# File Transfer Protocol (FTP) Server Software

## Objectives

1. To learn how a computer network protocol is outlined in a Request for Comments (RFC) document by Internet Engineering Taskforce (IETF) as an Internet Standard.
2. To understand File Transfer Protocol (FTP) from RFC 959.
3. To learn and use Unix Network or Socket programming API.
4. To implement FTP Server software according to Internet Standard outlined in RFC 959.

## Specifications

You have been using **File Transfer Protocol** (FTP) to get or to store files from or to FTP servers. You use a FTP Client software in your host computer to communicate with an FTP server. FTP server runs a FTP Server software. In the first project, you have implemented your own FTP Client software and tested your FTP Client software with a given FTP Server software. In this project, you are going to implement your own FTP Server software in C programming language and test your FTP Server software with a given FTP Client Software or with your FTP Client Software that you have built in project 1.

You have already learned that FTP is a client-server protocol to transfer files to/from the servers. An FTP client sends request message to an FTP server regarding a file transfer. FTP server interprets the request message, takes appropriate action, and sends back response message to the client. **RFC 959** describes FTP and its request and response messages in detail. You will need to read and understand **RFC 959** to complete this project. Although, **RFC 959** describes many request messages, you need to implement only the followings:

- **USER**
- **PASS**
- **PWD**
- **CWD**
- **CDUP**
- **PASV**
- **NLST**
- **RETR**
- **QUIT**

Your FTP Server software must be compatible with the FTP Client software of your first project, which has a command-line user interface (UI) and the users can enter following commands through this interface.

- **help**
- **user <username>**
- **pass <password>**
- **pwd**
- **dir**
- **cwd <dirname>**
- **cdup**
- **get  <filename>**
- **quit**

Command '**help**' displays the list of commands supported by this software, their syntaxes, and meanings.

Command '**user**' sends username to the FTP server for authentication. You need to support only one user with user name 'csci460' and password '460pass'.

Command '**pass**' sends the password to the FTP server for authentication.

Command '**pwd**' prints the current working directory of the FTP server.

Command '**dir**' lists the contents of the current working directory of the FTP server.

Command '**cwd**' changes the current working directory to a specified directory. If the specified directory is beyond current working directory an error is reported by the server.

Command '**cdup**' changes the current working directory to the parent directory. If the parent directory is beyond server's base directory, an error is reported by the server.

Command '**get**' fetches the specified file from the FTP server. If the specified file is not available an error is reported by the server.

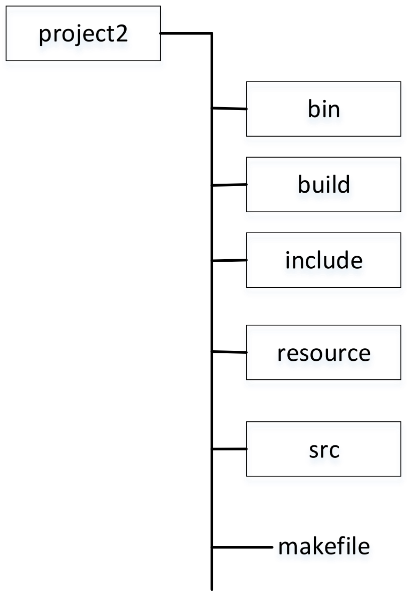
Command '**quit**' terminates the software.

FTP Client software interprets above user commands, translates them into appropriate FTP request messages, sends the FTP request messages to FTP server, receives the response messages from the server, and presents the response to the user in a user-friendly manner.

FTP Client and Server communicate request and response over a control connection. Some request or response involve a data transmission. For example, RETR request involves a data transmission to transfer file data as the part of a successful response. FTP server uses a separate connection for each data transmission.  A data connection is opened on demand and closed when a data transmission is complete. FTP server can operate either in active or in passive mode. In active mode, FTP server opens the data connection with the client on demand. If the client is behind a firewall, FTP active mode fails to open the data connection. In passive mode, when a data connection is required the server opens a connection listener so that client can send connection request to the listener and open a data connection. Client instructs the server to enter into passive mode by sending a PASV request message to the server. The server opens the connection listener on a port and sends the port number to the client in its PASV response. After receiving the PASV response, the client retrieves listener port number and sends a connection request to the listener port in order to open a data connection. In this project, you must implement passive mode. You assume that FTP client software always sends a PASV request before any data request, such as RETR and NLST.

## Tasks

1. You will submit this project using **GIT submission system**. A central repository named **project2** has been created for this project.
2. Repository **project2** has been organized as follows:



3. A **makefile** has already been created in the repository.
4. The code for **ftp_server.cpp** has been supplied into the **src** folder in the repository.
5. Following **hpp** files have been supplied into the **include** folder in the repository.
    a. **ftp_server_command.hpp**
    b. **ftp_server_connection_listener.hpp**
    c. **ftp_server_connection.hpp**
    d. **ftp_server_net_util.hpp**
    e. **ftp_server_nlist.hpp**
    f. **ftp_server_passive.hpp**
    g. **ftp_server_retrieve.hpp**
    h. **ftp_server_session.hpp**

    i. **ftp_server_string_uitl.hpp**
    j. **ftp_server_response.hpp**

6. If you are doing the project alone, create your own fork of **project2** on the central GIT repository using following command. You can skip this step if you are doing the project in a team or group.

   *ssh csci fork csci460/project2 csci460/$USER/project2*

7. A forked repository for your team has already been created if you are working in a team. You don't need to fork yourself. For example, the central repository for **team1** is **csci460/team1/project2**. In order to find out which team repository you have access, type following command:

   *ssh csci info*

8. You have already created a folder named **csci460** in your home folder in your first project, go into your **csci460** folder.

9. Create a clone of your forked **project2** repository using following command if you are working alone:

   *git clone csci:csci460/$USER/project2*

10. Create a clone of your forked team **project2** repository using following command if you are working in a team (assuming you are working in team1, replace team1 with your own team):

    *git clone csci:csci460/team1/project2*

11. Continue your work in your cloned or local **project2** repository and **commit** and **push** your work to your central **project2** repository as it progresses**.**

12. **Implement** following **cpp** files into your **src** folder.
    a. **ftp_server_command.cpp**
    b. **ftp_server_connection_listener.cpp**
    c. **ftp_server_connection.cpp**
    d. **ftp_server_net_util.cpp**
    e. **ftp_server_nlist.cpp**
    f. **ftp_server_passive.cpp**
    g. **ftp_server_retrieve.cpp**
    h. **ftp_server_session.cpp**
    i. **ftp_server_string_uitl.cpp**

13. The lists of internal dependencies of the **cpp** files in this project are as follows.
    a. **ftp_server_command.cpp**

    i. ftp_server_command.hpp
    ii. ftp_server_connection.hpp
    iii. ftp_server_passive.hpp
    iv. ftp_server_nlist.hpp
    v. ftp_server_retrieve.hpp
    vi. ftp_server_response.hpp

b. **ftp_server_connection_listener.cpp**
    i. ftp_server_connection_listener.hpp
    ii. ftp_server_net_util.hpp
c. **ftp_server_connection.cpp**
    i. ftp_server_connection.hpp
    ii. ftp_server_net_util.hpp
d. **ftp_server_net_util.cpp**
    i. ftp_server_net_util.hpp
e. **ftp_server_nlist.cpp**
    i. ftp_server_nlist.hpp
f. **ftp_server_passive.cpp**
    i. ftp_server_passive.hpp
    ii. ftp_server_connection_listener.hpp
    iii. ftp_server_connection.hpp
    iv. ftp_server_net_util.hpp
    v. ftp_server_command.hpp
    vi. ftp_server_string_util.hpp
    vii. ftp_server_response.hpp
g. **ftp_server_retrieve.cpp**
    i. ftp_server_retrieve.hpp
    ii. ftp_server_connection.hpp
    iii. ftp_server_response.hpp
h. **ftp_server_session.cpp**
    i. ftp_server_session.hpp
    ii. ftp_server_connection.hpp
    iii. ftp_server_command.hpp
    iv. ftp_server_response.hpp
i. **ftp_server_string_uitl.cpp**
    i. ftp_server_string_uitl.hpp
j. **ftp_server.cpp**
    i. ftp_server_connection_listener.hpp
    ii. ftp_server_session.hpp

14. **Organize your code nicely** and **add adequate comments** on your code.
15. **Build** your FTP Server software using **make**.

16. **Run** FTP Server software typing

    ***>bin/myftpserver  2019***

    This will run the FTP Server software in your computer and the server is now waiting for the client connection at port 2019.

    You can download this example **FTPServer Software** in your **bin** folder and run it to get an idea what is expected from you.

17. **Copy** this test FTP Client software **or** your **project 1** FTPClient Software in your **bin** folder and **Run** the FTP Client software typing

    ***>bin/myftp  127.0.0.1  2019***

    Above command will connect your client software to an FTP Server software running at your machine and listening for client connection at port ***2019.***

    Note that the example FTPServer Software and the test FTP Client Software were built and tested only in Linux Debian machines of the lab. Run them in other machines at your own risks.

18. **Test** all the commands in your FTP Client software against the connected FTP Server.
19. Submit a **printed copy** of all your cpp files to your instructor. You need to submit your **printed copy** of the **cpp** files before your final demonstration.
20. Submit the filled up **contribution form** if you have done your project in a partnership with another student.

## Partnership

You are allowed to complete this project with another partner. In that case, your contributions must be marked in your source code. If you are the only contributor of a function, you should mark yourself as the sole author of that function before the function header. If you are the only contributor of all the functions in a file, you should mark yourself as the sole author of the file at the beginning of the file. If both you and your partner have contributed to write a function or all the functions of a file, both of you should mark yourselves as the authors at appropriate position. You and your partner must complete, sign, and submit this **contribution form** to your instructor along with the **printed copy** of all your source codes (**cpp** files only).

## Deadline and Submission

The deadline to demonstrate and submit the project is **November 26, 2019, Tuesday by the lab hour.**

Submit a **printed copy** of all your cpp files to your instructor. You need to submit your **printed copy** of the **cpp** files before your final demonstration.

**Commit** and **push** your work from your **local** project2 **repository** to your **remote** project2 **repository** regularly. You will be allowed to commit and push until the deadline is over. Incremental and frequent commits and pushes are highly expected and recommended in this project.

## Evaluation

| Demonstration | Commands | Marks |
|---|---|---|
| | USER | 5 |
| | PASS | 5 |
| | PWD | 5 |
| | CWD | 5 |
| | CDUP | 10 |
| | PASV | 20 |
| | NLST | 10 |
| | RETR | 10 |
| | QUIT | 5 |
| Code Quality | | 25 |
| Total | | 100 |