Análise prática das diferenças entre FIFO e LRU
- **Gerenciamento de memória:** Experiência com estruturas de dados para controle de memória
- **Programação em C:** Desenvolvimento de habilidades em programação de sistemas

6.2 Desafios Enfrentados

Os principais desafios incluíram:

- **Gerenciamento de ponteiros:** Manipulação adequada de estruturas complexas
- **Sincronização de timestamps:** Manutenção consistente dos contadores de tempo
- **Depuração:** Identificação e correção de problemas na lógica dos algoritmos
- **Formatação de saída:** Produção de saída clara e informativa

6.3 Reflexões sobre o Projeto

O desenvolvimento deste simulador demonstrou a importância dos algoritmos de substituição de páginas no desempenho de sistemas operacionais. A diferença de comportamento entre FIFO e LRU evidencia como as escolhas de design podem impactar significativamente a eficiência do sistema.

O projeto também destacou a complexidade envolvida no gerenciamento de memória em sistemas reais, onde múltiplos processos competem por recursos limitados e decisões de alocação devem ser tomadas em tempo real.

7. Referências

1. TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 3ª ed. São Paulo: Pearson, 2010.

2. SILBERSCHATZ, A., GALVIN, P.B, GAGNE, G. **Fundamentos de Sistemas Operacionais: princípios básicos**. São Paulo: LTC, 2013.