VERTS
Winter Semester
Computer Science – Bachelor

Project „tw-mailer-basic“

28.11.2023
Petkovic Vladan
Carpentieri Luca

Protocol – TW-Mailer – pro version

1. Advanced Client and Server Architecture

1.1 Client-side Implementation in client-main.cpp and twmailer-client.cpp

- **Program Entry in client-main.cpp**: The main function checks for essential command-line arguments like the IP and the port. It demonstrates error handling for insufficient arguments, promoting robust user interaction.
- **MailClient Class in twmailer-client.cpp**: This class is important in establishing the network layer. It involves:
    - **Socket Initialization**: The constructor MailClient::MailClient initializes the socket and configures the server address using sockaddr_in structure. Error handling is evident in case socket initialization fails.
    - **Data Communication**: The class manages data communication with the server. It uses a buffer BUFFER_SIZE to read and write data, which is essential for handling network messages.

1.2 Server-side Implementation in server-main.cpp and twmailer-server.cpp

- **Server Initialization in server-main.cpp**: The main function of the server sets up essential parameters like port and mail spool directory. It exhibits error handling for command-line argument validation.
- **MailServer Class in twmailer-server.cpp**: The core of the server's functionality lies in the MailServer Class.
    - **Socket and Concurrency Setup**: It initializes server socket(AF_INET, SOCK_STREAM, 0) for listening to client connections. The server employs semaphores, initialized using sem_open, for managing concurrent access, indicating an advanced understanding of process synchronization.
    - **Client Handling**: The server is set up to handle multiple client connections. Each client connection is managed independently, with client authentication handled centrally via interactions with a single LDAP server.

1.3 LDAP Integration in ldap_fh.cpp

- **LDAP Functionality**: This file contains the LDAP handler class Ldap_fh, crucial for user authentication. The class contains methods for connecting to an LDAP server and verifying user credentials. The presence of a semaphore in its constructor suggests synchronization for logging in LDAP operations.

2. Used Technologies and Libraries

2.1 Socket Programming

- **TCP/IP Sockets**: The client and server use AF_INET IPv4 and SOCK_STREAM TCP for socket creation. This choice underlines the need for reliable, connection-oriented communication channels in email services.

2.2 LDAP Authentication

- **LDAP Operations**: The use of OpenLDAP C-API from #include <ldap.h> demonstrates integration with an LDAP server for secure authentication, crucial to validate users for any email server.

3. In-Depth Look at Development Strategy

3.1 Concurrent Handling of Client Requests

- **Semaphore Utilization:** The semaphores in the server's code, such as sem_wait and sem_post, are used to handle concurrency. This is important for synchronizing access to shared resources like log files or user data.

3.2 Custom Email Protocol

- **Protocol Adherence**: The communication between client and server adheres to a custom protocol, involving commands, such as LOGIN, SEND, LIST, READ, and DEL, each requiring request formatting and response handling.

4. Synchronization and Concurrency Control

4.1 Semaphore-Based Synchronization

- **Resource Management**: The server's use of semaphores is effective for controlling access to shared resources in a multi-process environment. This approach prevents race conditions, where two or more processes attempt to modify the same resource simultaneously, ensuring data integrity and consistency in operations like logging.

5. Handling of Large Messages

5.1 Buffer Management and Data Streaming

- **Efficient Data Handling**: The fixed-size buffer (**BUFFER_SIZE**) in both client and server indicates a well-thought-out strategy for data transmission. This buffer size likely represents a balance between memory efficiency and network performance.
- **Streaming for Large Emails**: For emails that exceed the buffer size, the code suggests a segment-based transmission approach, ensuring that large emails are handled effectively without data loss or overflow.