// Exp 2 Implement client-server model using socket programming

client.java

import java.net.\*;

import java.io.\*;

public class client {

public static void main(String [] args) {

String serverName = args[0];

int port = Integer.parseInt(args[1]);

try {

System.out.println("Connecting to " + serverName + " on port " + port);

Socket client = new Socket(serverName, port);

System.out.println("Just connected to " + client.getRemoteSocketAddress());

OutputStream outToServer = client.getOutputStream();

DataOutputStream out = new DataOutputStream(outToServer);

out.writeUTF("Hello from " + client.getLocalSocketAddress());

InputStream inFromServer = client.getInputStream();

DataInputStream in = new DataInputStream(inFromServer);

System.out.println("Server says " + in.readUTF());

client.close();

} catch(IOException e) {

e.printStackTrace();

}

}

}

server.java

import java.net.\*;

import java.io.\*;

public class server extends Thread {

private ServerSocket serverSocket;

public server(int port) throws IOException {

serverSocket = new ServerSocket(port);

serverSocket.setSoTimeout(60000);

}

public void run() {

while (true) {

try {

System.out.println("Waiting for client on port " +

serverSocket.getLocalPort() + "...");

Socket server = serverSocket.accept();

System.out.println("Just connected to " + server.getRemoteSocketAddress());

DataInputStream in = new DataInputStream(server.getInputStream());

System.out.println(in.readUTF());

DataOutputStream out = new DataOutputStream(server.getOutputStream());

out.writeUTF("Thank you for connecting to " + server.getLocalSocketAddress()

+ "\nGoodbye!");

server.close();

} catch(SocketTimeoutException s) {

System.out.println("Socket timed out!");

break;

} catch(IOException e) {

e.printStackTrace();

break;

}

}

}

public static void main(String[] args) {

int port = Integer.parseInt(args[0]);

try {

Thread t = new server(port);

t.start();

} catch(IOException e) {

e.printStackTrace();

}

}

//Exp 3 Implement chat application using client-server model

client1.java:

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class client1 {

public static void main(String [] args) {

Scanner sc = new Scanner(System.in);

String serverName = args[0];

int port = Integer.parseInt(args[1]);

int ch = 1;

while (ch == 1) {

try {

Socket client = new Socket(serverName, port);

OutputStream outToServer = client.getOutputStream();

DataOutputStream out = new DataOutputStream(outToServer);

System.out.println("Enter a message to send:");

String str = sc.nextLine();

out.writeUTF(str);

client.close();

System.out.print("Send another message? (0/1): ");

ch = sc.nextInt();

sc.nextLine();

} catch(IOException e) {

e.printStackTrace();

}

}

}

}

server1.java:

import java.net.\*;

import java.io.\*;

public class server1 extends Thread {

private ServerSocket serverSocket;

public server1(int port) throws IOException {

serverSocket = new ServerSocket(port);

serverSocket.setSoTimeout(60000);

System.out.println("Listening on "

+ Inet4Address.getLocalHost().getHostAddress() + ":6666 ...");

}

public void run() {

while (true) {

try {

Socket server = serverSocket.accept();

DataInputStream in = new DataInputStream(server.getInputStream());

System.out.println(in.readUTF());

DataOutputStream out = new DataOutputStream(server.getOutputStream());

server.close();

} catch(SocketTimeoutException s) {

System.out.println("Socket timed out!");

break;

} catch(IOException e) {

e.printStackTrace();

break;

}

}

}

public static void main(String[] args) {

int port = Integer.parseInt(args[0]);

try {

Thread t = new server1(port);

t.start();

} catch(IOException e) {

e.printStackTrace();

}

}

}

// Exp 4 Client-server based program using RPC

Steps:

$ sudo apt-get update

$ sudo apt-get install rpcbind

$ rpcinfo

Shows all services and program versions

Create new folder “newrpc” and create new file “add.x” and save it in the new folder. Add the following content in the file:

struct intpair {

int a;

int b;

};

program ADD\_PROG {

version ADD\_VERS {

int ADD(intpair) = 1;

} = 1;

} = 0x23451111;

Navigate to new folder

$ cd newrpc

Open first terminal to generate rpcgen

$ rpcgen -a -C add.x

Goto “newrpc” folder where new files are added

$ make -f Makefile.add

Open second terminal and run server

$ sudo ./add\_server

Open new terminal and goto “newrpc” path and run client

$ ./add\_client localhost

Goto folder newrpc, open add-server.c file and make changes like

printf("get the client request\n");

Close gedit with save file

$ make -f Makefile.add

Again open new terminal and run client

$ cd newrpc

$ ./add\_client localhost

You’ll see the server terminal display the message “get the client request”

File “add.x”:

struct intpair {

int a;

int b;

};

program ADD\_PROG {

version ADD\_VERS {

int ADD(intpair) = 1;

} = 1;

} = 0x23451111;

File “add\_server.c”:

/\*

\* This is sample code generated by rpcgen.

\* These are only templates and you can use them

\* as a guideline for developing your own functions.

\*/

#include "add.h"

int \*

add\_1\_svc(intpair \*argp, struct svc\_req \*rqstp)

{

static int result;

/\*

\* insert server code here

\*/

printf("Got client request\n");

return &result;

}

Installing “rpcbind” and running “rpcinfo”:

//DS exp 5Implement client-server based program using RMI.

**Calculator.java:**

public interface Calculator extends java.rmi.Remote {

public long add(long a, long b) throws java.rmi.RemoteException;

public long sub(long a, long b) throws java.rmi.RemoteException;

public long mul(long a, long b) throws java.rmi.RemoteException;

public long div(long a, long b) throws java.rmi.RemoteException;

}

**CalculatorImpl.java:**

public class CalculatorImpl extends java.rmi.server.UnicastRemoteObject

implements Calculator {

public CalculatorImpl() throws java.rmi.RemoteException {

super();

}

public long add(long a, long b) throws java.rmi.RemoteException {

System.out.println("Doing addition");

return a + b;

}

public long sub(long a, long b) throws java.rmi.RemoteException {

System.out.println("Doing subtraction");

return a - b;

}

public long mul(long a, long b) throws java.rmi.RemoteException {

System.out.println("Doing multiplication");

return a \* b;

}

public long div(long a, long b) throws java.rmi.RemoteException {

System.out.println("Doing division");

return a / b;

}

}

**CalculatorServer.java:**

import java.rmi.Naming;

public class CalculatorServer {

public CalculatorServer() {

try {

Calculator c = new CalculatorImpl(); Naming.rebind("rmi://localhost:1099/CalculatorService", c);

} catch (Exception e) {

System.out.println("Trouble: " + e);

}

}

public static void main(String[] args) {

new CalculatorServer();

}

}

**CalculatorClient.java:**

import java.util.\*;

import java.rmi.Naming;

import java.rmi.RemoteException;

import java.net.MalformedURLException;

import java.rmi.NotBoundException;

public class CalculatorClient {

public static void main(String[] args) {

try {

Calculator c = (Calculator)Naming.lookup("rmi://localhost/CalculatorService");

Scanner in = new Scanner(System.in);

long a = in.nextLong();

long b = in.nextLong();

System.out.println("Subtraction: " + c.sub(a, b));

System.out.println("Addition: " + c.add(a, b));

System.out.println("Multiplication: " + c.mul(a, b));

System.out.println("Division: " + c.div(a, b));

} catch (MalformedURLException murle) {

System.out.println("\nMalformedURLException\n" + murle);

} catch (RemoteException re) {

System.out.println("\nRemoteException\n" + re);

} catch (NotBoundException nbe) {

System.out.println("\nNotBoundException\n" + nbe);

} catch (java.lang.ArithmeticException ae) {

System.out.println("\njava.lang.ArithmeticException\n" + ae);

}

}

}

//ds exp 6Implementation of clock synchronization in Java.

**SCServer.java**

import java.io.\*;

import java.net.\*;

import java.sql.\*;

public class SCServer {

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

InetAddress lclhost;

lclhost = InetAddress.getLocalHost();

long maxtime, skewtime, datatime;

String maxtimestr, skewtimestr;

BufferedReader br;

ClntServer ser = new ClntServer(lclhost);

System.out.println("Enter the maximum time:");

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

maxtimestr = br.readLine();

System.out.println("Enter the maximum skew time:");

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

skewtimestr = br.readLine();

maxtime = Long.parseLong(maxtimestr);

skewtime = Long.parseLong(skewtimestr);

while(true) {

datatime = System.currentTimeMillis();

long G = datatime - maxtime - skewtime;

System.out.println("D = " + datatime);

System.out.println("G = " + G);

ser.setTimeStamp(new Timestamp(G));

ser.recPort(8001);

ser.recData();

} } }

class ClntServer {

InetAddress lclhost;

int recport;

Timestamp obtmp;

ClntServer(InetAddress lclhost) {

this.lclhost = lclhost;

}

void recPort(int recport) {

this.recport = recport;

}

void setTimeStamp(Timestamp obtmp) {

this.obtmp = obtmp;

}

void recData() throws Exception {

String msgstr = "";

DatagramSocket ds;

DatagramPacket dp;

BufferedReader br;

byte buf[] = new byte[256];

ds = new DatagramSocket(recport);

dp = new DatagramPacket(buf, buf.length);

ds.receive(dp);

ds.close();

msgstr = new String(dp.getData(), 0, dp.getLength());

System.out.println(msgstr);

Timestamp obtmp = new Timestamp(Long.parseLong(msgstr));

if (this.obtmp.before(obtmp) == true) {

System.out.println("The Message is accepted.");

} else {

System.out.println("The Message is rejected.");

} } }

**SCClient.java**

import java.io.\*;

import java.net.\*;

public class SCClient {

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

InetAddress lclhost;

lclhost = InetAddress.getLocalHost();

while (true) {

Client cntl = new Client(lclhost);

cntl.sendPort(9001);

cntl.sendData();

}

}

}

class Client {

InetAddress lclhost;

int senport;

Client(InetAddress lclhost) {

this.lclhost=lclhost;

}

void sendPort(int senport) {

this.senport=senport;

}

void sendData()throws Exception {

DatagramPacket dp;

DatagramSocket ds;

BufferedReader br;

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

System.out.println("Enter the data:");

String str = br.readLine();

ds = new DatagramSocket(senport);

dp = new DatagramPacket(str.getBytes(), str.length(), lclhost, senport - 1000);

ds.send(dp);

ds.close();

}

}

//ds exp 7Implementation of Election algorithm.

**RingElection.java:**

import java.util.Scanner;

class Process {

int processId;

boolean active;

public Process(int processId) {

this.processId = processId;

active = true;

}

}

public class RingElection {

private Scanner consoleInput;

private Process[] process;

private int NosOfProcess;

public RingElection() {

System.out.println("\n\t\t==:Ring Coordinator Election Algorithm:==\n");

consoleInput = new Scanner(System.in);

}

public void getInput() {

System.out.println("Enter number of process in ring:");

NosOfProcess = consoleInput.nextInt();

process = new Process[NosOfProcess];

for (int i = 0; i < NosOfProcess; i++) {

System.out.print("Enter Process ID of p" + i + ": ");

int pid = consoleInput.nextInt();

initializeProcess(i, pid);

}

sortProcess();

putOutput();

}

private void initializeProcess(int i, int pid) {

process[i] = new Process(pid);

}

public void conductElection() {

try {

Thread.sleep(2000);

} catch (Exception e) {

System.out.println(e);

}

System.out.println("process " + process[getMax()].processId +" Fail");

process[NosOfProcess-1].active = false;

while (true) {

System.out.print("Conduct Election?\nyes or exit: ");

String choice = consoleInput.next();

if("yes".equals(choice) || "Yes".equals(choice) || "y".equals(choice) || "Y".equals(choice)) {

System.out.println("Election initiated by: ");

int initiatorProcess = consoleInput.nextInt();

for (int i = 0; i < NosOfProcess; i++) {

if(process[i].processId == initiatorProcess) {

initiatorProcess = i;

break;

}

}

int prev = initiatorProcess;

int next = prev + 1;

while (true) {

if (process[next].active) {

System.out.println("Process " + process[prev].processId + " pass message to process " + process[next].processId);

prev = next;

}

next = (next + 1) % NosOfProcess;

if (next == initiatorProcess) {

break;

}

}

System.out.println("Process " + process[getMax()].processId + " becomes coordinator");

} else {

System.exit(0);

}

}

}

public void putOutput() {

System.out.println("Processes in the ring: ");

for (int i = 0; i < NosOfProcess; i++) {

System.out.print(process[i].processId + ", active: " + process[i].active);

if (i == getMax()) {

System.out.print(", Coordinator\n");

} else {

System.out.print("\n");

}

}

}

private void sortProcess() {

for (int i = 0; i < NosOfProcess - 1; i++) {

for (int j = 0; j < (NosOfProcess - i) - 1; j++) {

if (process[j].processId > process[j + 1].processId) {

int temp = process[j].processId;

process[j].processId = process[j + 1].processId;

process[j + 1].processId = temp;

}

}

}

}

private int getMax(){

int max = 0, indexOfMax = 0;

for(int i = 0; i < NosOfProcess; i++) {

if(process[i].active && max <= process[i].processId) {

max = process[i].processId;

indexOfMax = i;

}

}

return indexOfMax;

}

public static void main(String arg[]) {

RingElection ringElection = new RingElection();

ringElection.getInput();

ringElection.conductElection();

}

}

//ds exp 8 **Mutual Exclusion**

import java.util.concurrent.locks.Lock;

public class Peterson implements Runnable {

private static boolean[] in = { false, false };

private static volatile int turn = -1;

public static void main(String[] args) {

new Thread(new Peterson(0), "Thread - 0").start();

new Thread(new Peterson(1), "Thread - 1").start();

}

private final int id;

public Peterson(int i) {

id = i;

}

private int other() {

return id == 0 ? 1 : 0;

}

public void run() {

in[id] = true;

turn = other();

while (in[other()] && turn == other()) {

System.out.println("[" + id + "] - Waiting...");

}

System.out.println("[" + id + "] - Working ("

+ ((!in[other()]) ? "other done" : "my turn") + ")");

in[id] = false;

}

}

//ds exp9 **Implement chat application using client-server model**

client1.java:

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class client1 {

public static void main(String [] args) {

Scanner sc = new Scanner(System.in);

String serverName = args[0];

int port = Integer.parseInt(args[1]);

int ch = 1;

while (ch == 1) {

try {

Socket client = new Socket(serverName, port);

OutputStream outToServer = client.getOutputStream();

DataOutputStream out = new DataOutputStream(outToServer);

System.out.println("Enter a message to send:");

String str = sc.nextLine();

out.writeUTF(str);

client.close();

System.out.print("Send another message? (0/1): ");

ch = sc.nextInt();

sc.nextLine();

} catch(IOException e) {

e.printStackTrace();

}

}

}

}

server1.java:

import java.net.\*;

import java.io.\*;

public class server1 extends Thread {

private ServerSocket serverSocket;

public server1(int port) throws IOException {

serverSocket = new ServerSocket(port);

serverSocket.setSoTimeout(60000);

System.out.println("Listening on "

+ Inet4Address.getLocalHost().getHostAddress() + ":6666 ...");

}

public void run() {

while (true) {

try {

Socket server = serverSocket.accept();

DataInputStream in = new DataInputStream(server.getInputStream());

System.out.println(in.readUTF());

server.close();

} catch(SocketTimeoutException s) {

System.out.println("Socket timed out!");

break;

} catch(IOException e) {

e.printStackTrace();

break;

}

}

}

public static void main(String[] args) {

int port = Integer.parseInt(args[0]);

try {

Thread t = new server1(port);

t.start();

} catch(IOException e) {

e.printStackTrace();

}

}

}