Trabalho Prático #3

Simulação de um sistema de memória virtual

Marco Túlio Motta de Jesus <marcotmotta@gmail.com>

Rafael Augusto Botelho Perez <rafael.perez@dcc.ufmg.br>

Termo de compromisso

Os membros do grupo afirmam que todo o código desenvolvido para este trabalho é de autoria própria. Exceto pelo material listado no item 3 deste relatório, os membros do grupo afirmam não ter copiado material da Internet nem obtiveram código de terceiros.

Membros do grupo e alocação de esforço

- Marco Túlio Motta de Jesus <marcotmotta@gmail.com> 50%
- Rafael Augusto Botelho Perez <rafael.perez@dcc.ufmg.br> 50%

Referências bibliográficas

- A. Sılberschatz, P. Galvin, G. Gagne. Sistemas Operacionais. Campus, 2004.
- A. S. Tanenbaum. Sistemas Operacionais Modernos. Pearson, 2010.
- B. L. Stuart. Princípios de Sistemas Operacionais. Cengage Learning, 2011.

Resumo

1. Estrutura geral

Conforme a especificação, implementamos a entrada com quatro argumentos por padrão e um quinto opcional, para debug. Exemplo de uso:

```
tp3virtual lru arquivo.log 4 128
```

Para acessar o modo debug, basta incluir o quinto argumento com o valor inteiro 1:

```
tp3virtual lru arquivo.log 4 128 1
```

Caso deseje fazer debug somente leitura, insira a opção 2:

```
tp3virtual lru arquivo.log 4 128 2
```

Caso seja somente escrita, opção 3:

```
tp3virtual lru arquivo.log 4 128 3
```

Logo na entrada dos argumentos é feita uma validação que impede que o programa execute sem os argumentos corretos, retornando uma mensagem de erro informativa e guiando o usuário para o uso correto da aplicação.

Adotamos as indicações de faixas de valores dos itens 3 e 4 de “Forma de operação” da especificação em nosso programa. Para o formato de saída, tomamos por base o exemplo de saída dado em “Formato de saída” da especificação, e adicionamos alguns campos que achamos pertinentes: *Total de acessos, Operações de leitura, Operações de escrita, Hits, Misses* e *Writebacks*.

Criamos estruturas de dados auxiliares para armazenar as entradas e saídas do programa, do tipo registro: `struct argumentos` e `struct estatisticas`. Com

elas organizamos o fluxo de informação ao longo do programa de forma modularizada.

2. Estruturas importantes

2.1. Implementação da tabela de páginas

Utilizamos uma estrutura de lista encadeada para armazenar os dados das páginas. Outra opção válida seria implementar na forma de vetor, entretanto, lista encadeada é mais eficiente e, portanto, vantajosa, quando utiliza-se algoritmos que envolvem noções temporais, tais como `lru` e `fifo`, pois possibilita um fácil rearranjo entre as páginas; todavia, torna-se onerosa com o uso de outros algoritmos, tal qual o `random`, que necessita percorrer a lista em busca do índice aleatoriamente selecionado.

2.2. Implementação dos algoritmos de reposição

Implementamos os algoritmos `lru`, `fifo` e `random`. Não obtivemos sucesso em nossos testes com o `2a` - segunda reposição - e por isso optamos por descartá-lo.

3. Decisões de projeto

3.1. Ambiguidades na especificação e linhas sem comandos

A partir da leitura da especificação, a tabela de páginas foi tópico de dúvidas na fase inicial da implementação devido à forma com que o tópico “Implementação da tabela de páginas” foi escrito. Conforme explicado anteriormente, optou-se por implementar uma lista encadeada.

Análise de desempenho

Em linhas gerais, observou-se que o algoritmo `random` obteve desempenho notavelmente inferior aos outros algoritmos implementados para cenários com entrada suficientemente grande pois, nesse algoritmo, a taxa de acerto foi bem baixa. Outro fator que pesa negativamente para o tempo de execução durante a utilização do `random` é que, em diversos momentos, ele precisa percorrer porções grandes da lista em busca do índice selecionado, tomando mais tempo. Para os algoritmos `lru` e `fifo`, entretanto, não se faz presente tamanha sobrecarga de tempo devido à natureza da estrutura de dados.