UNIVERSIDADE PRESBITERIANA MACKENZIE
Faculdade de Computação e Informática

FCI – Faculdade de Computação e Informática

PROJETO 2 - SISTEMAS OPERACIONAIS

DESCRITIVO TÉCNICO

SIMULADOR DE PAGINAÇÃO DE MEMÓRIA

Nome Guilherme Teodoro de Oliveira
RA 10425362

Nome Luís Henrique Ribeiro Fernandes
RA 10420079

Nome Vinícius Brait Lorimier
RA 10420046

Nome Eduardo Henrique de Souza Cruz
RA 10358690

UNIVERSIDADE PRESBITERIANA MACKENZIE
Faculdade de Computação e Informática

Índice

1. Introdução: Contextualização e objetivos do projeto

O uso eficiente de memória é um dos principais desafios enfrentados pelos desenvolvedores nos dias atuais. À medida que os programas se tornam mais complexos e exigentes em termos de recursos, torna-se necessário implementar mecanismos que permitam a execução de múltiplos processos de forma eficiente. Um dos recursos mais críticos é a memória principal, cujo uso inadequado pode levar a pioras no desempenho do sistema.

Uma das técnicas mais utilizadas para o gerenciamento de memória é a paginação, que divide a memória física em blocos de tamanho fixo chamados quadros e os processos em blocos lógicos do mesmo tamanho chamados páginas. Esse mapeamento entre páginas e quadros permite um uso mais eficiente da memória e evita a fragmentação externa. Entretanto, quando a memória física está cheia, é necessário carregar uma nova página, fazendo com que o sistema tenha que decidir qual página remover, o que exige a adoção de um algoritmo de substituição de páginas.

A paginação tem como base o princípio da localidade, ou seja, a ideia de que os programas acessam um conjunto de páginas em um intervalo de tempo. Quando um processo tenta acessar uma página que não está na memória, ocorre uma falta de página (page fault), que pode ser extremamente custosa, pois implica acesso ao disco.

Entre os algoritmos de substituição, destacam-se o FIFO (First-In, First-Out) e o LRU (Least Recently Used). O algoritmo FIFO remove a página que está há mais tempo na memória, independentemente do seu uso recente. É simples de implementar, mas pode levar à substituição de páginas frequentemente usadas. Já o LRU baseia-se na ideia de que as páginas utilizadas recentemente tendem a ser reutilizadas em breve, sendo geralmente mais eficiente, embora mais complexo de implementar.

O projeto tem como objetivo implementar um simulador de substituição de páginas utilizando os algoritmos FIFO e LRU. Através da simulação, realizaremos observações e comparações do comportamento desses algoritmos em diferentes contextos, verificando o desempenho de ambos os algoritmos.

2. Descrição da Implementação:

Estruturas de Dados Utilizadas

O projeto utiliza estruturas de dados para simulação da memória e dos algoritmos de substituição. As principais estruturas são:

Vetores (Arrays): utilizados para armazenar os quadros de memória e as referências de páginas.

Fila (para FIFO): implementada com dois índices que controlam a inserção e a remoção de elementos, simulando o comportamento de uma fila circular.

Array de tempos (para LRU): utilizado para armazenar o tempo (ou ordem) de último acesso de cada página presente na memória.

Algoritmos Implementados

Algoritmo FIFO

O algoritmo FIFO é baseado em uma estrutura de fila. Seu funcionamento consiste em inserir as páginas nos quadros de memória em ordem de chegada. Quando a capacidade da memória é excedida e uma nova página precisa ser carregada, a página que está há mais tempo na memória (ou seja, a primeira inserida) é removida. Esse comportamento pode ser eficiente em implementações simples, mas pode gerar substituições não ótimas, especialmente em padrões de acesso com ciclos ou repetição frequente.

```c
// Algoritmo de substituição de página: FIFO (First-In, First-Out)
int substituir_pagina_fifo(Simulador *sim) {
    int mais_antigo = 0;
    int tempo_min = sim->tempo_atual + 1;

    for (int i = 0; i < sim->memoria.num_frames; i++) {
        if (sim->memoria.tempo_carga[i] < tempo_min) {
            tempo_min = sim->memoria.tempo_carga[i];
            mais_antigo = i;
        }
    }

    return mais_antigo;
}
```

Algoritmo LRU

O algoritmo LRU identifica qual página não é usada há mais tempo, ou seja, a menos recentemente utilizada, e a substitui. Isso exige que o sistema mantenha um histórico de acessos, normalmente implementado com vetores auxiliares que armazenam o tempo do último uso de cada página. A ideia principal do LRU é que páginas acessadas recentemente têm maior probabilidade de serem acessadas novamente em breve.

```c
// Algoritmo de substituição de página: LRU (Least Recently Used)
int substituir_pagina_lru(Simulador *sim) {
    int frame_lru = 0;
    int tempo_min = sim->tempo_atual + 1;

    for (int i = 0; i < sim->memoria.num_frames; i++) {
        int val = sim->memoria.frames[i];
        int pid = val >> 16;            // Extrai o PID (16 bits mais altos)
        int pagina = val & 0xFFFF;      // Extrai o número da página (16 bits mais baixos)

        Processo *proc = &sim->processos[pid - 1];
        int ultimo_acesso = proc->tabela_paginas[pagina].ultimo_acesso;

        if (ultimo_acesso < tempo_min) {
            tempo_min = ultimo_acesso;
            frame_lru = i;
        }
    }

    return frame_lru;
}
```

Decisões de Projeto

Algumas decisões importantes foram adotadas durante o desenvolvimento do projeto:

- Separação do código e da estrutura em vários arquivos .c e .h para manter uma modularização.

- Fixação do tamanho da página em 4 KB e da memória física em 16 KB, conforme sugerido na especificação, para manter compatibilidade com os testes esperados.

- Registro dos acessos e page faults diretamente dentro da função acessar_memoria para manter controle centralizado.

- A sequência de acessos foi armazenada como um vetor bidimensional {pid, endereco}.

- Implementação de FIFO usando um vetor de tempos de carregamento (tempo_carga[]), pois era mais simples de controlar a ordem de chegada.

Limitações da Implementação

Apesar do sucesso na simulação dos algoritmos, a implementação possui algumas limitações:

- Número fixo de processos e acessos: A sequência de acessos à memória e o número de processos foram definidos manualmente no código, dificultando a realização de simulações com diferentes cenários de forma automática ou dinâmica.

- Tamanho de página e memória física fixos: O simulador considera um tamanho de página de 4 KB e memória física de 16 KB fixos, conforme a especificação. No entanto, não há suporte a outros tamanhos configuráveis via entrada do usuário.

- Interface básica: não há interface gráfica, o menu interativo não é tão complexo e permite apenas escolher entre dois algoritmos de substituição.

- Sem persistência de resultados ou logs externos: Os resultados da simulação são exibidos apenas no terminal e não são salvos em arquivos para análise posterior.

3. Análise Comparativa dos Algoritmos:

Comparação entre os algoritmos implementados

O algoritmo FIFO possui uma fila que armazena as páginas na ordem em que elas foram carregadas na memória. Quando uma página precisa ser substituída, o algoritmo remove aquela que está há mais tempo na memória, ou seja, a página que entrou primeiro. Essa simplicidade torna a implementação de FIFO direta e eficiente em termos de tempo de execução, mas seu comportamento é limitado pois não considera o uso recente da página. Consequentemente, páginas ainda ativas podem ser substituídas prematuramente, aumentando o número de faltas de página.

Por sua vez, o algoritmo LRU utiliza o princípio de localidade temporal, assumindo que páginas usadas recentemente têm maior probabilidade de serem reutilizadas em breve. Para isso, o LRU mantém informações sobre o momento em que cada página foi usada pela última vez e substitui a página com o maior tempo desde seu último uso. Essa abordagem exige estruturas adicionais para monitorar o uso recente, aumentando a complexidade de implementação e o custo computacional.

Análise dos resultados obtidos

Durante os testes realizados, o algoritmo LRU apresentou um desempenho superior ao FIFO na maioria dos casos, resultando em uma menor quantidade de page faults. Isso se deve ao fato de que o LRU leva em consideração o histórico de uso das páginas, mantendo na memória aquelas que foram acessadas mais recentemente, o que favorece padrões com localidade temporal. O FIFO, por sua vez, remove a página mais antiga, independentemente de sua frequência de uso, o que pode causar substituições desfavoráveis. No entanto, foi percebido que, em cenários com uma quantidade maior de acessos e padrões menos localizados, o desempenho dos dois algoritmos tende a se aproximar, com uma diferença pequena na taxa de page faults. Isso demonstra que, embora o LRU seja teoricamente mais eficiente, sua vantagem prática depende fortemente do padrão de acesso à memória.

Exemplo de simulação:

```
int acessos[][2] = {
        {1, 0},
        {1, 4096},
        {2, 0},
        {2, 4096},
        {1, 0},
        {3, 0},
        {1, 0}
 };
```

FIFO

```
======== ESTATÍSTICAS DA SIMULAÇÃO ========
Total de acessos à memória: 7
Total de page faults: 6
Taxa de page faults: 85.71%
```

LRU

```
======== ESTATÍSTICAS DA SIMULAÇÃO ========
Total de acessos à memória: 7
Total de page faults: 5
Taxa de page faults: 71.43%
```

4. Conclusões: Reflexões sobre o projeto e aprendizados

O desenvolvimento do simulador de paginação com os algoritmos FIFO e LRU permitiu uma compreensão do gerenciamento de memória em sistemas operacionais. Através da implementação prática, foi possível não apenas consolidar conceitos teóricos, mas também observar o impacto direto das estratégias de substituição no desempenho do sistema, evidenciado pela quantidade de faltas de página geradas.

A análise comparativa demonstrou que, embora o FIFO seja simples e eficiente em termos de implementação, sua abordagem pode causar desempenho inferior em padrões de acesso reais, principalmente aqueles que apresentam repetição e localidade temporal. Já o LRU, mesmo com maior complexidade, apresentou melhor desempenho na maioria dos casos, confirmando sua eficiência em preservar páginas relevantes com base no histórico recente de uso.

A experiência prática contribuiu para o aprimoramento das habilidades em programação estruturada em C, destacando a importância de modularização, manipulação de arrays e controle rigoroso de fluxo para a simulação dos algoritmos.

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/