Lista de Exercícios 8 – Sincronização de processos Parte 2 – Monitores, Semáforos, Troca de mensagens e Deadlock

Lista Teórica

1) Explique o que são monitores e dê dois exemplos de sua utilização: um para a solução da exclusão mútua e outro para a sincronização condicional.

2) O que é deadlock, quais as condições para obtê-lo e quais as soluções possíveis?

3) Por que os monitores são considerados mecanismos de sincronização estruturados e os semáforos são considerados não estruturados?

4) Em uma aplicação concorrente que controla saldo bancário em contas-correntes, dois processos compartilham uma região de memória onde estão armazenados os saldos dos clientes A e B. Os processos executam concorrentemente os seguintes passos:

Processo 1 (Cliente A)	Processo 2 (Cliente B)
/* saque em A */	/*saque em A */
1a. x := saldo_do_cliente_A;	2a. y := saldo_do_cliente_A;
1b. x := x - 200;	2b. y := y - 100;
1c. saldo_do_cliente_A := x;	2c. saldo_do_cliente_A := y;
/* deposito em B */	/* deposito em B */
1d. x := saldo_do_cliente_B;	2d. y := saldo_do_cliente_B;
1e. x := x + 100;	2e. y := y + 200;
1f. saldo_do_cliente_B := x;	2f. saldo_do_cliente_B := y;

Supondo que os valores dos saldos de A e B sejam, respectivamente, 500 e 900, antes de os processos executarem, pede-se:

a. Neste trecho de código há a presença de condição de corrida (race condition)? Por quê?
b. Quais os valores corretos esperados para os saldos dos clientes A e B após o término da execução dos processos?
c. Quais os valores finais dos saldos dos clientes se a sequência temporal de execução das operações for: 1a, 2a, 1b, 2b, 1c, 2c, 1d, 2d, 1e, 2e, 1f, 2f?
d. Utilizando semáforos, proponha uma solução que garanta a integridade dos saldos e permita o maior compartilhamento possível dos recursos entre os processos.

Lista Prática 3 – Ambiente Linux

1) Antes de começarmos, vamos preparar o ambiente para a prática. Siga os passos abaixo:
   - Abra o terminal do Ubuntu (CTRL + ALT +T)
   - Digite o comando "ls" (sem aspas), sua função é listar o conteúdo de um diretório.
   - Crie uma pasta para você salvar os arquivos da prática. Para isso utilize o comando "mkdir Lab03", este comando vai criar uma pasta chamada Lab03.
   - Digite "ls" e veja se você consegue visualizar a nova pasta criada.
   - Após criar a pasta, digite "cd Lab03" para acessar o conteúdo da pasta.
   - Acesse o moodle e baixe o arquivo "codes3.zip". Salve-o na pasta "Lab03"
   - Digite o comando "unzip codes3.zip" para descompactar os arquivos.
   - Caso tenha alguma dúvida sobre os parâmetros de um comando, digite "man" e depois o nome do comando para ver a documentação.

2) Sincronização de Processos: Acesse o diretório onde você salvou os arquivos de "codes3.zip".
   - Procedimento 1:
     1. Abra o terminal do Ubuntu (CTRL + ALT +T)
     2. Acesse o diretório em que você salvou os arquivos
     3. Digite o comando nano p1.c
     4. Analise o código-fonte. Digite (Ctrl+X) para sair do editor
     5. Execute o código-fonte
     6. Para compilar digite: gcc -pthread p1.c – o p1
     7. Execute o comando: ./p1

Responda as questões a seguir:

a) O que aconteceu ao executar o código-fonte? Descreva.

b) Podemos dizer que há uma situação de deadlock neste código? Por quê?

3) Jantar dos Filósofos – Capítulo 7 do Livro. O problema dos filósofos é um exemplo clássico de sincronização de processos proposto por Dijkstra. Nesse problema, há uma mesa com cinco pratos e cinco garfos onde os filósofos podem sentar, comer e pensar. Toda vez que um filósofo para de pensar e deseja comer é necessário que ele utilize dois garfos, posicionados à sua direita e à sua esquerda. O procedimento 1 ilustra como executar o código-fonte "p2.c" que contém uma solução para o problema.

   Acesse o diretório onde você salvou os arquivos de "codes3.zip".
   - Procedimento 1:
     1. Abra o terminal do Ubuntu (CTRL + ALT +T)
     2. Acesse o diretório em que você salvou os arquivos
     3. Digite o comando nano p2.c
     4. Analise o código-fonte. Digite (Ctrl+X) para sair do editor

5. Execute o código-fonte
6. Para compilar gcc -pthread p2.c – o p2
7. Execute o comando: ./p2

a) O que ocorreu ao executar o procedimento 1? O algoritmo proposto resolve totalmente o problema? Por quê?
b) Caso, o algoritmo não tenha resolvido totalmente o problema, quais outras soluções poderiam ser utilizadas para resolver o problema? (Não é necessário implementar, apenas cite.)

- Procedimento 2:
  1. Abra o terminal do Ubuntu (CTRL + ALT +T)
  2. Acesse o diretório em que você salvou os arquivos
  3. Digite o comando nano p3.c
  4. Analise o código-fonte. Digite (Ctrl+X) para sair do editor
  5. Execute o código-fonte
  6. Para compilar digite: gcc -pthread p3.c – o p3
  7. Execute o comando: ./p3

Responda as questões a seguir:

a) O que ocorreu ao executar o procedimento 2? O algoritmo proposto resolve totalmente o problema? O que mudou em relação ao algoritmo executado no procedimento 1?

4) Pesquise sobre os comandos “nice” e “renice” no Linux, descrevendo qual é o seu funcionamento, sintaxe e argumentos.