LAB 1

Aim: To implement client server application

Lab Outcome:

Develop test and debug using Message-Oriented Communication or RPC/RMIbased

client-server programs

1 Client and 1 Server :

**Server.py**

import socket

def server\_program():

# get the hostname

host = socket.gethostname()

port = 5000 # initiate port no above 1024

server\_socket = socket.socket() # get instance

# look closely. The bind() function takes tuple as argument

server\_socket.bind((host, port)) # bind host address and port together

# configure how many client the server can listen simultaneously

server\_socket.listen(2)

conn, address = server\_socket.accept() # accept new connection

print("Connection from: " + str(address))

while True:

# receive data stream. it won't accept data packet greater than 1024 bytes

data = conn.recv(1024).decode()

if not data:

# if data is not received break

break

print("from connected user: " + str(data))

data = input(' -> ')

conn.send(data.encode()) # send data to the client

conn.close() # close the connection

if \_\_name\_\_ == '\_\_main\_\_':

server\_program()

**Client.py**

import socket

def client\_program():

host = socket.gethostname() # as both code is running on same pc

port = 5000 # socket server port number

client\_socket = socket.socket() # instantiate

client\_socket.connect((host, port)) # connect to the server

message = input(" -> ") # take input

while message.lower().strip() != 'bye':

client\_socket.send(message.encode()) # send message

data = client\_socket.recv(1024).decode() # receive response

print('Received from server: ' + data) # show in terminal

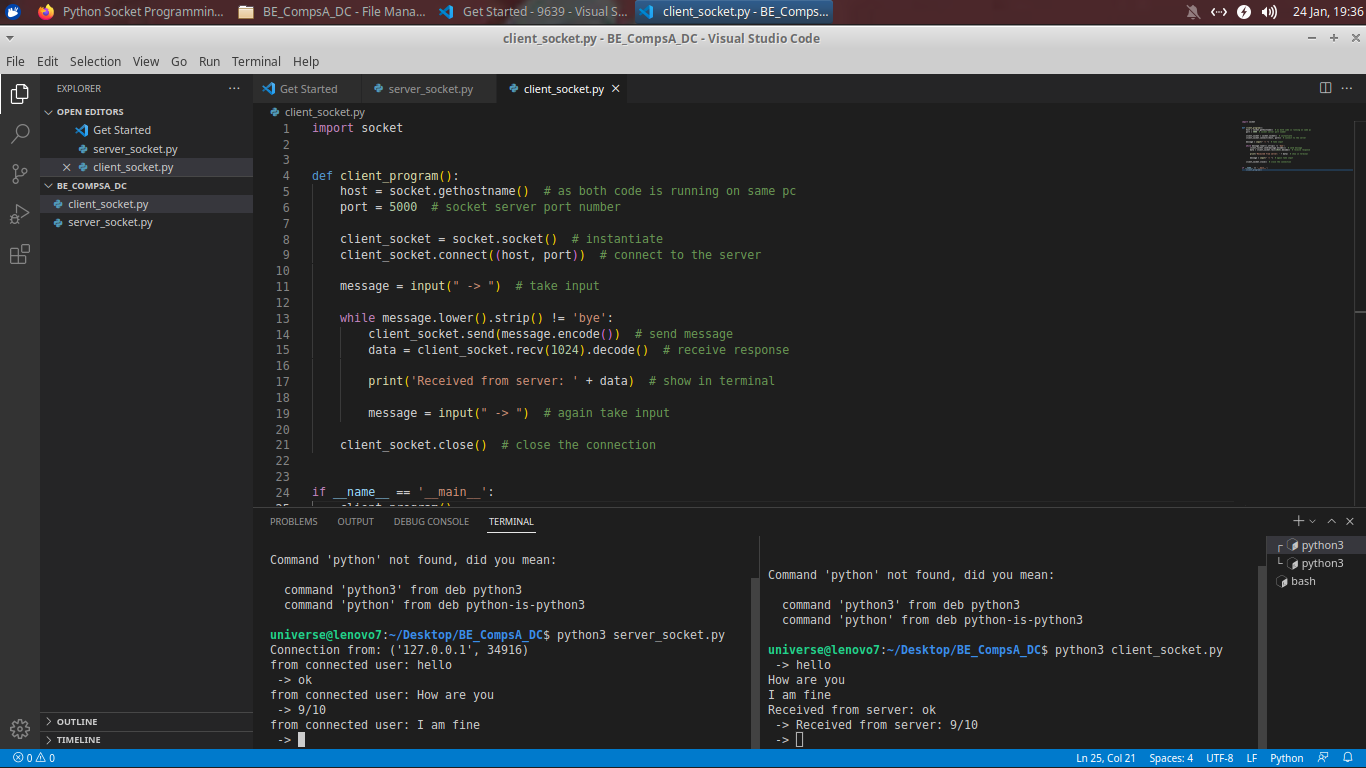
message = input(" -> ") # again take input

client\_socket.close() # close the connection

if \_\_name\_\_ == '\_\_main\_\_':

client\_program()

**Output :**

****

**Multithreading**

**2 Client and 1 Server**

**Server.py**

import socket

import threading

import time

# Function to handle client connections

def handle\_client(client\_socket, address):

print(f"Accepted connection from {address}")

while True:

# Send stock value to the client

client\_socket.send(str(stock\_value).encode())

# Sleep for 2 minutes before updating stock value

time.sleep(120)

# Function to update stock value

def update\_stock\_value():

global stock\_value

while True:

# Simulate updating stock value

stock\_value += 1

print(f"Updated stock value: {stock\_value}")

# Sleep for 2 minutes

time.sleep(120)

# Server configuration

host = "127.0.0.1"

port = 8888

# Create a socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Bind the socket to a specific address and port

server\_socket.bind((host, port))

# Listen for incoming connections

server\_socket.listen(5)

print(f"Server listening on {host}:{port}")

# Global variable for stock value

stock\_value = 100

# Start a thread to update stock value

update\_thread = threading.Thread(target=update\_stock\_value)

update\_thread.start()

while True:

# Accept a connection from a client

client\_socket, addr = server\_socket.accept()

# Start a new thread to handle the client

client\_handler = threading.Thread(target=handle\_client, args=(client\_socket, addr))

client\_handler.start()

**Client1.py**

import socket

# Client configuration

host = "127.0.0.1"

port = 8888

# Create a socket

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Connect to the server

client\_socket.connect((host, port))

print(f"Connected to {host}:{port}")

while True:

# Receive and print the stock value from the server

data = client\_socket.recv(1024)

stock\_value = data.decode()

print(f"Stock value received: {stock\_value}")

**Client2.py**

import socket

# Client configuration

host = "127.0.0.1"

port = 8888

# Create a socket

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Connect to the server

client\_socket.connect((host, port))

print(f"Connected to {host}:{port}")

while True:

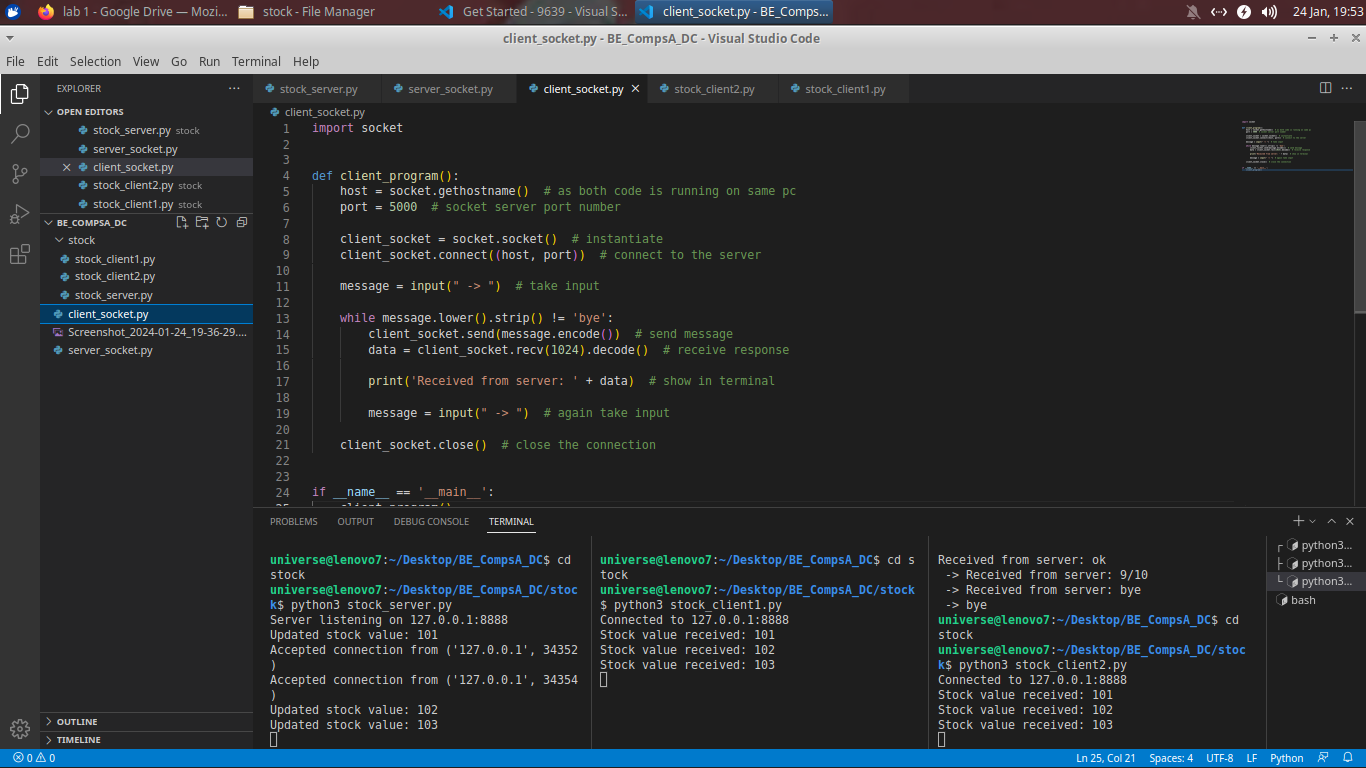
# Receive and print the stock value from the server

data = client\_socket.recv(1024)

stock\_value = data.decode()

print(f"Stock value received: {stock\_value}")

**Output**

****

**Steps to run the application**

1. Start server program. It will be ready to accept connection from the client.

2. On another terminal start client program and send some message to server.

3. Server will display the output.

Conclusions :

The experiment demonstrated the difference between single threading and multithreading in

a client-server model. Single threading showed limitations in handling multiple clients

simultaneously, whereas multithreading improved the performance by handling multiple

clients at the same time. This highlights the importance of multithreading in server design for

improved efficiency and scalability.

Post lab Questions:

1. Enlist the socket primitives.

2. What are the advantages of a Multithreaded Server?

3. With an example, explain the concept of multithreaded clients.

4. What are the motivations for using threads.

5.What are the typical models for Organizing threads.

