Introduction to Distributed Systems WT 20/21

Assignment 2

Submission Deadline: Monday, 07.12.2020, 10:00

- *Submit the solution in PDF via Ilias (only one solution per group).*
- *Respect the submission guidelines (see Ilias).*

---

1 Parameter Passing and RMI [11 points]

a) In the lecture, we discussed three parameter passing approaches *Call-By-Value*, *Call-By-Reference*, and *Call-By-Copy/Restore*.

What is the output of the MAIN procedure, i.e., the value of the array a on line 12) if FACTORIAL is called within an RPC scenario by using...

i. [1 point] ...Call-By-Value?

ii. [1 point] ...Call-By-Reference?

iii. [1 point] ...Call-By-Copy/Restore (provide all possibilities)?

```
 1: procedure FACTORIAL(var $x, y$ : array[5], var $i$ : integer)
 2:     var $bound, j$ : integer;
 3:     $bound \leftarrow x[i] - 1$;
 4:     for $j : bound \rightarrow 1$ do
 5:         $y[i] \leftarrow y[i] * j$;
 6:     end for
 7: end procedure
 8: procedure MAIN
 9:     var $a$ : array[5];
10:     $a \leftarrow [0, 2, 8, 4, 8]$;
11:     FACTORIAL($a$,$a$, 3);
12:     print $a$;
13: end procedure
```

b) [2 points] Listings 1 to 4 describe a RMI application, with some mistakes in the code. Write down the file name, line, and solution for each mistake. Assume all imports are correct (they have been omitted for readability reasons).

c) [3 points] Assuming all mistakes are corrected, what is the result of the three print statements in Listing 4?

1. : [________________ , ________________ , ________________]
2. : [________________ , ________________ , ________________]
3. : [________________ , ________________ , ________________]

Listing 1: Vector class

```
1 public class Vector {
  java.util.List<Integer> values =
        new  java.util.ArrayList<Integer>();
 public Vector (int x, int y, int z){
  this.values.of(x,y,z);
 }
 public String toString(){
  return this.values.toString();
 }
}
```

Listing 2: Remote Vector Interface

```
public interface RemoteVector {
 public Vector getVector() throws java.rmi.RemoteException ;
 public void vecAddition(RemoteVector rv)
        throws java.rmi.RemoteException ;
}
```

Listing 3: Server Application

```
public class Server extends java.rmi.server.UnicastRemoteObject
implements RemoteVector {

 private Vector myVector;
 public Server(int a, int b, int c) throws  java.rmi.RemoteException{
  super();
  this.myVector = new Vector(a,b,c);
 }

 public Vector getVector() throws java.rmi.RemoteException{
  return this.myVector;
 }

 public void vecAddition( RemoteVector rv)
  throws java.rmi.RemoteException{
  this.myVector = addition(this.myVector,rv.getVector());
 }

 public Vector addition( Vector v,  Vector b){
  for(int i=0; i< v.values.size(); i++) {
   b.values.set(i, v.values.get(i) + b.values.get(i));
  }
  return b;
 }

 public static void main ( String [] args ) throws Exception {
  Server serverVector1 = new Server(4,5,6);
  Server serverVector2 = new Server(1,2,3);
  java.rmi.Naming.connect("rmi://localhost/v1", serverVector1);
  java.rmi.Naming.rebind("rmi://localhost/v2", serverVector2);
 }
}
```

Listing 4: Client Application

```
import java.rmi.Naming;
public class Client {
 public static void main ( String [] args ) throws Exception {
  // Access the remote objects
  RemoteVector rb1 = java.rmi.Naming.lookup("rmi://localhost/v1");
  RemoteVector rb2 = java.rmi.Naming.lookup("rmi://localhost/v2");

  rb1.vecAddition(rb2);
  System.out.println ("1: " + rb1.getVector());
  System.out.println ("2: " + rb2.getVector());
  rb2.vecAddition(rb1);
  System.out.println ("3: " + rb2.getVector());
 }
}
```

d) [3 points] Name three reasons why Java RMI limits distribution transparency for the programmer?

2 Chord System [13 points]

The Chord system is an approach to implement a naming system for flat names. Consider the Chord system shown in Figure 1 for the following questions. It uses a 5-bit identifier space (0–31) and it contains the server nodes with IDs 1, 4, 8, 14, 22, 28.

a) [3 points] Provide the finger tables for all the nodes in the system.

b) [2 points] Node 4 wants to resolve the ID $e = 31$. Give the sequence of the nodes that are visited by this request, until e is found.

c) [2 points] Now, assume that Node 24 has just joined the network. What is the finger table of Node 24 after the join procedure has completed? What other finger tables in the network will actually change (do not compute the new contents for them)?

d) [3 points] In the following, we propose a modified Chord system in an attempt to reduce the storage requirements for the finger tables of all participating nodes while also reducing the number of lookups required to resolve a given key. In this new system, every node p stores a modified finger table with entries:

$$FT_p[i] = succ(p + 4^{i+1}), i \geq 1$$

Does the modified Chord system indeed result in reduced storage requirements while also reducing the number of lookup operations? Justify your answer.

e) [3 points] Instead of a single Chord ring, two logical rings should be used to increase fault tolerance. Each node N is inserted into both rings using different node IDs.

$ID_1(N)$ is chosen using an arbitrary hash function that maps the node's address to the identifier space $[0, 2^m - 1]$ of the first chord ring. The ID for the second ring is allocated based on the following relation: $ID_2(N) = 2^{m-1} + ID_1(N)$, having the same identifier space as $ID_1(N)$.

All insert and lookup operations are performed on both rings in parallel, using the same hash function for generating IDs of entities.

Does the insert operation guarantee that an entity's copy is stored on two different nodes? If no, provide an counter example, and propose a simple solution to solve the problem. *Assume that there are more than 2 nodes in the system.*

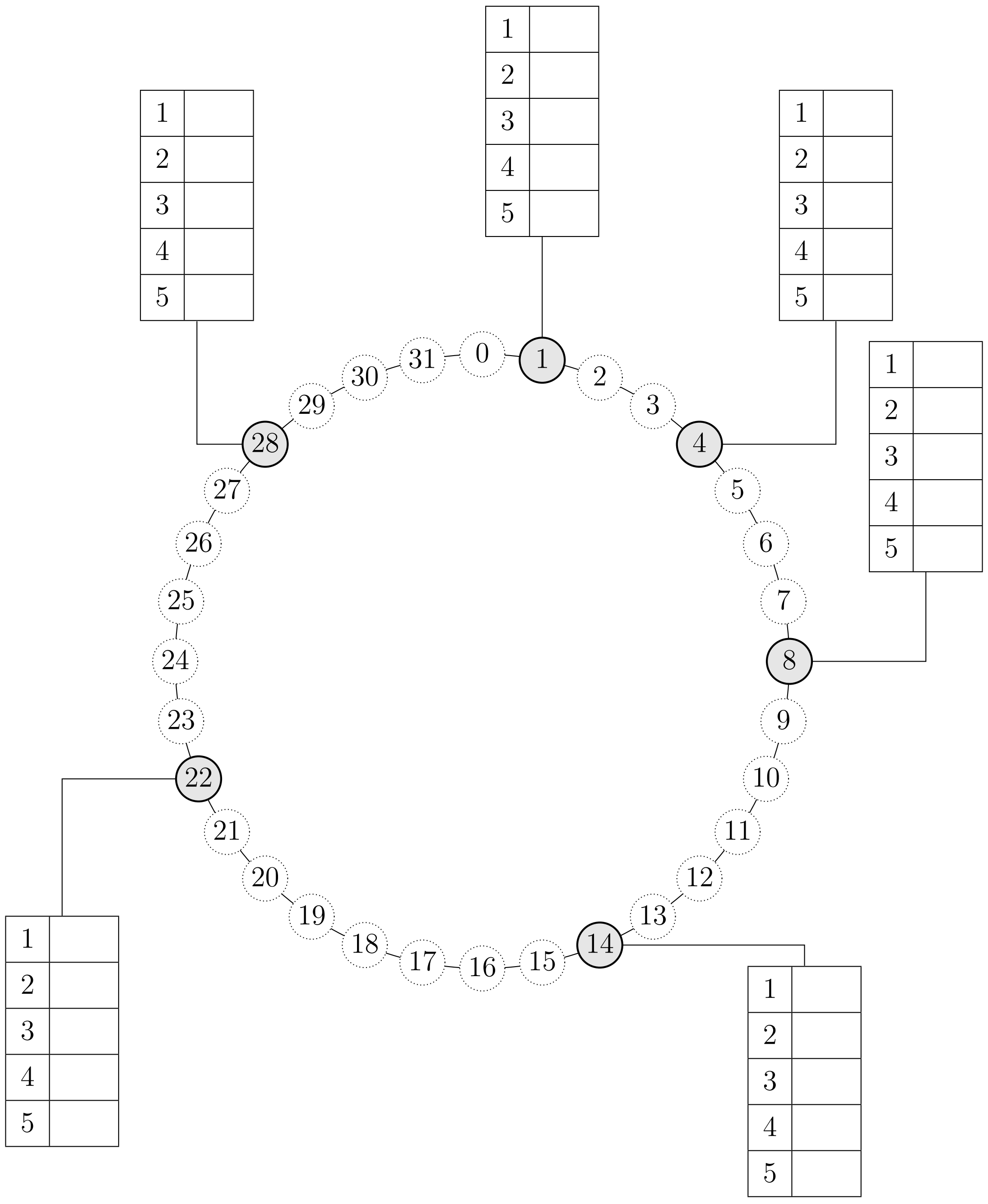

Figure 1: A 5-bit chord ring with blank finger tables.

3 Name Services [9 points]

Consider the (fictive) hierarchical name service given in Figure 2.

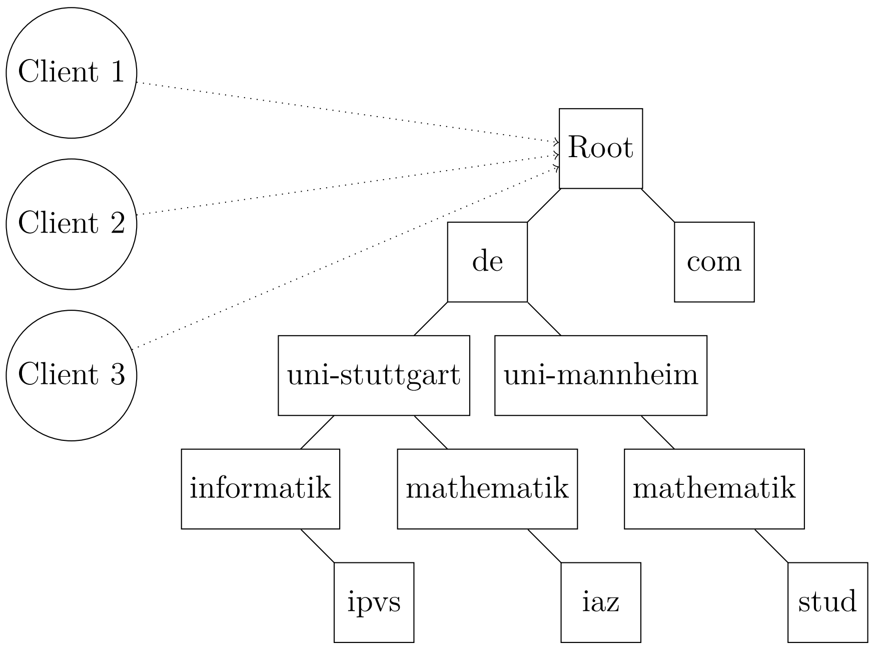

Figure 2: Name server hierarchy

Clients 1, 2, and 3 lookup the following names in the given order:

1. Client 1 looks up „`www.ipvs.informatik.uni-stuttgart.de`"
2. Client 2 looks up „`www.mathematik.uni-stuttgart.de`"
3. Client 3 looks up „`www.stud.mathematik.uni-mannheim.de`"

A name-server node can only contact other nodes that are directly connected by edges in the figure, unless they have an entry of other nodes in their cache.

a) [3 points] Assume the system does not use caching. State how many messages are exchanged in the system to resolve the lookups of the client.

	Iterative resolution	Recursive resolution
1. Lookup		
2. Lookup		
3. Lookup		

b) [3 points] Now, assume that server-side caching (cf. Slide 45, Chapter 4) is used in the system. Every name server node puts all information that is passing through in a cache. Assume the communication between the name server nodes requires 20ms, and the communication between a client and a name server node requires 40ms. Compute the number of messages exchanged in the system when using solely the iterative approach for each of the lookups. Afterwards, redo the same lookups by only using the recursive approach. Note, Caches are initially empty when using both approaches.

	Iterative resolution	Recursive resolution
1. Lookup		
2. Lookup		
3. Lookup		

c) From theory to practice: make yourself familiar to filter specific DNS entry types, reverse and iterative/non-recursive lookups with *dig* or *nslookup*.

i. [1 point] Lookup and provide the name server(s) of University of Stuttgart (domain uni-stuttgart.de)? Are they replicated?

ii. [2 points] Perform a manual iterative address resolution for the IPv4 address of www.uni-stuttgart.de. Start by querying one of the root name servers (root domain is ".") for the name servers of the next lower domain level, then this name server for the domain server of the next lower level, etc. Provide all name servers from the root server to the DNS server of the "leaf" domain. *Info: This does not work while being in the university network!*