**CHAT APPLICATION ON LINUX**

**PROJECT REPORT**



**BATCH 2023**

**BAI – 4A**

|  |  |  |
| --- | --- | --- |
| GROUP MEMBER #1: | FASIH HASAN | 23K-0018 |
| GROUP MEMBER #2: | **FAHAD FAHEEM** | **23K-0062** |
| GROUP MEMBER #3: | **ABDUL RAHMAN AZAM** | **23K-0061** |
| GROUP MEMBER #4: | **ARSH AL AMAN** | **23K-0078** |

**Course Instructor: Ms. Saeeda Kanwal**

**Lab Instructor: Khadija tul Kubra**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE**

**NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES – FAST**

**KARACHI CAMPUS**

**Table of Contents**

[Abstract: 3](#_Toc197545713)

[Introduction: 3](#_Toc197545714)

[Problem Statement: 3](#_Toc197545715)

[Project Overview: 3](#_Toc197545716)

[Development Methodology: 4](#_Toc197545717)

[Conclusion: 5](#_Toc197545718)

[Future Plans: 5](#_Toc197545719)

[References 6](#_Toc197545720)

# Abstract:

This project implements a simple **client-server chat application** using **socket programming** in C. The application allows multiple clients to connect to a central server, exchange messages in real-time, and disconnect gracefully. The system demonstrates **interprocess communication (IPC)** using TCP sockets, thread synchronization with mutexes, and dynamic memory management. The project serves as an educational tool to understand networking concepts, concurrent programming, and OS-level socket operations.

# Introduction:

The **Chat Application** is designed to facilitate real-time text communication between multiple clients through a central server. The server manages client connections, broadcasts messages to all connected clients, and handles disconnections. The client-side application allows users to send and receive messages seamlessly.

Key concepts demonstrated:

* **Socket Programming** (TCP-based communication)
* **Multithreading** (Handling multiple clients concurrently)
* **Mutex Locks** (Thread-safe operations on shared resources)
* **Dynamic Memory Management** (Efficient client storage and cleanup)

# Problem Statement:

Traditional single-threaded chat applications cannot handle multiple clients efficiently. A **scalable solution** is needed where:

* Clients can connect/disconnect dynamically.
* Messages are broadcast to all active clients.
* Thread safety is ensured when modifying shared data (client list).

# Project Overview:

The system consists of:

1. **Server**:
   * Listens for incoming connections.
   * Maintains a list of active clients.
   * Broadcasts messages to all clients except the sender.
2. **Client**:
   * Connects to the server.
   * Sends messages to the server.
   * Receives messages from other clients in real-time.

**Interprocess Communication (IPC) & Why Sockets?**

* **Sockets** provide a **standardized IPC mechanism** across different machines.
* **TCP** ensures reliable, ordered, and error-checked message delivery.
* **Alternative IPC methods** (pipes, shared memory) are limited to single-machine communication.

# Development Methodology:

**Server-Side Implementation**

1. **Initialization**:
   * Creates a TCP socket, binds to, and listens for connections.
   * Uses a **dynamic vector** to store connected clients.
2. **Client Handling**:
   * Spawns a **new thread** for each client using pthread\_create().
   * Uses **mutex locks** (pthread\_mutex\_t) to protect the client list.
3. **Message Broadcasting**:
   * Receives messages from a client and broadcasts them to all others.
   * Removes disconnected clients from the list.

**Client-Side Implementation**

1. **Connection Setup**:
   * Connects to the server (#127.0.0.1:8080)
2. **Message Handling**:
   * Uses **two threads**:
     + **Main thread**: Reads user input and sends messages.
     + **Listener thread**: Receives messages from the server.
   * Implements **mutex-protected console output** to avoid race conditions.
3. **Exit Mechanism**:
   * Typing /exit closes the connection gracefully.

**How They Work Together**

* The **server** acts as a relay, forwarding messages between clients.
* The **client** sends messages to the server, which broadcasts them to all other clients.
* **Thread synchronization** ensures no data corruption when multiple clients send messages simultaneously.

# Conclusion:

The project successfully demonstrates:  
✅ **TCP-based client-server communication**  
✅ **Concurrent client handling** (using threads)  
✅ **Thread-safe operations** (mutex locks)  
✅ **Dynamic memory management** (vector for client storage)

Challenges faced:

* **Race conditions** when modifying the client list (solved with mutexes).
* **Memory leaks** (fixed with proper cleanup in socket\_destroy).

# Future Plans:

1. **GUI Integration** (GTK/Qt for a user-friendly interface).
2. **Encryption** (SSL/TLS for secure messaging).
3. **File Transfer Support** (Extending the protocol for binary data).
4. **Database Logging** (Storing chat history).
5. **Multi-Server Support** (Distributed chat servers for scalability).

# References

1. **Beej’s Guide to Network Programming** - <https://beej.us/guide/bgnet/>
2. **POSIX Threads Programming** - <https://computing.llnl.gov/tutorials/pthreads/>
3. **GNU C Library (Socket API)** - <https://www.gnu.org/software/libc/>