UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO - CTC
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Atividade 2

Professores
Giovani Gracioli
Márcio Castro

Alunos
Augusto de Hollanda Vieira Guerner (22102192)
Micael Angelo Sabadin Presotto (22104063)

Florianópolis 2023/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO - CTC
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

1 Introdução

Este documento é uma descrição do desenvolvimento do segundo trabalho da matéria de Sistemas Operacionais 1. O tema é sobre simulação de algoritmos de substituição de páginas. O grupo deverá simular 3 dos algoritmos vistos em aula: FIFO (First In, First Out); OPT (algoritmo ótimo); e LRU (Least Recently Used). Além de procurar a corretude da execução, o grupo também deverá se preocupar com o desempenho dos algoritmos, isto é, fazer a simulação mais rápida possível.

Por fim, vale comentar que o relatório respeita a seguinte estrutura: primeira se comenta sobre o simulador (classe); em seguida, fala-se sobre como foi implementado as páginas; depois como os algoritmos foram idealizados conforme as boas práticas de OO; e, por último, é discutido sobre os resultados tanto os de desempenho quanto os de corretude das implementações dos algoritmos, ou seja, se os valores mostrados no console fazem sentido com a teoria vista em aula.

2 Desenvolvimento

Antes de prosseguir com a descrição do desenvolvimento, vale destacar que todas implementações foram pensadas para otimizar o tempo de execução do programa, isto é, fazer com que a simulação seja a mais rápida possível. Assim, muitas vezes foi priorizada a simplicidade, para não adicionar processamento desnecessário e otimizar o desempenho.

2.1 Simulador

Nesta seção será discutido sobre a classe Simulator. É falado para que ela serve e como ela funciona.

Primeiramente, no que se refere para que serve a classe, ela é responsável apenas pelo fluxo de execução do programa. Em alto nível, ela lê o arquivo de entrada e executa os algoritmos a serem simulados, ou seja, setup e simulação.

Já no que tange ao seu funcionamento, é tão simples quanto. Começa-se primeiro com a validação do argumento de linha de comando, denominado no código por frames ou número de paǵinas dentro da RAM. Se ele não for válido, o programa encerrará com erro e mostrará no console uma breve descrição de como usá-lo. Depois, se o parâmetro é válido, é feito a leitura do arquivo de referências, armazenando seu conteúdo em um vetor para ser utilizado na simulação de fato. Depois é instanciado os algoritmos a serem simulados. Seguidamente é feita a simulação percorrendo as linhas do arquivo por meio do vetor criado e enviando cada uma delas para cada um dos algoritmos. Por fim, é mostrado na tela o resultado da simulação, os misses de cada algoritmo e o número de páginas.

2.2 Páginas

Esta seção comenta sobre como a modelagem das páginas foi pensada. Destaque-se primeiro sobre o uso de classe para isso. Em seguida, comenta-se como foi de fato implementado no trabalho.

No começo, foi utilizado classe para a modelagem das páginas, denominada de Page. No entanto, depois que finalizado o trabalho, percebeu-se que era uma complexidade desnecessária para o presente problema tanto no quesito tempo computacional quanto no quesito de legibilidade. Tempo porque o gerenciamento de classes em C++ é custoso, de tal forma que o programa poderia ser mais rápido se não usasse (e é). Legibilidade pois o programa torna-se complexo desnecessariamente, uma vez que é possível fazer sem, como foi o caso, sem perder significativamente a legibilidade.

Tendo em visto o que foi comentado anteriormente, preferiu-se, então, trocar a classe criada pelo tipo size_t oferecido pela lib do C++. Isso não foi apenas mero capricho do grupo. Na verdade, o size_t é o ideal para o trabalho por dois motivos. O primeiro é pela sua simplicidade, pois, além de ser menos custoso para o computador gerenciá-lo, também é mais simples trabalhar do que uma classe. O segundo é que páginas não assumem valores negativos, assim otimiza-se os recursos. De fato, no final, percebeu-se que o código tornou-se muito mais enxuto e legível, bem como muito mais rápido, cerca de 7x.

A partir do supracitado, nota-se que o uso de páginas como o tipo size_t é muito melhor do que usando classe, tanto no sentido de tempo quanto no sentido de legibilidade. Obviamente, em programas mais complexos o uso de página como uma classe ou uma struct é necessária. Com efeito, uma página em programas reais de SÓ possuem muitas propriedades não utilizadas no simulador desta atividade de modo que é imprescindível uma abstração mais complexa que o size_t.

2.3 Algoritmos

No tópico presente é falado sobre a implementação dos algoritmos. Precisamente, foram feitas 4 classes para eles. A primeira classe na verdade é abstrata e as outras 3 são as concretas, uma para cada.

No que se refere a abstrata, ela é quem define a estrutura das demais. Nela tem-se um método realmente importante, o accessMemory. Vale destacar que este método é totalmente abstrato, cada uma das classes concretas terão que implementá-lo. Os demais métodos são apenas getters ou construtores/destrutores.

As classes concretas são 3: uma para o FIFO; outra para o LRU; e uma última para o OPT. A classe para o FIFO e para o LRU no geral são bem similares com uma pequena

mudança apenas no método accessMemory. O FIFO atualiza a lista apenas quando há um miss. Se tiver, ele exclui o primeiro elemento da fila e coloca a nova página no final. Se ocorrer um hit, a fila continua a mesma e o algoritmo não faz nada. Para o LRU tem-se o mesmo tratamento para misses, no entanto um diferente para hits. Quando a página se encontra na memória (hit), ela é colocada no final da fila.

Diferentemente do FIFO e do LRU, o algoritmo OPT precisa saber do futuro. Consequentemente, sua implementação torna-se mais complexa e custosa. O algoritmo primeiro verifica se ocorreu um hit, se sim, ele retorna sem fazer nada. Se não houver hit, então verifica-se se a memória está cheia. Se não tiver, ele apenas faz um push da página em que ocorreu o miss. Se tiver cheia, ele verifica, com o uso de um for, a página que tem a próxima referência mais longe ou que não tem mais referência (por isso saber do futuro). Quando ele determinar qual das páginas será referência por último ou não será mais referenciada, o algoritmo a apaga e insere a nova no fim da lista.

No mais, vale destacar que quando ocorre um miss é retornado 0 pelo método accessMemory de cada um dos algoritmos. Assim, para fazer a contagem de quantos houveram ao longo da execução, soma-se o retorno de cada chamada.

Nota-se, com exposto, que, além de implementação respeitar os fundamentos de OO, tem-se uma modelagem simples e extensível. Vê-se que facilmente se pode criar novos algoritmos de substituição de página estendendo da classe abstrata criada. O grupo tem a consciencia de que para programas reais essa modelagem certamente seria insuficiente. Contudo, para o presente projeto ela é mais que isso, ela é eficiente e simples.

2.4 Resultados

Agora serão discutidos e analisados os resultados, tanto no que concerne aos de desempenho quanto no que concerne a saída no console.

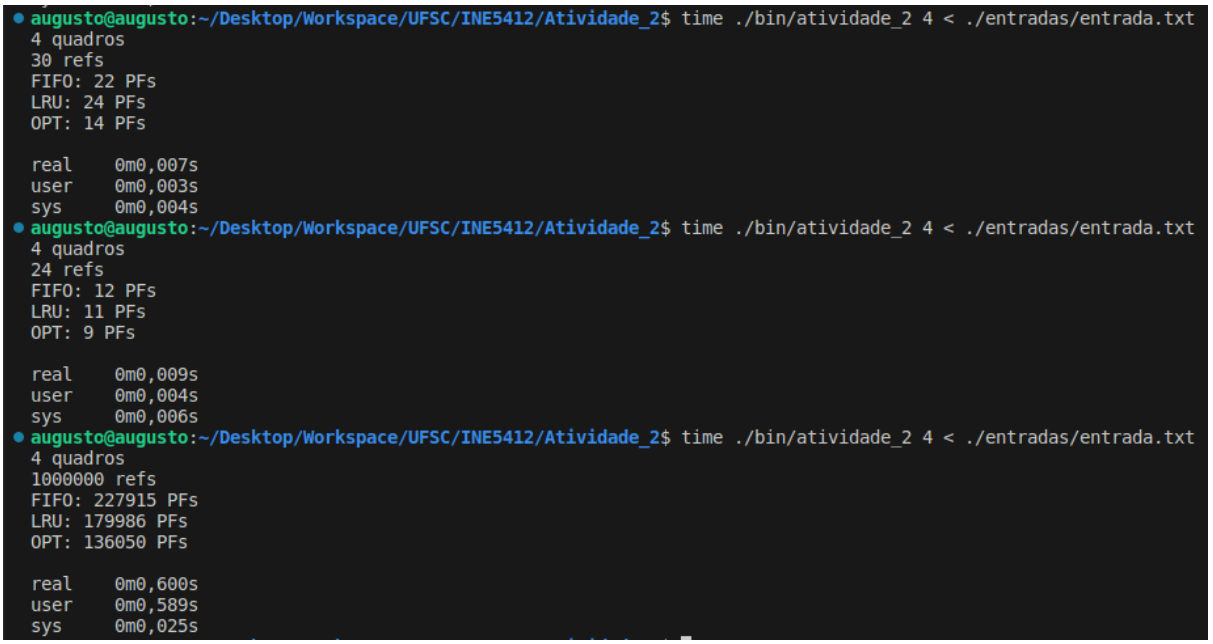
No que tange ao desempenho, o programa se demonstrou muito eficiente, atingindo marcas inesperadas. Com a entrada maior, de 1 milhão de referências, e com 4 quadros, o programa na grande maioria dos computadores testados, não passava de 1 segundo de execução. As outras entradas permaneceram na marca de 10 milissegundos para a mesma quantidade de quadros. Boa parte do bom desempenho do programa foi devido às otimizações de lógica e otimizações do próprio compilador usando -O3. Além disso, apesar de ter o uso de threads, pouco se notou a diferença.

Agora referentemente aos resultados, obteve-se o esperado. Para os dois arquivos menores, de 30 e 24 referências, foi feito à mão cada algoritmo para garantir a corretude do programa. De fato, chegou-se ao mesmo resultado. Para o arquivo maior, o de 1 milhão de referências, por questões óbvias, não foi verificado a corretude manualmente, mas apenas

para uma certa parcela do arquivo (bem pequena comparado a ele inteiro). No mais, vale destacar que foi testado com outros valores de quadros e obteve-se resultados esperados, ao menos para aqueles que foram verificados manualmente.

Na imagem 1, vê-se o que foi falado nos parágrafos acima.

Imagem 1 - Resultados para cada arquivo exemplo.



Fonte: Acervo dos autores.

Com o que foi falado nesta seção, nota-se que o desempenho do programa foi bom tanto em matéria de resultado quanto em matéria de complexidade de tempo. Com efeito, o desempenho do código foi muito melhor do que esperado e do que inicialmente, nas primeiras versões do código, obteve-se (na primeira versão atingimos a incrivel marca de cerca de 800 segundos). A corretude também não ficou para trás. Para todos os casos feitos manualmente, o programa retornou o mesmo resultado.

3 Conclusão

Com o presente relatório, aprendeu-se muito tanto sobre os algoritmos de substituição, precisamente o FIFO, LRU e OPT, quanto sobre a importância da conciliação de OO e desempenho. Além disso, a atividade contribui também com aprimoramento dos conceitos aprendidos em aulas e com o desenvolvimento do raciocínio crítico, muito devido principalmente às discussões criadas ao longo do desenvolvimento sobre que decisões tomar. Outro valor agregado pela atividade, mas não de menor importância, foi o conhecimento de

como trabalhar em equipe. Portanto, novamente o trabalho foi uma fonte de aprendizados tanto para as soft skills quanto para as hard skills.