FTP Server

CIS457 Project 1

Andrew Weston
Shane Stacy
Freeman Ogburn

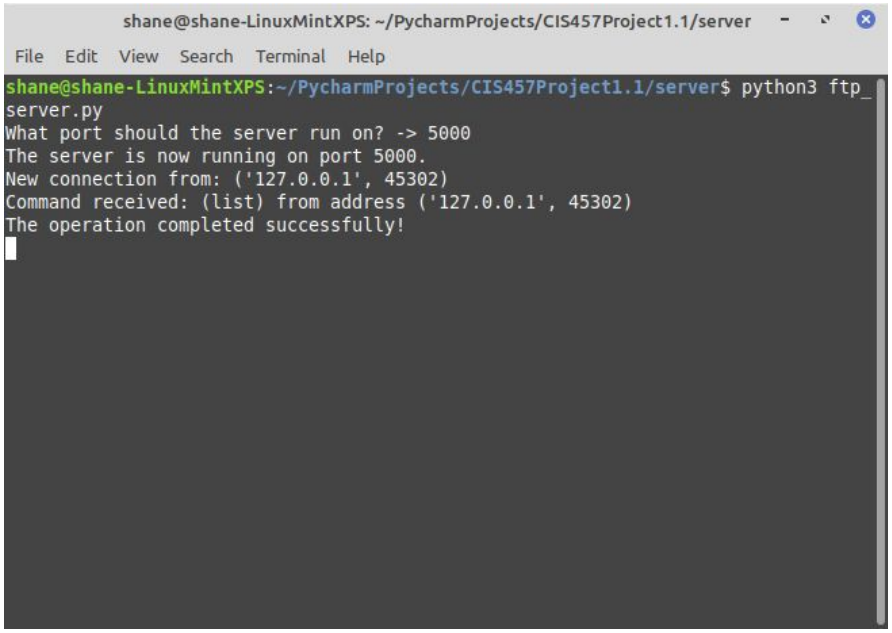
This report includes:

- Basic logic of the program
- What problems we encountered and how we solved them
- Screen capture demonstrating how each command works

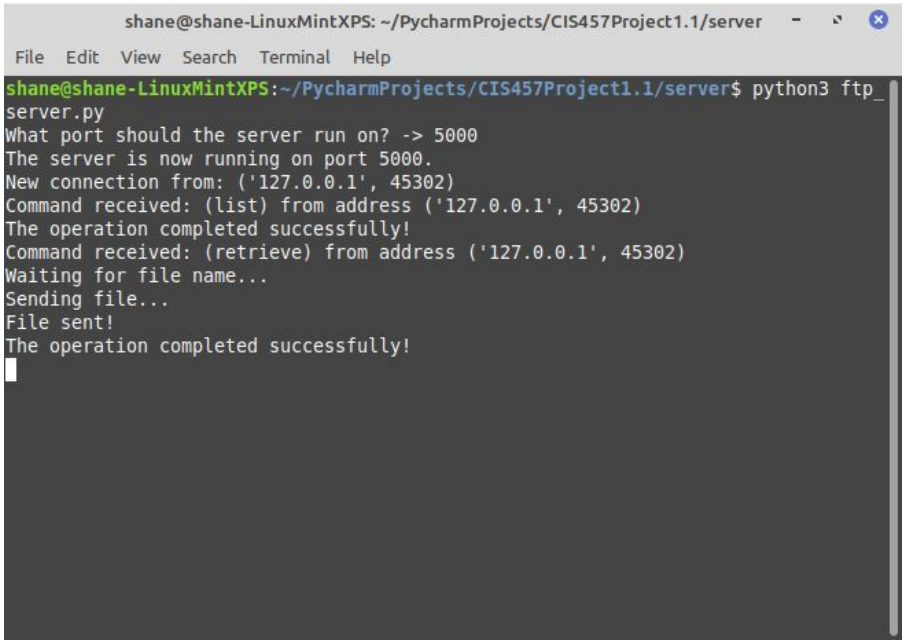
https://github.com/TRSlimShaney/CIS457/tree/master/FTPserver

Ftp_server.py is a Python 3.6 application designed to run a multithreaded FTP server. First, it asks the user what port they want to run the server on. Then it starts the process running on that port, listening for any clients. Once a client connects, it goes to a while loop where it waits for a command and then proceeds to a function based on the command received. Some additional feedback is supplied for debugging/logging purposes.

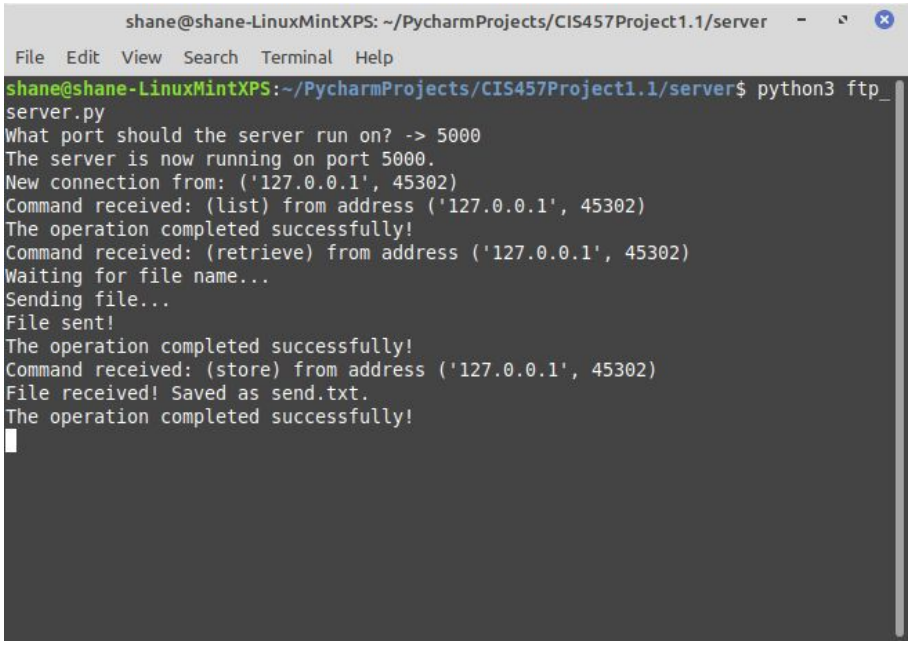
The list function has the server consult the OS directory for the files available, which is returned as a list. This list is sent back to the client as a formatted string.



The retrieve function waits for the filename from the client and then returns the file with that name to the client.

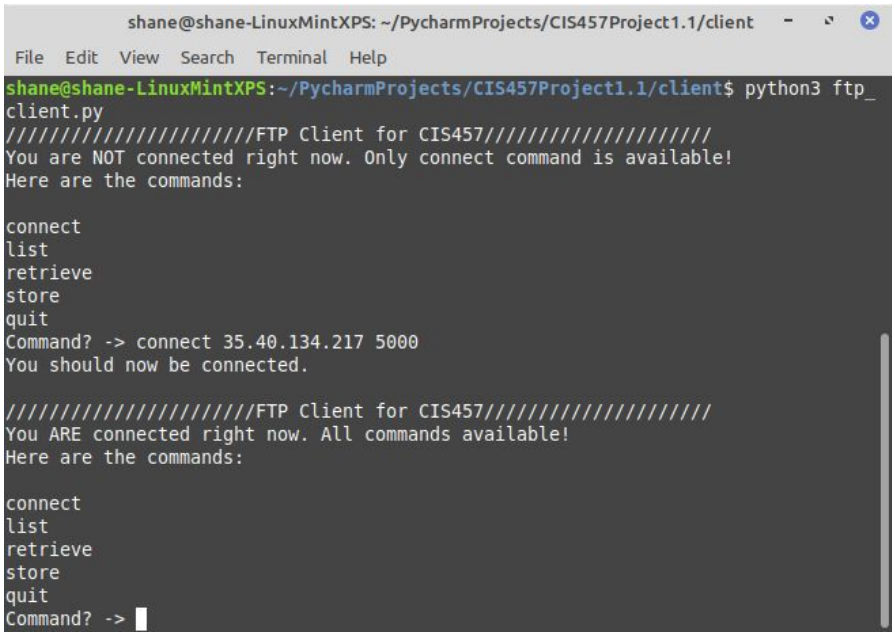


The store function waits for the filename from the client and then waits for the client to send the file which is then saved to the current directory on disk.

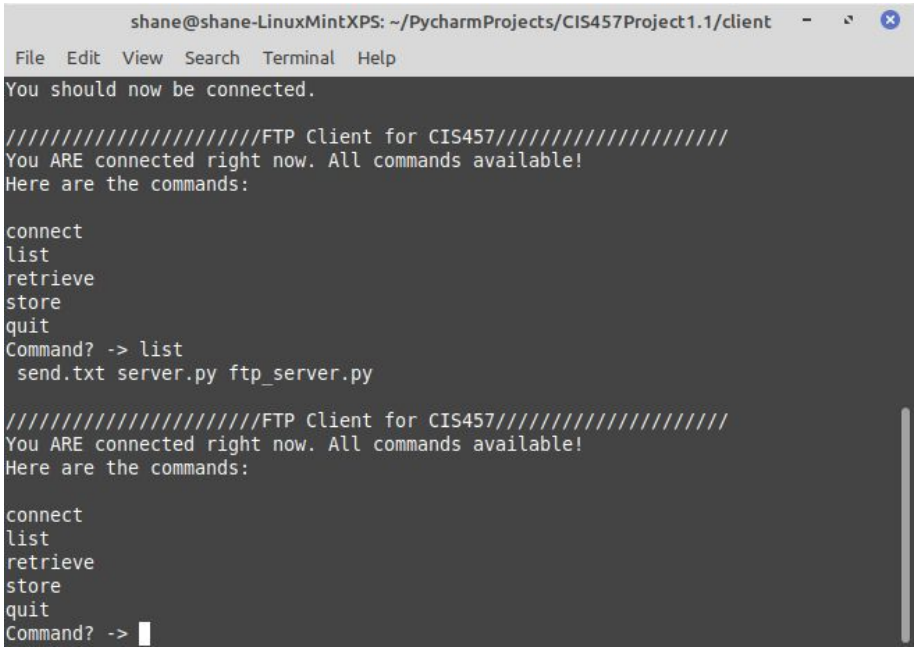


Ftp_client is a Python 3.6 script designed to run an FTP client. The first thing it does is list all the commands that can be entered. At the beginning, only connect is accepted since none of the other commands can be performed without connecting first.

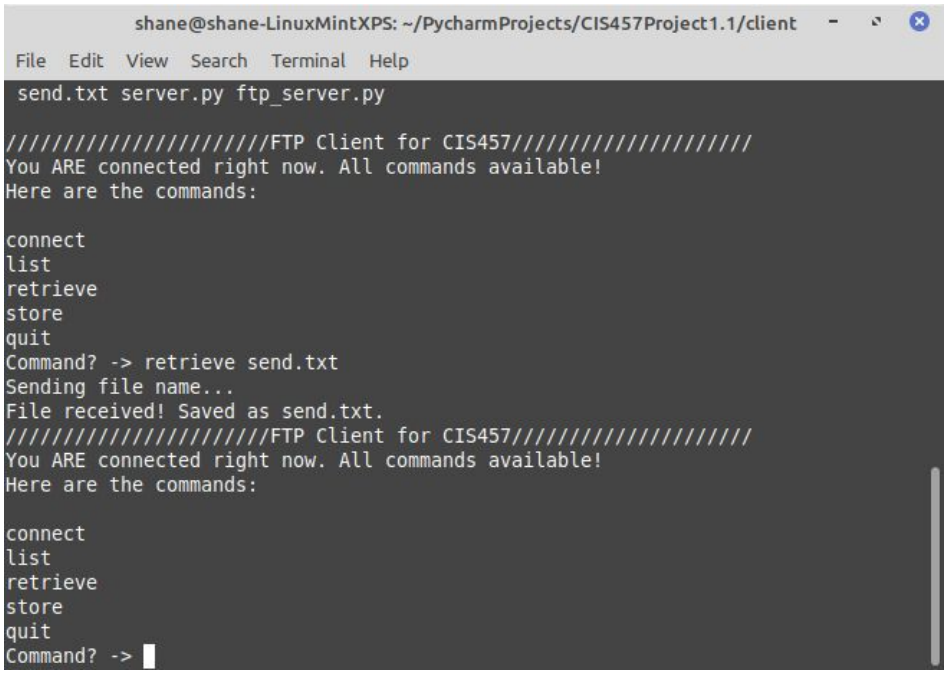
The **connect <ip> <port>** command opens a socket to the server with the entered ip and port. This allows the rest of the commands to become available, however connect itself is no longer available until the client is quit and restarted. The commands are relisted.



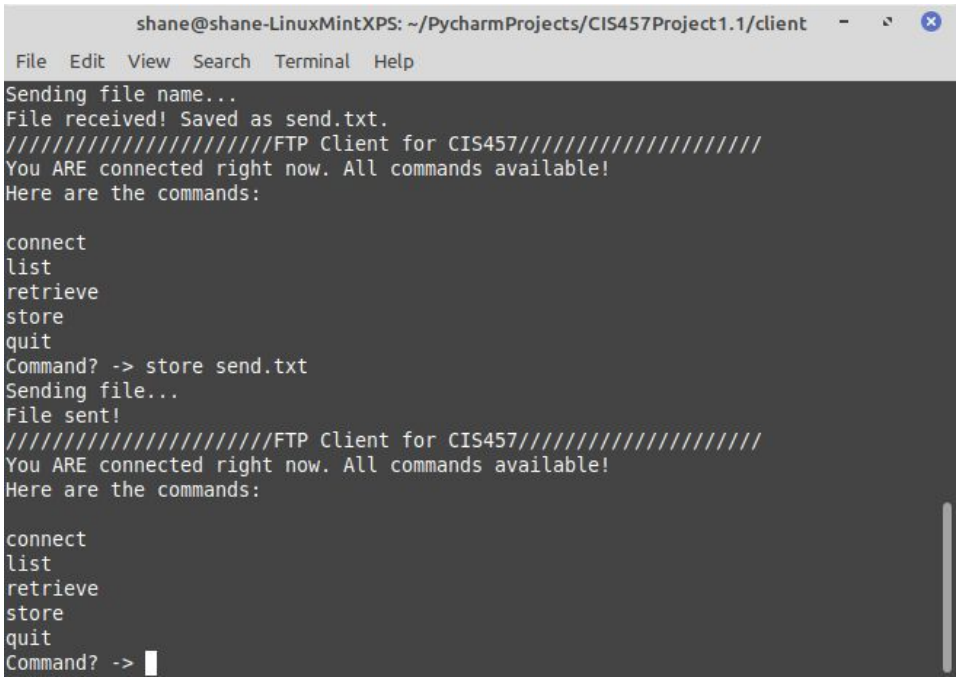
The **list** command sends the command to the server, which sends a string of the directory reading back to be displayed in the terminal.



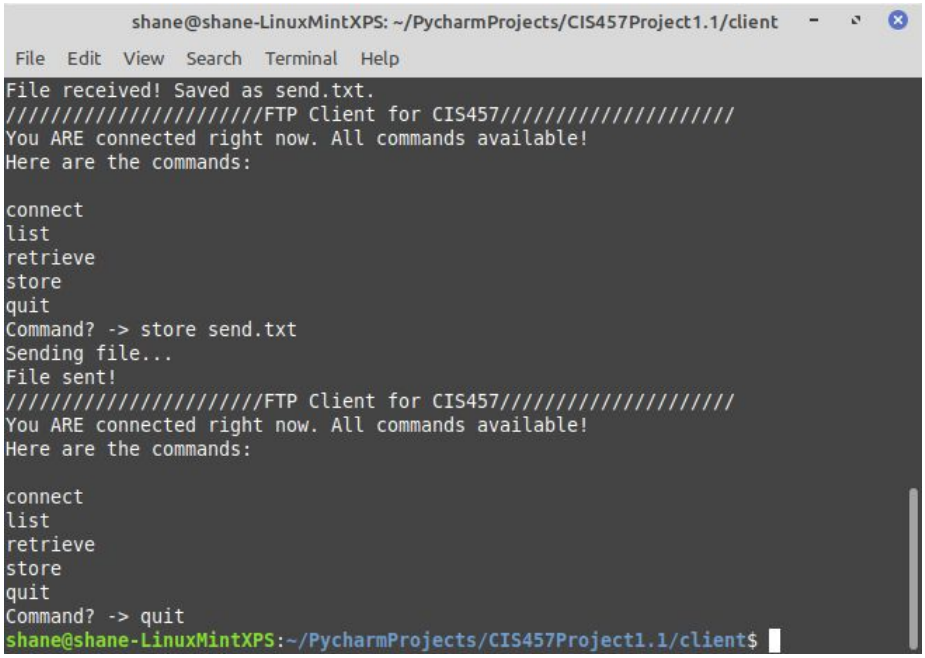
The **retrieve <filename>** command sends the filename to the server, and the server sends the file stream back which is then saved to disk.



The **store <filename>** command sends the filename to the server and the client then sends the file stream to the server.



The **quit** command terminates the client and it's connection to the server.



Problems:

We had some problems where data would arrive out of order or all at once and the server would get confused because a command and filename would arrive at the same time for the retrieve and store commands. We solved this by having the communication being call-response for everything so they client and server stayed in sync.

Another problem we had was with resolving hostnames. The Python API requires a hostname rather than an IP to create a socket so we had to write additional code to resolve a hostname based on an IP.