# Department of Computing

**Course Code: CS-432**

**Class: BSCS 12ABC**

**Lab 02:** **Client Server Communication**

**CLO-3: Assess distributed applications utilizing required evaluation measures.**

**Date: February 06, 2025, and February 07, 2025**

# Instructors: Dr. Shah Khalid, Dr. Khurram Shahzad

# Lab 02: Client Server Communication

**Introduction**

This lab focuses on the concepts of client server communication.

**Objectives**

To understand the concepts and implement client server communication.

**Tools/Software Requirement**

MS Visual Studio 2013

**Description**

By using simple network applications, we can send files or messages over the internet. A socket is an object that symbolize a low-level connection point to the IP stack. This socket can be opened or closed or one of a set amount of intermediate states. A socket can deliver and receive data down this connection. Files is generally sent in blocks of a few kilobytes at a time for effectiveness; each of these blocks is named as a packet. Here are the list of well-known port numbers which usually used.

* Port 20, FTP Data
* Port 21, FTP Control
* Port 25, SMTP (email and outgoing)
* Port 53, DNS
* Port 80, HTTP (Web)

In this lab, we will explore the client server programming model in C#, however you are free to choose any language for the lab task.

**Consider Simple Client and SERVER programs in C#:**

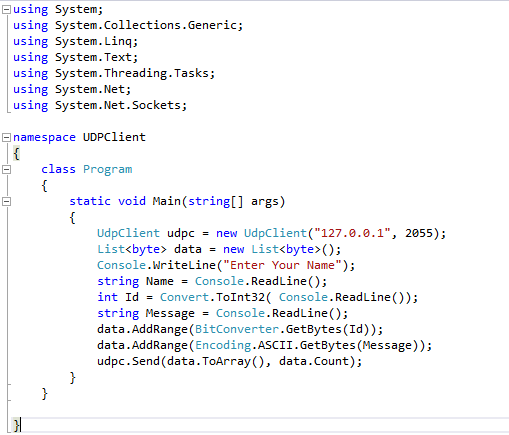


Figure : CLIENT CODE

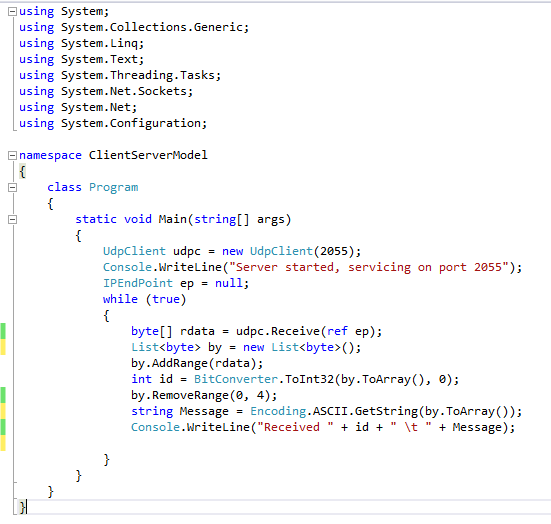


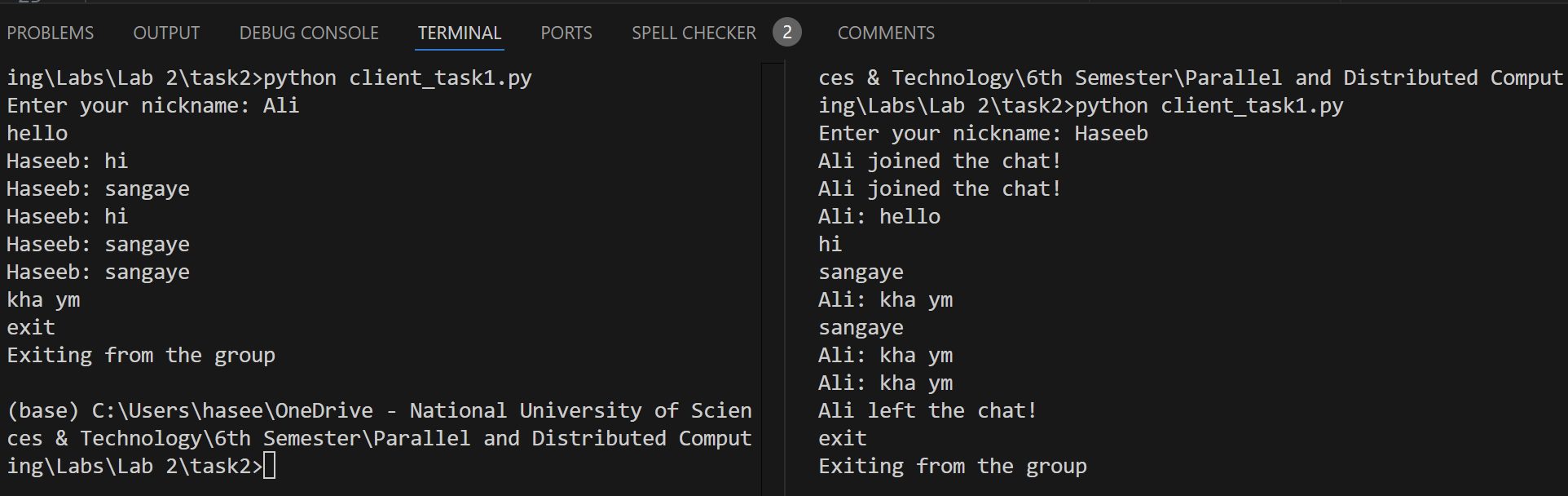
Figure : SERVER CODE

**Lab Tasks:**

1. Your task is to use the above-mentioned program and convert it to group chat using event publish/subscribe architecture as discussed in class. Write a middleware that act as a chat group. Users can join and leave that group. Middleware broadcast the received messages to all the connected users.

**A screen shot of a computer

Description automatically generated**

****

**Client Code:**

import socket

import threading

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

server\_address = ('localhost', 12345)

nickname = input("Enter your nickname: ")

client\_socket.sendto(f"{nickname} joined the chat!".encode(), server\_address)

# Function to continuously receive messages

def receive\_messages():

    while True:

        try:

            data, \_ = client\_socket.recvfrom(4096)

            print(f"{data.decode()}")

        except:

            break

# Start the receiving thread

receive\_thread = threading.Thread(target=receive\_messages, daemon=True)

receive\_thread.start()

# Sending messages

while True:

    message = input()

    if message.lower() == "exit":

        client\_socket.sendto(f"{nickname} left the chat!".encode(), server\_address)

        break

    client\_socket.sendto(f"{nickname}: {message}".encode(), server\_address)

client\_socket.close()

print("Exiting from the group")

**Server Code:**

import socket

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

server\_socket.bind(('localhost', 12345))

print("Server is listening on localhost:12345")

client\_list = set()

while True:

    data, client\_address = server\_socket.recvfrom(4096)

    # Add new clients to the list

    if client\_address not in client\_list:

        client\_list.add(client\_address)

    message = data.decode()

    print(f"Message received from {client\_address}: {message}")

    if message.lower() == "exit":

        client\_list.remove(client\_address)

        continue

    # Send the message to all clients except the sender

    for client in client\_list:

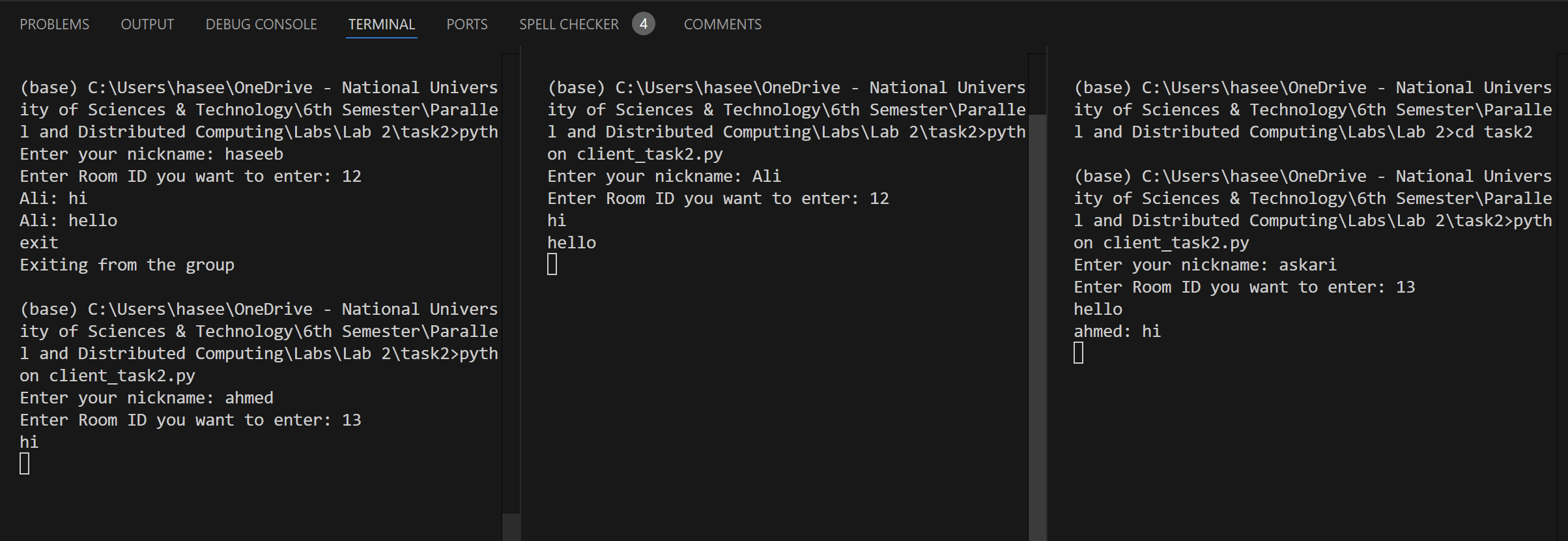
        if client != client\_address:

            server\_socket.sendto(data, client)

1. Add another feature in this middleware which is based on unicast i.e. message send to particular user will not be received by all other users in the group. For this you can assume the user ID of all users is well-known or middleware can provide you that. Moreover, all the communication must be go through the middleware.

A black screen with white text

Description automatically generated



**Client Code:**

import socket

import threading

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

server\_address = ('localhost', 12345)

nickname = input("Enter your nickname: ")

groupchat = input("Enter Room ID you want to enter: ")

# Send join message to the server

client\_socket.sendto(f"{nickname} joined the chat! {groupchat}".encode(), server\_address)

# Function to continuously receive messages

def receive\_messages():

    while True:

        try:

            data, \_ = client\_socket.recvfrom(4096)

            print(data.decode())

        except:

            break

# Start the receiving thread

receive\_thread = threading.Thread(target=receive\_messages, daemon=True)

receive\_thread.start()

# Sending messages

while True:

    message = input()

    if message.lower() == "exit":

        client\_socket.sendto(f"{nickname} left the chat!".encode(), server\_address)

        break

    client\_socket.sendto(f"{nickname}: {message}".encode(), server\_address)

client\_socket.close()

print("Exiting from the group")

**Server Code:**

import socket

import threading

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

server\_socket.bind(('localhost', 12345))

print("Server is listening on localhost:12345")

clients = {}  # Stores {username: (address, room\_id)}

rooms = {}  # Stores {room\_id: set(usernames)}

def handle\_client(data, client\_address):

    global clients, rooms

    message = data.decode()

    if "joined the chat!" in message:

        # Extract user and room info

        username, \_, room\_id = message.split(" ", 2)

        clients[username] = (client\_address, room\_id)

        if room\_id not in rooms:

            rooms[room\_id] = set()

        rooms[room\_id].add(username)

        print(f"{username} joined room {room\_id} from {client\_address}")

    elif "left the chat!" in message:

        username = message.split(" ")[0]

        if username in clients:

            \_, room\_id = clients[username]

            rooms[room\_id].discard(username)

            if not rooms[room\_id]:

                del rooms[room\_id]

            del clients[username]

            print(f"{username} left the chat.")

    else:

        sender\_username = message.split(":")[0]

        \_, room\_id = clients.get(sender\_username, (None, None))

        if sender\_username in clients:

            if message.startswith("@"):

                # Handle Unicast Message

                parts = message.split(" ", 2)

                if len(parts) < 3:

                    return

                target\_user, private\_message = parts[1], parts[2]

                if target\_user in clients:

                    target\_address, \_ = clients[target\_user]

                    server\_socket.sendto(f"[Private] {sender\_username}: {private\_message}".encode(), target\_address)

                else:

                    server\_socket.sendto("User not found.".encode(), client\_address)

            else:

                # Broadcast to the whole room

                for user in rooms.get(room\_id, set()):

                    if user != sender\_username:

                        target\_address, \_ = clients[user]

                        server\_socket.sendto(message.encode(), target\_address)

while True:

    data, client\_address = server\_socket.recvfrom(4096)

    threading.Thread(target=handle\_client, args=(data, client\_address), daemon=True).start()

1. Implement File Transfer in the Middleware  
   1. Extend the middleware to allow users to **send and receive files**.  
   2. The middleware should support **file segmentation**, ensuring that large files are **broken into packets** before being sent.  
   3. Users should be able to **request files from a specific user** using **unicast communication**.

### Deliverables:

1. Compile a single word document by filling in the solution part and submit this Word file on LMS
2. Submit all your code files on LMS.
3. Include screenshots of the program outputs.
4. Submit your Lab Word File and code files separately on submission link.