# 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

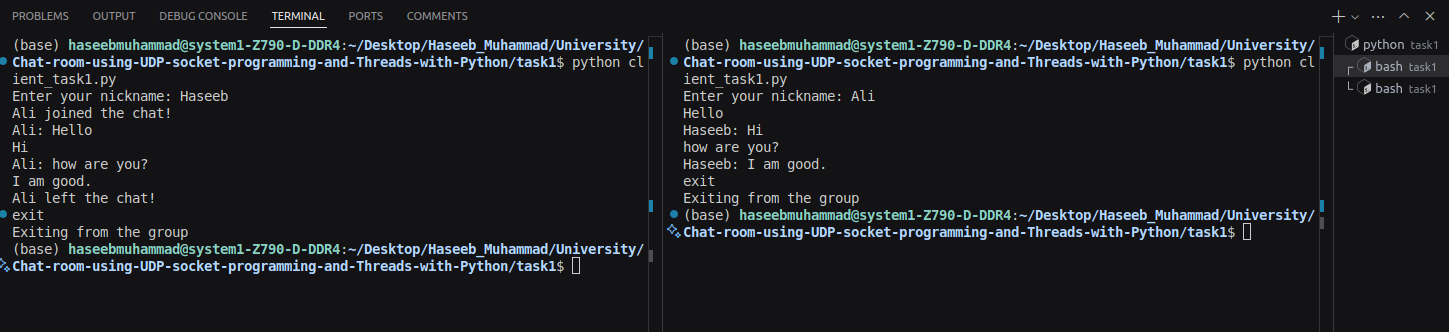
**Lab Tasks:**

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.

**Task analysis :**

I have implemented a simple UDP-based chat application using Python's socket module. The server listens for incoming messages on localhost:12345 and maintains a set of connected clients, broadcasting messages to all clients except the sender. Clients can join the chat by sending a message to the server and receive messages continuously using a separate thread. Each client identifies itself with a nickname and can send messages to the server, which relays them to other clients. A client can exit the chat by sending the "exit" command, which removes them from the active client list. This implementation demonstrates basic networking concepts, such as socket communication, message broadcasting, and multi-threading for concurrent message handling.





**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)

**Client 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

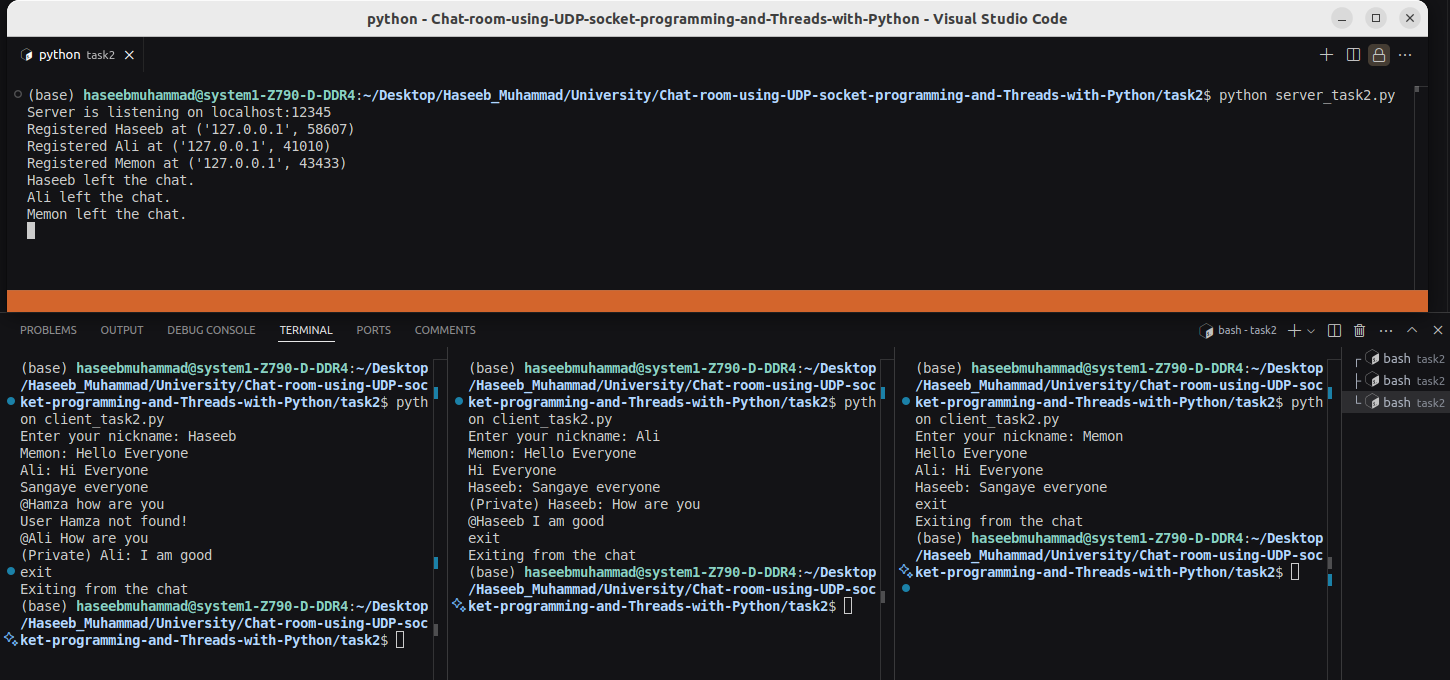
# Send the message to all clients except the sender

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.



**Task analysis :**

In this task the nickname variable acts as user ID. I have implemented a UDP-based chat system that supports both broadcast and unicast (private) messaging. The server listens on localhost:12345 and maintains a dictionary of registered clients, mapping nicknames to their respective addresses. Clients begin by registering their nickname with the server, allowing them to send and receive messages. Messages prefixed with "@" are treated as unicast messages and are directed to a specific user, while other messages are broadcasted to all connected users. The server efficiently handles user registration, message forwarding, and client disconnections. This implementation demonstrates fundamental networking concepts, such as socket communication, message parsing, and client management in a real-time chat environment.

**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"REGISTER:{nickname}".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"EXIT:{nickname}".encode(), server\_address)

break

elif message.startswith("@"): # Unicast message format: @nickname message

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

else: # Broadcast message

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

client\_socket.close()

print("Exiting from the chat")

**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")

clients = {} # Dictionary to store clients as {nickname: address}

while True:

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

message = data.decode()

if message.startswith("REGISTER:"):

nickname = message.split(":")[1]

clients[nickname] = client\_address

print(f"Registered {nickname} at {client\_address}")

elif message.startswith("EXIT:"):

nickname = message.split(":")[1]

if nickname in clients:

del clients[nickname]

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

elif message.startswith("UNICAST:"):

\_, sender, msg = message.split(":", 2)

target\_nickname, actual\_message = msg.split(" ", 1)

target\_nickname = target\_nickname.lstrip("@") # Remove @ symbol

if target\_nickname in clients:

target\_address = clients[target\_nickname]

server\_socket.sendto(f"(Private) {sender}: {actual\_message}".encode(), target\_address)

else:

server\_socket.sendto(f"User {target\_nickname} not found!".encode(), clients[sender])

elif message.startswith("BROADCAST:"):

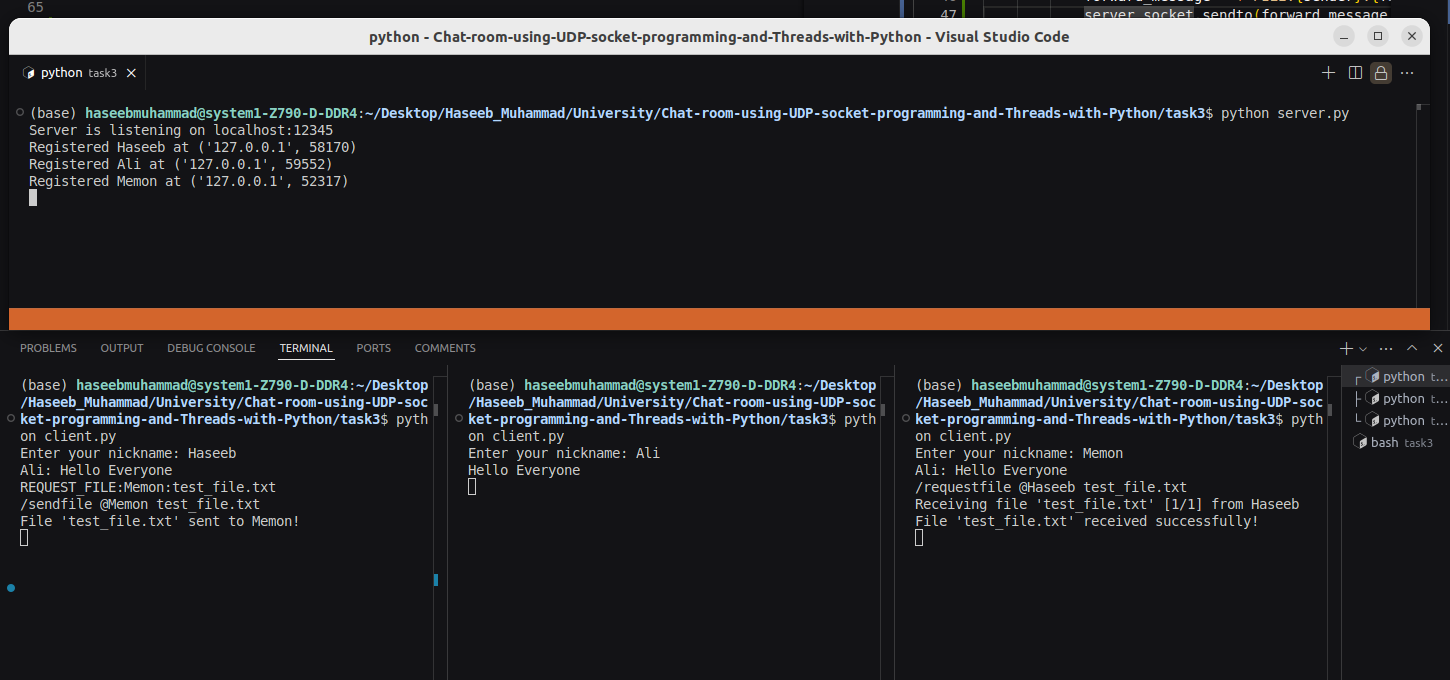
\_, sender, msg = message.split(":", 2)

for nickname, address in clients.items():

if nickname != sender: # Don't send the message back to the sender

server\_socket.sendto(f"{sender}: {msg}".encode(), address)

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**.



**Task Analysis:**

I have developed a UDP-based chat system that supports both text communication and file transfer. The client registers a nickname with the server and can send messages as either broadcasts or private (unicast) messages. Additionally, I have implemented a file transfer mechanism where clients can request and send files in chunks. The client-side script handles message reception, chunked file reception, and file reassembly. It also allows users to send files using the /sendfile command and request files using /requestfile. The server maintains a registry of active clients, forwards messages accordingly, and facilitates file transfers by relaying file chunks and handling requests. This implementation demonstrates essential networking concepts, including message parsing, data segmentation for file transfer, and real-time communication over a UDP socket.

**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)}

file\_buffer = {} # Stores {filename: {sender, data\_chunks, total\_chunks}}

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

global clients, rooms, file\_buffer

message = data.decode()

if message.startswith("JOIN:"):

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

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 message.startswith("EXIT:"):

username = message.split(":")[1]

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.")

elif message.startswith("FILE:"):

\_, sender, filename, total\_chunks, chunk\_index, chunk\_data = message.split(":", 5)

total\_chunks, chunk\_index = *int*(total\_chunks), *int*(chunk\_index)

if filename not in file\_buffer:

file\_buffer[filename] = {"sender": sender, "data\_chunks": {}, "total\_chunks": total\_chunks}

file\_buffer[filename]["data\_chunks"][chunk\_index] = chunk\_data

if len(file\_buffer[filename]["data\_chunks"]) == total\_chunks:

print(f"File {filename} received from {sender}.")

with open(f"received\_{filename}", "wb") as f:

for i in range(total\_chunks):

f.write(file\_buffer[filename]["data\_chunks"][i].encode())

del file\_buffer[filename]

elif message.startswith("REQUEST\_FILE:"):

\_, requester, target\_user, filename = message.split(":")

if target\_user in clients:

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

server\_socket.sendto(f"SEND\_FILE:{requester}:{filename}".encode(), target\_address)

else:

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

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

if sender\_username in clients:

if message.startswith("@"):

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:

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()

**Client Code:**

import socket

import threading

import os

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: ")

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

def receive\_messages():

while True:

try:

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

message = data.decode()

if message.startswith("SEND\_FILE:"):

\_, sender, filename = message.split(":")

print(f"File request received from {sender} for {filename}")

send\_file(sender, filename)

else:

print(message)

except:

break

def send\_file(*target\_user*, *filename*):

if not os.path.exists(filename):

print(f"File {filename} not found.")

return

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

file\_data = f.read()

chunk\_size = 1024

total\_chunks = (len(file\_data) // chunk\_size) + 1

for i in range(total\_chunks):

chunk = file\_data[i \* chunk\_size: (i + 1) \* chunk\_size]

client\_socket.sendto(f"FILE:{nickname}:{filename}:{total\_chunks}:{i}:{chunk.decode()}".encode(), server\_address)

def request\_file(*target\_user*, *filename*):

client\_socket.sendto(f"REQUEST\_FILE:{nickname}:{target\_user}:{filename}".encode(), server\_address)

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

receive\_thread.start()

while True:

message = input()

if message.lower() == "exit":

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

break

elif message.startswith("sendfile"):

\_, target, filename = message.split(" ", 2)

send\_file(target, filename)

elif message.startswith("getfile"):

\_, target, filename = message.split(" ", 2)

request\_file(target, filename)

else:

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

client\_socket.close()

print("Exiting from the group")

### 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.