ELL402
Client-Server Program Report

Assignment 1: Network Programming Using Internet Sockets

Submitted by:
Srijan Singh [2021EE10675]
Keshvi Tomer [2021EE10682]

August 20, 2024

1. Introduction

This report outlines the implementation of a client-server system for managing student marks and providing various functionalities to users based on their roles (students or instructors). The system uses socket programming in C++ to establish a connection between the client and the server, facilitating secure data exchange and operation execution.

2. System Architecture

The system is built on a **client-server architecture** where:

- **Server**: Listens for incoming connections, authenticates users, processes their requests, and sends back the appropriate responses.
- **Client**: Connects to the server via a local IP address (`127.0.0.1`) on port `8080` and communicates by sending requests and receiving responses.

The server can handle multiple clients concurrently using multithreading, enabling simultaneous request processing from different users.

3. Authentication and User Roles

User Authentication is the initial step after establishing a connection:

- **Client Authentication**: Users must provide a valid username and password to access the system. These credentials are verified against a predefined list stored on the server. If the credentials are incorrect, access is denied.

User Roles determine the operations available to the user:

- **Instructor**: The system identifies instructors through their username. Instructors have access to all student data and can perform operations such as viewing all marks, calculating class averages, and updating student marks.
- **Student**: Any other valid username is treated as a student account. Students can view their own marks, calculate their aggregate percentage, and identify their best and worst subjects.

4. Core Functionalities

The system provides different functionalities depending on whether the user is an instructor or a student:

4.1 For Students:

- **View Marks**: Students can view their marks in each subject.
- **Calculate Aggregate Percentage**: The server computes and returns the student's overall percentage.
- **Identify Best and Worst Subjects**: Students can identify their strongest and weakest subjects based on their marks.

4.2 For Instructors:

- **View All Students' Marks**: Instructors can retrieve the marks of all students.
- **Calculate Class Average**: The server calculates and returns the average marks across the class.
- **Identify Failing Students**: The server reports the number of students who failed in each subject.
- **Determine Best and Worst Performing Students**: The server identifies the highest and lowest-performing students based on average marks.
- **Update Student Marks**: Instructors can update the marks for any student in any subject.

5. Data Management and File Handling

The system manages user credentials and student marks through files:

- **User Credentials**: Stored in a file named `user_pass.txt`, which is read by the server for authenticating users.
- **Student Marks**: Stored in `student_marks.txt`, where each student's marks across multiple subjects are recorded. The server reads from and writes to this file as required.

6. Error Handling and Edge Cases

Basic error handling mechanisms are incorporated to manage:

- **Authentication Failures**: If the provided credentials do not match any entry in the `user_pass.txt` file, the server sends an "Authentication Failed" message and disconnects the client.
- **Invalid Operations**: The server checks for the existence of students and subjects before performing updates or calculations and provides appropriate feedback if errors are encountered.

7. Technical Overview

- **Programming Language**: The system is implemented in C++.
- **Libraries Used**: The implementation utilizes standard libraries for socket programming and file handling, including `iostream`, `sstream`, `arpa/inet.h`, `unistd.h`, `netinet/in.h`, and `<thread>` for concurrency.
- **Concurrency Management**: The server employs multithreading to handle multiple clients simultaneously, ensuring efficient processing and responsiveness.

8. Limitations and Future Enhancements

While the system meets the basic requirements, the following areas could be enhanced:

- **Concurrency Improvements**: The server's handling of multiple connections could be optimized for better performance under heavy load.
- **Data Management**: Migrating from text files to a database management system would improve scalability and data integrity, especially as the volume of data increases.

9. Conclusion

The client-server system implemented provides a functional and secure platform for managing student marks and handling various academic operations. The system’s design ensures that both students and instructors can access and manage data effectively. Future improvements, particularly in concurrency management and data storage, would further enhance the system's robustness and scalability.