

SPL-1 Project Report

nazeeb_server , a mail server

Submitted by

Nazeeb Ahmed Chowhury

BSSE Roll No. :1432

BSSE Session:2021-22

Submitted to

Kishan Kumar Ganguly,

Associate Professor,

Institute Of Information Technology

Supervisor's Approval:

(signature)



Institute of Information Technology
University of Dhaka

17-12-2023

Table of contents

| | |
|-----------------------------------|----|
| 1.Introduction | 3 |
| 2.Background of the project..... | 4 |
| 3.Description of the Project..... | 6 |
| 4.Implementation and Testing..... | 8 |
| 4.1 myhelper folder..... | 8 |
| 4.2 mysocket folder..... | 9 |
| 4.3 server segments..... | 12 |
| 4.4 SHA-Algo..... | 14 |
| 4.5 myserver.cpp..... | 14 |
| 4.6 myclient.cpp..... | 16 |
| 5.User interface..... | 17 |
| 6.Challenges Faced..... | 18 |
| 7.Conclusion..... | 19 |
| 8.References..... | 20 |

INTRODUCTION

In the dynamic landscape of communication technologies, email remains a cornerstone for professional and personal correspondence. At the heart of this digital communication infrastructure lies the mail server, a pivotal component facilitating the exchange of electronic messages across the vast expanse of the internet.

Definition and Functionality:

A mail server, also known as a mail transfer agent (MTA) or mail relay, is a specialized server responsible for sending, receiving, and storing email messages. Its primary function is to handle the routing of emails between sender and recipient, ensuring the timely and secure delivery of electronic messages.

- **key components:-** Mail servers consist of several key components, each playing a crucial role in the email transmission process. These components include the Mail User Agent (MUA), which is the email client used by individuals to compose and send messages, and the Mail Delivery Agent (MDA), responsible for storing incoming messages in the recipient's mailbox
- **Protocols:-** To achieve seamless communication between mail servers and email clients, various protocols are employed. The Simple Mail Transfer Protocol (SMTP) is fundamental for sending emails, while the Post Office Protocol (POP) and Internet Message Access Protocol (IMAP) govern the retrieval of messages by end-users.
- **security:-** Given the sensitive nature of email content, mail servers incorporate robust security measures. Encryption protocols such as Transport Layer Security (TLS) safeguard data during transmission, while authentication mechanisms, such as DomainKeys Identified Mail (DKIM) and Sender Policy Framework (SPF), enhance the integrity of email communication.

Project Overview:

In my project, I have simulated a multi threaded dynamic mail server, named nazeeb_server, which can handle multiple clients. It is developed in c++ language. The server is the pivot of communication between clients. It works with alliance to the IMAP protocol, meaning that user can view and manipulate messages (here in my project, user can read, delete) as though they are stored locally on their devices. Changes in one device of the user is reflected on the server.

Currently, it works with one pc, but later the programme can be extended and two or multiple pcs can communicate.

Once a client connects to the server via terminal, the server gives him three options,

1. Sign up
2. log in
3. Exit

A thing to note is that, the server provides a unique mail ID to the user while signing up. Meaning that, no two users will have the same mail ID.

Then the user has to provide his password. While providing the password, the password is masked with asterisk symbols, so that it can't be seen.

Then the password is Hashed via one of the most famous hashing algorithms, sha 256, by the server.

This sha 256 is a major part of my programme to ensure secure login and maintain user integrity.

The userID, his first name and hashed password is then stored as a record in a file in the server.

Later these records are used to verify user login. After login, users can SEND email messages to other userIDs, READ their own, DELETE email messages, see the LIST of their own inbox messages.

ALL of these are done via socket programming and connection with the server.

Background of the project

The main topics that are related to my projects are,

1. socket programming
2. sha 256 hashing algorithm
3. concept of threading
4. mail server protocol

One must have a clear understanding of these in order to understand my project. Let me give a brief definition one by one, then I will give the links and references of articles/videos of these concepts so that anyone can understand them easily.

In networking, a socket is a fundamental concept that represents an endpoint for communication between two machines. In C++, sockets are used to establish a connection between a client and a server, enabling them to send and receive data.

Most interprocess communication uses the client server model. These terms refer to the two processes which will be communicating with each other. One of the two processes, the client, connects to the other process, the server, typically to make a request for information. A good analogy is a person who makes a phone call to another person. Notice that the client needs to know of the existence of and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established.

Notice also that once a connection is established, both sides can send and receive information.

The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. A socket is one end of an interprocess communication channel. The two processes each establish their own socket.

Socket programming shows how to use socket APIs to establish communication links between remote and local processes.

Here is the must read link for understanding basics of socket programming :-

<https://marketsplash.com/tutorials/cpp/cplusplus-socket/>

In my project, I have done some modifications to fit the programme to linux, so if one understands the basics, he/she can easily interpret my programme.

At number two, there is the SHA 256 hashing algorithm. It is primarily used in blockchain and bitcoin to ensure proof of work. SHA means Secure Hash Algorithm and 256 means that it outputs any message into 256 bits long string. Basically it outputs 64 HEX characters, as we know a HEX character is 4 bits, so $64 \times 4 = 256$ bits. This cryptographic hash function converts a character, a word, a sentence, a chapter, even a book into 64 HEX characters long string. Meaning, the input message can be of variable length but the output string will always be 64 hex characters long string.

For the same input message, the output hash value will always be the same. But a slight change in the input message, suppose changing a lower case letter to upper case, will greatly impact the output hash value. It is called the Avalanche effect.

I have used the SHA 256 algorithm to ensure secure login of users. During signup, the password that a user inputs, its hash value is calculated by the SHA 256 Hash() function and registered in the server. While logging in, the string that the user inputs as password, its hash value is calculated and is compared with the registered hash value that is stored in the server's file. If matched, only then the user can login, otherwise he can't.

Some properties of the SHA 256:-

- Deterministic: Produces the same hash for the same input.
- Quick to compute: Easy to compute the hash Value of any given message(here,password).
- Preimage resistance: Infeasible to generate a password from a given hash value.
- Avalanche Effect: Impossible to modify a password without changing the hash.
- Resistance to collision: Impossible to find two different passwords with the same hash.

To understand the project implementation, watching the following video is a must. I have just turned it into executable code in my project. SHA-256 (COMPLETE CONCEPT & DETAILED STEP-BY-STEP EXPLANATION):-

<https://youtu.be/9xs4eWOAG7Y?si=vvtsZhV87G5cEvWI>

At third, we have the concept of threading. In a real-world scenario, a server often needs to handle multiple clients simultaneously. One common approach to achieve this is by using multithreading. Each client connection is handled by a separate thread, allowing the server to manage multiple clients at the same time. Basically mail server is a server which can handle multiple clients at the same time. So, in order to handle multiple clients, we use the power of threading.

One concept that is also very important is "mutual exclusion". It is performed by the <mutex.h> header file. In computer programming, a mutual exclusion (mutex) is a program object that prevents multiple threads from accessing the same shared resource simultaneously. A shared resource in this context is a code element with a critical section, the part of the code that should not be executed by more than one thread at a time. In my programme, the critical section is where I write messages to the user's inbox file. This is a critical section because, if another thread runs the same section, file override can happen and the actual message can get corrupted.

At fourth, we have the mail server protocols. I have implemented the IMAP protocol, which keeps emails on the server, allowing for synchronized access from multiple devices (here terminals) with changes reflected across all platforms.

3. Description of the Project

nazeeb_server is a mail server which provides unique mail addresses to users while signing up. Users can send mail to others using mail addresses and read or delete mail from their mailbox, which reflects on the server. while signing up, the server uses sha 256 algorithm to hash the password and register the user's account.

The main programme parts of the project are as follows-

1)myhelper(folder)

a)myhelper.cpp → helps with manipulating files and folders.

b)myhelper.h

2)mysocket(folder)

a)mysocket.cpp → has all the necessary codes for socket programming.

b)mysocket.h

3)SHA-Algo

a)sha.h

b)sha256().h → programme to generate hash value for password.

c)testmain.cpp → to test and print the hash value of inputs.

4)uniqueMailID.txt → unique mail IDs and a corresponding boolean value for user signup.

5)Accounts.txt → registered userIDs, their firstname and the hash value of user password.

6)validate_signup.cpp → programme for server to provide unique id to a new user and

7)myclient.cpp → Necessary client code for mail operations

8)myserver.cpp → Necessary server code for mail operations(SEND,READ,DEL,LOGOUT)

9)MailBoxServer → server folder which has multiple userID folders(inboxes of users)

| nazeeb_server | | |
|---|--|--|
| uniqueMailID.txt uniqueids Availability | Accounts.txt(registered ids) | MailBoxServer(Folder Inboxes) |
| bsse1401@nazeeb.com 1 bsse1402@nazeeb.com 1 bsse1403@nazeeb.com 0 bsse1404@nazeeb.com 1 bsse1405@nazeeb.com 0 bsse1406@nazeeb.com 1 bsse1407@nazeeb.com 1 | bsse1403@nazeeb.com <firstname> <HashValue> bsse1405@nazeeb.com <firstname> <HashValue> | bsse1403@nazeeb.com (inbox folder) →1_subject1.txt →2_subject2.txt bsse1405@nazeeb.com (inbox folder) →1_subject1.txt →2_subject2.txt |

Here, as you can see, nazeeb_server has three segments. **uniqueMailID.txt** has stored all the unique mailIDs and a boolean value corresponding to it to keep track of whether the ID has been previously given to a user or not while signing up. Boolean '1' means its available and a new user can have this id. Boolean '0' means the corresponding id has already been given to a user and therefore a new user can't have that id to sign up.

While signing up, the terminal prompts the user to enter his userID in the format of "bsse+4digits+nazeeb.com". The input format is checked by regex. Then the background server programme checks in the uniqueMailID.txt if the input username is available or not. If not available, it again asks the user to give another username in the same format. If available, then it marks the boolean value corresponding to the username from 1 to 0 (i.e, making it unavailable for another user). Furthermore, the user proceeds to give his firstname and password in the terminal. The password is hashed by the SHA 256 algorithm in the backend.

When he/she presses 'enter', his uniqueID, firstname and hashed password is stored in the '**Accounts.txt**' file as a registered account.

later when the user tries to login with his mail ID, two things happen in the backend. Firstly, his entered userID is checked whether it exists in the registered 'Accounts.txt' file or not, and secondly, if exists, the server matches the hash of the input password of user with the corresponding hash value of the userID stored in the 'Accounts.txt' file. IF matched, only then he can proceed to after login operations (SEND, READ, LIST, DEL, QUIT (logout)).

After login, users can SEND mail messages by specifying the SENDER and RECEIVER mailIDs, subject and message. Here, the server checks if the receiver is registered or not, i.e the server checks in the Accounts.txt file to verify the receiver. The server also checks whether the sender input is the same as the one that was used to login.

By typing 'enter+dot+enter' contiguously, the message is marked as END and is sent to the receiver.

Now, the question arises, how does a receiver save the messages in his inbox?

Here the 'MailBoxServer' segment of nazeeb_server comes into play. It is the super folder of all the username Inbox folders where the messages get stored. The messages are stored in the format of - "aMailNumber_subject.txt". The mail_number is unique. Later it is used to delete or read the specified mail message.

This mail_number is counted as follows-

mail_number = amounts of txt files present in the unique userID inbox folder + 1;

A standard practice would have been the use of an UUID. But as it is not supported by c++, a rather ugly solution is used.

In the MailBoxServer, if the receiver directory doesn't exist but the receiver is a registered ID, the server uses the mkdir() function call to create a receiver directory and store the mail message in the specified format (mailnumber_subject.txt)

Now let's dive into the LIST operation. Here the user has to give the usernameID. If it exists in the MailBoxServer, then only the terminal shows the list of messages present in that inbox folder. But the user doesn't have the Access to read them.

In case of READ and DEL operations, the terminal prompts the user to enter his own userID and the mail number he/she wants to read or delete. If the user enters other people's id, the terminal shows an error message and asks the user to enter his own userID, as one should not have the right to read or delete other people's inbox messages. BY typing 'QUIT' the user can logout and by entering option '3' the user can exit the programme.

4.Implementation and Testing

Now lets go through the classes and programmes one by one,then I will provide the user manual so that one can run the programme easily.

4.1)myhelper folder:It has a myhelper.cpp programme which has MyHelper class. This class is designed to provide utility functions for managing directories, files, and command-line arguments in the context of a mail server application. The methods support operations like validating arguments, checking directories, and handling files.

```
class MyHelper
{
private:
    struct dirent *dirent;
    DIR *dir;
    string buffer;

public:
    MyHelper();
    void checkServerArguments(int argc, char *argv[]);
    void checkClientArguments(int argc, char *argv[]);
    bool stringIsInt(string string);
    int filesInDirectory(string dir);
    vector<string> filenamesInDirecotry(string dir);
    vector<string> subjectsInDirectory(string dir);
    bool fileExists(string filename);
    ~MyHelper();
};
#endif
```

a)Private Data Members:

- struct dirent *dirent: A pointer to a dirent structure, typically used for directory entries. It's used in methods that interact with directories.
- DIR *dir: A pointer to a DIR structure, representing an open directory stream. Used for directory-related operations.
- string buffer: A string buffer, used for temporary storage in various methods.

b)Public Member Functions:

- Constructor (MyHelper()):
The default constructor, used to initialize the object. It doesn't take any parameters and sets up any necessary resources or performs initializations.
- Destructor (~MyHelper()):
The destructor, used to clean up resources when the object is destroyed. It releases memory, closes files, or performs other cleanup tasks.
- void checkServerArguments(int argc, char *argv[]):
Takes the command-line arguments (argc and argv[]) and likely checks if they are valid for a mail server. The command line arguments for the server are-
<./programme name> <port number> <argDirectory(directory of the MailBoxServer)>
.The function performs validation and prints messages or exits the program if the arguments are not correct.
- void checkClientArguments(int argc, char *argv[]):

Similar to checkServerArguments, but for client-side arguments. The client side arguments are <./programme name> <server ip address> <server port number>.

- bool stringIsInt(string string):
Takes a string as input and checks if it represents an integer. Returns true if the string is an integer, otherwise false.
- int filesInDirectory(string dir):
Takes a directory path as input and returns the number of files in that directory. It uses the dirent structure and related functions to interact with the filesystem.
- vector<string> filenamesInDirecotry(string dir):
Takes a directory path as input and returns a vector of strings containing the filenames in that directory.
- vector<string> subjectsInDirectory(string dir):
Takes a directory path as input and returns a vector of strings containing the subjects in that directory.
- bool fileExists(string filename):
Takes a filename as input and checks if the file exists. Returns true if the file exists, otherwise false.

4.2)mysocket folder: it has the necessary programme codes for socket programming. The main part is the mysocket.cpp programme.lets understand it segment by segment.

```
MySocket::MySocket(int port)
{
    memset(&address, 0, sizeof(address));
    address.sin_family = AF_INET;
    address.sin_port = htons(port);
    address.sin_addr.s_addr = INADDR_ANY;
}
```

This constructor initializes a server socket. It sets up the socket address structure (address) with the specified port and binds to any available network interface.

```
MySocket::MySocket(const char *addr, int port)
{
    memset(&address, 0, sizeof(address));
    address.sin_family = AF_INET;
    address.sin_port = htons(port);
    inet_aton(addr, &address.sin_addr);
}
```

This constructor initializes a client socket. It sets up the socket address structure (address) with the specified IP address and port.

```

void MySocket::createSocket()
{
    create_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (create_socket == -1)
    {
        socketError("create socket error");
    }
}

```

This function creates a socket using the socket system call. It sets the socket type to SOCK_STREAM, indicating a TCP socket.

```

void MySocket::connectSocket()
{
    if (connect(create_socket, (struct sockaddr *)&address, sizeof(address)) == -1)
    {
        socketError("connect error");
    }
    cout << "Connection with server " << inet_ntoa(address.sin_addr) << " established" << endl;
}

```

This function connects a client socket to the server using the connect system call. It also prints a message indicating that the connection has been established.

```

void MySocket::bindSocket()
{
    if (bind(create_socket, (struct sockaddr *)&address, sizeof(address)) == -1)
    {
        socketError("bind error");
    }
}

```

This function binds a server socket to a specific IP address and port using the bind system call.

```

void MySocket::listenSocket()
{
    if (listen(create_socket, 5) == -1)
    {
        socketError("listen error");
    }
    addrlen = sizeof(struct sockaddr_in);
}

```

This function puts the server socket in a listening state, allowing it to accept incoming connections.

```

int MySocket::acceptNewConnection()
{
    new_socket = accept(create_socket, (struct sockaddr *)&cliaddress, &addrlen);
    if (new_socket == -1)
    {
        socketError("accept error");
    }
    cout << "Client connected from " << inet_ntoa(cliaddress.sin_addr) << ":" << ntohs(cliaddress.sin_port) << endl;
    strcpy(buffer, "Welcome to nazeeb's server, Please enter your command:\0");
    if (send(new_socket, buffer, strlen(buffer), 0) == -1)
    {
        socketError("send error");
    }
    return new_socket;
}

```

This function accepts a new connection from a client, prints information about the client, and sends a welcome message to the client.

```

const char *MySocket::recvMessage(int socket)
{
    if (socket == -1)
    {
        // client
        size = recv(create_socket, buffer, BUF - 1, 0);
    }
    else
    {
        // server
        size = recv(socket, buffer, BUF - 1, 0);
    }

    if (size == -1)
    {
        socketError("recv error");
    }
    else if (size == 0)
    {
        return "QUIT\n\0";
    }
    buffer[size] = '\0';
    return buffer;
}

```

This function receives a message from either the client or the server, depending on the value of the socket parameter.

```

void MySocket::sendMessage(const char *message, int socket)
{
    if (socket == -1)
    {
        // client
        if (send(create_socket, message, strlen(message), 0) == -1)
        {
            socketError("send error");
        }
    }
    else
    {
        // server
        if (send(socket, message, strlen(message), 0) == -1)
        {
            socketError("send error");
        }
    }
}

```

This function sends a message to either the client or the server, depending on the value of the socket parameter.

```

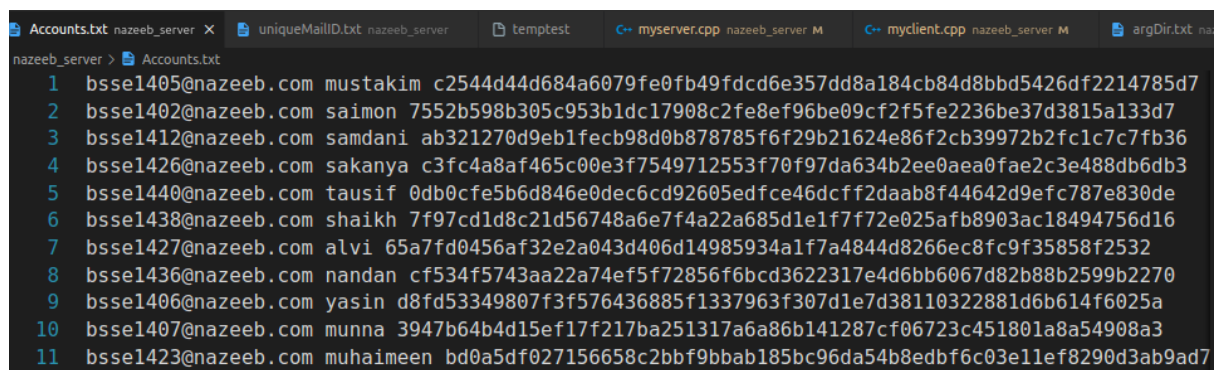
void MySocket::socketError(const char *errorMessage)
{
    perror(errorMessage);
    exit(EXIT_FAILURE);
}

```

This function prints an error message using perror and exits the program with an error code.

4.3) server segments:

As mentioned before, the server has 3 segments- uniqueMailID.txt, Accounts.txt, and MailBoxServer. The requirement and functionality of these 3 are mentioned in Chapter 3 (Description of the project) of the report. Please visit that one first. Here are some pictures of these in the programme-



```

nazeeb_server > Accounts.txt
1 bsse1405@nazeeb.com mustakim c2544d44d684a6079fe0fb49fdcd6e357dd8a184cb84d8bbd5426df2214785d7
2 bsse1402@nazeeb.com saimon 7552b598b305c953b1dc17908c2fe8ef96be09cf2f5fe2236be37d3815a133d7
3 bsse1412@nazeeb.com samdani ab321270d9eb1fecb98d0b878785f6f29b21624e86f2cb39972b2fc1c7c7fb36
4 bsse1426@nazeeb.com sakanya c3fc4a8af465c00e3f7549712553f70f97da634b2ee0aea0fae2c3e488db6db3
5 bsse1440@nazeeb.com tausif 0db0cfe5b6d846e0dec6cd92605edfce46dcff2daab8f44642d9efc787e830de
6 bsse1438@nazeeb.com shaikh 7f97cd1d8c21d56748a6e7f4a22a685d1e1f7f72e025afb8903ac18494756d16
7 bsse1427@nazeeb.com alvi 65a7fd0456af32e2a043d406d14985934a1f7a4844d8266ec8fc9f35858f2532
8 bsse1436@nazeeb.com nandan cf534f5743aa22a74ef5f72856f6bcd3622317e4d6bb6067d82b88b2599b2270
9 bsse1406@nazeeb.com yasin d8fd53349807f3f576436885f1337963f307d1e7d38110322881d6b614f6025a
10 bsse1407@nazeeb.com munna 3947b64b4d15ef17f217ba251317a6a86b141287cf06723c451801a8a54908a3
11 bsse1423@nazeeb.com muhaimeen bd0a5df027156658c2bbf9bbab185bc96da54b8edbf6c03e11ef8290d3ab9ad7

```

Figure:Accounts.txt file of the server

```
uniqueMailID.txt nazeeb_server X C++ validate_signup.cpp nazeeb_server M Accounts.t
nazeeb_server > uniqueMailID.txt
1 bsse1401@nazeeb.com 1
2 bsse1402@nazeeb.com 0
3 bsse1403@nazeeb.com 1
4 bsse1404@nazeeb.com 1
5 bsse1405@nazeeb.com 0
6 bsse1406@nazeeb.com 0
7 bsse1407@nazeeb.com 0
8 bsse1408@nazeeb.com 1
9 bsse1409@nazeeb.com 1
10 bsse1410@nazeeb.com 1
11 bsse1411@nazeeb.com 1
12 bsse1412@nazeeb.com 0
```

Figure: uniqueMailID.txt

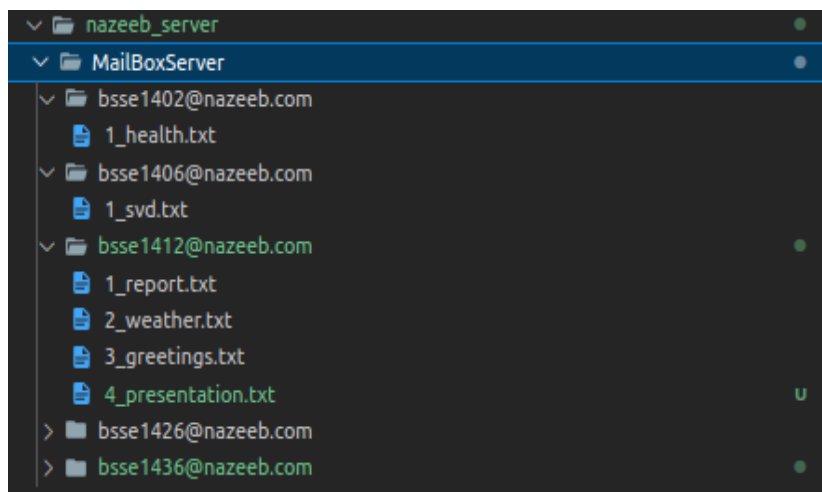


Figure:MailBoxServer(super folder of all username inboxes)

We also have a programme named **validate_signup.cpp** which is the programme that helps to provide unique IDs to a new user and register his/her userID,firstname and hashed password in Accounts.txt file.

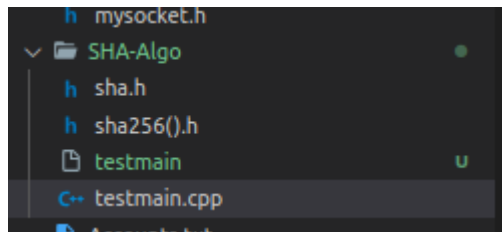
```
C++ validate_signup.cpp nazeeb_server M X uniqueMailID.txt nazeeb_server Accounts.txt nazeeb_server temptest
nazeeb_server > C++ validate_signup.cpp > ...
1 #include<bits/stdc++.h>
2 using namespace std;
3 #include<regex>
4 #include "SHA-Algo/sha.h"
5 #include<conio.h>
6
7
8 > bool checkID_and_signup(string str){...
86
87 > void signUpHelper(){...
110
```

4.4)SHA-ALGO:

This folder is responsible for calculating the 64 Hex character long Hash value for passwords.

To understand the code,you must visit and watch the following video.

<https://youtu.be/9xs4eWOAG7Y?si=PKqyv2k-zyQq0kdQ>



You can run the testmain.cpp in this folder to compute the hash value of any message(a word,a paragraph,even a chapter will generate 64 characters long hash value).

```
dhrubo@dhrubo-ThinkPad-T470s:~/Desktop/spl1/nazeeb_server/SHA-Algo$ cd "/home/dhrubo/
-o testmain && "/home/dhrubo/Desktop/spl1/nazeeb_server/SHA-Algo/"testmain
Welcome to SHA-256 Hash Generator!

Enter the message : mypassword
SHA-256 hash : 89e01536ac207279409d4de1e5253e01f4a1769e696db0d6062ca9b8f56767c8
Hash computed successfully!
dhrubo@dhrubo-ThinkPad-T470s:~/Desktop/spl1/nazeeb_server/SHA-Algo$
```

I have included this folder in two programmes, one is the validate_signup.cpp and the other is myclient.cpp for signup or login related operations.

4.5)myserver.cpp:

This C++ program is a simple implementation of a mail server using the IMAP (Internet Message Access Protocol) protocol. The server supports basic functionalities like sending, listing, reading, and deleting emails. Below is a short explanation of the key components of the code:

4.5.1)Header and Namespace:

```
#include <sys/types.h>
#include <sys/socket.h>
// ... (other header files)
#include "myhelper/myhelper.cpp"
#include "mysocket/mysocket.cpp"
```

The code includes several header files for socket programming, file operations, threading, and other standard libraries. Additionally, it includes custom header files for MyHelper and MySocket classes.

4.5.2)Mutex Header For thread Safety:

```
mutex mtx;//code
```

This mutex (mtx) is used to provide thread safety when multiple threads (client connections) access shared resources concurrently.

4.5.3)Server Thread Function:

```
void serverThread(MyHelper helper, MySocket &serverSocket, int new_socket, string argDir)
{
    // ... (thread function implementation)
}
```

The serverThread function contains the logic for handling different IMAP commands sent by clients, such as sending emails ("SEND"), listing emails ("LIST"), reading emails ("READ"), deleting emails ("DEL"), and quitting the connection ("QUIT"). The actual email data is stored in files within directories, and the server communicates with clients using socket-based communication. Thread safety is ensured through the use of a mutex (mtx).

4.5.4)Main Function:

```
int main(int argc, char *argv[])
{
    // ... (main function implementation)
}
```

The main function initializes the server, creates a socket, binds it to a specified port, listens for incoming connections, and spawns a new thread for each connected client using the serverThread function.

- **Server initialization:**

```
MyHelper helper;
helper.checkServerArguments(argc, argv);
```

An instance of the MyHelper class is created to handle server-specific functionalities. The command-line arguments are checked for correctness using checkServerArguments. The server arguments are <./programme name> <port number> <argDirectory(directory of the MailBoxServer)>

- **Socket initialization and listening:**

```
MySocket serverSocket(stoi(argv[1]));
serverSocket.createSocket();
serverSocket.bindSocket();
serverSocket.listenSocket();
```

An instance of the MySocket class is created for handling socket operations. The server socket is created, bound to the specified port, and set to listen for incoming connections.

- **Handling incoming connections:**

```
int new_socket;
while (1)
{
    new_socket = serverSocket.acceptNewConnection();
    thread(serverThread, helper, ref(serverSocket), new_socket, argv[2]).detach();
}
```

The server enters a loop to continuously accept incoming connections. For each new connection, a new thread is spawned, passing the client socket (new_socket) and necessary information to the serverThread function.

The program continues running until manually terminated, and each connected client is handled concurrently in a separate thread.

4.6)myclient.cpp:

This C++ program is a simple implementation of a mail client using the IMAP (Internet Message Access Protocol) protocol. The client allows users to perform various operations like signing up, logging in, sending emails, listing emails, reading emails, and deleting emails. Below is an explanation of the key components of the code:

4.6.1)Header and Namespace:

```
#include <sys/types.h>
#include <sys/socket.h>
// ... (other header files)
#include "myhelper/myhelper.cpp"
#include "mysocket/mysocket.cpp"
#include "validate_signup.cpp"
#include <conio.h>
```

The code includes several header files for socket programming, file operations, and other standard libraries. Additionally, it includes custom header files for MyHelper and MySocket classes, as well as validate_signup.cpp and conio.h for signing up and password masking functionalities.

4.6.2)Global Variables:

```
pair<string, string> pairloggedInUser;           // logged-in user's corresponding firstname
pair<pair<string, string>, string> pairloggedInUserRecord;
```

These global variables store information about the logged-in user, such as their username, firstname, and hashed password.

4.6.3)Functions to Check User and Receiver Existence:

```
bool checkIfReceiverIsRegisteredOrNot(string str);
bool checkIfUserIsInLoggedInPairOrNot(string str);
bool checkIfUsernameExistsInServerOrNot(string str);
```

These functions check whether a given username or receiver email is registered in the server. checkIfUserIsInLoggedInPairOrNot verifies if the user is the currently logged-in user.

4.6.4)Login Helper Function:

```
bool login_helper();
```

This function handles the login process. It prompts the user to enter their username and password, validates the credentials, and sets the global variables pairloggedInUser and pairloggedInUserRecord upon successful login.

4.6.4)Main Function:

```
int main(int argc, char *argv[])
{
    // ... (main function implementation)
}
```

The main function initializes the client, creates a socket, connects to the server, and allows the user to perform various operations.

- **Client Initialization:**

```
MyHelper helper;
helper.checkClientArguments(argc, argv);
```

An instance of the MyHelper class is created to handle client-specific functionalities. The command-line arguments are checked for correctness using checkClientArguments. The client command line arguments are: <./programme name> <server ip address> <server port number>.

- **Socket initialization and connection:**
MySocket clientSocket(argv[1], stoi(argv[2]));
clientSocket.createSocket();
clientSocket.connectSocket();

An instance of the MySocket class is created for handling socket operations. The client socket is created, and a connection is established with the server.

- **Handling user input:**
char buffer[BUF], command[BUF];

cout << clientSocket.recvMessage() << endl;

while (1)
{
// ... (user interface and command processing)
}

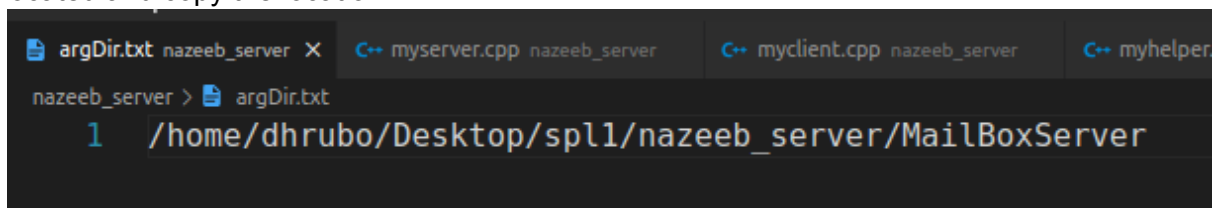
The user is presented with a menu to sign up, log in, or exit. After successful login, the user can perform operations like sending emails, listing emails, reading emails, and deleting emails. The user interacts with the program through the command-line interface.

To sum up,
The program uses socket-based communication to interact with the server, and the user's login information and email data are processed and validated during various operations.

5. User Interface and Manual

STEPS:

1. Shootout at least two terminals ,(one for the server and the others for the client) inside of where nazeeb_server is located in your pc.
2. you must change the argDIR.txt to the path directory of where the MailBoxServer is located and copy the location.



3. type "g++ myserver.cpp -o server" to compile the myserver.cpp programme. Then type "./server portNumber argDIR" to start nazeeb_server. you will get a "waiting for connections" message showing in the terminal. similarly, type "g++ myclient.cpp -o client" to compile the myclient.cpp programme. Then type "./client serverIP serverPortNumber" . Then you will get a menu to run your programme.

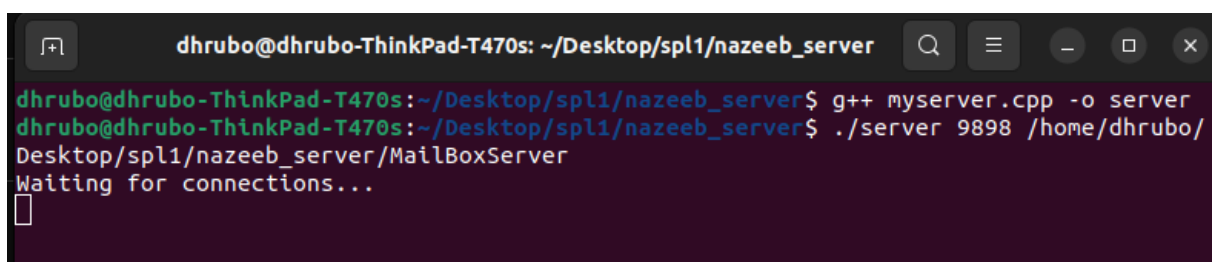


Figure: Server terminal

```

dhrubo@dhrubo-ThinkPad-T470s:~/Desktop/spl1/nazeeb_server$ g++ myclient.cpp -o client
dhrubo@dhrubo-ThinkPad-T470s:~/Desktop/spl1/nazeeb_server$ ./client 127.0.0.1 9898
Connection with server 127.0.0.1 established
Welcome to nazeeb's server, Please enter your command:
-----
1.SignUp
2.Login
3.Exit
-----
Enter your choice(OPTION NO.):

```

Figure:Client terminal

4) I leave the rest to you, just follow what the client terminal tells you to do and you can run the programme!!

I want to let you know that there is a little bug in the programme. while doing the operations, the client process may unexpectedly terminate. In that case, just press the UpArrow in your keyboard and press enter.(meaning running the client programme again, ./client ...).

This might happen because of the existence of an unexpected character in the buffer. You might fix it later.

6.Challenges Faced

1)I had some problems with maintaining the uniqueness of new incoming file.As UUID is not supported in c++, I had to come up with a solution to provide a unique mailID to each of the incoming messages.I did that by counting the number of total messages present in the inbox folder of a user and then incrementing it by one.

Then I named the new message file by " mailID_subject.txt"

2)I had the challenge of ensuring unique username IDs given by the server. I overcame this with the help of regex and created a uniqueMailID.txt file where uniqueIDs have their corresponding boolean value.

3)When entering the mail number that I want to read or delete, the server was doing the operation on wrong mail message.Therefore, it was seen that if i entered mail number 2 to delete, mail number 3 would have gotten deleted.If I entered mail number 2 to read, the server would send me mail number 3 to read.

I solved this problem by simply sorting the mails vector.Then it worked properly.

4)I faced a challenge to ensure that no unexpected character was in the buffer. To ensure this, I have written cin.ignore() in some places of my code.

But even after this, the client process sometimes terminates unexpectedly when pressing enter.

That I couldn't figure out , so, the programme has this small problem.

To continue running the programme,user just have to type "UpArrow+enter" to continue running his programme.

5)In my proposal, I said that I would compress files and send them.As I was having a problem with the previously mentioned buffer bug, I couldn't implement this .But surely, after a lot of debugging, the problem can be solved.(meaning printing the buffer each time before sending or after receiving).

7. Conclusion

In conclusion, the development and implementation of the Mail Server project have provided valuable insights into the complexities of designing a robust and secure email communication system. This project aimed to create a simplified mail server that supports essential functionalities such as user registration, login, sending, receiving, listing, reading, and deleting emails. The key takeaways from this project are as follows:

Socket Programming Skills:

The project heavily relied on socket programming to facilitate communication between the client and server. Implementing a reliable socket-based connection was crucial for the seamless exchange of email-related data.

File Handling and Directory Operations:

The server component involved handling user accounts and storing email messages as files in specific directories. File operations were used to create, read, and delete files, while directory operations managed the organization of user-specific data.

User Authentication and Security:

The project addressed user authentication through hashed passwords, adding a layer of security to protect user credentials. While the project focused on basic security measures, a real-world application would require more sophisticated security mechanisms, such as encryption for data in transit.

Multi-Threading and Concurrent Processing:

Multi-threading was implemented to handle multiple client connections concurrently. This feature enhances the server's efficiency by allowing it to serve multiple clients simultaneously, preventing one user's actions from affecting others.

User Interface and Interaction:

The command-line interface provided users with a straightforward means of interacting with the system. This simplicity is intentional, and for a real-world application, a more user-friendly graphical interface would be essential.

Error Handling and User Feedback:

The project included error handling mechanisms to notify users of issues, such as invalid commands, nonexistent email accounts, or incorrect login credentials. Providing meaningful feedback to users contributes to a more user-friendly experience.

Limitations and Future Improvements:

Acknowledging the limitations of this project, there is room for enhancement and expansion. Implementing advanced features such as attachments, searching, spam filtering, and integrating with external services like LDAP were identified as potential areas for future development. It can be expanded where two pcs can communicate with each other.

Learning Outcomes:

Undertaking this project has been a valuable learning experience, contributing to a deeper understanding of network programming, file handling, and server-client interactions. It has provided an opportunity to apply theoretical knowledge in a practical context, fostering skills in problem-solving and software development.

In summary, while the implemented Mail Server project represents a simplified version of a real-world mail server, it serves as a foundation for further exploration and refinement. As technology continues to evolve, addressing emerging challenges and incorporating additional features will be essential for creating a comprehensive and resilient email communication system.

References:

1)Socket programming Basics:-

<https://marketsplash.com/tutorials/cpp/cplusplus-socet/>

2)Full implementation of SHA 256:-

<https://youtu.be/9xs4eWOAG7Y?si=PKqyv2k-zyQq0kdQ>

3)Mail server Protocols:-

<https://www.geeksforgeeks.org/email-protocols/>