Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação Concorrente

Verão de 2020/2021, Terceira Série de Exercícios

Resolva os seguintes exercícios e apresente os programas de teste com os quais validou a correção da implementação de cada exercício. A entrega deve ser feita através da criação da tag **`1.0.0`** no repositório de cada aluno ou grupo de alunos.

1. Considere o seguinte método:

```
public T Oper<T>(T[] xs, T[] ys, T initial)
{
   if (xs.Length != ys.Length) throw new ArgumentException("...");
   var acc = initial;
   for (var i = 0; i < xs.Length; ++i)
   {
       acc = D(C(B(A(xs[i])), B(A(ys[i]))), acc);
   }
   return initial;
}
```

   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`** e **`D`**. Todos os métodos não produzem efeitos colaterais e podem ser chamados em concorrência. O método **D** não é associativo. Tire partido do paralelismo potencial existente. Não é necessário suporte para cancelamento.

2. Realize em C# a class **`Exchanger`**

```
public class Exchanger<T>
{
   public Task<T> ExchangeAsync(T message);
}
```

   Este tipo suporta a troca assíncrona de mensagens entre pares de chamadas a **`ExchangeAsync`**: se a *task* retornada pela chamada **C1** se completar com a mensagem fornecida pela chamada **C2**, então a *task* retornada na chamada **C2** completa-se com a mensagem fornecida pela chamada **C1**. As chamadas ao método **`ExchangeAsync`** devem retornar o mais depressa possível, com uma *task* completada ou não completada.

   a. Implemente a versão da classe **`Exchanger`** sem suporte a cancelamento no método **`ExchangeAsync`**.

   b. Implemente a versão da classe **`Exchanger`** com suporte a cancelamento no método **`ExchangeAsync`**, que passa a receber também uma instância de **`CancellationToken`**. Note que se a *task* retornada pela chamada **C1** se completar com cancelamento, então a mensagem fornecida pela chamada **C1** não pode ser usada para completar com sucesso qualquer *task* retornada por outra chamada.

3. Realize em C# a classe **`MessageQueue`**, com os métodos apresentados em seguida.

```
public class MessageQueue<T>
{
   Task EnqueueAsync(T message, CancellationToken cancellationToken);
   Task<T> DequeueAsync(CancellationToken cancellationToken)
   Task ShutdownAsync()
}
```

O método **`EnqueueAsync`** adiciona uma mensagem à fila, retornando uma *task* que será completada quando a mensagem for entregue a um receptor. O método **`EnqueueAsync`** recebe também um *cancellation token* para solicitar o cancelamento dessa adição. Note-se que o cancelamento não é sempre possível de realizar com sucesso, nomeadamente quando a mensagem adicionada já tiver sido entregue a um receptor. O cancelamento com sucesso deve ser refletido na *task* retornada pelo método **`EnqueueAsync`**.

O método **`DequeueAsync`** inicia uma remoção da fila, retornando uma *task* que será completada com a mensagem retirada, quando esta estiver disponível. Este método também recebe um *cancellation token* para solicitar o pedido de cancelamento da remoção, de forma similar ao método **`EnqueueAsync`**.

O método **`ShutdownAsync`** inicia o processo de shutdown da fila, retornando uma *task* que será completada quando o processo de *shutdown* estiver concluído. O início do *shutdown* deve impedir a entrega de mais mensagens à fila, contudo as mensagens já presentes devem manter-se na fila. O processo de shutdown é considerado concluído quando não existirem mais mensagens na fila. O método **`ShutdownAsync`** deve ser idempotente, podendo ser chamado múltiplas vezes.

Note-se que todos os métodos podem retornar task já completadas, caso a operação associada seja realizada de forma síncrona (e.g. **`DequeueAsync`** realizado sobre uma fila contendo elementos).

Data limite de entrega: 10 de julho de 2021

ISEL, 5 de junho de 2021