Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação Concorrente, Verão de 2013/2014

Teste Final (2ª Época)

1. [3] Considere a classe UnsafeSpinSemaphore, não *thread-safe*, cuja implementação em C# é apresentada a seguir:

```
public class UnsafeSpinSemaphore {
   private int permits;

   public UnsafeSpinSemaphore(int initial) {
      if (initial < 0) throw new InvalidOperationException();
      permits = initial;
   }

   public void Acquire(int permitsToAcquire) {
      if (permits >= permitsToAcquire) {
         permits -= permitsToAcquire;
         return;
      }
      SpinWait sw = new SpinWait();
      do {
         sw.SpinOnce();
      } while (permits < permitsToAcquire);
      permits -= permitsToAcquire;
   }

   public void Release(int permitsToRelease) {
      if (permits + permitsToRelease < permits) throw new InvalidOperationException();
      permits += permitsToRelease;
   }
}
```

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

2. [4] Usando a linguagem *Java* e os respectivos monitores implícitos, implemente o sincronizador *count down latch* com a classe CountdownLatch, cuja interface pública é a seguinte:

```
public class CountdownLatch {
   public CountdownLatch(int partners);
   public void signal();
   public boolean await(int timeout) throws InterruptedException;
}
```

   Este sincronizador suporta sincronização do tipo *join* de uma ou mais *threads* com um grupo de *worker threads*. O número de *worker threads* envolvidas é passado ao construtor do objecto através do parâmetro partners. Quando terminam o respectivo processamento, cada uma das *worker threads* invoca o método signal, o que tem como consequência o decremento do contador com o número de partners ainda por terminar. (Se o método signal for invocado quando o contador é zero, a implementação deve lançar a excepção IllegalStateException.) Quando o contador com o número de partners por terminar chega a zero, são libertadas todas as *threads* que se tenham bloqueado no sincronizador com o método await. Uma *thread* podem retornar do métodos await porque todas as *worker threads* já invocaram o método signal, expirou o limite de tempo especificado através do parâmetro timeout, ou porque a espera das *thread* foi cancelada por interrupção.

3. [5] Usando os monitores implícitos *Java* ou da CLI ou os monitores explícitos do *Java*, implemente a classe MessageQueue<M> que implementa o sincronizador *message queue* que suporta a comunicação entre *threads* através de mensagens do tipo genérico M. A interface pública em C# desta classe é a seguinte:

```
public class MessageQueue<M> where M : class {
   public void PostMessage(M message);
   public bool SendMessage(M message, int timeout);
   public M GetMessage(int timeout);
}
```

Existem duas formas de enviar mensagens para a fila. O método PostMessage entrega a mensagem especificada à fila, e nunca bloqueia a *thread* invocante. O método SendMessage entregar uma mensagem à fila, mas bloqueia a *thread* invocante até que a mensagem seja recebida por outra *thread*, expire o limite de tempo especificado através do parâmetro timeout ou a espera da thread seja cancelada por interrupção. O método GetMessage bloqueia a *thread* invocante até que exista uma mensagem na fila disponível para recepção, expire o limite de tempo especificado através do parâmetro timeout ou a espera da *thread* seja cancelada por interrupção. As mensagens devem ser entregues às *threads* receptoras por ordem de chegada (FIFO) mas as *threads* receptoras devem ser selecionadas por ordem inversa àquela com que invocam GetMessage (LIFO).

4. [8] Considere a classe AsyncProcessFile, já implementada, cuja interface pública é a seguinte:

```
public class AsyncProcessFile {
   public ProcessFile(string filePath);
   public IAsyncResult BeginProcess(IAsyncCallback callback, object state);
   public long EndProcess(IAsyncResult asyncResult);
   public Task<long> ProcessAsync();
}
```

A classe AsyncProcessFile realiza, assincronamente, um determinado tipo do processamento sobre o conteúdo do ficheiro cujo nome e passado como argumento ao seu construtor; o resultado desse processamento é um inteiro expresso a 64-*bit*. Esta classe tem duas interfaces: uma, ao estilo APM (*Asynchronous Programming Model*) suportada pelos métodos BeginProcess e EndProcess, e; outra, ao estilo TAP (*Task-based Asynchronous Pattern*), suportada pelo método ProcessAsync.

Considere também a classe AsyncProcessFiles, a implementar, que define métodos estáticos para processar uma lista de ficheiros, usando a funcionalidade implementada pela classe AsyncProcessFile.

```
public static class AsyncProcessFiles {
   public static IAsyncResult BeginProcessFiles(string[] filePaths, Func<long> initial,
                           Func<long, long, long> reducer,  IAsyncCallback callback, object state);
   public static long EndProcessFiles(IAsyncResult asyncResult);
   public static Task<long> ProcessFilesAsync(string[] filePaths, Func<long> initial,
                           Func<long, long, long> reducer);
}
```

O resultado final do processamento dos ficheiros é um inteiro, expresso com 64-*bit*, e calculado do seguinte modo: o resultado parcial é iniciado com o valor devolvido pela função initial; depois, por cada ficheiro processado, o resultado parcial é substituído pelo valor retornado pela função reducer invocada como resultado parcial corrente como primeiro argumento e o resultado do processamento do ficheiro como segundo argumento. As implementação desta classe devem explorar todo o paralelismo potencial possível, tendo em consideração que o processamento dos ficheiros é independente entre si.

a) [4] Implemente os métodos BeginProcessFilesAsync e EndProcessFiles.

b) [4] Usando a TPL (*Task Parallel Library*) e/ou os métodos assíncronos do .NET *Framework*, implemente o método ProcessFilesAsync.

Duração: 2 horas e 30 minutos
ISEL, 16 de Julho de 2014