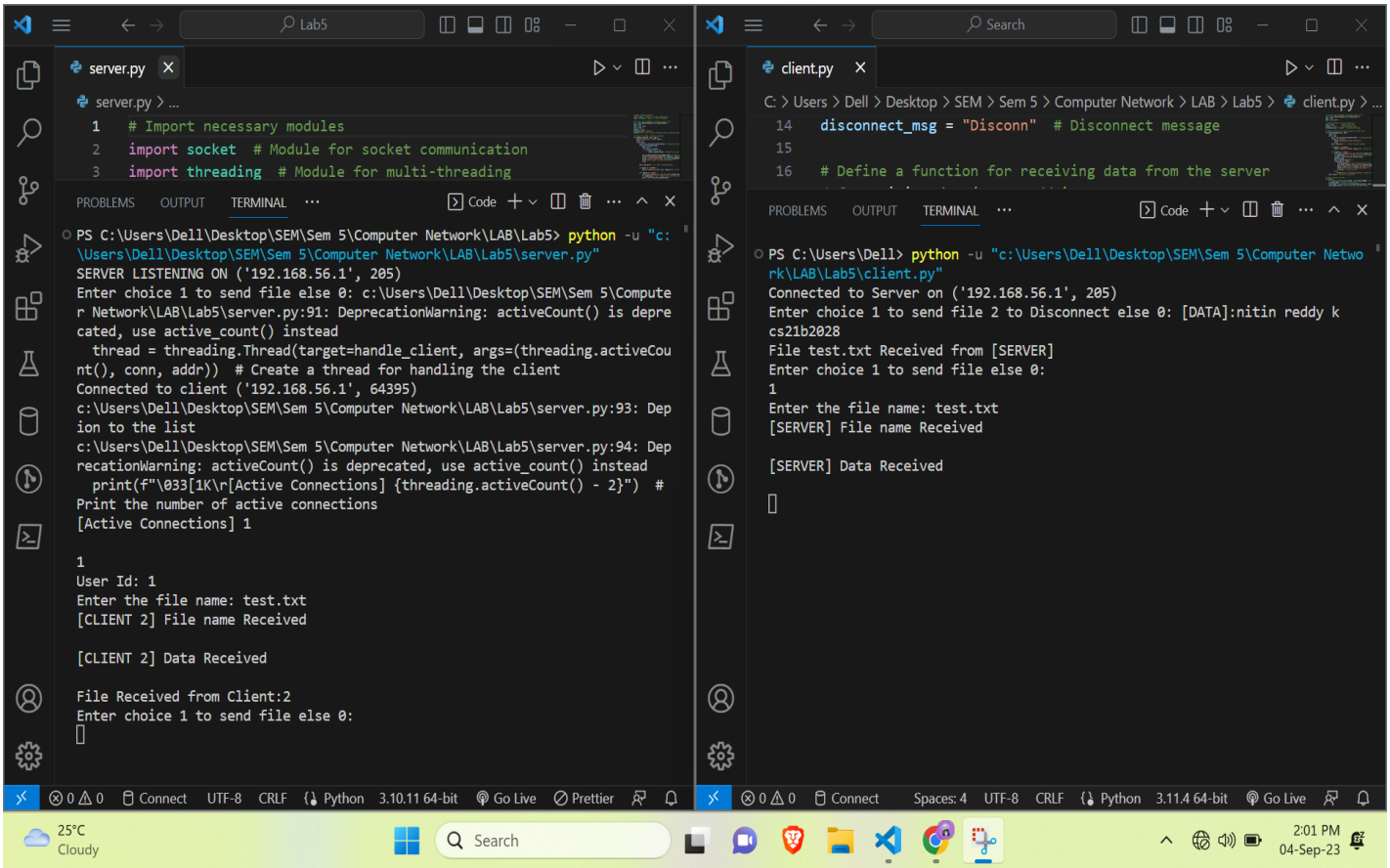
NITIN REDDY K

CS21B2028

LAB 5:

Terminal After Execution:



Client Code:

```python
# Import the necessary modules
import socket        # Module for socket communication
import threading     # Module for multi-threading
import time          # Module for handling time

# Get the local IP address and define a port to use
ip = socket.gethostbyname(socket.gethostname())
port = 205
addr = (ip, port)

# Define constants for data transmission
size = 1024          # Maximum data size
format = 'utf-8'     # Data encoding format
disconnect_msg = "Disconn"  # Disconnect message

# Define a function for receiving data from the server
def receiving_data(conn, addr):
    flag = True
    while flag:
        msg = conn.recv(size).decode(format)  # Receive and decode data
        if msg == 'Disconn':
            print("Disconnected From the Server")
            break
        msg = msg.split(':')  # Split received message

        if msg[0] == "[ACKNOW]":
            print(f"\033[1K\r[SERVER] {msg[1]}\n")  # Print server acknowledgment message

        elif msg[0] == '[FILE]':
            conn.send("[ACKNOW]:File name Received".encode(format))  # Send acknowledgment to server
            f = open(msg[1], 'w')  # Open a file with the received filename
            ed_data = conn.recv(size).decode(format)
            print(ed_data)
            data = ed_data.split(":")
            if data[0] == '[DATA]':
                f.write(data[1])  # Write received data to the file
```

```python
                conn.send("[ACKNOW]:Data Received".encode(format))  # Send acknowledgment to server
                print(f"\033[1K\rFile {msg[1]} Received from [SERVER]")
                print('Enter choice 1 to send file else 0: ')
                f.close()  # Close the file
            else:
                print("Error in receiving file\n")  # Handle error in receiving file
    conn.close()  # Close the connection

# Define the main function
def main():
    c = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # Create a socket object
    c.connect(addr)  # Connect to the server
    recv_thread = threading.Thread(target=receiving_data, args=(c, addr))  # Create a thread for receiving data
    recv_thread.start()  # Start the receiving data thread
    print(f"Connected to Server on {addr}")
    while True:
        k = int(input('Enter choice 1 to send file 2 to Disconnect else 0: '))  # Get user input
        if k == 1:
            file_name = input("Enter the file name: ")  # Prompt for the file name
            try:
                f = open(file_name, 'r')  # Open the file for reading
                ed_file_name = '[FILE]:' + file_name
                c.send(ed_file_name.encode(format))  # Send the file name to the server
                data = f.read()  # Read the file contents
                data = '[DATA]:' + data
                c.send(data.encode(format))  # Send the file data to the server
                f.close()  # Close the file
            except FileNotFoundError:
                print("File Not Found\n")  # Handle file not found error
        elif k == 2:
            c.send("Disconn".encode(format))  # Send a disconnect message to the server
            break  # Exit the loop and close the program

# Check if the script is run as the main program
if __name__ == '__main__':
    main()  # Call the main function
```

Server Code:

server.py

```python
# Import necessary modules
import socket  # Module for socket communication
import threading  # Module for multi-threading

# Get the local IP address and define a port to use
IP = socket.gethostbyname(socket.gethostname())
PORT = 205
ADDR = (IP, PORT)
size = 1024
format = 'utf-8'
disconnect_msg = "Disconn"
clients = []  # List to store information about connected clients

# Function to handle a client's connection
def handle_client(id, conn, addr):
    print(f"Connected to client {addr}")
    flag = True
    while flag:
        msg = conn.recv(size).decode(format)  # Receive and decode data from the client
        if msg == disconnect_msg:
            flag = False
            for client in clients:
                if client['id'] == id:
                    clients.remove(client)  # Remove the client from the list when disconnected

            conn.send(disconnect_msg.encode(format))  # Send disconnect acknowledgment to the client
            print(f"[DISCONNECTED CONNECTION] {addr[0]} {addr[1]} USER {id} is disconnected from the server")
            print(f"[ACTIVE CONNECTIONS] {threading.activeCount() - 2}")  # Count active connections
            print('Enter choice 1 to send file else 0: ')
            break

        msg = msg.split(':')  # Split received message

        if msg[0] == "[ACKNOW]":
            print(f"\033[1K\r[CLIENT {id}] {msg[1]}\n")  # Print client acknowledgment message

        elif msg[0] == "[FILE]":
            conn.send("[ACKNOW]:File name Received".encode(format))  # Send acknowledgment to the client
            f = open(msg[1], '+w')  # Open a file with the received filename for writing
            data = conn.recv(size).decode(format)
            data = data.split(':')
            if data[0] == '[DATA]':
                f.write(data[1])  # Write received data to the file
                conn.send("[ACKNOW]:Data Received".encode(format))  # Send acknowledgment to the client
                print(f"\033[1K\rFile Received from Client:{id}")
                print('Enter choice 1 to send file else 0: ')
            else:
                print("Error in receiving file\n")  # Handle error in receiving file

            f.close()  # Close the file

    conn.close()  # Close the connection when done

# Function to handle sending data to clients
def handle_send_data():
    while True:
        k = int(input('Enter choice 1 to send file else 0: '))  # Get user input
        if k == 1:
            x = int(input("User Id: "))  # Get the user ID
            for client in clients:
                if client['id'] == x:
                    c = client['conn']  # Get the client's connection
                    a = client['addr']  # Get the client's address
                    break
            else:
                print("User does not exist")
                x = -1  # Set x to -1 if the user does not exist
            if x != -1:
                file_name = input("Enter the file name: ")  # Prompt for the file name
                try:
                    f = open(file_name, 'r')  # Open the file for reading
```

server.py

```python
                try:
                    f = open(file_name, 'r')  # Open the file for reading
                    ed_file_name = '[FILE]:' + file_name
                    c.send(ed_file_name.encode(format))  # Send the file name to the client
                    data = f.read()  # Read the file contents
                    ed_data = '[DATA]:' + data
                    c.send(ed_data.encode(format))  # Send the file data to the client
                    f.close()  # Close the file
                except FileNotFoundError:
                    print("File Not Found\n")  # Handle file not found error

# Main function
def main():
    print(f"SERVER LISTENING ON {ADDR}")
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # Create a socket object
    s.bind(ADDR)  # Bind the socket to the server address
    s.listen()  # Listen for incoming connections
    send_thread = threading.Thread(target=handle_send_data)  # Create a thread for sending data
    send_thread.start()  # Start the sending data thread
    while True:
        conn, addr = s.accept()  # Accept a client connection
        thread = threading.Thread(target=handle_client, args=(threading.activeCount(), conn, addr))  # Create a thread for handling the client
        thread.start()  # Start the client handling thread
        clients.append({'id': threading.activeCount() - 2, 'conn': conn, 'addr': addr})  # Add client information to the list
        print(f"\033[1K\r[Active Connections] {threading.activeCount() - 2}")  # Print the number of active connections
        print('')

# Check if the script is run as the main program
if __name__ == "__main__":
    main()  # Call the main function
```