Disciplina: INE 5611 – Sistemas Operacionais **Turma**: 0432 **Semestre**: 2013.2
Professora: Carla Merkle Westphall

Exercícios sobre Semáforos e sincronização Java

1. Escreva um programa concorrente formado por duas threads que sincronizam suas ações com o uso de semáforos. Uma das threads escreve na tela os números de 1 até 10, e então deve esperar pela outra que escreve na tela os números de 20 até 30, e vice-versa. Considere que a execução começa pela THREAD que conta até 10.

2. Escreva um programa concorrente formado por três threads – T0, T1, T2 – que executam um loop infinito. Cada thread escreve na tela a sua identificação. A ordem de escrita na tela deve ocorrer sempre na seguinte seqüência: T1, T2, T0 e assim sucessivamente. Utilize somente SEMÁFOROS para a sincronização.

3. Escreva um programa formado por três processos concorrentes, `observador`, `tratador1` e `tratador2`, que executam um loop infinito, e que sincronizam suas ações com o uso de semáforos. O processo observador lê valores inteiros, que representam a temperatura de um certo dispositivo. Se o valor lido é menor ou igual a 40, deve ser notificado o processo tratador1. Caso o valor seja maior que 50, deve ser notificado o processo tratador2. Cada processo tratador deverá fazer um printf a cada 10 notificações recebidas.

4. Escreva um programa formado por dois processos concorrentes (threads), leitor e impressor, que executam um loop infinito, e que sincronizam suas ações com o uso de semáforos. O leitor fica lendo caracteres do teclado e colocando em um buffer de 15 posições. Quando o buffer está cheio o processo impressor deve imprimi-lo.

5. (Questão de prova) Uma pessoa compra água e refrigerante usando uma máquina automática: a pessoa insere moedas e a máquina dá o produto escolhido. Cada moeda (R$ 0,50 e R$ 1,00) é colocada em um local apropriado e a máquina não fornece troco. Escreva um programa formado por uma classe Pessoa (thread) que acessa, em um loop infinito, uma Máquina automática de venda de água e refrigerante. Se a pessoa depositar moeda de R$ 0,50 a máquina libera uma garrafa de água, mas se a pessoa depositar moeda de R$ 1,00 a máquina libera uma lata de refrigerante. Aleatoriamente a pessoa escolhe o valor e o produto a ser comprado. Sincronize o acesso a esta máquina: a máquina primeiro recebe o pagamento (método paga) e somente depois pode dar o produto (método liberaProduto). Implemente a classe Pessoa, a classe Máquina sincronizando seu acesso somente com o uso de semáforos e uma classe *main* para instanciar os objetos.

6. Considere uma sala de exposições e um conjunto de visitantes. Cada visitante faz o seguinte ciclo infinito: **`entra`**`()` numa sala de exposições e **`sai`**`()` dela ao fim de um determinado tempo. Cada exposição numa sala começa e acaba a uma hora determinada, e os visitantes só podem entrar e sair na hora determinada. Cada sala tem N lugares; se houver mais visitantes, estes esperam pela próxima sessão. Elabore duas soluções completas para este cenário: a primeira solução deve usar somente semáforos e a segunda solução deve usar somente sincronização Java (métodos sincronizados, wait e notify).

7. Suponha que existem *n* passageiros e um carro. Os passageiros, repetidamente, esperam para embarcar no carro, que pode suportar *c* passageiros (*c* < *n*). Entretanto, o carro só inicia o passeio quando está cheio. Implemente a solução deste problema usando semáforos para sincronização das threads.

8. Em uma Corrida de Fórmula 1 todos os pilotos concorrem a uma parada no Box mas apenas um deles pode ocupar o Box de cada vez. O Box fica bloqueado enquanto estiver sendo usado, impedindo o acesso de outro piloto. Implemente em Java a `classe Box` e a `classe PilotoBox`. Implemente a sincronização usando semáforos e depois usando a sincronização Java. Observe o main (classe `TestaPilotoBox`) que já está pronto.

```
public class Box {}
public class PilotoBox extends Thread { }
public class TestaPilotoBox{
   public static void main(String args[]){
      Box b=new Box();
      PilotoBox um = new PilotoBox("Rubinho",b);
      PilotoBox dois = new PilotoBox("Massa",b);
      PilotoBox tres = new PilotoBox("Alonso",b);
      um.start();
      dois.start();
      tres.start();
   }}
```

9. Uma solução para o problema do *jantar dos filósofos* foi apresentada nos slides (slides 51 e 52 do material 8). Desenvolva uma solução para o problema do jantar dos filósofos usando sincronização com semáforos (a variável condicional pode ser implementada como semáforos) .

10. (Questão de prova) Em um serviço de caixa postal de telefonia celular, a cada 5 mensagens recebidas por um telefone, o dono do telefone recebe o seguinte aviso "5 mensagens – Enviadas por: 111 222 333 444 555". O aviso, além de notificar o recebimento de 5 mensagens ainda mostra quais os números dos celulares que enviaram as mensagens (neste exemplo os números são 111 222 333 444 555). Implemente a classe/thread "Emissora" de mensagens – esta classe fica enviando as mensagens para o mesmo telefone celular em um loop infinito. Implemente a classe/thread "Usuário", que fica em loop infinito recebendo o aviso. Implemente o recurso compartilhado e seus métodos, **usando sincronização com semáforos**. Implemente também um `main` para instanciar as threads.

11. (Questão de prova) Implemente uma partida de tênis usando **semáforos e threads Java**. Instancie as threads Jog1 (jogador 1) e Jog2 (jogador 2), que executam um loop infinito. Implemente a classe do recurso compartilhado e use sincronização com semáforos. Implemente também um `main` para instanciar as threads.

12. [Jantar] Uma tribo de selvagens janta em comunidade a partir de uma mesa enorme com espaço para M javalis grelhados. Quando um selvagem quer comer, serve-se do javali da mesa a menos que esta já esteja vazia. Nesse caso o selvagem acorda o cozinheiro e aguarda que este reponha javalis na mesa. O código seguinte representa o código que implementa o selvagem e o cozinheiro.

```
void selvagem ()
{     while(true) {
            Javali j = RetiraJavali();
            ComeJavali(j);    }
}

void Cozinheiro()
{     while(true) {
            ColocaJavalis(M);
      }
}
```

Implemente o código das funções `RetiraJavali()` e `ColocaJavalis()` incluindo código de sincronização que previna *deadlock* e acorde o cozinheiro apenas quando a mesa está vazia. Implemente em Java, usando **semáforos**.

- Prof. Islene - http://www.ic.unicamp.br/~islene/2s2013-mc514/
- Semáforos e problemas de sincronização: The Little Book on Semaphores, de Allen B. Downey.