UNIVERSIDADE FEDERAL DE SANTA CATARINA CENTRO TECNOLÓGICO (CTC)

Professor Givani Gracioli
INE5412 – Sistemas Operacionais 1

SIMULAÇÃO DE ALGORITMOS DE SUBSTITUIÇÃO DE PÁGINAS

Ana Flávia Schwarz
Lívia Corazza Ferrão

Florianópolis - Santa Catarina
2023

UNIVERSIDADE FEDERAL DE SANTA
CATARINA CENTRO TECNOLÓGICO (CTC)

RESUMO

Neste trabalho, foi desenvolvido um programa de simulação para analisar o desempenho de algoritmos de substituição de páginas em sistemas operacionais modernos. A estrutura do código é organizada com base em princípios de orientação a objetos e segue um diagrama de classes que ilustra as relações entre as classes envolvidas na simulação.

O programa inclui a implementação dos algoritmos de substituição de páginas FIFO (First In, First Out), LRU (Least Recently Used) e OPT (Algoritmo Ótimo).
Foi desenvolvido um programa que recebe o número de quadros disponíveis na memória RAM e uma sequência de referências às páginas. A simulação é conduzida para cada algoritmo, registrando o número de falta de páginas obtidas em cada caso.

Cada algoritmo é encapsulado em classes separadas que herdam de uma classe abstrata chamada "AbstractPaging". A classe "Memory" representa a memória RAM e é responsável por gerenciar as páginas na memória, oferecendo métodos para adicionar, remover e verificar a presença de páginas. A classe "Page" representa uma página na memória.

A implementação da interface gráfica usando QT Creator foi realizada, mas resultou em vazamento de memória e limitações de desempenho em testes com muitas referências. Como resultado, duas versões do projeto foram entregues: uma com a interface gráfica e outra sem, esta última sendo a mais compatível com os requisitos do trabalho.

Foram realizados testes que demonstram o desempenho dos algoritmos em diferentes cenários. No geral, o algoritmo OPT demonstrou ser o mais eficiente na minimização de faltas de página, seguido pelo FIFO e LRU. O tempo de execução variou de acordo com o tamanho das sequências de referências, sendo rápido em testes curtos e aumentando significativamente em testes longos.

SUMÁRIO

INTRODUÇÃO

O gerenciamento eficiente da memória é uma preocupação crítica em sistemas operacionais modernos e ambientes computacionais. Uma parte essencial desse gerenciamento envolve a alocação e a substituição de páginas na memória RAM, visando otimizar o desempenho dos processos em execução. A escolha adequada do algoritmo de substituição de páginas desempenha um papel fundamental nesse contexto, impactando diretamente o número de faltas de páginas e, por conseguinte, a eficiência do sistema.

Este trabalho se concentra na simulação e análise de algoritmos de substituição de páginas, oferecendo uma compreensão mais profunda de como esses algoritmos funcionam e como eles se comportam em diferentes cenários de uso de memória. Três algoritmos específicos são considerados: FIFO (First In, First Out), LRU (Least Recently Used) e OPT (Algoritmo Ótimo).

O Algoritmo FIFO mantém uma fila de páginas na memória. Quando uma falta de página ocorre, o algoritmo remove a página mais antiga da fila e é inserida a nova no final da fila. Este algoritmo tem fácil implementação, mas por não considerar quais páginas são mais frequentemente usadas pode não ser muito eficiente em diversos casos.

Já no LRU, os registros são mantidos na ordem que as páginas foram acessadas pela última vez. Ao ocorrer um a falta de página, é escolhido para substituição a página que esta a mais tempo sem ser usada, marcando a página recém-solicitada como a mais recente. O LRU é eficiente na maioria dos cenários, mas sua implementação pode ser custosa, pois requer o rastreamento da ordem de acesso das páginas.

Por fim, o algoritmo Ótimo é uma abordagem teórica que tem como objetivo minimizar as faltas de páginas ao realizar a busca com base no conhecimento antecipado das futuras referências.

A motivação por trás deste estudo é a necessidade de avaliar o desempenho desses algoritmos em ambientes reais e compreender como eles respondem a variações no número de quadros disponíveis na memória RAM. Essa compreensão é crucial para a seleção apropriada do algoritmo de substituição de páginas em sistemas operacionais e aplicações que dependem da gestão eficiente da memória. Os resultados obtidos por meio dessas simulações possibilitarão uma análise comparativa detalhada, destacando os pontos fortes e as limitações de cada algoritmo.

DESENVOLVIMENTO

2.1. ESTRUTURA DO CÓDIGO

A estrutura do código completo é organizada seguindo os princípios da orientação de objetos. O diagrama apresentado na Figura 1 ilustra as relações entre as classes envolvidas na simulação, destacando a hierarquia e a colaboração entre elas.

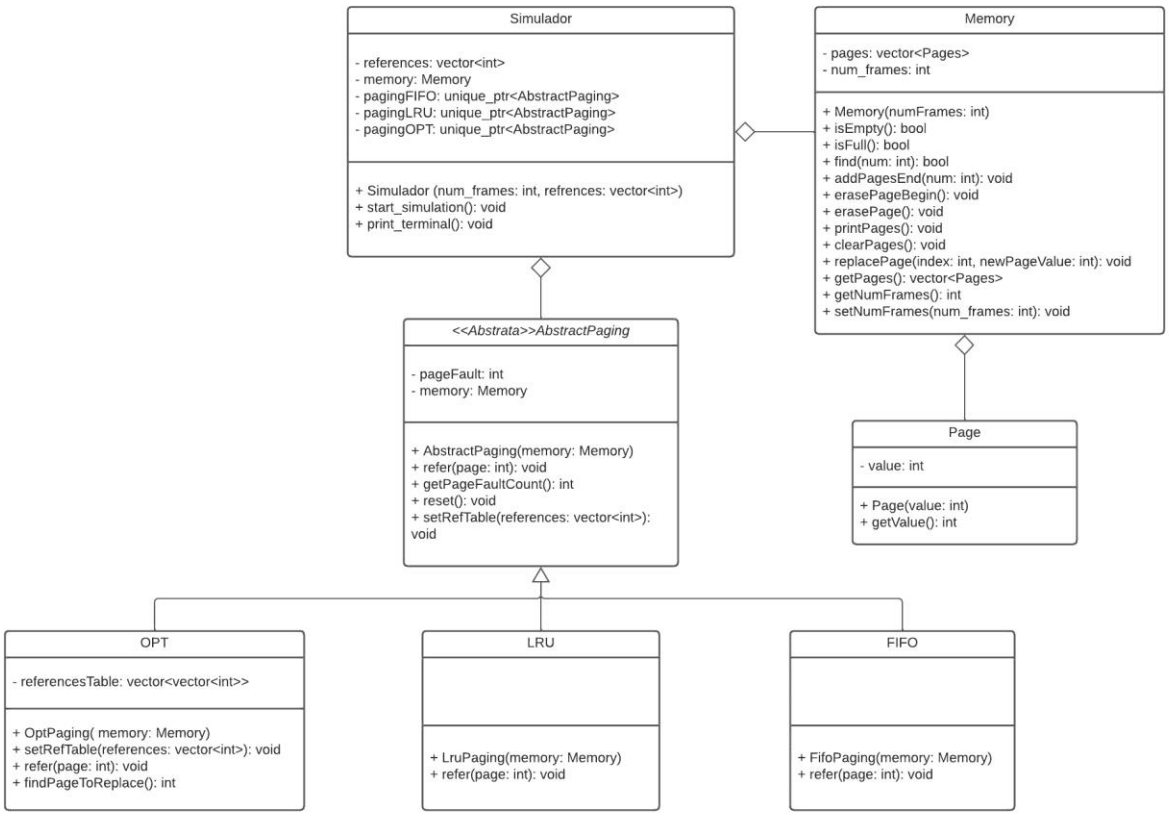

Figura 1: Diagrama de classes

O “main” recebe a entrada do usuário e inicia a simulação, interagindo com a classe “Simulador”. A classe “Simulador” é responsável por coordenar a simulação, cria instâncias da classe “Memory” e das classes de algoritmos de substituição de páginas, como FIFO, LRU e OPT. O “Simulador” também inicia a simulação e coleta os resultados para análise.

Os algoritmos usam a classe “Memory” para gerenciar as páginas na memória e tomar decisões de substituição. “Memory” é a classe que representa a memória RAM. Ela mantém uma lista de páginas na memória e fornece métodos para adicionar, remover e verificar a presença de páginas na memória. Já a classe “Page” representa uma página na memória. Cada instância contém um valor que a identifica, e é usada para criar objetos que representam as páginas dentro da memória.

Todas as classes de algoritmos (FIFO, LRU e OPT) herdam da classe abstrata “AbstractPaging”, a qual desempenha um papel central, fornecendo métodos comuns entre eles. Cada algoritmo implementa os métodos de acordo com suas regras.

2.2. IMPLEMENTAÇÃO DOS ALGORITMOS

A lógica do FIFO é baseada em uma fila simples. Quando ocorre uma falta de página, a página mais antiga na memória é escolhida para substituição. Para isso, implementamos a classe que mantém um registro da ordem em que as páginas foram adicionadas à memória. O método *refer* da classe FIFO adiciona páginas no final da fila e, quando a memória está cheia, remove a página mais antiga do início da fila para liberar espaço.

O algoritmo LRU foi implementado na classe LRU, que herda da classe abstrata “AbstractPaging”. Sua lógica é baseada no princípio de que a página menos recentemente usada deve ser escolhida para substituição. Para implementar isso, a classe mantém um registro da ordem em que as páginas foram acessadas. Quando ocorre uma falta de página, a página menos recentemente usada é escolhida para substituição. O método refer da classe LRU reorganiza a ordem das páginas na memória, movendo a página acessada mais recentemente para o final da lista.

A lógica realizada para implementar o algoritmo do OPT é baseada em uma estratégia ideal em que a página a ser substituída é aquela que não será referenciada no futuro próximo. Para isso, foi criada a função “setRefTable”, responsável por rastrear a próxima ocorrência de cada página na lista de referências. Antes de iniciar a alocação na memória, a função gera uma matriz que registra as distâncias (posições no vetor de referências) até a próxima referência de cada página.

A função refer é chamada sempre que uma página é referenciada pelo processo. Ela verifica se a página já está na memória física (RAM). Se a página não estiver na memória, ocorre uma falta de página (page fault) e o algoritmo precisa decidir qual página na memória física deve ser substituída pela nova página. O algoritmo busca a página que terá a referência mais distante pela função findPageToReplace, que consulta a estrutura de referencesTable para determinar quando a página será referenciada novamente.

2.2. INTERFACE GRÁFICA

Foi implementada uma interface gráfica com QT Creator como parte do projeto, visando proporcionar uma experiência mais amigável ao usuário. No entanto, a interface gráfica resultou em um notável vazamento de memória, impactando negativamente o desempenho do programa. Além disso, a interface não responde a teste com muitas referências.

Como resultado, optamos por criar o diagrama de classes sem a inclusão da interface gráfica. Isso nos permitiu focar na implementação dos algoritmos de substituição de páginas e na análise de seu desempenho sem as complexidades adicionais introduzidas pela interface. O código-fonte completo, incluindo a implementação da interface gráfica, está disponível na entrega do projeto.

O código sem interface gráfica não apresenta vazamento de memória o e resultado final está compatível com o requerido no trabalho. Assim, entregamos duas versões do projeto, com e sem a interface.

RESULTADOS

Foram realizados três testes com número fixo de 4 quadros na memória RAM. Abaixo estão os resultados obtidos para cada teste e o tempo de execução de cada um.

```
4 quadros
24 refs
FIFO: 12 PFs
LRU: 11 PFs
OPT: 9 PFs
Tempo de execução: 0 milliseconds
```

Figura 2: simulação vsim-exemplo.txt

```
4 quadros
30 refs
FIFO: 22 PFs
LRU: 24 PFs
OPT: 14 PFs
Tempo de execução: 0 milliseconds
```

Figura 3: simulação vsim-belady.txt

```
4 quadros
1000000 refs
FIFO: 227915 PFs
LRU: 179986 PFs
OPT: 136050 PFs
Tempo de execução: 1992 millisecond
```

Figura 4: simulação vsim-gcc.txt

Na simulação com interface gráfica, os resultados do vsim-exemplo.txt e vsim-belady.txt geraram os mesmos valores. No entanto, no teste vsim-gcc.txt, a implementação não apresenta os resultados, devido ao grande número de referências.

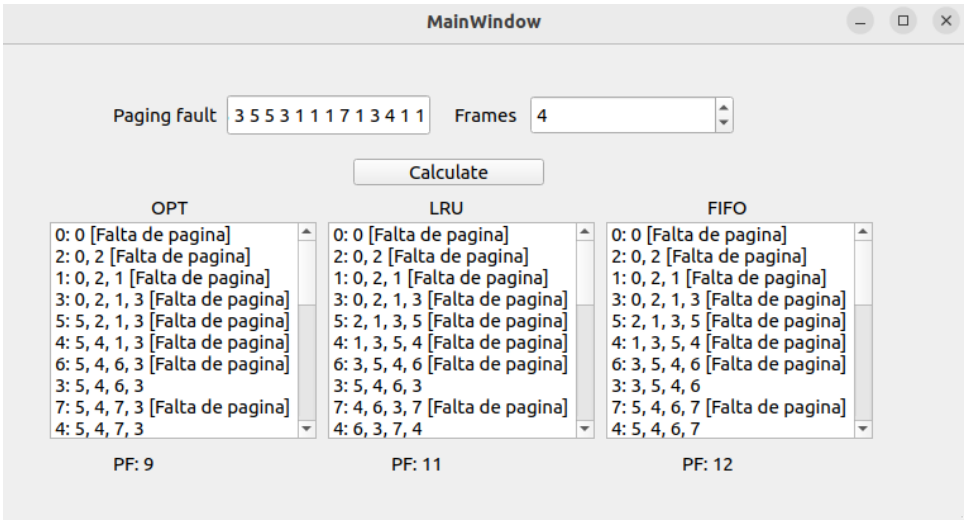

Figura 5: simulação vsim-exemplo.txt com interface gráfica

De acordo com a entrada do terminal, a interface lê o arquivo e imprime na tela cada passo do algoritmo. Também pode-se trocar os parâmetros inicializados na parte superior da tela.

Nesses testes, observou-se que o algoritmo OPT (Algoritmo Ótimo) geralmente obteve o melhor desempenho, resultando em menos faltas de página em comparação com FIFO e LRU. O algoritmo LRU tende a ter um desempenho intermediário, enquanto o FIFO teve um desempenho um pouco menos eficiente, especialmente em cenários com sequências de referências mais longas. Vale ressaltar que o tempo de execução foi muito baixo para os testes mais curtos, indicando uma simulação rápida. No entanto, em cenários de teste extremamente longos, como o terceiro teste, o tempo de execução aumentou significativamente devido à grande quantidade de referências a serem processadas. Estes resultados demonstram a importância de escolher o algoritmo de substituição de páginas apropriado, dependendo do cenário de uso da memória em sistemas operacionais.

CONCLUSÃO

Este projeto apresentou a implementação de algoritmos de substituição de páginas e a geração de resultados de simulação para análise de desempenho, proporcionando uma compreensão aprofundada desses importantes componentes de sistemas operacionais. Além disso, a estrutura do código, com a classe AbstractPaging, facilita a adição de novos algoritmos para futuras análises.

Os resultados dos testes destacaram a relevância da escolha do algoritmo apropriado com base no padrão de acesso à memória. O algoritmo OPT, que opera idealmente com conhecimento completo das futuras referências, mostrou ser o mais eficiente na minimização de faltas de página. O FIFO e o LRU, por outro lado, oferecem abordagens práticas para situações com previsões limitadas.

O trabalho proporcionou uma compreensão aprofundada dos algoritmos de substituição de páginas e sua importância na otimização do uso de memória em sistemas operacionais