\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Computer Network Lab

CEN-593

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Mini Project

Q. Write a socket program to implement TCP client and Server such that server should be able to send text to client and client check that the received number is composite or prime.

Team Members

20BCS027 - Harsh Raina

20BCS060 - Kovid Sharma

20BCS061 - Kshitiz Kumar

20BCS064 - Mohammed Haziq

20BCS070 - Vicky Gupta (Group Leader)

**S**erver’s Code :-

*import* random

*import* socket

*import* threading

*import* json

print("\033c")

PORT = 4000

*# size of data in bytes that can go in one packets*

HEADER = 1024

FORMAT = "utf-8"

MAX\_CLIENT = 2

DISCONNECT\_MESSAGE = "!DISCONNECTED!"

FIRST\_CONNECTION = "!FIRST\_CONNECTION!"

SERVER = socket.gethostbyname(socket.gethostname())

ADDRESS = (SERVER, PORT)

MAX\_SIZE = 1000001

*# stores the client information like username*

user\_list = {}

*# store the information that number is prime or composite*

*# True means prime and False means composite*

isPrime = [True]\*MAX\_SIZE

*"""*

*Here we made a socket instance and passed it two parameters. The first parameter is AF\_INET and the second one is SOCK\_STREAM. AF\_INET refers to the address-family ipv4. The SOCK\_STREAM means connection-oriented TCP protocol.*

*"""*

*try*:

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

*except* socket.error *as* err:

    print(f"[UNABLE TO CREATE SOCKET] : {err}...\n")

    exit(0)

*"""*

*A server has a bind() method which binds it to a specific IP and port so that it can listen to incoming requests on that IP and port.*

*"""*

*try*:

    server.bind(ADDRESS)

*except* socket.error *as* err:

    print(f"[UNABLE TO BIND TO THE SPECIFIC IP AND PORT] : {err}...\n")

    exit(0)

*# precomputes all prime and composite in the range of 2, MAX\_SIZE*

def sieve\_Of\_Eratosthenes():

    global isPrime

    isPrime[0] = isPrime[1] = False

*for* i *in* range(2, MAX\_SIZE):

        j = i\*i

*while* (j < MAX\_SIZE):

            isPrime[j] = False

            j += i

*# send message to the client*

def sendMessage(msg, client\_connection, client\_address):

*try*:

        client\_connection.send(msg.encode(FORMAT))

*except* socket.error *as* err:

        global user\_list

        print(

            f"[UNABLE TO SEND MESSAGE TO THE {user\_list[client\_address]['name']}] : {err}...\n")

*del* user\_list[client\_address]

*# exit the helper thread created not the main thread*

        exit(0)

*# decode the message if it was the first message or the other message and respond accordingly*

def decodeMessage(str, client\_connection, client\_address):

    client\_object = json.loads(str)

*if* client\_object['msg'] == FIRST\_CONNECTION:

*# for first connection stores the name of the user corresponding to the client address and the number which*

*# is to be send*

        global user\_list, isPrime

        user\_list[client\_address] = {

            "name":  client\_object['name'],

            "number": 2,

        }

*return* f"joined the server."

*else*:

*if* (client\_object['msg'] == 'start'):

*# start the game message so we have to send a random number to the client*

            num = random.randrange(2, MAX\_SIZE)

            user\_list[client\_address]['number'] = int(num)

            print(num)

            num = f"{num}"

            sendMessage(num, client\_connection, client\_address)

*else*:

*# its the respose from the client of the game so we have to check whether its correct or not*

            num = user\_list[client\_address]['number']

*if* (client\_object['msg'] == 'p'):

*if* (isPrime[num] == True):

                    sendMessage("Your answer is correct!",

                                client\_connection, client\_address)

*else*:

                    sendMessage("Your answer is incorrect!",

                                client\_connection, client\_address)

*elif* (client\_object['msg'] == 'c'):

*if* (isPrime[num] == False):

                    sendMessage("Your answer is correct!",

                                client\_connection, client\_address)

*else*:

                    sendMessage("Your answer is incorrect!",

                                client\_connection, client\_address)

*else*:

                sendMessage("Invalid Option!",

                            client\_connection, client\_address)

*return* client\_object['msg']

*# handle's client queries*

def handleClient(client\_connection, client\_address):

    print(f"[NEW CONNECTION] {client\_address} connected.\n")

    global user\_list

    connected = True

*while* connected:

*# reciveing response from client*

*try*:

            str = client\_connection.recv(HEADER).decode(FORMAT)

*except* socket.error *as* err:

            print(

                f"[UNABLE TO RECIVE MESSAGE FROM THE {user\_list[client\_address]['name']}] : {err}...\n")

*del* user\_list[client\_address]

*# exit the helper thread created not the main thread*

            exit(0)

*if* len(str) == 0:

*continue*

        msg = decodeMessage(str, client\_connection, client\_address)

*if* msg == DISCONNECT\_MESSAGE:

*# disconnect the client from the server if message is !DISCONNECTED!*

            connected = False

            print(f"{user\_list[client\_address]['name']} is offline now.")

*continue*

        print(f"{user\_list[client\_address]['name']} : {msg}")

*# removing the client from the list after he/she get disconnected*

*del* user\_list[client\_address]

    client\_connection.close()

def start():

*"""*

*A server has a listen() method which puts the server into listening mode. This allows the server to listen to incoming connections.*

*"""*

    server.listen(MAX\_CLIENT)

    sieve\_Of\_Eratosthenes()

    print(f"[LISTENING]  server is listening on {SERVER}\n")

    connected = True

*while* connected:

*"""*

*And last a server has an accept() and close() method. The accept method initiates a connection with the client and the close method closes the connection with the client.*

*"""*

*try*:

            client\_connection, client\_address = server.accept()

*except* socket.error *as* err:

            print(f"[UNABLE TO CONNECT TO THE CLIENTS] : {err}...\n")

            exit(0)

*try*:

            thread = threading.Thread(target=handleClient, args=(

                client\_connection, client\_address))

            thread.start()

*except* socket.error *as* err:

            print(f"[UNABLE TO CREATE THREAD] : {err}...\n")

            exit(0)

*# -1 bcoz one thread is running the server*

        print(f"[ACTIVE CONNECTIONS] {threading.active\_count()-1}\n")

print("[STARTING] server is starting...\n")

start()

Client’s Code:-

*import* socket

*import* json

print("\033c")

PORT = 4000

*# size of data in bytes that can go in one packets*

HEADER = 1024

FORMAT = "utf-8"

DISCONNECT\_MESSAGE = "!DISCONNECTED!"

*# get the server ip address*

SERVER = socket.gethostbyname(socket.gethostname())

ADDRESS = (SERVER, PORT)

*"""*

*Here we made a socket instance and passed it two parameters. The first parameter is AF\_INET and the second one is SOCK\_STREAM. AF\_INET refers to the address-family ipv4. The SOCK\_STREAM means connection-oriented TCP protocol.*

*"""*

*try*:

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

*except* socket.error *as* err:

    print(f"[UNABLE TO CREATE SOCKET] : {err}...\n")

    exit(0)

*try*:

*# coneecting the client to the server*

    client.connect(ADDRESS)

*except* socket.error *as* err:

    print(f"[UNABLE TO CONNECT TO THE SERVER] : {err}...\n")

    exit(0)

*# function to send message to the server*

def sendMessage(msg):

    json\_object = {'msg': msg}

    msg = json.dumps(json\_object)

*try*:

        client.send(msg.encode(FORMAT))

*except* socket.error *as* err:

        print(f"[UNABLE TO SEND MESSAGE TO THE SERVER] : {err}...\n")

        exit(0)

def reciveMessage():

*try*:

        server\_msg = client.recv(HEADER).decode('utf8')

*except* socket.error *as* err:

        print(f"[UNABLE TO RECIEVE MESSAGE FROM THE SERVER] : {err}...\n")

        exit(0)

    print(f"Server : {server\_msg}")

*return* server\_msg

*"""*

*This is the first message sent to the server from the client side to know that another client is connected*

*so we have to store the client information and don't ask the info again and again from the client side*

*"""*

user\_name = input("Enter your name : ")

json\_object = {'name': user\_name, 'msg': '!FIRST\_CONNECTION!'}

msg = json.dumps(json\_object)

client.send(msg.encode(FORMAT))

connected = True

*while* connected:

    print("\033c")

    print("\_\_\_\_\_\_\_[GAME\_SERVER]\_\_\_\_\_\_\_\_\n")

    msg = input("Play a game [y/n]: ")

*if* (msg != 'y'):

*if* msg != 'n':

            print("Invalid Option!")

*# user don't want to play the game*

        msg = DISCONNECT\_MESSAGE

        connected = False

        sendMessage(msg)

*continue*

*# client response to the server that user wants to play the game*

    sendMessage('start')

*# server response to the client with a question*

    server\_msg = reciveMessage()

*# client response to the server to answer the question*

    msg = input("Prime or Composite [p/c] : ")

    sendMessage(msg)

*# server response to the client wheter the answer is correct or not*

    server\_msg = reciveMessage()

    input("Press Enter to continue...")

*# closing the connection fromt the server*

print("Connection Closed!")

client.close()

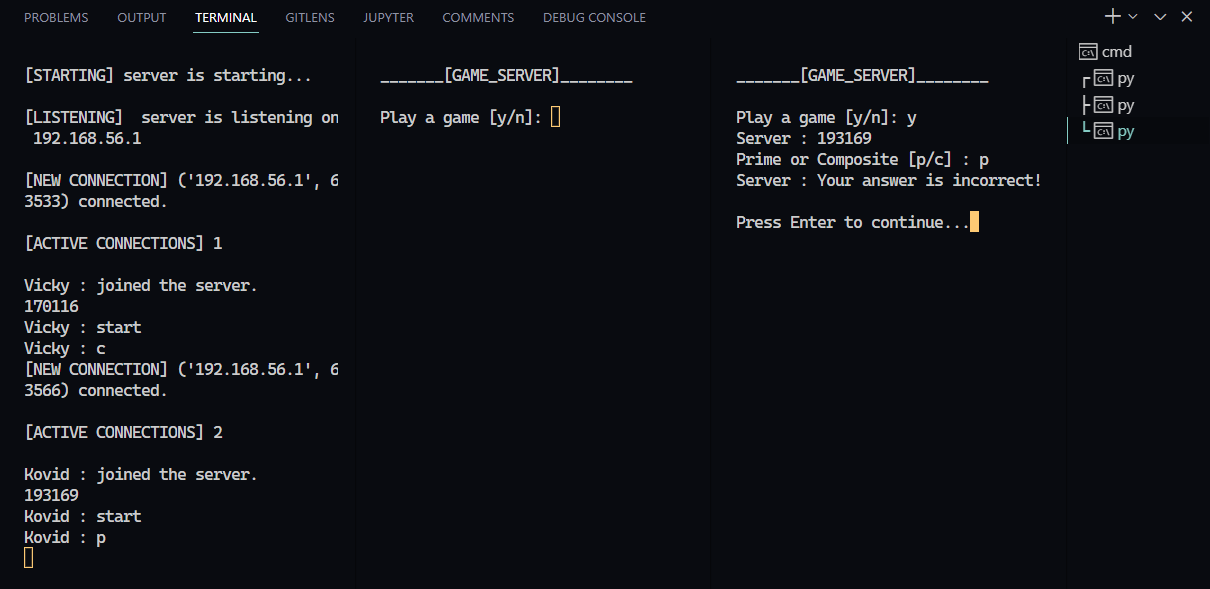
Output :-

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence



Graphical user interface, text

Description automatically generated

Text

Description automatically generated