Lab 3: Socket Programming

In this lab, we will write Python programs to create a client and server and explore how they communicate with each other using the concept of socket programming. Recalling what we studied in Lab 2, where a server responds to client requests, we'll develop our own programs using Python's socket library. Additionally, we will be introduced to the concept of multi-threading, enabling our server to handle requests from multiple clients simultaneously and respond appropriately.

WHAT IS SOCKET PROGRAMMING?

A socket is a way for programs to communicate with each other over a network. It's like a phone line that lets computers send and receive data.

Socket programming involves writing programs that run on different computers and communicate over a network. This can be either connection-oriented (like making a phone call) or connectionless (like sending a letter). The basic way to do this is to import the socket library in Python, then create a socket object and then bind that object with a socket address. A socket address is a combination of an IP address and a port number. It uniquely identifies a network connection on a specific device. For example, a device with an IPv4 address of 192.168.10.1 and port number 5050 will have a socket address of 192.168.10.1:5050.

PYTHON'S SOCKET LIBRARY

Python provides a socket library to make socket programming easier. Here are some important methods:

- **socket():** Creates a new socket object.

- **bind((hostname, port)):** Binds the server to a specific IP address and port number.
- **listen(backlog):** Prepares the server to accept connections. The backlog defines how many connections can wait in line.
- **accept():** Accepts a new connection. Returns a new socket object and the address of the client.
- **connect((hostname, port)):** Connects the client to the server at the specified address.
- **send(message):** Sends data to the connected socket.
- **recv(buffer_size):** Receives data from the connected socket.
- **close():** Closes the socket connection.

MULTI-THREADING CONCEPT

In our initial tasks, we will see the communication between one server and one client. But in real life, multiple clients request the same server at the same time. It is not feasible and efficient to build individual servers for each client. So, what can we do in this case? This is when we apply the concept of multi-threading. Multi-threading allows a server to handle multiple client requests at the same time. This means the server can talk to many clients without making them wait for each other.

We can implement this by importing another Python library named ‘threading’. It allows the server to handle multiple client connections at the same time by creating a separate thread for each connection. A new thread is created specifically to handle the communication with the connected client. This thread can send and receive messages independently for that specific client. Each thread runs independently, enabling the server to manage multiple clients concurrently without waiting for one connection to finish before starting another. Furthermore, if one client's connection is slow or blocked, it doesn't affect the handling of other clients.

OUTCOME

By the end of this lab, you should be able to write Python programs that use sockets to communicate over a network and handle multiple clients at the same time.