**22BCE0476**

**AMAN CHAUHAN**

****

BCSE308P - Computer Networks Lab

LAB REPORT

DA-4 Socket Programming

**3. Socket**

1. UDP socket

2. TCP socket

3. HTTP TCP

4. Echo TCP

5. ARP TCP

6. RARP TCP

7. Chat TCP

8. File Transfer TCP

9. DNS TCP

10. Multiuser chat TCP

11. Math server TCP

1. UDP socket programs:

Server:

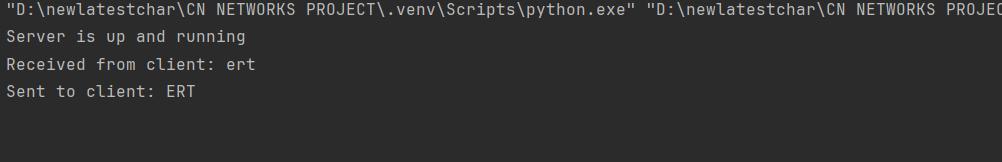
import socket  
  
def udp\_server():  
 # Create a UDP socket and bind it to port 9876  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 server\_socket.bind(('localhost', 9876))  
 print("Server is up and running")  
  
 while True:  
 # Receive data from the client  
 receive\_data, client\_address = server\_socket.recvfrom(1024)  
 sentence = receive\_data.decode('utf-8').strip()  
 print(f"Received from client: {sentence}")  
  
 # Process the data (convert to uppercase)  
 capitalized\_sentence = sentence.upper()  
 send\_data = capitalized\_sentence.encode('utf-8')  
  
 # Send the response back to the client  
 server\_socket.sendto(send\_data, client\_address)  
 print(f"Sent to client: {capitalized\_sentence}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 udp\_server()

Client:

import socket  
  
def udp\_client():  
 # Create a UDP socket  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 server\_address = ('localhost', 9876)  
  
 # Get input from the user  
 sentence = input("Enter a string to convert to uppercase: ")  
 send\_data = sentence.encode('utf-8')  
  
 # Send data to the server  
 client\_socket.sendto(send\_data, server\_address)  
  
 # Receive response from the server  
 receive\_data, \_ = client\_socket.recvfrom(1024)  
 capitalized\_sentence = receive\_data.decode('utf-8').strip()  
 print(f"From server: {capitalized\_sentence}")  
  
 # Close the client socket  
 client\_socket.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 udp\_client()

Output:

Server output:



Client output:

A screen shot of a computer

Description automatically generated

2. TCP socket:

Server:

import socket  
import os  
  
def tcp\_server():  
 # Set up the server socket to listen on port 4000  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket.bind(('localhost', 4000))  
 server\_socket.listen(1)  
 print("Server is ready for connection")  
  
 # Accept a connection from a client  
 client\_socket, client\_address = server\_socket.accept()  
 print("Connection established with:", client\_address)  
  
 # Receive the filename from the client  
 file\_name = client\_socket.recv(1024).decode('utf-8').strip()  
 print(f"Client requested file: {file\_name}")  
  
 try:  
 # Check if the file exists and read its contents  
 if os.path.isfile(file\_name):  
 with open(file\_name, 'r') as file:  
 for line in file:  
 client\_socket.sendall(line.encode('utf-8'))  
 print(f"Sent line to client: {line.strip()}")  
 else:  
 client\_socket.sendall(b"File not found.")  
 print("File not found.")  
  
 except Exception as e:  
 print(f"An error occurred: {e}")  
  
 finally:  
 # Close the connections  
 client\_socket.close()  
 server\_socket.close()  
 print("Server closed the connection.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 tcp\_server()

Client:

import socket  
  
def tcp\_client():  
 # Connect to the server running on localhost at port 4000  
 server\_address = ('localhost', 4000)  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client\_socket.connect(server\_address)  
  
 # Get the filename from the user  
 file\_name = input("Enter the filename: ")  
 client\_socket.sendall(file\_name.encode('utf-8'))  
  
 # Receive the file content from the server and display it  
 try:  
 while True:  
 data = client\_socket.recv(1024)  
 if not data:  
 break  
 print(data.decode('utf-8'), end="")  
  
 except Exception as e:  
 print(f"An error occurred: {e}")  
  
 finally:  
 # Close the client socket  
 client\_socket.close()  
 print("\nClient closed the connection.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 tcp\_client()

Output:

Server output:

A computer screen shot of a program code

Description automatically generated

Client output:

A screen shot of a computer

Description automatically generated

3. HTTP TCP:

import requests  
import os  
  
def download\_web\_page(url, file\_path="D:/newlatestchar/CN NETWORKS PROJECT/DownloadedPage.html"):  
 try:  
 # Ensure the directory exists  
 os.makedirs(os.path.dirname(file\_path), exist\_ok=True)  
  
 # Send a GET request to the URL with a user-agent header  
 headers = {  
 "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537.36"  
 }  
 response = requests.get(url, headers=headers, timeout=10)  
  
 # Check if the request was successful (status code 200)  
 if response.status\_code == 200:  
 # Write the page content to a file with the specified encoding  
 with open(file\_path, "w", encoding="utf-8") as file:  
 file.write(response.text)  
 print(f"Successfully downloaded the webpage to {file\_path}")  
 else:  
 print(f"Failed to retrieve webpage. Status code: {response.status\_code}")  
  
 except requests.exceptions.MissingSchema:  
 print("Invalid URL format. Please check the URL.")  
 except requests.exceptions.ConnectionError:  
 print("Failed to establish a connection. Please check the URL or your internet connection.")  
 except requests.exceptions.Timeout:  
 print("The request timed out. Please try again later.")  
 except requests.exceptions.RequestException as e:  
 print(f"An error occurred: {e}")  
 except Exception as e:  
 print(f"An unexpected error occurred: {e}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 url = "https://www.google.com/"  
 download\_web\_page(url)

Output

A screen shot of a computer

Description automatically generated

A screen shot of a computer screen

Description automatically generated

4. Echo TCP:

Server:

import socket  
  
  
class EchoServer:  
 def \_\_init\_\_(self, host='127.0.0.1', port=9999):  
 self.server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.server\_socket.bind((host, port))  
 self.server\_socket.listen(1)  
 print(f"Server started. Listening on port {port}")  
  
 def serve(self):  
 try:  
 while True:  
 client\_socket, client\_address = self.server\_socket.accept()  
 print(f"Client connected from {client\_address}")  
  
 client\_socket.sendall("Welcome to the Python EchoServer. Type 'bye' to close.\n".encode())  
 while True:  
 message = client\_socket.recv(1024).decode()  
 if message.strip().lower() == 'bye':  
 print("Client disconnected.")  
 break  
 print(f"Received from client: {message}")  
 client\_socket.sendall(f"Got: {message}".encode())  
  
 client\_socket.close()  
 except Exception as e:  
 print(f"Error: {e}")  
 finally:  
 self.server\_socket.close()  
 print("Server closed.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 server = EchoServer()  
 server.serve()

Client:

import socket  
  
  
class EchoClient:  
 def \_\_init\_\_(self, host='127.0.0.1', port=9999):  
 self.client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.client\_socket.connect((host, port))  
 print("Connected to the server. Type 'bye' to end the connection.")  
  
 def chat(self):  
 try:  
 # Receive the initial welcome message from the server  
 welcome\_message = self.client\_socket.recv(1024).decode()  
 print("Server:", welcome\_message)  
  
 while True:  
 message = input("You: ")  
 self.client\_socket.sendall(message.encode())  
  
 if message.strip().lower() == 'bye':  
 print("Disconnected from server.")  
 break  
  
 response = self.client\_socket.recv(1024).decode()  
 print("Server:", response)  
 except Exception as e:  
 print(f"Error: {e}")  
 finally:  
 self.client\_socket.close()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 client = EchoClient()  
 client.chat()

Output:

A computer screen shot of a program

Description automatically generated

Server output:

Client output:

A screenshot of a computer

Description automatically generated

5. ARP TCP:

Server:

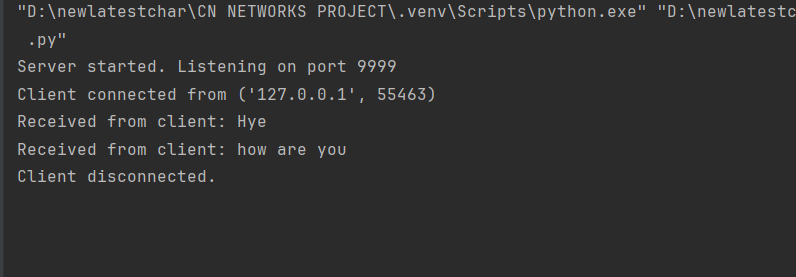
import socket  
  
class ARPServer:  
 def \_\_init\_\_(self, host='127.0.0.1', port=5604):  
 self.server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.server\_socket.bind((host, port))  
 self.server\_socket.listen(1)  
 print(f"ARP Server started and listening on port {port}")  
  
 def serve(self):  
 # Predefined IP to MAC address mapping  
 ip\_to\_mac = {  
 "165.165.80.80": "6A:08:AA:C2",  
 "165.165.79.1": "8A:BC:E3:FA"  
 }  
  
 try:  
 while True:  
 client\_socket, client\_address = self.server\_socket.accept()  
 print(f"Client connected from {client\_address}")  
  
 # Receive IP address from client  
 ip\_address = client\_socket.recv(1024).decode().strip()  
 print(f"Received IP Address: {ip\_address}")  
  
 # Send corresponding MAC address if found  
 mac\_address = ip\_to\_mac.get(ip\_address, "MAC address not found")  
 client\_socket.sendall(mac\_address.encode())  
 print(f"Sent MAC Address: {mac\_address}")  
  
 client\_socket.close()  
 print("Client disconnected.")  
 except Exception as e:  
 print(f"Error: {e}")  
 finally:  
 self.server\_socket.close()  
 print("Server closed.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 server = ARPServer()  
 server.serve()

Client:

import socket  
  
  
class ARPClient:  
 def \_\_init\_\_(self, host='127.0.0.1', port=5604):  
 self.client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.client\_socket.connect((host, port))  
 print("Connected to ARP server.")  
  
 def request\_mac\_address(self, ip\_address):  
 try:  
 # Send IP address to the server  
 self.client\_socket.sendall(ip\_address.encode())  
  
 # Receive MAC address from the server  
 mac\_address = self.client\_socket.recv(1024).decode().strip()  
 print(f"The Physical Address (MAC) for {ip\_address} is: {mac\_address}")  
 except Exception as e:  
 print(f"Error: {e}")  
 finally:  
 self.client\_socket.close()  
 print("Disconnected from server.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 client = ARPClient()  
 ip\_address = input("Enter the Logical address (IP): ")  
 client.request\_mac\_address(ip\_address)

Output:

Server output:



Client output:

A screen shot of a computer

Description automatically generated

6. RARP TCP:

Server:

import socket  
  
def main():  
 try:  
 # Create a UDP socket  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 server\_socket.bind(('localhost', 1309))  
 print("Server is listening...")  
  
 # Define the MAC to IP mapping  
 mac\_to\_ip = {  
 "6A:08:AA:C2": "165.165.80.80",  
 "8A:BC:E3:FA": "165.165.79.1"  
 }  
  
 while True:  
 # Receive the MAC address from client  
 data, client\_address = server\_socket.recvfrom(1024)  
 mac\_address = data.decode().strip()  
 print(f"Received MAC address: {mac\_address}")  
  
 # Find the corresponding IP address  
 ip\_address = mac\_to\_ip.get(mac\_address, "Not found")  
 print(f"Sending IP address: {ip\_address}")  
  
 # Send the IP address back to client  
 server\_socket.sendto(ip\_address.encode(), client\_address)  
 except Exception as e:  
 print(f"Server Error: {e}")  
 finally:  
 server\_socket.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Client:

import socket  
  
def main():  
 try:  
 # Create a UDP socket  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
  
 # Server address and port  
 server\_address = ('localhost', 1309)  
  
 # Get MAC address input from the user  
 mac\_address = input("Enter the Physical address (MAC): ").strip()  
 print(f"Sending MAC address: {mac\_address}")  
  
 # Send MAC address to the server  
 client\_socket.sendto(mac\_address.encode(), server\_address)  
  
 # Receive IP address from the server  
 data, \_ = client\_socket.recvfrom(1024)  
 ip\_address = data.decode().strip()  
 print(f"The Logical Address is (IP): {ip\_address}")  
 except Exception as e:  
 print(f"Client Error: {e}")  
 finally:  
 client\_socket.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Output:

Server output:

A computer screen shot of a black screen

Description automatically generated

Client output:

A screen shot of a computer

Description automatically generated

7. Chat TCP:

Server:

import socket  
  
  
def start\_server():  
 # Create a server socket, bind to port 2000, and listen for incoming connections  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket.bind(('0.0.0.0', 2000))  
 server\_socket.listen(1)  
 print("Server is listening on port 2000...")  
  
 # Accept a connection from a client  
 client\_socket, client\_address = server\_socket.accept()  
 print(f"Connected to {client\_address}")  
  
 # Create input and output streams for communication  
 client\_input = client\_socket.makefile('r')  
 client\_output = client\_socket.makefile('w')  
  
 while True:  
 # Read message from client  
 client\_message = client\_input.readline().strip()  
 if client\_message.lower() == "end":  
 client\_output.write("BYE\n")  
 client\_output.flush()  
 break  
 print(f"Client: {client\_message}")  
  
 # Respond to client  
 server\_message = input("Server: ")  
 client\_output.write(server\_message + "\n")  
 client\_output.flush()  
  
 # Close the connections  
 client\_input.close()  
 client\_output.close()  
 client\_socket.close()  
 server\_socket.close()  
  
  
# Start the server  
start\_server()

Client:

import socket  
  
def start\_client():  
 # Create a client socket and connect to the server at localhost on port 2000  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client\_socket.connect(('127.0.0.1', 2000))  
  
 # Create input and output streams for communication  
 server\_input = client\_socket.makefile('r')  
 server\_output = client\_socket.makefile('w')  
  
 while True:  
 # Get message from user to send to server  
 client\_message = input("Client: ")  
 server\_output.write(client\_message + "\n")  
 server\_output.flush()  
  
 # Read response from server  
 server\_message = server\_input.readline().strip()  
 print(f"Server: {server\_message}")  
  
 if server\_message.lower() == "bye":  
 break  
  
 # Close the connection  
 server\_input.close()  
 server\_output.close()  
 client\_socket.close()  
  
# Start the client  
start\_client()

Output:

Server output:

A computer screen shot of a computer

Description automatically generated

Client output:

A screen shot of a computer

Description automatically generated

8. File Transfer TCP:

Server:

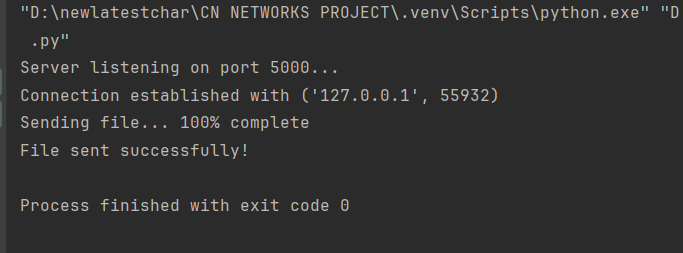
import socket  
import os  
  
def start\_server():  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket.bind(('localhost', 5000))  
 server\_socket.listen(1)  
 print("Server listening on port 5000...")  
  
 client\_socket, client\_address = server\_socket.accept()  
 print(f"Connection established with {client\_address}")  
  
 file\_path = "hello.txt"  
 file\_size = os.path.getsize(file\_path)  
 with open(file\_path, 'rb') as file:  
 bytes\_sent = 0  
 while bytes\_sent < file\_size:  
 # Read in chunks of 10000 bytes  
 chunk = file.read(10000)  
 client\_socket.send(chunk)  
 bytes\_sent += len(chunk)  
 print(f"Sending file... {(bytes\_sent \* 100) // file\_size}% complete")  
  
 client\_socket.close()  
 server\_socket.close()  
 print("File sent successfully!")  
  
# Start the server  
start\_server()

Client:

import socket  
  
def start\_client():  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client\_socket.connect(('localhost', 5000))  
  
 file\_path = "received\_Text.txt"  
 with open(file\_path, 'wb') as file:  
 while True:  
 data = client\_socket.recv(10000)  
 if not data:  
 break  
 file.write(data)  
  
 client\_socket.close()  
 print("File saved successfully!")  
  
# Start the client  
start\_client()

Output:

Server output:



Client output:

A screen shot of a computer

Description automatically generated

9. DNS TCP:

Server:

import socket  
  
  
def index\_of(array, str):  
 str = str.strip()  
 for i in range(len(array)):  
 if array[i] == str:  
 return i  
 return -1  
  
  
def start\_server():  
 hosts = ["zoho.com", "gmail.com", "google.com", "facebook.com"]  
 ip = ["172.28.251.59", "172.217.11.5", "172.217.11.14", "31.13.71.36"]  
  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 server\_socket.bind(('localhost', 1362))  
 print("DNS Server is running. Press Ctrl+C to quit.")  
  
 while True:  
 # Receive data from client  
 data, client\_address = server\_socket.recvfrom(1024)  
 request\_host = data.decode('utf-8')  
 print(f"Request for host: {request\_host}")  
  
 # Find the IP for the requested host  
 if index\_of(hosts, request\_host) != -1:  
 response = ip[index\_of(hosts, request\_host)]  
 else:  
 response = "Host Not Found"  
  
 # Send the response back to the client  
 server\_socket.sendto(response.encode('utf-8'), client\_address)  
  
  
# Start the server  
start\_server()

Client:

import socket  
  
def start\_client():  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 server\_address = ('localhost', 1362)  
  
 host\_name = input("Enter the hostname: ")  
 client\_socket.sendto(host\_name.encode('utf-8'), server\_address)  
  
 data, server = client\_socket.recvfrom(1024)  
 print(f"IP Address: {data.decode('utf-8')}")  
  
 client\_socket.close()  
  
# Start the client  
start\_client()

Output:

Server output:

A screen shot of a computer

Description automatically generated

Client output:

A screen shot of a computer

Description automatically generated

10. Multiuser chat TCP:

import socket  
import threading  
  
# List to hold all connected clients  
clients = []  
  
  
# Function to broadcast messages to all clients except the sender  
def broadcast(message, client\_socket):  
 for client in clients:  
 if client != client\_socket:  
 try:  
 client.send(message.encode())  
 except:  
 clients.remove(client)  
  
  
# Function to handle communication with each client  
def handle\_client(client\_socket, client\_address):  
 # Generate default names for users  
 client\_name = f"User{len(clients) + 1}"  
 print(f"{client\_name} has joined the chat.")  
  
 # Add client to the list of clients  
 clients.append(client\_socket)  
  
 # Broadcast the join message to all other clients  
 broadcast(f"{client\_name} has joined the chat!", client\_socket)  
  
 while True:  
 try:  
 message = client\_socket.recv(1024).decode()  
 if message.lower() == "exit":  
 break  
 broadcast(f"{client\_name}: {message}", client\_socket)  
 except:  
 break  
  
 # Remove client from list and close connection  
 clients.remove(client\_socket)  
 client\_socket.close()  
 broadcast(f"{client\_name} has left the chat.", client\_socket)  
 print(f"{client\_name} disconnected.")  
  
  
# Function to start the server  
def start\_server():  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket.bind(('127.0.0.1', 12345))  
 server\_socket.listen(5)  
 print("Server started on port 12345")  
  
 while True:  
 client\_socket, client\_address = server\_socket.accept()  
 print(f"Connection from {client\_address} established.")  
  
 # Handle each client in a new thread  
 client\_thread = threading.Thread(target=handle\_client, args=(client\_socket, client\_address))  
 client\_thread.start()  
  
  
# Start the server  
if \_\_name\_\_ == "\_\_main\_\_":  
 start\_server()

import socket  
import threading  
  
# Function to receive messages from the server  
def receive\_messages(client\_socket):  
 while True:  
 try:  
 message = client\_socket.recv(1024).decode()  
 if message:  
 print(message)  
 except:  
 print("Connection lost.")  
 break  
  
# Function to start the client  
def start\_client(client\_id):  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client\_socket.connect(('127.0.0.1', 12345))  
  
 # Start a thread to listen for incoming messages from the server  
 receive\_thread = threading.Thread(target=receive\_messages, args=(client\_socket,))  
 receive\_thread.daemon = True # Allow thread to exit when the main program exits  
 receive\_thread.start()  
  
 # Set the default name for the client  
 name = f"User{client\_id}"  
 client\_socket.send(name.encode())  
  
 # Start sending messages to the server  
 while True:  
 message = input(f"{name}: ")  
 if message.lower() == "exit":  
 client\_socket.send(message.encode())  
 break  
 client\_socket.send(message.encode())  
  
 client\_socket.close()  
  
# Start the client  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Assume the client\_id is passed as an argument or can be assigned manually  
 client\_id = 1 # For example, this client will be "User1"  
 start\_client(client\_id)

Output:

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

Multi user:

11. Math server TCP:

Server:

import socket  
import threading  
  
  
def handle\_client(client\_socket):  
 while True:  
 try:  
 # Receive equation from client  
 equation = client\_socket.recv(1024).decode()  
 if equation.lower() == "bye":  
 print("Client disconnected.")  
 break  
  
 print(f"Equation received: {equation}")  
  
 # Parse the equation  
 tokens = equation.split()  
 if len(tokens) != 3:  
 result = "Error: Invalid equation format"  
 else:  
 try:  
 operand1 = int(tokens[0])  
 operator = tokens[1]  
 operand2 = int(tokens[2])  
  
 # Perform the operation  
 if operator == "+":  
 result = operand1 + operand2  
 elif operator == "-":  
 result = operand1 - operand2  
 elif operator == "\*":  
 result = operand1 \* operand2  
 elif operator == "/":  
 if operand2 != 0:  
 result = operand1 / operand2  
 else:  
 result = "Error: Division by zero"  
 else:  
 result = "Error: Invalid operator"  
 except ValueError:  
 result = "Error: Invalid operand(s)"  
  
 print(f"Sending result: {result}")  
 client\_socket.send(str(result).encode()) # Ensure result is sent as a string  
  
 except Exception as e:  
 print(f"Error: {e}")  
 break  
  
 client\_socket.close()  
  
  
def start\_server():  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket.bind(('127.0.0.1', 4444))  
 server\_socket.listen(5)  
 print("Server started on port 4444")  
  
 while True:  
 client\_socket, client\_address = server\_socket.accept()  
 print(f"Connection from {client\_address} established.")  
  
 # Handle each client in a new thread  
 client\_thread = threading.Thread(target=handle\_client, args=(client\_socket,))  
 client\_thread.start()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start\_server()

Client:

import socket  
  
  
def start\_client():  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 client\_socket.connect(('127.0.0.1', 4444))  
  
 while True:  
 # Get user input for the equation  
 equation = input("Enter the equation in the form 'operand operator operand' (e.g., 2 + 3): ")  
 if equation.lower() == "bye":  
 client\_socket.send(equation.encode())  
 break  
  
 # Send equation to the server  
 client\_socket.send(equation.encode())  
  
 # Receive the result from the server  
 result = client\_socket.recv(1024).decode()  
 print(f"Answer = {result}")  
  
 client\_socket.close()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start\_client()

Output:

Server output:



Client output:

A screenshot of a computer program

Description automatically generated