

Institution of Technology

School of Computing

Department of Information Technology and computer science

Course Name: Introduction to Distributed System

Instructor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ [Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_](mailto:yimer@wku.edu.et)

\

Lab 3: Message Oriented Communication

Objectives: This Lab is a demonstration on how to implement message oriented communication by implementing message passing interface (MPI). We are going to implement some of the primitives defined in message passing interface by using java sockets. We will have two computers communicating, a server and a client and they both will implement the MPI primitives. They can then be able to communicate by using the defined primitives.

• The following are the primary objectives of this lab session:

– Understanding the basics concepts of message oriented communication.

– Understanding the primitives that are used in message passing interface (MPI).

– Write our own implementation of MPI primitives.ke remote method invocations from the client

3.1 Creating Distributed Applications by Using Java MPI

Aim: Write an implementation of message passing interface and enable the communication of a client and server computers using MPI primitives.

Requirements: We are going to use Java networking and sockets to implement MPI. So, we need a computer that has

JDK and some decent java IDE.

Practical: Write a server and client that implements the following MPI primitives.

1. MPI-send():

- The primitive MPI-send may either block the caller until the specified message has been copied to the MPI

runtime system at the sender’s side, or until the receiver has initiated a receive operation.

2. MPI-recv()

- The operation MPI-recv is called to receive a message; it blocks the caller until a message arrives. There is also an asynchronous variant, called MPI-irecv, by which a receiver indicates that is prepared to accept a message. The receiver can check whether or not a message has indeed arrived, or block until one does.

3. MPI-sendrecv()

- Finally, the strongest form of synchronous communication is also supported: when a sender calls MPI- sendrecv, it sends a request to the receiver and blocks until the latter returns a reply. Basically, this primitive corresponds to a normal RPC.

3-1

Procedure 1: Both client and server should provide a user interface (Menu) to enable the user to select which primitive to use.

Procedure 2: They will accept the user selection and call the appropriate MPI primitive implementation.

Implementing the server:

Step 1: Importing the necessary packages:

import java.io.\*;

import java.net.\*;

Step 2: Writing the server class:

public class Server {

PrintStream out = null; BufferedReader in = null; BufferedReader br = null; String str = "";

}

Step 3: Writing the server class constructor, everything is done in the constructor including the menu and the method calls:

public Server() throws IOException { ServerSocket ss = new ServerSocket(7000); System.out.println("Server Started"); Socket s = ss.accept();

out = new PrintStream(s.getOutputStream());

in = new BufferedReader(new InputStreamReader(s.getInputStream()));

br = new BufferedReader(new InputStreamReader(System.in)); String str;

while (s != null) {

System.out.println("Select the operation to be performed"); System.out.println("1. To send message without any response MPI\_send"); System.out.println("2. To send message with response MPI\_sendrecv "); System.out.println("3. To receive message MPI\_recv"); System.out.println("4. To end the program");

str = br.readLine();

out.println(str);

if (str.equals("1")) { MPI\_send();

} else if (str.equals("2")) { MPI\_sendrec();

} else if (str.equals("3")) { MPI\_recv();

} else {

break;

}

}

}

Program: Complete Echo server program implementation

EchoServer.java

package MPI;

import java.io.\*;

import java.net.\*;

public class EchoServer implements Runnable {

Socket socket = null;

static ServerSocket ss;

EchoServer(Socket newSocket) {

this.socket = newSocket;

}

public static void main(String args[]) throws IOException {

ss = new ServerSocket(7000); System.out.println("Server Started"); while (true) {

Socket s = ss.accept();

EchoServer es = new EchoServer(s); Thread t = new Thread(es); t.start();

}

}

public void run() {

try {

while (ss != null) {

BufferedReader in = new BufferedReader(new

InputStreamReader(socket.getInputStream()));

PrintStream out = new PrintStream(socket.getOutputStream()); BufferedReader br = new BufferedReader(new InputStreamReader(System.in)); String str = in.readLine();

if (str.equals("1")) { System.out.println(in.readLine());

} else if (str.equals("2")) { System.out.println(in.readLine()); System.out.println("Enter Data "); str = br.readLine(); out.println(str);

} else if (str.equals("3")) { System.out.println("Enter Data "); str = br.readLine(); out.println(str);

}

}

System.out.println("Exiting...");

} catch (Exception e) {

}

}

}

Program: Complete Echo client program implementation

EchoClient.java

public class EchoClient { PrintStream out = null; BufferedReader in = null; BufferedReader br = null; String str = "";

public EchoClient() throws IOException { Socket s = new Socket("localhost", 7000); out = new PrintStream(s.getOutputStream());

in = new BufferedReader(new InputStreamReader(s.getInputStream()));

br = new BufferedReader(new InputStreamReader(System.in));

while (s != null) {

System.out.println("Select the operation to be performed"); System.out.println("1. To send message without any response"); System.out.println("2. To send message with response"); System.out.println("3. To recieve message"); System.out.println("4. To end the program");

str = br.readLine();

out.println(str);

if (str.equals("1")) { MPI\_send();

} else if (str.equals("2")) { MPI\_sendrec();

} else if (str.equals("3")) { MPI\_recv();

} else {

break;

}

}

}

public static void main(String args[]) throws IOException { EchoClient ec = new EchoClient(); System.out.println("Exiting...");

}

void MPI\_send() throws IOException { System.out.println("Enter Data "); str = br.readLine(); out.println(str);

}

void MPI\_sendrec() throws IOException { System.out.println("Enter Data "); str = br.readLine(); out.println(str); System.out.println(in.readLine());

}

void MPI\_recv() throws IOException { System.out.println(in.readLine());

}

}