Robert Griswold

Professor Robert Dimpsey

CSS 432 A

06 Jun 2016

Program 5 Report: FTP Client

File Transfer Protocol Program Description

The FTPClient.cpp program begins by checking whether or not the user gave a parameter when launching the program to immediately load it as an address with default port 21 for ftp. If the user did not specify this parameter, they will be greeted with "ftp>" where they can enter in commands. In the clocked socket state, only open, quit, and help will function. Upon opening a connection (the command takes 1 or 2 operands), the user will be prompted by the remote ftp server to enter a username and password before displaying the server's banner message. When in this connected state, the user can perform all commands except open to include cd, ls, get, put, close, quit, and help.

The ls command (LIST over FTP) will request the ftp server to list all the files and directories in the currently selected directory at once. The cd command (CWD over FTP) will change the current directory for the remote ftp client in order to select different files or locations for get and put. The get command (RETR over FTP) takes one parameter for filename and requests the server to transmit a file to the folder where the local client is being executed. The put command (STOR over FTP) does the same such that files that are in the folder where the local client is being executed can be sent to the remote server. The close command simply closes the open socket. The quit command will close the socket and quit the program. The help command simply lists the current state of the program, and the available commands.

When creating the file socket descriptor, the program can create either a read only, write only, or hybrid instance. In read only state, the file is opened with the O_RDONLY mode which simply prevents writing to the file. In the write only state, the options O_WRONLY | O_CREAT | O_TRUNC are used to prevent reading, create a file when it does not exist, and to override the file if it exists rather than appending data to an existing file.

Whenever the user executes either a ls, get, or put operation, the program sends signals to the remote FTP server to enter binary/image mode (TYPE I over FTP) and passive mode (PASV over FTP). Then with a response from the server confirming the quest, the client opens a separate data connection for that one transaction. Table 1 gives a brief example of the execution of the program.

Table 1: Execution output of FTPClient.cpp using a Linux machine to connect to FTP server: ftp.tripod.com.

ftp> open ftp.tripod.com bob Name (ftp.tripod.com:robbob4): css432

```
220 Welcome to Tripod us FTP.
331 Username set to css432. Now enter your password.
Password: UWB0TH3LL
230- ==========================================================
[…]
230 User 'css432' logged on.
Using binary mode to transfer files.
ftp> ls
227 Entering Passive Mode (209,202,252,54,248,218)
 […]
226 Transfer complete.
ftp> cd null
250 Directory set to '/null'.
ftp> quit
221 Goodbye...
```

Discussion

Design Changes

When originally thinking of how to implement this FTP client, I brainstormed the implementation using FTPClient, TCPSocket, and DataConnection classes seen in Figure 1. However, after beginning development on the program, I found some other sections of code easier to separate, while others were more difficult due to the visibility of the data.

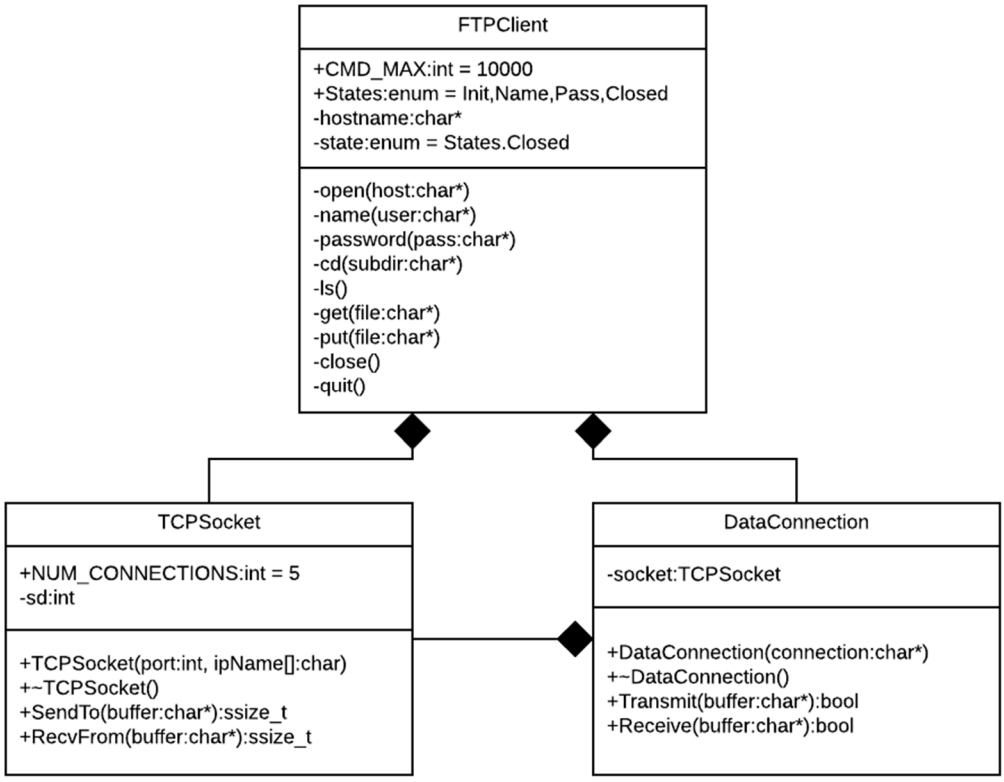

Figure 1: Initial class diagram for the FTP client class using FTPClient, TCPSocket, and DataConnection classes.

Rather than using a DataConnection class, I found both the InputParsing and FileIO classes to be useful, especially with how much similarity there is between the TCPSocket and FileIO classes. This led me to redesign my class diagram with the intention of separating as much code as I could reasonably do from the FTPClient class seen in Figure 2.

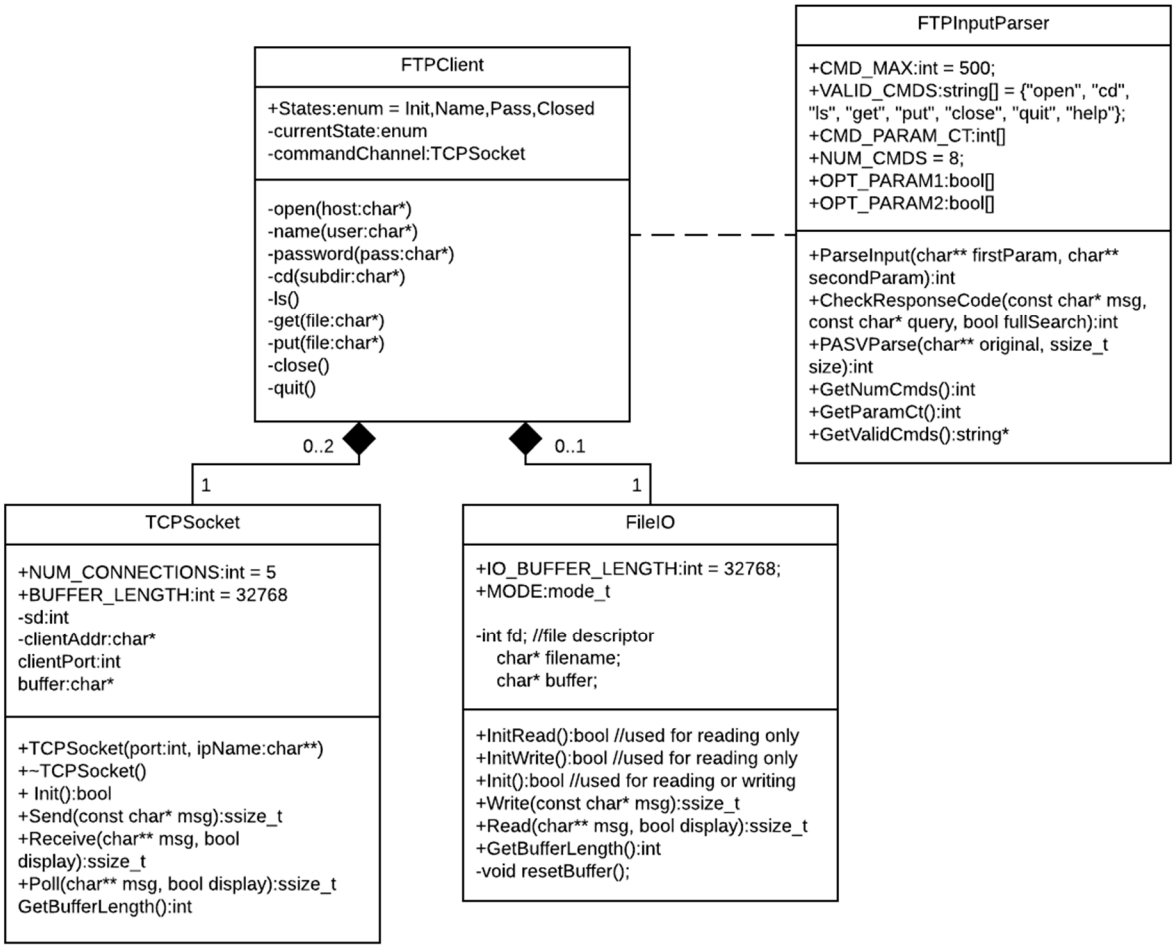

Figure 2: Final class diagram for the FTP client class using FTPClient, TCPSocket, FTPInputParser, and FileIO classes.