610 – Advanced Concepts in Operating Systems

Instructor: Dr. Herath Jayantha

Assignment – 3

Chatsystem – 1

Date: 2/22/2026

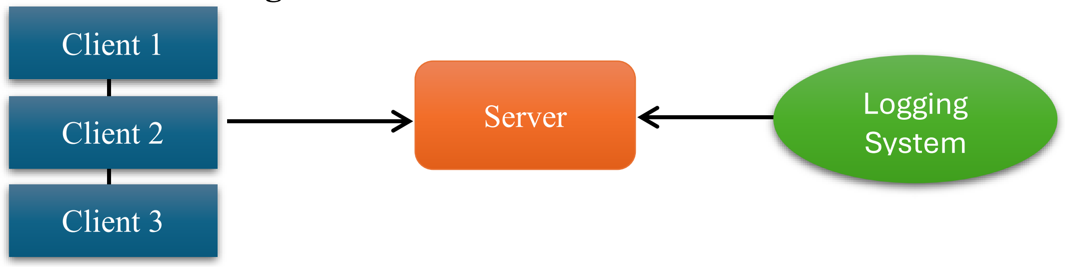
Submitted by Dinesh Seveti

Client–Server Chat System
Using Python Socket Programming

Introduction
This project presents the design and implementation of a client–Server Chat System using Python's socket programming capabilities. The system enables multiple clients to connect to a central server and exchange messages in real time using TCP communication.

The primary objective of this project is to demonstrate:
- TCP Socket Programming
- Client–server architecture
- Concurrency using threading
- Message protocol design
- Robust error handling
- Server-side logging
- Command Line Interface (CLI) interaction

SYSTEM ARCHITECHTURE
The system follows a centralized client–server model:
- A single server listens to incoming client connections.
- Multiple clients connect to the server simultaneously.
- Messages sent by any client are broadcast to all connected clients.
- The server manages each client's connection using a separate thread.

Architecture Diagram


Technology Stack

Component	Technology
Programming Language	Python 3
Networking Protocol	TCP
Socket Library	socket
Concurrency	threading
Logging	logging module
Interface	Command Line Interface

SYSTEM DESIGN

Communication Model

The server uses TCP sockets to:

- Bind to a host and port
- Listening to incoming connections
- Accept multiple clients
- Create separate threads for each client

Clients:

- Establish TCP connection to server
- Send nickname during handshake
- Send and receive messages asynchronously.

Message Protocol Design

A simple text-based protocol is implemented.

Message	Purpose
NICK	Server requests client nickname
<nickname>	Client sends nickname
<nickname>: message	Standard chat message
System messages	Join/leave notifications

All messages:

- UTF-8 encoded
- Broadcast to all clients

IMPLEMENTATION

Server Implementation (server.py)

```
import socket
import threading
import logging

HOST = '127.0.0.1'
PORT = 5555

logging.basicConfig(
    filename='server.log',
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)

clients = []
nicknames = []

def broadcast(message):
    for client in clients:
        try:
            client.send(message)
        except:
            remove_client(client)

def remove_client(client):
    if client in clients:
        index = clients.index(client)
        nickname = nicknames[index]
        clients.remove(client)
        nicknames.remove(nickname)
        client.close()
        broadcast(f"{nickname} left the chat.".encode('utf-8'))
        logging.info(f"{nickname} disconnected")

def handle_client(client):
    while True:
        try:
            message = client.recv(1024)
            broadcast(message)
            logging.info(f"Message: {message.decode('utf-8')}")
        except:
            remove_client(client)
            break

def receive_connections():
    server.listen()
    print(f"Server running on {HOST}:{PORT}")
    logging.info("Server started")

    while True:
        client, address = server.accept()
        print(f"Connected with {address}")

        client.send("NICK".encode('utf-8'))
        nickname = client.recv(1024).decode('utf-8')

        nicknames.append(nickname)
        clients.append(client)

        broadcast(f"{nickname} joined the chat!".encode('utf-8'))
        client.send("Connected to server!".encode('utf-8'))

        logging.info(f"{nickname} connected from {address}")

        thread = threading.Thread(target=handle_client, args=(client,))
        thread.start()

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((HOST, PORT))

receive_connections()
```

Client Implementation (client.py)

```
import socket
import threading

HOST = '127.0.0.1'
PORT = 5555

nickname = input("Choose your nickname: ")

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((HOST, PORT))

def receive_messages():
    while True:
        try:
            message = client.recv(1024).decode('utf-8')
            if message == "NICK":
                client.send(nickname.encode('utf-8'))
            else:
                print(message)
        except:
            print("Connection lost!")
            client.close()
            break

def send_messages():
    while True:
        message = f"{nickname}: {input()}"
        client.send(message.encode('utf-8'))

threading.Thread(target=receive_messages).start()
threading.Thread(target=send_messages).start()
```

Concurrency Implementation
The system uses Python’s threading module. Each client connection runs in its own thread:

```
thread = threading.Thread(target=handle_client, args=(client,))
thread.start()
```

This ensures:
- Multiple clients can operate simultaneously
- Server remains responsive
- No blocking behavior

ERROR HANDLING STRATERGY

Server Side
- Handles abrupt disconnections
- Removing inactive clients
- Prevents server crash
- Logs for all failures

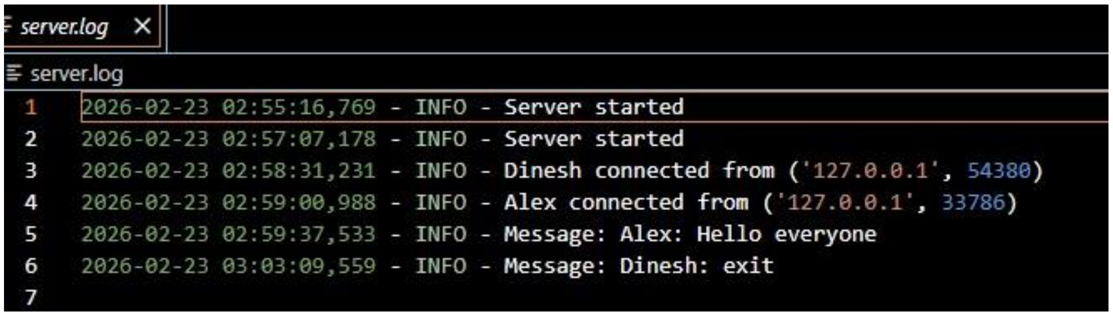
Client Side
- Detects connection loss
- Closes socket gracefully
- Displays error message to user

LOGGING IMPLEMENTATION

Server-side logging tracks:
- Server startup
- Client connections
- Client disconnections
- Message exchanges

TESTING PROCESS

Testing Environment
- OS: Windows 11
- Python Version: 3.11
- Number of Clients Tested: 5



```
2026-02-23 02:55:16,769 - INFO - Server started
2026-02-23 02:57:07,178 - INFO - Server started
2026-02-23 02:58:31,231 - INFO - Dinesh connected from ('127.0.0.1', 54380)
2026-02-23 02:59:00,988 - INFO - Alex connected from ('127.0.0.1', 33786)
2026-02-23 02:59:37,533 - INFO - Message: Alex: Hello everyone
2026-02-23 03:03:09,559 - INFO - Message: Dinesh: exit
```

Test Cases and Results

Test Case	Expected Result	Status
Server startup	Server listens on port 5555	Pass
Single client connection	Client connects successfully	Pass
Multiple clients	Concurrent connections allowed	Pass
Message broadcast	All clients receive message	Pass
Graceful disconnection	Leave message displayed	Pass
Abrupt termination	Server remains stable	Pass
Logging verification	Events recorded in server.log	Pass
Stress test (5 clients)	No crash or message loss	Pass

Edge Cases Tested

- Rapid message sending
- Invalid port connection
- Port conflict
- Server restart
- Client forced shutdown (Ctrl + C)

The system remained stable during all tests.

CONCLUSION

The chat system successfully implements:

- TCP socket communication
- Client–server architecture
- Multithreading for concurrency
- Text-based communication protocol
- Robust error handling
- Server-side logging

Testing confirmed that the system is reliable, stable, and meets all specified project requirements.