Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação Concorrente, Inverno de 2020/2021

Terceira Série de Exercícios

1. Realize na plataforma .NET um servidor com interface TCP para acesso a *transfer queues* usado o modelo de interação pedido-resposta.
   - Os pedidos e as respostas são objetos JSON.
   - Os pedidos têm 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`**.
     - **`SHUTDOWN`** - inicia o processo de encerramento do servidor, ficando à espera que este esteja concluído. O tempo máximo de espera, em milisegundos, é definido no *header* **`timeout`**. O processo de encerramento deve esperar que todas as filas estejam vazias. Para tal deve aceitar novos pedidos de leitura, contudo não deve aceitar novos pedidos de inserção. Os pedidos de inserção síncrona (*transfer*) devem ser cancelados.
   - O servidor usa os seguintes códigos de **`status`**:
     - 200 - operação realizada com sucesso.
     - 204 - *timeout*.
     - 400 - pedido incorreto (e.g. formato inválido).
     - 404 - fila não existente.
     - 405 - operação não existente.
     - 500 - erro de servidor.
     - 503 - serviço indisponível (e.g. em processo de encerramento).

- 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 nas operações **`TRANSFER`** e **`TAKE`**, e na espera pelo encerramento do servidor. 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.
- Implemente um cliente de exemplo para demonstrar o correto funcionamento do servidor.
- Tenha em consideração o exemplo presente no GitHub.

2. Considere o seguinte método:

```
public R Compute(T[] elems, R initial)  {
  R acc = initial;
  foreach (var elem in elems)
    acc = C(B(A(elem)), acc);
  return acc;
}
```

Os métodos **`A`** e **`B`** são funções sem efeitos colaterais e passíveis de múltiplas execuções em paralelo. Ambos os métodos recebem **`T`** e retornam **`T`**. O método **`C`** realiza uma operação não associativa.

a) Realize uma versão assíncrona do método **`Compute`** seguindo o padrão TAP (*Task-based Asynchronous Pattern*) usando os métodos assíncronos do C# e/ou a funcionalidades disponíveis na TPL. Assuma que tem disponível as versões TAP dos métodos **`A`**, **`B`**, e **`C`**. Tire partido do paralelismo potencial existente.

b) Realize a variante da versão assíncrona do método implementado no exercício da alínea anterior, onde o número de operações **`A`** pendentes (i.e., iniciadas e não concluídas) está limitado a 10. Assuma que tem disponível uma implementação do semáforo com interface assíncrona realizado nas aulas.

Data limite de entrega: 23 de Janeiro de 2021

Esta série de exercícios pode ser realizada em grupos de até três alunos.

ISEL, 9 de Dezembro de 2020