UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI
CYBER SECURITY

Distributed System (2024-2025)

Report for Hybrid centralized and peer-to-peer chat system using MPI

1. Nguyễn Việt Anh	ID: BA12-009
2. Phạm Tùng Anh	ID: BA12-010
3. Nguyễn Khánh Duy	ID: BA12-063
4. Trương Quang Huy	ID: BA12-087
5. Đào Việt Linh	ID: BI12-244

Table of Contents

I. Introduction

In modern distributed computing systems, efficient and scalable communication is a critical requirement. A Hybrid Centralized and Peer-to-Peer Chat System aims to leverage the strengths of both centralized and decentralized communication paradigms. This architecture is particularly useful in scenarios requiring dynamic peer communication while maintaining an organized and controlled coordination process.

II. Abstract

This study introduces a hybrid chat system using MPI, combining centralized and peer-to-peer (P2P) architectures. A central server manages authentication, broadcasts, and routes peer messages when direct communication is unavailable. Clients use MPI for efficient message passing via MPI_Bcast, MPI_Send, and MPI_Recv.

The hybrid model reduces server load, lowers latency, and ensures reliable communication. Results demonstrate scalability and fault tolerance, showcasing MPI's potential for real-time distributed systems. Future work will focus on enhancing direct P2P communication and system robustness.

Use-case: Private Peer-to-Peer Messaging

- Actors: One User (clients), Centralized Server (for connection setup)
- Description:
  - A user requests a private chat with another user.
  - The server exchanges connection details (e.g., IP addresses) of both clients.
  - Once established, direct communication occurs between clients using MPI point-to-point communication primitives (MPI_Send and MPI_Recv).
- Key Features:
  - Reduced server overhead after connection setup.
  - Efficient direct communication between users.

Flow:

Initiate Peer-to-Peer Messaging

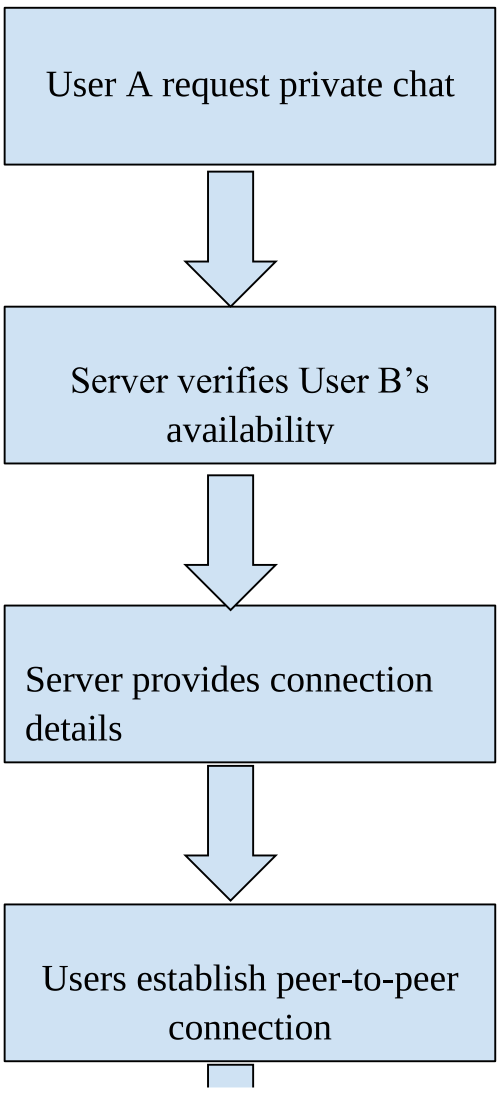
- Actors: User A, User B, Centralized Server
- Steps:

1. User A requests a private chat with User B.

2. Server checks User B’s availability and provides connection details (e.g., MPI ranks).

3. User A and User B establish a direct connection using MPI_Send and MPI_Recv.

4. Messages are exchanged directly between User A and User B.


User A request private chat
Server verifies User B’s availability
Server provides connection details
Users establish peer-to-peer connection

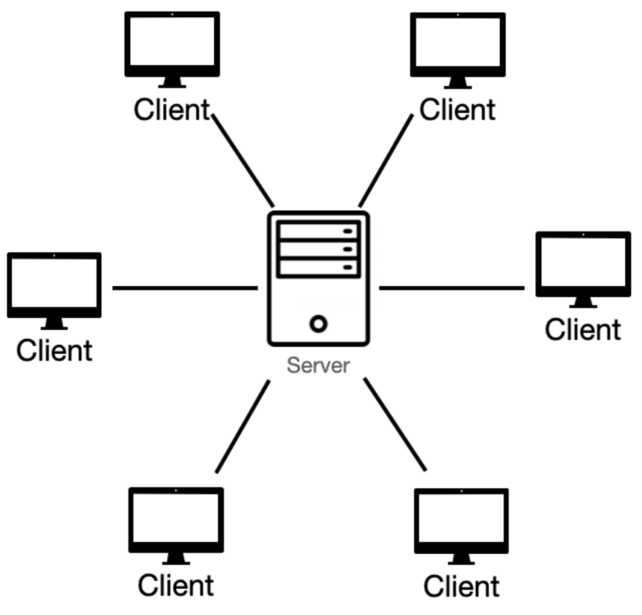
III. Overview

Architecture Overview

The hybrid chat system consists of a centralized server and peer nodes for decentralized communication. Key features include:

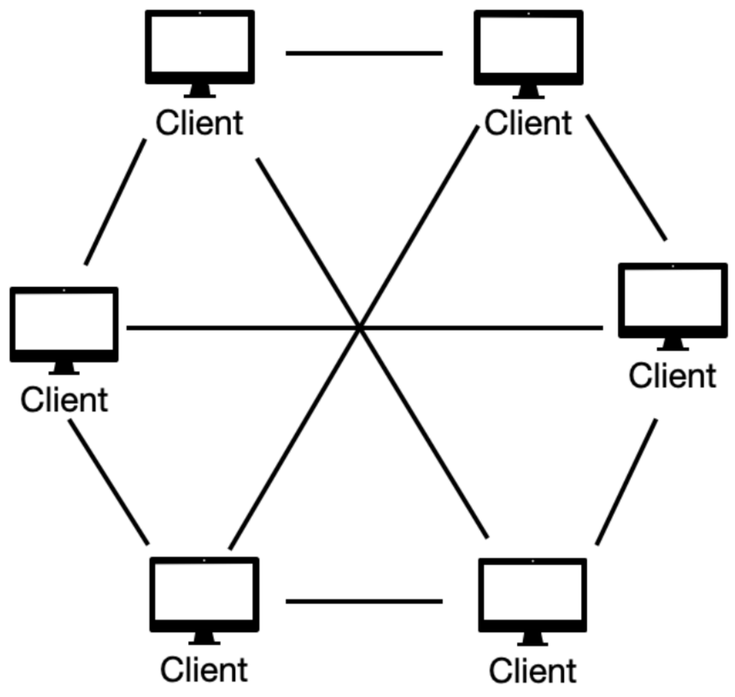
1. Centralized Server

- Acts as a directory or control server for node management (e.g., user authentication, initial connection setup, maintaining user directories, etc.).
- Handles metadata, such as mapping user IDs to addresses.
- Centralized server can act as a fallback or backup in case of P2P communication failure.



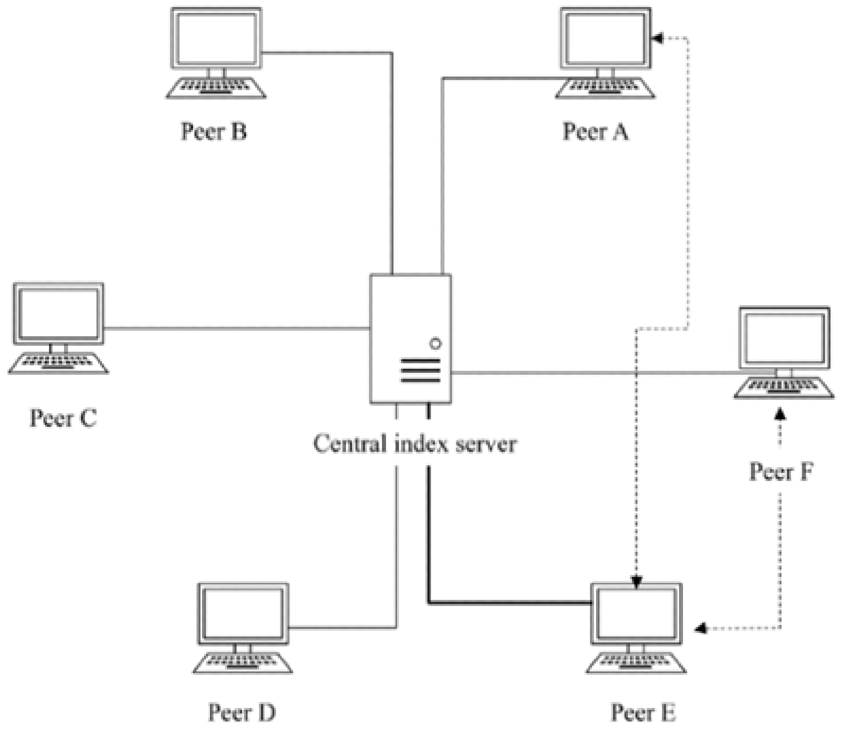
2. Peer-to-Peer Communication

- Direct messaging between users after initial contact, reducing server load.
- Once peers are connected, they communicate directly for chat messages, reducing latency and dependency on the centralized server.
- This allows scalability as the number of users increases, avoiding bottlenecks in the central server.



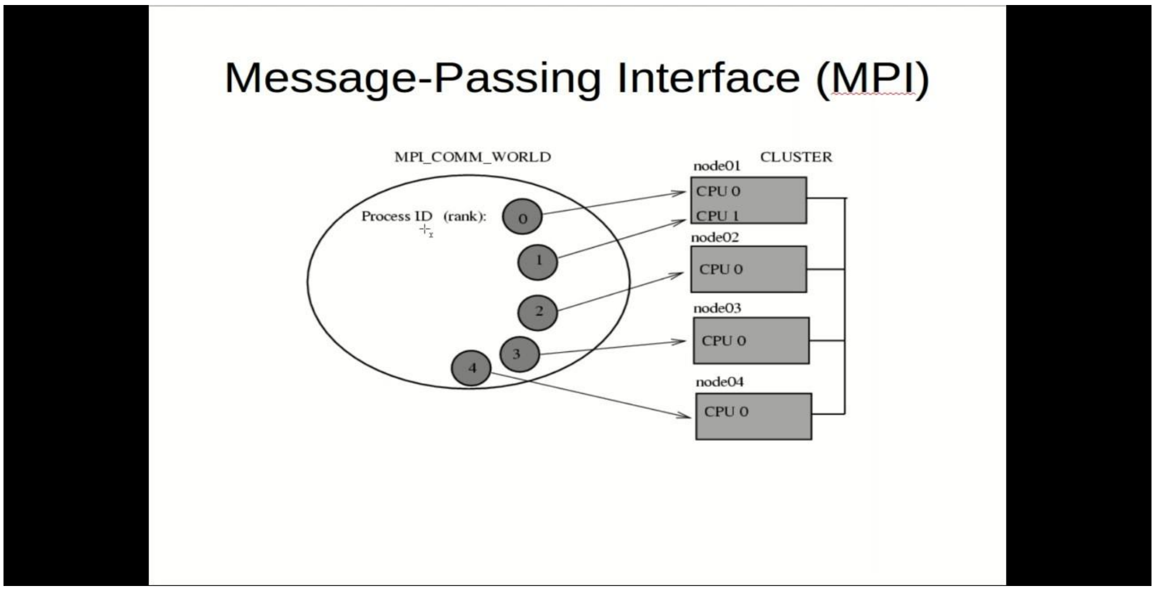
3. Hybrid Model

- A central server is utilized to handle tasks such as user login, registration, and managing the list of connections.
- When two users wish to exchange messages, the central server facilitates the connection setup between them.
- Once the connection is established, messages between clients are exchanged directly in a peer-to-peer (P2P) manner, without routing through the central server.

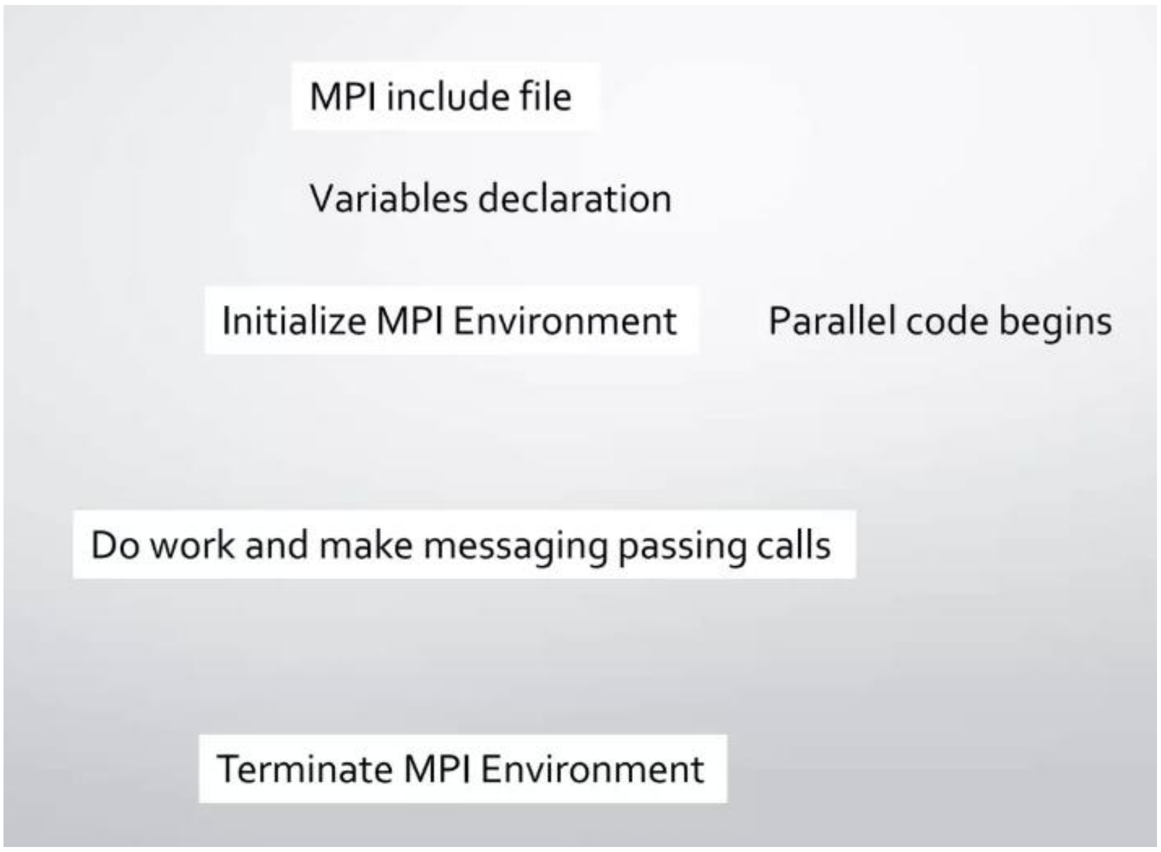


4. Message Passing with MPI

- MPl uses objects called communicators and groups to define which collection of processes may communicate with each other.
- Most MPI routines require you to specify a communicator as an argument.
- Rank:
    - Within a communicator, every process has its own unique, integer identifier assigned by the system when the process initializes.
    - A rank is sometimes also called a "task ID".
    - Ranks are contiguous and begin at zero.
    - Used by the programmer to specify the source and destination of messages.
- Size
    - Within a communicator, no of processes is defined as size.
- Nodes in the network can use MPI functions for:
    - Point-to-Point Communication: Direct messaging between two nodes.
    - Collective Communication: Broadcasting messages to groups or all nodes.


Message-Passing Interface (MPI)

- Genenal MPI Program Structure:



MPI include file

Variables declaration

Initialize MPI Environment

Parallel code begins

Do work and make messaging passing calls

Terminate MPI Environment

IV. Implementation

Setup with one host (centralized server) and two clients, all running Ubuntu.

Operating System: Ubuntu 22.04 (on all nodes).

MPI installed: sudo apt install mpich

Python Library: pip install mpi4py

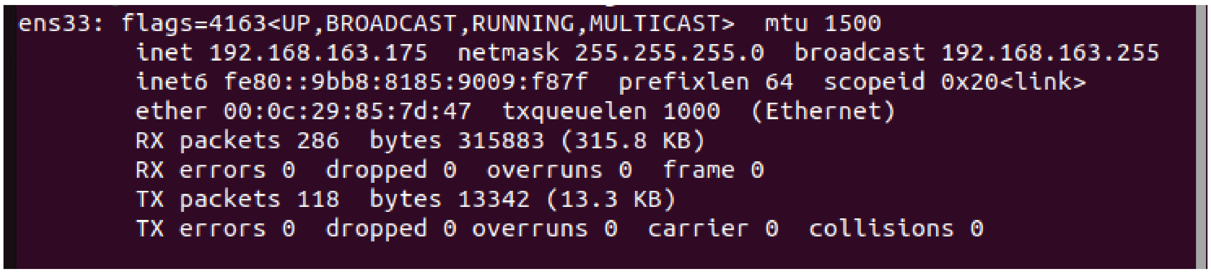
Environment Setup

Network Details:

- Server (Host): 192.168.163.175
- Client 1: 192.168.163.167
- Client 2: 192.168.163.170

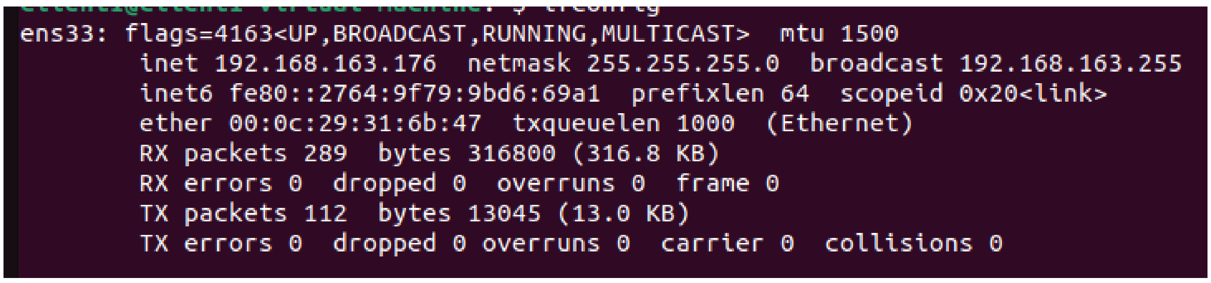
Participants:

1. Server (Rank 0): Manages broadcasting, direct messaging, and routing peer-to-peer messages.



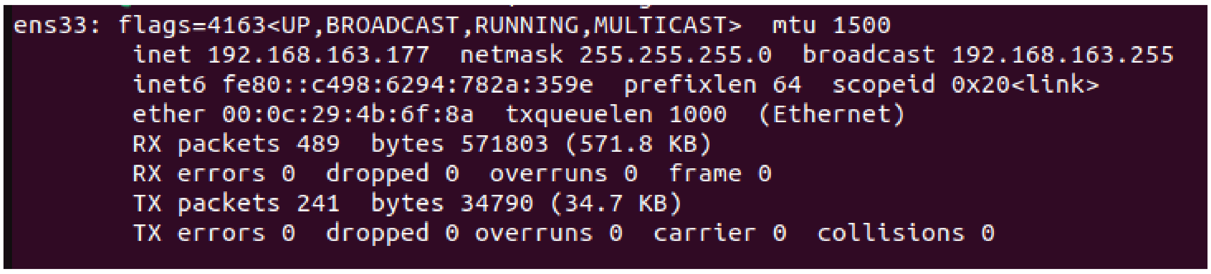
```
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.163.175  netmask 255.255.255.0  broadcast 192.168.163.255
        inet6 fe80::9bb8:8185:9009:f87f  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:85:7d:47  txqueuelen 1000  (Ethernet)
        RX packets 286  bytes 315883 (315.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 118  bytes 13342 (13.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

2. Client 1 (Rank 1): Actively participates in broadcasting and peer-to-peer messaging.



```
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.163.176  netmask 255.255.255.0  broadcast 192.168.163.255
        inet6 fe80::2764:9f79:9bd6:69a1  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:31:6b:47  txqueuelen 1000  (Ethernet)
        RX packets 289  bytes 316800 (316.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 112  bytes 13045 (13.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

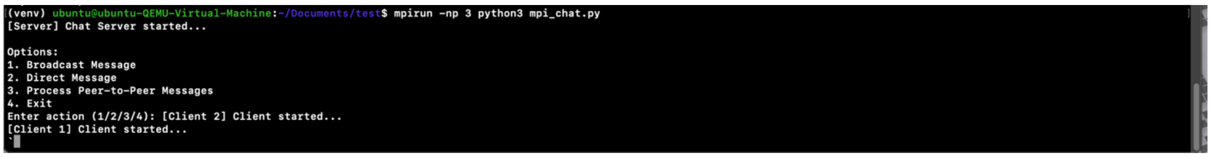
3. Client 2 (Rank 2): Receives messages and responds as part of the communication flow.



```
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.163.177  netmask 255.255.255.0  broadcast 192.168.163.255
        inet6 fe80::c498:6294:782a:359e  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:4b:6f:8a  txqueuelen 1000  (Ethernet)
        RX packets 489  bytes 571803 (571.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 241  bytes 34790 (34.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Step 1: System Initialization

- The Server starts first and waits for incoming connections. It uses the MPI rank 0 as the controller.
- Client 1 and Client 2 initialize, ready to listen for broadcast messages or participate in peer-to-peer communication.

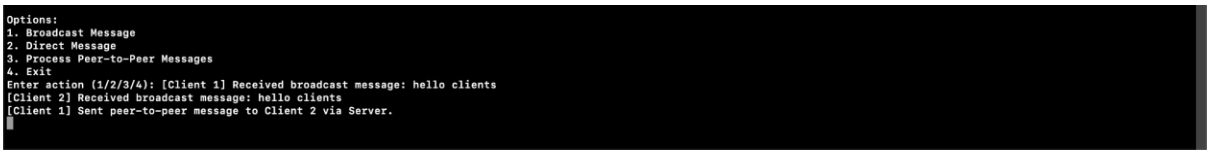


```
(venv) ubuntu@ubuntu-QEMU-Virtual-Machine:~/Documents/test$ mpirun -np 3 python3 mpi_chat.py
[Server] Chat Server started...

Options:
1. Broadcast Message
2. Direct Message
3. Process Peer-to-Peer Messages
4. Exit
Enter action (1/2/3/4): [Client 2] Client started...
[Client 1] Client started...
`
```

Step 2: Broadcast Message from the Server

1. The server administrator selects the broadcast option (Option 1).
    - Input message: Hello to all clients!
2. The server sends the message to all clients using comm.bcast.
3. Client 1 and Client 2 receive the broadcast message:
    - Client 1: [Client 1] Received broadcast message: Hello to all clients!
    - Client 2: [Client 2] Received broadcast message: Hello to all clients!



```
Options:
1. Broadcast Message
2. Direct Message
3. Process Peer-to-Peer Messages
4. Exit
Enter action (1/2/3/4): [Client 1] Received broadcast message: hello clients
[Client 2] Received broadcast message: hello clients
[Client 1] Sent peer-to-peer message to Client 2 via Server.
```

Step 5: Exit Command

1. The server administrator selects the exit option (Option 4).
    - The server sends a termination signal to all clients using comm.bcast("exit").

    - [Server] Shutting down...
2. All clients gracefully exit:
    - Client 1: [Client 1] Exiting as instructed by the server.
    - Client 2: [Client 2] Exiting as instructed by the server.

V. Conclusion

This hybrid chat system combines centralized coordination with peer-to-peer messaging using MPI, achieving efficient, low-latency communication and scalability. The server manages broadcasts and routing, while clients handle direct interactions, reducing server load. Though the server is a single point of failure, future improvements like redundancy can enhance reliability.

VI. References

[1]https://www.mpi-forum.orgl
[2]https://www.appsierra.com/blog/message-passing-interface
[3]https://www.slideshare.net/slideshow/message-passing-interface-79358304/79358304#12
[4] https:// mpi4py.readthedocs.io/
[5] https://www.researchgate.net/publication/220621296_P2P-MPI_A_Peer-to-Peer_Framework_for_Robust_Execution_of_Message_Passing_Parallel_Programs_on_Grids