Practical Assignment-1: Design and Implementation of Mail Server and Client

---

1. Assignment Introduction

The goal of this assignment is to learn how to use SMTP and POP3 to send and receive emails from a remote host. You will use the Simple Mail Transfer Protocol (SMTP) and Post Office Protocol 3 (POP3) 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 via an email client.

A TCP connection is established between two SMTP servers to transmit and receive emails. The sender establishes the connection and then delivers a series of text commands to the recipient. There are additional SMTP details in [1,2] and RFC 821 [3]. You will implement a subset of the SMTP command/replies in this project.

POP3 is a protocol for managing a mailbox and retrieving emails. However, POP3 does not deal with sending email. As with SMTP, POP3 uses text commands over a TCP connection. A sequence diagram of the protocol with more details are available in [4,5] and RFC 1939 [6]. A POP3 client and server will be implemented in this project.

The mail server will host both the SMTP and POP3 servers. You can access your mailbox using the POP3 client on your home client computer, and you can use the SMTP server to send emails using the SMTP client. 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 ***HomeClient***.

All servers should support concurrent client connections. Consider scenarios in which many users connect to the same mail server to access their mailboxes, or numerous mail servers deliver mail to the same mail server simultaneously.

2. Assignment Task Description

2.1 Task 1 – MailServer

The **MailServer** will do two things:

1) Run an **SMTP mail server** to accept and store emails from other mail clients or servers using SMTP.
2) Run a **POP3 server** to provide mailbox management and access from the **HomeClient**.

But first, we need to set up a few things. Make a file called ***userinfo.txt*** in the directory where the software will be run. On each line of this file, there are one or more spaces between a user's login name and password (recommend one-word). Make the file so that it has one user for each member of the group. Then, for each user in the same directory, make a

subfolder with the same name as the user. Each user's mailboxes will be kept in the subdirectories that go with them.

Program ***mailserver_stmp.py*** completes task 1) above. The program will accept the integer argument ***my_port*** on the command line to provide the port. The received message is then added to a file called "***my_mailbox***" in the user's subfolder when it has been received (remember that the subdirectories are created manually). The mail address will contain the username. The mail is attached in the manner described below:

From: <username>@<domain_name>
To: <username>@<domain_name>
Subject: <subject string, max 150 characters>
Received: <time at which received, in date : hour : minute>
<Message body – one or more lines>.

Take note of the new Received line. As a result, at any given time, the *my_mailbox* file includes 0 or more of these messages, each of which is separated by a full stop (only the full stop character in one line).

To do task 2) above, you will write a program call ***popserver.py***, which will be the POP3 server. The program will accept the integer *POP3_port* as a command-line input to specify the port on which the POP3 server will be running. More POP3 information is provided later.

2.2 Task 2 – HomeClient

On the **HomeClient** machine, sending and getting emails will be done by a program called ***mail_client.py***. The program will accept the *server_IP* address command line option, which tells it where to connect to the SMTP and POP3 servers (i.e., the IP address of the **MailServer** machine). Before saving the login and password locally, the program will first ask for them. After that, it will ask the user to choose and wait for a response from the keyboard. The program lets you choose from the following three options:

a) **Mail Sending**: Enables sending emails by the user
b) **Mail Management**: Perform a series of operations on the mail in the mailbox (such as delete, display the number of mails, retrieve mail, etc.)
c) **Exit**: Exits the application.

Here is a description of option a) and b).

Selecting option "Mail Sending":

The user should type the message precisely as it is shown below:

From: <username>@<domain name>
To: <username>@<domain name>
Subject: <subject string, max 150 characters>
<Message body – one or more lines, terminated by a final line with only a full stop character>

The process examines the message's format after receiving the entire message. If the format is not correct, then "This is an incorrect format" is printed, and the three options are given again. If the format is correct, client process prints the message "Mail sent successfully" on the screen and displays the three options once again if the mail was sent successfully. If the format is incorrect, the client give corresponding error information.

Keep in mind that mail is kept on the **MailServer** machine. The program must interface with the **POP3 server** running on the **MailServer** machine in order to read and delete them. By establishing a connection with a POP3 server and utilizing the POP3 protocol, this will be accomplished. You must also determine which POP3 instructions must be delivered in order to show and remove the message as previously mentioned.

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_FROM, RCPT_TO, DATA, QUIT.

And each command has an argument it can accept.

Then, the reply codes/error codes/messages we will implement (see below).

```
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
```

A typical sequence of commands in a mail conversation between sender and receiver is as follows (the comments are not part of the conversation, they are for explanation only).

```
Cliend: <client connects to SMTP port>
Server: 220 <kuleuven.be> Service ready
Cliend: HELO kuleuven.be // sending host identifies self
Server: 250 OK Hello kuleuven.be // receiver acknowledges
Cliend: MAIL_FROM: <cnlab@kuleuven.be> // identify sending user
Server: 250 <cnlab@kuleuven.be>... Sender ok // receiver acknowledges
Cliend: RCPT_TO: cn@kuleuven.be // identify target user
Server: 250 root... Recipient ok // receiver acknowledges
Cliend: DATA
Server: 354 Enter mail, end with "." on a line by itself
Cliend: From: cnlab@kuleuven.be
Cliend: To: cn@kuleuven.be
Cliend: Subject: Test
Cliend: This is a test mail.
Cliend: Are you going well?
Cliend: . // end of multiline send
Server: 250 OK Message accepted for delivery
Cliend: QUIT // sender signs off
Server: 221 kuleuven.be closing connection // receiver disconnects
Cliend: <client hangs up>
```

Selecting option "Mail Management":

When user selects option b), the Client program should perform validation of the username and password on the **POP3 Server** by passing "USER" and "PASS". If authentication is unsuccessful, an error message should be displayed along with a prompt user to re-enter the credentials. If authentication is successful, a connection should be established between the client and POP3 server and the client should receive and display a greeting message from the server (e.g., +OK POP3 server is ready)

Then, the program receives a list of emails from the POP3 server for the authenticated user and displays it on the console in the format below:

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

The "No." is the number of the mail in the *my_mailbox* file. The program provides the

following POP3 commands:

STAT, LIST, RETR, DELE, RSET.

STAT – Count the number of emails.
LIST – Should list all the email for the user
RETR – Retrieve email based on the serial number
DELE – Should delete the email based on the serial number
QUIT – Close the connection and terminate the program

Here, the user should provide multiple options in the terminal to enter and accordingly manage emails. Each of these options will correspond to the commands mentioned above.

When the client sends the QUIT command, the server closes the connection, delivers a "goodbye" message (e.g., "Thanks for using POP3 server"), and cleans up any resources utilized by the connection.

Recommended Testing environment:

For testing purposes, you can also use the minimal SMTP server smtp4dev [7] along with these Mailosour [8] website to create fake emails for testing purposes.

3. Practical Guidelines

The assignment can be done in groups of two, however, you will be individually graded. Your grade will be determined based on the demonstration of your submitted project and a Q&A session, whose details will follow later. The submission deadline is due to **7PM on 17th of March.** Any changes you make beyond the deadline will not be considered. We expect every student to be able to defend the code that they submitted.

Please enter your full name, r-number and group number (Group_x) in this Google Sheet before **10 March**. Note: Even if you are alone, you also need fill in the group number.

4.1 Submission Details:

Your work should be submitted through this Github_classroom with the name **<Group_your goup No.x>**; It is expected that one member of the group will initiate the creation of the group repository, and subsequently, the other members can join the repository by referencing the assigned group number.

Run **git commit -a** and **git push**. You are allowed to make multiple submissions but the last one before the deadline only will be considered.

1. The structure of the repository should be as follow:

- ***mailserver_stmp.py***
- ***pop_server.py***
- ***mail_client.py***
- ***userinfo.txt***

- ***my_mailbox.txt***

2. Make sure to commit and changes to your branch. And copy your all program (source code) and output file to that branch.

3. You will implement this assignment using sockets programming in **Python v3.x**.

4. Please copy your input and output and paste them at the end of the source code (please comment it)/ if necessary, take a screen shot and name it with the question number and save it in the same folder.

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

Note*: Branch name, repository structure, file names* should be respected, or the submission will not be considered.

5 Marking Specifications

1. Correctness of the email client implementation (6 marks)
2. Correctness of the SMTP server implementation (6 marks)
3. Correctness of the POP3 server implementation (5 marks)
4. Outstanding code quality and implementation (3 marks)

Total Available Marks = 20 marks

All software implementation (1-3) will be assessed based upon standard test-cases and evaluation of code-quality. Code style and norm for each part will also be taken into consideration in scoring.

6 References:

[1]. https://www.scaler.com/topics/computer-network/smtp-protocol/
[2]. https://www.geeksforgeeks.org/simple-mail-transfer-protocol-smtp/
[3]. https://www.rfc-editor.org/rfc/rfc821
[4]. https://www.yumpu.com/en/document/read/38291989/pop3-tutorial-sequence-diagram-eventhelixcom
[5]. https://www.techtarget.com/whatis/definition/POP3-Post-Office-Protocol-3
[6]. https://www.rfc-editor.org/info/rfc1939
[7]. https://github.com/rnwood/smtp4dev
[8]. https://mailosaur.com/blog/setting-up-a-fake-smtp-server-for-testing/