Trabalho 2

Clara Rosa Oliveira Gonçalves, matrícula 22103511

Maykon Marcos Júnior, matrícula 22102199

Professor Giovani Gracioli

23 de outubro de 2023

Estrutura

INE5412: Sistemas Operacionais 1
Clara Rosa Oliveira Gonçalves, matrícula 22103511
Maykon Marcos Júnior, matrícula 22102199

Resumo

O trabalho 2 traz a resolução da problemática de simulação de algoritmos de escalonamento, proposta na disciplina. Nesse sentido, o relatório apresentado aborda os pontos envolvidos na proposta do trabalho, elaboração da solução, dificuldades enfrentadas, detalhamento do código produzido, avaliação de desempenho e referências bibliográficas. Assim, é possível acompanhar todos os passos envolvidos em seu desenvolvimento e compreender melhor os materiais produzidos.

Clara Rosa Oliveira Gonçalves, matrícula 22103511
Maykon Marcos Júnior, matrícula 22102199

Apresentação do problema

O problema proposto pelo trabalho 2 da disciplina consiste na elaboração de um algoritmo em C++ capaz de simular os algoritmos de substituição de páginas FIFO (first in, first out), LRU (least recently used) e OPT (optimal). Sabe-se que a disponibilidade de memória em um computador é uma problemática complexa: não é viável obter um espaço infinito de armazenamento, e ainda associado a velocidades de acesso à memória exorbitantes. Dessa forma, os algoritmos de substituição de páginas apresentam um fator fundamental na ampliação da capacidade de trazer dados para a memória principal do computador, consideravelmente mais rápido, e também transferi-los para a memória secundária, de forma a manter dados relevantes na memória principal de forma rápida e eficiente.

Clara Rosa Oliveira Gonçalves, matrícula 22103511
Maykon Marcos Júnior, matrícula 22102199

Diagrama de classes

A seguir, é apresentado o diagrama de classes produzido para o desenvolvimento do trabalho:

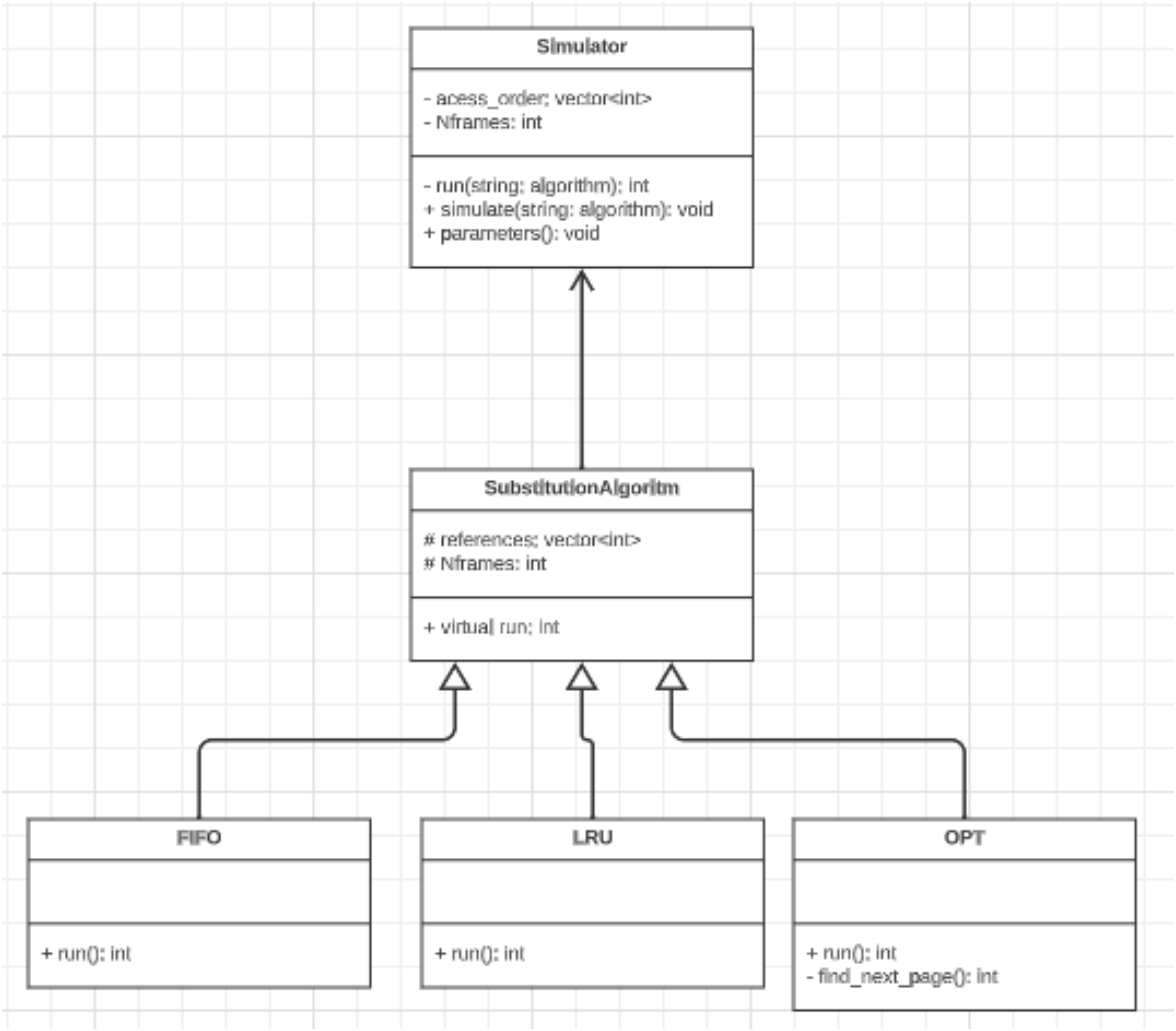

Imagem 1 - diagrama de classes

Também é possível acessar o relatório pelo Moodle da disciplina na entrega do trabalho 2, bem como pelo link a seguir:

https://lucid.app/lucidchart/10c834eb-a586-43a5-8009-b0a13e91a5b7/edit?viewport_loc=-132%

2C44%2C2380%2C1004%2CHWEp-vi-RSFO&invitationId=inv_52f46508-ddc1-472e-9536-7469b9c4efff .

As classes são modeladas da seguinte forma:

- **Simulator:** classe responsável por fazer a leitura do arquivo de referências por entrada padrão (std::cin) no construtor, registrar a ordem de acessos no vetor de páginas “access_order”, armazenar o número de frames recebido por parâmetro e criar instâncias dos algoritmos implementados (método run). O método “parameters” imprime o número de quadros e a quantidade de referências feitas e o método “simulate“ executa o método run dos algoritmos e imprime as estatísticas finais;
- **SubstitutionAlgorithm:** classe abstrata que modela os algoritmos, instanciada no método run do Simulator. Possui como atributos o vetor de referências às páginas e o número de frames, e a função virtual run;
- **FIFO:** A estrutura frame_queue é uma fila que representa os quadros (frames) disponíveis na memória. Ele é usado para manter o controle da ordem de chegada das páginas. Page_table é um conjunto que mantém o controle das páginas que estão na memória. Quando ocorre uma page fault, caso frame_queue esteja cheio (não há frames disponíveis para alocação), a página mais antiga é removida;
- **LRU:** quando ocorre um page fault, a página mais antiga é removida e uma nova é adicionada no começo da page_list. Caso contrário, quando um acesso é feito a uma página já pertencente ao frame, ele é movido para a cabeça da lista;
- **OPT:** no algoritmo OPT, o método find_next_use percorre o vetor de referências até encontrar a posição em que a página será referenciada novamente. O método

run percorre todas as referências a páginas. Caso ocorra page fault e haja frames disponíveis, a página é adicionada. Caso contrário, um loop percorre cada página nos frames, calcula seu próximo uso e substitui a página que será referenciada no momento mais tarde.

Clara Rosa Oliveira Gonçalves, matrícula 22103511
Maykon Marcos Júnior, matrícula 22102199

Desenvolvimento

A implementação foi dividida entre header files (com a declaração das classes) e arquivos cpp (com a implementação de fato), o que torna a recompilação do código mais eficiente em casos de alterações em implementações, traz mais clareza na leitura do código e auxilia a garantir o princípio de encapsulamento. Também foram utilizadas as melhores práticas de orientação a objetos, comentários claros e nomenclaturas de variáveis objetivas e formatação ideal.

A interface foi desenvolvida com base na biblioteca SFML, e exibe na tela cada ação feita por acesso a uma página, bem com um quadrado, que fica vermelho a cada page fault, e verde a cada page hit. A estrutura do código original foi alterada de forma a devolver um vetor, o qual possui em cada posição par a página a ser substituída e ímpar a nova página. Esses valores são obtidos no novo arquivo main e atualizados pelo método presente no arquivo GameClass.

Clara Rosa Oliveira Gonçalves, matrícula 22103511
Maykon Marcos Júnior, matrícula 22102199

Dificuldades encontradas

Uma das dificuldades encontradas durante a elaboração do trabalho foi a criação do diagrama de classes, devido à extensa gama de possibilidades de organização do fluxo de execução. Além disso, a implementação do Algoritmo Ótimo foi a mais complexa.

Avaliação de desempenho

Percebe-se, pela execução dos arquivos referenciados, que a melhor eficiência foi obtida, em termos de page hits, pelo algoritmo OPT, seguido pelo FIFO e LRU.

Para a entrada vsim-gcc.txt, com 4 page frames:

- Tempo médio para (apenas) FIFO: real: 0m0.304s ;; user: 0m0.301s ;; sys: 0m0.003s
- Tempo médio para (apenas) LRU: real: 0m0.318s ;; user: 0m0.296s ;; sys: 0m0.022s
- Tempo médio para (apenas) OPT: real: 0m0.511s ;; user: 0m0.498s ;; sys: 0m0.009s
- Tempo médio para executar os 3: real: 0m0.628 ;; user: 0m0.615s ;; sys: 0m0.014s
- (Naturalmente afetado pelo uso de polimorfismo (exigindo alocação dinâmica) e pelas operações E/S, pois os 3 tem prints separados e independentes)

O programa foi testado e encontrado funcional para:

- IDE: Visual Studio Code
- Especificações do dispositivo:

- Sistema Operacional: Linux-gnu x86_64, Ubuntu 20.04
- Compilador: g++ (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
- Processador: Intel Core i5, 8GB de RAM, 14 Cores.

Clara Rosa Oliveira Gonçalves, matrícula 22103511
Maykon Marcos Júnior, matrícula 22102199

Conclusões

Em suma, percebe-se que o algoritmo FIFO, na maioria dos casos, resultou em mais page faults do que o LRU. Como esperado, o algoritmo OPT demonstrou ser o mais eficiente nesse quesito, apesar de sua implementação na forma original não ser possível devido à incapacidade de se saber previamente a ordem de acessos a páginas. Dessa forma, a escolha do algoritmo de substituição de páginas dependerá das características do sistema e das prioridades de otimização.

Enquanto o FIFO é simples de implementar e pode ser adequado para cenários de baixa complexidade, o LRU tende a apresentar melhor desempenho em sistemas onde a minimização das page faults é uma prioridade. Em casos em que se pode fazer estimativas razoáveis sobre os acessos futuros, o algoritmo OPT pode ser a opção ideal, mesmo que sua implementação prática envolva desafios adicionais. Em última análise, a escolha do algoritmo dependerá das necessidades específicas de cada aplicação e do equilíbrio entre complexidade e desempenho desejado.

Adicionalmente, elaborar algoritmos análogos ao OPT, tentando sempre se aproximar dele ao máximo, é uma solução viável para implementações que exijam maior eficiência. Assim, pesar de mais complexo de se desenvolver, a solução encontrada apresentaria uma melhor taxa de page hits se comparada com outras políticas de substituição de páginas.

Referências

- Materiais da disciplina.