Department of Electrical Engineering and Computer Science

ECCE 356 – Computer Networks

Project - SMTP

Group Members

Tiemar Berhe 100049901

Yaphet Elias 100049897

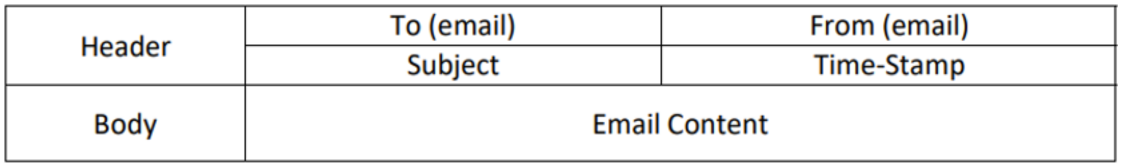
Amine Kidane 100053678

Introduction

By the end of this assignment, we are expected to be able to modify the multithreading code that was used in the prior assignments in order to implement the SMTP functionalities outlined in the Project Description. These functions include message construction and validation, client's validation, and email forwarding between clients through a mail server, saving the emails in the server's and the clients' 'inbox' and 'sent' directories, a client sending email to another unconnected client through a server, and enabling the unconnected client to view their email after connection with the server . The main problem that is expected to be solved is differentiating between different mail server's reply codes as well as transferring mail between different clients connected to a server at different computers.

Background

Simple Mail Protocol (SMTP) is a protocol that handles email transmissions between hosts. Client connects to a mail server to send an email, such that the message has the following header and body.



Header	To (email)	From (email)
	Subject	Time-Stamp
Body	Email Content	

Figure 1. SMTP Message Structure

The mail server verifies the TO and FROM fields of the header before pushing the email to the queue. If these fields are successfully verified, the server replies with a "250 OK" message. Otherwise, the reply is "501 error". 501 error code implies that an email's header argument is not verified. Another code that the server can reply with is "550 error", implying that the email address does not exist.

Project Description

This project extends the functionalities of the SMTP which was completed in the prior assignment. The previous assignment completed the transfer of email from one client to another client through the server. The server was on and waiting for connections as the receiver client connected to the server waiting to receive mail from a sender, while a sender consequently joined the connection to send the mail. The user from the sender side created a new mail in the console. All the input messages of the email i.e. To, From, Subject, Body, an optional attachment as well as a timestamp was processed and sent to the server via TCP. After the email had been sent, the server accepted the email and responded back to the sending client with the appropriate response message. The response message is '250 OK' if the whole process of mail transfer was successful, '501 Error' if the header fields of the mail are not verified or '550 Error' if the client does not exist. And depending on the response message, the client showed the status of the mail transfer.

In addition to all the above-mentioned functionalities, the project should implement scenarios divided into two parts, namely the complete basic scenario, as well as at least one of the advanced functionalities.

The basic scenario:

- Server creates directory for each connected client with ‘inbox’ and ‘sent’ directories each, where each directory name is the client’s email address.
- Client 1 constructs email and sends it to Client 2 with an attachment. The attachment should be any type (pdf, word, png) and more than 2Mb. It should be sent in multiple packets.
- Client 1 connects to the mail server where the name of the mail server is entered by the client
- Client 1 sends email to the server
- Server receives the email to client 2’s ‘inbox’ directory as well as to client 1’s sent directory with the file name format ‘Subject_time’
- Client 2 connects to the mail server, where the name of the mail server is enterd by the client
- Client 2 requests updating the local ‘inbox’ directory by the server
- Server sends client 2 a copy of the local ‘inbox’ directory
- Client 2 saves a copy of emails into the local inbox directory
- Client 1 and 2 list emails in their inbox/sent directories

The advanced features:

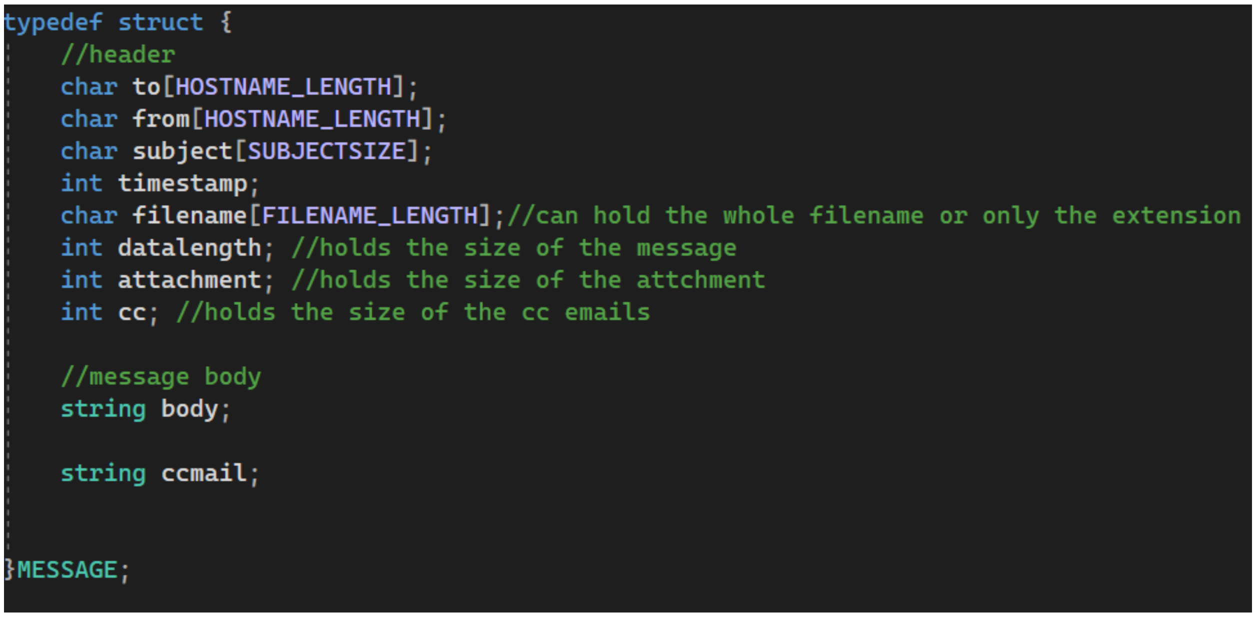
- Multiple receivers: allowing the sender to send an email to multiple receivers
- Forward emails: a client can resend an existing email from ‘inbox’ or ‘sent’ where the new email will be automatically constructed from the existing email
- Mark email as read: once the email is read, as symbol should be added once listing the emails to indicate that it was read

Solution

Important Structures

<u>Message Structure:</u>

To start with we had to modify our previous header to accommodate the new requirements. The new header is a struct that holds the to and from addresses, the timestamp, the length of the message body, size of the attachment, the filename, the length of the string that holds the address of the multiple clients, a string variable that holds the message body as well as a string variable that holds the cc’d email addresses.

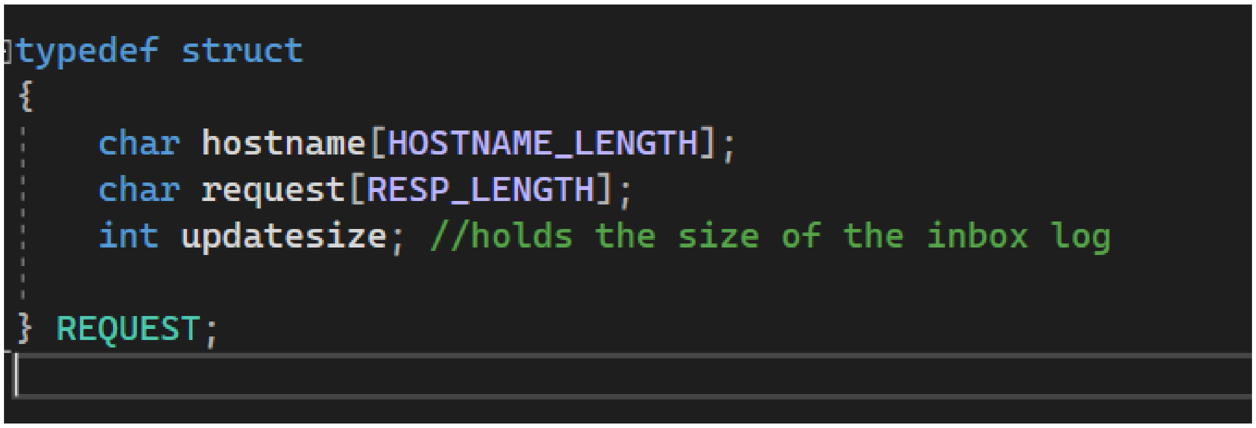


```
typedef struct {
    //header
    char to[HOSTNAME_LENGTH];
    char from[HOSTNAME_LENGTH];
    char subject[SUBJECTSIZE];
    int timestamp;
    char filename[FILENAME_LENGTH];//can hold the whole filename or only the extension
    int datalength; //holds the size of the message
    int attachment; //holds the size of the attchment
    int cc; //holds the size of the cc emails

    //message body
    string body;

    string ccmail;


}MESSAGE;
```

Figure 2. Header - Message Structure

The Request Structure

This structure is used when the server updates the client from the server's inbox directory and sending email. The update process is carried out with the help of a csv file named as logfile, which contains the list of the received inbox emails. To make this possible, the server and the client communicate the request and depending on the request received from the client, email sending or updating the client on the inbox is done.

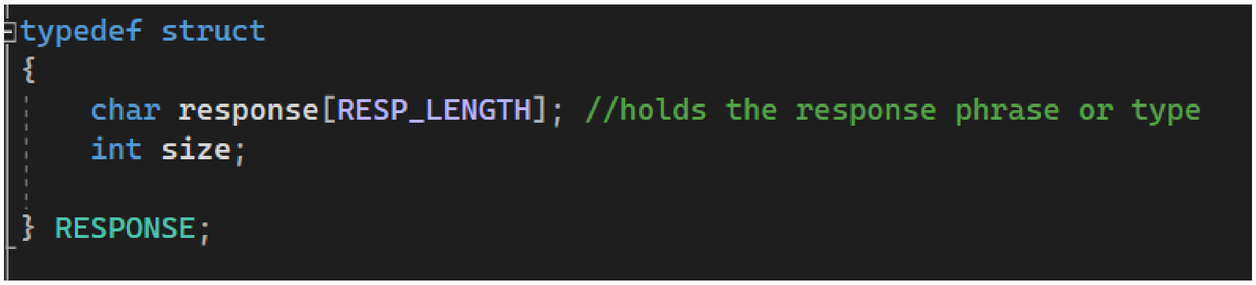


```
typedef struct
{
    char hostname[HOSTNAME_LENGTH];
    char request[RESP_LENGTH];
    int updatesize; //holds the size of the inbox log

} REQUEST;
```

Figure 3. Request Structure

Response Structure

This structure is part of the server/client coordination communication facilitators that holds a response character array and an integer variable that can be used for various purposes depending on the nature of the response message.

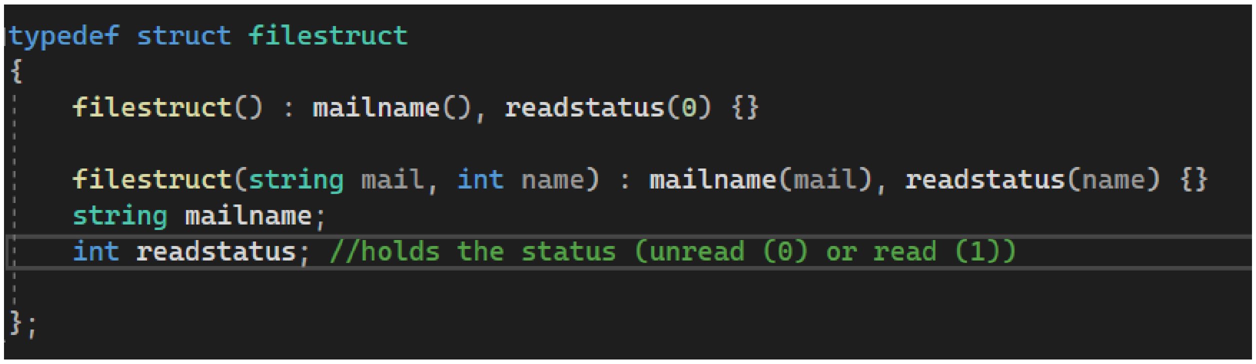


```
typedef struct
{
    char response[RESP_LENGTH]; //holds the response phrase or type
    int size;

} RESPONSE;
```

Figure 4. Response Structure

File Structure

Our approach to the solution involves holding all the inbox and sent email into global map variables which make it easy to work with. The filestruct structure is used as the inbox map value which holds the mail name (subject_timestamp) and an integer status that we are using to tell if the mail is read or not.



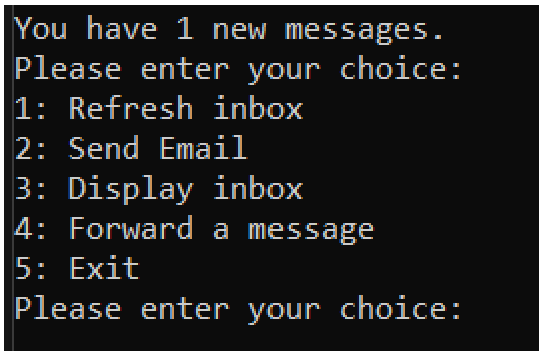
```
typedef struct filestruct
{
    filestruct() : mailname(), readstatus(0) {}

    filestruct(string mail, int name) : mailname(mail), readstatus(name) {}
    string mailname;
    int readstatus; //holds the status (unread (0) or read (1))

};
```

Figure 5. File Structure

Approach and implementation of basic scenario

The scenario flow starts with the client entering the address of the server after which the server checks if the client is registered or not. If the client is not registered the server disconnects the client after showing a message that the client is not registered and being disconnected otherwise the server enters into a loop where it continuously receives the request from the client and performs the required task.

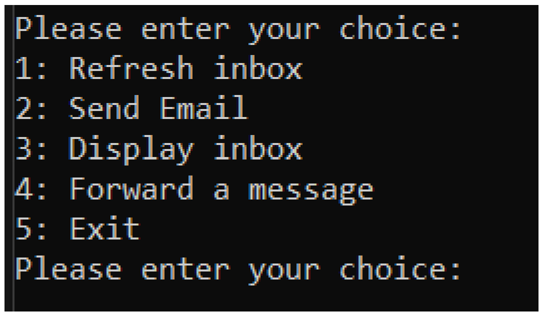
Upon successful connection the client inbox is automatically updated. The updating process starts by the client sending an update request and the server replies which is then followed by sending the log file ("logfile.csv"). The log file holds an entry of every mail name and read status pair. The server after receiving the log file, compares it with what is saved on the server inbox folder of the corresponding client and sends only those emails that the sender doesn't have with their respective attachments if they have one.



```
You have 1 new messages.
Please enter your choice:
1: Refresh inbox
2: Send Email
3: Display inbox
4: Forward a message
5: Exit
Please enter your choice:
```

Figure 6. New mail notification

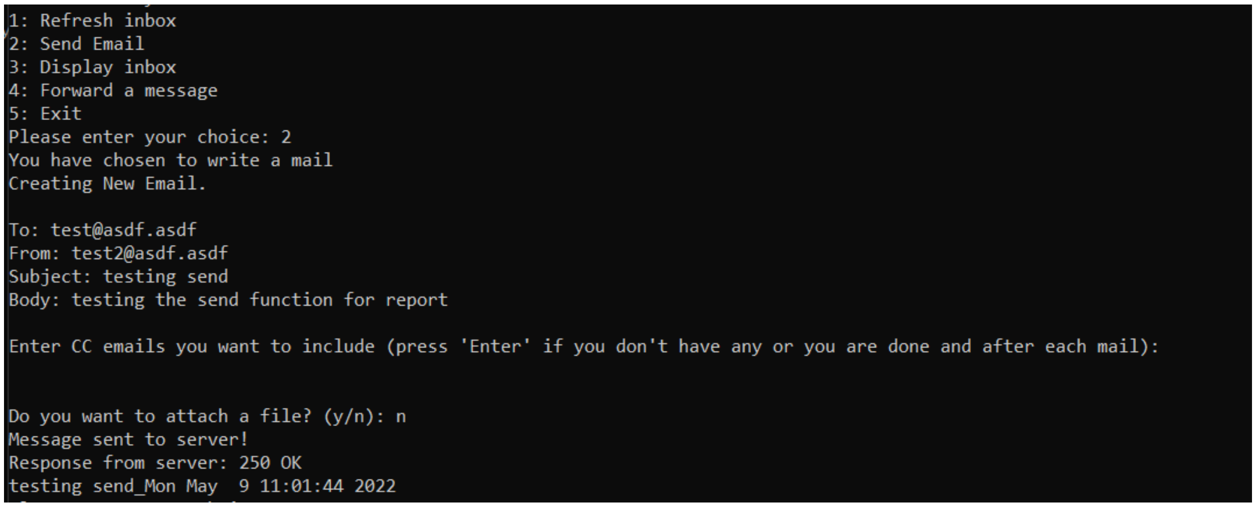
After that the client is prompted to choose from a set of functions we implemented.



```
Please enter your choice:
1: Refresh inbox
2: Send Email
3: Display inbox
4: Forward a message
5: Exit
Please enter your choice:
```

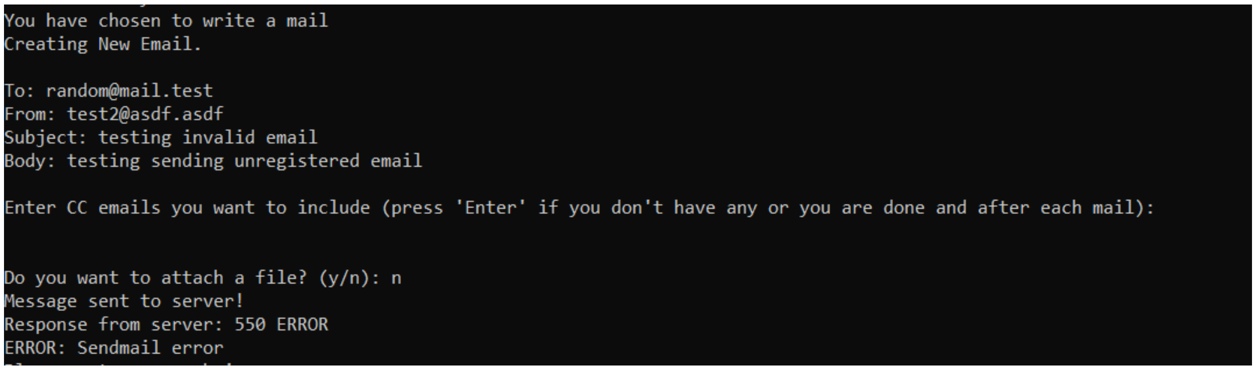
Figure 7. Prompt for user to choose

Another part of the basic scenario is to be able to send a message. Sending a message is initiated by choosing the number that corresponds to the 'Send mail' choice which in turn sends a message-send request to the server. Choosing this directs the client to a prompt where they have to enter the fields of the message being constructed and the message is sent after all the input fields are validated. From the server-side, once it receives a message send request, it prepares to receive the constructed message. After it receives the message, the server first validates the 'To' and 'From' fields and if any of the addresses are entered in an incorrect format, the user is prompted to enter a valid email address. On the other hand, if the email addresses entered by the user are not registered in the mapping file, the server responds with '550 ERROR'. If the addresses are valid, the server then places the messages into their respective client-mail directory. After a successful saving operation, the server responds with '250 OK' to the sending client, and if there is an error in receiving or saving the file it responds with '501 ERROR'.



```
1: Refresh inbox
2: Send Email
3: Display inbox
4: Forward a message
5: Exit
Please enter your choice: 2
You have chosen to write a mail
Creating New Email.

To: test@asdf.asdf
From: test2@asdf.asdf
Subject: testing send
Body: testing the send function for report

Enter CC emails you want to include (press 'Enter' if you don't have any or you are done and after each mail):


Do you want to attach a file? (y/n): n
Message sent to server!
Response from server: 250 OK
testing send_Mon May  9 11:01:44 2022
```

Figure 8. Successful Message Send



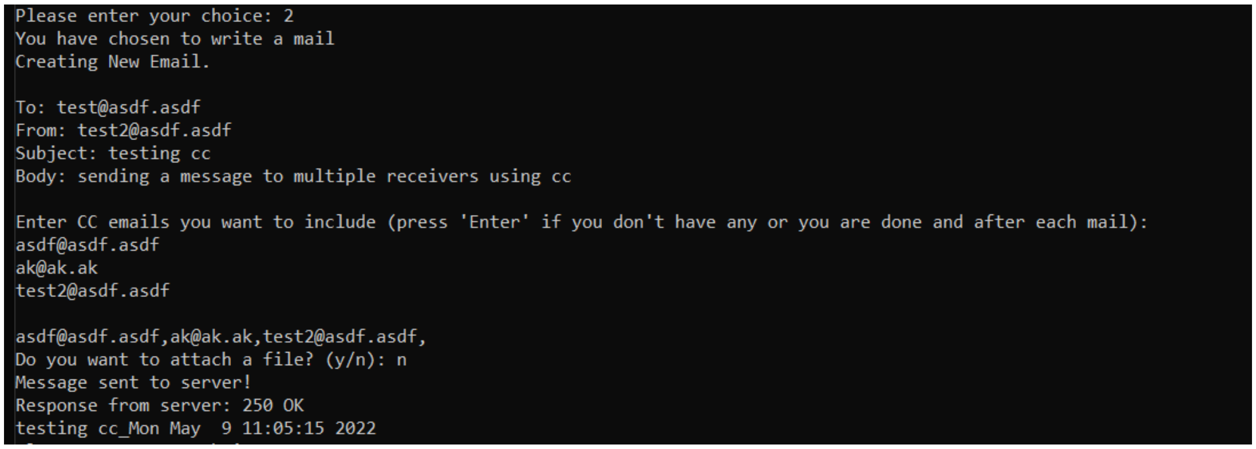
```
You have chosen to write a mail
Creating New Email.

To: random@mail.test
From: test2@asdf.asdf
Subject: testing invalid email
Body: testing sending unregistered email

Enter CC emails you want to include (press 'Enter' if you don't have any or you are done and after each mail):


Do you want to attach a file? (y/n): n
Message sent to server!
Response from server: 550 ERROR
ERROR: Sendmail error
```

Figure 9. Message sent with unregistered receiver mail address

Advanced Features

Multiple receivers

To be able to include more than one receiver we used a ‘ccmail’ string variable, part of the message structure, that is used as a pool to hold all the emails as a string separated by a ‘,’. To get all the mail addresses, we are prompting the user to enter the mails one by one and after each entry we are checking the entered mail address if it is valid and adding it to the ‘ccmail’ string. After the server receives the ccmail it puts all the email addresses into a queue (including the ‘TO’ address) and saves the message into all the address mails that are on the queue.

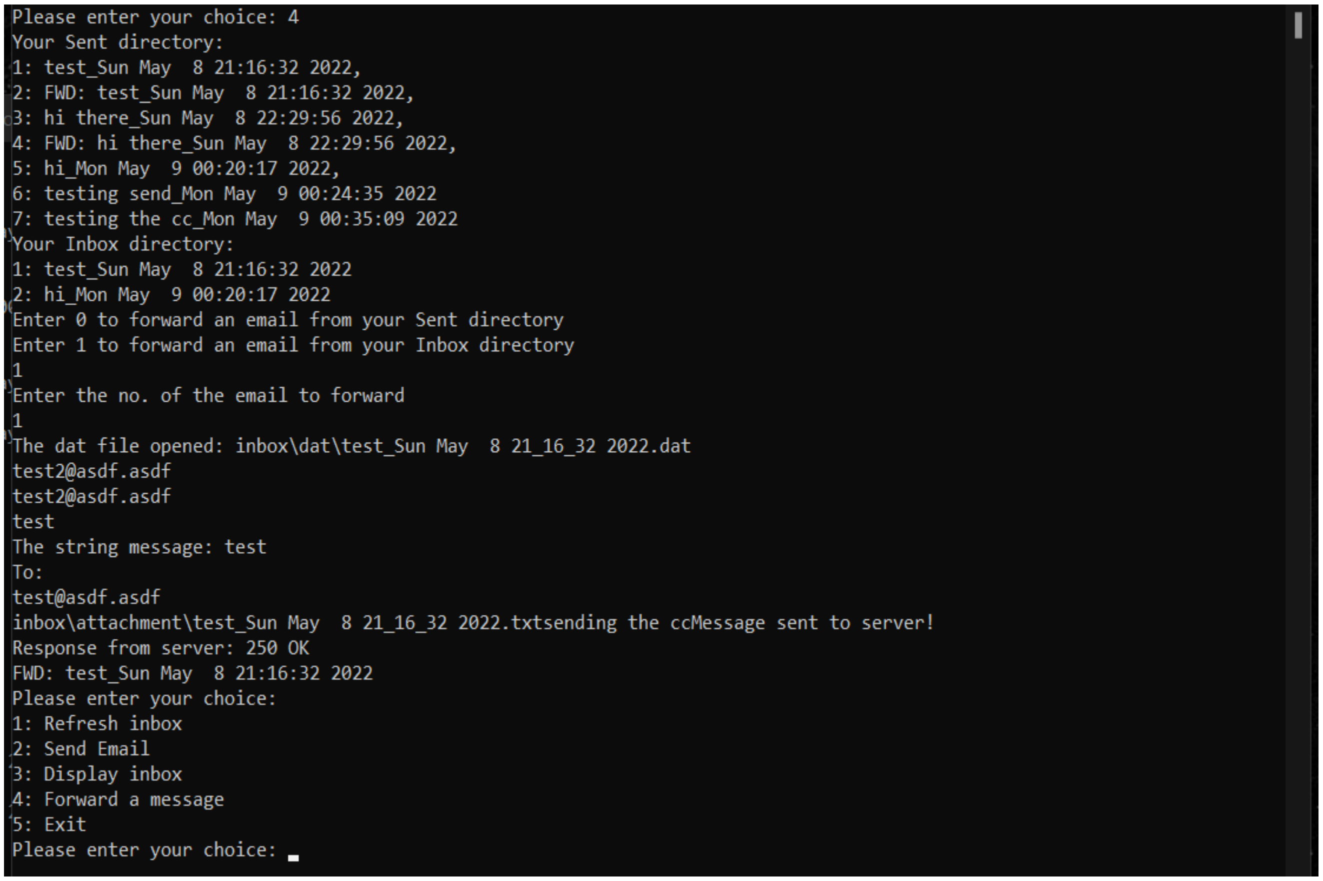


```
Please enter your choice: 2
You have chosen to write a mail
Creating New Email.

To: test@asdf.asdf
From: test2@asdf.asdf
Subject: testing cc
Body: sending a message to multiple receivers using cc

Enter CC emails you want to include (press 'Enter' if you don't have any or you are done and after each mail):
asdf@asdf.asdf
ak@ak.ak
test2@asdf.asdf

asdf@asdf.asdf,ak@ak.ak,test2@asdf.asdf,
Do you want to attach a file? (y/n): n
Message sent to server!
Response from server: 250 OK
testing cc_Mon May  9 11:05:15 2022
```

Figure 10. Successful sending of messages with a couple of cc mail addresses

Forwarding Email

To implement the forwarding feature, first, the client is prompted to enter which email they want to forward (if sent is included, add sen here). The emails are listed for the client to choose from, in a format ‘emailid : email name’. The listing of the emails is done by making use of a map that holds the email id and the email as a key-value pair. After the client chooses the id of the email they want to forward, the corresponding mail name is used to look for the email in the inbox directory. When forwarding an email, the user is prompted to enter the email address of the client they want the email forwarded. This new “To” address goes through the validation before the process proceeds to look for the email to be forwarded. Next, when the file containing the email is located, the email is forwarded by reading the contents of the file and sending.

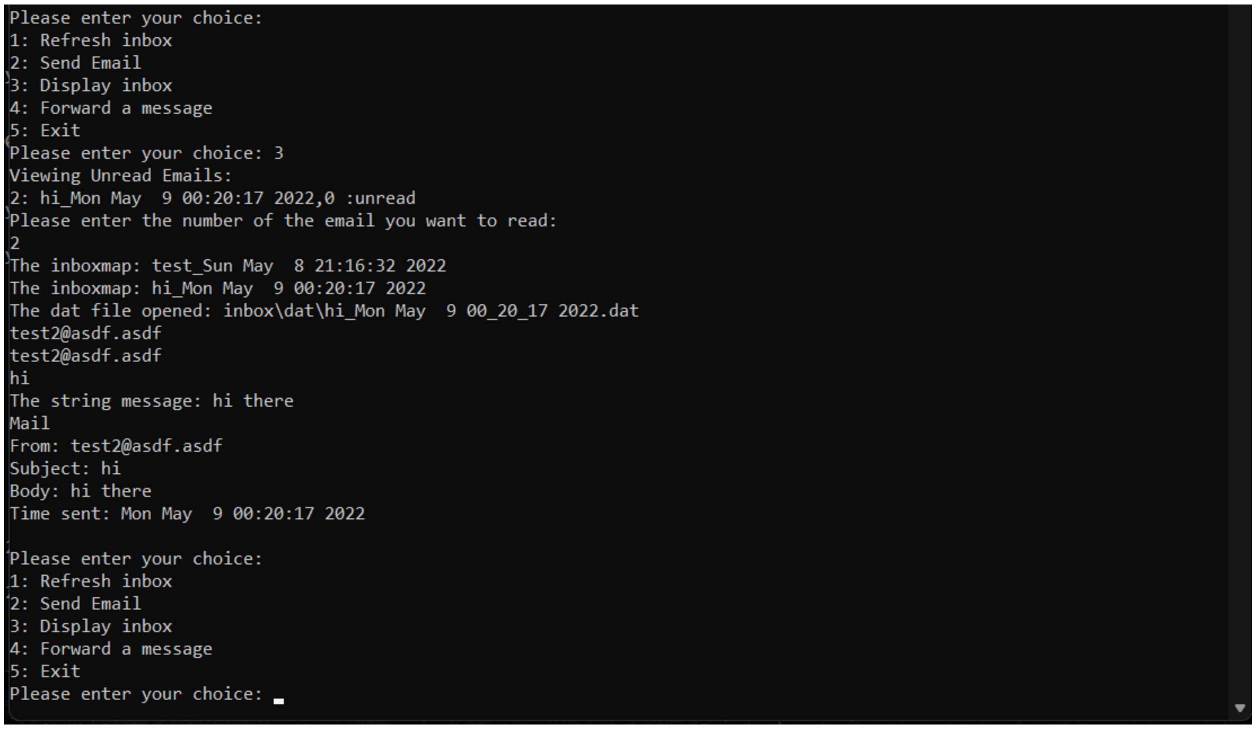


```
Please enter your choice: 4
Your Sent directory:
1: test_Sun May  8 21:16:32 2022,
2: FWD: test_Sun May  8 21:16:32 2022,
3: hi there_Sun May  8 22:29:56 2022,
4: FWD: hi there_Sun May  8 22:29:56 2022,
5: hi_Mon May  9 00:20:17 2022,
6: testing send_Mon May  9 00:24:35 2022
7: testing the cc_Mon May  9 00:35:09 2022
Your Inbox directory:
1: test_Sun May  8 21:16:32 2022
2: hi_Mon May  9 00:20:17 2022
Enter 0 to forward an email from your Sent directory
Enter 1 to forward an email from your Inbox directory
1
Enter the no. of the email to forward
1
The dat file opened: inbox\dat\test_Sun May  8 21_16_32 2022.dat
test2@asdf.asdf
test2@asdf.asdf
test
The string message: test
To:
test@asdf.asdf
inbox\attachment\test_Sun May  8 21_16_32 2022.txtsending the ccMessage sent to server!
Response from server: 250 OK
FWD: test_Sun May  8 21:16:32 2022
Please enter your choice:
1: Refresh inbox
2: Send Email
3: Display inbox
4: Forward a message
5: Exit
Please enter your choice:
```

Figure 11. Successful Forwarding of an Email

Mark as read

One of the options given to the client is to read a mail and after the message is read it is automatically marked as read. This feature is done with the help of the log file in the client inbox folder and the inbox map in the client side. Once the client chooses to read a mail, they are provided with a display of all the unread mail and they are then prompted to enter the email-ID (used as a serial number to identify the email) of the provided unread mails. Once the client choses a valid mail number from the displayed list the program displays the mail and automatically marks the mail in the inbox map which then updates the inbox log file.



```
Please enter your choice:
1: Refresh inbox
2: Send Email
3: Display inbox
4: Forward a message
5: Exit
Please enter your choice: 3
Viewing Unread Emails:
2: hi_Mon May  9 00:20:17 2022,0 :unread
Please enter the number of the email you want to read:
2
The inboxmap: test_Sun May  8 21:16:32 2022
The inboxmap: hi_Mon May  9 00:20:17 2022
The dat file opened: inbox\dat\hi_Mon May  9 00_20_17 2022.dat
test2@asdf.asdf
test2@asdf.asdf
hi
The string message: hi there
Mail
From: test2@asdf.asdf
Subject: hi
Body: hi there
Time sent: Mon May  9 00:20:17 2022

Please enter your choice:
1: Refresh inbox
2: Send Email
3: Display inbox
4: Forward a message
5: Exit
Please enter your choice:
```

Figure 12. Successful mark as read operation

CONCLUSION

Upon the completion of this project, the knowledge we gained in the previous assignments with regards to the basics of network programming was extended. As we implemented the SMTP functionalities more extensively, adding all the fundamental features, we understood socket programming and protocol, namely the TCP/IP internet family environment and its application programming interface (API), as well as a little bit of multi-threading. We also were able to understand how to use the TCP as a delivery service to carry the packets or messages upon mail transfer between multiple clients.

Code:

Server Header

```
#ifndef SER_TCP_H
#define SER_TCP_H

#define HOSTNAME_LENGTH 20
#define RESP_LENGTH 40
#define FILENAME_LENGTH 20
```

```
#define REQUEST_PORT 5001
#define BUFFER_LENGTH 10
#define MAXPENDING 10
#define MSGHDRSIZE 148 //Message Header Size (adjust this based on any changes)
#define SUBJECTSIZE 72
#define MAXCLIENTS 2
#define MappingFile "mappingfile.csv"

#include <map>
//definitions
using namespace std;
typedef struct {
	//header

	char to[HOSTNAME_LENGTH];
	char from[HOSTNAME_LENGTH];
	char subject[SUBJECTSIZE];
	int timestamp;
	char filename[FILENAME_LENGTH];//can hold the whole filename or only the extension
	int datalength; //holds the size of the message
	int attachment; //holds the size of the attchment
	int cc; //holds the size of the cc emails
	//message body
	std::string body;
	std::string ccmail;
	//attachment file name

}MESSAGE;

//requesting for stuff

typedef struct
{
	char hostname[HOSTNAME_LENGTH];
	char request[RESP_LENGTH];
	int updatesize; //holds the size of the inbox log
} REQUEST;  //request

//response message (confirmation and stuff)

typedef struct
{
	char response[RESP_LENGTH]; //holds the response phrase or type
	int size; //can be used for anything

} RESPONSE;
```

```
class TcpServer
{
        int serverSock, clientSock;     /* Socket descriptor for server and client*/
        struct sockaddr_in ClientAddr; /* Client address */
        struct sockaddr_in ServerAddr; /* Server address */
        unsigned short ServerPort;     /* Server port */
        int clientLen;            /* Length of Server address data structure */
        char servername[HOSTNAME_LENGTH];

public:
        TcpServer();
        ~TcpServer();
        void start();
};


class TcpThread :public Thread
{

        int cs;
public:

        TcpThread(int clientsocket) :cs(clientsocket)
        {}
        virtual void run();
        int msg_send(int sock, MESSAGE* msg_ptr);
        int request_send(int sock, REQUEST* msg_ptr);
        int response_send(int sock, RESPONSE* msg_ptr);
        int attach_send(int sock, char* filename, int size); //reading the file while sending it

        int msg_recv(int sock, MESSAGE* msg_ptr);
        int request_recv(int sock, REQUEST* msg_ptr);
        int response_recv(int sock, RESPONSE* msg_ptr);
        int attach_recv(int sock, int size, char* clientname); //try saving the file after receiving it

        bool findReceiver(std::string& receiverfound, char email_addres[]);
        bool isValid(char[]);

        unsigned long ResolveName(char name[]);
        static void err_sys(const char* fmt, ...);



        int getcode(REQUEST);//translates the request to number
        bool getdirs(char* clientmail, std::string& inbox, std::string& sent);//checks if the directories are
there and returns the directory paths

        bool updatelogfile(MESSAGE message, std::string dir, int stat); //updates the log file in the sent or
received depending on the dir

        int ReceiveEmail(MESSAGE* msg_ptr, int sock, RESPONSE* resp); //does all the receiving and
saving to file
```

```
        int mappedReceiver(std::string& clientName, char mailaddress[]);

        int UpdateClient(int sock, char* clientmail, int size);

        int Readlogfile(map<string, string>* fileholder, char* inbox);

        int Receivelogfile(int sock, char* file, int size);

        int Readdatfile(MESSAGE* msg, char* clientmail, char* filename);

        int SendEmail(MESSAGE* msg_ptr, int sock);
        int checkDir(char name[]);

        int saveemailtofile(MESSAGE* msg_ptr, char* directory);

        void conditionString(string* str);

};

#endif
```

Server.cpp code

```
#pragma once
#pragma comment (lib, "Ws2_32.lib")
#define _CRT_SECURE_NO_WARNINGS 1
#define _WINSOCK_DEPRECATED_NO_WARNINGS 1

#include <winsock2.h>
#include <iostream>
#include <windows.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <process.h>
#include <ws2tcpip.h>
#include "Thread.h"
#include "server.h"
#include <map>
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include <queue>
#include <algorithm>
#include <regex>
```

```
#include <queue>

//int sockets[2];
using namespace std;
int k = 0;

//socket number


TcpServer::TcpServer()
{
	WSADATA wsadata;
	if (WSAStartup(0x0202, &wsadata) != 0)
		TcpThread::err_sys("Starting WSAStartup() error\n");

	//Display name of local host
	if (gethostname(servername, HOSTNAME_LENGTH) != 0) //get the hostname
		TcpThread::err_sys("Get the host name error,exit");

	printf("Server: %s waiting to be contacted for mail transfer...\n", servername);


	//Create the server socket
	if ((serverSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
		TcpThread::err_sys("Create socket error,exit");



	ServerPort = REQUEST_PORT;
	memset(&ServerAddr, 0, sizeof(ServerAddr));      /* Zero out structure */
	ServerAddr.sin_family = AF_INET;                 /* Internet address family */
	ServerAddr.sin_addr.s_addr = htonl(INADDR_ANY);  /* Any incoming interface */
	ServerAddr.sin_port = htons(ServerPort);         /* Local port */



	//Bind the server socket
	if (bind(serverSock, (struct sockaddr*)&ServerAddr, sizeof(ServerAddr)) < 0)
		TcpThread::err_sys("Bind socket error,exit");

	//Successfull bind, now listen for Server requests.
	if (listen(serverSock, MAXPENDING) < 0)
		TcpThread::err_sys("Listen socket error,exit");
}

TcpServer::~TcpServer()
{
	WSACleanup();
}
```

```
void TcpServer::start()
{
	for (;;) /* Run forever */
	{
		/* Set the size of the result-value parameter */
		clientLen = sizeof(ClientAddr);

		/* Wait for a Server to connect */
		if ((clientSock = accept(serverSock, (struct sockaddr*)&ClientAddr,
			&clientLen)) < 0)
			TcpThread::err_sys("Accept Failed ,exit");


		Thread* pt;
		pt = new TcpThread(clientSock);
		pt->start();
	}
}

void TcpThread::err_sys(const char* fmt, ...)
{
	perror(NULL);
	va_list args;
	va_start(args, fmt);
	fprintf(stderr, "error: ");
	vfprintf(stderr, fmt, args);
	fprintf(stderr, "\n");
	va_end(args);
	exit(1);
}

unsigned long TcpThread::ResolveName(char name[])
{
	struct hostent* host;		/* Structure containing host information */

	if ((host = gethostbyname(name)) == NULL)
		err_sys("gethostbyname() failed");

	/* Return the binary, network byte ordered address */
	return *((unsigned long*)host->h_addr_list[0]);
}

//Thread functions

int TcpThread::msg_send(int sock, MESSAGE* msg_ptr) {
	//send the header
	int n;
	if ((n = send(sock, (char*)msg_ptr, MSGHDRSIZE, 0)) != MSGHDRSIZE) {
		cout << "ERROR: could not send the message header";
```

```
        }

        // send the message body
        int size;
        size = msg_ptr->datalength;
        char* buffer = new char[BUFFER_LENGTH];
        //hold the string in the buffer
        if (size < BUFFER_LENGTH) {

                msg_ptr->body.copy(buffer, size, 0);
                if ((n = send(sock, (char*)buffer, size, 0)) != (size)) {
                        return n;
                }

        }
        else {
                int count = size;
                while (count > BUFFER_LENGTH) {
                        msg_ptr->body.copy(buffer, BUFFER_LENGTH, size - count);
                        cout << "Inside the while loop" << endl;
                        if ((n = send(sock, (char*)buffer, BUFFER_LENGTH, 0)) !=
(BUFFER_LENGTH)) {
                                return n;
                        }
                        count -= (BUFFER_LENGTH);
                }

                cout << buffer;
                msg_ptr->body.copy(buffer, count, size - count);
                if ((n = send(sock, (char*)buffer, count, 0)) != (count)) {
                        return n;
                }
                n = size;
        }

        return (n);
}


int TcpThread::request_send(int sock, REQUEST* respp) {

        int n;
        if ((n = send(sock, (char*)respp, sizeof(REQUEST), 0)) != (sizeof(REQUEST)))
                cout << "ERROR: sending the request";
        return n;

}

int TcpThread::response_send(int sock, RESPONSE* respp) {
	int n;
	if ((n = send(sock, (char*)respp, sizeof(RESPONSE), 0)) != (sizeof(RESPONSE)))
		cout << "ERROR: sending the response";
	return n;
}


int TcpThread::msg_recv(int sock, MESSAGE* msg_ptr) {
	//start with the header
	int rbytes, n, m,count = 0;

	memset(msg_ptr, 0, MSGHDRSIZE);
	for (rbytes = 0; rbytes < MSGHDRSIZE; rbytes += n) {
		if ((n = recv(sock, (char*)msg_ptr + rbytes, MSGHDRSIZE, 0)) <= 0) {
		cout << "ERROR: receiveing the header" << endl;
		cout << n<<endl;
		}
		count += n;

	}

	//receive the message body
	cout << msg_ptr->datalength<<", The size of the string";
	int size = msg_ptr->datalength;
	count = 0;
	msg_ptr->body = "";
	char* buffer = new char[BUFFER_LENGTH];
	if (size < BUFFER_LENGTH - 1) {

		for (rbytes = 0; rbytes < size; rbytes += n) {
			if ((n = recv(sock, (char*)buffer + rbytes, size, 0)) <= 0) {
				cout << "The n: " << n << endl;
				err_sys("Recv BODY Error");
			}
			count += n;
		}
		buffer[size] = '\0';
		msg_ptr->body = buffer;
		m = count;
	}

	else {

		int counter = size;
		while (counter > BUFFER_LENGTH - 1) {

			for (rbytes = 0; rbytes < BUFFER_LENGTH - 1; rbytes += n) {
				std::cout << "here!!";
				for (rbytes = 0; rbytes < BUFFER_LENGTH-1; rbytes += n) {
```

```
                                        if ((n = recv(sock, (char*)buffer + rbytes,
BUFFER_LENGTH-1, 0)) <= 0)
                                                err_sys("Recv BODY inside Error");
                                }

                                counter -= (BUFFER_LENGTH-1);

                        }
                        buffer[BUFFER_LENGTH - 1] = '\0';
                        msg_ptr->body += buffer;
                }

                for (rbytes = 0; rbytes < counter; rbytes += n) {
                        if ((n = recv(sock, (char*)buffer + rbytes, counter, 0)) <= 0)
                                return n;
                        }

                        cout << "outside" << endl;
                }
                buffer[counter] = '\0';
                msg_ptr->body += buffer;


                m = size;


        }


        ///receiving the cc

        size = msg_ptr->cc;
        count = 0;

        if (size < BUFFER_LENGTH - 1) {

                for (rbytes = 0; rbytes < size; rbytes += n) {
                        if ((n = recv(sock, (char*)buffer + rbytes, size, 0)) <= 0) {
                                cout << "The n: " << n << endl;
                                err_sys("Recv BODY Error");
                        }
                        count += n;
                }
                buffer[size] = '\0';
                msg_ptr->ccmail = buffer;
                //buffer[size] = '\0';
        }

        else {
                int counter = size;
                while (counter > BUFFER_LENGTH - 1) {

                        for (rbytes = 0; rbytes < BUFFER_LENGTH - 1; rbytes += n) {
                                std::cout << "here!!";
                                for (rbytes = 0; rbytes < BUFFER_LENGTH - 1; rbytes += n) {
                                        if ((n = recv(sock, (char*)buffer + rbytes,
BUFFER_LENGTH - 1, 0)) <= 0)
                                                err_sys("Recv BODY inside Error");
                                }

                                counter -= (BUFFER_LENGTH - 1);

                        }
                        buffer[BUFFER_LENGTH - 1] = '\0';
                        msg_ptr->ccmail += buffer;
                }

                for (rbytes = 0; rbytes < counter; rbytes += n) {
                        if ((n = recv(sock, (char*)buffer + rbytes, counter, 0)) <= 0)
                                return n;
                        }

                        cout << "outside" << endl;
                }
                buffer[counter] = '\0';
                msg_ptr->ccmail += buffer;


                count = size;


        }
        cout << msg_ptr->ccmail;


        return m; //returning the body of the message (which means that the header has been
received successfully)

}


int TcpThread::request_recv(int sock, REQUEST* msg_ptr) {
        int rbytes, n, count = 0;

        for (rbytes = 0; rbytes < sizeof(REQUEST); rbytes += n) {
                if ((n = recv(sock, (char*)msg_ptr + rbytes, sizeof(REQUEST), 0)) <= 0)
                        cout << "ERROR: receiveing the request";
```

```
	}
	cout << msg_ptr->hostname << " the hostname" << endl;
	return rbytes;
}

int TcpThread::response_recv(int sock, RESPONSE* msg_ptr) {
	int rbytes, n, count = 0;

	for (rbytes = 0; rbytes < sizeof(RESPONSE); rbytes += n) {
		if ((n = recv(sock, (char*)msg_ptr + rbytes, sizeof(RESPONSE), 0)) <= 0)
			cout << "ERROR: receiveing the response";
		count += n;

	}
	return count;
}

bool TcpThread::findReceiver(string& receiverfound, char email_addres[]) {
	fstream myFile;
	bool found = false;
	map<string, string> addresses;
	string email_address(email_addres);//repurposing this as the client name
	myFile.open(MappingFile, ios::in);
	if (myFile.is_open()) {
		string clientName;
		string clientEmail;
		while (!myFile.eof()) {
			getline(myFile, clientName, ',') && getline(myFile, clientEmail);
			addresses.insert(pair<string, string>(clientName, clientEmail));
		}
		myFile.close();
		if (addresses[email_address].length() != 0) {
			found = true;
			receiverfound = addresses[email_address];
		}
		return found;
	}
	cout << "Could not open file!";
	//could not open file message and return
	return found;
}

bool TcpThread::isValid(char email[]) {
	string email_(email);
	if (regex_match(email_, regex("(\\w+)(\\.|_)?(\\w*)@(\\w+)(\\.(\\w+))+")))
		return 1;

	return 0;
}
```

```
int TcpThread::attach_send(int sock, char* filename, int size) { //reading the file while sending
it

	int n;
	int count = 0;
	//open the file reading
	//the filename should hold the file name and place (the whole directory) else you can
pass the whole message to the function
	ifstream file(filename, ios::binary);
	if (!file) {
		cout << "Error opening file";
		return 0;
	}


	if (!file) {
		cout << "ERROR: could not open the saved file!" << endl;
		return 0;
	}
	//hold the string in the buffer
	char buff[BUFFER_LENGTH];


	if (size < BUFFER_LENGTH) {

		char* buf = (char*)malloc(size);
		memset(buf, 0, size);
		if (!buf)
		{
			file.close();
			cout << "ERROR: Creating a dynamic char pointer error" << endl;
			return 0;
		}
		if (!file.read(buf, size))
		{
			file.close();
			cout << "ERROR: Reading into buffer error 1" << endl;
			return 0;
		}

		if ((n = send(sock, buf, size, 0)) != (size)) {
			cout << "size n n here";
			cout << size << endl;
			cout << n << endl;
			cout << "ERROR: Sending the file error" << endl;
			return n;
		}
		free(buf);
```

```
        }
        else {
                count = size;
                while (count > BUFFER_LENGTH) {
                        //read them on buffer
                        memset(buff, 0, BUFFER_LENGTH);
                        if (!file.read(buff, BUFFER_LENGTH))
                        {
                                file.close();
                                cout << "ERROR: Reading into buffer error" << endl;
                                return 0;
                        }
                        if ((n = send(sock, buff, BUFFER_LENGTH, 0)) !=
(BUFFER_LENGTH)) {
                                cout << "ERROR: Sending the file error (outside)" << endl;
                        }

                        count -= BUFFER_LENGTH;
                }
                memset(buff, 0, BUFFER_LENGTH);
                if (!file.read(buff, count))
                {
                        file.close();
                        cout << "ERROR: Reading into buffer error" << endl;
                        return 0;
                }
                if ((n = send(sock, buff, count, 0)) != (count)) {
                        cout << "ERROR: Sending the file error (outside)" << endl;
                }

                count -= count;

        }
        return size-count;
}

int TcpThread::attach_recv(int sock,int size, char* file) { //return the file in file
        //better to put it in a buffer
        cout << "The attach size in attach_rcv: " << size << endl;
        int rbytes, n, count = 0;
        memset(file, 0, size);

        for (rbytes = 0; rbytes < size; rbytes += n) {
                n = recv(sock, (char*)file + rbytes, size, 0);
                printf("%d\n", n);
                count += n;
        }
```

```
	return count;




}

int TcpThread::saveemailtofile(MESSAGE* msg_ptr, char* directory) {//directory comes from
the run method (would be better if we change it to string)
#pragma warning(suppress : 4996)

	string savedfile;
	string saveddatfile;
	ofstream file;

	//get the proper time format
	time_t time = (time_t)msg_ptr->timestamp;
	savedfile = directory;
	//construct the subject_time filename
	savedfile += "\\";
	string test = savedfile + "dat";
	checkDir((char*)test.c_str());
	saveddatfile = savedfile +"dat"+'\\' + msg_ptr->subject + '_' + asctime(localtime(&time))
+ ".dat";
	//using from_time for the dat file name (could be changed)
	savedfile += msg_ptr->subject;
	savedfile += "_";
	savedfile += (string)asctime(localtime(&time));//change this to proper time format
	savedfile += ".txt";
	replace(savedfile.begin(), savedfile.end(), ':', '_');
	replace(saveddatfile.begin(), saveddatfile.end(), ':', '_');
	savedfile.erase(std::remove(savedfile.begin(), savedfile.end(), '\n'), savedfile.end());
	saveddatfile.erase(std::remove(saveddatfile.begin(), saveddatfile.end(), '\n'),
saveddatfile.end());


	file.open(savedfile.c_str());

	//write all the info to the file(from,to,subject,cc,body,timestamp)

	file << "From: " << msg_ptr->from << endl;
	file << "To: " << msg_ptr->to << endl;
	file << "cc: " << msg_ptr->ccmail << endl;
	file << "Subject: " << msg_ptr->subject << endl;
	file << "Body: " << endl << msg_ptr->body << endl;
	file << "Time: " << asctime(localtime(&time)) <<endl;
	file.close();

	//save the dat file

	FILE* outfile;
	outfile = fopen(saveddatfile.c_str(), "w");
	cout << "The dat file: " << saveddatfile << endl;
	if (outfile == NULL)
	{
		fprintf(stderr, "\nError opened file\n");
		exit(1);
	}

	// write struct to file

	fwrite(msg_ptr, MSGHDRSIZE, 1, outfile);
	fwrite(msg_ptr->body.c_str(), msg_ptr->datalength, 1, outfile);

	//add the rest of the message

	fclose(outfile);
}

int saveattachtofile(char* filename, char* file, int filesize) {
	//create a file
#pragma warning(suppress : 4996)
	FILE* attachment;
	attachment = fopen(filename, "wb");
	if (!attachment)
	{

		free(file);
		cout << "ERROR: Could not open file" << endl;
		return 0;
	}

	// Write the entire buffer to file
	if (!fwrite(file, sizeof(char), filesize, attachment))
	{
		free(file);
		fclose(attachment);
		cout << "ERROR:saving the file to local dir" << endl;
		return 0;
	}
	fclose(attachment);
	return 1;
}

int TcpThread::getcode(REQUEST req) {
	/*
	1. receive (update)
	2. client want to send
	*/
	if (strcmp(req.request, "update") == 0)
		return 1;
	else if (strcmp(req.request, "send") == 0)
		return 2;
	else if (strcmp(req.request, "disconnect") == 0)
		return 3;
	return 0;

}

bool TcpThread::getdirs(char* clientmail,string& inbox, string& sent) {
	//construct sent and inbox directory

	//inbox directory
	inbox = clientmail;
	checkDir(clientmail);
	inbox += "\\";
	inbox += "inbox";
	checkDir((char*) inbox.c_str());
	inbox += "\\";
	//create the inbox directory if its not there
	sent = clientmail;
	sent += "\\";
	sent += "sent";
	checkDir((char*) sent.c_str());
	sent += "\\";

	//attachment dir
	string attachmentdir = inbox+"\\"+"attachment";
	checkDir((char*)attachmentdir.c_str());
	//create the sent directory if its not there
	return 1;

}


int TcpThread::checkDir(char name[]) {
	struct stat buffer;
	if (stat(name, &buffer) != 0) {
		wchar_t wtext[40];
		mbstowcs(wtext, name, strlen(name) + 1);
		LPWSTR ptr = wtext;
		CreateDirectory(ptr, NULL);
		return 0;
	}
	else {
		return -1;

	}
}

int TcpThread::mappedReceiver(string& clientName, char mailaddress[]) {
	//This function takes the given receiver email address and returns the client name and
true/false if the client is in the mapping

	char client_name[20];
	char client_address[20];
	string line, word;
	string fname = "mappingfile.csv"; //using csv file might be helpfull if we want to
manage is manually
	//open the mapping file
	fstream file(fname, ios::in);
	if (file.is_open()) {
		while (getline(file, line))
		{//reading the whole line in 'line' and then separating them to deal with them
addresses and names separately
			stringstream str(line);//not the most efficient way but could be improved
			getline(str, word, ',');
			strcpy(client_name, word.c_str());
			getline(str, word, ',');
			strcpy(client_address, word.c_str());
			if (strcmp(client_address, mailaddress) == 0) {
				clientName = client_name;
				file.close();
				return 1;
			}
		}
		file.close();
		return 0;
	}

}

bool TcpThread::updatelogfile(MESSAGE message, string inboxdir, int stat) {//stat isnt used
for anything
	string directory;
	directory = inboxdir;
	directory += "\\";
	directory += "logfile.csv";
	fstream file;
	file.open(directory, ios::app);
	if (!file.is_open())
		file.open(directory, ios::out);
	time_t time = (time_t)message.timestamp;
	string savedname;
	cout << "Subject: " << message.subject << endl;
	savedname = message.subject;
	savedname += '_';
	savedname += (string)asctime(localtime(&time));
	savedname += ',';
	conditionString(&savedname);
	if (file.is_open()) {


		file << savedname;
		return 1;

	}
	return 0;

}

//receiveing the mail and saving it to all the directories

int TcpThread::ReceiveEmail(MESSAGE* msg_ptr, int sock, RESPONSE* resp) { //returns 0
is something is wrong else returns 1 (might change this to include wrong receiver address)
	int n;
	char* file;
	strcpy(resp->response, "250 OK");
	msg_ptr = new MESSAGE();
	string inbox, sent;
	if ((n = msg_recv(sock, msg_ptr)) != msg_ptr->datalength) {
		strcpy(resp->response, "501 ERROR");
		cout << "ERROR: msg_recv error" << endl;
	}
	cout << "The file size: " << msg_ptr->attachment << endl;
	//check receiver address and stuff here
	//check if they are valid
	if (!isValid(msg_ptr->from) || !isValid(msg_ptr->to)) {
		strcpy(resp->response, "550 ERROR");
		cout << "ERROR: invalid email formats" << endl;
	}
	//check if the receiver exists (the sender should be checked in the run method)
	string holder;
	if (!mappedReceiver(holder, msg_ptr->to)) {
		cout << "ERROR: The given receiver address " << msg_ptr->to << " is not
registered." << endl;
		strcpy(resp->response, "550 ERROR");

	}
```

```
//get all the cc mails
queue<string> ccmails;
ccmails.push(msg_ptr->to);
string line;
stringstream str(msg_ptr->ccmail);
if (msg_ptr->cc != 0) {
        while (getline(str, line, ',')) {
                if (isValid((char*)line.c_str())) {
                        ccmails.push(line);
                }
        }
}
queue<string> filequeue(ccmails);
while (!filequeue.empty()) {
        cout << filequeue.front() << endl;
        filequeue.pop();

}


//check if we have attachment and do stuff accordingly
if (msg_ptr->attachment != 0) {
        file = (char*)malloc(msg_ptr->attachment);
        //set the file name
        string filename, temp;

        if ((n = attach_recv(sock, msg_ptr->attachment, file)) != msg_ptr->attachment) {
                strcpy(resp->response, "501 ERROR");
                cout << "ERROR: attach_recv error" << endl;
                return 0;
        }
        queue<string> filequeue(ccmails);
        while (!filequeue.empty()) {
                string holder = filequeue.front();
                filequeue.pop();
                getdirs((char*)holder.c_str(), filename, temp);

                time_t time = (time_t)msg_ptr->timestamp;
                filename += "\\";
                filename += "attachment";
                filename += "\\";
                filename += msg_ptr->subject;
                filename += "_";
                filename += (string)asctime(localtime(&time));
                filename += ".";
                filename += msg_ptr->filename;

                replace(filename.begin(), filename.end(), ':', '_');
```

```
				filename.erase(std::remove(filename.begin(), filename.end(), '\n'),
filename.end());


				saveattachtofile((char*)filename.c_str(), file, msg_ptr->attachment);
			}

	}

	if (strcmp(resp->response, "550 ERROR") == 0 || strcmp(resp->response, "501
ERROR") == 0) {
		return 0;
	}
	//save to the sender and receiver


	//receiver
	while (!ccmails.empty()) {
		inbox = "";
		cout << "The receiver inbox: " << ccmails.front() << endl;
		if (!getdirs((char*)ccmails.front().c_str(), inbox, sent)) {
			cout << "ERROR: Can not generate inbox and sent directory" << endl;
			return 0;
		}
		saveemailtofile(msg_ptr, (char*)inbox.c_str());

		ccmails.pop();
		//update the log form (only on the inbox might be much easier)
		updatelogfile(*msg_ptr, inbox, 0);
	}
	//sender
	if (!getdirs(msg_ptr->from, inbox, sent)) {
		cout << "ERROR: Can not generate inbox and sent directory" << endl;
		return 0;
	}

	saveemailtofile(msg_ptr, (char*)sent.c_str());

	return 1;

}

int TcpThread::Receivelogfile(int sock, char* file, int size) {
	int rbytes, n, count = 0;
	memset(file, 0, size);
	if (size < BUFFER_LENGTH) {
		for (rbytes = 0; rbytes < size; rbytes += n) {
			if ((n = recv(sock, (char*)file + rbytes, size, 0)) <= 0) {
				err_sys("Recv BODY Error");
				}
				count += n;
			}

		}

		else {
			int counter = size;
			while (counter > BUFFER_LENGTH) {
				for (rbytes = 0; rbytes < BUFFER_LENGTH; rbytes += n) {
					std::cout << "here!!";
					for (rbytes = 0; rbytes < BUFFER_LENGTH; rbytes += n) {
						if ((n = recv(sock, (char*)file + (size - counter) + rbytes,
BUFFER_LENGTH, 0)) <= 0)
							err_sys("Recv BODY inside Error");
					}
					counter -= (BUFFER_LENGTH);

				}
			}
			for (rbytes = 0; rbytes < counter; rbytes += n) {
				if ((n = recv(sock, (char*)file + (size - counter) + rbytes, counter, 0)) <=
0)
					return n;
				//err_sys("Recv BODY Error");
			}
			count = size;
		}
		file[size] = '\0';
		cout << "the received file: " << file << endl;
		return count; //returning the body of the message (which means that the header has
been received successfully)

}

void TcpThread::conditionString(string* str) {
		str->erase(std::remove(str->begin(), str->end(), '\n'), str->end());
		str->erase(std::remove(str->begin(), str->end(), ','), str->end());
}

int TcpThread::Readlogfile(map<string,string>* fileholder,char* inbox) { //inbox if the whole
directory till inbox

		char from[20];
		char timestamp[20];
		string line, word;
		string fname = inbox;
		//open the log file
		cout << fname;
```

```
	fstream file(fname, ios::in);
	if (file.is_open()) {
		while (getline(file, line,','))
		{//reading the whole line in 'line' and then separating them to deal with them
addresses and names separately

			//stringstream str(line);//not the most efficient way but could be
improved
			fileholder->insert(pair<string, string>(line, "mail"));
			cout << "The log file in server: " << line << endl;
		}
		file.close();
		return 1;
	}
	cout << "WARNING: no logfile" << endl;

	return 1;
}

int TcpThread::Readdatfile(MESSAGE* msg, char* clientmail,char* filename) {
	string filedir;
	filedir = clientmail;
	filedir += "\\";
	filedir += "dat";
	filedir += "\\";
	cout << "Filename: " << filename << endl;
	filedir += filename;
	filedir += ".dat";

	filedir.erase(std::remove(filedir.begin(), filedir.end(), '\n'), filedir.end());
	cout << filedir << endl;
	FILE* outfile;
	outfile = fopen(filedir.c_str(), "r");
	cout << "The dat file: " << filedir << endl;
	if (outfile == NULL)
	{
		fprintf(stderr, "\nError opened file\n");
		exit(1);
	}

	// read struct from file
	fread(msg, MSGHDRSIZE, 1, outfile);
	char* buffer = new char[BUFFER_LENGTH];
	int i = msg->datalength;
	msg->body = "";
	while (i > BUFFER_LENGTH-1) {
		fread(buffer, BUFFER_LENGTH-1, 1, outfile);
		buffer[BUFFER_LENGTH - 1] = '\0';
		msg->body += buffer;
```

```
			i -= BUFFER_LENGTH-1;
		}
		fread(buffer, i, 1, outfile);
		buffer[i] = '\0';
		msg->body += buffer;

		cout << "The string message: " << msg->body << endl;
		fclose(outfile);

}

int TcpThread::SendEmail(MESSAGE* msg_ptr, int sock) {
		//send the message
		int n;
		if ((n = msg_send(sock, msg_ptr)) != msg_ptr->datalength) {
			cout << "ERROR: msg_send error" << endl;
			return 0;
		}
		//send the attachment
		if (msg_ptr->attachment != 0) {
			//use attach send to do this
			//use a proper filename
			string filename, temp;
			getdirs(msg_ptr->to, filename, temp);
			time_t time = (time_t)msg_ptr->timestamp;
			filename += '\\';
			filename += "attachment";
			filename += "\\";
			filename += msg_ptr->subject;
			filename += "_";
			filename += (string)asctime(localtime(&time));
			filename += ".";
			filename += msg_ptr->filename;
			replace(filename.begin(), filename.end(), ':', '_');
			filename.erase(std::remove(filename.begin(), filename.end(), '\n'),
filename.end());
			cout << "The file name: " << filename << endl;
			if ((n = attach_send(sock,(char*) filename.c_str(), msg_ptr->attachment)) !=
msg_ptr->attachment) {
				cout << "Sent file size: " << n;
				cout << "ERROR: attach_send error" << endl;
				return 0;
			}
		}
		return 1;

}

int TcpThread::UpdateClient(int sock,char* clientmail, int size) { //returns 0 if error else 1
```

```
//receives the client logfile and sends all what needs sending
map<string, string> logfile;
map<string, string> receivedlog;

RESPONSE resp;
//receive the client logfile
int n;
cout << "inside update client receiveing the logfile of size: " << size << endl;
char* file = (char*)malloc(size);
if ((n = Receivelogfile(sock, file, size)) != size) {
        cout << "ERROR: Receivelogfile error" << endl;
        return 0;
}

stringstream str(file);
string mail;


//read and compare the client logfile

//inbox folder
string inbox;
string sent;
getdirs(clientmail, inbox, sent);
inbox += "\\";
inbox += "logfile.csv";
if (Readlogfile(&logfile, (char*)inbox.c_str()) != 1) {
        cout << "ERROR: Readlogfile error" << endl;
        return 0;
}
getdirs(clientmail, inbox, sent);
//compare the received file with the local one
if (size != 0) {
        while (getline(str, mail, ',')) {
                cout << "THE mail list from the client: " << mail << endl;
                logfile.erase(mail);
        }
}

//send response with the number of mails to send
resp.size = logfile.size();

if ((n = response_send(sock, &resp)) != sizeof(RESPONSE)) {
        cout << "ERROR: response_send error (update client)" << endl;
        return 0;
}

if (!logfile.empty()) {
        for (map<string, string>::iterator it = logfile.begin(); it != logfile.end(); ++it) {
```

```
                        mail = it->first;
                        replace(mail.begin(), mail.end(), ':', '_');

                        //sendmail
                        MESSAGE* msg;
                        //read the dat file
                        msg = new MESSAGE();


                        if (Readdatfile(msg, (char*)inbox.c_str(), (char*)mail.c_str()) == 0) {
                                cout << "ERROR: Readdatfile error" << endl;
                                return 0;
                        }
                        cout << msg->datalength << endl;
                        //send message and the file
                        if (SendEmail(msg, sock) == 0) {
                                cout << "ERROR: SendEmail error" << endl;
                                return 0;
                        }

                }

        }
        cout << "done";
        return 1;
}



void TcpThread::run() //cs: Server socket
{

        //MESSAGE *msg;

        REQUEST req;
        int n; //for getting the sent or received size
        int k; //for holding the request or response code
        string clientmail; //for holding the client email (which we are getting from the client
name
        string inbox; //inbox directory
        string sent; //sent directory
        int status = 0; //using this to avoid checking for the directory everytime
        bool stat = true;

        while (stat) {
                MESSAGE* msg = new MESSAGE();
                RESPONSE* resp = new RESPONSE();
                //receive client request and update clients inbox
                cout << "Starting the while loop" << endl;
```

```
		if ((n = request_recv(cs, &req)) != sizeof(REQUEST)) {
			cout << "ERROR: receiving the  request error" << endl;
			if (n == -1) {
				cout << "ERROR: Client disconnected.\nExiting thread..." << 
endl;
				//terminate the whole thread and close the connection

			}
			stat = false;
			continue;
		}

		//checking if the connected client is registered
		cout << req.hostname << " The hostname " << endl;
		if (!findReceiver(clientmail, req.hostname)) {
			cout << "ERROR: The connected client " << req.hostname << " is not 
registered." << endl;
			cout << "Disconnecting the client " << req.hostname << endl;
			//terminate the whole thread and close the connection
			stat = false;
			continue;
		}


		//get clientmail/inbox and clientmail/sent directories if the client is connecting 
for the first time
		if (status == 0) {
			getdirs((char*)clientmail.c_str(), inbox, sent);
			status = 1;
		}//this is done to check if the directories have to be there beforehand

		//check request and do stuff accordingly
		k = getcode(req);
		switch (k) {
		case 1:
			//update the client inbox
			if (UpdateClient(cs, (char*)clientmail.c_str(), req.updatesize) == 0) {
				cout << "ERROR: UpdateClient error";

				//terminate thread
				stat = false;
			}

			break;
		case 2:
			//receive the mail
			if (ReceiveEmail(msg, cs,resp) == 0) {
				//error
				cout << "ERROR: ReceiveEmail error" << endl;
```

```
                                        stat = false;
                                        //terminate the thread
                              }
                              //send the confirmation
                              if ((n = response_send(cs, resp))!=sizeof(RESPONSE)) {
                                        cout << "ERROR: can not sent the confirmation message to
client " << req.hostname << endl;

                              }
                              cout << "The response to client: " << resp->response << endl;

                              break;
                    case 3:
                              //disconnect the client
                              stat = false;
                              //terminate the thread
                              break;


                    }

          }
          closesocket(cs);

}


int main(void)
{

          TcpServer ts;
          ts.start();

          return 0;
}
```

Thread header:

```
#ifndef THREAD_HPP
#define THREAD_HPP

#include <stdio.h>
#include <stdlib.h>
#include <process.h>

#define        STKSIZE        16536
class Thread{
          public:
                Thread()
                {}
                virtual ~Thread()
                {}

                static void * pthread_callback (void * ptrThis);

                virtual void run () =0 ;
                void  start();
};
#endif
```

Thread.cpp code

```
#include <stdio.h>
#include "Thread.h"

/*
 * This is the callback needed by the Thread class
 */
void * Thread::pthread_callback (void * ptrThis)
{

   if ( ptrThis == NULL )
            return NULL ;
   Thread  * ptr_this =(Thread *)(ptrThis) ;
   ptr_this->run();
   return NULL;
}

void Thread::start ()
{
        int result;
   if(( result = _beginthread((void (*)(void *))Thread::pthread_callback,STKSIZE,this ))<0)
        {
                printf("_beginthread error\n");
                exit(-1);
        }


}
```

Client header

```
#ifndef SER_TCP_H
#define SER_TCP_H

#define HOSTNAME_LENGTH 20
#define RESP_LENGTH 40
#define FILENAME_LENGTH 20
```

```
#define REQUEST_PORT 5001
#define BUFFER_LENGTH 10
#define MAXPENDING 10
#define MSGHDRSIZE 148 //Message Header Size (adjust this based on any changes)
#define SUBJECTSIZE 72
#define MAXCLIENTS 2
#define inboxdir "inbox"
#define sentdir "sent"
#define MAXEMAILS 10 //10 is arbitray


#define MESSAGESIZE 1024
#endif

#include <winsock2.h>
#include <stdio.h>
#include <iostream>
#include <string>
#include <windows.h>
#include <ws2tcpip.h>
#include <sstream>
#include <queue>
#include <filesystem>//consider removing this if we are not using them
#include <fstream>
#include <regex>
#include <map>
// definitions
using namespace std;

typedef struct {
	//header
	char to[HOSTNAME_LENGTH];
	char from[HOSTNAME_LENGTH];
	char subject[SUBJECTSIZE];
	int timestamp;
	char filename[FILENAME_LENGTH];//can hold the whole filename or only the
extension
	int datalength; //holds the size of the message
	int attachment; //holds the size of the attchment
	int cc; //holds the size of the cc emails

	//message body
	std::string body;
	std::string ccmail;
	//attachment file name

}MESSAGE;

//requesting for stuff
enum TYPE { INBOX = 1, SENT };
```

```
typedef struct
{
	char hostname[HOSTNAME_LENGTH];
	char request[RESP_LENGTH];
	int updatesize; //holds the size of the inbox log
} REQUEST;  //request

typedef struct filestruct
{
	filestruct() : mailname(), readstatus(0) {}

	filestruct(std::string mail, int name) : mailname(mail), readstatus(name) {}
	std::string mailname;
	int readstatus; //holds the read status



};  //filestruct

//response message (confirmation and stuff)

typedef struct
{
	char response[RESP_LENGTH]; //holds the response phrase or type
	int size; //can be used for anything

} RESPONSE;




class TcpClient
{
	int sock;			/* Socket descriptor */
	struct sockaddr_in ServAddr; /* server socket address */
	unsigned short ServPort;	/* server port */
	RESPONSE resp;
	MESSAGE* msg;
	REQUEST req;
	map<int, struct filestruct> inboxmap;
	int inboxcount, sentcount;
	map<int, string> sentmap;


	WSADATA wsadata;
public:
	TcpClient();
	void run(int argc, char* argv[]);
	~TcpClient();

	// methods
	int msg_send(int sock, MESSAGE* msg_ptr);
	int request_send(int sock, REQUEST* msg_ptr);
	int response_send(int sock, RESPONSE* msg_ptr);
	int attach_send(int sock, char* filename, int size, MESSAGE* msg_ptr); //reading the
file while sending it

	int msg_recv(int sock, MESSAGE* msg_ptr);
	int request_recv(int sock, REQUEST* msg_ptr);
	int response_recv(int sock, RESPONSE* msg_ptr);
	int attach_recv(int sock, char* filename, int size); //try saving the file after receiving it
	int isValid(char email[]);
	int setupenv();

	int checkDir(char name[]);

	int Updateinbox(int, MESSAGE*);
	int Sendmail(int sock, MESSAGE* msg, bool fromForward, string attachFromFWD);
	int Sendlogfile(int sock, char* file, int size);

	int updatelogfile(MESSAGE message, int stat);


	int Readlogfile(std::string* fileholder);
	int getsize(std::string filename);
	unsigned long ResolveName(char name[]);
	void err_sys(const char* fmt, ...);

	void initiateconnection(char[]);

	int saveattachtofile(char* filename, char* file, int filesize, int dir);
	int saveemailtofile(MESSAGE* msg_ptr, char* directory);
	int getmessageinput(MESSAGE* msg, std::string* filepath);

	int loadmaps();
	int addtomap(string toadd, int stat);
	int markasread(int k);
	int updatelogfiles();
	bool Displayinbox();
	bool Displaysent();
	int Displayinboxmail(int k);
	int readdatfile(MESSAGE* msg, int k, int stat);//stat = 1 if inbox
	void conditionfilename(string* str);
	void conditionString(string* str);

	void DisplayMessage();

	int list_read();
	int list_unread();
	int updateinboxlog();

	int forwardEmail(int sock, MESSAGE* msg_ptr);
	int displayEmail(string dir, string allemails[MAXEMAILS]);


};
```

Client.cpp code

```
#pragma once
#pragma comment (lib, "Ws2_32.lib")
#define _CRT_SECURE_NO_WARNINGS 1
#define _WINSOCK_DEPRECATED_NO_WARNINGS 1

#include "clientTcp.h"
#include <winsock2.h>
#include <stdio.h>
#include <iostream>
#include <string>
#include <windows.h>
#include <ws2tcpip.h>
#include <sstream>
#include <queue>
#include <filesystem>//consider removing this if we are not using them
#include <fstream>
#include <regex>


using namespace std;

TcpClient::~TcpClient()
{
	/* When done uninstall winsock.dll (WSACleanup()) and exit */
	WSACleanup();
}

TcpClient::TcpClient() {
	//setting the counter to zero and loading the maps
	inboxcount = 0;
	sentcount = 0;
	loadmaps();

}

void TcpClient::err_sys(const char* fmt, ...) //from Richard Stevens's source code
{
	perror(NULL);
	va_list args;
	va_start(args, fmt);
	fprintf(stderr, "error: ");
	vfprintf(stderr, fmt, args);
	fprintf(stderr, "\n");
	va_end(args);
	exit(1);
}

void TcpClient::initiateconnection(char inputserverhostname[]) {
	if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) //create the 
socket
		err_sys("Socket Creating Error");
	ServPort = REQUEST_PORT;
	memset(&ServAddr, 0, sizeof(ServAddr));     /* Zero out structure *///clearing the 
ServAddr
	ServAddr.sin_family = AF_INET;             /* Internet address family */ //IPV4
	//inet_pton(AF_INET, "192.168.43.132", &(ServAddr.sin_addr));
	ServAddr.sin_addr.s_addr = ResolveName(inputserverhostname);   /* Server IP 
address */
	ServAddr.sin_port = htons(ServPort); /* Server port *///get the server ip

	if (connect(sock, (struct sockaddr*)&ServAddr, sizeof(ServAddr)) < 0)
		err_sys("Socket Creating Error");
}

unsigned long TcpClient::ResolveName(char name[])
{
	struct hostent* host;           /* Structure containing host information */

	if ((host = gethostbyname(name)) == NULL) {
		printf("Make sure you have entered the right Mail Server name!\n");
		err_sys("gethostbyname() failed");
	}

	/* Return the binary, network byte ordered address */
	return *((unsigned long*)host->h_addr_list[0]);
}



int TcpClient::msg_send(int sock, MESSAGE* msg_ptr) {
	//send the header
	int n, k;
	int m;
	k = sock;
	if ((n = send(k, (char*)msg_ptr, MSGHDRSIZE, 0)) != MSGHDRSIZE) {
		cout << "ERROR: could not send the message header";
	}

	// send the message body
	int size;
	size = msg_ptr->datalength;
	char* buffer = new char[BUFFER_LENGTH];
	if (size < BUFFER_LENGTH) {
		msg_ptr->body.copy(buffer, size, 0);
		if ((n = send(k, (char*)buffer, size, 0)) != (size)) {
```

```
                    cout << n;
                    return n;
            }
            m = n;

    }
    else {
            int count = size;
            while (count > BUFFER_LENGTH) {
                    msg_ptr->body.copy(buffer, BUFFER_LENGTH, size - count);
                    if ((n = send(k, (char*)buffer, BUFFER_LENGTH, 0)) !=
(BUFFER_LENGTH)) {
                            return n;
                    }
                    count -= (BUFFER_LENGTH);
            }
            msg_ptr->body.copy(buffer, count, size - count);
            if ((n = send(k, (char*)buffer, count, 0)) != (count)) {
                    return n;
            }
            m = size;
    }




    ///sending the cc

    size = msg_ptr->cc;
    if (size < BUFFER_LENGTH) {
            msg_ptr->ccmail.copy(buffer, size, 0);
            if ((n = send(k, (char*)buffer, size, 0)) != (size)) {
                    cout << n;
                    return n;
            }

    }
    else {
            int count = size;
            while (count > BUFFER_LENGTH) {
                    msg_ptr->ccmail.copy(buffer, BUFFER_LENGTH, size - count);
                    if ((n = send(k, (char*)buffer, BUFFER_LENGTH, 0)) !=
(BUFFER_LENGTH)) {
                            return n;
                    }
                    count -= (BUFFER_LENGTH);
            }
            msg_ptr->ccmail.copy(buffer, count, size - count);
            if ((n = send(k, (char*)buffer, count, 0)) != (count)) {
                    return n;
            }
```

```
                n = size;
        }

        return (m);
}

int TcpClient::request_send(int sock, REQUEST* msg_ptr) {
        int n;
        if ((n = send(sock, (char*)msg_ptr, sizeof(REQUEST), 0)) != (sizeof(REQUEST)))
                cout << "ERROR: sending the request" << endl;
        return n;
}

int TcpClient::response_send(int sock, RESPONSE* msg_ptr) {
        int rbytes, n, count = 0;

        for (rbytes = 0; rbytes < sizeof(RESPONSE); rbytes += n) {
                if ((n = recv(sock, (char*)msg_ptr + rbytes, sizeof(RESPONSE), 0)) <= 0)
                        cout << "ERROR: receiveing the response";
                count += n;

        }
        return count;
}

int TcpClient::attach_send(int sock, char* filename, int size, MESSAGE* msg_ptr) {//reading
the file while sending it
        int n;
        int count = 0;
        //open the file reading
        //the filename should hold the file name and place (the whole directory) else you can
pass the whole message to the function
        //char* toFile = (char*)malloc(size);

        ifstream file(filename, ios::binary);
        if (!file) {
                cout << "Error opening file";
                return 0;
        }


        if (!file) {
                cout << "ERROR: could not open the saved file!" << endl;
                return 0;
        }
        //hold the string in the buffer
        char buff[BUFFER_LENGTH];


        if (size < BUFFER_LENGTH) {
```

```
		char* buf = (char*)malloc(size);

		if (!buf)
		{
			file.close();
			cout << "ERROR: Creating a dynamic char pointer error" << endl;
			return 0;
		}
		if (!file.read(buf, size))//(!fread(buf, size, 1, fp))		//changes here
		{
			file.close();
			cout << *buf;
			cout << "ERROR: Reading into buffer error 1" << endl;
			return 0;
		}

		//memcpy( toFile, buf ,size);
		//strncat(toFile, buf, size);

		if ((n = send(sock, buf, size, 0)) != (size)) {
			cout << "size n n here";
			cout << size << endl;
			cout << n << endl;
			cout << "ERROR: Sending the file error" << endl;
			return n;
		}
		free(buf);

	}
	else {
		count = size;
		while (count > BUFFER_LENGTH) {
			//read them on buffer
			if (!file.read(buff, BUFFER_LENGTH))//(!fread(buff, BUFFER_LENGTH,
1, fp))
			{
				file.close();
				cout << "ERROR: Reading into buffer error" << endl;
				return 0;
			}
			//strncat(toFile, buff, BUFFER_LENGTH);
			if ((n = send(sock, buff, BUFFER_LENGTH, 0)) !=
(BUFFER_LENGTH)) {
				cout << "ERROR: Sending the file error (outside)" << endl;
			}

			count -= BUFFER_LENGTH;
		}
		if (!file.read(buff, count))//(!fread(buff, BUFFER_LENGTH, 1, fp))
		{
			file.close();
```

```
				cout << "ERROR: Reading into buffer error" << endl;
				return 0;
			}
			//strncat(toFile, buff, count);
			if ((n = send(sock, buff, count, 0)) != (count)) {
				cout << "ERROR: Sending the file error (outside)" << endl;
			}

			count -= count;

		}


		//free(buffer);


		//saveattachtofile((char*)attachfile.c_str(), toFile, size, SENT);

		return size - count;
}


int TcpClient::msg_recv(int sock, MESSAGE* msg_ptr) {
		//start with the header
		int rbytes, n, count = 0;

		memset(msg_ptr, 0, MSGHDRSIZE);
		for (rbytes = 0; rbytes < MSGHDRSIZE; rbytes += n) {
			if ((n = recv(sock, (char*)msg_ptr + rbytes, MSGHDRSIZE, 0)) <= 0) {
				cout << "ERROR: receiveing the header" << endl;
				cout << n << endl;
			}
			count += n;

		}

		//receive the message body
		int size = msg_ptr->datalength;
		count = 0;
		char* buffer = new char[BUFFER_LENGTH];

		if (size < BUFFER_LENGTH) {

			for (rbytes = 0; rbytes < size; rbytes += n) {
				if ((n = recv(sock, (char*)buffer + rbytes, size, 0)) <= 0) {
					cout << "The n: " << n << endl;
					err_sys("Recv BODY Error");
				}
				count += n;
			}
			buffer[size] = '\0';
			msg_ptr->body = buffer;
		}

		else {
			int counter = size;
			while (counter > BUFFER_LENGTH - 1) {

				for (rbytes = 0; rbytes < BUFFER_LENGTH - 1; rbytes += n) {
					for (rbytes = 0; rbytes < BUFFER_LENGTH - 1; rbytes += n) {
						if ((n = recv(sock, (char*)buffer + rbytes,
BUFFER_LENGTH - 1, 0)) <= 0)
							err_sys("Recv BODY inside Error");
					}

					counter -= (BUFFER_LENGTH - 1);

				}
				buffer[BUFFER_LENGTH - 1] = '\0';
				msg_ptr->body += buffer;
			}

			for (rbytes = 0; rbytes < counter; rbytes += n) {
				if ((n = recv(sock, (char*)buffer + rbytes, counter, 0)) <= 0) {//this just
dont work
					return n;
				}

			}
			buffer[counter] = '\0';
			msg_ptr->body += buffer;

			count = size;


		}
		return count; //returning the body of the message (which means that the header has
been received successfully)
}


int TcpClient::request_recv(int sock, REQUEST* msg_ptr) {
		int rbytes, n, count = 0;

		for (rbytes = 0; rbytes < sizeof(REQUEST); rbytes += n) {
			if ((n = recv(sock, (char*)msg_ptr + rbytes, sizeof(REQUEST), 0)) <= 0)
				cout << "ERROR: receiveing the request";
			count += n;

		}
		return count;
}
```

```
int TcpClient::response_recv(int sock, RESPONSE* msg_ptr) {
	int rbytes, n, count = 0;

	for (rbytes = 0; rbytes < sizeof(RESPONSE); rbytes += n) {
		if ((n = recv(sock, (char*)msg_ptr + rbytes, sizeof(RESPONSE), 0)) <= 0)
			cout << "ERROR: receiveing the response";
		count += n;

	}
	return count;

}

int TcpClient::attach_recv(int sock, char* filename, int size) {//try saving the file after receiving
it
	//better to put it in a buffer
	char buffer[BUFFER_LENGTH];
	int rbytes, n, count = 0;
	char* file = (char*)malloc(size);

	for (rbytes = 0; rbytes < size; rbytes += n) {
		n = recv(sock, (char*)file + rbytes, size, 0);
		cout << n << endl;
		count += n;
	}



	//if (size <= BUFFER_LENGTH) {
	//	for (rbytes = 0; rbytes < size; rbytes += n) {
	//		if ((n = recv(sock, (char*)file + rbytes, size, 0)) <= 0)
	//			cout << "ERROR: Receiving the file error" << endl;
	//		count += n;
	//	}
	//	return count;
	//}
	//else {
	//	int counter = size;
	//	while (counter > BUFFER_LENGTH) {
	//		for (rbytes = 0; rbytes < BUFFER_LENGTH; rbytes += n) {
	//			if ((n = recv(sock, (char*)file + (size - counter) + rbytes,
BUFFER_LENGTH, 0)) <= 0) {
	//				cout << "ERROR: Receiving the file error" << endl;
	//				return 0;
	//			}

	//		}
	//		counter -= BUFFER_LENGTH;
	//			cout << endl << "The container: " << file;//remove this
	//		}
	//		for (rbytes = 0; rbytes < counter; rbytes += n) {
	//			if ((n = recv(sock, (char*)file + (size - counter) + rbytes, counter, 0)) <=
0) {
	//				cout << "ERROR: Receiving the file error (outside)" << endl;
	//				return 0;
	//			}


	//		}
	//		count = size;
	//}

	////save the file using the filename (name and directory is held in filename)
	saveattachtofile(filename, file, size, INBOX);




	return count;
}

int TcpClient::updatelogfile(MESSAGE message, int stat) {//stat is used to tell if the directory
is inbox or sent 1:inbox, 2:sent
	string directory;
	directory = stat == INBOX ? inboxdir : sentdir;
	directory += "\\";
	directory += "logfile.csv";
	fstream file;
	file.open(directory, ios::app);
	if (!file.is_open())
		file.open(directory, ios::out);

	time_t time = (time_t)message.timestamp;
	if (file.is_open()) {
		string tosave = message.subject;
		tosave += '_';
		tosave += (string)asctime(localtime(&time));
		conditionString(&tosave);
		file << tosave;
		cout << tosave << endl;
		file << ',';
		if (stat == INBOX)
			file << 0;
		file << endl;
		addtomap(tosave, stat);
		return 1;
	}
	cout << "ERROR: could not open logfile" << endl;
	return 0;
	//logfile format
}


int TcpClient::getsize(string filename) {
	int size;
	FILE* fp = NULL;
	fp = fopen(filename.c_str(), "rb");
	if (!fp) return 0;
	if (fseek(fp, 0, SEEK_END) != 0)  // This should normally work
	{                              // (beware the 2Gb limitation, though)
		fclose(fp);
		return -1;
	}

	size = ftell(fp);
	fclose(fp);
	return size;
}

int TcpClient::isValid(char email[]) {
	string email_(email);
	if (regex_match(email_, regex("(\\w+)(\\.|_)?(\\w*)@(\\w+)(\\.(\\w+))+")))
		return 1;

	return 0;
}

int TcpClient::checkDir(char name[]) {
	struct stat buffer;
	if (stat(name, &buffer) != 0) {
		wchar_t wtext[20];
		mbstowcs(wtext, name, strlen(name) + 1);
		LPWSTR ptr = wtext;
		CreateDirectory(ptr, NULL);
		return 0;

	}
	else {
		return -1;

	}
}

int TcpClient::setupenv() {

	checkDir((char*)inboxdir);
	checkDir((char*)sentdir);
	string dir = inboxdir;
	dir += "\\";
	dir += "attachment";
	checkDir((char*)dir.c_str());
	string dir1 = sentdir;
	dir1 += "\\";
	dir1 += "attachment";
	checkDir((char*)dir1.c_str());
	return 1;


}

int TcpClient::Readlogfile(string* fileholder) { //inbox if the whole directory till inb) {
	//using another method
	for (int i = 1; i <= inboxcount; i++)
		*fileholder += inboxmap[i].mailname + ',';
	return 1;

	//string line, word;
	//string fname = inboxdir;
	//fname += "\\";
	//fname += "logfile.csv";
	////open the log file
	//fstream file(fname, ios::in);
	//if (file.is_open()) {
	//	while (getline(file, line,',')) {//reading the whole line in 'line' and then separating
them to deal with them addresses and names separately
	//		*fileholder += line;
	//		*fileholder += ',';
	//		getline(file, line, ',');
	//		cout << line << endl;
	//	}
	//	file.close();
	//	return 1;
	//}

	//cout << "ERROR: could not open logfile" << endl;
	//return -1;
}

int TcpClient::Sendlogfile(int sock, char* file, int size) {

	//hold the string in the buffer
	int n;
	if (size < BUFFER_LENGTH) {

		if ((n = send(sock, (char*)file, size, 0)) != (size)) {
			return n;
		}
	}
	else {
		int count = size;
		while (count > BUFFER_LENGTH) {

			if ((n = send(sock, (char*)file + (size - count), BUFFER_LENGTH, 0)) !=
(BUFFER_LENGTH)) {
				return n;
			}
			count -= (BUFFER_LENGTH);
		}
		if ((n = send(sock, (char*)file + (size - count), count, 0)) != (count)) {
			return n;
		}
		n = size;
	}
	return (n);

}

int TcpClient::Updateinbox(int sock, MESSAGE* msg_ptr) {
	//read the logfile (it's on the inbox dir)
	string logfile;
	int k = Readlogfile(&logfile);
	REQUEST req;
	int n;
	req.updatesize = 0;
	if (k == 0) {
		cout << "ERROR: Readlogfile error " << endl;
		return 0;
	}
	char hostname[HOSTNAME_LENGTH];
	gethostname(hostname, HOSTNAME_LENGTH);

	if (k != -1) {
		//send the request with the size set to the logfile length
		int size = (int)logfile.length();

		req.updatesize = size;
	}
	strcpy(req.hostname, hostname);
	strcpy(req.request, "update");
	cout << "Hostname: " << req.hostname << " connected." << endl;
	if ((n = request_send(sock, &req)) != sizeof(REQUEST)) {
		cout << "ERROR: request_send error (update)" << endl;
		return 0;
	}
	//send logfile
	if (req.updatesize != 0) {
		if (Sendlogfile(sock, (char*)logfile.c_str(), req.updatesize) != req.updatesize) {
			cout << "ERROR: Sendlogfile error" << endl;
```

```
                        return 0;
                }
        }
        //receive the response to get how many mail we should expect
        RESPONSE* resp = new RESPONSE();
        if (response_recv(sock, resp) != sizeof(RESPONSE)) {
                cout << "ERROR: response_recv error (update)" << endl;
                return 0;
        }

        if (resp->size > 0) {
                cout << "Receiving new messages ..." << endl;
                for (int i = 0; i < resp->size; i++) {

                        if (msg_recv(sock, msg_ptr) != msg_ptr->datalength) {
                                cout << "ERROR: msg_recv error (update)" << endl;
                                return 0;
                        }

                        time_t time = (time_t)msg_ptr->timestamp;
                        string attachfile;
                        attachfile = msg_ptr->subject;
                        attachfile += '_';
                        attachfile += (string)asctime(localtime(&time));
                        attachfile += '.';
                        attachfile += msg_ptr->filename;

                        conditionfilename(&attachfile);

                        saveemailtofile(msg, (char*)inboxdir);
                        if (msg_ptr->attachment != 0) {
                                // receive the attachment and save it

                                //filename += msg_ptr->filename;

                                if ((n = attach_recv(sock, (char*)attachfile.c_str(), msg_ptr-
>attachment)) != msg_ptr->attachment) {
                                        cout << "ERROR: attach_recv error (update) " << endl;
                                }
                                //attachment will be saved in the attach recv method
                                //now save the message




                        }
                        //update the logfile
                        //update the maps

                        if (updatelogfile(*msg_ptr, 1) == 0) {
                                cout << "ERROR: updatelogfile error" << endl;
```

```
                                        return 0;
                                }

                        }
                        cout << "You have " << resp->size << " new messages." << endl;

                }
                else {
                        //no message to receive display a message accordingly
                        cout << "You have no new messages." << endl;
                }
}


int TcpClient::saveemailtofile(MESSAGE* msg_ptr, char* directory) {//directory comes from
the run method (would be better if we change it to string)
#pragma warning(suppress : 4996)

                string savedfile;
                string saveddatfile;
                ofstream file;

                //get the proper time format
                time_t time = (time_t)msg_ptr->timestamp;
                savedfile = directory;

                //construct the subject_time filename
                savedfile += "\\";

                string test = savedfile + "dat";
                checkDir((char*)test.c_str());

                saveddatfile = savedfile + "dat" + "\\" + msg_ptr->subject + '_' +
(string)asctime(localtime(&time)) + ".dat";
                //using from_time for the dat file name (could be changed)
                savedfile += msg_ptr->subject;
                savedfile += "_";
                savedfile += (string)asctime(localtime(&time));//change this to proper time format
                savedfile += ".txt";
                conditionfilename(&saveddatfile);
                conditionfilename(&savedfile);


                file.open(savedfile.c_str());
                //write all the info to the file(from,to,subject,cc,body,timestamp)

                file << "From: " << msg_ptr->from << endl;
                file << "To: " << msg_ptr->to << endl;
                file << "cc: " << msg_ptr->ccmail << endl;
                file << "Subject: " << msg_ptr->subject << endl;
                file << "Body: " << endl << msg_ptr->body << endl;
```

```
	file << "Time: " << asctime(localtime(&time)) << endl;

	file.close();

	//save the dat file

	FILE* outfile;
	outfile = fopen(saveddatfile.c_str(), "w");
	if (outfile == NULL)
	{
		fprintf(stderr, "\nError opened file\n");
		exit(1);
	}

	// write struct to file
	fwrite(msg_ptr, MSGHDRSIZE, 1, outfile);
	fwrite(msg_ptr->body.c_str(), msg_ptr->datalength, 1, outfile);

	fclose(outfile);
}

int TcpClient::saveattachtofile(char* filename, char* file, int filesize, int dir) {
	//create a file

	string savedfile;
	if (dir == 1) savedfile = inboxdir;
	else savedfile = sentdir;
	savedfile += "\\";
	savedfile += "attachment";
	savedfile += "\\";
	savedfile += filename;


	FILE* attachment;
#pragma warning(suppress : 4996)
	attachment = fopen(savedfile.c_str(), "wb");//didnt test this should work though
	if (!attachment)
	{

		free(file);
		cout << "ERROR: Could not open file" << endl;
		return 0;
	}

	// Write the entire buffer to file
	if (!fwrite(file, sizeof(char), filesize, attachment))
	{
		free(file);
		fclose(attachment);
		cout << "ERROR:saving the file to local dir" << endl;
		return 0;
```

```
        }
        fclose(attachment);
        return 1;
}

int TcpClient::getmessageinput(MESSAGE* msg, string* filepath) {
        char data[HOSTNAME_LENGTH];

        cout << "Creating New Email." << endl;
        cout << endl;

        cout << "To: ";
        cin >> data;

        while (!isValid(data)) {
                cout << "Please enter the right email address format (test@example.com): ";
                cout << "To: ";
                cin >> data;
        }
        strcpy(msg->to, data);

        cout << "From: ";
        cin >> data;
        while (!isValid(data)) {
                cout << "Please enter the right email address format (test@example.com): ";
                cout << "From: ";
                cin >> data;
        }
        strcpy(msg->from, data);

        cin.ignore();
        cout << "Subject: ";
        cin.getline(msg->subject, SUBJECTSIZE);

        cout << "Body: ";
        string newdata;
        msg->body = "";
        while (getline(cin, newdata))
        {
                if (newdata.empty()) {
                        break;
                }
                msg->body += newdata;
        }

        msg->datalength = msg->body.length();

        //include CC if we are doing that
        cout << "Enter CC emails you want to include (press 'Enter' if you don't have any or
you are done and after each mail): " << endl;
```

```
	while (getline(cin, newdata))
	{
		if (newdata.empty()) {
			break;
		}
		//use isvalid
		if (!isValid((char*)newdata.c_str())) {
			cout << "Please enter the right email address format
(test@example.com): ";
			continue;
		}
		msg->ccmail += newdata;
		msg->ccmail += ',';
	}
	msg->cc = msg->ccmail.length();
	cout << msg->ccmail << endl;
	/////// the cc ends here

	//check if they have an attachment
	char answer[5];
	msg->attachment = 0;
	cout << "Do you want to attach a file? (y/n): ";
	cin >> answer;
	while (strcmp(answer, "y") != 0 && strcmp(answer, "yes") != 0 && strcmp(answer, "n")
!= 0 && strcmp(answer, "no") != 0) {
		cout << "Do you want to attach a file? (Please enter either y or n only): ";
		cin >> answer;
	}
	if (strcmp(answer, "y") == 0 || strcmp(answer, "yes") == 0) {
		cout << "Enter the file location(The full path) : ";
		string filename;
		cin.ignore();
		getline(cin, filename);
		while ((msg->attachment = getsize(filename)) == -1) {
			cout << "Could not open the file please enter the file with its full path : ";
			getline(cin, filename);
		}

		string temp = filename.substr(filename.find_last_of(".") + 1);
		strcpy(msg->filename, temp.c_str());
		*filepath = filename;
	}
	msg->timestamp = (int)time(nullptr);
	return 1;

}

int TcpClient::Sendmail(int sock, MESSAGE* msg, bool fromForward, string attachFromFWD)
{
	//construct message
	string filepath;
	if (!fromForward) {

		if (getmessageinput(msg, &filepath) == 0) {
			cout << "ERROR: getmessageinput error" << endl;
			return 0;
		}
	}

	//send info that i am sending
	//
	gethostname(req.hostname, HOSTNAME_LENGTH);
	strcpy(req.request, "send");
	int n;
	if ((n = request_send(sock, &req)) != sizeof(REQUEST)) {
		cout << "ERROR: request_send error (send)" << endl;
		return 0;
	}
	//send mail

	if (msg_send(sock, msg) != msg->datalength) {
		cout << "ERROR: msg_send error" << endl;
		return 0;
	}


	if (msg->attachment != 0) {
		if (fromForward) {
			filepath = attachFromFWD;
		}
		if (attach_send(sock, (char*)filepath.c_str(), msg->attachment, msg) != msg-
>attachment) {
			cout << "ERROR: attach_send error" << endl;
			return 0;
		}
		time_t time = (time_t)msg->timestamp;
		string attachfile;
		attachfile = msg->subject;
		attachfile += '_';
		attachfile += (string)asctime(localtime(&time));
		attachfile += '.';
		attachfile += msg->filename;
		conditionfilename(&attachfile);
		attachfile = "sent\\attachment\\" + attachfile;

		fstream src(filepath, ios::in);
		fstream dst(attachfile, ios::out);
		dst << src.rdbuf();//this will copy and paste the file
		src.close();
		dst.close();
	}
	cout << "Message sent to server!" << endl;
	//receive the confirmation
	RESPONSE* resp = new RESPONSE();
	if ((n = response_recv(sock, resp)) != sizeof(RESPONSE)) {
		cout << "ERROR: could not receive the response from the server . . ." << endl;
		return 0;
	}
	cout << "Response from server: " << resp->response << endl;
	if (strcmp(resp->response, "550 ERROR") == 0 || strcmp(resp->response, "501
ERROR") == 0) {
		return 0;
	}
	saveemailtofile(msg, (char*)sentdir);

	if (updatelogfile(*msg, SENT) == 0) {
		cout << "ERROR: updatelogfile error (sent)" << endl;
	}
	return 1;
}



int TcpClient::loadmaps() {//loads the sentmap and inboxmap
	string line, word;
	string fname = inboxdir;
	fname += "\\";
	fname += "logfile.csv";
	//open the log file

	fstream file(fname, ios::in);
	if (file.is_open()) {
		while (getline(file, line)) {
			inboxcount += 1;
			stringstream holder(line);
			getline(holder, word, ',');
			filestruct* structholder = new filestruct();
			structholder->mailname = word;
			int read;
			getline(holder, word, ',');
			structholder->readstatus = stoi(word);
			inboxmap.insert(pair<int, struct filestruct>(inboxcount, *structholder));
		}
		file.close();
	}
	fname = sentdir;
	fname += "\\";
	fname += "logfile.csv";
	//open the log file

	fstream file2(fname, ios::in);
	if (file2.is_open()) {
		while (getline(file2, line)) {
```

```
                        sentcount += 1;
                        sentmap.insert(pair<int, std::string>(sentcount, line));
                }
                file2.close();
                return 1;
        }
        return -1;
}

int TcpClient::addtomap(string toadd, int stat) {//stat is used to tell where to import toadd, 1 for
inboxmap, else for sentmap and the string holds subject_timestamp
        if (stat == 1) {//inboxmap
                filestruct* holder = new filestruct();
                holder->mailname = toadd;
                holder->readstatus = 0;
                inboxcount += 1;
                inboxmap.insert(pair<int, filestruct>(inboxcount, *holder));
                cout << "Added to inbox map." << endl;

        }
        else {//sentmap
                sentcount += 1;
                sentmap.insert(pair<int, string>(sentcount, toadd));
        }
        return 1;

}

int TcpClient::markasread(int k) {//k is the key of the inboxmap (which should the input from
the user to read a mail)

        inboxmap[k].readstatus = 1;
        if (updatelogfiles() != 1) {
                cout << "ERROR: updating the inbox logfile error" << endl;
        }
        list_unread();
        return 1;
}

int TcpClient::updatelogfiles() { //updating the inbox only
        string filepath = inboxdir;
        filepath += "\\";
        filepath += "logfile.csv";

        ofstream File(filepath);
        for (int i = 1; i <= inboxcount; i++) {
                cout << "The inboxmap: " << inboxmap[i].mailname << endl;
                File << inboxmap[i].mailname << ',' << inboxmap[i].readstatus << endl;
        }
        return 1;
```

```
}

int TcpClient::updateinboxlog() { //updating the inbox only
	string filepath = inboxdir;
	filepath += '\\';
	filepath += "logfile.csv";
	cout << filepath << endl;
	ofstream file(filepath, ios::trunc);
	for (int i = 1; i <= inboxcount; i++) {
		cout << inboxmap[i].mailname << endl;
		file << inboxmap[i].mailname << ',' << inboxmap[i].readstatus; //THE
PROBLEM IS HERE FOR SOME REASON THE FILE IS NOT BEING OVERWRITTEN
WHENEVER THE MAP IS UPDATED
	}
	if (list_read() != 1) {
		cout << "ERROR: Listing the  read error" << endl;
		return -1;
	}
	file.close();
	return 1;

}

bool TcpClient::Displayinbox() {
	bool exists = false;
	for (int i = 1; i <= inboxcount; i++) {
		cout << i << ": " << inboxmap[i].mailname << endl;
		exists = true;
	}
	return inboxcount>0;
}

bool TcpClient::Displaysent() {
	bool exists = false;
	for (int i = 1; i <= sentcount; i++) {
		cout << i << ": " << sentmap[i] << endl;
		exists = true;
	}
	return exists;
}

void TcpClient::DisplayMessage() {

	cout << "Please enter your choice:" << endl;
	cout << "1: Refresh inbox" << endl;
	cout << "2: Send Email" << endl;
	cout << "3: Display inbox" << endl;
	cout << "4: Forward a message" << endl;
	cout << "5: Exit" << endl;
}
```

```
int TcpClient::Displayinboxmail(int k) { //k is the chosen inbox mail number

	MESSAGE msg;
	if (readdatfile(&msg, k, 1) != 1) {
		cout << "ERROR: readdatfile error" << endl;
		return 0;
	}
	//display each fields
	time_t time = (time_t)msg.timestamp;
	cout << "Mail" << endl;//not really sure how this is done
	cout << "From: " << msg.from << endl;
	cout << "Subject: " << msg.subject << endl;
	cout << "Body: " << msg.body << endl;
	cout << "Time sent: " << asctime(localtime(&time)) << endl;

	return 1;


}

int TcpClient::readdatfile(MESSAGE* msg, int k, int stat) { //used to read dat file (k: holds the
chosen mail , stat=1 if inobx else sent)
	string dir = stat == 1 ? inboxdir : sentdir;
	dir += '\\';
	dir += "dat";
	dir += '\\';
	string filename = stat == 1 ? inboxmap[k].mailname : sentmap[k];
	dir += filename.substr(0,filename.length());
	dir += ".dat";
	conditionfilename(&dir);
	FILE* outfile;
	outfile = fopen(dir.c_str(), "rb");
	if (outfile == NULL)
	{
		cout << "ERROR: can not open dat file" << endl;
		return 0;
	}

	// read struct from file
	fread(msg, MSGHDRSIZE, 1, outfile);//this definitely is wrong (do some trick to set the
right message size)
	//cout << msg->from<<endl;
	char* buffer = new char[BUFFER_LENGTH];
	int i = msg->datalength;
	msg->body = "";
	while (i > BUFFER_LENGTH - 1) {
		fread(buffer, BUFFER_LENGTH - 1, 1, outfile);
		buffer[BUFFER_LENGTH - 1] = '\0';
		msg->body += buffer;
		i -= BUFFER_LENGTH - 1;
	}
```

```
		fread(buffer, i, 1, outfile);
		buffer[i] = '\0';
		msg->body += buffer;

		fclose(outfile);
		return 1;

}

void TcpClient::conditionfilename(string* str) {
		replace(str->begin(), str->end(), ':', '_');
		str->erase(std::remove(str->begin(), str->end(), '\n'), str->end());
		str->erase(std::remove(str->begin(), str->end(), ','), str->end());



}
void TcpClient::conditionString(string* str) {
		str->erase(std::remove(str->begin(), str->end(), '\n'), str->end());
		str->erase(std::remove(str->begin(), str->end(), ','), str->end());
}


int TcpClient::list_read() {   //from the map
		for (int i = 1; i <= inboxcount; i++) {
				if (inboxmap[i].readstatus == 1) {
						cout << i << " " << inboxmap[i].mailname << ',' <<
inboxmap[i].readstatus << " :read" << endl;
				}
		}
		return 1;
}
int TcpClient::list_unread() {
		int k = 0;
		for (int i = 1; i <= inboxcount; i++) {
				if (inboxmap[i].readstatus == 0) {
						cout << i << ": " << inboxmap[i].mailname << ',' <<
inboxmap[i].readstatus << " :unread" << endl;
						k++;
				}
		}
		if (k == 0)
				return 2;
		return 1;
}
int TcpClient::displayEmail(string dir, string allemails[10]) {
		cout << "from displayEmail: " << dir << endl;
		ifstream fpS;
		fpS.open(dir);
		if (!fpS) {
				cout << "There might be no emails in this directory" << endl;
				return 0;
```

```
        }
        char emailChar;
        int counter = 1;
        //string allemails[10];//10 is arbitrary
        fpS >> noskipws;
        fpS >> emailChar;
        cout << counter << ". ";
        while (!fpS.eof()) {

                if (emailChar == ',') { cout << counter << ". "; counter++; }
                else if (emailChar == ':') { cout << '_'; allemails[counter] += '_'; }
                else {
                        cout << emailChar;
                        allemails[counter] += emailChar;
                }
                fpS >> emailChar;
        }
        fpS.close();
        return 1;
}
int TcpClient::forwardEmail(int sock, MESSAGE* msg_ptr) {

        //FILE* fp = NULL;
//#pragma warning(suppress : 4996)
        cout << "Your Sent directory:" << endl;
        string dir =  "\\logfile.csv";
        dir = sentdir + dir;
        bool sentEmailsExist = true;
        if(!Displaysent()) sentEmailsExist = false;//check if there are emails

        cout << "Your Inbox directory:" << endl;
        string dir1 = "\\logfile.csv";
        dir1 = inboxdir + dir1;
        bool inboxEmailsExist = true;
        if(!Displayinbox()) inboxEmailsExist = false;//check if there are emails


        if(sentEmailsExist)cout << "Enter 0 to forward an email from your Sent directory" << 
endl;
        if(inboxEmailsExist)cout << "Enter 1 to forward an email from your Inbox directory" << 
endl;

        int sentORinbox;
        cin >> sentORinbox;

        cout << "Enter the no. of the email to forward" << endl;
        int emailID;
        cin >> emailID;
        //read the email to buffer
        if (readdatfile(msg_ptr, emailID, sentORinbox) != 1) cout << "ERROR: reading from 
the 'readdatfile'"<<endl;
```

```
	cout << "To: " << endl;
	char newTo[HOSTNAME_LENGTH];
	cin >> newTo;
	while (!isValid(newTo)) {
		cout << "Please enter the right email address format (test@example.com): ";
		cout << "To: ";
		cin >> newTo;
	}
	strcpy(msg->to, newTo);

	string addFWD = msg_ptr->subject;
	addFWD = "FWD: " + addFWD;
	strcpy(msg_ptr->subject, addFWD.c_str());

	string attachmentPath="";
	if (msg_ptr->attachment != 0) {
		attachmentPath = (sentORinbox == 1) ? inboxdir : sentdir;
		attachmentPath += "\\attachment\\";

		if (sentORinbox == 1) attachmentPath += inboxmap[emailID].mailname;
		else attachmentPath += sentmap[emailID];
		attachmentPath += '.';
		attachmentPath += msg_ptr->filename;
		conditionfilename(&attachmentPath);
	}
	cout << attachmentPath;

	if (Sendmail(sock, msg, true, attachmentPath) == 0) {
		cout << "ERROR: ForwardEmail error" << endl;
		return 0;
	}
	return 1;
}

void TcpClient::run(int argc, char* argv[]) {
	if (WSAStartup(0x0202, &wsadata) != 0)
	{
		WSACleanup();
		err_sys("Error in starting WSAStartup()\n");
	}

	char* inputserverhostname = new char[HOSTNAME_LENGTH];

	std::cout << "Type name of Mail server: ";
	std::cin >> inputserverhostname;
	std::cout << endl;

	initiateconnection(inputserverhostname);
```

```
    setupenv();//sets the inbox and sent directories
    //might include a dat dir if we plan to resend mail

    //update mail before getting into the while loop

    int k = 1; //holds all the choices we intend to have

    //1 means update
    msg = new MESSAGE();
    while (k != 5) {//k=3(or any number we choose to use) means exit
        switch (k) {
        case 1:
            cout << "Updating the inbox..." << endl;
            if (Updateinbox(sock, msg) == 0) {
                cout << "ERROR: Updateinbox error" << endl;
                //you might wanna terminate the thread
            }
            break;

        case 2:
            cout << "You have chosen to write a mail" << endl;
            //construct and send mail
            if (Sendmail(sock, msg, false, "") == 0) {
                cout << "ERROR: Sendmail error" << endl;
                //you might wanna terminate the thread
            }


            break;
        case 3:
            //reads mail
            cout << "Viewing Unread Emails: " << endl;
            //list all inbox
            //
            //Displayinbox(); // to see all of them,
            int m;

            m = list_unread();
            if (m != 1 && m != 2) {//to see the unread only - EITHER ONE IS
ENOUGH
                cout << "error: listing unread error" << endl;
                break;
            }
            if (m == 1) {
                cout << "Please enter the number of the email you want to read:
" << endl;
                int num;
                cin >> num;

                //validation check
                while (num > inboxcount && inboxmap[num].readstatus != 0) {
```

```
                                        cout << "Please enter the number of the email you want
to read (from the list only) : " << endl;
                                        cin >> num;
                                }

                                if (markasread(num) != 1)
                                        cout << "ERROR: Marking as read error" << endl;
                                //display the mail
                                Displayinboxmail(num);

                        }
                        else {
                                cout << "No unread mails." << endl;
                        }

                        break;
                case 4:
                        //forwards mail
                        if (forwardEmail(sock, msg) == 0) {
                                cout << "ERROR: forwarding email" << endl;
                        }
                        break;
                case 5:
                        //exit
                        cout << "Closing the application . . ." << endl;
                        exit(0);
                        break;

                }
                //show the prompt and we done
                DisplayMessage();
                cout << "Please enter your choice: ";
                cin >> k;
        }

}

int main(int argc, char* argv[])
{
        TcpClient* tc = new TcpClient();
        tc->run(argc, argv);
        return 0;
}
```