Robert Boyer
CSS 432
Winter 2017

Final Project

Documentation

1. **Client's interface in terms of OPEN, PASS/SYST, and QUIT**

   Open checks if connection already exists, if so stops continuation. If no connection exists, call connectToLogin(). This is done in two places: (1) first time while going through argument validation and (2) later if *'open'* ftp command is given by user. If the connectToLogin() function is called: an attempted connection is made to the hostname provided, user is prompted for username & password. The USER command + username + "\r\n", PASS command + password + "\r\n", and lastly SYST command + "\r\n" are all sent sequentially using client's sd and their responses are validated. The QUIT command + "\r\n" is sent if either *'close'* or *'quit'* ftp commands are given.

2. **Client's get function in terms of PASV, RETR, and local file options (O_CREAT | O_WRONLY and S_IRUSR | S_IWUSR |S_IRGRP | S_IROTH)**

   The get execution code is called when user has an established connection and the *'get'* ftp command is given. Passive mode is entered by sending PASV command with "/r/n" using client's sd, check response & use last 2 parts of IP address provided to compute the server port number to passively connect. Create new socket based on new connection and then connect.

   The RETR command + filename + "/r/n" is sent using client's sd. Response is checked and a file stream is created using the local file options: (O_CREAT | O_WRONLY and S_IRUSR | S_IWUSR |S_IRGRP | S_IROTH). A fork process is started where the child reads in from the from the server using the new socket and writes to disk before receiving next chunk of data. After all the data is received the response is checked and all the while the parent process waits for the child to be complete & then closes the connection.

3. **Client's cd function (i.e., CWD)**

   The get execution code is called when user has an established connection and the *'cd'* ftp command is given. The CWD command + sub directory + "/r/n" is sent using client's sd and response is checked.

4. **Client's ls function in terms of PASV, LIST, and file output to stdout**

   The get execution code is called when user has an established connection and the *'ls'* ftp command is given. First set Type to "I" (image, binary mode) is sent and response is checked. Then passive mode is entered by sending PASV command with "/r/n" using client's sd, check response & use last 2 parts of IP address provided to compute the server port number to passively connect. Create new socket based on new connection and then connect.

A fork process is started where the parent sends the LIST command with "r/n/" using client's sd, the response from the server is checked to see if data connection is open. The parent then waits the child process to finish polling data using newly created socket and appending data to string. After the child has received all the data, it prints the string and closes the connection. The parent then checks for a final response from the other connection to the server that the transfer was completed.

5. Client's put function in terms of PASV, STOR, and local file option (O_RDONLY)

The get execution code is called when user has an established connection and the *'put'* ftp command is given. An open file stream is created from the local file provided with the option (O_RDONLY). Type to "I" (image, binary mode) is sent and response is checked.

Then passive mode is entered by sending PASV command with "/r/n" using client's sd, check response & use last 2 parts of IP address provided to compute the server port number to passively connect. Create new socket based on new connection and then connect.

STOR command + filename + "/r/n" using client's sd is sent and response is checked. A fork process is started where the child gets the length of the file, reads the data as a block, and then transmits the buffer. After which it closes the connection and checks for a final response from the other connection to the server that the transfer was completed. The parent process waits for the entire time for the child process to complete.

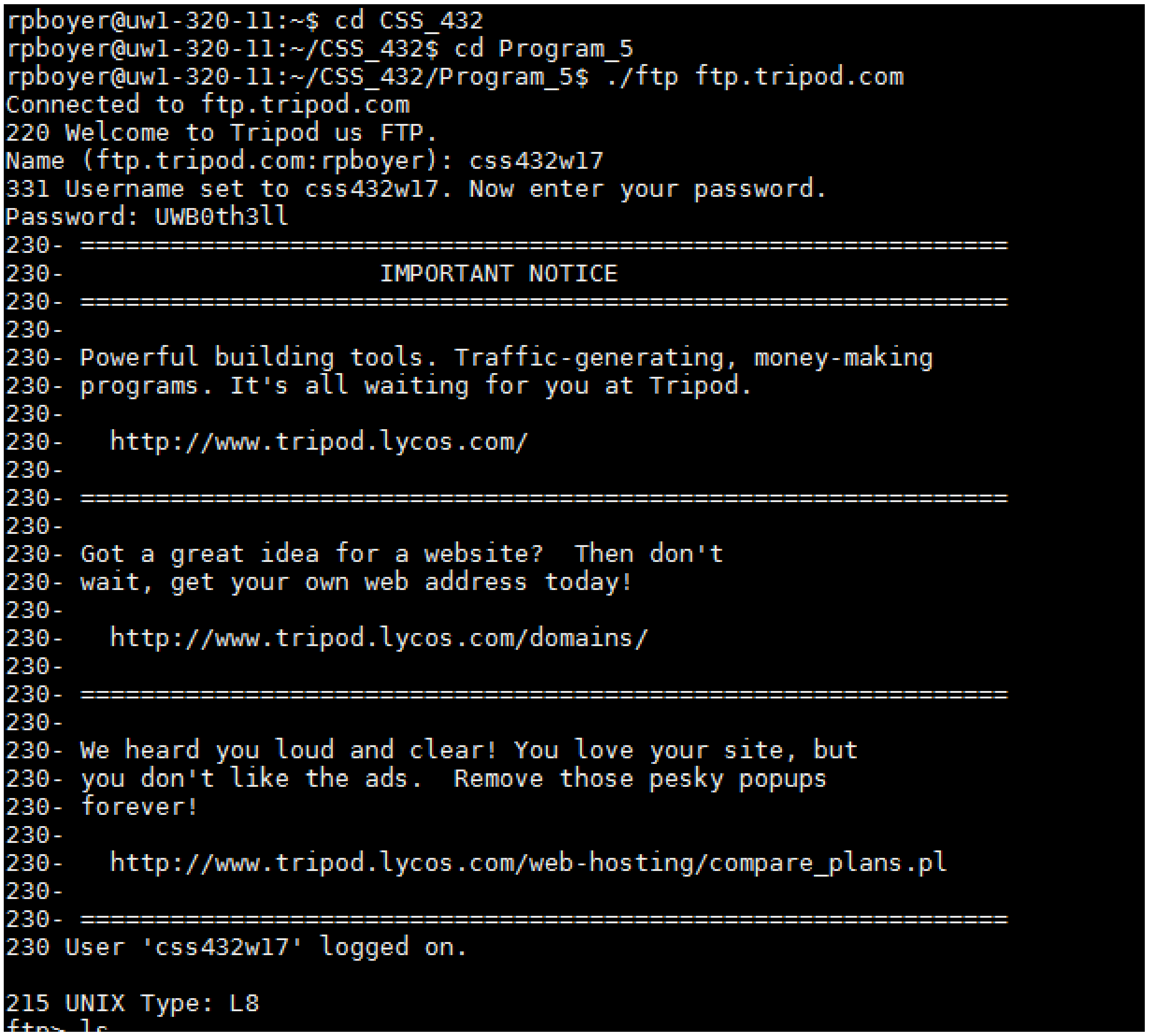
6. Limitations (how much you complied with RFC 959)

Output:

- I formatted so output looks like example in description document. Meaning I expect get + filename… not get by itself and then followed filename. For put I expect it then local file name, then remote file name. The output format will look weird if these guidelines aren't followed.
- Another issue with output is that I have two instances in which I have an extra space. I wasn't able to track down the specific source of the issue and fix but he two conditions are: (1) when logging in, after 230 – Important Notice section of intro code. (2) Whenever using the *'ls'* command.
- I also noticed that when using the 'ls' command sometimes instead of the expected order 227, 150, list, 226… I would see 227, list, 150, 226. This is because I cannot control which executes first list in the child or 150 in the parent. I only guarantee 226 is waited on by the parent until the child completes.

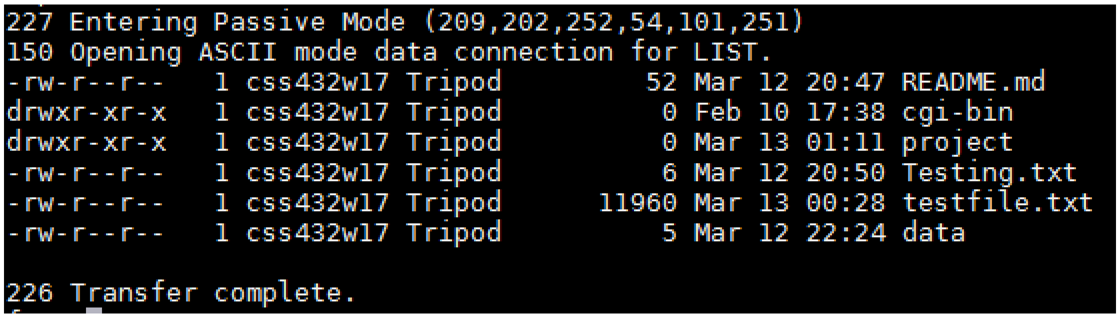
I didn't thoroughly test my code for 'normal' expected error situations. Example: not inputting actual characters for username and password. The instability when using ftp.tripod.com is quite normal for some reason and a 421 error frequently happens which often results in my program freaking out iterating the 421 message repeatedly. Please be aware of this and re-execute the code.

Execution Output



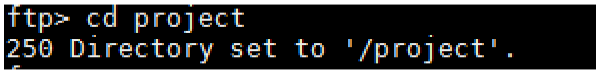
```
rpboyer@uw1-320-11:~$ cd CSS_432
rpboyer@uw1-320-11:~/CSS_432$ cd Program_5
rpboyer@uw1-320-11:~/CSS_432/Program_5$ ./ftp ftp.tripod.com
Connected to ftp.tripod.com
220 Welcome to Tripod us FTP.
Name (ftp.tripod.com:rpboyer): css432w17
331 Username set to css432w17. Now enter your password.
Password: UWB0th3ll
230- ==============================================================
230-                     IMPORTANT NOTICE
230- ==============================================================
230-
230- Powerful building tools. Traffic-generating, money-making
230- programs. It's all waiting for you at Tripod.
230-
230-   http://www.tripod.lycos.com/
230-
230- ==============================================================
230-
230- Got a great idea for a website?  Then don't
230- wait, get your own web address today!
230-
230-   http://www.tripod.lycos.com/domains/
230-
230- ==============================================================
230-
230- We heard you loud and clear! You love your site, but
230- you don't like the ads.  Remove those pesky popups
230- forever!
230-
230-   http://www.tripod.lycos.com/web-hosting/compare_plans.pl
230-
230- ==============================================================
230 User 'css432w17' logged on.

215 UNIX Type: L8
ftp> ls
```

Figure 1: Connection Establishment



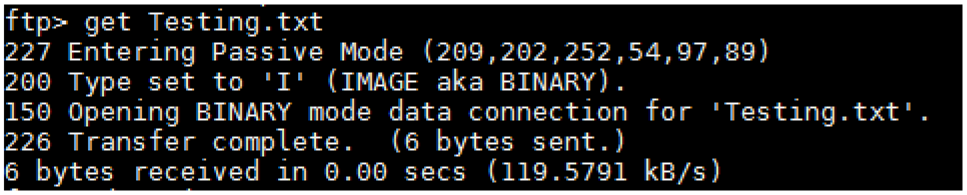
```
227 Entering Passive Mode (209,202,252,54,101,251)
150 Opening ASCII mode data connection for LIST.
-rw-r--r--   1 css432w17 Tripod        52 Mar 12 20:47 README.md
drwxr-xr-x   1 css432w17 Tripod         0 Feb 10 17:38 cgi-bin
drwxr-xr-x   1 css432w17 Tripod         0 Mar 13 01:11 project
-rw-r--r--   1 css432w17 Tripod         6 Mar 12 20:50 Testing.txt
-rw-r--r--   1 css432w17 Tripod     11960 Mar 13 00:28 testfile.txt
-rw-r--r--   1 css432w17 Tripod         5 Mar 12 22:24 data

226 Transfer complete.
```

Figure 2: Client's ls



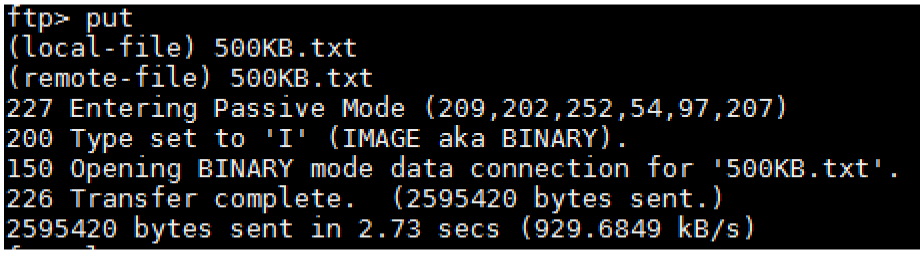
```
ftp> cd project
250 Directory set to '/project'.
```

Figure 3: client's cd function



```
ftp> get Testing.txt
227 Entering Passive Mode (209,202,252,54,97,89)
200 Type set to 'I' (IMAGE aka BINARY).
150 Opening BINARY mode data connection for 'Testing.txt'.
226 Transfer complete.  (6 bytes sent.)
6 bytes received in 0.00 secs (119.5791 kB/s)
```

Figure 4: Client's get



```
ftp> put
(local-file) 500KB.txt
(remote-file) 500KB.txt
227 Entering Passive Mode (209,202,252,54,97,207)
200 Type set to 'I' (IMAGE aka BINARY).
150 Opening BINARY mode data connection for '500KB.txt'.
226 Transfer complete.  (2595420 bytes sent.)
2595420 bytes sent in 2.73 secs (929.6849 kB/s)
```

Figure 5: Client's put

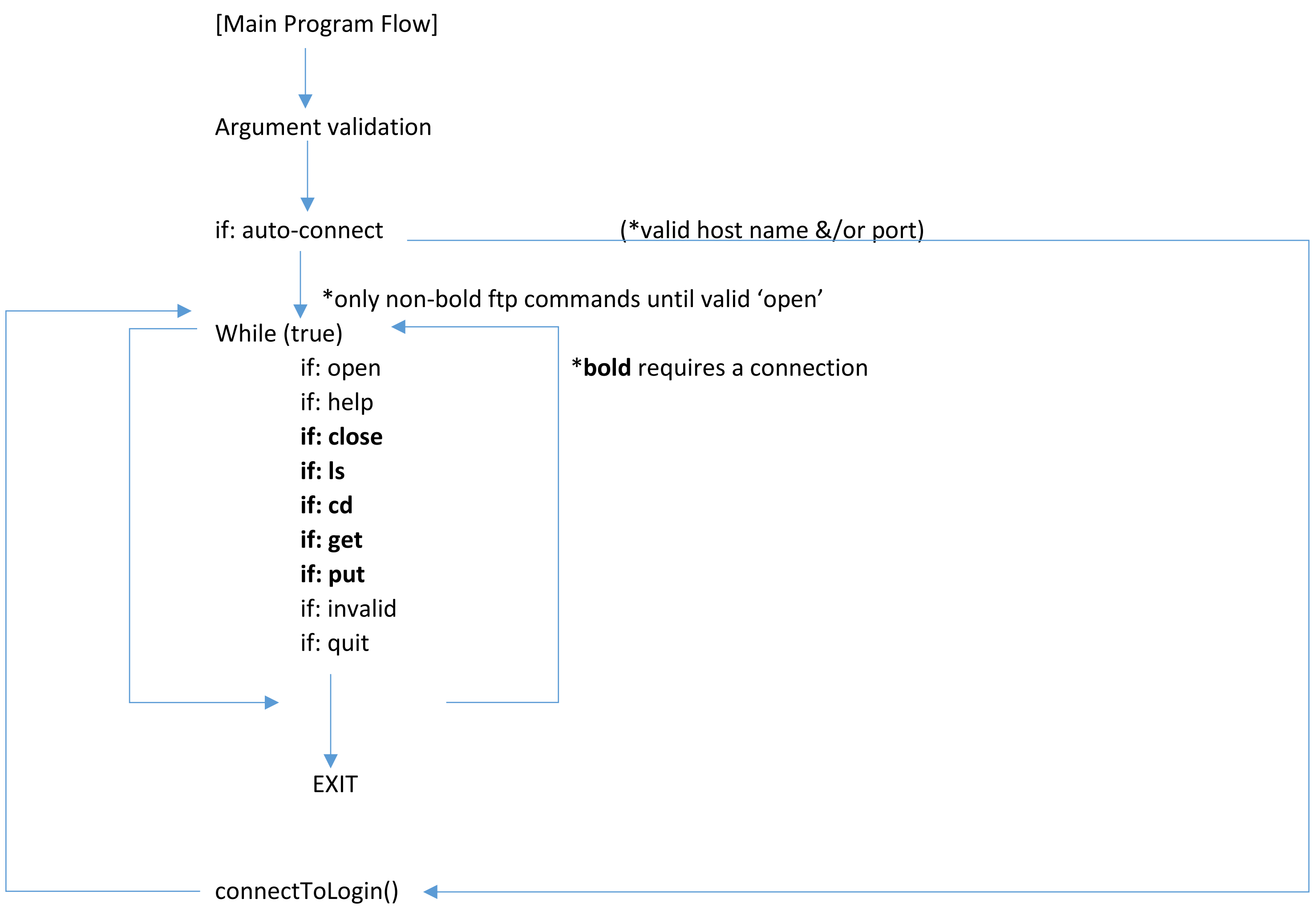
Performance Evaluation

Compare the difference in elapsed time between your ftp client program and the Unix ftp, when executing a GET command.

Execution:	500KB File - my ftp	500KB - Unix ftp	1000KB File - my ftp	1000KB - Unix ftp
1	1.21 sec (417kB/s)	1.14 sec (443kB/s)	1.95 sec (520kB/s)	1.87 sec (541kB/s)
2	1.22 sec (413kB/s)	1.21 sec (417kB/s)	1.95 sec (518kB/s)	1.77 sec (574kB/s)
3	1.29 sec (392kB/s)	1.22 sec (415kB/s)	1.94 sec (523kB/s)	1.84 sec (551kB/s)
Average	**1.24 sec (407kB/s)**	**1.19 sec (425kB/s)**	**1.95 sec (520kB/s)**	**1.83 sec (555kB/s)**

Discussions

1. **Discussions should be given for possible functional extensions of your own program and/or the Unix ftp. You should also mention about the difference in performance between your own program and the Unix ftp.**



My program was designed in a very linear way. Future extensions of it are possible by adding new ftp commands via new if statements however a restructuring of the code into classes and a more modular/OOP fashion would create a much more ideal structure of code that would be much more easily extendable.

The performance table compares multiple get execution times for both my ftp and the standard Unix ftp commands with different sized files. While Unix ultimately beat my implementation, it wasn't by much… indicating a correct implementation on my end. I believe with more data the margin might reduce even further.