FTP, or File Transfer Protocol, is a method of transmitting files from one computer to another over a

TCP/IP connection. This FTP Client is written from scratch and is designed to mimic the functionality and

interaction of the native Linux console FTP client. It operates by establishing a TCP connection between

the client and a remote server over port 21 (or another user specified port). Commands and responses

are handled on the full-duplex control socket connection as character streams and decoded to

determine requested interactions. For ASCII or binary data streams, a second TCP socket connection

must be established.

Using commands such as LS, GET, or PUT require a second data channel to be established with the FTP server using “passive mode.” For each issuance of commands requiring a data channel, the client requests passive mode which invokes a response from the server, returning a secondary hostname and port number for the client to connect to. The client creates a second full-duplex connection to this address and receives data from the server. When the transfer is complete, the data channel is terminated. In active mode, the client begins listening for incoming data connections from the server on a local port to transfer data. This FTP client supports passive mode only and does not support

active mode.

Design

* OPEN, PASS/SYST, and QUIT

A close up of a map

Description automatically generated

Figure 1. Login with OPEN, USER, and PASS

The command “open” establishes a connection with the server, and only tries to log in the user if the FTP client is started with a server or port arguments such as the following: "usage: ./ftp <hostname>" && "usage: ./ftp <hostname> <port>"

When command open is invoked, the FTP client connects to the server using the IP name through a socket. If the FTP client was not started with any argument, the user must manually login prior to requesting services from the server. Figure 1 illustrates the algorithm used for a manual login.

A close up of a map

Description automatically generated

Figure 2. Closing the connection with server

The FTP client program has commands “close” and “quit” (or “exit) for disconnecting the connection with the server and exiting the program respectively. If the “quit” or “exit” command is invoked then the FTP client program disconnects only if the program is still connected to the server, and then gracefully terminates the program using the UNIX exit() command. The command close only disconnects the program from the server. Figure 2 illustrates the algorithm used for the command close.

* PASV, RETR

A close up of a map

Description automatically generated

Figure 3. ftp client retrieves a file from server

The FTP client program modularizes a function for enabling passive mode. The client sends a “PASV” command through the control channel, and then process the reply from the server to

Faraon 3

tokenize the numbers needed to calculate the port for the server’s data channel such as the following where MULTIPLER is equivalent to 256: int dataPort = (num1 \* MULTIPLIER) + num2;

The “dataPort” value is used to construct a socket and to establish connection with the server’s data channel. After enabling passive mode, the client can issue the commands “TYPE I” and “RETR <filename>” to the server. If the file that we are trying to retrieve exists within the server, then we fork a child to download the file from the server. Figure 3 illustrates how the command “get” works.

* CWD

A close up of a map

Description automatically generated

Figure 4. FTP client changes directory

The command “cd” or changing the working directory is pretty straightforward. The FTP client program sends the command “CWD <dirname>” to the server, and retrieves the reply from the server. Figure 4 illustrates how command “cd” works.

* PASV, LIST

A close up of a map

Description automatically generated

Figure 5. ls’s child process reads from pipe

The command “ls” lists the contents of the current working directory. This is done by enabling passive mode, then reading the standard output from the data channel. The implementation requires for the parent to wait for the child to terminate prior to reading the last reply from the server. Figure 5 illustrates how command “ls” works.

* PASV, STOR

A close up of a map

Description automatically generated

Figure 6. put determines if local file is accessible for uploading

The command “put” uses passive mode and follows the reverse algorithm of the command “get.” A “FIN” flag is sent to the data channel after the last chunk of data is sent to the data channel. Additionally, the “put” command only opens the file in read only mode to prevent the losing or corrupting the file. Figure 6 illustrates how command “put” works.