UNIVERSITY of STIRLING

Concurrent and Distributed Systems

Distributed Systems Practical 1 – Sockets

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!

Taking the server-client code presented in the lecture as a starting point, write a server and client that allows the client to access the time from the server. Initially run the server and client on the same host. Try with two clients running at once.

Now alter the client and the server (Connection class) to respond with the time 3 times before completing. Again, try with more than one client. You may find java.lang.Thread.getName() useful with System.out.println(..) in the server-connection code to follow what is happening. A one-second delay within the loop on the server side helps too.

Next, have the server and client on separate hosts. Again, have more than one client operating at once. You may find java.net.InetAddress.getLocalHost() useful with System.out.println(..) in both the server and client to follow what is happening. This might be along the lines of :

```
// as an experiment let's get the IP address of the server
   try
    { InetAddress server_inet_address =  InetAddress.getLocalHost() ;
      String server_host_name = server_inet_address.getHostName();
      System.out.println ("Server hostname is " +server_host_name ) ;
      System.out.println ("Server port is xxxx") ;
    }  // end of try
   catch (java.net.UnknownHostException e)
    { System.out.println(e);
         System.exit(1);
    }  // end of catch
```

Also, it will be easier if you modify the client along the lines of :

```
public static void main(String argv[]) {
  if ((argv.length < 1) || (argv.length > 2)) {
    System.out.println("Usage: [host] <port>") ;
    System.exit(1);
  }

  String server_host = argv[0] ;
  int server_port = 5156 ;
  if (argv.length == 2)
      server_port = Integer.parseInt (argv[1]) ;

  Client client = new Client(server_host, server_port);
} // end of main
```

This will allow you enter the server name and port when you run your client.
Finally try your client with someone else's server.

Now, try moving three double values and adding them in the client to obtain their total. To carry out such arithmetic operations it is better not to send the information as strings. So instead of writing out text with the PrintWriter class, write the data as encode data with the DataOutputStream class:

```
  pout = new PrintWriter(outputLine.getOutputStream(), true);
```

becomes

```
  poutdata = new DataOutputStream(outputLine.getOutputStream());
```

The DataOutputStream supports methods such as writeDouble():

```
  try {poutdata.writeDouble(1.12345);}
  catch (java.io.IOException e)
    { System.out.println(e);
      System.exit (8) ;  }
```

Note you *could* output doubles as text with: pout.println(1.12345) ; But the client can only receive this as a string. You might convert the string, but it's better to use the readDouble() method of the DataInputStream object. The client can receive the data simply by creating a DataInputStream object: `ip = new DataInputStream(in);`

And using its readDouble method to access and decode the data. Note you now have the received data as a double which you can add etc. to other variables:

```
  double local_double = 0.0 ;
  local_double = ip.readDouble() ;
```

Strictly, it's all transmitted as bytes – it's simply a matter of how the data is encoded and decoded. Here both server *and* client know the data structure and are coded appropriately.

Checkpoint