UNIVERSITY of STIRLING

CSCU9V5: Concurrent & Distributed Systems

Distributed Laboratory 3 - Distributed Mutual Exclusion (DME) within a Token Ring

This lab contains a checkpoint at the end.

IMPORTANT: Only start coding when you have a design. This stage may take half of the lab class.

It is important that you use the lab session to ask staff any questions you might have on the material.

We want to implement a Distributed Mutual Exclusion system based using sockets. You will construct a program that will be replicated across a number of JVMs. The JVMs may reside on different computers, but initially they will all run on a single computer. The JVMs will form a ring, and "a token" will be passed from one node to the next. Only when in possession of the token can the program running on the JVM enter its critical region. You may want to simulate the critical region with a 3-second sleep. Also, each program will write to a common file an identifier and a date, so you can see access to the file being shared fairly.

Each instance of the program will be run concurrently on one JVMs, acting as a node in a ring, each one connected to the previous and the next one. The program should accept three parameters:

- the port you want the node to receive the token on,
- the host of the next node to receive the token, and
- the port on that next node.

The program should have two classes: they will be based on the *server* and *connection* classes from the first distributed lab. The token does not need to be an object in the program: the previous program in the ring will signal the passage of the token by simply making a connection to the serverSocket. After receiving such a connection, the program can spawn a connection class thread, which will also execute the critical region.

The critical region consists of a 3-second sleep (allowing you to see the progression of the token around the nodes) and of writing the identifier of the node and a timestamp to the file shared between the nodes. You may want some code along the lines of:

```
// So enter the critical region
```

```
//  ... record snap-shot on text file (our shared resource)

try {
      System.out.println("Writing to file: record.txt" );
      Date timestmp = new Date() ;
      String timestamp = timestmp.toString() ;

      // Next create fileWriter - true means writer *appends*
      FileWriter fw_id = new FileWriter("record.txt",true);

      // Create PrintWriter - true = flush buffer on each println
      // println means adds a newline (as distinct from print)
      PrintWriter pw_id = new PrintWriter(fw_id, true) ;
      pw_id.println ("Record from ring node on host " +this_host+
                  ", port number " +this_port+ ", is " +timestamp);
      . . .

      pw_id.close() ;
      fw_id.close() ;
      }
catch (java.io.IOException e)
      { System.out.println("Error writing to file: "+e); }

try {sleep (3000);}
catch (java.lang.InterruptedException e)
      { System.out.println("sleep failed: "+e);}
```

Finally the critical region should connect to the server socket of the next node signalling the transfer of the token to the next node. Remember you have both the hostname and port number of the next node from the command line. The connection alone signals the passing of the token, it is not necessary to pass an object representing the token; although you may wish to do so.

```
try {// connect to next node in the ring - signals passing the token.
      Socket s = new Socket(next_host, next_port);

      if (s.isConnected() ) // Did it connect OK?
            System.out.println("Socket to next node (" +next_host+ ":
                                          "+next_port+") connected OK");
      else
            System.out.println("** Socket to next ring node ("+
                  next_host+ ": " +next_port+ ") failed to connect");

      try {sleep (100);} // a short delay before closing socket.
      catch (java.lang.InterruptedException e)
      {System.out.println("sleep fail: "+e);}

      s.close() ; // token now passed.
      try {sleep (100);} // another short delay
      catch (java.lang.InterruptedException e)
      {System.out.println("sleep fail: "+e);}

      if (s.isClosed() )
```

```
        System.out.println("Socket to next ring node (" +next_host+ ": "
                                    +next_port+ ") is now closed") ;
        else
        System.out.println("** Socket to next ring node (" +next_host+ ":"
                                    +next_port+ ") is still open!!") ;
        } // end of socket try

        . . .
```

When using the Server from the first distributed lab as the basis of each node in the ring, you need to enter 3 parameters. The following code snippet may help:

```
public static void main(String argv[]) {

if ((argv.length < 3) || (argv.length > 3)) {
 System.out.println("Usage: [this port][next host][next port]");
 System.out.println("Only "+argv.length+" parameters entered");
 System.exit (1) ;
}

 int    this_port = Integer.parseInt (argv[0]) ;
 String next_host = argv[1] ;
 int    next_port = Integer.parseInt (argv[2]) ;

Server ring_host = new Server(this_port, next_host, next_port); }
// end main
```

This allows you to say where the next server/node is, and at what port number. If on the same machine, each server/node will need to be at a different port.

```
C:\Users\mkolb\workspace\DistributedSys3-RNode\bin>java Server 7000 127.0.0.1 7001
ring member hostname is LAPTOP-68EF1E4I
ring member port is 7000
Server socket (LAPTOP-68EF1E4I: 7000) on ring node is listening ....
```

```
C:\Users\mkolb\workspace\DistributedSys3-RNode\bin>java Server 7001 127.0.0.1 7002
ring member hostname is LAPTOP-68EF1E4I
ring member port is 7001
Server socket (LAPTOP-68EF1E4I: 7001) on ring node is listening ....
```

```
C:\Users\mkolb\workspace\DistributedSys3-RNode\bin>java Server 7002 127.0.0.1 7000
ring member hostname is LAPTOP-68EF1E4I
ring member port is 7002
Server socket (LAPTOP-68EF1E4I: 7002) on ring node is listening ....
```

Now you have a ring of 3, say, connected programs!

In practice you will also need to write a small *starter program* to launch the first token

into the ring to get a token circulating around the ring. This can be based on the *client* class used in the first distributed lab. The starter program will behave as one of the programs in the ring passing on the token. The starter program will be connected to one of the programs already in the ring and it should accept two parameters: the *host name* and the *port number* for the serverSocket service of the selected program within the ring where you want to inject the token into the ring.

The starter program is also a good place to clear the contents of the shared file, let us call it *record.txt*, which represents a shared resource within the ring. You may wish to use code along the lines of :

```
// clear out file used by ring nodes
System.out.println("Clearing record.txt file");

try {
      // create fileWriter - false = new file so clear contents
      FileWriter fw_id = new FileWriter "record.txt", false);
      // that's it - all now cleaned up
      fw_id.close() ;
} catch (java.io.IOException e)
{ System.err.println("Exception in clearing file: main: " +e);}
```

The nodes will then continue to pass the token between them as shown below.

```
C:\Users\mkolb\workspace\DistributedSys3-StartManager\bin>java Client 127.0.0.1 7000
Clearing record.txt file
Manager hostname is LAPTOP-68EF1E4I
ring element hostname is 127.0.0.1
ring element port is 7000

C:\Users\mkolb\workspace\DistributedSys3-StartManager\bin>
```

```
C:\Users\mkolb\workspace\DistributedSys3-RNode\bin>java Server 7000 127.0.0.1 7001
ring member hostname is LAPTOP-68EF1E4I
ring member port is 7000
Server socket (LAPTOP-68EF1E4I: 7000) on ring node is listening ....
Server socket (LAPTOP-68EF1E4I: 7000) has received the token
Ring node Thread-0 has the token.
Writing to file: record.txt
   ... started a thread to perform critical region
Socket from previous ring node is now closed
Ring node Thread-0 now releasing the token
Socket to next ring node (127.0.0.1: 7001) is connected OK
Socket to next ring node (127.0.0.1: 7001) is now closed
Ring node Thread-0 has released the token
-----------------------------------------------

Server socket (LAPTOP-68EF1E4I: 7000) has received the token
Ring node Thread-1 has the token.
Writing to file: record.txt
   ... started a thread to perform critical region
Socket from previous ring node is now closed
Ring node Thread-1 now releasing the token
Socket to next ring node (127.0.0.1: 7001) is connected OK
Socket to next ring node (127.0.0.1: 7001) is now closed
Ring node Thread-1 has released the token
-----------------------------------------------
```

```
C:\Users\mkolb\workspace\DistributedSys3-RNode\bin>java Server 7001 127.0.0.1 7002
ring member hostname is LAPTOP-68EF1E4I
ring member port is 7001
Server socket (LAPTOP-68EF1E4I: 7001) on ring node is listening ....
Server socket (LAPTOP-68EF1E4I: 7001) has received the token
Ring node Thread-0 has the token.
Writing to file: record.txt
   ... started a thread to perform critical region
Socket from previous ring node is now closed
Ring node Thread-0 now releasing the token
Socket to next ring node (127.0.0.1: 7002) is connected OK
Socket to next ring node (127.0.0.1: 7002) is now closed
Ring node Thread-0 has released the token
------------------------------------------------

Server socket (LAPTOP-68EF1E4I: 7001) has received the token
   ... started a thread to perform critical region
Socket from previous ring node is now closed
Ring node Thread-1 has the token.
Writing to file: record.txt
Ring node Thread-1 now releasing the token
Socket to next ring node (127.0.0.1: 7002) is connected OK
Socket to next ring node (127.0.0.1: 7002) is now closed
Ring node Thread-1 has released the token
------------------------------------------------
```

```
C:\Users\mkolb\workspace\DistributedSys3-RNode\bin>java Server 7002 127.0.0.1 7000
ring member hostname is LAPTOP-68EF1E4I
ring member port is 7002
Server socket (LAPTOP-68EF1E4I: 7002) on ring node is listening ....
Server socket (LAPTOP-68EF1E4I: 7002) has received the token
Ring node Thread-0 has the token.
Writing to file: record.txt
   ... started a thread to perform critical region
Socket from previous ring node is now closed
Ring node Thread-0 now releasing the token
Socket to next ring node (127.0.0.1: 7000) is connected OK
Socket to next ring node (127.0.0.1: 7000) is now closed
Ring node Thread-0 has released the token
------------------------------------------------

Server socket (LAPTOP-68EF1E4I: 7002) has received the token
Ring node Thread-1 has the token.
Writing to file: record.txt
   ... started a thread to perform critical region
Socket from previous ring node is now closed
Ring node Thread-1 now releasing the token
Socket to next ring node (127.0.0.1: 7000) is connected OK
Socket to next ring node (127.0.0.1: 7000) is now closed
Ring node Thread-1 has released the token
------------------------------------------------
```

Checkpoint.