Introduction to Distributed Systems
WT 19/20

Assignment 2

Submission Deadline: Monday, 18.11.2019, 08:00

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

1 Parameter Passing and RMI [9 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 11) 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?

```
1:  procedure INVERSE(var x, y :array[5], var i : integer)
2:      x[i] ← y[4 − i];
3:      if i < 4 then
4:          INVERSE(x,y,i+1);
5:      end if
6:  end procedure
7:  procedure MAIN
8:      var a :array[5];
9:      a ← [1, 2, 3, 4, 5];
10:     INVERSE(a,a,0);
11:     print a;
12: end procedure
```

b) [4 points] Java RMI uses two different approaches for parameter passing. Determine the result of the remote method invocations shown in the Java program on the next page (i.e., give the result printed on the system console). *You can find the documentation of the StringBuffer Class here.* [1].

Listing 1: Remote Interface

```
public interface RemoteBuffer extends java.rmi.Remote {
        public StringBuffer buffer() throws java.rmi.RemoteException;
        public void append_R( RemoteBuffer r)
                throws java.rmi.RemoteException;
        public void append_S( StringBuffer s)
                throws java.rmi.RemoteException;
}
```

[1] https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuffer.html

Listing 2: Server/Client Implementation

```
public class StringServer extends java.rmi.server.UnicastRemoteObject
                                          implements RemoteBuffer {
        private StringBuffer buffer;
        public StringServer(String s)
                        throws java.rmi.RemoteException {
                super();
                this.buffer = new StringBuffer(s);
        }
        public StringBuffer buffer() {
                return this.buffer;
        }
        public void append_R( RemoteBuffer r)
                        throws java.rmi.RemoteException {
                r.append_S(new StringBuffer("R"));
                this.buffer.append(r.buffer());
        }
        public void append_S( StringBuffer s)
                throws java.rmi.RemoteException {
                s.append("S");
                this.buffer.append(s);
        }
// ----------------- MAIN METHOD: -------------------
        public static void main(String[] args) throws Exception {
         StringServer server1 = new StringServer("A");
         StringServer server2 = new StringServer("B");
         java.rmi.Naming.bind("rmi://test.de:99/MyBuffer", server1);
         RemoteBuffer rb =
          (RemoteBuffer) java.rmi.Naming.lookup("rmi://test.de:99/MyBuffer");
//How are the parameters passed for the following RMI-calls?
         rb.append_S(server2.buffer());
         System.out.println("1: " + server2.buffer());
         rb.append_R(server2);
         System.out.println("2: " + server2.buffer());
         System.out.println("3: " + rb.buffer());
        }
}
```

c) [2 points] Both approaches for parameter passing in Java RMI cause an unreasonable runtime overhead in certain situations. For each approach, describe an example in which the RMI-call becomes very inefficient.

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 4, 11, 16, 19, 23, 26, and 31.

a) [4 points] Provide the finger tables for all the nodes in the system.

b) [1 point] Node 4 wants to resolve the ID $e = 22$. 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) [2 points] Now, assume that nodes 7 and 9 simultaneously join the setup in Fig. 1. Argue why the chord ring might become inconsistent, if no special synchronization measures were taken.

e) [2 points] To resolve the issue from the previous question, develop an algorithm to repair such inconsistencies. Fill in the missing section between lines 7 and 8.
Assume for simplicity, that when a node determines its successor, the successor responds with its ID and the ID of its predecessor.

```
1: Run the following on each node p:
2: while true do
3:     SLEEP(n seconds);
4:     q ← SUCC(p + 1).PRED;
5:     if q = p then
6:         continue;
7:     else
           ..... // TODO: complete
8:     end if
9: end while
```

f) [2 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.

Argue why an insert operation guarantees that an entity's copy is stored on two different nodes. *Assume that there are more than 2 nodes in the system.*

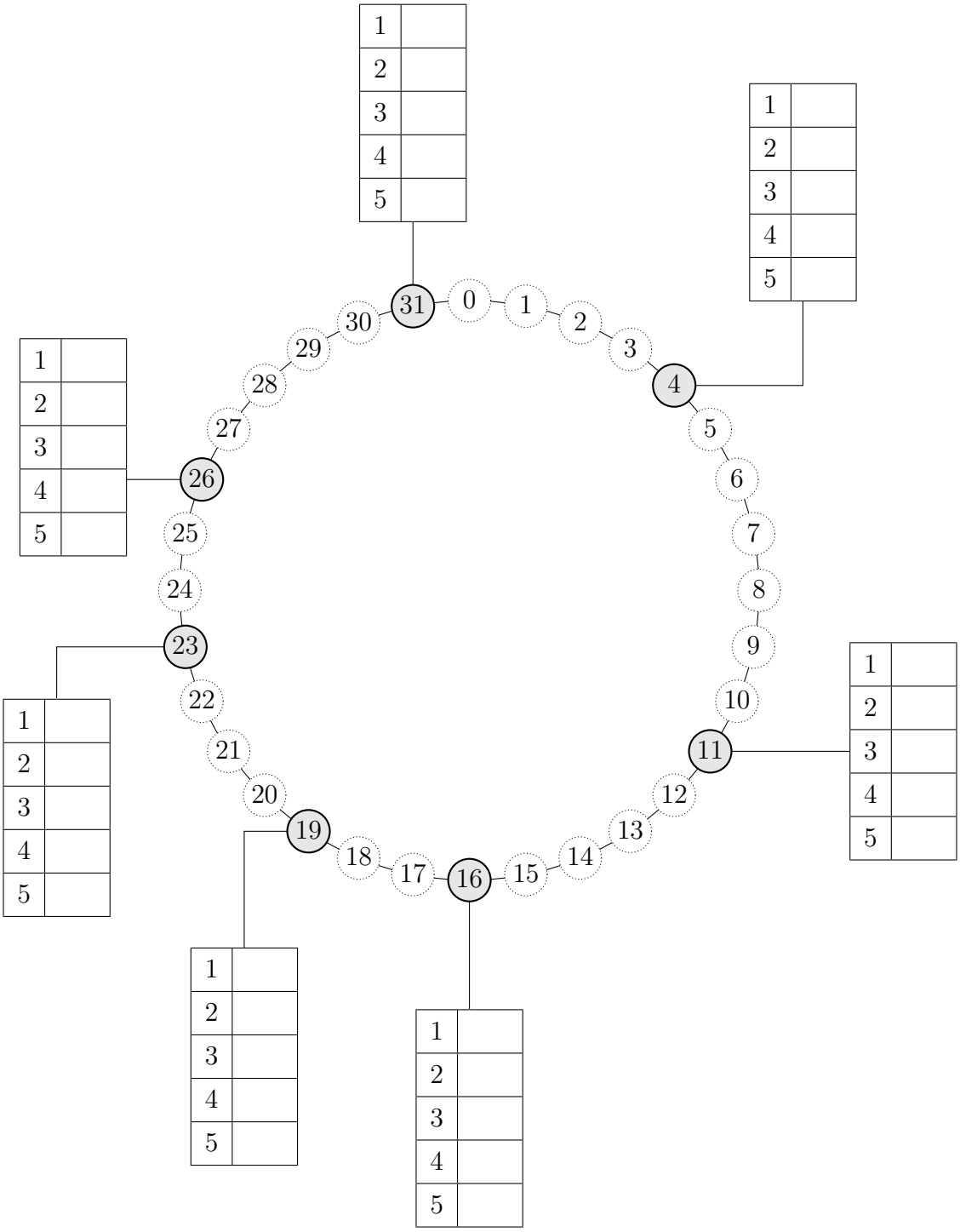

Figure 1: A 5-bit chord ring with blank finger tables.

3 Name Services [9 points]

Consider the hierarchical name service given in Figure 2.

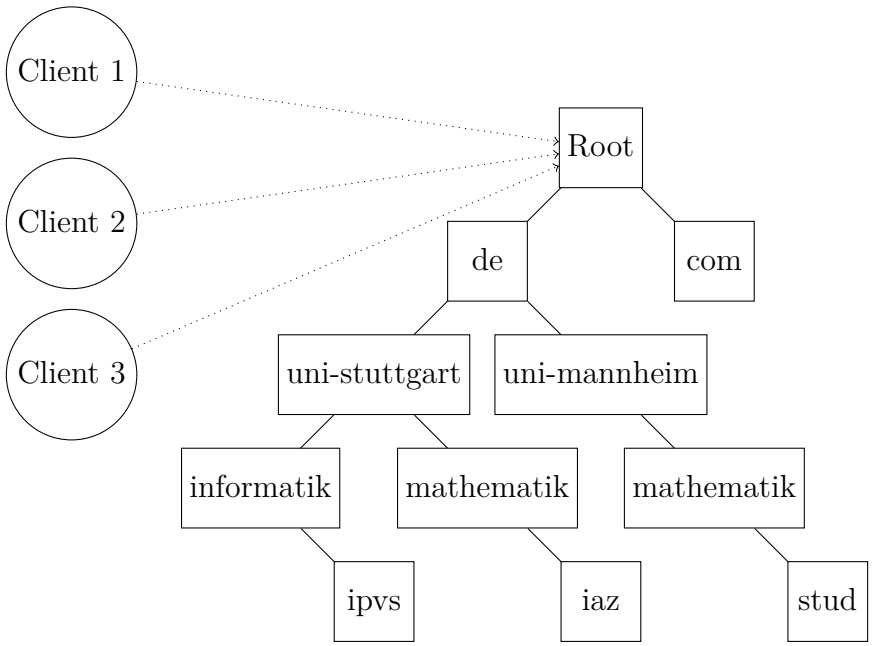

Figure 2: Name server hierarchy

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.iaz.mathematik.uni-stuttgart.de`“
3. Client 3 looks up „`www.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) [2 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) [2 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] What information does the PTR record provide? Perform a reverse lookup for the IP address 129.69.216.249. Provide the name of the server with the given IP address.

Hint: use the in-addr.arpa. domain

iii. [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!