**Ex No: SOCKET PROGRAMMING**

**Date:**

**Aim:**

The aim is to implement Java programs for various socket communication scenarios,including UDP and TCP socket communication.

**Algorithm**

**STEP 1: Square Root**

**1.0 UDP Socket to Get the Square Root of a Given Number**

**Server (serverudp):**

* Create a DatagramSocket on a specified port (e.g., 8070).
* Enter a loop to listen for incoming UDP packets.
* Receive a DatagramPacket containing a message from the client.
* Convert the received message to an integer, calculate its square root, and print the result.

**Client (clientudp)**:

* Create a DatagramSocket.
* Take user input for a number.
* Convert the input to bytes and send a DatagramPacket to the server.
  1. **TCP Socket to Get the Square Root of a Given Number**

**Server (Servertcp):**

* Create a ServerSocket on a specified port (e.g., 8070).
* Enter a loop to accept incoming client connections.
* For each client, create a new thread to handle the client.
* In the client-handling thread, read the client's message, calculate the square root, and send the result back to the client.

**Client (Clienttcp):**

* Create a Socket and connect to the server on a specified IP and port (e.g., "127.0.0.1", 8070).
* Set up input and output streams.
* Take user input for a number.
* Send the input to the server.

**STEP 2: Sort an Array**

**2.0 UDP Socket to Sort an Array of Inputs**

**Server (serverudpsort):**

* Create a DatagramSocket on a specified port (e.g., 8070).
* Enter a loop to listen for incoming UDP packets.
* Receive a DatagramPacket containing an array of integers from the client.
* Convert the received bytes back to an array of integers, sort it, and print the sorted array.

**Client (Clientudpsort):**

* Create a DatagramSocket.
* Take user input for an array of numbers separated by commas.
* Convert the input to bytes and send a DatagramPacket to the server.

**2.1 TCP Socket to Sort an Array of Inputs**

**Server (clientudpsort):**

* Create a DatagramSocket on a specified port (e.g., 8090).
* Enter a loop to listen for incoming client connections.
* For each client, create a new thread to handle the client.
* In the client-handling thread, read the client's message, convert it to an array of integers, sort it, and print the sorted array.

**Client (clienttcpsort):**

* Create a Socket and connect to the server on a specified IP and port (e.g., "127.0.0.1", 8070).
* Take user input for an array of numbers separated by commas.
* Convert the input to integers, send the array size, and then send the array elements to the server.

**STEP 3: Multi-Client**

**3.0 UDP Chat Using Multi-Client**

**UDP Client (udpclientmessage):**

* Take user input for a name.
* Create a DatagramSocket.
* Start a new thread to handle incoming messages.
* In the main thread, send messages to the server until the user types "exit".

**UDP Server (udpservermessage):**

* Create a DatagramSocket on a specified port (e.g., 9876).
* Enter a loop to listen for incoming UDP packets.
* Receive a DatagramPacket containing a message from a client.

**STEP 4: Multi-Client Socket**

**4.0 Movie Ticket Booking System Using Multi-Client Socket Programming**

**Client**

* Take user input for the number of seats to book.
* Create a Socket and connect to the server on a specified IP and port (e.g., "localhost", 8888).
* Set up input and output streams.
* Send the number of seats to the server.
* Receive the booking response and available seats from the server.
* Print the result and repeat the process until the user enters 0.
* Close the socket, input, and output streams.

**Server**

* Create a ServerSocket on a specified port (e.g., 8888) and initialize the MovieTicket object.
* Enter a loop to accept incoming client connections.
* For each client, create a new thread (ClientHandler) to handle the client.
* In the ClientHandler thread, continuously read the client's request, process it, and send the response back to the client.
* Close the client socket when the client disconnects.

1. **TCP/UDPSocket to get the squareroot of the given number.**

**UDP Socket**

**UDP Server Program**

import java.net.DatagramPacket;

import java.net.DatagramSocket;

public class serverudp{

public static void main(String[] args) {

final int PORT = 8070;

try {

DatagramSocket serverSocket = new DatagramSocket(PORT);

System.out.println("Server listening on port " + PORT);

while (true) {

byte[] receiveData = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

serverSocket.receive(receivePacket);

String clientMessage = new String(receivePacket.getData(), 0, receivePacket.getLength());

int p= Integer.parseInt(clientMessage);

double squareRoot = Math.sqrt(p);

System.out.println("Square root : " + squareRoot);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

**UDP Client Program**

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.util.Scanner;

public class clientudp{

public static void main(String[] args) {

final String SERVER\_IP = "127.0.0.1"; // Replace with the server's IP address

final int SERVER\_PORT = 8070;

try {

DatagramSocket clientSocket = new DatagramSocket();

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number: ");

String userInput = scanner.nextLine();

byte[] sendData = userInput.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, InetAddress.getByName(SERVER\_IP), SERVER\_PORT);

clientSocket.send(sendPacket);

System.out.println("Sent to server: " + userInput);

scanner.close();

clientSocket.close();

} catch (Exception e) {

e.printStackTrace();

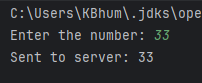
}

}

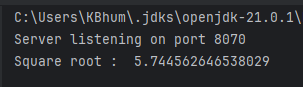
}

**UDP OUTPUT**

**Client**



**Server**



**TCP Socket**

**TCP Server Program**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class Servertcp {

public static void main(String[] args) {

final int PORT = 8070;

try {

ServerSocket serverSocket = new ServerSocket(PORT);

System.out.println("Server listening on port " + PORT);

while (true) {

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected: " + clientSocket.getInetAddress());

// Create a new thread to handle the client

Thread clientHandler = new Thread(() -> handleClient(clientSocket));

clientHandler.start();

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static void handleClient(Socket clientSocket) {

try {

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

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

// Read client's message

String clientMessage = in.readLine();

int p=Integer.parseInt(clientMessage);

double ans= Math.sqrt(p);

System.out.println("Square root is :" + ans );

in.close();

out.close();

clientSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

**TCP Client Program**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

import java.util.Scanner;

public class Clienttcp {

public static void main(String[] args) {

final String SERVER\_IP = "127.0.0.1";

final int SERVER\_PORT = 8070;

try {

Socket socket = new Socket(SERVER\_IP, SERVER\_PORT);

// Set up input and output streams

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

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

// Get user input

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number: ");

String userInput = scanner.nextLine();

// Send user input to the server

out.println(userInput);

System.out.println("Sent to server: " + userInput);

in.close();

out.close();

socket.close();

scanner.close();

} catch (IOException e) {

e.printStackTrace();

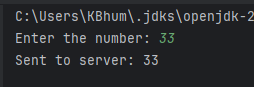
}

}

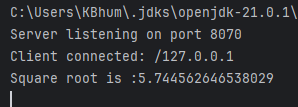
}

**TCP OUTPUT**

**CLIENT**

****

**SERVER**

****

1. **TCP/UDP socket to sort the array of inputs**

**UDP SOCKET**

**UDP Client Program**

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.util.Arrays;

import java.util.Scanner;

public class Clientudpsort {

public static void main(String[] args) {

final String SERVER\_IP = "127.0.0.1"; // Replace with the server's IP address

final int SERVER\_PORT = 8070;

try {

DatagramSocket clientSocket = new DatagramSocket();

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the numbers separated by commas: ");

String userInput = scanner.nextLine();

// Convert the input string to an array of integers

int[] numbers = Arrays.stream(userInput.split(","))

.mapToInt(Integer::parseInt)

.toArray();

// Convert the array to bytes

byte[] sendData = new byte[4 \* numbers.length];

for (int i = 0; i < numbers.length; i++) {

sendData[i \* 4] = (byte) (numbers[i] >> 24);

sendData[i \* 4 + 1] = (byte) (numbers[i] >> 16);

sendData[i \* 4 + 2] = (byte) (numbers[i] >> 8);

sendData[i \* 4 + 3] = (byte) (numbers[i]);

}

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,

InetAddress.getByName(SERVER\_IP), SERVER\_PORT);

clientSocket.send(sendPacket);

System.out.println("Sent to server: " + Arrays.toString(numbers));

scanner.close();

clientSocket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

**UDP Server Program**

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.util.Arrays;

public class serverudpsort {

public static void main(String[] args) {

final int PORT = 8070;

try {

DatagramSocket serverSocket = new DatagramSocket(PORT);

System.out.println("Server listening on port " + PORT);

while (true)

byte[] receiveData = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

serverSocket.receive(receivePacket);

// Convert the received bytes back to an array of integers

int[] numbers = new int[receivePacket.getLength() / 4];

for (int i = 0; i < numbers.length; i++) {

numbers[i] = (receivePacket.getData()[i \* 4] << 24) |

((receivePacket.getData()[i \* 4 + 1] & 0xFF) << 16) |

((receivePacket.getData()[i \* 4 + 2] & 0xFF) << 8) |

(receivePacket.getData()[i \* 4 + 3] & 0xFF);

}

// Sort the array

Arrays.sort(numbers);

System.out.println("Sorted array: " + Arrays.toString(numbers));

}

} catch (Exception e) {

e.printStackTrace();

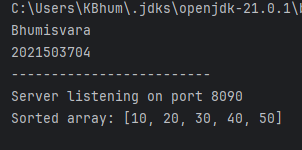
}

}

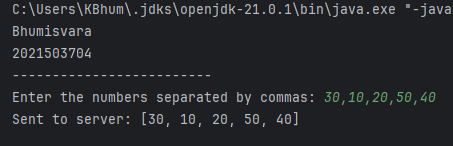
}

**OUTPUT**

**UDP Server**



**UDP Client**



**TCP SOCKET**

**TCP SERVER Program**

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.util.Arrays;

import java.util.Scanner;

public class clientudpsort {

public static void main(String[] args) {

System.out.println("Bhumisvara");

System.out.println("2021503704");

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

final String SERVER\_IP = "127.0.0.1"; // Replace with the server's IP address

final int SERVER\_PORT = 8090;

try {

DatagramSocket clientSocket = new DatagramSocket();

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the numbers separated by commas: ");

String userInput = scanner.nextLine();

// Convert the input string to an array of integers

int[] numbers = Arrays.stream(userInput.split(","))

.mapToInt(Integer::parseInt)

.toArray();

// Convert the array to bytes

byte[] sendData = new byte[4 \* numbers.length];

for (int i = 0; i < numbers.length; i++) {

sendData[i \* 4] = (byte) (numbers[i] >> 24);

sendData[i \* 4 + 1] = (byte) (numbers[i] >> 16);

sendData[i \* 4 + 2] = (byte) (numbers[i] >> 8);

sendData[i \* 4 + 3] = (byte) (numbers[i]);

}

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,

InetAddress.getByName(SERVER\_IP), SERVER\_PORT);

clientSocket.send(sendPacket);

System.out.println("Sent to server: " + Arrays.toString(numbers));

scanner.close();

clientSocket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

**TCP CLIENT Program**

import java.io.DataOutputStream;

import java.net.Socket;

import java.util.Arrays;

import java.util.Scanner;

public class clienttcpsort {

public static void main(String[] args) {

System.out.println("Bhumisvara");

System.out.println("2021503704");

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

final String SERVER\_IP = "127.0.0.1"; // Replace with the server's IP address

final int SERVER\_PORT = 8070;

try {

Socket clientSocket = new Socket(SERVER\_IP, SERVER\_PORT);

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the numbers separated by commas: ");

String userInput = scanner.nextLine();

// Convert the input string to an array of integers

int[] numbers = Arrays.stream(userInput.split(","))

.mapToInt(Integer::parseInt)

.toArray();

// Send the array size to the server

DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());

outToServer.writeInt(numbers.length);

// Send the array to the server

for (int number : numbers) {

outToServer.writeInt(number);

}

System.out.println("Sent to server: " + Arrays.toString(numbers));

scanner.close();

clientSocket.close();

} catch (Exception e) {

e.printStackTrace();

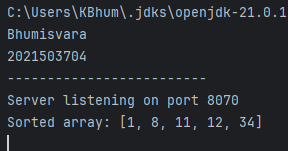
}

}

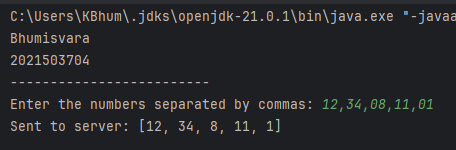
}

**OUTPUT**

**TCP SERVER**

****

**TCP CLIENT**

****

1. **TCP/UDP chat using multiclient**

**UDP CLIENT**

**Program**

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.util.Scanner;

public class udpclientmessage {

private static final int SERVER\_PORT = 9876;

public static void main(String[] args) {

try {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter your name: ");

String name = scanner.nextLine();

InetAddress serverAddress = InetAddress.getByName("localhost");

DatagramSocket clientSocket = new DatagramSocket();

// Start a separate thread to handle incoming messages

new Thread(() -> {

while (true) {

receiveMessage(clientSocket);

}

}).start();

// Start sending messages

sendMessage(clientSocket, serverAddress, name);

} catch (IOException e) {

e.printStackTrace();

}

}

private static void sendMessage(DatagramSocket socket, InetAddress serverAddress, String name) {

Scanner scanner = new Scanner(System.in);

try {

while (true) {

System.out.print("Enter message (type 'exit' to stop): ");

String message = scanner.nextLine();

// Format the message with name and send to the server

message = "[" + name + "]: " + message;

byte[] sendData = message.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, SERVER\_PORT);

socket.send(sendPacket);

// Check for the exit condition

if (message.equalsIgnoreCase("[exit]")) {

System.out.println("Exiting the chat. Goodbye!");

break;

}

}

} catch (IOException e) {

e.printStackTrace();

} finally {

if (!socket.isClosed()) {

socket.close();

}

}

}

private static void receiveMessage(DatagramSocket socket) {

try {

byte[] receiveData = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

socket.receive(receivePacket);

String message = new String(receivePacket.getData(), 0, receivePacket.getLength());

System.out.println("Received message: " + message);

} catch (IOException e) {

e.printStackTrace();

}

}

}

**UDP SERVER**

**Program**

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.util.HashMap;

import java.util.Map;

public class udpservermessage {

private static final int PORT = 9876;

private static Map<InetAddress, Integer> clientMap = new HashMap<>();

public static void main(String[] args) {

try {

DatagramSocket serverSocket = new DatagramSocket(PORT);

System.out.println("Server is running on port " + PORT);

while (true) {

byte[] receiveData = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

serverSocket.receive(receivePacket);

String message = new String(receivePacket.getData(), 0, receivePacket.getLength());

InetAddress clientAddress = receivePacket.getAddress();

int clientPort = receivePacket.getPort();

if (!clientMap.containsKey(clientAddress)) {

clientMap.put(clientAddress, clientPort);

System.out.println("New client connected: " + clientAddress + ":" + clientPort);

}

System.out.println(clientAddress + ":" + clientPort + " - " + message);

broadcastMessage(message, clientAddress, clientPort);

if (message.equalsIgnoreCase("exit")) {

System.out.println("Client " + clientAddress + ":" + clientPort + " has exited.");

clientMap.remove(clientAddress);

}

}

} catch (IOException e) {

e.printStackTrace();

}

private static void broadcastMessage(String message, InetAddress senderAddress, int senderPort) {

DatagramSocket socket = null;

try {

socket = new DatagramSocket();

for (Map.Entry<InetAddress, Integer> entry : clientMap.entrySet()) {

InetAddress clientAddress = entry.getKey();

int clientPort = entry.getValue();

if (!(clientAddress.equals(senderAddress) && clientPort == senderPort)) {

byte[] sendData = message.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, clientAddress, clientPort);

socket.send(sendPacket);

}

}

} catch (IOException e) {

e.printStackTrace();

} finally {

if (socket != null && !socket.isClosed()) {

socket.close();

}

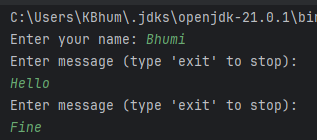
}

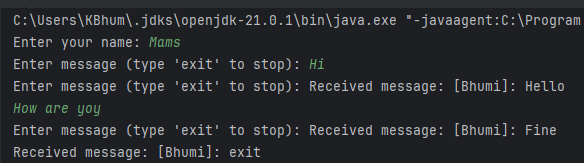
}

}

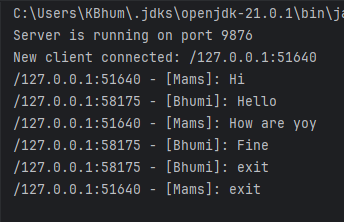
**OUTPUT**

**CLIENT**





**SERVER**



1. **Write a java code for movie ticket booking system using multi-client socket programming. Implement the synchronization mechanism of ticket booking operations at server side to handle multiple clients concurrently.**

**CLIENT:**

**Program**

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.Socket;

import java.util.Scanner;

public class ques4C {

public static void main(String[] args) {

final String SERVER\_IP = "localhost";

final int PORT = 8888;

try (Socket socket = new Socket(SERVER\_IP, PORT);

ObjectOutputStream outputStream = new ObjectOutputStream(socket.getOutputStream());

ObjectInputStream inputStream = new ObjectInputStream(socket.getInputStream());

Scanner scanner = new Scanner(System.in)) {

while (true) {

System.out.print("Enter the number of seats to book (0 to exit): ");

int numSeats = scanner.nextInt();

if (numSeats == 0) {

break;

}

// Send the request to the server

outputStream.writeInt(numSeats);

outputStream.flush();

// Receive the response from the server

boolean bookingSuccess = inputStream.readBoolean();

int availableSeats = inputStream.readInt();

if (bookingSuccess) {

System.out.println("Booking successful! Available seats: " + availableSeats);

} else {

System.out.println("Booking failed. Not enough seats available.");

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**SERVER:**

**Program**

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.HashMap;

import java.util.Map;

class MovieTicket {

private int availableSeats;

public MovieTicket(int totalSeats) {

this.availableSeats = totalSeats;

}

public synchronized boolean bookTicket(int numSeats) {

if (numSeats > 0 && numSeats <= availableSeats) {

availableSeats -= numSeats;

return true;

}

return false;

}

public synchronized int getAvailableSeats() {

return availableSeats;

}

}

class ClientHandler extends Thread {

private Socket clientSocket;

private MovieTicket movieTicket;

private ObjectOutputStream outputStream;

private ObjectInputStream inputStream;

public ClientHandler(Socket socket, MovieTicket movieTicket) {

this.clientSocket = socket;

this.movieTicket = movieTicket;

try {

outputStream = new ObjectOutputStream(clientSocket.getOutputStream());

inputStream = new ObjectInputStream(clientSocket.getInputStream());

} catch (IOException e) {

e.printStackTrace();

}

}

@Override

public void run() {

try {

while (true) {

// Read the client's request

int numSeatsRequested = inputStream.readInt();

// Process the request

boolean bookingSuccess = movieTicket.bookTicket(numSeatsRequested);

// Send the response back to the client

outputStream.writeBoolean(bookingSuccess);

outputStream.writeInt(movieTicket.getAvailableSeats());

outputStream.flush();

}

} catch (IOException e) {

// Handle IOException

} finally {

try {

clientSocket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

public class ques4S {

public static void main(String[] args) {

final int PORT = 8888;

final int TOTAL\_SEATS = 50;

MovieTicket movieTicket = new MovieTicket(TOTAL\_SEATS);

try (ServerSocket serverSocket = new ServerSocket(PORT)) {

System.out.println("Server is running and waiting for clients...");

while (true) {

Socket clientSocket = serverSocket.accept();

System.out.println("New client connected");

ClientHandler clientHandler = new ClientHandler(clientSocket, movieTicket);

clientHandler.start();

}

} catch (IOException e) {

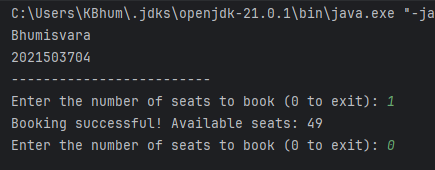
e.printStackTrace();

}

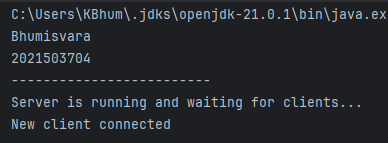
}

}

**OUTPUT  
  
CLIENT**

****

**SERVER**

****

**RESULT**

Thus, The Socket program has been successfully implemented.