Client and server architecture

Server:

1. Initialization
2. Create a socket - socket()
3. Bind the socket - bind()
4. Listen on the socket - listen()
5. Accept a connection - accept()
6. Send and receive data – send(), recv()
7. Disconnect – shutdown(), close()

Client:

1. Initialization
2. Creat a socket - socket()
3. Connect to the server - connect()
4. Send and receive data – send(), recv()
5. Disconnect – shutdown(), close()

Used technologies

This socket-based client-server application is written in C. The communication type is a SOCK_STREAM: TCP. We used the standard c library it provides utility functions such as perror() for printing error messages. Also, the used socket library (sys/types.h and sys/socket.h) allows the code to create, bind, listen, accept, read, and write data over a network connection.

For the program we implemented the following functions:

For the twmailer-server:

- void *client Communication(void *data, char* mailSpoolDirectory);
  - here we implemented our main menu and integrated the commands SEND, LIST, READ, DEL and QUIT in a switch statement. The implementation of the individual commands is also included in this function.
- void signalHandler(int sig);
  - Within this function the shutdown() function is invoked to initiate a normal TCP close sequence.
- void createDir(char* newDir);
  - The function creates a new directory for a user to store his/her messages (after checking that the directory for the user does not already exist).
- char messeagesFind(char buffer[BUF], char* mailSpoolDirectory);
  - The function opens the directory (char* mailSpoolDirectory) where the messages are stored and searches for the folder (name) that the user has entered in the application. The name of the user folder is stored in char buffer[BUF].
- char* receiveMessage(char buffer[BUF], int *current_socket);
  - This function is called several times in the program to receive messages. It returns the buffer (the user's input) without line breaking characters.

For the twmailer-client:

In the main function the communication with the server is established as well as most of the functionality of the client.

- char userInput(char buffer[BUF])
  - fgets() function with error handling

Development strategy and needed adaption

We used the client/server sample from the Moodle course as a basis for our program. Then we made the relevant settings to receive inputs related to IP and port in the client side and the settings to receive IP and the folder to persist the incoming and outgoing messages and meta-data in the server side, by using argv and argc variables. After establishing communication between the client and the server, various sections were implemented as follows:

- **SEND:**
  - The server receives information from the user step by step and performs the necessary functions according to the received information. First, the name of the sender and then the name of the receiver are received from the client, and accordingly, the corresponding folders with the name of sender and receiver are created in the ,,users'' folder. Then, upon receiving the subject of the letter, a text file with this name is created in the receiver and sender folders. After the message is entered by the sender and end with a ,, . ", this text is stored in the created text file.
- **LIST:**
  - the client sends the username to the server. The server returns the text files in the folder of this user name to the client.
- **READ:**
  - The client sends the username, the server enters the folder related to the username in the users folder. The desired file subject is sent from the client side. The server searches for a text file with this name in this folder. If the file exists, its text will be returned to the client.
- **DEL:**
  - The client sends the username, the server enters the folder related to the username in the users folder. The desired file subject is sent from the client side. The server searches for a text file with this name in this folder. If the file exists, this file will be deleted from this folder.
- **QUIT:**
  - When this code is entered by the client, the connection between the client and the server is interrupted and the socket is closed.

For clarity, a menu has been used which is called again, after each command is executed. Also, functions for summarizing the program are assigned for repetitive parts.