UNIVERSITY of STIRLING

Concurrent and Distributed Systems

Distributed Systems Practical 3 – Mutual Exclusion using a Token Ring

This lab contains a checkpoint at the end. Please contact any lecturing staff when you reach this point to receive your credit.

It is important that you use the lab session to ask staff any questions you might have on the material. There are no tutorials! Please do not sit and be stuck!

Using sockets, construct a program that can be replicated across a number of JVMs. 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 a JVM enter its critical region. You may want to simulate the critical region with a 3-second sleep so you can see what is going on. It is also a good idea to write to a common file so you can see access to the file being shared fairly. In the first instance place the JVMs on the one host – you can distribute them across hosts later.

In this manner, the program will be run concurrently as separate instances on a number of JVMs, which can be run on different hosts. Each instance acts as a node in a ring. Each node will be connected to another so that together they form a ring. 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 are based on the server and connection classes from the first distributed lab. A connection to the serverSocket represents receipt of the token, so a connection class thread can then be run allowing the critical region to be executed.

The critical region should have a sleep (say 3 seconds) to allow you to see the progression of the token around the nodes. It should also write a line of text to a file containing identifiers of the node and include a timestamp. You may want some code along the lines of:

```
// Enter critical region and record snap-shot on text file
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 file_writer_id = new  FileWriter("record.txt", true);
  // Create PrintWriter - true = flush buffer on each println
  PrintWriter print_writer_id = new PrintWriter(file_writer_id, true);
```

```
    print_writer_id.println ("Record from ring node on host "
        +this_host+ ", port number " +this_port+ ", is " +timestamp);
    print_writer_id.close() ;
    file_writer_id.close() ; } // end try
  catch (java.io.IOException e){ System.err.println(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. You have both the hostname and port number of the next node from the command line parameters. The connection signals the passing of the token, it is not necessary to pass an object representing the token; although you may wish to do so.

In practice, you will need to write a small second 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. It should accept two parameters: the host name and the port number for the serverSocket of the node within the ring where you want to inject the token into the ring. This is also a good place to clear the contents of the shared file (e.g. record.txt) you will use as a shared resource within the ring. You may wish to use code along the lines of:

```
System.out.println("Clearing record.txt file");
try { // delete file contents
  // create fileWriter - false = new file so clear contents
  FileWriter file_writer_id = new  FileWriter "record.txt", false) ;
  file_writer_id.close() ;
} // end try
catch (java.io.IOException e)
{System.err.println("Exception in clearing file: main: " +e);}
```

When using the Server in 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);
   }  // endif

   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.

Checkpoint