Fr. Conceicao Rodrigues College of Engineering

Department of Computer Engineering

Academic Year 2022-23

Distributed Computing Lab (B.E. Computer Engineering)

Experiment List

Sr.No.	Title of Experiment
1	To implement Client Server Application.
2	To implement Remote Procedure Call.
3	To implement Remote Method Invocation.
4	To implement Message Queueing System.
5	To implement Group Communication.
6	To implement Lamport Algorithm for Logical Clock Synchronization.
7	To implement techniques for Election Algorithms.
8	To implement Mutual Exclusion or Deadlock Detection.
9	To implement Stateful and Stateless File Server.
10	To study HDFS and MapReduce.

LAB 1

Aim: To implement client server application

Lab Outcome:
Develop test and debug using Message-Oriented Communication or RPC/RMIbased client-server programs

Theory:

Client-Server Model

The client-server model is a decentralized computing architecture that involves dividing tasks or workloads between servers and clients. The servers provide centralized resources and services, while clients request and utilize these resources. This model enables clients to access shared data and services, while servers can manage and control access to resources. It helps to distribute the workload and reduce the burden on any single device or component. The client-server model is used in a variety of applications, including web services, email, and database systems, and is particularly useful for supporting large numbers of users or clients. This architecture is also flexible and scalable, making it a popular choice for many organizations.

Socket

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.
They are the real backbones behind web browsing. In simpler terms, there is a server and a client. Socket programming is started by importing the socket library and making a simple socket.

Function Call	Description
Socket()	To create a socket
Bind()	It's a socket identification like a telephone number to contact
Listen()	Ready to receive a connection
Connect()	Ready to act as a sender
Accept()	Confirmation, it is like accepting to receive a call from a sender
Write()	To send data
Read()	To receive data
Close()	To close a connection

Single-threading

Single-threading is a computer programming model that processes and executes tasks sequentially in a single thread of execution. In this model, the processor runs one task at a time and is unable to execute multiple tasks simultaneously. As a result, single-threaded applications can only perform one operation at a time and cannot take advantage of multi-core processors. Single-threading is simple and easy to implement, but it can be limited in terms of performance, particularly for complex or demanding tasks. However, it is still commonly used in applications where the task execution time is relatively short and predictable, or where the need for multi-threading is low. Single-threaded programming can also be used to ensure that tasks are executed in a specific order, which can be important in certain applications, such as financial transactions.

Multi-threading

Multithreading is the ability of a program or an operating system process to manage its useby more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer. Each user request for a program or system service(and here a user can also be another program) is kept track of as a thread with a separate identity. As programs work on behalf of the initial request for that thread and are interrupted by other requests, the status of work on behalf ofthat thread is kept track of until the work is completed.

Developing Client-Server Application

In this experiment we aim at developing Client–Server application using multithreading concepts. In this application server handles the requests from multiple clients.

For this experiment socket programing is used. In client program the parameters like port number, server name are taken in order to connect with server. In the server program it acceptsthe connection from client and assigns them to new connection handler object. Then one thread at server side handles new connections and another thread is for clients.

Multithread client server
example:Web client:

- Web browser scans an incoming HTML page, and finds that more files need to befetched
- Each file is fetched by a separate thread, each doing a (blocking) HTTP request
- As files come in, the browser displays them

A multithreaded server organized in a dispatcher/worker model.

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.

(Code & Output)

1 client 1 server

Client.py

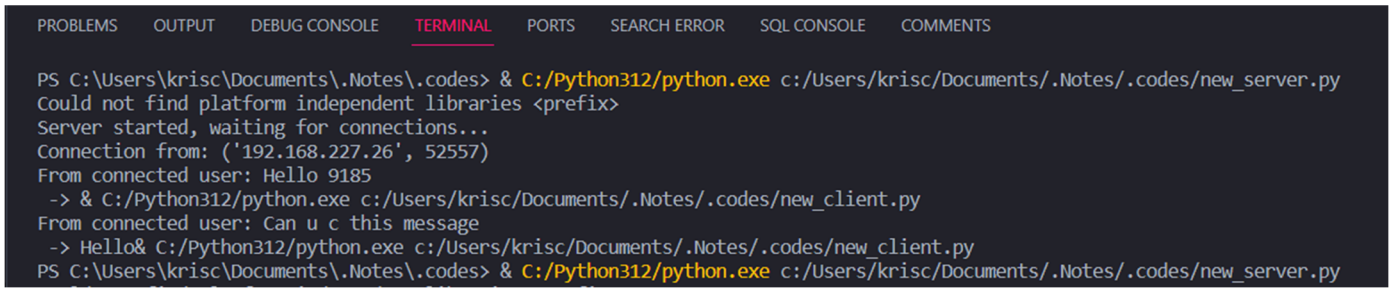
```
universe@lenovo12:~/Desktop/afdz$ cat client.py
import socket


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()
universe@lenovo12:~/Desktop/afdz$
```

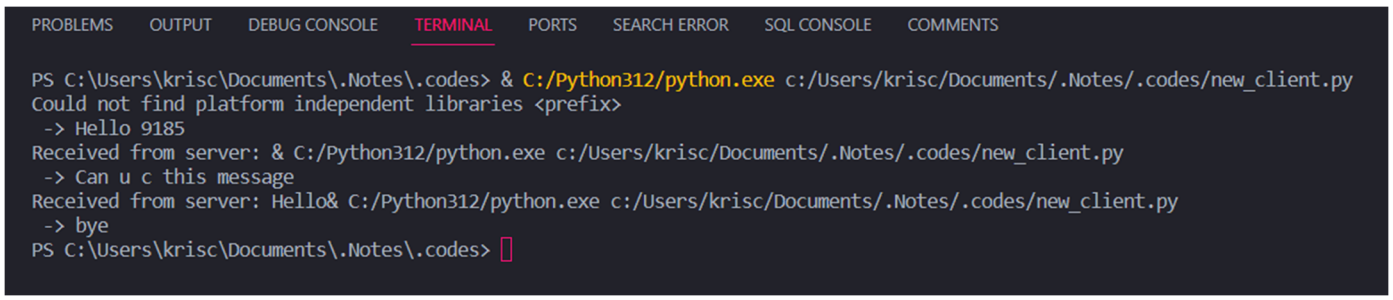
Server.py

```
universe@lenovo12:~/Desktop/afdz$ cat 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()
```

OUTPUT:





Multithreading
2 client 1 server

Client1.py

```
import socket

host = socket.gethostname()
port = 2004
BUFFER_SIZE = 2000
MESSAGE = input("tcpClientA: Enter message/ Enter exit:")

tcpClientA = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcpClientA.connect((host, port))

while MESSAGE != 'exit':
    tcpClientA.send(MESSAGE.encode())
    data = tcpClientA.recv(BUFFER_SIZE).decode()
    print (" Client2 received data:", data)
    MESSAGE = input("tcpClientA: Enter message to continue/ Enter exit:")

tcpClientA.close()
```

Client2.py

```
import socket
host = socket.gethostname()
port = 2004
BUFFER_SIZE = 2000
MESSAGE = input("tcpClientB: Enter message/ Enter exit:")

tcpClientB = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcpClientB.connect((host, port))
```

```
while MESSAGE != ‘exit’:
    tcpClientB.send(MESSAGE.encode())
    data = tcpClientB.recv(BUFFER_SIZE).decode()
    print (“ Client received data:”, data)
    MESSAGE = input(“tcpClientB: Enter message to continue/ Enter exit:”)

tcpClientB.close()
```

Server.py

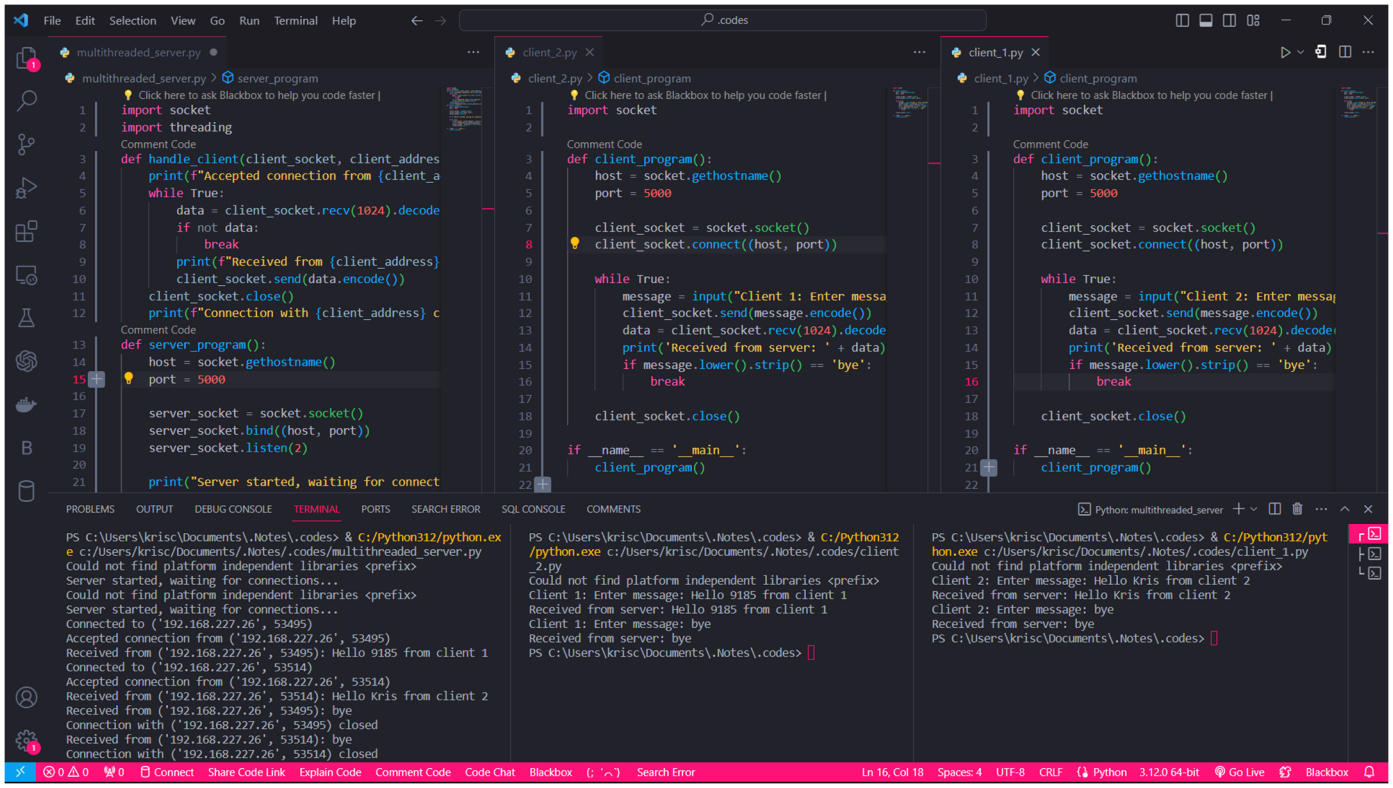
```
import socket
from threading import Thread
from socketserver import ThreadingMixIn
# Multithreaded Python server : TCP Server Socket Thread Pool
class ClientThread(Thread):

    def __init__(self,ip,port):
        Thread._init_(self)
        self.ip = ip
        self.port = port
        print (“[+] New server socket thread started for “ + ip + “:” + str(port))
    def run(self):
        while True :
            data = conn.recv(2048).decode()
            print (“Server received data:”, data)
            MESSAGE = input(“Multithreaded Python server : Enter Response from Server/Enter exit:”)
            if MESSAGE == ‘exit’:
                break
            conn.send(MESSAGE.encode())  # echo

# Multithreaded Python server : TCP Server Socket Program Stub
TCP_IP = ‘0.0.0.0’
TCP_PORT = 2004
BUFFER_SIZE = 20  # Usually 1024, but we need quick response

tcpServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcpServer.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
tcpServer.bind((TCP_IP, TCP_PORT))
threads = []

while True:
    tcpServer.listen(4)
    print (“Multithreaded Python server : Waiting for connections from TCP clients…”)
    (conn, (ip,port)) = tcpServer.accept()
    newthread = ClientThread(ip,port)
    newthread.start()
    threads.append(newthread)

for t in threads:
    t.join()
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  SQL CONSOLE  COMMENTS

PS C:\Users\krisc\Documents\.Notes\.codes> & C:/Python312/python.exe c:/Users/krisc/Documents/.Notes/.codes/multithreaded_server.py
Could not find platform independent libraries <prefix>
Server started, waiting for connections...
Connected to ('192.168.227.26', 53495)
Accepted connection from ('192.168.227.26', 53495)
Received from ('192.168.227.26', 53495): Hello 9185 from client 1
Connected to ('192.168.227.26', 53514)
Accepted connection from ('192.168.227.26', 53514)
Received from ('192.168.227.26', 53514): Hello Kris from client 2
Received from ('192.168.227.26', 53495): bye
Connection with ('192.168.227.26', 53495) closed
Received from ('192.168.227.26', 53514): bye
Connection with ('192.168.227.26', 53514) closed
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  SQL CONSOLE  COMMENTS

PS C:\Users\krisc\Documents\.Notes\.codes> & C:/Python312/python.exe c:/Users/krisc/Documents/.Notes/.codes/client_2.py
Could not find platform independent libraries <prefix>
Client 1: Enter message: Hello 9185 from client 1
Received from server: Hello 9185 from client 1
Client 1: Enter message: bye
Received from server: bye
PS C:\Users\krisc\Documents\.Notes\.codes>
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  SQL CONSOLE  COMMENTS

PS C:\Users\krisc\Documents\.Notes\.codes> & C:/Python312/python.exe c:/Users/krisc/Documents/.Notes/.codes/client_1.py
Could not find platform independent libraries <prefix>
Client 2: Enter message: Hello Kris from client 2
Received from server: Hello Kris from client 2
Client 2: Enter message: bye
Received from server: bye
PS C:\Users\krisc\Documents\.Notes\.codes>
```



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. Advantages of a Multithreaded Server
3. Example of Multithreaded Clients
4. Motivations for Using Threads
5. Typical Models for Organizing Threads