# Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação Concorrente, Verão de 2019/2020

Série de Exercícios 3

1. Considere o seguinte método:

```
public R[] Compute(T[] elems)  {
  var res = new R[elems.Length];
  for (int i = 0 ; i < elems.Length ; i++)
    res[i] = Oper(elems[i]);
  return res;
}
```

O método **`Oper`** é uma função sem efeitos colaterais e passível de múltiplas execuções em paralelo. Esporadicamente pode lançar uma excepção (e.g. devido a erro de comunicação). Realize uma versão assíncrona do método **`Compute`** seguindo o padrão TAP (*Task-based Asynchronous Pattern*) usando a TPL e/ou os métodos assíncronos do C#. Assuma que tem disponível uma versão TAP do método **`Oper`**. Tire partido do paralelismo potencial existente. A versão assíncrona deve tolerar erros na operação **`Oper`**, tentando realizar de novo a operação para o mesmo parâmetro. Contudo, não devem ser realizadas mais do que **`maxRetries`** no `total`, onde **`maxRetries`** é um parâmetro de entrada. Caso seja excedido o número de tentativas, devem ser canceladas todas as operações pendentes.
Realize um programa para verificar a correcção da sua implementação, incluindo uma versão de teste do método **`Oper`**. Esta versão de teste deve simular uma operação com tempo de execução variável, sem recorrer a bloqueio de *threads*, e com erros esporádicos.

2. Realize na plataforma .NET um servidor com interface TCP para acesso a *transfer queues* usado o modelo de interacção pedido-resposta.
   - Os pedidos e as respostas são objetos JSON.
   - Os pedidos tem a seguinte estrutura genérica, inspirada na estrutura dos pedidos HTTP.

```
{
    “method”: “a string with the command operation”,
    “path”: “a string with a resource path”,
    “headers”: { … a JSON object where all fields are strings … },
    “payload”: { … a JSON object …}
}
```

   - As respostas tem a seguinte estrutura genérica, inspirada na estrutura das respostas HTTP.

```
{
    “status”: … an integer … ,
    “headers”: { … a JSON object where all fields are strings… },
    “payload”: { … a JSON object …}
}
```

   - O servidor aceita as seguintes operações:
     - **`CREATE`** - garante a existência da fila com o nome definido no campo **`path`**.
     - **`PUT`** - envia a mensagem presente em **`payload`** para a fila com nome definido em **`path`**, retornando imediatamente.
     - **`TRANSFER`** - envia a mensagem presente em **`payload`** para a fila com nome definido em **`path`**. A operação só é concluída quando a mensagem for removida ou o tempo máximo de espera, definido em milésimos de segundo no header **`timeout`**, for ultrapassado. Se esta operação for cancelada por ter sido excedido o limite de tempo, a mensagem subjacente deve ser descartada.

    - **`TAKE`** - retira uma mensagem da fila com nome definido em **`path`** e retorna-a no campo **`payload`**. O tempo máximo de espera, em milésimos de segundos, é definido no *header* **`timeout`**.
- O servidor usa os seguintes *status codes*:
    - 200 - operação realizada com sucesso.
    - 204 - *timeout* ou cancelamento por término do servidor.
    - 400 - pedido incorrecto (e.g. formato inválido).
    - 404 - fila não existente.
    - 405 - operação não existente.
    - 500 - erro de servidor.
    - 503 - serviço indisponível (e.g. em processo de *shutdown*).

- A implementação do servidor deve também ter em conta os seguintes aspectos:
    - Utilização do modelo TAP (*Task based Asynchronous Pattern*), evitando o bloqueio de *threads*, nomeadamente na recepção de mensagens, operações **`TRANSFER`** e **`TAKE`**. Tire partido dos métodos assíncronos disponíveis a partir do C# 5.0.
    - Limitação, por concepção, do número máximo de pedidos que o servidor pode processar simultaneamente.
    - Funcionalidade de registo suportada por uma *thread* de baixa prioridade (*logger thread*) criada para o efeito. As mensagens com os relatórios devem ser passadas das *threads* que servem pedidos (produtoras) para a *logger thread* usando um mecanismo de comunicação que minimize o tempo de bloqueio das *threads* produtoras. A funcionalidade de registo deve ter o mínimo de influência no tempo de serviço, admitindo-se inclusivamente a possibilidade de ignorar relatórios.
    - Terminação ordeira do servidor, finalizando os pedidos que já iniciarem processamento com o *status code* 204.
- Implemente um cliente de exemplo para demonstrar o correcto funcionamento do servidor.
- Tenha em consideração o exemplo presente no GitHub.

Data limite de entrega: 29 de Junho de 2020

ISEL, 9 de Junho de 2020