Project I: File Transfer Protocol (FTP)

Assigned: February 25, 2020
Groups formation and git repository setup: February 27, 2020 (10% of the grade)
Checkpoint I due: March 5, 2020 (30% of the grade)
Final version due: March 14, 2020 (60% of the grade)

Your task is to build an application protocol that runs on top of the sockets. In this project, your task is to create a simplified version of the FTP protocol. As you have seen in the lecture, FTP works in a client/server model.

Notice that the FTP server must work in concurrent mode, that is, the server must be able to handle simultaneous users' requests. This can be done using the Select() function. You can read more about Select() under [2].

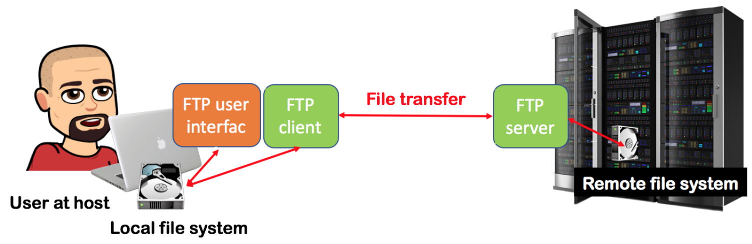
1 Your task:

Your first task revolves around creating two separate programs, FTPclient.c and FTPserver.c. FTP works in connection-oriented mode, which means a FTP client needs to establish a TCP connection to the server. According to the standard (RFC 959), this TCP connection is going to be a persistent connection which is used as the control channel. Once the client decides to start a data stream, which could be either uploading a file to the server or downloading a file from the server, a new TCP connection is established on a different port for the data transfer. Once the file transfer ends, the TCP connection of the data transfer must be closed.

You need to first start the server process in a server host. As you have seen in the lecture (see figure below), at the client side we have two components: FTP user interface, and FTP client. In this project we will create our own simple user interface with a command prompt asking the user to input a command (similar to telnet). To start this, the client needs to start his FTP program specifying the FTP server's IP address and the it's port number as:

```
./FTPclient ftp-server-ip-address ftp-server-port-number
```

After the establishment of the control TCP connection with the FTPserver, the client process should responds with prompt $ftp >$, waiting for user's ftp commands.



1.1 FTP commands

FTP RFC 959 details a large number of commands that the FTP protocol, in this project we are only going to implement a couple of these commands. Below is a list of all the required commands that you need to implement.

USER username with this command the client gets to identify which user is trying to login to the FTP server. This is mainly used for authentication, since there might be different access controls specified for each user.

PASS password once the client issues a USER command to the server and gets the "331 Username OK, password required" status code from the server. The client needs to authenticate with the user password by issuing a PASS command followed by the user password.

PUT filename this command is used to upload a file named filename from the current client directory to the current server directory. This command should trigger the transfer of the file all the way to the server. Remember that a separate TCP connection to the server needs to be opened for the data transfer.

GET filename this command is used to download a file named *filename* from the current server directory to the current client directory. Similar to PUT, a new separate TCP connection to the server needs to be opened for the data transfer.

LS this command is used to list all the files under the current server directory

!LS this command is used to list all the files under the current client directory

CD this command is used to change the current server directory

!CD this command is used to change the current client directory

PWD this command displays the current server directory

!PWD this command displays the current client directory

QUIT this command quits the FTP session and closes the control TCP connection

<u>**Side note:**</u>

Any command other than the ones above should not be accepted. It should be considered as an invalid FTP command. The server should respond with an "Invalid FTP command" message.

1.2 Considerations

Your FTP server should be able to respond with an appropriate error messages when some error occurs. For example, no command should be accepted by the server if the user hasn't authenticated itself with the server. Instead the server must respond with either a "User authentication is pending" message, or a "Password authentication is pending" message. The authentication process comprises of both sending a USER command and a PASS command.

It goes without saying, that if the user chooses the wrong *filename* and the file does not exist, or if the directory the user is trying to CD into does not exist, that a relevant error message should be issued from the server to the client.

Here are a couple of more things that you should consider:

1. The implementation requires to use a stream socket that has no boundaries between consecutive messages. Its your responsibility to recognize each message from the stream socket.

2. When you transfer a file using a socket, you need to indicate the end of the file transfer.

1.3 Testing

Code quality is of particular importance to server robustness in the presence of client errors and malicious attacks. Thus, a large part of this assignment (and programming in general) is knowing how to test and debug your work. There are many ways to do this; be creative. We would like to know how you tested your server and how you convinced yourself it actually works. To this end, you should submit your test code along with brief documentation describing what you did to test that your server works. The test cases should include both generic ones that check the server functionality and those that test particular corner cases.

If your server/client fails on some tests and you do not have time to fix it, this should also be documented (we would rather appreciate that you know and acknowledge the pitfalls of your implementation, than miss them). Several paragraphs (or even a bulleted list of things done and why) should suffice for the test case documentation.

To help you get started on testing, we have provided a pseudo-codes detailing the implementation flow.

1.4 Pseudo-codes

In an attempt to help you with your first project, we have provided pseudo-codes detailing the flow on how you can implement both the server and the client [3]. This is not a requirement, and you don't have to follow it if you don't want to.

FTPClient.c pseudo-code

```
sd=socket(AF_INET, SOCK_STREAM, 0)
get server host name from argv[1] and server port number from argv[2]
connect(sd, &server-socket-address,...)

While (1) {
show ftp>
fgets a ftp command line from keyboard

if the command is "USER username"
        0. send the command to the server
        1. fgets a reply line from the socket to see if the command
        is successfully executed and display it to the user

if the command is "PASS password"
        0. send the command to the server
        1. fgets a reply line from the socket to see if the command
        is successfully executed and display it to the user

if the command is "PUT a existed file"
        0. send the command to the server
        1. open the file
        2. read the file
        3. open a new TCP connection to server
        3. write the file to server
        4. close the file
        5. close the TCP connection

if the command is "GET a file"
        1. send the command to the server
        2. fgets first line from the server:existed or non-existed
        3. if existed
                3.1 open a new TCP connection to server
                3.1 read the file from the server
                3.2 write the file to the local directory
                3.3 close the new TCP connection
        4. if non-existed display "filename: no such file on server"
if the command is "CD ..." , "LS ...", or "PWD"
        1. send the command to the server
        2. fgets a reply line from the socket to see if the command
        is successfully executed
        3. read from the socket and display correspondingly.

if the command is "!LS ..." or "!PWD"
        1. call system(command) locally

if the command is "!CD directory"
        1. call chdir (directory) locally. Note that system( )
        cannot execute "cd ..."

if the command is "QUIT"
        1. close the socket
        2. break or exit

otherwise: show "An invalid ftp command."
}
```

FTPServer.c pseudo-code

```
While (1) {
fgets a ftp command line from client via stream socket

if the command is "USER username"
        1. checks if the username against authorized users
        2. if username exists
                send "Username OK, password required"
                message to client
        3. if user name does not exist
                send "Username does not exist" to the client

if the command is "PASS password"
        1. if the username is set
                1.1 check if the password
                1.2 if password matches
                        send "Authentication complete"
                1.3 if password does not match
                        send "wrong password"
        2. if the username is not set
                send "set USER first"
if user has authenticated
        if the command is "PUT a file"
                1 create a file
                2 read the file from socket
                3 write to the file
                4 close the file

        if the command is "GET a file"
                1. open the file
                2. if existed
                        2.1 send "existed" to client
                        2.2 read the file
                        2.3 write the file to client
                        2.4 close the file
                3. if nonexitested
                        send "nonexisted" to client

        if the command is "ls ..." or "pwd"
                1. fp =popen(command,"r")
                2. read the result from fp
                3. if no result from fp, reply "wrong command
                usage!" to client, otherwise reply "successfully
                executed!" to client
                4. send the result to client

        if the command is "cd directory..."
                1. call chdirr(directory).
                2.reply to the client if the command is
                successfully executed.

if the command is "QUIT"
        1. close socket
        2. exit

if user has not authenticated yet
        Regardless of what you receive
                send "Authenticate first" message

}
```

1.5 Hand-in

Handing in code for checkpoints and the final submission deadline will be done through your Github repositories. You can setup a private repository on Github and share it with yzaki, moumena19.

When you commit your repository, make sure that you name it FTP-name1-name2, for example ”FTP-Yasir-Steve”. Make sure that you do regular commits with meaningful comments so that we can track your progress.

Your repository should contain the following files:

1. Makefile – Make sure all the variables and paths are set correctly such that your program compiles in the hand-in directory. The Makefile should build executable named FTPclient and FTPserver.
2. All of your source code – (files ending with .c, .h, etc. only, no .o files and no executables)
3. readme.txt – File containing a brief description of your design of your FTP server and client.
4. tests.txt – File containing documentation of your test cases and any known issues you have.

Groups formation and git repository setup
This project is a group project and you must find exactly one partner to work with. Once you have settled on a partner, you need to inform us by setting a git repository and sharing it with us no later than the 26th of February. In you repository name you will have the names of the partners such as ”FTP-Yasir-Steve”.

Check Point I
For check point I, you should commit a server code that is capable of handling multiple clients using Select(). It also should include the basic authentication commands (USER and PASS). The commit should clearly be identified as ”Check Point I”

Final Submission
The final submission should include the full FTP client and serer functionalities.

Late submissions
Late submissions will be handled according to the policy given in the course syllabus.

References

[1] Stevens, W.R. and Fenner, B. and Rudoff, A.M. *UNIX Network Programming (assigned textbook)*. Addison-Wesley, Reading, Massachusetts, 2014.

[2] The World of Select(),
`http://www.lowtek.com/sockets/select.html`

[3] G Chen, ECE5650: Network Programming course,
`http://www.ece.eng.wayne.edu/ gchen/ece5650/myftp.pdf`