LETI 2019/20, Repescagem do 1º Teste de Sistemas Distribuídos

31 de janeiro de 2020

Responda no enunciado, usando apenas o espaço fornecido. Identifique todas as folhas.
Uma resposta errada numa escolha múltipla com N opções desconta 1/(N-1) do valor da pergunta.
Duração da prova: **1h15m**

Grupo I [7 valores]

Considere o seguinte programa em Java:

```
public class Car implements Serializable {
	private String make;
	private int mileage;
	// ...
}

public interface ICarManager extends Remote {

	// Looks for an available car that is parked within a specified radius of the
	// coordinates supplied as arguments;
	// if more than one car is available, the nearest is returned
	Car findNearestFreeCar(float latitudeCoordinates, float longitudeCoordinates,
		int maxRadiusMeters) throws RemoteException, NoAvailableCar;
...
```

1) Considere a interface acima, procure escrevê-la numa IDL de um RPC genérico.
   a) [1,0v] Em primeiro lugar, defina as estruturas de dados necessárias para os parâmetros dos RPC do de modo a obter o mesmo funcionamento desta interface em Java.

   b) [1,0v] Complete agora a IDL com a operação `findNearestFreeCar`. Inclua todos os elementos que achar necessários, propondo os que não conseguir obter diretamente da interface acima.

2) Considere que o RPC referido usa apenas o protocolo UDP.
A semântica do RPC utilizada é "pelo-menos-uma-vez" (*at-least-once*). O cliente efetua a chamada a `findNearestFreeCar`. Indique no campo "Resultado para o cliente" se o RPC retorna um resultado válido ou inválido (`RPC_SUCCESS` ou `RPC_ERROR`, respetivamente) e indique no segundo campo o número de vezes que a função é executada no servidor.

a) [0,7v] A mensagem de invocação inicial é perdida

Resultado para o cliente	Número de vezes executada no servidor

b) [0,7v] A mensagem de resposta do servidor é perdida pela rede 2 vezes

Resultado para o cliente	Número de vezes executada no servidor

c) [0,7v] As mensagens de resposta do servidor são perdidas pela rede excedendo o *timeout* do cliente. O *timeout* do cliente é 1s e a repetição demora 100 ms.

Resultado para o cliente	Número de vezes executada no servidor

3) [1,1v] Se a semântica fosse "no-máximo-uma-vez" (*at-most-once*) qual (ou quais) dos três quadros anteriores seria diferente? Identifique claramente qual ou quais e justifique.

4) Considere que o RPC tinha uma política de "receptor-converte" (*receiver makes it right*) que não é a do gRPC. Suponha que o servidor tem uma arquitetura de dados TIPO I e tem um cliente A com a mesma arquitetura (TIPO I) e um cliente B com uma arquitetura diferente TIPO II.

a) [0,9v] Represente o que teria de enviar numa invocação do serviço `findNearestFreeCar` quando o cliente A invoca o serviço (arbitre os valores dos parâmetros).

b) [0,9v] Represente o mesmo quando o cliente B invoca o serviço (arbitre os valores dos parâmetros).

Grupo II [7 valores]

Considere novamente **o programa em Java do Grupo I**. Mantendo o serviço descrito pretende-se agora que o sistema use RMI e as funcionalidades deste sistema de objetos distribuídos.

1) [0,9v] Defina a interface `ICar` com os seguintes requisitos:
   - Os objetos da classe correspondente ficam residentes nos sistemas informáticos dos automóveis, mantendo-se ativos e enviando periodicamente a localização GPS do veículo.
   - O objeto tem, entre outros, um método:

     ```
     CarReservation reserve(int userId) throws CarNotAvailable
     ```

   Este método deverá poder ser executado remotamente pelo cliente para reservar um carro. O parâmetro de resposta é uma reserva (ignore, por agora, a informação que a compõe) para reservar um carro durante 1 hora que, quando apresentado ao sistema do carro, o permite usar.

```
public interface ICar
```

2) [1,0v] Modifique agora a interface que centraliza a informação sobre os carros: `ICarManager`.

As principais diferenças a incluir são:
- O método `findNearestFreeCar` deve retornar o objeto que representa o carro com o tipo programado na alínea anterior
- Deve haver um método para o carro se registar no sistema indicando os elementos fabricante e quilometragem.
- Ignore outros métodos que porventura seriam necessários para o sistema ter funcionalidade completa

3) Suponha que o `CarReservation` deve conter `int pinCode, DigSign pinDigSign`
   a) [0,8v] Na lógica desta aplicação, este objeto deveria ser passado por valor ou por referência? Justifique.

   b) [0,8v] Que diferença existe a nível da programação para especificar a decisão da alínea anterior?

4) O cliente do sistema obtém de acordo com as alíneas anteriores uma referência remota para um objeto, podemos supor neste caso o `car1234`.
   a) [0,7v] Comente a afirmação: "Para obter esta referência remota o objeto car1234 tem de registar-se no *RMI Registry* de outra forma não será possível encontrar a sua referência remota".

   b) [0,6v] **Quando** é criada a referência remota inicial para este objeto? Justifique.

   c) [0,5v] **Quem** é responsável por criá-la? Justifique.

   d) [0,5v] **Quando** o cliente recebe a referência remota para um veículo **quem lha transmite?**

   e) Suponha que o sistema implementa um *garbage collector* baseado na contagem de referências. Na situação descrita nesta alínea procure explicar:
      i) [0,6v] Que valor terá o contador de referências na máquina virtual onde se executa o objeto `car1234`?

      ii) [0,6v] Justifique a sua resposta detalhando os passos que permitem calcular esse valor.

Grupo III [6 valores]

Considere um sistema de encomendas eletrónicas baseado na tecnologia de gRPC.

O servidor, implementado em Java, recebe pedidos de encomenda através da operação `placeOrder` que recebe uma data, morada de origem e morada de destino e um ou mais códigos de produtos a encomendar.

1) [1,4v] Capturou-se na rede uma mensagem de um cliente, a caminho do servidor. Consegue inferir a linguagem de programação em que o cliente está programado? Justifique.

2) Considere agora que o cliente foi desenvolvido em **Java** com gRPC.
   a) [1,4v] Escreva em pseudo-código as classes `MoradaPostal` e `Encomenda`, geradas pela para transporte de dados.

   b) [1,8v] Escreva em pseudo-código um programa cliente gRPC que faz uma encomenda.

   c) [1,4v] Quando o servidor recebe uma data na mensagem de encomenda, por exemplo, o dia 12 de Junho de 2019, como é evitada a ambiguidade com o dia 6 de Dezembro de 2019? Que componente da arquitetura do gRPC é responsável por resolver este problema?