Ramzey Ghanaim

CMPE 156 Lab 5

March 20, 2018

Introduction

The purpose of this lab was to create a basic ftp client and server to list files, directories, as well as download and upload files.

Design and Error Messages.

When the ftp server first starts out, it waits for clients. When a client first connects, the PORT command is automatically sent out that connects the server and client with a provided port and IP address from the client. The format of this command is: "PORT n1,n2,n3,n4,n5,n6" where the IP address is n1.n2.n3.n4 and the port is n5*256+n6. A "200 Command OK." Message is sent back to the client from the server to confirm the connection.

After ports are established, the user is prompted with the prompt "ftp>" and th user can input four types of commands. The "ls" command allows users to list files and folders in the current directory or filenames. When the user first enters ls, I first verify the command is accurate, I then parse it by a space character to see if there are any arguments. I then attach found arguments to the string "LIST" before sending to the ftp server. The client does not understand UNIX/LINUX commands like ls but instead its own 4-character long commands. Ls happens to translate to the LIST command for ftp. When the command arrives at the server, a thread is fired. This thread executes the command while allowing the main process still able to receive commands (like an abort command). This will be discussed later. When the result is computed, and error message or 200 OK is sent back on the command channel while the actual string data is sent on the data channel. As the data arrives, it is displayed to the user in the client terminal. Once complete another prompt is displayed.

The second command a user can do is download a file with the "get" command. This command is translated to an FTP command "RETR". I parse the get command by space to extract the file name, ensuring the syntax is proper, followed by sending the command to the server. The server replies with a 200 OK message signaling the message is good for execution. The server then fires a thread and executes the command. It opens the file and begins sending the file. By this time a file has been made client side

so as data come through from the server on the data port the data is written to the new file.

The "put" command allows users to upload a file to the ftp server. This is the same process as the get command except the data transfer is reversed.

Next, I implemented the "ABOR" command. When the client hits CTRL+C, a signal handler is used to catch this event and send the "ABOR" message to the server. Upon receiving this message, the running thread that is transferring data has the data connection closed, following by a killing of that thread.

The last command is the "quit" command. This command is entered by the user when they want to exit the client program. The message I translated and sends a QUIT command to the ftp server. The server then closes the command connection with the client and waits to accept new clients. Meanwhile, the client process is ended.

Message Protocol

The supported messages are 200 Ok for good/successful requests. Messages like 500 unrecognized command is sent if a command does not exist. The 501 invalid arguments message is sent when a file is not found or the directory or file argument in the ls command is not valid.

Usage

These instructions assume the user is in the main directory of this project

1. To build the program simply type:

   "make"

2. To run the server, in one terminal in the serverBin/ folder type:

   "./ftpserver <portNum>"

Note: The <portNum> must match ports used when running the ftp client.

3. To runt the client, in a different terminal in the clientBin/ folder, type:

   "./ftpclient <ip> <portNum>"

Note: The <ip> must be a local host ip when running locally.

4. When one wants to clean the projects simply type:

   "make clean"