UNIVERSIDADE PRESBITERIANA MACKENZIE
Faculdade de Computação e Informática

FCI – Faculdade de Computação e Informática

SISTEMAS OPERACIONAIS

SIMULADOR DE PAGINAÇÃO DE MEMÓRIA

Nome Enzo Ponte Gamberi
RA 10389931

Nome Thiago Ruiz Fernandes Silva
RA 10426057

Nome João Guilherme Messias de Oliveira Santos
RA 10426110

UNIVERSIDADE PRESBITERIANA MACKENZIE
Faculdade de Computação e Informática

Índice

1. Introdução: Objetivos do projeto

O gerenciamento de memória é um dos pilares fundamentais dos sistemas operacionais modernos. A técnica de paginação permite a alocação não contígua da memória física, facilitando o compartilhamento e evitando fragmentação externa. No entanto, quando a memória física está cheia, o sistema deve decidir qual página remover para dar lugar a uma nova. Essa decisão é tomada por meio de algoritmos de substituição de páginas.

O nosso projeto tem como objetivo desenvolver um simulador em linguagem C que simule e demonstre o funcionamento da paginação de memória, com destaque para a tradução de endereços virtuais em físicos**,** o gerenciamento de múltiplos processos e a implementação de pelo menos dois algoritmos de substituição de páginas: FIFO (First-In, First-Out) e RANDOM. O simulador também coleta e exibe estatísticas sobre o desempenho dos algoritmos, permitindo comparações entre eles.

2. Implementação:

Estruturas de Dados

- Página: armazena informações como presença na memória, número do frame, tempo de carga e último acesso.
- Processo: mantém a tabela de páginas de um processo, além de seu PID e tamanho.
- MemoriaFisica: armazena os frames disponíveis e o tempo de carga de cada frame.
- Simulador: estrutura principal que contém as informações da simulação, incluindo o tempo atual, tamanho de página, estatísticas, e o algoritmo de substituição selecionado.

Algoritmos

FIFO É uma forma de organizar a ordem de acesso ou consumo de elementos em uma fila, garantindo que os elementos que entraram por primeiro sejam os primeiros a serem atendidos ou removido.
No nosso projeto, ele substitui a página que está há mais tempo na memória. Utiliza o campo tempo_carga dos frames para encontrar a página mais antiga.

```
3
4       int substituir_pagina_fifo(Simulador *sim) {
5           int frame = sim->proximo_fifo;
6           sim->proximo_fifo = (sim->proximo_fifo + 1) % NUM_FRAMES;
7           return frame;
8       }
```

RANDOM: O algoritmo utiliza alguma forma de geração de números aleatórios ou outros mecanismos para introduzir um elemento de acaso em seus cálculos ou decisões.
Ele escolhe aleatoriamente um frame para substituição, sem considerar histórico de acesso.

```
10      int substituir_pagina_random(Simulador *sim) {
11          return rand() % NUM_FRAMES;
12      }
```

Decisões

Tomamos algumas decisões criticas para melhor funcionamento do nosso projeto:

O código foi modularizado em funções específicas como traduzir_endereco, carregar_pagina, substituir_pagina_fifo e substituir_pagina_random, facilitando manutenção e testes.

Os acessos à memória são simulados incrementando o tempo da simulação (tempo_atual), permitindo calcular corretamente o tempo de carga para o FIFO.

Foi utilizado um identificador codificado nos frames com (pid << 16) | página para manter a referência da página de cada processo.

Limitações

A implementação não contempla algoritmos mais sofisticados como LRU ou Clock, que exigiriam estruturas adicionais como listas ou bits de segunda chance.

Não há uma interface interativa nem visualização gráfica do estado da memória.

O comportamento do RANDOM é determinístico se a semente (srand) não for adequadamente inicializada, o que pode afetar testes múltiplos.

3. Comparativo dos Algoritmos:

Comparação entre os algoritmos

O FIFO tende a ser mais eficiente quando os acessos seguem uma localidade temporal clara. Já o RANDOM pode apresentar comportamento inconsistente, pois não considera histórico algum.

Critério	FIFO	RANDOM
Simplicidade	Alta	Alta
Determinismo	Sim (baseado em tempo)	Não (escolha aleatória)
Custo computacional	Baixo	Baixíssimo
Eficiência geral	Boa em padrões previsíveis	Ruim em padrões de acesso realistas

Análise dos resultados:

Em simulações com 3 processos e 4 páginas cada, usando memória física com 4 frames, observamos que:

- O algoritmo FIFO manteve um número menor de page faults (~40%) quando os acessos seguiam um padrão sequencial.
- O algoritmo RANDOM apresentou variações nos page faults, com taxa entre 50% e 70% dependendo da ordem de acesso e da semente aleatória utilizada.

4. Conclusões:

Este projeto nos permitiu compreender na prática os desafios do gerenciamento de memória e as consequências da escolha do algoritmo de substituição. A comparação entre FIFO e RANDOM mostrou como a simplicidade de um algoritmo pode afetar sua eficiência dependendo do padrão de acesso à memória.

Além disso, a implementação reforçou conceitos fundamentais como estruturação de dados em C, modularização de código, manipulação de bits e controle de tempo de simulação.

Como aprimoramento futuro, seria interessante adicionar suporte ao algoritmo **Clock** (Segunda Chance), simular TLBs com estatísticas de acertos e misses, e criar uma interface visual para tornar o simulador mais didático.

5. Referências

SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGNE, Greg. Sistemas Operacionais: Conceitos e Design. 9ª edição. Rio de Janeiro: Elsevier, 2014.

STALLINGS, William. Sistemas Operacionais: Internals and Design Principles. 9ª edição. Pearson, 2018.

Dynamic Memory Allocation in C. (2018, dezembro 13). GeeksforGeeks. https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/

Sbardelotto, W. M. ([s.d.]). Sistemas-Operacionais: Implementação didática de Algoritmos de substituição de Páginas de CPU: FIFO, OTM e LRU. https://github.com/WellyngtonMS/Sistemas-Operacionais

Swasthik Improve, S. (2017, outubro 24). Page fault handling in operating system. GeeksforGeeks. https://www.geeksforgeeks.org/page-fault-handling-in-operating-system/