TW-Mailer – Pro Version Protocol

1.) Client and Server Architecture:

In this project, we implemented a client-server architecture for a messaging system. The client and server communicate over a TCP/IP connection.

Client Responsibilities:

The client component serves as the user interface and interaction point.
The client sends various commands to the server, initiating actions like login, message sending, retrieval, and deletion.
After sending a command, the client awaits and processes the server's response. The responses communicate the outcomes of user actions, such as successful login (OK), error messages (ERR), or requested data.

Server Responsibilities:

The server is the backbone of the system, managing the core functionalities and interactions with external services.
The server receives and interprets user commands coming from the client, parses login requests, processes message-related commands (SEND/LIST/READ/DEL), and manages user interactions.
The server integrates with the FHTW LDAP server for user authentication to enhance security. LDAP verifies user credentials against the centralized directory of FH Technikum Wien, ensuring valid access.
The server also interacts with the file system to manage user-specific message folders. It creates, reads, updates, and deletes messages in response to user commands. The server also ensures persistent storage and retrieval of all messages in a dedicated mail-spool-directory.

2.) Used Technologies and Libraries:

- Programming Language: C/C++
- Networking: Socket programming is employed for communication between the client and server. TCP/IP is used to ensure reliable and ordered data transmission.
- LDAP Integration: LDAP is employed for user authentication, adding an extra layer of security to the system. With the help of LDAP, the server can validate user credentials against the FHTW directory, minimizing the risk of unauthorized access.
- File System Interaction: Standard C++ filesystem library is used to interact with user message folders.

3.) Development Strategy and Protocol Adaptations:

- Our development strategy focuses on creating modular and organized code by assigning distinct responsibilities to the client and the server. This separation of tasks allows us to easily maintain, debug, and improve our project in the future.
  The modular design supports scalability, allowing us to extend functionality or make modifications to specific components without affecting the entire system.
- We implemented a simple text-based approach for the communication protocol between the client and server. The use of predefined strings for commands and data ensures readability and ease of implementation.
  This protocol simplifies debugging, as we can easily trace and understand the exchanged messages between the client and server.
- We implemented a messaging protocol, which ensures secure transmission of login credentials by integrating LDAP for user authentication.
  The protocol is designed to handle errors gracefully. Error messages are communicated effectively between the client and server, providing meaningful feedback to users and administrators.
- The adaptability of the messaging protocol ensures it can handle various user commands such as SENDing, LISTing, READing, and DELeting messages.
  The design allows for flexibility and extensibility, that enables us to introduce new features or commands in the future without significant modifications to the existing code.

4.) Synchronization Methods:

In the project we employed synchronization methods to ensure thread safety in the server. A combination of mutexes and conditional variables is utilized to protect shared resources. The server uses mutexes to control access to critical sections, ensuring that multiple threads do not interfere with each other during LDAP authentication or file system operations.

Additionally, the server handles multiple client connections concurrently using a multi-threaded approach. Each client connection is managed in a separate thread, allowing simultaneous processing of multiple user requests without blocking the main server thread.

5.) Handling of Large Messages:

The system efficiently handles large messages during the send and receive operations. The protocol for message transmission is designed to handle multiline messages. When sending a message, the client can input multiple lines, and the server interprets the message termination, ensuring accurate processing. This design allows users to send and receive messages of varying lengths without encountering limitations.