LETI 2019/20, 1º Teste de Sistemas Distribuídos
23 de outubro de 2019

Identifique todas as folhas. Responda no enunciado, usando apenas o espaço fornecido.

Nas perguntas de escolha múltipla assinaladas com EM existe apenas uma resposta certa. Uma resposta errada desconta 1/(N-1) do valor de uma pergunta com N opções de resposta.

Duração da prova: **1h15m**

Grupo I – RPC [7 valores]

Considere um RPC que suporta uma aplicação de aluguer de bicicletas numa cidade.
O RPC comunica com o protocolo **UDP** e usa um formato canónico para o seu sistema de tipos de dados.
São implementadas as seguintes operações:

```
Program RENT_A_BIKE {
  TAKE_BIKE (string userId, string bikeId) ! NotEnoughBalance
  RETURN_BIKE (string userId, string bikeId) -> (integer userAccountBalance)
}
```

A operação **`TAKE_BIKE`** destranca uma bicicleta para o utilizador. Se o utilizador tiver pelo menos 30 pontos na sua conta, a bicicleta é destrancada. Caso contrário, é devolvido o erro **`NotEnoughBalance`**.

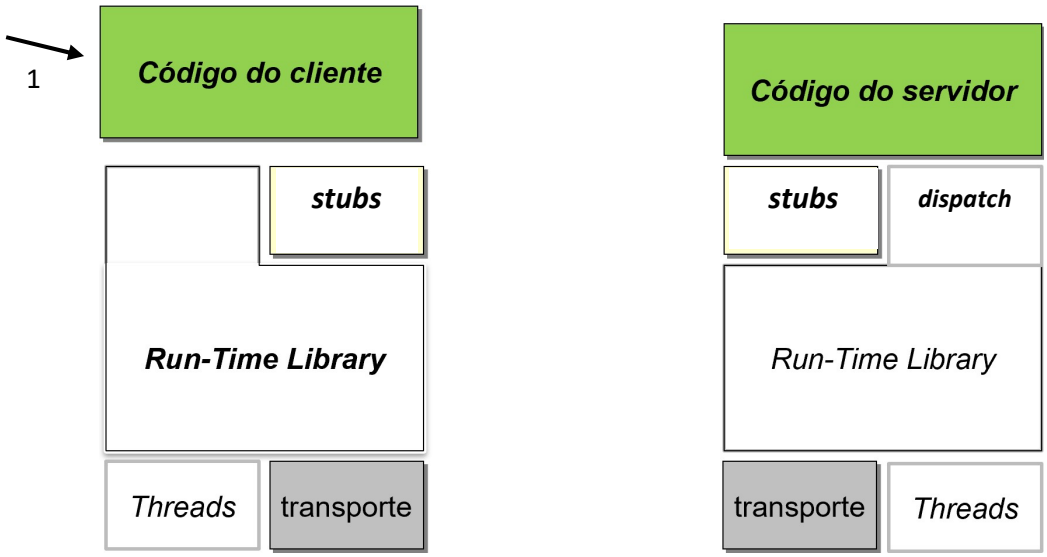
A operação **`RETURN_BIKE`** devolve a bicicleta ao sistema, decrementa os pontos na conta do utilizador e retorna o saldo após a utilização.

1) [0,9v] Indique uma vantagem potencial na programação por se usar RPC comparando com o uso direto de *sockets* UDP. Justifique a vantagem apontada.

2) Considere a seguinte chamada do procedimento remoto
   `TAKE_BIKE("joana", "LX35")`.

   a) [1,0v] Que informação é enviada pelo cliente no conteúdo do datagrama UDP do pedido? Desenhe uma representação do datagrama. Identifique e descreva cada campo numa legenda.

b) [0,7v] A figura seguinte mostra a arquitetura do RPC com o cliente à esquerda e o servidor à direita. Assinale na figura, com **setas numeradas**, qual o percurso de uma chamada à operação **`TAKE_BIKE`** iniciada no "código do cliente" até que seja executado o procedimento no "código do servidor".



c) [EM 0,7v] A rotina de adaptação (*stub*) do lado do cliente:

A. Faz a conversão dos tipos de dados do cliente para o formato canónico.
B. Define qual o protocolo de transporte com que o pedido pode ser recebido.
C. Cria uma *thread* para enviar o pedido e aguardar resposta.
D. Cria o *socket* para enviar o pedido.
E. Define os tipos de dados de interfaces das operações RPC.
F. Identifica qual o procedimento remoto que está a ser chamado.

A

d) [EM 0,7v] A rotina de despacho (*dispatch*) do lado do servidor:

A. Faz a conversão dos tipos de dados do formato canónico para o servidor.
B. Faz a conversão dos tipos de dados usados no cliente para o formato canónico usado na rede.
C. Define qual o protocolo de transporte a utilizar.
D. Define os tipos de dados de interfaces das operações RPC.
E. Cria uma *thread* para tratar o pedido.
F. Identifica qual o procedimento que está a ser chamado e chama o *stub* respetivo.

F

3) Considere que existem duas contas de utilizador, **`joana`** e **`maria`**, com saldo inicial de 60 e 30 pontos, respetivamente. Para terminar a utilização da bicicleta, o cliente chama: **`RETURN_BIKE("joana", "LX35")`** que vai decrementar 20 pontos no saldo do servidor.

a) [1,0v] Qual é o saldo final de pontos no servidor, nas seguintes situações?
Analise a situação em cada linha partindo do estado inicial.

Semântica configurada	**Acontecimentos diferentes do normal**	**Saldo final de "joana" no servidor**
Talvez	Resposta perde-se 1 vez.	40
Pelo-menos-uma-vez	Pedido perde-se 1 vez. Resposta perde-se 2 vezes.	0
No-máximo-uma-vez	Pedido perde-se 2 vezes. Resposta perde-se 1 vez.	40

b) [1,0v] Considera a operação **`RETURN_BIKE`** idempotente? Justifique.

c) [1,0v] Qual é a semântica de execução garantida pelo RPC caso se troque o protocolo UDP por TCP? Justifique detalhadamente.

Grupo II – RMI [7 valores]

Considere o seguinte definição de um programa remoto:

```
import java.rmi*;

public interface Account extends Remote {
    float debit(float amount) throws RemoteException, InsufficientFundsException;
    float credit(float amount) throws RemoteException;
}

public interface AccountList extends Remote {
    Account getAccount(int id) throws RemoteException;
}
```

1) A definição da interface herda da classe `Remote`. Indique se concorda ou discorda das seguintes afirmações e justifique.
   a. [0,6v] "Esta interface só pode ser usada por clientes em máquina remotas. Não pode ser invocada localmente".

   b. [0,6v] "A única notação que é necessário para um método ser invocável remotamente é a sua interface herdar de `Remote`".

Considere agora o programa de um cliente remoto que invoca objectos definidos de acordo com a interface anterior.

```
import java.rmi*;
import java.rmi.server.*;

public class AccountListClient {
1   public static void main(String args[]){
2       AccountList acList = null;
3       try {
4           acList = (AccountList) Naming.lookup("//bank.com:1099/AccountList");
5           Account a = acList.getAccount(2);
6           a.debit(1000);
7       } catch(Exception e) { System.out.println(e.getMessage()); }
```

Considere o cliente e em particular a tabela de referências remotas e as classes carregadas.

2) Considere a linha número 4.
   a. [0,4v] Descreva a operação executada nessa linha.

   b. [0,6v] Que modificação se efectua na tabela de referências remotas do cliente? Justifique.

3) Considere a linha número 5.
   a. [0,4v] Em que classe é invocado o método getAccount no cliente?

   O método getAccount é invocado na proxy da classe AccountList.

   b. [0,4v] Em que classe é realmente executado e em que máquina se executa essa classe?

4) [0,5v] Considere a linha número 6. O que acontece se a rede não estiver a funcionar corretamente no momento da invocação? Tenha em conta as características da comunicação com RMI que usa TCP.

5) Considerando o nome `"//bank.com:1099/AccountList"` usado na linha número 4.
   a. [0,6v] Que programa está à escuta no porto 1099? Qual é a sua função?

   RMI Registry. Permite o registo e pesquisa de objetos remotos associados a nomes.

   b. [0,8v] O nome é puro? Justifique.

   c. [1,0v] Indique pelo menos 3 autoridades associadas a este nome.
   Para cada autoridade, indique a componente do nome gerida por essa autoridade.

   Dentro do nome DNS, existem as autoridades dos servidores de nomes.

   O sistema operativo gere os portos, incluindo o 1099 associado ao processo do RMI Registry.

   O restante, nome do objeto RMI, a autoridade é o RMI Registry.

6) O mecanismo de Java RMI tem inerentemente um recuperador de memória (GC - *Garbage Collector*). Por omissão este recuperador usa contagem de referências.
   a. [0,4v] Na linha 4 o que deve a máquina do cliente indicar ao GC local do servidor para garantir o correcto funcionamento deste?

   b. [0,7v] Comparando contagem de referências com *leases*, qual dos dois processos de recuperação é mais vantajoso caso a rede tenha falhas ocasionais? Justifique.

Grupo III – gRPC [6 valores]

1) Considere a arquitetura do gRPC estudado nas aulas.
   a) [1,0v] Assinale com X qual o código que desempenha a tarefa indicada em cada linha da seguinte tabela. Apenas deve indicar um X por linha.

Tarefa	**Código**			
	protobuf	gerado por protoc	parte da biblioteca gRPC	feito à medida para a aplicação
Definir estrutura das mensagens				
Localizar o porto do servidor				
Estabelecer canal de comunicação				
Criar *sockets*				
Para um dado pedido:				
Indicar argumentos a enviar				
Criar mensagem de pedido				
Converter e serializar parâmetros				
Enviar, reenviar, filtrar duplicados				

   b) [EM 0,6v] O protocolo de comunicação usado por omissão no gRPC é:

A. SMTP
B. FTP
C. HTTP 1.1
D. HTTP/2

D

2) Considere a seguinte definição na linguagem **protobuf** (*Protocol Buffers*) usada no **gRPC** do jogo do galo (Tic-Tac-Toe):

```
message GetBoardRequest { }
message GetBoardResponse { string board = 1; }

message PlayRequest {
  uint32 row = 1;
  uint32 column = 2;
  uint32 player = 3;
}
message PlayResponse {
  enum PlayResult {
    UNKNOWN = 0;
    OUT_OF_BOUNDS = 1;
    SQUARE_TAKEN = 2;
    WRONG_TURN = 3;
    GAME_FINISHED = 4;
    SUCCESS = 5;
  };
  PlayResult result = 1;
}
```

```
message CheckWinnerRequest { }
message CheckWinnerResponse { sint32 result = 1; }

service TTT {
  rpc GetBoard(GetBoardRequest) returns (GetBoardResponse);
  rpc Play(PlayRequest) returns (PlayResponse);
  rpc CheckWinner(CheckWinnerRequest) returns (CheckWinnerResponse);
}
```

a. [1,0v] O que indica a notação "`= número;`" usada no protobuf?

b. [1,2v] Que informação é devolvida pelo servidor numa resposta à operação `Play` quando a jogada teve sucesso. Desenhe um diagrama com a estrutura da mensagem de resposta e indique a dimensão aproximada de cada campo em octetos. Se necessário, considere que um caracter ocupa 1 octeto.

Dois inteiros (4 octetos cada):

1 (result)	5 (valor correspondente ao enumerado 'success')

c. [1,0v] Considera o protocolo de apresentação de dados usada pelo gRPC explícito ou implícito? Justifique.

d. [1,2v] Escreva pseudo-código Java de um programa cliente que use a biblioteca gRPC para se ligar ao servidor recebido nos argumentos, faça uma jogada na linha 1, coluna 2 do tabuleiro em nome do jogador 0, e que depois obtenha o tabuleiro de jogo e imprima o novo tabuleiro na consola.

```
public static void main(String[] args) throws Exception {
    final String hostPort = args[0] + ":" + args[1];
    Channel channel = null;

    {
        channel = ChannelBuilder.forTarget(hostPort).build();

        TTTStub stub = new BlockingStub(channel);

        PlayRequest playReq = PlayRequestBuilder.newBuilder().setRow(1).setCol(2).
            setPlayer(0).build();
        PlayResponse playRes = stub.play(playReq);

        GetBoardRequest boardReq = GetBoardRequestBuilder().newBuilder().build();
        GetBoardResponse boardRes = stub.getBoard(boardReq);

        System.out.println(boardRes.getBoard());

    } finally {
        if (channel != null)
            channel.close();
    }
}
```