Chat Application over Local Area Network through multi-threader server

---

Author:

Zain Ul Abidin

Description:

A project initially developed for an Operating Systems course at FAST Karachi to demonstrate multi-threading and synchronization initiatives in communication servers

Date:

May 2025

Repository:

https://github.com/zain-anwer/NodeTalk_ChatApp.git

Table Of Contents

1. Introduction

The project is a chat application that utilizes socket communication between a multi-threaded server and various client programs. Clients use IP addresses and port numbers passed as command line arguments to connect to a server on a local area network. The server then facilitates the inter-process communication. The application also implements various synchronization techniques along with encryption to provide a robust and secure messaging environment

2. Objectives

1. Implement a multi-threaded server to facilitate multiple client requests
2. Implement multi-processing and multi-threading to keep the Graphical User Interface responsive whilst providing parallel message encryption
3. Resolve producer-consumer problem in the server message queue through binary and counting semaphores

3. Tools & Technologies

1. Languages

    - C language (C++ used only for the GUI thread function)

2. Libraries

    - stdio.h
    - stdlib.h
    - sys/wait.h
    - sys/socket.h
    - unistd.h
    - pthread.h
    - semaphore.h
    - arpa/inet.h
    - SFML/Graphics.h

3. OS Concepts

    - Multi-processing
    - Multi-threading
    - Thread synchronization
    - Pipe and Socket communication

4. System Design

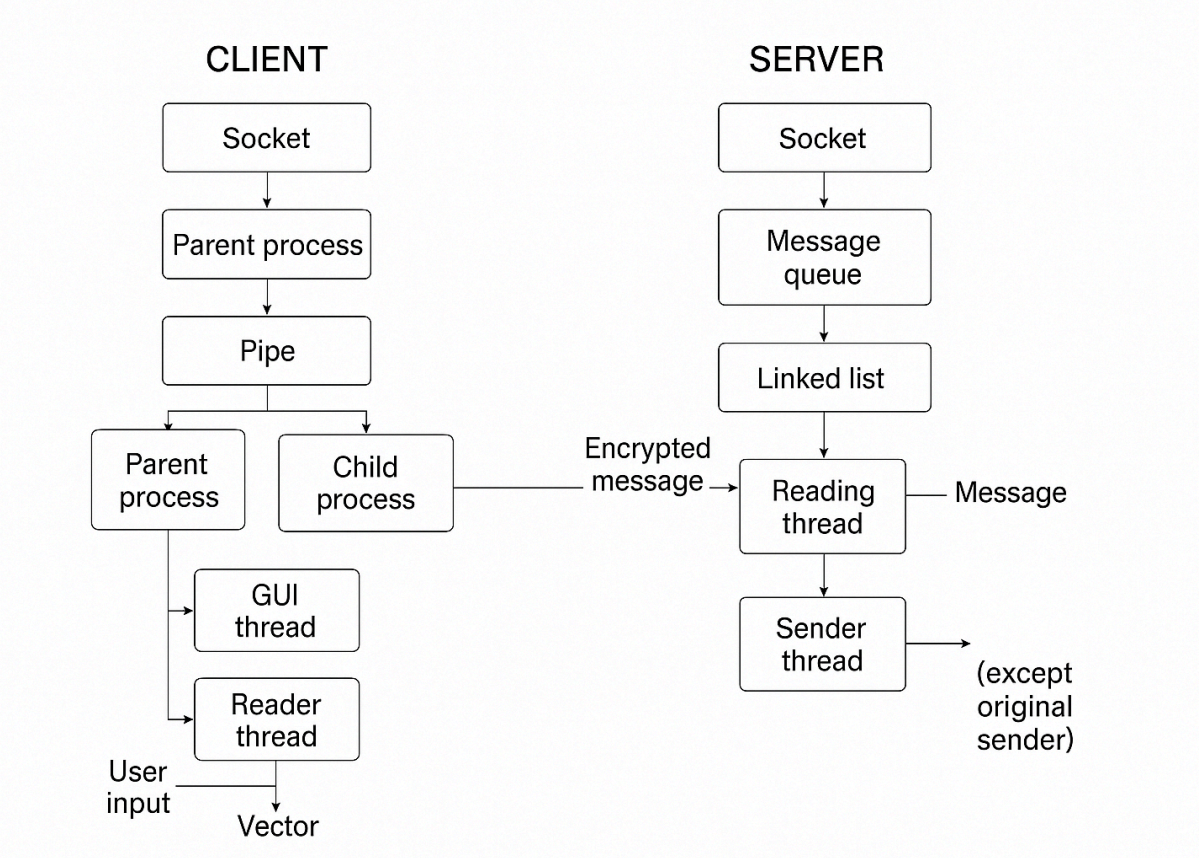
1. Client Program

   The client program begins by initializing a socket file descriptor and socket address to connect to the server port. It then creates a pipe and forks a child process to split responsibilities. The parent process runs two threads: a GUI thread that handles user input and message display, and a reader thread that receives messages from the server, decrypts them, and stores them in a vector for the GUI. When the user types a message, the parent sends it to the child, which encrypts it and writes it to the server through the socket.

2. Server Program

   The server program starts by creating a socket file descriptor and binding it with a socket address to a predefined port. It listens for incoming connections and creates separate reading threads for each client. Incoming messages are placed in a message queue along with the sender's file descriptor. The server also maintains a linked list to track all connected clients. Finally, a sender thread consumes messages from the queue and sends them to all connected clients except the original sender.

5. Architecture Diagram



6. Testing & Evaluation

The program was thoroughly tested by passing the local host IP address to clients, simulating the inter-process communication on a single PC. The messages were sent, received and displayed appropriately without encountering any bugs. The server program as of now continues to run as it is supposed to wait for any further connection

7. Conclusion

The project successfully demonstrates socket-communication through a multi-threaded server. The application of multi-processing and multi-threading in the client program allows parallel execution and a responsive GUI despite the blocking nature of read and write system calls. Lastly, the implementation of synchronization mechanisms and encryption functions make the chat application robust and secure.

8. Future Work

- Make GUI more interactive with scrolling functionality and client validation
- Resolve potential memory leaks and edge cases
- Simplify and generalize program architecture to make application scalable