Computer Network Laboratory
Assignment 4
Assignment given on: 07th February 2020
Submission deadline: 19th February 2020(5:30PM)

Assignment 4: Design of a Mail Server and Client

In this assignment, you have to learn about the SMTP protocol and POP3 protocol and build a simple mail server and client to send and receive mails. SMTP is used to send mails from a mail client to the mail server, while POP3 is used to access mail from a mail server by a mail client.

Two SMTP servers connect using a TCP connection to send and receive mails. The connection is initiated by the sender. Once the TCP connection is made, the sender sends a sequence of text commands to the receiver. The receiver replies with a code (an integer) and a short message to each command. All commands and replies are nothing but text strings in a specific format. The connection is finally closed by the sender.
More details of SMTP are available in RFC 821, and in Stalling's book (Ch. 22).
In this assignment, you will first implement a subset of the SMTP command/replies.

POP3 is a protocol to retrieve mails and manage the mailbox, maybe from a remote machine. However, POP3 does not handle sending of mails. It also uses text commands and responses. More details of POP3 are available in RFC 1939. In this assignment, you will also implement a POP3 client-server.

Typically, the SMTP server and the POP3 server will run on the mail server (like your cse mail machine). Your home machine will run the POP3 client to access the mailbox, and a SMTP client to send mail through the SMTP server. In this assignment, you will learn how to send/receive mails from a remote host using SMTP/POP3. In the rest of this assignment, we will refer to the mail server machine as ***MailServer***, and the remote machine from which you will access mail as the ***Home***.

All servers described in this assignment are TCP concurrent servers. You have to think about cases of multiple users connecting to the same mail server to access their mailboxes, or multiple mail servers sending mail to a single mail server or both at the same time. You do not have to worry about the case when a single user tries to access his mailbox from multiple clients.

PROGRAMS ON THE *MailServer* MACHINE

The *MailServer* machine will do two things:

1. run a SMTP mail server to receive mails from another mail client/server using SMTP and store them

2. run a POP3 server to let the *Home* machine access and manage the mailbox

But first, we need to setup a few things. In the directory from which you will run the program, create a file called *user.txt*. Each line of the file contains a one word user login name and a one word password, separated by one or more spaces. Create the file to contain one user for each group member. Then, in the same directory, create one subdirectory for each of the users named after the
users. The mailboxes for each user will be stored in the respective sub directories.

To do task 1 above, you will write a C program named *smtpmail.c*. The program will take a command line integer argument *my_port* that will indicate the port on which the mail server will run. We will refer to the process corresponding to this program as R. The process R handles receive of incoming mail. It simply waits on a socket bound at port *my_port*. When a connection is made from a sender, the process follows the SMTP protocol to receive the message. The received
message is then **appended** to the end of a file named *mymailbox* that is stored in the user's subdirectory (remember that the subdirectories are created manually). The username for the mail will be in the mail address, so R knows which subdirectory to go to. The mail is appended in the following format

.

From: <username>@<domain_name>
To: <username>@<domain_name>
Subject: <subject string, max 100 characters>
Received: <time at which received, in date : hour : minute>
<Message body – one or more lines>

.

Note the added *Received* line. You can get the time from the system clock easily. Thus, the *mymailbox* file at any instant contains 0 or more such messages, each separated by a fullstop character (only the fullstop character in one line).

To do task 2, you will write a program call *popserver.c*, which will be the POP3 server. The program will take a command line integer argument *pop3_port* that will indicate the port on which the POP3 server will run. Details of POP3 are given later.

PROGRAM ON THE *Home* MACHINE

The *Home* machine will have a program called *mailclient.c* that implements both send and receive of mail. The program will take a command line argument *server_IP* that will indicate the IP address of the SMTP and POP3 server to connect to (i.e., the IP address of the *MailServer* machine). The program will first ask for the username and password, and store it locally. It will then ask for options from the user and wait for user input from the keyboard. The program should support the following 3 options:

1. *Manage Mail* : Shows the stored mails of the logged in user only
2. *Send Mail* : Allows the user to send a mail
3. *Quit* : Quits the program.

The operations to be done at each option are described below (except 3, which just quits the program).

If the user chooses option 2:

The program will open a connection to the process R in the *MailServer* machine. It will then use the SMTP protocol to communicate with R and send the mail to the mail server machine.

The user should enter the mail to be sent in exactly the following format:

From: <username>@<domain name>
To: <username>@<domain name>
Subject: <subject string, max 50 characters>
<Message body – one or more lines, terminated by a final line with only a fullstop character>

Wherever a space character is shown, one or more spaces can be there. You can assume that no single line in the message body has more than 80 characters and the maximum no. of lines in the message is 50. Also, you can assume that the *To* line will not contain more than one email address.

An example typed by a user can be

From: agupta@10.5.20.22
To: ng@10.5.21.32
Subject: This week's lab assignment
This is the first week after midsem.
The students are eager to do assignments (ha ha ha!)
Let us give them something interesting.
.

Note the single fullstop, immediately followed by an <Enter> at the last line.

On getting the complete message, the process first checks the format of the message. The following checks must be done:

1. The *From*, *To*, and *Subject* field must be there, in that order and in proper format. The message body can be empty (just the fullstop at the last line).
2. The format for the *From* and *To* fields must be *X@Y*.

If the format is not correct, then "Incorrect format" is printed, and the three options are given again. The entire mail has to be entered again, there is no editing facility.

If the format is correct, the process sends it to R on the *MailServer* using SMTP command-responses (described later). If the mail is sent successfully, the client process prints the message "Mail sent successfully" on the screen and the three options are shown again If there is an error (shown by one of the reply messages in SMTP), the message "Error in sending mail: <Actual SMTP reply>" is sent back by R, and is printed on the screen, where <Actual SMTP reply> is the SMTP error code and message returned from the receiver (see below).

SMTP works on simple text commands/replies that are sent across the TCP connection between the sender and the receiver. The sender's commands that we will implement are the following:

HELO, MAIL, RCPT, DATA, QUIT

Each command may also take a parameter. The exact syntax of these messages and their descriptions are given in RFC 821, in Stallings book (Ch.22).

The reply code/error codes/messages that we will implement (with small explanation beside each)

220 <domain name> Service Ready *//given by receiver when TCP conn. is accepted*
221 <domain> Service closing transmission channel *// by receiver in response to QUIT*
250 OK <message> *// by receiver on success*
354 Start mail input; end with <CRLF>.<CRLF> *// by receiver, on DATA comm..*
550 No such user *// by receiver, on RCPT, if no user by that name*

There are a lot more codes, but this will suffice for us. The command formats and the reply codes are specified in RFC 821, though the messages following the reply code in the reply can be system specific. Please use the messages specified above. For 250 OK, use a descriptive message for the appropriate case (see example below).

A typical ordering of the commands in a mail session between the sender and the receiver is shown below (with comments added that are not actually part of the session):

C: <client connects to SMTP port>
S: 220 <iitkgp.edu> Service ready
C: HELO iitkgp.edu // *sending host identifies self*
S: 250 OK Hello iitkgp.edu // *receiver acknowledges*
C: MAIL FROM: <ag@iitkgp.edu> // *identify sending user*
S: 250 <ag@iitkgp.edu>... Sender ok // *receiver acknowledges*
C: RCPT TO: gb@iitkgp.edu // *identify target user*
S: 250 root... Recipient ok *// receiver acknowledges*
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: From: ag@iitkgp.edu
C: To: gb@iitkgp.edu
C: Subject: Test mail
C: This is a test mail.
C: How are you doing?
C: . // *end of multiline send*
S: 250 OK Message accepted for delivery
C: QUIT // *sender signs off*
S: 221 iitkgp.edu closing connection // *receiver disconnects*
C: <client hangs up>

The above serves as an example of what will be exchanged over the TCP connection during the mail transfer.

Note that the above implementation does not require any authentication. In practice, SMTP servers will require authentication. We will add the authentication commands later after the basic setup is complete.

If the user chooses option 1:

In this case, first a list of the mails in the user's *mymailbox* file is shown by the program on the screen in the following format:

Sl. No. <Sender's email id> <When received, in date : hour : minute> <Subject>

The Sl. No. is the serial no. of the mail in the *mymailbox* file. The program then gives a prompt "Enter mail no. to see:" and waits for the user to enter a number on the screen. If the number entered is –1, the program goes back to the main menu (the three options). If the number entered is

out of range, an error message “Mail no. out of range, give again” is printed and the user is prompted to give the number of the mail to read again. If the user enters a valid mail number, that mail (the entire content including *From*, *To*, *Subject*, *Received*, and message body) is shown on the screen. The program waits on a *getchar()* after showing the mail. If the character is ‘d’, the mail is deleted. Otherwise, it returns to show the list of emails again when the user hits any other character after reading the mail.

Note that the mails are stored in the *MailServer* machine. To read and delete them, the program will need to communicate with the POP3 server running on the *MailServer* machine. This will be done by opening a connection to the POP3 server and using the POP3 protocol. You will also have to see what POP3 commands have to be sent to show the mails and delete them as required above.

Similar to SMTP, commands in POP3 consist of a text keyword, possibly followed by one or more text arguments. All commands are terminated by a CRLF pair. Keywords are three or four characters long. Each argument may be up to 40 characters long. Responses in the POP3 consist of a status indicator and a keyword possibly followed by additional information. All responses are terminated by a CRLF pair. There are currently two status indicators: positive ("+OK") and negative ("-ERR"). Servers MUST send the "+OK" and "-ERR" in upper case. Text information sent after the status indicators can be system-dependent, but should be meaningful (as in SMTP).

Responses to certain commands are multi-line. In these cases, after sending the first line of the response and a CRLF, any additional lines are sent, each terminated by a CRLF pair. When all lines of the response have been sent, a final line is sent, consisting of just a "." and a CRLF pair. You can assume that no other line in the response will have just the “.” in it.

Once the TCP connection to the POP3 server has been opened, the POP3 server sends a greeting message (say “+OK POP3 server ready”). The server is now in the AUTHORIZATION state. Similar to SMTP, the client first sends the username and password using the USER and PASS commands. These are checked by the POP3 server, and +OK or –ERR is returned as appropriate.

Once the client has successfully logged in, it requests actions on the part of the POP3 server by sending commands. When the client issues the QUIT command, the server cleans up any resources used by the connection, sends the message “goodbye” and closes the connection.

POP3 DETAILS

Once the TCP connection has been opened by a POP3 client, the POP3 server issues a one line greeting. This can be any positive response. An example might be:

S: +OK POP3 server ready

The POP3 session is now in the AUTHORIZATION state. The client must now identify and authenticate itself to the POP3 server using the USER and PASS command combination.

Once the POP3 server has determined that the client should be given access to the appropriate mailbox, the POP3 server then acquires an exclusive-access lock on the mailbox, as necessary to prevent messages from being modified or removed before the session enters the UPDATE state. If the lock is successfully acquired, the POP3 server responds with +OK. The POP3 session now enters the TRANSACTION state, with no messages marked as deleted. If the mailbox cannot be opened for some reason (for example, a lock can't be acquired, the client is denied access to the appropriate mailbox, or the mailbox cannot be parsed), the POP3 server responds with -ERR. (If a lock was acquired but the POP3 server intends to respond with-ERR, the POP3 server must release the lock prior to rejecting the command.) If a –ERR is sent, the server closes the connection.

After the POP3 server has opened the mailbox, it assigns a message-number to each message, and notes the size of each message in octets. The first message in the mailbox is assigned a message-number of "1", the second is assigned "2", and so on, so that the nth message in a mailbox is assigned a message-number of "n". In POP3 commands and responses, all message-numbers and message sizes are expressed in decimal.

Once the client has successfully identified itself to the POP3 server and the POP3 server has locked and opened the appropriate mailbox, the client may issue any of the following POP3 commands repeatedly. After each command, the POP3 server issues a response. Eventually, the client issues the QUIT command and the POP3 session enters the UPDATE state.

Here are the POP3 commands valid in the TRANSACTION state:

STAT , LIST, RETR, DELE, RSET [for details see RFC 1939]

When the client issues the QUIT command, the server checks if there are any messages marked to be deleted. If yes, the messages are deleted. If all marked messages are deleted, +OK is sent, otherwise –ERR is sent. The server then sends a "goodbye" message, releases locks and any other resources, and closes the connection.

What To Submit:

Submit the following C files:

<ROLLNUM>*_smtpmail.c,* ***<ROLLNUM>_ popserver.c,*** ***<ROLLNUM>_mailclient.c***

Submission Details:

1.Please read the questions carefully and complete it.

2. Make a directory with name <Your_Roll_Number> and copy your all program (source code)

and output file to that folder.

3. You can use C or Cpp for this assignment

4. Please copy your output and paste it to related program at the end of source code(please comment it)/ if necessary you can take screen shot name it with its question number and put it in a same folder.

5. Test well before submission. Follow some coding style uniformly. Provide proper comments in your code.

6. After completing all the tasks please show it to TA's. Once the result is correct then only we allow you to upload to the moodle.

7. Submit only through moodle and well in advance. Any hiccups in the moodle/Internet at the last minute is never acceptable as an excuse for late submission. Submissions through email will be ignored.

8. Please zip your folder and submit to moodle within 19-02-2020 (5:30 PM).