Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação Concorrente

Teste Global de 1ª Época, Inverno de 2014/2015

---

1. [3] Considere a classe `UnsafeSpinLifoMsgQueue`, cuja implementação em C# é apresentada a seguir:

```
public class UnsafeSpinLifoMsgQueue<T> {

   private class Node<E> {
      internal Node<E> next;
      internal E msg;
      internal Node(E msg) {
         this.msg = msg;
      }
   }

   private Node<T> top;

   public void Send(T msg) {
      Node<T> node = new Node<T>(msg);
      node.next = top;
      top = node;
   }

   public T Receive() {
      SpinWait sw = new SpinWait();
      Node<T> oldTop;
      while ((oldTop = top) == null)
         sw.SpinOnce();
      top = oldTop.next;
      return oldTop.msg;
   }
}
```

   Indique as razões pelas quais esta classe não é *thread-safe* e, sem recorrer à utilização de *locks*, apresente as alterações necessárias para a tornar *thread-safe*.

2. [4] Usando a linguagem *Java* ou a linguagem C# e os respectivos monitores intrínsecos, implemente o sincronizador *generic synchronizer*, com a classe `GenericSync`, cuja interface pública é a seguinte:

```
public class GenericSync {
   public GenericSync(int initial);
   public bool Wait(int timeout);
   public void ReleaseOne();
   public void ReleaseAll();
}
```

   O *generic synchronizer* pode assumir a semântica de um semáforo ou de um *manual reset event*, sobrepondo-se esta última semântica à primeira. O parâmetro do construtor do sincronizador especifica o número de chamadas ao método `Wait` que retornam de imediato, antes da invocação de qualquer dos métodos `ReleaseXxx`. Quando se utiliza o método `Wait` em conjunção com o método `ReleaseOne`, o sincronizador apresenta semântica de semáforo, em que cada chamada a `ReleaseOne` satisfaz uma chamada ao método `Wait`. Quando se utiliza o método `Wait` em conjunção com o método `ReleaseAll`, a chamada ao método `ReleaseAll` satisfaz todas as chamadas, pendentes e futuras, ao método `Wait`. A implementação do sincronizador deverá suportar desistência por *timeout* e a interrupção das *threads* bloqueadas em `Wait`.

3. [4] Implemente em Java ou C#, com base nos monitores implícitos, o sincronizador *data aggregator* para a suportar a comunicação entre *threads* produtoras e consumidoras em cenários onde as *threads* consumidoras recolhem para processamento toda a informação disponível até ao momento.

```
public class DataAggregator<D> {
   public void Put(D data);
   public List<D> TakeAll();
}
```

   O método `Put` entrega um item de dados ao sincronizador. O método `TakeAll` recolhe todos os itens de dados disponíveis, bloqueando a *thread* invocante enquanto estes não existirem ou até que a *thread* seja alvo de uma interrupção. As chamadas ao método `TakeAll` devem ser servidas com disciplina LIFO (*last-in-first-out*).

4. [9] A classe `Reviews` fornece um serviço assíncrono de consulta remota de registos de opinião. Por outro lado, a classe `Translations` dá acesso a um serviço remoto de tradução de texto. Tirando partido destes serviços, pretende-se criar um novo serviço, também assíncrono, que permite obter conjuntos de opiniões traduzidas para uma língua-alvo. Internamente, por cada pedido de `ObtainAllInfo` são realizados pedidos paralelos para todos os *reviews*, cujos resultados são traduzidos (`inLang="en"`) ainda em paralelo, ficando a operação concluida quando todas as traduções estiverem concluídas. Todos os serviços envolvidos disponibilizam interfaces assincronas no estilo *Asynchronous Programming Model* (APM) e *Task-based Asynchronous Pattern* (TAP).

```
public class Reviews { /* implementada */
   public IAsyncResult BeginObtainReview(int ReviewId,
                                         AsyncCallback callback, object state);
   public String EndObtainReview(IAsyncResult operation);
   public Task<String> ObtainReviewAsync(int ReviewId);
}

public class Translations { /* implementada */
   public IAsyncResult BeginTranslate(String text, String inLang, String outLang,
                                      AsyncCallback callback, object state);
   public String EndTranslate(IAsyncResult operation) { ... }
   public Task<String> TranslateAsync(String text, String inLang, String outLang);
}

public class AllInfo { /* para implementar */
   public IAsyncResult BeginObtainAllInfo(int[] ReviewIds, String outLang,
                                    AsyncCallback callback, object state);
   public String[] EndTranslate(IAsyncResult operation);
   public Task<String[]> ObtainAllInfoAsync(int[] ReviewIds, String outLang);
}
```

   a. [5] Implemente os métodos `BeginObtainAllInfo` e `EndObtainAllInfo`.

   b. [4] Tirando partido da *Task Parallel Library* (TPL), implemente o método `ObtainAllInfoAsync`.

Duração: 2 horas e 30 minutos
ISEL, 6 de Fevereiro de 2015