Trabalho de Sistemas Operacionais - Algoritmos de Substituição de Página

Universidade Federal de Alfenas | Professor Fellipe Rey

Pedro Henrique Botelho da Silva - RA: 2023.1.08.027

Diogo da Silva Moreira - RA: 2023.1.08.003

1. OBJETIVO

O objetivo deste trabalho foi simular o gerenciamento de memória em um sistema operacional, implementando conceitos como alocação dinâmica, gerenciamento de bits de referência (R) e modificação (M), e também o algoritmo de substituição de página NUR (Not Recently Used). A ideia central foi criar um programa que permitisse a alocação e desalocação de processos na memória, além de acompanhar suas alterações ao longo do tempo. A simulação também foi projetada para mostrar como o sistema atualiza automaticamente os bits de referência e modificação, fornecendo uma visão mais clara sobre como a memória é gerenciada em um sistema real.

2. LINGUAGEM UTILIZADA

O programa foi desenvolvido em C, uma linguagem de baixo nível que oferece controle direto sobre a memória, o que facilita a simulação de como o gerenciamento de memória.

2.1. COMPILAÇÃO E EXECUÇÃO

Para compilar e executar o programa, siga os passos abaixo:

1. Salve o código em um arquivo chamado memoria.cpp.
2. Abra um terminal ou console no mesmo diretório do arquivo.
3. Compile o programa utilizando o seguinte comando:

```
gcc memoria.c -o memoria

./memoria
```

A implementação utiliza duas estruturas principais: **processos** e **blocos de memória**. Cada processo tem um identificador único, tamanho e dois bits (R para referência e M para modificação), além de manter o histórico de acessos. A memória é dividida em blocos que

podem ser livres ou ocupados por processos, com fusão de blocos adjacentes para otimizar o espaço.

3.1 Funções Principais

- **initializeMemory()**: Inicializa a memória com blocos.
- **printMemory()**: Exibe o estado da memória.
- **liberarProcesso()**: Libera processos e tenta fundir blocos.
- **resetRBits()**: Reseta os bits R dos processos.
- **updateMBits()**: Atualiza e reseta os bits M após 10 unidades de tempo.
- **aplicarNUR()**: Aplica o algoritmo NUR para substituir processos com base nos bits R e M.

3.2 Funcionalidades

- **Inserção de Processos**: Aloca processos ou aciona NUR para liberar espaço.
- **Exibição de Memória**: Mostra o estado atual da memória.
- **Avanço de Ciclos de Clock**: Reseta os bits R a cada 10 unidades de tempo.
- **Referência de Processo**: Atualiza os bits R e M quando um processo é acessado.

4. SAÍDA

O programa exibe o estado atual da memória, detalhando os blocos livres e ocupados, além dos processos alocados com seus respectivos bits **R** (referência) e **M** (modificação). A cada inserção ou remoção de processo, o estado da memória é atualizado, e o algoritmo **NUR** é acionado para liberar espaço, se necessário. A cada 10 unidades de tempo, o bit **R** de todos os processos é resetado, e o bit **M** é atualizado, caso o processo não tenha sido modificado. Quando um processo é referenciado, seus bits **R** e **M** são ajustados conforme a ação.

5. EXECUÇÃO

Pode-se executar o programa em qualquer ambiente que suporte C, basta seguir o passo a passo descrito na seção 2.1. O programa também pode ser adaptado para rodar em IDEs online que suportem a linguagem.