Name: Maheshwari Prathamkumar

Class: Btech II

Roll No: U23CS070

Q1.

import socket

import threading

import os

import json

import base64

class Client:

    def \_\_init\_\_(self, host='127.0.0.1', port=5555):

        self.host = host

        self.port = port

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

        self.username = None

        self.running = True

        self.download\_dir = 'downloads'

        # Create download directory if it doesn't exist

        if not os.path.exists(self.download\_dir):

            os.makedirs(self.download\_dir)

    def start(self):

        try:

            # Connect to server

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

            # Set username

            self.username = input("Enter your username: ")

            self.client\_socket.send(self.username.encode('utf-8'))

            # Start threads for receiving messages and user input

            receive\_thread = threading.Thread(target=self.receive\_messages)

            receive\_thread.start()

            self.send\_messages()

        except Exception as e:

            print(f"Error: {e}")

        finally:

            self.client\_socket.close()

    def receive\_messages(self):

        while self.running:

            try:

                message = self.client\_socket.recv(1024).decode('utf-8')

                if not message:

                    print("Connection to server lost")

                    self.running = False

                    break

                # Check if it's a JSON message (file transfer related)

                try:

                    data = json.loads(message)

                    if data.get('type') == 'file\_request':

                        self.handle\_file\_request(data)

                        continue

                    elif data.get('type') == 'file\_data':

                        self.handle\_file\_data(data)

                        continue

                    elif data.get('type') == 'file\_accepted':

                        recipient = data.get('recipient')

                        filename = data.get('filename')

                        print(f"File request accepted by {recipient}, sending file...")

                        self.send\_file(recipient, filename)

                        continue

                except json.JSONDecodeError:

                    pass  # Not a JSON message, treat as regular text

                print(message)

            except Exception as e:

                print(f"Error receiving message: {e}")

                self.running = False

                break

    def send\_messages(self):

        while self.running:

            try:

                message = input()

                if message.lower() == '/quit':

                    self.running = False

                    break

                elif message.startswith('/sendfile') and len(message.split()) >= 3:

                    parts = message.split()

                    recipient = parts[1]

                    filename = ' '.join(parts[2:])

                    if not os.path.exists(filename):

                        print(f"File '{filename}' not found!")

                        continue

                    # Send command to server

                    self.client\_socket.send(message.encode('utf-8'))

                    # Wait for recipient to accept

                    # (This will be handled by receive\_messages thread)

                else:

                    self.client\_socket.send(message.encode('utf-8'))

            except Exception as e:

                print(f"Error sending message: {e}")

                self.running = False

                break

    def handle\_file\_request(self, data):

        sender = data.get('sender')

        filename = data.get('filename')

        print(f"\n{sender} wants to send you the file: {filename}")

        response = input("Accept file? (y/n): ").lower()

        if response == 'y':

            # Send acceptance to server

            accept\_msg = {

                'type': 'file\_accept',

                'recipient': sender,

                'filename': filename

            }

            self.client\_socket.send(json.dumps(accept\_msg).encode('utf-8'))

            print(f"Waiting to receive file from {sender}...")

        else:

            # Send rejection to server

            reject\_msg = {

                'type': 'file\_reject',

                'recipient': sender,

                'filename': filename

            }

            self.client\_socket.send(json.dumps(reject\_msg).encode('utf-8'))

            print(f"Rejected file from {sender}")

    def handle\_file\_data(self, data):

        sender = data.get('sender')

        filename = data.get('filename')

        file\_content\_b64 = data.get('content')

        if not all([sender, filename, file\_content\_b64]):

            print("Error: Invalid file data received")

            return

        try:

            # Convert base64 to bytes and save file

            file\_content = base64.b64decode(file\_content\_b64)

            # Save file to downloads directory with original filename

            save\_path = os.path.join(self.download\_dir, os.path.basename(filename))

            with open(save\_path, 'wb') as f:

                f.write(file\_content)

            print(f"File received and saved to {save\_path}")

        except Exception as e:

            print(f"Error saving received file: {e}")

    def send\_file(self, recipient, filename):

        try:

            # Read file content

            with open(filename, 'rb') as f:

                file\_content = f.read()

            # Convert to base64 for transfer

            file\_content\_b64 = base64.b64encode(file\_content).decode('utf-8')

            # Create file data message

            file\_data = {

                'type': 'file\_data',

                'sender': self.username,

                'recipient': recipient,

                'filename': os.path.basename(filename),

                'content': file\_content\_b64

            }

            # Send file data to server, which will forward to recipient

            self.client\_socket.send(json.dumps(file\_data).encode('utf-8'))

            print(f"File '{filename}' sent to {recipient}")

        except FileNotFoundError:

            print(f"Error: File '{filename}' not found")

        except Exception as e:

            print(f"Error sending file: {e}")

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

    client = Client()

    client.start()

import socket

import threading

import os

import json

import base64

class Server:

    def \_\_init\_\_(self, host='127.0.0.1', port=5555):

        self.host = host

        self.port = port

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

        self.server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

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

        self.clients = {}  # {username: (client\_socket, client\_address)}

    def start(self):

        self.server\_socket.listen(5)

        print(f"Server started on {self.host}:{self.port}")

        try:

            while True:

                client\_socket, client\_address = self.server\_socket.accept()

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

                client\_handler.start()

        except KeyboardInterrupt:

            print("Server shutting down...")

        finally:

            self.server\_socket.close()

    def handle\_client(self, client\_socket, client\_address):

        try:

            # Get username

            username = client\_socket.recv(1024).decode('utf-8')

            if username in self.clients:

                client\_socket.send("Username already exists! Try again.".encode('utf-8'))

                client\_socket.close()

                return

            self.clients[username] = (client\_socket, client\_address)

            print(f"{username} connected from {client\_address}")

            # Notify all clients about new user

            self.broadcast(f"SERVER: {username} has joined the chat!", exclude=username)

            client\_socket.send(f"Welcome {username}! You are now connected.".encode('utf-8'))

            # Send list of connected users

            connected\_users = list(self.clients.keys())

            client\_socket.send(f"SERVER: Connected users: {', '.join(connected\_users)}".encode('utf-8'))

            while True:

                try:

                    data = client\_socket.recv(1024).decode('utf-8')

                    if not data:

                        break

                    # Check if it's a command

                    if data.startswith('/'):

                        self.handle\_command(username, data)

                    # Check if it's a JSON message (file transfer related)

                    elif data.startswith('{') and data.endswith('}'):

                        self.handle\_command(username, data)

                    else:

                        # Regular message to all

                        self.broadcast(f"{username}: {data}", exclude=None)

                except:

                    break

        finally:

            if username in self.clients:

                del self.clients[username]

                self.broadcast(f"SERVER: {username} has left the chat!", exclude=None)

                client\_socket.close()

                print(f"{username} disconnected")

    def handle\_command(self, sender, command):

        try:

            # Check if it's a JSON command

            if command.startswith('{') and command.endswith('}'):

                data = json.loads(command)

                if data.get('type') == 'file\_accept':

                    self.handle\_file\_accept(sender, data)

                    return

                elif data.get('type') == 'file\_reject':

                    self.handle\_file\_reject(sender, data)

                    return

                elif data.get('type') == 'file\_data':

                    self.forward\_file\_data(data)

                    return

        except json.JSONDecodeError:

            pass  # Not a JSON command, process as normal command

        parts = command.split()

        if parts[0] == '/msg' and len(parts) >= 3:

            recipient = parts[1]

            message = ' '.join(parts[2:])

            self.send\_private\_message(sender, recipient, message)

        elif parts[0] == '/sendfile' and len(parts) >= 3:

            recipient = parts[1]

            filename = ' '.join(parts[2:])

            self.initiate\_file\_transfer(sender, recipient, filename)

        elif parts[0] == '/list':

            connected\_users = list(self.clients.keys())

            sender\_socket = self.clients[sender][0]

            sender\_socket.send(f"SERVER: Connected users: {', '.join(connected\_users)}".encode('utf-8'))

    def send\_private\_message(self, sender, recipient, message):

        if recipient not in self.clients:

            sender\_socket = self.clients[sender][0]

            sender\_socket.send(f"SERVER: User {recipient} not found!".encode('utf-8'))

            return

        recipient\_socket = self.clients[recipient][0]

        recipient\_socket.send(f"PM from {sender}: {message}".encode('utf-8'))

        sender\_socket = self.clients[sender][0]

        sender\_socket.send(f"PM to {recipient}: {message}".encode('utf-8'))

    def initiate\_file\_transfer(self, sender, recipient, filename):

        if recipient not in self.clients:

            sender\_socket = self.clients[sender][0]

            sender\_socket.send(f"SERVER: User {recipient} not found!".encode('utf-8'))

            return

        # Notify recipient about file transfer request

        recipient\_socket = self.clients[recipient][0]

        file\_request = {

            'type': 'file\_request',

            'sender': sender,

            'filename': filename

        }

        recipient\_socket.send(json.dumps(file\_request).encode('utf-8'))

        # Notify sender that request has been sent

        sender\_socket = self.clients[sender][0]

        sender\_socket.send(f"SERVER: File transfer request for '{filename}' has been sent to {recipient}".encode('utf-8'))

    def handle\_file\_accept(self, recipient, data):

        sender = data.get('recipient')  # The original sender is now the recipient for the notification

        filename = data.get('filename')

        if sender not in self.clients:

            recipient\_socket = self.clients[recipient][0]

            recipient\_socket.send(f"SERVER: User {sender} not found!".encode('utf-8'))

            return

        # Notify original sender to start sending the file

        sender\_socket = self.clients[sender][0]

        accept\_msg = {

            'type': 'file\_accepted',

            'recipient': recipient,

            'filename': filename

        }

        sender\_socket.send(json.dumps(accept\_msg).encode('utf-8'))

        print(f"File transfer initiated: {sender} -> {recipient}: {filename}")

    def handle\_file\_reject(self, recipient, data):

        sender = data.get('recipient')  # The original sender

        filename = data.get('filename')

        if sender not in self.clients:

            return

        # Notify original sender that the file was rejected

        sender\_socket = self.clients[sender][0]

        sender\_socket.send(f"SERVER: {recipient} rejected your file transfer request for '{filename}'".encode('utf-8'))

    def forward\_file\_data(self, data):

        recipient = data.get('recipient')

        sender = data.get('sender')

        filename = data.get('filename')

        if recipient not in self.clients:

            sender\_socket = self.clients[sender][0]

            sender\_socket.send(f"SERVER: User {recipient} not found!".encode('utf-8'))

            return

        # Forward the file data to the recipient

        recipient\_socket = self.clients[recipient][0]

        recipient\_socket.send(json.dumps(data).encode('utf-8'))

        print(f"File transfer completed: {sender} -> {recipient}: {filename}")

    def broadcast(self, message, exclude=None):

        for username, (client\_socket, \_) in self.clients.items():

            if username != exclude:

                try:

                    client\_socket.send(message.encode('utf-8'))

                except:

                    print(f"Error broadcasting to {username}")

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

    server = Server()

    server.start()