RELATÓRIO DO SISTEMA COMPUTACIONAL (4-1-1.8)

"ALGORITMO DO BANQUEIRO"

Lucas João Martins

1 - REQUISITOS SOLICITADOS

- RF01. Implementar o algoritmo solicitado, seguindo a estrutura de classes apresentada no VPL do Moodle;
- RF02. Coletar estatísticas sobre o algoritmo, como tempo médio gasto, quantidade de pedidos de alocação aprovados ou reprovados, quantidade de estados seguros e inseguros, etc;
- RF03. Implementar testes unitários para cada desenvolvimento feito;
- RNF01. O sistema deve executar no VPL do Moodle.

2 - RF01

O algoritmo foi implementado conforme a estrutura de classes do VPL/código disponibilizado no GitHub na nova classe Banker.cpp (https://goo.gl/LXg4Np) que possui a implementação do algoritmo no método Banker::algorithm. A implementação foi baseada na explicação do algoritmo que o livro do Tanenbaum (Modern Operating Systems 4th, seção 6.5.4) fornece.

3 - RF02

As estatísticas solicitadas são impressas no terminal ao fim da execução em modo normal. É apresentado uma porcentagem que indica quantos pedidos dos quinze realizadas serão aprovadas/reprovadas e quantas gerarão estados seguros/inseguros. Esse valor não muda, já que as solicitações não são dinâmicas e sempre são as mesmas. O tempo médio de execução é calculado da seguinte forma: todo o tempo gasto com a simulação dividido por quinze, que é a quantidade de pedidos realizados.

4 - RF03

Onze testes unitários foram desenvolvidos na classe ProblemTester.cpp (https://goo.gl/UtBrdW). Todos os métodos da nova classe Banker foram testados ao menos

duas vezes, com exceção dos ‘getters’ que não possuem testes por causa da sua simplicidade. Para a execução em modo teste, precisa-se descomentar a linha 52 em main.cpp (https://goo.gl/r8K7G2) e comentar qualquer chamada ao método Banker::printHelperDebug para que a execução dos testes não quebre (essa última necessidade já vai pronta no código submetido). Além disso, para uma melhor visualização no terminal, recomenda-se alterar para ‘false’ os atributos ‘trace’ e ‘info’ em Traits.h (https://goo.gl/Bpmkcl).

5 - RNF01

Considerando que o VPL do Moodle é o mesmo que o código disponibilizado pelo professor no GitHub, então o sistema computacional executa no VPL. Aqui aproveito para comentar sobre alguns aspectos do desenvolvimento realizado:

- No ‘switch case’ do OperatingSystem::ExecuteTestCode o esperado era que todas as chamadas fossem separadas em diversos cases, para que assim elas fossem executadas conforme o ‘executionStep’ aumentasse. Contudo, ao implementar dessa maneira a simulação não funcionava corretamente, porque em cada chamada do ‘switch case’ com um ‘executionStep’ diferente a variável ‘banker’ era inicializada do zero novamente e com isso os antigos valores setados em ‘cases’ anteriores eram perdidos. Por exemplo, o que uma chamada de método realiza quando ‘executionStep’ é igual a zero não persiste quando se realiza outra chamada de método quando ‘executionStep’ é igual a um.
- Na execução normal são realizada quinze solicitações de recursos em OperatingSystem::ExecuteTestCode. As doze primeiras solicitações e a montagem do cenário de simulação que é realizada antes das solicitações são feitos para simular a figura 6-12 do livro do Tanenbaum (Modern Operating Systems 4th, páginas 454 e 455). Vale notar que os valores das outras três solicitações foram definidos por mim. Finalmente, por ter se baseado no exemplo do Tanenbaum, pode-se fazer a seguinte associação nesse sistema computacional:
    - Processo 1 = Processo A;
    - Processo 2 = Processo B;
    - Processo 3 = Processo C;
    - Processo 4 = Processo D;
    - Processo 5 = Processo E;

- Recurso 1 = Tape drivers;
- Recurso 2 = Plotters;
- Recurso 3 = Printers;
- Recurso 4 = Blu-rays.

6 - CONSIDERAÇÕES FINAIS

No mesmo diretório do .pdf deste relatório há screenshots que mostram as estatísticas sendo ‘printadas’ no terminal e os resultados dos testes sendo ‘printados’ no terminal. Além disso, há um vídeo que mostra todo o processo de compilação e execução do sistema computacional em ambos os modos (teste e normal).