Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Programação Concorrente
Teste Global da Época Normal, Verão de 2020/2021
Duração: 2 horas

---

1. [6] Implemente usando a linguagem Java o sincronizador *message box*, com os métodos apresentados em seguida.

```
public class MessageBox<T> {
   public Optional<T> waitForMessage(long timeout) throws InterruptedException;
   public int sendToAll(T message);
}
```

O método **`waitForMessage`** bloqueia a *thread* invocante até que uma mensagem seja enviada através do método **`sendToAll`**. O método **`waitForMessage`** pode terminar com: 1) um objecto **`Optional`** contendo a mensagem enviada; 2) um objecto **`Optional`** vazio, caso o tempo de espera definido por **`timeout`** seja excedido sem que uma mensagem seja enviada; 3) com o lançamento duma excepção do tipo **`InterruptedException`**, caso a *thread* seja interrompida enquanto em espera.
O método **`sendToAll`** deve retornar o número exacto de *threads* que receberam a mensagem, podendo este valor ser zero (não existiam *threads* à espera de mensagem), um, ou maior que um. A mensagem passada na chamada **`sendToAll`** não deve ficar disponível para chamadas futuras do método **`waitForMessage`**.

2. [6] Realize usando a linguagem Java o sincronizador **`NArySemaphore`** com a interface apresentada em seguida.

```
public class NArySemaphore {
   public NArySemaphore(int initial)
   public boolean acquire(int n, long timeout) throws InterruptedException;
   public boolean lowPriorityAcquireAll(long timeout) throws InterruptedException;
   public void release(int n);
}
```

Este sincronizador é um semáforo com aquisição e libertação *n*-ária, e ordem de aquisição FIFO (*first in first out*), semelhante ao realizado nas aulas. A principal diferença é a adição do método potencialmente bloqueante **`lowPriorityAcquireAll`**, que tem por objectivo a aquisição de todas as unidades do semáforo, com prioridade inferior aos pedidos realizados através do método **`acquire`**. Uma chamada a este método pode retornar com sucesso quando: **`initial`** unidades estiverem disponíveis; não existir nenhum pedido pendente de aquisição feito através do método **`acquire`** (anterior ou posterior à chamada). Os pedidos realizados através de chamadas ao método **`lowPriorityAcquireAll`** também devem ser satisfeitos por ordem FIFO.

3. [5] Considere a classe **`BroadcastBox`** com os métodos apresentados em seguida. Cada instância oferece uma interface assíncrona para envio de uma mensagem do tipo **`T`**, através do método **`SendToAll`**, para um conjunto de receptores que previamente manifestarem interesse na recepção da mensagem, através do método **`WaitForMessageAsync`**. O método **`SendToAll`** deve retornar o número de receptores que receberam a mensagem enviada (este número pode ser zero, um, ou maior do que um). A mensagem passada na chamada **`sendToAll`** não deve ficar disponível para chamadas futuras do método **`WaitForMessageAsync`**.

```
public class BroadcastBox<T>
{
   public Task<T> WaitForMessageAsync();
   public int SentToAll(T message);
}
```

   a. Implemente a classe **`BroadcastBox`** com a interface apresentada.
   b. Implemente a variante da classe **`BroadcastBox`** em que o método **`WaitForMessageAsync`** é cancelável através de um *cancellation token* passado como parâmetro.

4. [3] Considere o seguinte método:

```
public T Oper<T>(T[] xs, T initial)
{
   var acc = initial;
   for (var i = 0; i < xs.Length; ++i)
   {
       var ai = A(xs[i]);
       acc = E(D(B(ai), C(ai)), acc);
   }
   return acc;
}
```

   Realize na linguagem C# a versão assíncrona do método **`Oper`**, seguindo o padrão TAP (*Task-based Asynchronous Pattern*). Assuma que tem à disposição versões assíncronas dos métodos **`A`**, **`B`**, **`C`**, **`D`** e **`E`**. Todos os métodos, com excepção do método **`E`**, não produzem efeitos colaterais e podem ser chamados em concorrência. O método **`E`** não é associativo. Tire partido do paralelismo potencial existente. Não é necessário suporte para cancelamento.

Duração: 2 horas
ISEL, 13 de julho de 2021