LEIC/LETI 2017/18, 1º Teste de Sistemas Distribuídos, 3 de abril de 2018

Responda no enunciado, usando apenas o espaço fornecido. Identifique todas as folhas.
Uma resposta errada numa escolha múltipla desconta 1/N do valor da pergunta (sendo N o número de opções).
Duração da prova: 1h30m

Grupo I [7v]

1) Considere um projeto RPC que gere um sistema de estações de *bike-sharing*, em que cada estação de entrega/recolha de bicicletas tem um servidor com a seguinte interface remota:

```
struct info {
  string stationid<20>;
  int coord_x;
  int coord_y;
  int capacity;
  int free_docks;
  int total_gets;
  int total_returns;
};
```

```
program BINA {
  version BINA_VERS {
    /* se existirem bicicletas na estação, remove uma */
    void getbina(void) = 0;
    /* se existirem vagas livres na estação, junta uma bicicleta */
    void returnbina(void) = 1;
    /* lê o estado e estatísticas da estação */
    info getinfo(void) = 2;
  } = 1;
} = 1009823;
```

Usando o compilador **rpcgen**, foi gerado um exemplo de um programa cliente. De seguida apresenta-se um excerto desse programa.

```
1  CLIENT *clnt;
2  void  *result_1;
3  char *getbina_1_arg;
4  void  *result_2;
5  char *returnbina_1_arg;
6  info  *result_3;
7  char *getinfo_1_arg;

8  clnt =
   clnt_create (host, BINA, BINA_VERS, "udp");
9  if (clnt == NULL) {
10  clnt_pcreateerror (host);
11  exit (1);
12 }
```

```
13 result_1 =
           getbina_1((void*)&getbina_1_arg, clnt);
14 if (result_1 == (void *) NULL) {
15  clnt_perror (clnt, "call failed");
16 }
17 result_2 =
     returnbina_1((void*)&returnbina_1_arg, clnt);
19 if (result_2 == (void *) NULL) {
20   clnt_perror (clnt, "call failed");
21 }
22 result_3 =
           getinfo_1((void*)&getinfo_1_arg, clnt);
23 if (result_3 == (info *) NULL) {
24  clnt_perror (clnt, "call failed");
25 }
```

a) [2,0v] Pretende-se desenvolver um programa cliente que retira uma bicicleta (chamando `getbina_1`) da estação A e seguidamente coloca a bicicleta (chamando `returnbina_1`) na estação B.
Componha uma variante do programa cliente acima que cumpra estes novos requisitos funcionais. Assuma que os servidores que gerem cada estação estão nas máquinas `estacaoA.tecnico.pt` e `estacaoB.tecnico.pt` (respetivamente).

b) [0,7v] Indique uma função *stub* chamada no programa cliente inicial (nome da função ou linha).

c) [0,8v] Uma das funções auxiliares gerada (que não surge no excerto acima) chama-se `xdr_info`. Para que serve esta função?

Esta função, gerada pelo compilador rpcgen, trata do *marshalling/unmarshalling* da estrutura do tipo *info*,
seguindo o protocolo de apresentação XDR. É chamada sempre que uma estrutura deste tipo é enviada
numa mensagem como retorno da função remota *getinfo*.

d) [1,4v] Entre as funções remotas, identifique uma função *idempotente* e uma *não idempotente*. Justifique.

Função idempotente:
Função não idempotente:

2) Assuma agora um outro sistema cliente-servidor, suportado por RPC, em que:
i) o canal de comunicação pode ocasionalmente **perder** ou **duplicar** mensagens; e
ii) o servidor **nunca** falha.

Assuma que um cliente chamou uma função remota; no entanto, o programa cliente recebeu um retorno de erro RPC da função.
Para cada semântica, indique quantas vezes a função remota se pode ter executado no servidor e justifique.

a) [0,7v] RPC implementa semântica Talvez:

Pode ter executado: zero vezes ◯ uma vez ◯ mais que uma vez ◯
Justificação:

b) [0,7v] RPC implementa semântica Pelo-menos-uma-vez:

Pode ter executado: zero vezes ◯ uma vez ◯ mais que uma vez ◯
Justificação:

c) [0,7v] RPC implementa semântica No-máximo-uma-vez:

Pode ter executado: zero vezes ◯ uma vez ◯ mais que uma vez ◯
Justificação:

Grupo II [6v]

Usando Java RMI, pretende-se implementar uma tabela de dispersão (*hash table*) distribuída. Tal como uma tabela de dispersão vulgar, esta estrutura de dados mantém um vetor de `N` entradas, e cada entrada referencia uma instância do tipo `ElementList`. Sendo distribuída, as `N` listas referenciadas podem estar em diferentes máquinas numa rede.

Nas alíneas seguintes, considere que a interface `ElementList` é definida como se segue:

```
public interface ElementList extends Remote {
    /* Procura na lista pelo elemento com a chave indicada,
       retornando-o caso exista ou null caso contrário */
    Element get(int key) throws RemoteException;

    /* Adiciona elemento à lista; caso já exista elemento
       com a mesma chave, substitui o elemento anterior */
    void put(Element e, int key) throws RemoteException;
}
```

1) [1,0v] Complete o método construtor da classe que implementa a tabela de dispersão distribuída. Assuma que as N listas (do tipo `ElementList`) já estão instanciadas em máquinas remotas e registadas num *RMI registry*. Cada lista remota tem um nome "`//sd.tecnico.pt/lista`" + *k*, em que *k* é um inteiro entre 0 e N-1. Assim sendo, o método construtor resume-se a preencher o vetor com as referências remotas para cada uma das N listas.
   Nota: não necessita apanhar/tratar as exceções RemoteException, pois os métodos da classe deixam-na passar (fazem *throws*).

```
public class DistHashTable {
  private Vector<ElementList> vetor;
  private final int N;

  public DistHashTable(int N) throws RemoteException {
    this.N = N;
    vetor = new Vector<ElementList>(N);









  }
```

2) [1,0v] Complete agora os métodos `get` e `put` da mesma classe.
   Recorra ao método *hash* (já implementado abaixo) para determinar qual a posição do vetor que deve ser acedida para uma determinada chave. Tal como na alínea anterior, não necessita apanhar/tratar as exceções RemoteException.

```
private int hash(int key) {
    if (vetor) return (key % N);
    else return -1;
}
/* (continua na próxima página) */
```

```
public Element get(int key) throws RemoteException{









}

public void put(Element e, int key)  throws RemoteException {









}
```

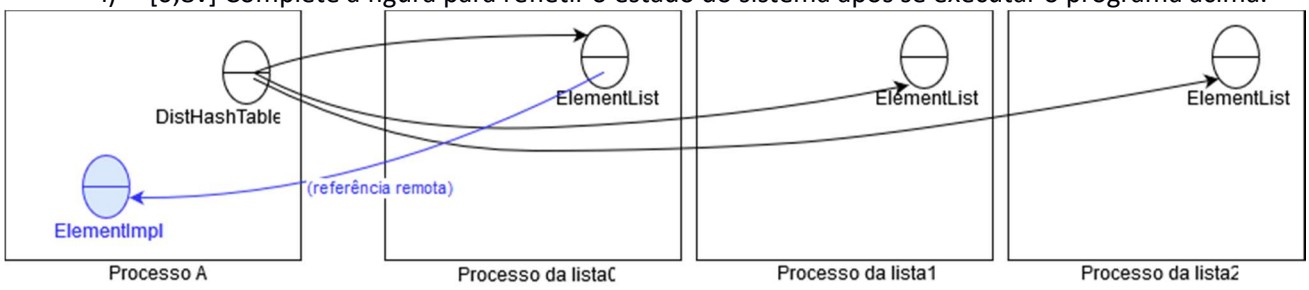
3) Considere o seguinte programa que instancia uma `DistHashTable` que referencia 3 listas remotas já existentes.

```
DistHashTable t = new DistHashTable(3);
Element e = new ElementImpl();
t.put(e, 1230);
```

Assuma que cada uma das listas remotas está **vazia** no início do programa acima e existe num processo distinto. Assuma também que o novo elemento `1230` é adicionado à `lista0` (ou seja, hash(1230)=0).

a) Assumindo que a interface `Element` herda da interface `Remote`:

i) [0,8v] Complete a figura para refletir o estado do sistema após se executar o programa acima.

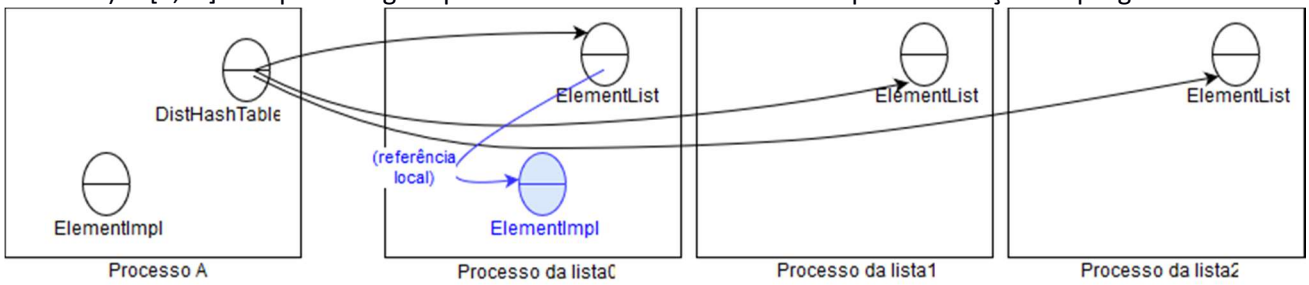


ii) [0,4v] Neste caso, qual a classe da instância que é criada no processo *lista0 (*que contém a lista onde o novo elemento é inserido)?

Classe proxy da interface Element

b) Assuma agora que `Element` herda de `Serializable` e **não** herda da interface `Remote`.

i) [0,8v] Complete a figura para refletir o estado do sistema após a execução do programa acima.



ii) [0,4v] Neste caso, qual a classe da instância que é criada no processo que contém a lista onde o novo elemento é inserido?

Classe ElementImpl

4) Considere o nome "`//sd.tecnico.pt/lista4`".

a) [0,9v] Resolver este nome implica recorrer a diferentes serviços de nomes.
Indique **dois** desses serviços de nomes, referindo qual a componente do nome acima que cada serviço de nomes recebe.

b) [0,7v] "Este nome é hierárquico". Concorda com a afirmação? Justifique.

Grupo III [7]

1) Considere a seguinte mensagem de um pedido SOAP à Amazon Product API para pesquisa de produtos a partir de palavras-chave:

```
1.  <?xml version="1.0" encoding="UTF-8" ?>
2.  <soapenv:Envelope
3.      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
4.      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5.      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
6.    <soapenv:Body>
7.     <ItemSearch xmlns="http://webservices.amazon.com/AWSECSCommerce/onca/soap">
8.       <AWSAccessKeyId>AKIA2IOSF2DNN7EX</AWSAccessKeyId>
9.       <Request>
10.        <SearchIndex>Books</SearchIndex>
11.        <Keywords>Harry%20Potter</Keywords>
12.      </Request>
13.    </ItemSearch>
14.   </soapenv:Body>
15.  </soapenv:Envelope>
```

a) [1,0v] Considera que a mensagem tem codificação de dados *implícita* ou *explícita*? Justifique.

A mensagem tem codificação de dados explícita pois os dados são acompanhados de delimitadores (*tags*) que subdividem a informação e descrevem a sua estrutura.

b) [1,0v] Observando apenas a mensagem SOAP acima, consegue inferir qual o protocolo de comunicação utilizado para enviar a mensagem? Justifique.

Não é possível inferir o protocolo de comunicação utilizado porque as mensagens SOAP podem ser usadas genericamente sobre qualquer protocolo. Por exemplo, HTTP ou SMTP ou outros.

c) [1,2v] Construa uma possível mensagem de resposta ao pedido acima, correspondente à situação em que o **AWSAccessKeyId** foi rejeitado. Siga as convenções ditadas pelo SOAP para casos de erro (pode simplificar a sintaxe, desde que a estrutura da resposta seja apresentada de forma clara).

```
<soap:envelope>
  <soap:body>
    <soap:fault>
      <faultcode>soap:server</faultcode>
      <faultstring>AWSAccessKeyId is not valid!</faultstring>
      <faultactor>http://amazon.com</faultactor>
      <detail />
    </soap:fault>
  </soap:body>
</soap:envelope>
```

d) Considere agora os seguintes excertos do documento WSDL disponibilizado pela Amazon:

```
1359. <portType name="AWSECommerceServicePortType">
1360.    <operation name="ItemSearch">
1361.       <input message="tns:ItemSearchRequestMsg"/>
1362.       <output message="tns:ItemSearchResponseMsg"/>
1363.    </operation>

1397. <binding name="AWSECommerceServiceBinding" type="tns:AWSECommerceServicePortType">
1398.    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
1399.    <operation name="ItemSearch">
1400.       <soap:operation soapAction="http://soap.amazon.com/ItemSearch"/>
1401.       <input>
1402.          <soap:body use="literal"/>
1403.       </input>
1404.       <output>
1405.          <soap:body use="literal"/>
1406.       </output>
1407.    </operation>
```

i) [0,8v] Qual a diferença entre a secção que começa na linha 1359 e a que começa na linha 1397?

A secção da linha 1359 define o *portType* que corresponde à definição da interface abstrata do serviço onde são definidas as operações, as suas entradas, saídas e erros. A secção da linha 1397 define o *binding* que indica o formato concreto das mensagens e o protocolo de transporte a utilizar nas invocações.

ii) [0,6v] Indique o número da linha do segundo excerto acima que considera mais relevante. Justifique a sua escolha.

Linha:
Justificação:

e) Atualmente, a Amazon disponibiliza informação sobre este Web Service apenas na sua página de documentação em formato HTML.

i) [0,8v] Indique um argumento técnico a favor da utilização de UDDI como alternativa.

ii) [0,8v] O registo de um serviço no UDDI inclui os seguintes itens de informação:

Name: AWSECommerceServicePort
Address: http://webservices.amazon.com/onca/soap?Service=AWSECommerceService

Caracterize as propriedades destes nomes:
A. "*Name*" é local, "*Address*" é impuro e local.
B. "*Name*" é local, "*Address*" é puro e global
C. "*Name*" é global, "*Address*" é puro e global.
D. "*Name*" é local, "*Address*" é impuro e global.

D

iii) [0,8v] Que informação adicional pode ser guardada no registo do serviço no UDDI? Identifique dois itens de informação em concreto e dê um exemplo da sua utilização.

Item 1: Informação sobre a organização que presta o serviço
Item 2: Classificação do serviço usando uma taxonomia
Exemplo de utilização: O cliente pode pesquisar no UDDI a partir da organização, por exemplo, a Amazon,
e encontrar registos sobre os diferentes serviços disponibilizados por esta empresa.
O cliente pode alternativamente pesquisar por uma taxonomia, por exemplo, uma classificação geográfica
e encontrar registos de serviços disponibilizados na Europa.