Computação Concorrente (DCC/UFRJ)

Aula 13: Exercícios

Prof. Silvana Rossetto

19 de junho de 2012

Exemplo de uso de monitor

- **Problema**: gerenciar o acesso por várias threads de N unidades replicadas de um determinado recurso
- **Implemente uma classe Java que encapsule as operações de acesso às unidades do recurso**
  - caso uma thread solicite uma unidade quando todas estão alocadas, a thread deve ser bloqueada até que uma unidade seja liberada

..sugestão inicial..

```
class Recurso {
  //variaveis
  Recurso() { //construtor (inicializações)}
  public synchronized void DevolveUnd(int und) {...}
  public synchronized int SolicitaUnd() {...}
}
```

```
class Recurso {
  static final int N; int[] pool;
  int count, in=0, out=0, esperando=0;
  Recurso() {
     pool = new int[N];
     for (int i=0;i<N;i++)  pool[i] = i+1;
     count = N; //N unidades disponiveis
  }
  public synchronized void DevolveUnd(int und) {
      pool[in%N] = und;
      in++; count++;
      if (esperando > 0) {
        esperando--; notify(); }
  }
  (...)
```

```
  (...)
  public synchronized int SolicitaUnd() {
   int unidade;
   try {
     while (count==0) {
        esperando++; wait(); }
     unidade = pool[out%N];
     out++; count--;
     return unidade;
    } catch (InterruptedException e) {...}
  }
}
```

Canais de mensagens

Veja o código no próximo slide:

1. Qual é o tipo do canal implementado (mailbox, porta ou enlace)?
2. Qual é o modo desse canal (síncrono ou assíncrono)?
3. Altere o código para que o canal passe para o modo inverso (i.e., se assíncrono, vire síncrono; ou vice-versa).

```
public class Channel {
 private Object m = null, s = new Object();
 private Object r = new Object();
 private Semaphore enviado = new Semaphore(0);
 private Semaphore recebido = new Semaphore(0);

 public void send(Object msg) {
  synchronized(s)
   { m = msg; enviado.POST(); recebido.WAIT(); }
 }

 public Object receive() {
  Object msg = null;
  synchronized(r)
   { enviado.WAIT(); msg = m; recebido.POST(); }
  return msg;
 }}
```

Canais de mensagens

1. Qual é o tipo do canal implementado (mailbox, porta ou enlace)?
   Mailbox, pois vários emissores e vários receptores podem acessar o mesmo canal
2. Qual é o modo desse canal (síncrono ou assíncrono)?
   Síncrono, pois o emissor fica bloqueado até que um receptor receba a mensagem enviada
3. Altere o código para que o canal passe para o modo inverso (i.e., se assíncrono, vire síncrono; ou vice-versa)
   (ver atividades do Lab12)

Primitivas send/receive e anel de comunicação

- Considere um anel com N processadores
- Cada processador executa um processo com identificador único entre **0** e **N-1**
- Usando as primitivas básicas de troca de mensagem **send/receive**, escreva um algoritmo (pseudo-código) que passe uma mensagem, contendo um inteiro positivo, ao longo desse canal
- O **processo 0** é quem inicia a transmissão, e cada vez que a mensagem passa por ele novamente ela é decrementada
- Quando um processo recebe a mensagem com valor 0 ele a passa adiante e então termina a execução

..sugestão inicial..

```
int dest = (id+1) % N; //N é o número de processos
int fonte = (id+N-1) % N; int msg;

if (id == 0) {
    ...//carrega msg inicial e inicia comunicação
}
while (true) {
    ... //recebe e repassa msgs
    ... //tratamento especial P0
    ... //testa condição de término
}
... //finalização
```

```
int dest = (id+1) % N; //N é o número de processos
int fonte = (id+N-1) % N; int msg;

if (id == 0) {
  msg = //carrega o valor inicial da msg
  send(dest, msg); //envia a msg para o vizinho
}
while (true) {
 receive(fonte, &msg); //recebe a msg do vizinho
 if (id == 0)
   msg--;
 send(dest, msg); //envia a msg para o vizinho
 if (msg == 0) break; //encerra o loop
}
//o ultimo processo faz um send extra para o P0,
//então P0 faz outro receive antes de terminar
if (id == 0) receive(fonte, &msg);
```

Aplicação cliente/servidor

Veja o código no próximo slide:

1. Qual é a lógica de operação do servidor?
2. Qual deve ser a semântica das primitivas *send/receive*?
3. Implemente uma versão para o código do lado do cliente.

```
Servidor:
int aval = N; unit unidade[N];
list pend; int id, any_id, tipo; unit unid;
while(true) {
 receive(&any_id, &tipo, &unid);
 if(tipo==REQUISICAO) {
   if(aval>0) {
     aval--; unid=remove(unidade);
     send(any_id, unid);
   } else insert(pend, any_id);
 } elseif (tipo==LIBERACAO) {
   if(vazio(pend)) {
     aval++; insert(unidade,unid);
   } else {
     id=remove(pend); send(id,unid);}
 } }
```

Aplicação cliente/servidor

1. Qual é a lógica de operação do servidor a seguir?
   O servidor trata da gerência de um recurso computacional com várias unidades (ex. blocos de um arquivo, linhas de uma tabela). Processos clientes solicitam, usam e liberam unidades do recurso (por meio das primitivas REQUISICAO/LIBERACAO). Caso um cliente solicite uma unidade quando todas estão alocadas, o servidor salva a requisição e retarda o envio da resposta até que uma unidade seja liberada (o processo cliente fica bloqueado enquanto isso)
2. Qual deve ser a semântica das primitivas *send/receive*?
   send não-bloqueante e receive bloqueante

Implementação do lado do cliente

```
Cliente:
 unit unid;
 send(servidor_id, REQUISICAO, 0);
 receive(servidor_id, &unid);
 //...secao critica: usa a unidade
 send(servidor_id, LIBERACAO, unid);
 //...prossegue a sua execucao
```