



جامعة خليفة
Khalifa University

Department of Electrical Engineering and Computer Science

**ECCE 356 – Computer Networking
Assignment 1 - SMTP**

Group Members

Tiemar Berhe 100049901
Yaphet Elias 100049897
Amine Kidane 100053678

Contents

Introduction	3
Background	3
Assignment Description	3
Solution	3
Constant Values:.....	4
Changes to the message structure:	4
Changes to the Buffer size:.....	4
Changes to the message header size:	4
Changes to the request structure:.....	5
The new function that validates the email address format:	5
The change to how and what the server responds to the client:.....	6
Demo.....	7
Conclusion.....	9

Introduction

By the end of this assignment, we are expected to grasp the basics of network programming within the TCP/IP Internet family environment, a little bit of multi-threading, and socket API. As part of the assignment, we are going to implement part of the SMTP protocol from the client and the server side.

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.

Assignment Description

Initially, the server should be on and waiting for connections. Once the client connects to the running server, the user should be able to create a new mail in the console. All the input messages of the email i.e. To, From, Subject, Body including a timestamp should be processed and sent to the server via TCP. After the email is sent, until the server accepts the email and responds back, the client’s application should be waiting for response.

Later, when the server application accepts the email, it should be able to verify the header to check if the email’s header contents are valid and respond accordingly. If the email is found to be valid the server is going to initialise a send_request of ‘250 OK’, otherwise ‘501 Error’.

Finally, when the client receives the response, it will display to the user a message stating that the email is received successfully along with a timestamp if the response received is ‘250 OK’ and an error message if ‘501 Error’ is received. And the server application should always be running unless halted by control-C interrupt.

Solution

For our solution, we had to do a couple of changes to the provided code so that it would accommodate the above-mentioned functionalities. Those changes include message structure, request structure, message header size, and we added an extra function that validates the to and from mail address. We also changed the response message of the server depending on the condition that we get the right to and from mail addresses.

Constant Values:

```
#define HOSTNAME_LENGTH 20
#define RESP_LENGTH 40
#define REQUEST_PORT 5001
#define BUFFER_LENGTH 1044
#define MAXPENDING 10
#define MSGHDRSIZE 120 //Message Header Size
#define MESSAGE_SIZE 1024
#define SUBJECTSIZE 70
```

Figure 2. Constant values

These are the constant values that we are using. Most of them are simply used as they are given on the assignment, for the ones that we changed or added they will be discussed below.

Changes to the message structure:

As seen from the background the message structure has to include the addresses of to and from, the subject, the timestamp, and the length of the message as the headers, as well as buffer to hold the data. So, the new structure would look like this:

```
typedef struct
{
    char to[HOSTNAME_LENGTH];
    char from[HOSTNAME_LENGTH];
    char subject[SUBJECTSIZE];
    int length;
    int timestamp;
    char buffer[BUFFER_LENGTH];
}SMTPMSG;
```

Figure 3.Modified Message Structure

This change is implemented on both the client and on the header part of the server.

Changes to the Buffer size:

The buffer holds the message and the host name hence its size should be big enough to accommodate those. Since the hostname is a char array of 20 elements and the message is a char array of 1024 elements the buffer has a size of 1044 elements.

Changes to the message header size:

From the given specifications, the header includes the 'to' and 'from' addresses, the subject, and the timestamp only, but to aid in transmitting the right message size, we added the length variable in the header too. The new message header would be equal to the size of the new message structure minus the size of the buffer:

$\text{MessageHeaderSize} = \text{sizeof}(\text{SMTP}) - \text{sizeof}(\text{buffer})$

Since the HOSTNAME=20 and SUBJECTSIZE=70 we have the MessageHeaderSize = 120

```
#define MSGHDRSIZE 120 //Message Header Size
```

Figure 4. Message Header Size

This change is implemented on both the client and on the header part of the server. As we can see that the MessageHeaderSize is not equal to the size of the 'to', 'from', subject, length and timestamp fields but this is due to the padding that is applied by the compiler. The actual size of the header would have been 118 but since 118 is not divisible by 4 the compiler adds a 2-byte padding to the first 5 fields of the SMTPSMG structure.

Changes to the request structure:

The request structure now includes the hostname and the data, which is the body of the mail. For this assignment, we chose the mail to let you send about 1024 characters (this can be adjusted accordingly), so the data variable of the request is a character array of size 1024 as seen below:

```
typedef struct {
    char hostname[HOSTNAME_LENGTH];
    char data[MESSAGE_SIZE];
} NReq;
```

Figure 5. Modified Request Structure

The new function that validates the email address format:

For the server to confirm the validity of the email we implemented a function that checks if the given address input is a real email. The function basically checks if the address has "@" and "." character in the address and makes sure that those characters are in the right places ('@' comes before '.', '@' and '.' shouldn't be next to each other, the first letter of the mail should be an alphabet character and the "." shouldn't come at the end). The function looks like this:

```
bool TcpThread::isvalid(char str[]) {
    if (strlen(str) == 0)
        return 0;
    if (!ischar(str[0])) {
        return 0;
    }
    int at = -1;
    int dot = -1;
    for (int i = 0; i < strlen(str); i++) {
        if (str[i] == '@')
            at = i;
        if (str[i] == '.')
            dot = i;
    }
    if (dot == -1 || at == -1 || at > dot || dot == at+1)
        return 0;
    return !(dot == (strlen(str) - 1));
}

bool TcpThread::ischar(char k) {
    return ((k >= 'a' && k <= 'z') || (k >= 'A' && k <= 'Z'));
}
```

Figure 6. Function to validate email address format

This change is only in the server code only since the mail checking is done on the server only.

The change to how and what the server responds to the client:

As discussed above, the `isvalid()` function helps in identifying if the address is right or wrong. If the addresses are right, the server replies with a message “250 OK” and the timestamp, but if the addresses are wrong the server responds with the message “501 ERROR” which then the client agent uses to give the right response to the user.

The code is shown here:

```
bool validfrom = isvalid(rmsgnew.from);
bool validto = isvalid(rmsgnew.to);
if(!(validfrom && validto))
    sprintf(respp->response, "501 Error");
else {
    printf("\nMail Received from: %s\n", reqpp->hostname);
    printf("From: %s\n", rmsgnew.from);
    printf("To: %s\n", rmsgnew.to);
    printf("Subject: %s\n", rmsgnew.subject);
    time_t time = (time_t)rmsgnew.timestamp;
    printf("Time: %s\n", asctime(localtime(& time)));
    printf("%s\n", reqpp->data);
    memset(respp->response, 0, sizeof(Resp));
    sprintf(respp->response, "250 OK");
}

if(msg_send(cs, &smtpmsg) != smtpmsg.length)
    err_sys("send Respose failed, exit");
printf("Response for %s has been sent out\n\n", reqpp->hostname);

closesocket(cs);
```

Figure 7. Server Code on Responding to the sender client

```
if (strcmp(respp->response, "250 OK")==0) {
    printf("Email received successfully at %s", asctime(localtime(&timeFromServer)));
}
else {
    printf("Email is not received. Check if the emails were written correctly.\n");
}

closesocket(sock);
```

Figure 8. Client Code on interpreting the server response

Demo

The following figures illustrate the workings (and some errors handled) of the whole assignment:

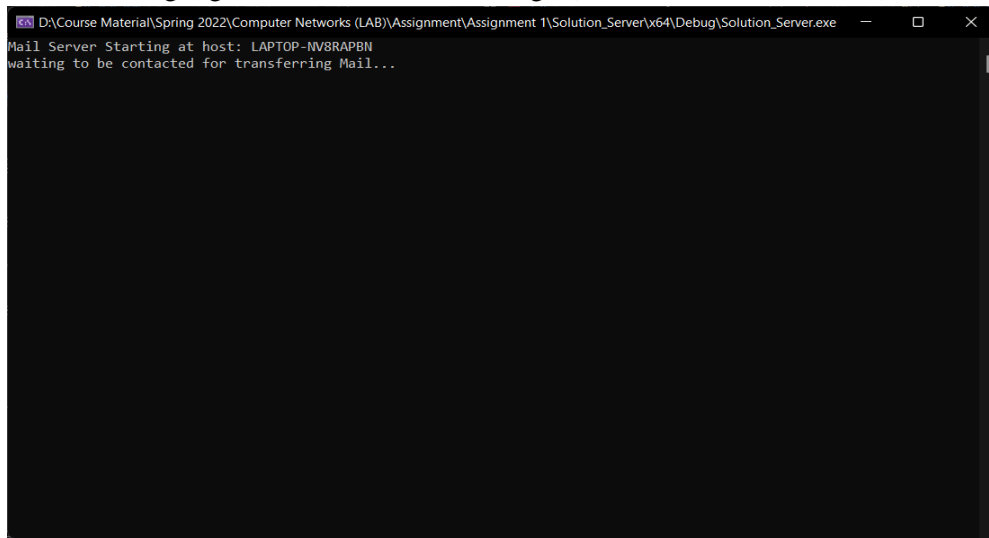


Figure 9. Initializing the server

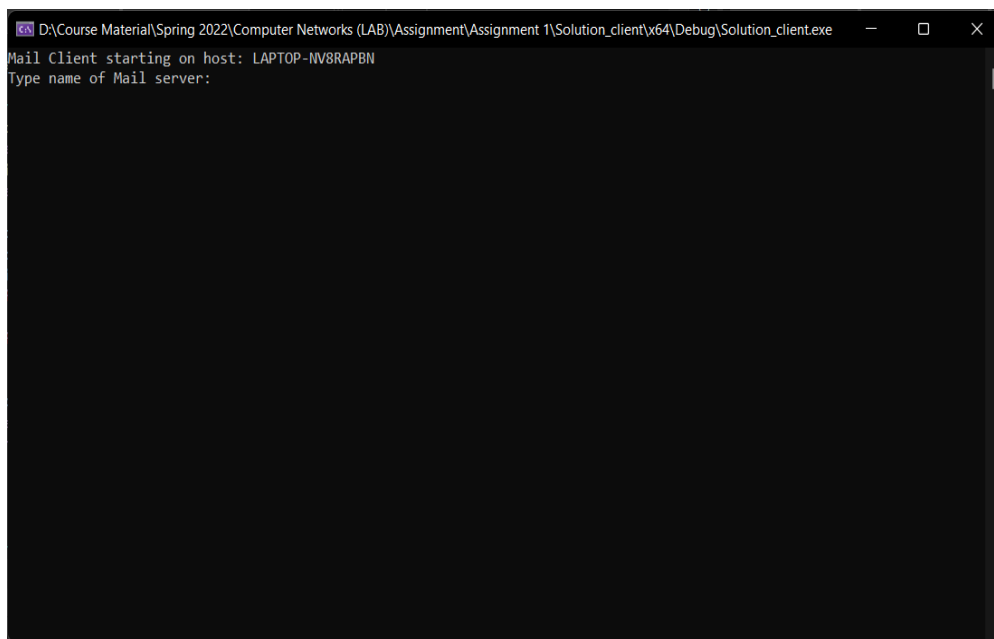
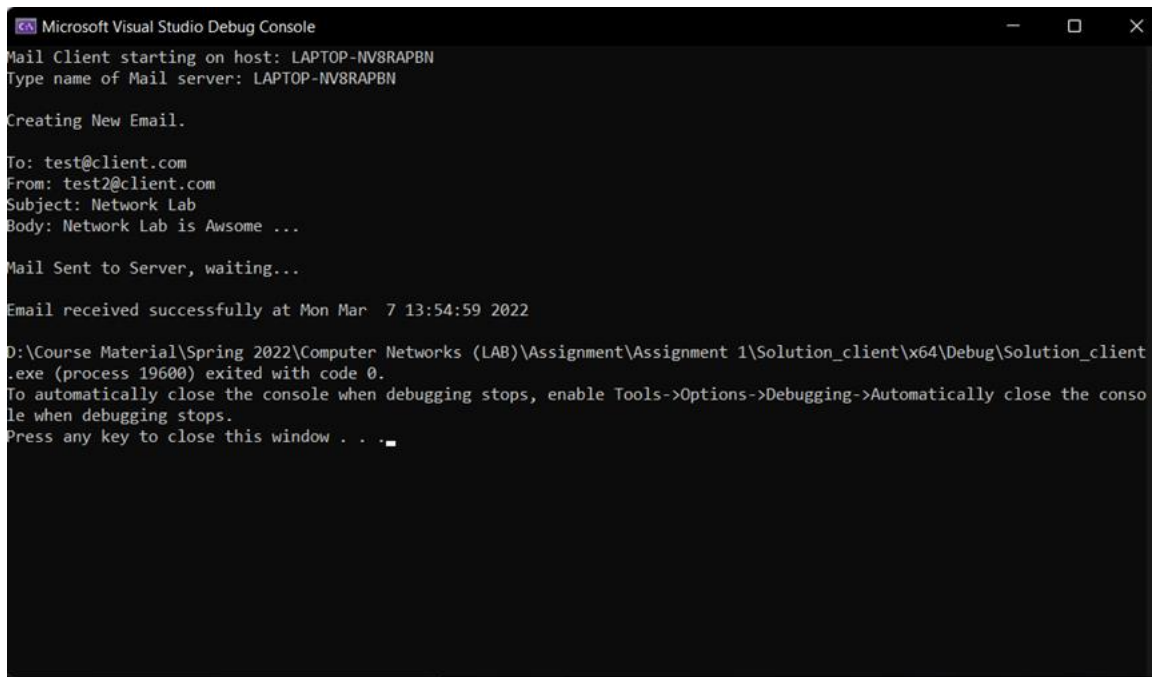


Figure 10. Initializing the Client Agent



```
Microsoft Visual Studio Debug Console
Mail Client starting on host: LAPTOP-NV8RAPBN
Type name of Mail server: LAPTOP-NV8RAPBN

Creating New Email.

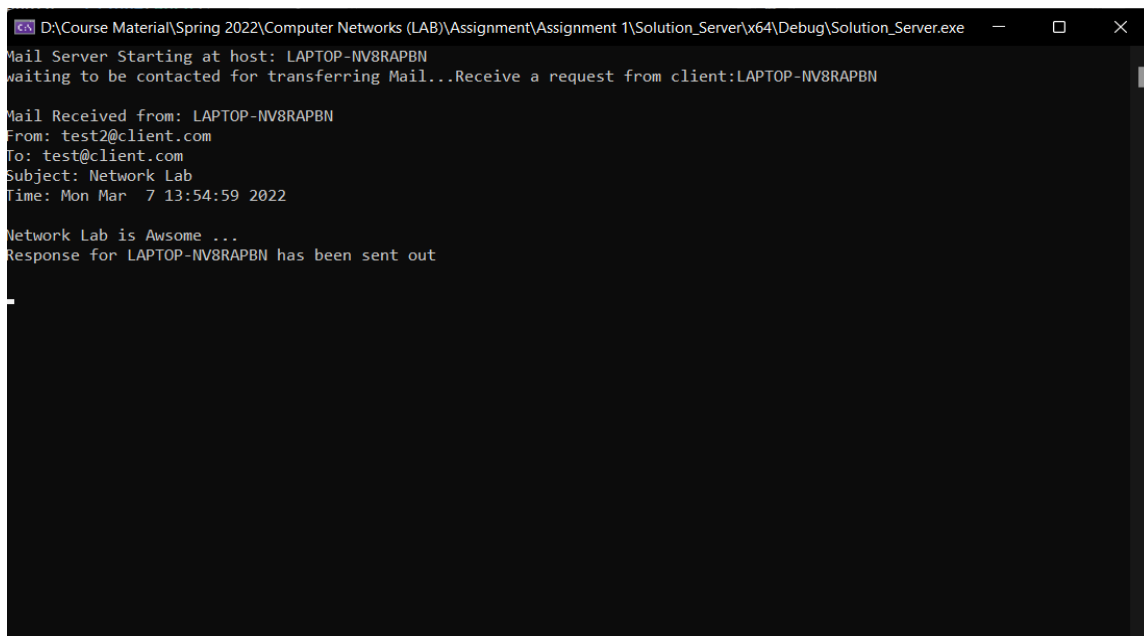
To: test@client.com
From: test2@client.com
Subject: Network Lab
Body: Network Lab is Awsome ...

Mail Sent to Server, waiting...

Email received successfully at Mon Mar 7 13:54:59 2022

D:\Course Material\Spring 2022\Computer Networks (LAB)\Assignment\Assignment 1\Solution_client\x64\Debug\Solution_client.exe (process 19600) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Figure 11. Entering the message - address and body, and Server's response



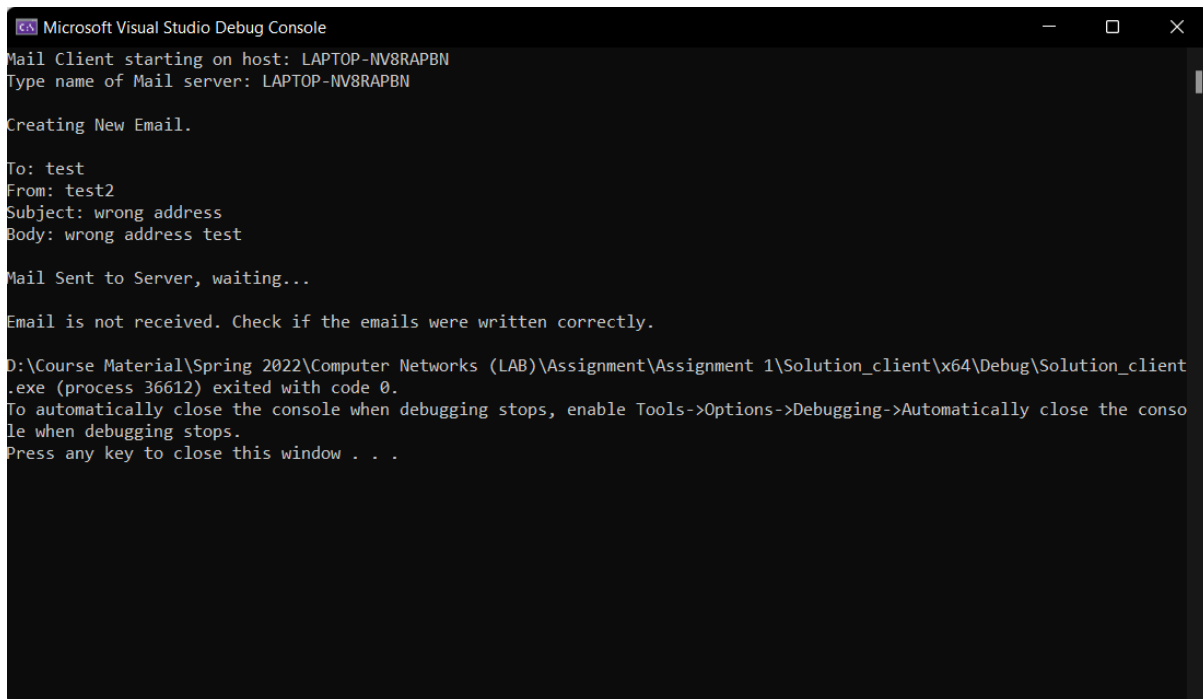
```
D:\Course Material\Spring 2022\Computer Networks (LAB)\Assignment\Assignment 1\Solution_Server\x64\Debug\Solution_Server.exe
Mail Server Starting at host: LAPTOP-NV8RAPBN
waiting to be contacted for transferring Mail...Receive a request from client:LAPTOP-NV8RAPBN

Mail Received from: LAPTOP-NV8RAPBN
From: test2@client.com
To: test@client.com
Subject: Network Lab
Time: Mon Mar 7 13:54:59 2022

Network Lab is Awsome ...
Response for LAPTOP-NV8RAPBN has been sent out
```

Figure 12. Message and Response display

The server, after sending the response for the client, goes into waiting state to wait for another client to connect.

The image is a screenshot of the Microsoft Visual Studio Debug Console window. The title bar at the top reads "Microsoft Visual Studio Debug Console". The console output shows the following text:
Mail Client starting on host: LAPTOP-NV8RAPBN
Type name of Mail server: LAPTOP-NV8RAPBN

Creating New Email.
To: test
From: test2
Subject: wrong address
Body: wrong address test

Mail Sent to Server, waiting...

Email is not received. Check if the emails were written correctly.

D:\Course Material\Spring 2022\Computer Networks (LAB)\Assignment\Assignment 1\Solution_client\x64\Debug\Solution_client.exe (process 36612) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

Figure 13. Error on the address fields

From figure 13 above we can see that the server has identified that one of the email addresses were wrong, so it responded with the value “501 Error” and the client agent recognised this and displayed the respective information.

Conclusion

The aim of the assignment was to let us have a hands-on experience on dealing with network applications. It helped us understand how to program a socket, a protocol and how to structure a data depending on a protocol specification. Moreover, this simple program we implemented on our assignment can be better upgraded by adding sign-up/sign-in and mail access functionalities.