COMP 431 — INTERNET SERVICES & PROTOCOLS

Spring 2020

Homework 3, Feb 1

Due: 10:30 AM, Feb 15

Note: The TAs have prepared an initial code for the client that implements most of the description highlighted in yellow in this document. You would need to extend the client to implement the rest of the functionality. Instructions to access the initial code are available at the course web page.

Building an SMTP Client/Server System Using Sockets

In this assignment you will write two programs. The first will extend and convert your SMTP "server" program (HW 2) into a "real" SMTP server. The second will extend and convert the SMTP "client" program provided by the TAs (SMTP2.py) into a real user mail agent and SMTP client. And as real clients and servers, your HW 3 programs will interoperate over a network by using a TCP socket.

To do this you will need to:

- Implement the final portions of the SMTP protocol,
- Replace the `stdin`/`stdout` I/O for SMTP commands with socket I/O, and
- Do some additional message processing and formatting in the SMTP client.

The final portions of the SMTP protocol are the generation of a greeting message by the server upon receipt of a connection, and the generation of the HELO message by the client in response to the server's greeting message.

SMTP Server Program

Your SMTP server program requires only a single command line argument: the port number on which it should listen for connections from clients. When your server accepts a connection from a client, in accordance with the SMTP protocol, it should generate and send the SMTP server "greeting" message (remember that SMTP is a unique protocol in that the server "speaks" first):

```
220 Random "greeting" text that includes the name of the server
```

The "220" return code is required; the text of the greeting message is arbitrary. However, it is traditional for the greeting text to specify the hostname (or at least the domain) of the server. For this assignment your program should generate the greeting message:

`220` *`hostname`*`.cs.unc.edu`

where *hostname* is the name of the machine that is running your server (*i.e.*, most likely *comp431-1sp20* or *comp431-1sp20b*). After sending this message the server must receive and acknowledge a valid SMTP HELO command from the client before proceeding with mail processing (see the following section). The HELO message is acknowledged by the server with a 250 message. By convention, the text of the acknowledgement of the HELO message echoes the text of the HELO message and includes the phrase "pleased to meet you." If the server receives an invalid HELO message (as defined by the grammar below) it should emit either a 500 or a 501 error as appropriate.

For example, a sample set of "traditional" messages sent at the start of an SMTP session between an SMTP server running on *classroom.cs.unc.edu* and a client running on *snapper.cs.unc.edu* might look like the following:

```
220 classroom.cs.unc.edu
HELO snapper.cs.unc.edu
250 Hello snapper.cs.unc.edu pleased to meet you
```

Once the HELO command has been received (the formal grammar for the HELO command is given below) and acknowledged, the server enters its message processing loop and begins processing messages until the QUIT command is received.

After receiving a QUIT command, the server should send the following response to the client:

```
221 <server hostname> closing connection
```

where "`<server hostname>`" is just the name of the host on which the server is running (printed without the angle brackets). After emitting this response, the server should close its socket to the client and return to its initial state and await the next connection from a client.

As in Homework 1, when the server processes mail messages, it should append each message that it receives into a "forward-file" (in the subdirectory "*forward*" which you can assume already exists). However, different from Homework 1, the forward-file filename should be the domain name of the receiver as specified in the RCPT TO field of the SMTP command. Thus, for example, if a message is received by the server addressed to *jeffay@cs.unc.edu* (*i.e.,* the server received the command "RCPT TO: <jeffay@cs.unc.edu>"), then the message should be appended to the file *forward/cs.unc.edu* in the current working directory. Each message appended to a forward file should only contain the SMTP commands associated with that mail transaction (in particular, HELO and QUIT commands should never be written to forward-files).

Your server should deal with any protocol errors it encounters as your HW1 program did. That is, if your server encounters a protocol error in some message it receives from the client, your server program should generate the appropriate 500, 501, or 503 response message. However, now your server should send the error response message to the client via the connection socket and not print the message to standard output. If your server encounters any non-protocol error from which it cannot recover, the server should close its connection to the client (if a connection existed), print to standard output a meaningful 1-line error message, and return to its initial state and await the next connection from a new client. If an error occurs when creating the welcoming socket simply print a meaningful 1-line error message and terminate your program.

Your server program should echo (print) all of the message it receives from the client program to standard output. Other than these messages, your server program will perform *all* of its protocol I/O to a socket. In particular, it should not interact with the user directly in any way. Specifically, it should not output prompts ~~or other messages~~ (other than non-protocol error messages) and it should not read commands or other inputs from the keyboard or a file.

The server program, like most servers, will conceptually never terminate. It will be terminated by some external means such as typing *control-C* in the shell.

User Mail Agent/SMTP Client Program

Your second program will be both a simple user mail agent and an SMTP client. This mail agent program will interactively allow a user to compose and "send" an email message using a command line interface (just like email clients from the 1980s!) – the code provided by the TAs named "STMP2.py" implements a lot of the user interaction logic (most of the text highlighted in yellow below is already implemented by this code).

Your mail agent should take two command line arguments: a hostname and a port number. The hostname is the name of the host where the SMTP server that the program will use is running (*e.g.*, *snapper.cs.unc.edu* or *classroom.cs.unc.edu*). The port number specifies the port number on which your SMTP server is listening for connections. (As explained below, for this assignment you will *not* be using the reserved port numbers for SMTP.) This port number should be the same as the corresponding port number parameter given to your server program.

When your client program starts executing, it will create a TCP socket to the SMTP server at the host and port number specified on the command line. This connection should persist until the client program sends a QUIT command (discussed below) to the server and the server's response is received. As with the server program, all messages read from this socket should be echoed (printed) to standard output.

Your mail agent should run a loop that generates text prompt messages to standard output to prompt the user to enter in the contents of an email message. Specifically, in each loop iteration, there should be a prompt message for each of the message's main components generated in the following order:

From:
To:
Subject:
Message:

The From/To fields will be used to get the sender/recipient information from the user.

For example, if a user used your mail agent to compose a message, the following outputs and inputs would be seen in your terminal window (the bold text indicates outputs generated by your program and the non-bold text indicates inputs typed in by the user):

```
From:
jasleen@cs.unc.edu
To:
smithfd@cs.unc.edu
Subject:
How the heck are we going to grade HW4?!
Message:
Don – How are we going to grade HW4? It's going to require executing two programs for each student on
two separate machines and this is going to be difficult to automate
.
```

Assume users will terminate their message text by entering in a period on an otherwise blank line. (Note the period on the blank line in the example above.)

After the user has entered in a valid email message, your mail agent will ~~create a TCP socket to the SMTP server at the host and port number specified on the command line, and~~ forward the user's message to the server using the SMTP protocol. To do this, there are four new operations that must be added to the SMTP client portion of your mail agent program. First, when your program now connects to the server it must be prepared to receive a correct server greeting message. (Your program will do nothing with the greeting message other than receive it.) Your program should reply to the greeting with the SMTP HELO message whose format is given by the grammar:

```
HELO <whitespace> <domain> <nullspace> <CRLF>
```

where `domain` is simply the domain name of the machine on which you are running your mail agent program (*e.g.*, *cs.unc.edu*). The syntax for the domain name should be consistent with the non-terminal `<domain>` described in Homework 1. (Note that many SMTP clients use a hostname rather than a domain name but technically the protocol specifies the use of a domain name and syntactically a host name is the same as a domain name.)

Second, when your client sends the user's message to the server (via the DATA command), your client should format the message as follows (and as described in RFC 822):

From: *reverse-path (with angle brackets)*
To: *forward path (with angle brackets)*

Subject: *Text of the subject line formatted exactly as typed in by the user*

Text of the message formatted exactly as typed in by the user.

Notice that in this message format, the email headers are separated from the body of the email message by a blank line. Note also that this is the format for the message and the headers here are different from the SMTP protocol commands.

For example, if I wanted to send to Prof Smith a one-line email message with the text "This is a test," then ultimately, your mail agent would send the following commands and data (lines of message text) to the SMTP server (after the SMTP "HELO" handshaking):

```
MAIL FROM: <jasleen@cs.unc.edu>
RCPT TO: <smithfd@cs.unc.edu>
DATA
From: <jasleen@cs.unc.edu>
To: <smithfd@cs.unc.edu>
Subject: Test message

This is a test.
.
QUIT
```

Third, your mail agent program will terminate when either end-of-file is reached on standard input, or when any SMTP or socket error is encountered (*e.g.*, errors opening the socket, errors reading from or writing to the socket, SMTP protocol errors encountered, *etc*.). If your client encounters an error it should close its connection to the server and print to standard output a meaningful 1-line error message before terminating.

Lastly, notice that when your client emits the QUIT command, the client now has to be prepared to receive the server's final 221 "connection closed" response.

In total, the dialog between your client and server programs should be as shown in the SMTP examples shown in the textbook and lecture notes (although the text of the response messages are allowed to vary).

Notes on Picking a Port Number For Your Socket

For this assignment the client and server programs will execute on different computers. You should run your server on *comp431-1sp20.cs.unc.edu* and your mail agent/client on *comp431-1sp20b.cs.unc.edu* (or vice versa). The port number you should use for your server is 8000 + the last 4 digits of your PID (that is, numerically add the number 8000 to the number formed by the last 4 digits of your PID). This will minimize the possibilities of a port conflict between two students trying to run their server on the same computer using the same port number. However, it will not guarantee that port conflicts do not occur! Thus, your program *must* be prepared to deal with errors that occur when trying to create a socket on a port number that is in use by someone else (and generally deal with any errors that occur on any socket operation).

Grading

As per the instructions for HW 1, you will turn in your program by placing it in a special directory on the course server. Once again, you should perform the steps below precisely and in *exactly* the order listed. WARNING: Failure to follow these steps exactly will result in the TA being unable to read our files. Should this occur, you will receive a grade of "0" for the assignment!

- Log onto the Linux server named comp431-1sp20.cs.unc.edu (using your onyen login and password) and in your home directory, create the directory comp431/submissions/hw3.
- In this hw3 directory, you should place your final client program ("Client.py") and your final server program ("Server.py"). Please include any "helper" classes you write inside of your client and server programs so that we only have to execute these two programs to test your code.

- Do not change any of your files for this assignment after the submission deadline! We will use the file timestamps to determine the lateness of your assignments, so if these timestamps change after the submission deadline, you will be penalized for turning in a late assignment.
- We will test your programs on the course servers (comp431-1sp20.cs.unc.edu and comp431-1sp20b.cs.unc.edu) using python3. It is your responsibility to test your code and ensure the programs work properly in this environment.
- As before, your programs should be neatly formatted (*i.e.*, easy to read) and well documented.
- The homework grade will have the following distribution:
  - 20% Valid input processing (client)
  - 20% Valid input processing (server)
  - 20% 500 error processing (server)
  - 20% 501 error processing (server)
  - 20% 503 error processing (server)
  - 20% Extra Credit: MIME Processing

Interoperating with other COMP 431 students' programs

Before submitting your client and server programs for grading, test them for interoperation with the client and server programs (both combinations) of ***at least two*** other COMP 431 students from the Spring 2020 semester. Use test input files from your testing partners as well as your own in performing these tests. Use Gradescope to submit the onyen-ids of the students you tested with.

Extra Credit — MIME Processing

For extra credit write a second user mail agent/SMTP client program (name this program *ClientEC.py*) that will allow a user to attach a file to their email. In addition, you will make your second email user agent work with the Department's SMTP server *smtp.cs.unc.edu*. You should not attempt any aspect of the extra credit portion of this assignment until you're *certain* that your *Client.py* and *Server.py* programs are *completely* correct. Place the program *ClientEC.py* in the HW3 directory.

After reading in an email message from the user, your second mail-agent program will prompt the user for the name of a file (a Linux file path) to attach to the email. Thus, the sequence of prompts your extra credit client will generate for the user are:

```
From:
To:
Subject:
Message:
Attachment:
```

You may assume that when prompted for an attachment, the user will specify a valid file name. Once your client has the message contents and attachment file name, it will compose a multipart MIME email message containing the text of the message as the first part and the contents of the attachment MIME encoded as a binary file. (See the textbook, the lecture notes, or RFC 2045 and 2056 for the proper MIME formats.) Your SMTP client will then send the MIME encoded email message to an SMTP server, starting with your SMTP server program (for testing purposes).

Once you have your *ClientEC* program working correctly, test it further by using your new mail agent program to send a MIME encoded email message to yourself (and only yourself) at your *cs.unc.edu* email address (and only this address) by using the Department's SMTP server *smtp.cs.unc.edu*. Note that in order to use the Departmental SMTP server your mail agent program must be running on a Departmental server such as *classroom*. You will not to be able to connect to our SMTP server from your laptop.

If your program is working correctly you should be able read the message you sent using a standard POP/IMAP client (*i.e.*, using whatever program you currently use for reading your email). The message should appear as a text message with a file attachment.

<u>REALLY IMPORTANT NOTE</u>: You can only send messages to either your HW 3 server program or *smtp.cs.unc.edu* and these messages <u>must</u> be sent from your *cs.unc.edu* email address and sent only to your *cs.unc.edu email* addresses. In particular, sending email to anyone other than yourself, the instructor, or the TA, or sending email that appears to be from someone other than yourself will be considered an Honor Code violation.

When you're "certain" your MIME client program is working correctly, send a single multipart email message to the TA and instructor (using our *cs.unc.edu* email addresses) that contains some polite text with a jpeg attachment of your favorite or most goofy image of you (or your pet if you're shy). This email has to be sent as a single message with multiple recipients listed on the "To:" field, and the email must contain a Subject: line.

<u>REALLY REALLY IMPORTANT NOTE</u>: To get extra credit for this assignment, you have ONLY ONE chance to send your multipart-MIME email message to the instructor and TA. Therefore, don't send us an email message until you are certain that your program works! If you send us multiple email messages only the first one received will be used to determine if you get extra credit.

Also, to be eligible for extra credit, this assignment must be submitted on-time. That is, if you submit the assignment after the due date and time, you CANNOT receive extra credit.