Distributed Systems (335) Assessed Coursework

Lecturer: M. Sloman Hand-in Date: Mon. 17 November 2003

Question

1) Using the code given in the directories `rmi-common`, `rmi-server` and `rmi-client` as a basis:
   a) Create an RMI server with a single method, `getResponse(..)`, which takes a String and an Integer and returns a String of the form:

      `“You've said " + stringArgument + " " + integerArgument + " times?!"`

   b) Create an RMI client which calls the server a user-specified number of times. The integer is the count of the number of times the client has called the server. The client has to measure the time it takes to make n calls and determine if any messages are lost.

2) Using the code given in directory `udp` as a basis:
   a) Create a server that listens for datagrams, extracts from these a string and an integer value and sends back a message as in 1a) to the client. The server should maintain a count of lost messages (assume no reordering in a LAN).

   b) Create a client which sends datagrams containing an integer message number and a string to this server until it has received a user-specified number of replies. Your client must measure the time from sending the first message until n replies are received. It should also maintain a count of lost messages, assuming no reordering in a LAN.

3) Run your experiments in a single computer, and in 2 computers connected by a LAN e.g. 2 machines in the Lab.
   a) Calculate the average time for the RMI call and datagram request-reply in each case.
   b) Indicate how many messages have been lost.
   c) Explain the variations, if any, in your 4 sets of results, suggesting reasons for these variations.

Your solution should deal with exceptions appropriately and include reasonably formatted listings of your programs and a console log or screen dump to show the program actually running.

Notes:

You can use the `socket.getReceivebufferSize` method to determine the size of the data structure required to hold an incoming packet.

Some source code outlines and supporting files are provided. You are not obliged to use these, however they should simplify the process of achieving working solutions. A few notes on the layout and support files follow:

After extracting the files from the archive provided, you should run `install.csh` (or `install.bat` on Windows) to obtain the appropriate build and execution scripts which are described below.

The directory `rmi-common` contains code required for both 1a and 1b. The directory `rmi-server` contains code required for only 1a; `rmi-client` contains code required for only 1b. This highlights that there is no need to have the source code of the server available to the client.

The file "policy" is a simple configuration file required for 1. More constrained policies are possible, but this should provide the lowest barrier to testing.

The directory `udp` contains code required for 2.

The `Makefile` allows Linux users to use make to compile the various parts of the exercise. It can also be used to help configure your preferred development environment with the correct commands, flags and parameters. Windows users can use the `build.bat` script to compile the exercise in the same way.

The four shell scripts (`RMIClient` (.bat or .csh) etc) allow users to execute the various parts of the exercise. They can also be used to help configure your preferred execution environment with the correct commands, flags and parameters. The "time" command here should help with providing timing. The java Date class may also be used. Note that time returns 0 for parameters that a given OS version does not measure.

```
/***************************************************************
RMIServerI.java

Remote interface for the RMI Server class
***************************************************************/

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface RMIServerI
    extends Remote
    {
    public String getResponse(String message, int count) throws RemoteException;
}


/***************************************************************
RMIServer.java

Implements the getResponse method that builds a message from the parameters passed and
returns it as a string.

TO DO: Modify this class so that it is a valid RMI server class and add the code that
will bind it to a RMI Registry

Your code should ensure that it is *not* necessary to start an RMI registry manually
before starting this server.
***************************************************************/

import java.rmi.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.lang.reflect.Array;
import java.net.InetAddress;
import java.net.MalformedURLException;
import java.net.UnknownHostException;

public class RMIServer
     /** TO DO make valid RMI server class **/
    implements RMIServerI
    {

    public static void main(String args[]) {
    String      serverURL   = null;
    RMIServer   server      = null;

        // Define the URL we'll offer ourselves as
        try {
            serverURL = new String("rmi://" +
                InetAddress.getLocalHost().getHostName() + "/RMIServer");
        } catch (UnknownHostException e) {
            System.err.println("RMIServer start-up failed (defining URL),
                unknown host  exception: " + e);
            System.exit(-1);
        }

        // Security set-up
        /** TO DO: Set up security manager **/
        }

        /** TO DO: Create Server Object and bind to registry **/

        System.out.println("RMIServer ready at " + serverURL);
    }

    public RMIServer()
        throws RemoteException  // this is a result of extending UnicastRemoteObject
    {

    }

    /** TO DO: Implement the getResponse(..) method **/


}
```

```
/*************************************************************
RMIClient.java

Binds to server and calls getResponse method for as many
iterations as specified in command line parameters.  Times
how long the entire process takes.

TO-DO:
Bind to your server and make the calls to getResponse in
the manner specified in Question 1b.

*************************************************************/

import java.rmi.*;
import java.lang.reflect.Array;
import java.net.MalformedURLException;
import java.util.Date;

public class RMIClient {

    public static void main(String args[]) {
        String      serverURL;
        int         countTo;
        String      message;

        // Get the parameters
        if (Array.getLength(args) < 3) {
            System.err.println("Arguments required: server IP/name, message count,
                    message");
            System.exit(-1);
        }
        serverURL = new String("rmi://" + args[0] + "/RMIServer");
        countTo = Integer.parseInt(args[1]);
        message = args[2];

        /** TO-DO: Set up a security manager **/

        new RMIClient(serverURL, countTo, message);
    }

    public RMIClient(String serverURL, int countTo, String message) {
        RMIServerI  server;
        int         answers, tries;
        Date        fromTime, toTime;
        String      response;

        /** TO-DO: Bind to the server **/

        // set up house keeping
        answers = 0;
        tries = 0;
        response = null;
        fromTime = new Date();

        /** TO-DO: Send/Receive messages according to problem specification **/

        toTime = new Date();

        // Report the time
        System.out.println(answers + " answers from " + tries + " RMI call attempts");
        System.out.println("Last message was: " + response);
        System.out.println("Took " + (toTime.getTime() - fromTime.getTime()) +
             "ms in the loop ");
    }

}
```

```
/*************************************************************
MessageHandlerI.java

Interface class that must be impelemented to handle incoming
message

*************************************************************/

import java.net.DatagramPacket;

public interface MessageHandlerI {
    public void receiveMsg(DatagramPacket pac);
}


/*************************************************************
MessageInfo.java

This class holds the UDP message information and parses it as
necessary.

*************************************************************/

import java.net.DatagramPacket;
import java.net.InetAddress;
import java.util.StringTokenizer;

public class MessageInfo {

    public  InetAddress senderAddress;
    public  String  msgStr;
    public  int     msgInt;

    public MessageInfo(DatagramPacket pac) {
        String          dataString;
        StringTokenizer toker;
        String          token;

        // Parse the message. This makes some assumptions, including:
        // 1) that the message format is iiii;sssss
        // 2) that the message is contained in one datagram
        dataString = new String(pac.getData());
        toker = new StringTokenizer(dataString, ";");
        if (toker.hasMoreTokens()) {

                // First Token is the Integer
                token = toker.nextToken();
                msgInt = (Integer.valueOf(token)).intValue();

                // Next token is the String
                token = toker.nextToken();
                msgStr = token;

                token = toker.nextToken();
        }

        // record source address
        senderAddress = pac.getAddress();
    }
}
```

```
/**************************************************************
PortListener.java

Generic listener that will performs a call back to the handler
object that created it with any messages received
```

TO-DO:
Listen for incoming data, extract it into Datagram and pass it back to the handler

```
*************************************************************/

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.io.IOException;

public class PortListener
    implements  Runnable {

    protected       MessageHandlerI         handler;
    protected       DatagramSocket          soc;
    protected       boolean                 close;

    public PortListener(MessageHandlerI handler, DatagramSocket soc) {
        this.handler = handler;
        this.soc = soc;
        close = false;
    }

    public void run() {
        // a separate thread to listen for incoming datagrams
        // this allows the client to handle sending and receiving asynchronously
                        & without time-out on loss

        while (!close) {

            /** TO-DO: Receive data and return it using callback method of handler **/

        }
    }

    public void close() {
        close = true;
        soc.close();
    }
}

/**************************************************************
UDPBase.java

Provides basic UDP packet send/recieve functionality, must
be extended by any class that wishes to use it's features.
```

TO-DO:
Open sockets on the appropriate ports for sending/receiving
Construct the datagram and send it

```
*************************************************************/

import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.SocketException;
import java.net.InetAddress;
import java.io.IOException;

public abstract class UDPBase
    implements  MessageHandlerI {

    protected       int                 sendPort, recvPort;
    protected       DatagramSocket      sendSoc, recvSoc;
    protected       PortListener        rec;

    public UDPBase(int sendPort, int recvPort) {
        String          message, response;
        Thread          runner;

        this.sendPort = sendPort;
        this.recvPort = recvPort;

        /*TO-DO: Open sockets on the ports to send and recv from **/

        // rec listens to the port for packets and delivers them to receiveMsg
        rec = new PortListener(this, recvSoc);
```

```
        runner = new Thread((Runnable)rec);
        runner.start();
    }

    protected void send(String msgStr, int msgInt, InetAddress destAddr, int destPort) {
        String      message;

        message = new String(msgInt + ";" + msgStr + ";");
        sendDatagram(message, destAddr, destPort);
    }

    protected synchronized void sendDatagram(String payload, InetAddress destAddr,
            int destPort) {

        /** TO-DO: Construct datagram and send it **/

    }

    public abstract void receiveMsg(DatagramPacket pac);

    public synchronized void close() {
        rec.close();
    }
}

/**************************************************************
UDPServer.java

Waits for a message and responds with a string that combines
parameters from the message body.

*************************************************************/

import java.net.DatagramPacket;
import java.lang.reflect.Array;

public class UDPServer
    extends UDPBase
    implements  MessageHandlerI {

    public static void main(String args[]) {
        int recvPort, sendPort;

        // Get the parameters
        if (Array.getLength(args) < 1) {
            System.err.println("Arguments required: send port, recv port");
            System.exit(-1);
        }
        sendPort = Integer.parseInt(args[0]);
        recvPort = Integer.parseInt(args[1]);

        new UDPServer(sendPort, recvPort);
    }

    public UDPServer(int sendPort, int recvPort) {
        super(sendPort, recvPort);
        System.out.println("UDPServer ready");
    }

    public synchronized void receiveMsg(DatagramPacket pac) {
        MessageInfo     msg;
        String          reply;

        // Parse the message
        msg = new MessageInfo(pac);

        // Build our reply & send it back
        // Assume their recvPort == our recvPort
        send(getResponse(msg.msgStr, msg.msgInt), msg.msgInt, msg.senderAddress,
                recvPort);
    }

    public String getResponse(String message, int count) {
        return new String ("You've said " + message + " " + count + " times?!");
    }
}
```

```
/************************************************************
UDPClient.java

Uses UDPBase features to send messages to the server.
Performs timing operations to determine to elapsed time to
send/receive the messages.

************************************************************/

import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.lang.reflect.Array;
import java.util.Date;

public class UDPClient
    extends     UDPBase
    implements  MessageHandlerI {

    protected       String      lastMsgRecvd;
    protected       int         countTo, answersRecvd, msgsLost;

    public static void main(String args[]) {
        InetAddress serverAddr = null;
        int         recvPort, sendPort;
        int         countTo;
        String      message;

        // Get the parameters
        if (Array.getLength(args) < 4) {
            System.err.println("Arguments required: server name/IP, send port, recv port,
                        message count, message");
            System.exit(-1);
        }

        try {
            serverAddr = InetAddress.getByName(args[0]);
        } catch (UnknownHostException e) {
            System.out.println("Bad server address in UDPClient, " + args[0] +
                    " caused an unknown host exception " + e);
            System.exit(-1);
        }
        sendPort = Integer.parseInt(args[1]);
        recvPort = Integer.parseInt(args[2]);
        countTo = Integer.parseInt(args[3]);
        message = args[4];

        // during function test - to stop this blasting the network
        //if (countTo > 10)
        //  System.exit(-2);

        new UDPClient(serverAddr, sendPort, recvPort, countTo, message);
    }

    public UDPClient(InetAddress serverAddr, int sendPort, int recvPort, int countTo,
                        String message) {
        super(sendPort, recvPort);
        lastMsgRecvd = null;
        this.countTo = countTo;
        this.msgsLost = 0;
        testLoop(serverAddr, message);
        close();
    }

    protected void testLoop(InetAddress serverAddr, String message) {
        int             tries;
        Date            fromTime, toTime;
        DatagramPacket  pkt;

        // set up house keeping
        answersRecvd = 0;
        tries = 0;
        fromTime = new Date();

        /** TO-DO: Send messages according to problem specification **/

        toTime = new Date();
```

```
        // Report the time
        System.out.println(answersRecvd + " answers from " + tries + " UDP messages");
        System.out.println("Lost " + msgsLost + " UDP messages");
        System.out.println("Last message was: " + lastMsgRecvd);
        System.out.println("Took " + (toTime.getTime() - fromTime.getTime()) +
            "ms in the loop");
    }

    public synchronized void receiveMsg(DatagramPacket pac) {
        MessageInfo         msg;
        String              reply;

        /** TO-DO: Handle the received message **/

    }
}
```