Módulo 2 - Lista de Exercícios 2 (2020/2 REMOTO)

Computação Concorrente (MAB-117)
Prof. Silvana Rossetto

[1]DCC/IM/UFRJ
11 de maio de 2021

Questão 1 (2,5 pontos): Considere uma aplicação com M threads do tipo Foo e N threads do tipo Bar. Nessa aplicação é necessário que ocorra uma alternância rigorosa entre as execuções das threads Foo e as execuções das threads Bar: primeiro, todas as threads Foo devem executar, depois todas as threads Bar devem executar, depois todas as threads Foo devem executar novamente, depois todas as threads Bar devem executar novamente, e assim sucessivamente. *Sua tarefa:*

(a) verificar se o código das threads Foo e Bar mostrado abaixo está correto e se garante que a regra de alternância entre as execuções dos grupos de threads seja cumprida;
(b) verificar se o código das threads garante ausência de condição de corrida e ausência de *deadlock*.

Apresente argumentos detalhados para embasar suas respostas.

```
int contaFoo=0, contaBar=0; //variaveis de estado
pthread_cond_t condFoo, condBar; //inicializadas na main
pthread_mutex_t mutex; //inicializado na main

void * Foo(void* args) {
  while(1) {
     //...codigo principal da thread
     //força alternancia com as threads Bar
     pthread_mutex_lock(&mutex);
     contaFoo++;
     if(contaFoo==M) {
        contaFoo=0;
        pthread_cond_broadcast(&condBar);
     }
     pthread_cond_wait(&condFoo, &mutex);
     pthread_mutex_unlock(&mutex);
  } }

void * Bar(void* args) {
  //aguarda as threads Foo executarem primeiro
  pthread_mutex_lock(&mutex);
  pthread_cond_wait(&condBar, &mutex);
  pthread_mutex_unlock(&mutex);
  while(1) {
     //...codigo principal da thread
     //força alternancia com as threads Foo
     pthread_mutex_lock(&mutex);
     contaBar++;
     if(contaBar==N) {
        contaBar=0;
        pthread_cond_broadcast(&condFoo);
     }
     pthread_cond_wait(&condBar, &mutex);
     pthread_mutex_unlock(&mutex);
  } }
```

Questão 2 (2,5 pontos): A classe Java abaixo implementa uma solução para o problema dos *produtores e consumidores* (o tratamento de exceções foi omitido). Lembrando, as condições do problema são: *(i) os produtores não podem inserir novos elementos quando a área de dados está cheia; (ii) os consumidores não podem retirar elementos quando a área de dados está vazia; (iii) os elementos devem ser retirados na mesma ordem em que foram inseridos; (iv) os elementos inseridos não podem ser sobreescritos por novos elementos antes de serem consumidos; (v) um mesmo elemento só pode ser consumido uma única vez.*

A solução apresentada funcionou corretamente em várias execuções de uma aplicação alvo. Mas em uma execução qualquer, a aplicação entrou em estado de *deadlock*. *Sua tarefa:*

(a) descobrir o que causou o *deadlock*;
(b) mostrar uma possível sequência de execução que levou até ele;
(c) mostrar como o código poderia ser corrigido para eliminar a possibilidade de *deadlock* nessa aplicação.

Apresente argumentos detalhados para embasar suas respostas.

```
class Buffer {
  static final int N = 10; //qtde de elementos no buffer
  private int[] buffer = new int[N]; //area de dados compartilhada
  //variaveis de estado
  private int count=0; //qtde de posicoes ocupadas no buffer
  private int in=0;  //proxima posicao de insercao
  ptivate int  out=0;  //proxima posicao de retirada

  // Construtor
  Buffer() {... // inicializa o buffer }

  // Insere um item
  public synchronized void Insere (int item) {
      while (count==N) {
           wait();
      }
      buffer[in%N] = item;
      in++;
      count++;
      notify();
  }

  // Remove um item
  public synchronized int Remove (int id) {
     int aux;
     while (count==0) {
          wait();
     }
     aux = buffer[out%N];
     out++;
     count--;
     notify();
     return aux;
  }
}
```

Questão 3 (2,5 pontos): No código mostrado abaixo, a thread B foi programada para imprimir o valor de X quando ele for divisível por 10. Em uma execução com *dez* threads A e *uma* thread B, o valor *11* foi incorretamente impresso. *Sua tarefa*:

(a) descrever o que aconteceu;
(b) mostrar como o código pode ser corrigido para que essa situação não se repita.

Apresente argumentos detalhados para embasar suas respostas.

```
int x = 0; pthread_mutex_t x_mutex; pthread_cond_t x_cond;
void *A (void *tid {
   for (int i=0; i<100; i++) {
     pthread_mutex_lock(&x_mutex);
     x++;
     if(!(x%10))
        pthread_cond_signal(&x_cond);
     pthread_mutex_unlock(&x_mutex);
} }

void *B (void *tid) {
  pthread_mutex_lock(&x_mutex);
  if(x%10)
     pthread_cond_wait(&x_cond, &x_mutex);
  printf("X=%d\n", x);
  pthread_mutex_unlock(&x_mutex);
}
```

Questão 4 (2,5 pontos): A classe Java abaixo implementa um `monitor` para resolver o problema dos leitores/escritores com a escrita sendo feita diretamente dentro do monitor. Os requisitos do problema continuam sendo os seguintes: *(i) mais de um leitor pode ler ao mesmo tempo uma área de dados compartilhada; (ii) apenas um escritor pode escrever de cada vez nessa mesma área; e (iii) enquanto o escritor está escrevendo os leitores não podem ler*. Antes de ler a área de dados compartilhada, as threads leitoras deverão chamar o método `EntraLeitor()` e após terminar a leitura deverão chamar o método `SaiLeitor()`. As threads escritoras deverão chamar o método `Escrita()` passando o dado que deve ser escrito. *Sua tarefa*:

(a) verifique se essa solução está correta e atende a todos os requisitos do problema;
(b) verifique se a chamada ao método `notifyAll()` da linha 11 poderia ser substituída por uma chamada ao método `notify()` sem afetar a corretude da solução.
(c) verifique se a chamada ao método `notify()` da linha 19 poderia ser excluída sem afetar a corretude da solução;

Apresente argumentos detalhados para embasar suas respostas.

```
1: class LEMonitor {
2:   private int leit;
3:   LEMonitor() { this.leit = 0; }

5:   // Entrada para leitores
6:   public synchronized void EntraLeitor() { this.leit++; }

7:   // Saida para leitores
8:   public synchronized void SaiLeitor() {
9:      this.leit--;
10:     if (this.leit == 0)
11:        notifyAll();
12:  }

13:  // Metodo para escritores
14:  public synchronized void Escrita(String str) {
15:     try {
16:         while (this.leit>0) { wait(); }
17:      } catch(InterruptedException e) {}
18:      //realiza a escrita de 'str'
19:      notify();
20:  } }
```