EXPERIMENT 7

Aim

Introduction to Socket Programming- Design and Implement client-server elements of a few network applications e.g., Echo client and server, Time client and server, Online Quiz and Buzzer Application, etc.

Prerequisite

Nil

Outcome

To impart knowledge of Computer Networking Technology

Theory

Socket programming is a fundamental technique for establishing communication between two or more processes, either on the same machine or over a network. Sockets serve as the endpoints for this communication, allowing data to be transmitted and received. These sockets can be created and utilized in various programming languages, with Java and Python being commonly used choices for implementing network applications.

Key concepts in socket programming include:

1. **Socket Types:** There are two primary types of sockets: stream sockets (e.g., TCP) and datagram sockets (e.g., UDP). Stream sockets provide reliable, connection-oriented communication, while datagram sockets offer connectionless, unreliable communication. The choice of socket type depends on the requirements of the application.
2. **IP Addresses and Port Numbers:** In socket programming, each socket is associated with an IP address and a port number. The IP address identifies the host (machine) in the network, and the port number specifies the endpoint within that host. This combination is used to establish connections and facilitate data exchange.
3. **Server-Client Model:** Socket programming typically follows a client-server architecture, where one entity (the server) listens for incoming connections, while other entities (clients) connect to the server to exchange data. This model is used in various network applications, including web servers, chat applications, and online games.
4. **Socket Operations:**
    - **Socket Creation:** Sockets are created using programming language-specific functions or classes. The creation process involves specifying the socket type (stream or datagram) and the communication protocol (e.g., TCP, UDP).
    - **Binding:** Servers must bind their sockets to a specific IP address and port number, enabling clients to connect to the server at that address and port.
    - **Listening:** Server sockets listen for incoming connection requests from clients. This listening state allows multiple clients to connect to a single server.

- **Connection Establishment:** Clients initiate connections to the server by specifying the server's IP address and port number. Once a connection is established, data can be exchanged bidirectionally.
- **Sending and Receiving Data:** Both clients and servers can send and receive data through their sockets. Stream sockets offer reliable, ordered data transmission, while datagram sockets provide a connectionless mode for sending data.
- **Closing Sockets:** Properly closing sockets is essential to release network resources and terminate communication. Failing to close sockets can lead to resource leaks and other issues.

Procedure

1. Write Simple Client Server Program using Java/Python Programming Languag
2. Execute the program using appropriate compiler.
3. Verify the working of the program.

Steps

1. Echo Server-Client Implementation

EchoServer.java

```
import java.io.*;
import java.net.*;

public class EchoServer {
    public static void main(String[] args) throws Exception {
        ServerSocket serverSocket = new ServerSocket(12345);
        System.out.println("Echo Server is running and listening on port 12345...");

        while (true) {
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected: " + clientSocket.getInetAddress());

            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

            String message;
            while ((message = in.readLine()) != null) {
                System.out.println("Received from client: " + message);
                out.println("Server Echo: " + message);
            }

            clientSocket.close();
            System.out.println("Client disconnected.");
        }
    }
}
```

EchoClient.java

```
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) throws Exception {
        Socket socket = new Socket("127.0.0.1", 12345);
        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

        String message = "Hello, server!";
        out.println(message)
        String response = in.readLine();
        System.out.println("Received from server: " + response);

        socket.close();

    }
}
```

2. Time Server-Client Implementation

TimeServer.java

```
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Date;

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

        ServerSocket serverSocket = new ServerSocket(12345);
        System.out.println("Time Server is running and listening on port 12345...");

        while (true) {
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected: " + clientSocket.getInetAddress());

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

            clientSocket.close();
            System.out.println("Client disconnected.");
        }
    }
}
```

TimeClient.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.Socket;

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

        Socket socket = new Socket("127.0.0.1", 12345);
        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

        String response = in.readLine();
        System.out.println("Server time: " + response);

        socket.close();

    }
}
```

3. Online Quiz Buffer Implementation

QuizServer.java

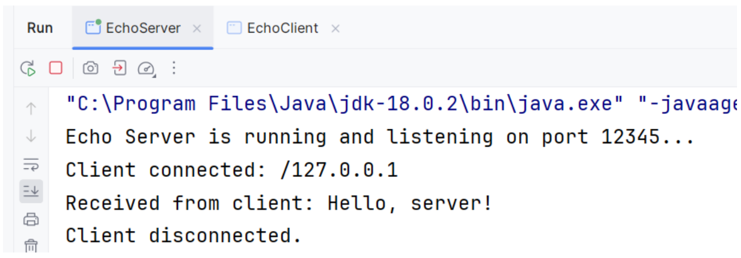
```
import java.io.*;
import java.net.*;
import java.util.*;

public class QuizServer {

    private Map<Integer, String> questions;
    private Map<Integer, String> answers;

    public QuizServer() {
        questions = new HashMap<>();
        answers = new HashMap<>();
        questions.put(1, "What is the capital of France?");
        answers.put(1, "Paris");

        questions.put(2, "Which planet is known as the Red Planet?");
        answers.put(2, "Mars");

        questions.put(3, "How many continents are there on Earth?");
        answers.put(3, "7");
    }
    public void start() throws IOException {
        ServerSocket serverSocket = new ServerSocket(12345);

        while (true) {
            Socket clientSocket = serverSocket.accept();

            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

            // Start the quiz
            for (int questionNumber = 1; questionNumber <= 3; questionNumber++) {
                // Send the question to the client
                out.println("Question " + questionNumber + ": " + questions.get(questionNumber));

                // Receive the client's answer
                String answer = in.readLine();

                // Check if the answer is correct
                if (answer.equals(answers.get(questionNumber))) {
                    // The answer is correct
                    out.println("Correct!");
                } else {
                    // The answer is incorrect
                    out.println("Incorrect. The correct answer is: " + answers.get(questionNumber));
                }
            }

            // The quiz is finished
            out.println("Quiz finished!");

            // Close the socket
            clientSocket.close();
        }
    }

    public static void main(String[] args) throws IOException {
        QuizServer quizServer = new QuizServer();
        quizServer.start();
    }
}
```

QuizClient.java

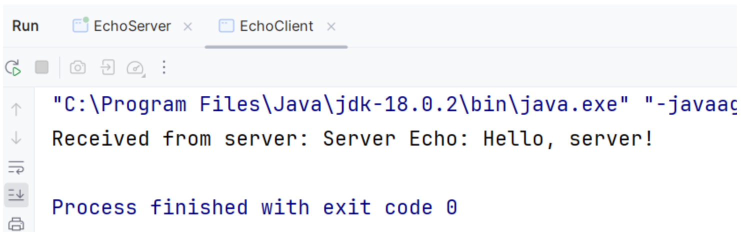
```
import java.io.*;
import java.net.*;

public class QuizClient {

    public static void main(String[] args) throws Exception {
        Socket socket = new Socket("localhost", 12345);

        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

        // Start the quiz
        while (true) {
            // Receive the question from the server
            String question = in.readLine();

            // Print the question to the console
            System.out.println(question);

            // Get the user's answer
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
            String answer = reader.readLine();

            // Send the answer to the server
            out.println(answer);

            // Receive the result from the server
            String result = in.readLine();

            // Display the result to the user
            System.out.println(result);

            // Check if the quiz is finished
            if (result.equals("Quiz finished!")) {
                break;
            }
        }

        // Close the socket
        socket.close();
    }
}
```

Output

Echo Server-Client

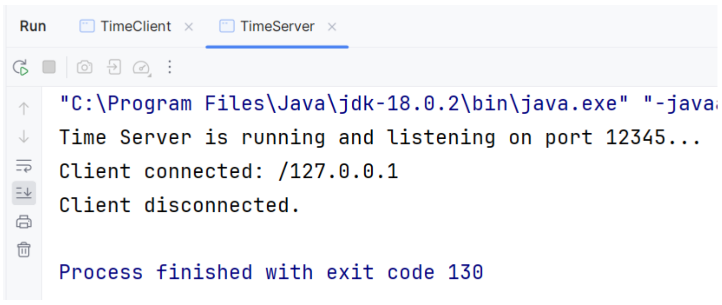


```
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" "-javaage
Echo Server is running and listening on port 12345...
Client connected: /127.0.0.1
Received from client: Hello, server!
Client disconnected.
```

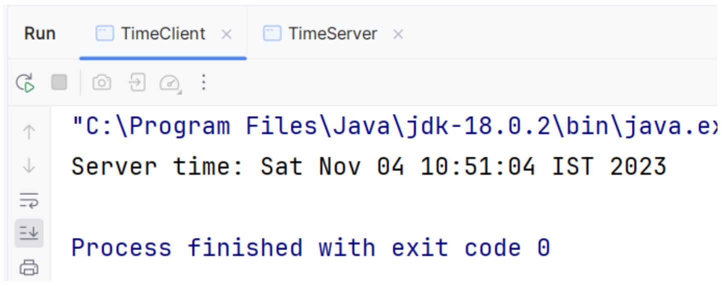


```
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" "-javaag
Received from server: Server Echo: Hello, server!

Process finished with exit code 0
```

Time Server-Client

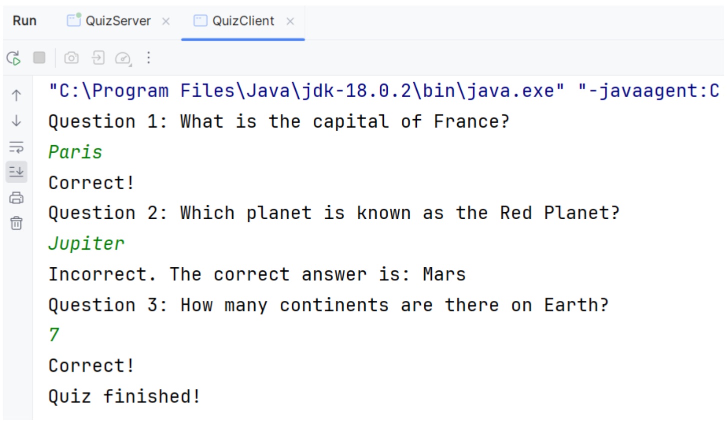


```
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" "-java
Time Server is running and listening on port 12345...
Client connected: /127.0.0.1
Client disconnected.

Process finished with exit code 130
```



```
"C:\Program Files\Java\jdk-18.0.2\bin\java.e
Server time: Sat Nov 04 10:51:04 IST 2023

Process finished with exit code 0
```

Quiz Buffer Server-Client



```
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" "-javaagent:C
Question 1: What is the capital of France?
Paris
Correct!
Question 2: Which planet is known as the Red Planet?
Jupiter
Incorrect. The correct answer is: Mars
Question 3: How many continents are there on Earth?
7
Correct!
Quiz finished!
```

Observation & Learning

During the experiment, participants observed and learned the following key points:

- How to create and use socket objects for communication.
- The basics of establishing a connection between a client and a server.
- How data is exchanged between the client and server.
- The importance of port numbers and IP addresses in socket programming.
- How to implement specific functionalities for different network applications, such as Echo, Time, Quiz, or Buzzer.

Conclusion

In conclusion, this practical has demonstrated the implementation of various client-server applications using Java socket programming. These applications showcase the basic principles of client-server communication, including network protocols, socket programming, and error handling. The practical has also provided insights into designing scalable and efficient server applications.

Questions

1. **What is a Socket?**

   A socket is a software endpoint used for communication between processes over a network. It allows data to be sent and received over a network connection.

2. **Which socket is used for communication between the client and server?**

   Both the client and server use sockets to establish communication. The client and server each create a socket to send and receive data.

3. **What are the different operations supported on sockets?**

   Sockets support various operations, including:

   - Socket creation
   - Binding a socket to an IP address and port number
   - Listening for incoming connections (for servers)
   - Establishing connections (for clients)
   - Sending and receiving data
   - Closing the socket connection