Jon Hearn
9/28/2018

Project 1 Report

Introduction

This project was over all a success. Though it took me longer than I wanted, I was able to simulate the SMTP protocol for sending emails and the HTTP protocol for receiving emails.

Design Choices

The first big design choice I made was to make this project out of four separate programs instead of two. This was made more out of ease than anything else. While the client could have had a choice to either send or receive, making one server that did both was out side the scope of what I was able to figure out in time.

The next big decision was made rather late in the game, and was to make the programs require the user to input proper protocol inputs. This was made due to the removal of Technical Requirement 10, which stated that the opposite should be done. Unsure whether this meant it was up to us to decide or if it was implied that they needed to have knowledge of the protocols to use the programs, I sided on the latter.

The third choice was also a last minute choice and was to keep the sending user in the server on a wrong command. Prior to a discussion about this in class, if the user entered in anything wrong they would need to relaunch the client program and start over. I rewrote the code to have a looping mechanism that kept the user inside, but also kept track of where in the command sequence they were.

In the SMTP server, I implemented the following return codes; 220, 221, 250, 354, 500, 503, and 555. These four success and three fail codes allowed me to express a large range of information while not attempting to implement all the possible variations of action codes.

For the HTTP server, I simply implemented the three required codes; 200, 400, and 404. I read through the other codes, and felt that no others were needed with the simplistic range that we were using the protocol for.

Sample output

As the client programs have the most useful output, they will be the two that I am showcasing.

```
./sendClient 146.163.112.151 8080
220 smtp.447f18.edu Mail Server Ready
HELO
250 Hello
MAIL FROM <me@447f18.edu>
250 OK
RCPT TO <you@447f18.edu>
250 OK
data
354 Send message content; end with <CRLF><CRLF>
this is a demo email. I am using this to showcase the abilities of the program.

and this is the end of the email.
```

```
250 OK, message accepted for delivary
helo
250 Hello
rcpt to
503 Wrong ordered command
mail from <me@447f18.edu>
250 OK
rcpt to <you@447f18>
555 Invalid email
garbage input
500 Nonrecognized command
rcpt to <you@447f18.edu>
250 OK
quit
221 Bye
```

```
./recvClient 146.163.112.151 8080
Greetings Master
GET /db/you/ HTTP 1.1
HOST: 146.163.112.151
COUNT: 10
HTTP/1.1 200 OK
```

Summary

I ran into many problems along the way. From teaching myself sockets for both TCP and UDP to bizarre errors where the programs would freeze for no foreseeable reasons. The latter being a large contributing factor to the lateness of this project. There were also problems with the UDP server “dropping” the last message and the client waiting on a response that never came, though that may be due to the “unreliability” of UDP. Over all, I liked the learning process for this project and look forward to the next one.