LAB Manual – 1

Code/Course: 23CSE312/Distributed Systems

Topic: Distributed Applications using Sockets(Chat Program)

[CO05] [BTL-3]
Student Name: _____
Roll No.: _____

Batch/Section: _____
Date: _____

1 Introduction

Socket programming is a technique used to enable communication between two programs over a network, such as the Internet or a local area network, using sockets as communication endpoints.

2 What is a Socket?

A socket is an endpoint of a two-way communication link between two processes. It enables data exchange between applications running on different machines or on the same machine.

- One socket belongs to the server.
- One socket belongs to the client.
- Communication occurs by sending and receiving data through these sockets.

A socket can be visualized as a virtual plug point through which data flows between communicating entities.

3 Why Socket Programming is Used

Socket programming is used when applications need to:

- Communicate over a network.
- Exchange data in real time.
- Follow a client–server communication model.

3.1 Common Applications

- Web servers (HTTP)
- Email services (SMTP, POP, IMAP)
- Chat applications
- File transfer systems (FTP)
- Online games
- Distributed systems

4 Types of Sockets

4.1 TCP Sockets (Stream Sockets)

TCP sockets use the Transmission Control Protocol and provide reliable, ordered, and error-checked delivery of data.

- Connection-oriented
- Reliable and ordered data delivery
- Slower but accurate

Examples: Web browsing, file transfer.

4.2 UDP Sockets (Datagram Sockets)

UDP sockets use the User Datagram Protocol and provide faster data transmission without delivery guarantees.

- Connectionless
- Faster but unreliable
- No guarantee of data delivery or order

Examples: Video streaming, online gaming.

5 Client–Server Model in Socket Programming

5.1 Server Side

1. Create a socket.
2. Bind the socket to an IP address and port number.
3. Listen for incoming connections.
4. Accept the client connection.
5. Send and receive data.
6. Close the socket.

5.2 Client Side

1. Create a socket.
2. Connect to the server socket.
3. Send and receive data.
4. Close the socket.

1. Aim

To build a basic distributed chat application using Python sockets with:

- **Part A: Iterative server and client** (single connection served at a time).
- **Part B: Concurrent server and client** (multiple clients served simultaneously).

2. Objectives

At the end of this laboratory experiment, the student will be able to:

- Understand the fundamentals of socket-based communication in Python.
- Implement a basic client–server architecture using TCP sockets.
- Develop an iterative server capable of handling one client connection at a time.
- Design a concurrent server that can handle multiple client connections simultaneously using threading.
- Execute and validate client–server communication across multiple systems within the same network.

3. Reference (Get Started)

Use the following resource to study socket concepts and start from a basic server-client example:

`https://www.geeksforgeeks.org/socket-programming-python/`

4. System Requirements

- Python 3.x installed
- Two terminals (server + client) on the same machine for initial testing
- (For cross-system test) Two systems connected to the same network (LAN/Wi-Fi)

Task 1) Iterative-server and client

Create a most basic distributed application consisting of a client and a server program which can chat with each other. Make the server iterative, i.e., accept and serve a single connection at a time. Follow the steps below:

a) Try the "simple server-client program" from the link given before

b) Modify it for the chat application

c) Once the above works in your system, pair up with your neighbour and try the client and server across two systems connected in the same network

Task 2) Concurrent-server and client

Modify the chat program to make the server concurrent, i.e., the server can establish connections with multiple clients simultaneously.

Refeerence: Multi-threading

Study the associated socket concepts from the above link.

Submission Instruction

The submission for both Task 1 and Task 2 must include the following components, accompanied by appropriate and well-documented comments.

1. `server.py`
2. `client.py`
3. Screenshot of output from both server and client

Conclusion