Design

(1) client's interface in terms of OPEN, PASS/SYST, and QUIT:

The open interface only occurs if the user does not provide an argument to the ftp client program. If no argument is provided “open: ” will appear and allow the user to specify what ftp server to connect to. Since this is specifically an ftp client program, I opted out of asking the user to enter the port since it should always be 21. If you type in a port, it will NOT work. Once the user enters the name of the server to connect to, the login process begins. The user is prompted to enter a username and password. They will be prompted repeatedly until a correct username/password is established with the server. SYST is immediately called afterwards to return the server system data. The user can quit any time by typing “quit”. The program will disconnect from the ftp server and exit the ftp client program.

(2) client's get function in terms of PAV, RETR, and local file options(O_CREAT | O_WRONLY and S_IRUSR | S_IWUSR |S_IRGRP | S_IROTH)

When the user wants to get a file from the server, they may type the command “get (filename)” to retrieve the file from the server. If the file does not exists a warning is sent to the user. If a filename is specified and does exists, a passive command is sent to the server followed by a Type I command. A new part is passed back to the client program and is given to a child process in order to open a data channel to the ftp client. Once the data channel is open the control channel passes the ftp server the RETR command in order to tell the server to begin sending the files byte data over the data channel. The data is caught by the child, saved, and then the child kills itself. This is the first time iam head=ring about these file options. I did not use them, and it appears i did not need to use them. On second thought, I will add them now. I do not want to lose points.

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

The user has the option to send the command “cd (directory)” which will change the ftp’s working directory to the directory specified by the user, if the directory exists. If the directory does not exist, the ftp’s error is forwarded to the user to let them know. The cd command is implemented by sending the server the CWD command and then reading the server's response. The response is read and then send directly to the user's console for viewing. The user may now perform all other commands in this directory such as ls, get, put, etc.

(4) client's ls function in terms of PASV, LIST, and file output to stdout

The client has the option to use the ls command. This command will list all the contents of the current working directory so that the user can see what files and directories are in the current directory. This is implemented by sending the ftp server a LIST command. For this to work properly the ftp server must first receive a PASV command. This passive command will cause the server to return a new port number for the data channel that can be connected to in order to retrieve the list data containing all the files and directories in the current working directory. After the pasv command is sent, A Type I command is send to enable binary data transfer. Then a child process is forked in order to listen on the data channel for incoming file and directory data. While the child is listening for data, the parent makes the LIST call. This tells the ftp server that it may send file and directory data over the data channel to the child. The child reads this data and then kills itself. The data is printed to the console so that the user can read it.

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

The client may also make the command “put, (filename)”. This command sends a file with filename to be send to the ftp server. For this to work properly the ftp server must first receive a PASV command. This passive command will cause the server to return a new port number for the data channel that can be connected to in order to receive the file that the user has specified to send to the ftp server’s working directory. After the pasv command is sent, A Type I command is send to enable binary data transfer. Then a child process is forked in order to write to the data channel, the file that the user has specified. Before the child writes the data, the parent makes the STOR call. This tells the ftp server that the child is sending data now over the data channel and it should listen for incoming data that is the file. The child sends this data and then kills itself. A read is then made on the client side to inform the user of the transfer and the number of bytes transferred.