Computer Networks Programming Assignment 1

Fall 2023

Due: September 29, 2023, 11:59 PM

In this project, you will write two simple programs: One acts like a basic POP3 server and the other acts like a simple POP3 client. Your programs do not need to implement the full POP3 specification, only the behaviors described in Section 1 and 2.

1 The Setup

The server program should run forever, listening for **TCP** requests. When the request arrives, a conversation between the POP server and the POP client will occur (you will learn socket programming in Module 6). You only need to implement the list of POP3 commands described in Section 2. After the `QUIT` command, the TCP connection should be closed and the server returns to listening for new connections.

The client will be an interactive program, rather than the "behind the scenes" behavior of a real email client. The user should be prompted to type any of the commands listed in Section 2.

There are three plaintext (RFC 5322) emails provided with the project description and you will simply assume there are all the email messages stored in the server. Upon starting up, the server should detect that these files exist and respond to the client's queries based on this information. For example, if the client issues the `LIST` command, the server should report those three emails and their respective sizes. A `DELE` command should have the effect of truly deleting that message from the "store" (i.e., deleting the file). If I fire up the client again, the deleted messages should not be available for download again. After a session communicating with the server, the client should exit.

2 Some Details

The server should be listening on a port number that is passed as a command-line argument (e.g., `./server 5000`). You can use any port number large than 1024 and smaller than 65536. Also, please make sure the port number is a parameter input into the server program, and the grader can change it. If you are developing on MTU machines, there is a possibility that two servers may try to use the same port. To reduce this possibility,

use the last four digits of your M-number (provided they are greater than 1024). This has historically worked pretty well.

The client should be passed with the IP address and port of the server as command line arguments (e.g., `./client localhost 5000`). The program should accept `localhost` or a numeric IP address. Please make sure the port number should be a parameter input into the client program, and the grader can change it.

The server should support the following POP3 commands issued by the client (refer to `https://datatracker.ietf.org/doc/html/rfc1081` for the detailed descriptions of each command):

- STAT: The server will send back the total number of email messages stored.
- LIST: The server will list out all the email messages stored, including the number and the size of each message in bytes. The message size should the sum of both the email header and the email body.
- RETR <X>: The server will send back email message with number X. It is up to you to display the email correctly in the console of the client. For simplicity, your client can simply receive the emails sent by the server, and store them locally in the same folder (if you don't display the received emails in the client, please output "the client has successfully received the email, and stored it locally").
- DELE <X>: The server will delete email message with number X locally.
- TOP <X> <n>: The server will send back the header and the first n lines of the body of the email message X. Note that if the number of lines requested by the POP3 client is greater than the number of lines in the body, then the server sends the entire message. Similar to the RETR, your client can simply receive the partial emails sent by the server, and store them locally in the same folder (if you don't display the received partial emails in the client, please output "the client has successfully received the partial email, and stored it locally").
- QUIT: The server will terminate the conversation with the client.

In addition, each response from the server should start with either +OK or -ERR (check out the examples in Module 5).

3 Deliverables

You will be submitting the source code for the two programs. They can be written in C or Python 3. If you choose to use Python 3 and somehow feel the need for some fancy package, explicitly indicate in your README file about how to install that. No "notebooks" (e.g., Jupyter) are permitted.

If you choose to develop in C, you should provide a makefile (or at least a shell which can be used to easily compile your program).

Be sure to include the test email files provided with your submission. You can simply place the test email files in the same folder as your source code files. The grader will deduct 5 points from your final score if he/she has to copy those files to test your code. You should provide a README file which describes how to test your programs. If any special instructions are needed for compilation or running, please describe those in your README file.

It is preferred that all files be zipped up and submitted as a single package. Tarballs, Gzipped and "plain" Zip files are great. Please name the package in the format as "your last name-your first name-P1".

4 Grading

Each program is worth 50 points. The grader will test all of the commands listed in section 2, which will be worth 8 points each. 2 points per program are just for successful compilation/execution of code.

5 Collaboration Policy

This is an individual project. No collaboration is allowed.