

Project 2: File Transport Protocol

Computer Networks (CS-UH 3012) - Spring 2025

1 Code of Conduct

In the following, we outline the rules and expectations that govern the submission of academic assignments. All students are expected to adhere to the rules of academic honesty and ethical conduct in all academic work and take responsibility for their academic work.

1. **Originality:** All assignments submitted must be the original work of the student. Any document and program code that is submitted must be **entirely** written by the student.
2. **Discussions:** While students are encouraged to discuss their questions with other students, these discussions should be limited to general solution techniques and concepts. Discussions about concrete code or specific results that the student intends to submit are not allowed and are considered an integrity violation.
3. **Solution Sharing:** Discussing an assignment with others should never lead to the possession or temporary access complete or partial solutions from others. Possessing and accessing solutions, regardless of whether in paper or digital form, and irrespective of the source of the solution, is not allowed and is considered an integrity violation.
4. **Unauthorized Assistance:** Obtaining solutions from sources outside of the course, including but not limited to students of current or previous years or sections, other universities, the Internet, AI-Chatbots, etc. is not allowed and is considered an integrity violation.
5. **Code Sharing:** Sharing submissions with fellow students is not allowed and will be considered an integrity violation. Students must maintain confidentiality and keep their code private.
6. **Publication:** Publishing submissions or assignment descriptions online in any form is not allowed.
7. **Academic Responsibility:** Each student is solely responsible for the work submitted for an assignment. Any doubts regarding the authenticity of the work may lead to an investigation and necessary action by the academic administration. If any suspected violations are detected, they will be reported to the academic administration as per the policies of NYU Abu Dhabi. Disciplinary actions may include but are not limited to failing the assignment, failing the course, or suspension from the university. More details can be found at:

<https://students.nyuad.nyu.edu/academics/registration/academic-policies/academic-integrity/>

Penalty for integrity violations:

Every student found to be involved in an integrity violation will be penalized. In case a first integrity violation is reported, the penalty is a 10% deduction from the final grade, or the full percentage of the graded assessment, whichever is higher. A second integrity violation will result in an automatic course failure.

2 Project Objectives

The task of this project is to build a simplified version of the FTP application protocol, consisting of two separate programs: an FTP client and an FTP server. The FTP server is responsible for maintaining FTP sessions and providing file access. The FTP client is split into two components: an FTP user interface and an FTP client to make requests to the FTP server. The client provides a simple user interface with a command prompt asking the user to input a command (see Section 4) that will be issued to the FTP server.

Note that the FTP server must be started first and must support concurrent connections. That is, the server must be able to handle multiple simultaneous requests from different clients. This can be done using the `select()` and `fork()` functions of C. `Fork()` should be used for handling resource-intensive requests/commands such as file transfer. For the control connection using `select()` is sufficient. **Please note that if `select()` is not used to handle simple commands of control connection and `fork()` is used instead, there will be a penalty as this is inefficient due to the expensive extra OS resources used by the `fork()`.**

This project is a group project and you must find exactly one partner to work with. If you decide to change your partner (from your current group previously formed for Project 1), then you need to inform us by sending an email to the TA. See the due date under submission details and policy. There will be a 10% penalty if the deadline of the group formation is not met.

Please read the complete project description carefully before you start so that you know exactly what is being provided and what functionality you are expected to add. Please also look into the RFC 959 of the FTP protocol.

3 FTP in a Nutshell

In contrast to other protocols that use the same TCP connection for both session control and data transfers, the FTP protocol uses a separate connection for the file transfers and directory listings. That means, an FTP client connects from a random unprivileged port ($N > 1024$) to the server on port 21 which is used as the control channel, i.e. exchanging commands listed in Section 4. For each data transfer the client initiates (using **RETR**, **STOR** or **LIST** command), the client first sends the **PORT N+1** command to the server, specifying what client-side port the server should connect to for the upcoming data transfer. The client then starts listening to port $N+1$ and the server connects from port 20 to the client port ($N+1$) to establish the data channel. Once the TCP connection is established, the data transfer is then made through the data channel. The end of the data transfer must be indicated by closing the data channel. Consequently, for every subsequent data transfer (e.g. file request), the client must send the **PORT** command again to the server to announce a new port number, e.g. $N+2$, $N+3$, etc.. Ports may be reused after some time. **Please note that your implementation should only use the active FTP mode.** The passive mode does not need to be implemented.

Once the client disconnects (**QUIT** command), the TCP control channel must be closed and the client should terminate.

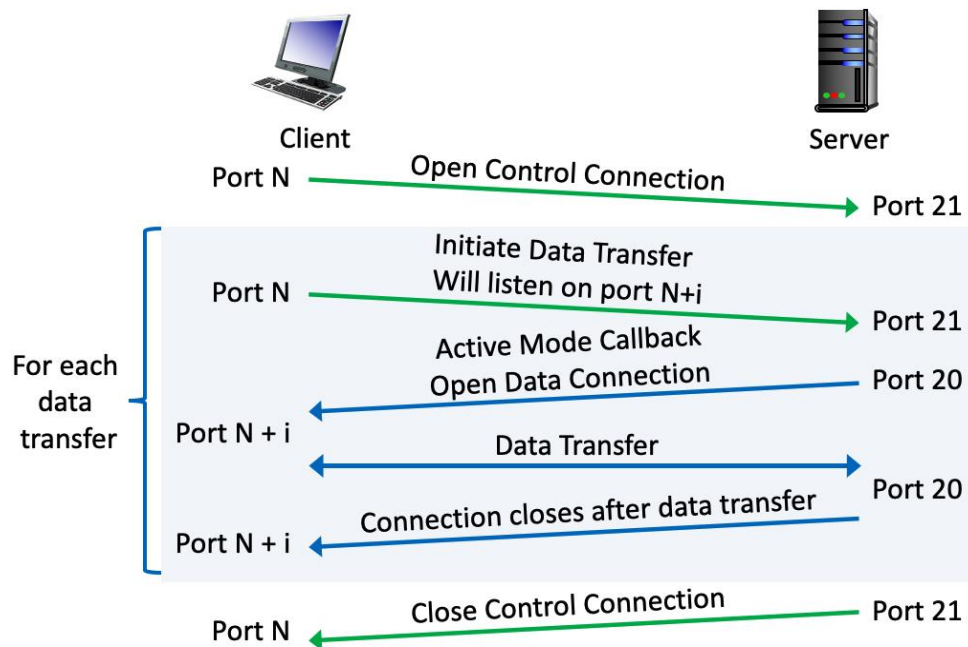


Figure 1: Active FTP Mode: Server initiates data callback connection

4 FTP Commands and Replies

After the establishment of the control TCP connection with the FTPserver, the client process should respond with a prompt *ftp>* waiting for the user's ftp commands. The FTP RFC 959 details a large number of commands that the FTP protocol uses. In this project, we use only a subset of these commands and restrict the transfer mode to stream mode. Note that the commands that start with a **!** are executed locally at the client and should execute the corresponding shell commands through the `system()` function. Below is a list of all the required commands that should be implemented.

PORT h1,h2,h3,h4,p1,p2 This command is used to specify the client IP address and port number for the data channel and is sent automatically by the client before sending a **RETR**, **STOR** or **LIST** command. The server must reply with "200 PORT command successful." before starting the data transfer. The argument is the concatenation of a 32-bit internet host address and a 16-bit TCP port address. This address information is broken into 8-bit fields and the value of each field is transmitted as a decimal number (in character string representation). The fields are separated by commas, where **h1** is the high order 8 bits of the internet client address and **p1** is the high order 8 bits of the client port.

You can use the following formulas to convert the port into a character string representation and vice versa:

```
// convert port to p1 and p2
p1 = port/256
p2 = port%256
// convert p1 and p2 to port
port = (p1 * 256) + p2
```

USER *username* This command identifies 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. Once the client issues a **USER** command to the server and the user exists, the server must reply with "331 Username OK, need password.", otherwise with "530 Not logged in."

PASS *password* The client needs to authenticate with the user password by issuing a **PASS** command followed by the user password. The server must reply with either of the following reply codes: "230 User logged in, proceed." or "530 Not logged in.". The user names and passwords should be saved locally in a file called users.txt and should be read by the server program upon launch. The file must contain an entry for the username bob with the password donuts.

STOR *filename* This command is used to upload a local file named filename from the current client directory to the current server directory.

RETR *filename* This command is used to download a file named filename from the current server directory to the current client directory.

LIST This command is used to list all the files under the current server directory.

!LIST This command is used to list all the files under the current client directory.

CWD *foldername* This command is used to change the current server directory. The server must reply with "200 directory changed to pathname/foldername."

!CWD *foldername* This command is used to change the current client directory.

PWD This command displays the current server directory. The server must reply with "257 pathname."

!PWD This command displays the current client directory.

QUIT This command quits the FTP session and closes the control TCP connection. The server must reply with "221 Service closing control connection." and the client must terminate after receiving the reply. If this command is issued during an ongoing data transfer, the server/client must abort the data transfer. Note that the **RETR**, **STOR** and **LIST** commands trigger a data transfer between the client and the server. That is, the server replies with "150 File status okay; about to open data connection." and opens a new TCP data connection to transfer the data. In the control connection, the server creates the socket (on port 21) to which the client binds to open the connection with the server. In the case of data transfer, clients create the sockets on port no: **PORT N + i** (port N is the client side port that it used before to connect to the server), and the server connects to it using bind for opening the data connection with the client. The server connects from port 20 to the client port received (**N + i**) with the **PORT** command and starts transferring the data (**RETR** and **LIST**) or waits to receive a file (**STOR**). If the transferred file already exists, it is replaced. Since this implementation (for simplification purposes) will use the stream transfer mode of FTP, the end of the file is indicated by closing the data connection. That is, the sender (server for **RETR** command, client for **STOR** command) closes the data connection when the file is

completely transferred and the server must send a "226 Transfer completed.". Please note that this mode is inherently unreliable, since the client can not determine if the connection closed prematurely or not, i.e. if the file has been entirely transferred or not. You can ignore this unreliability in this implementation. However, since FTP is a stateful protocol, a session must be reset if the client disconnects. Also, since the server must support concurrent file transfers of multiple connected clients and a concurrent file transfer of the same file (or with the same filename) could be a possible scenario, it is highly recommended to save the received file content in a temporary file (e.g. rand#.tmp) or temporary folder during the data transfer and renaming the file to its actual filename after the transfer completed.

Also, the FTP server must be able to respond with appropriate error messages when an error occurs:

- Any command other than the ones above should not be accepted and should be considered as an invalid FTP command. The server must respond with a "202 Command not implemented."
- The authentication process consists of both a **USER** command and a **PASS** command. No other command should be accepted by the server if the user has not successfully authenticated itself and should result in a "530 Not logged in." response.
- If the user requests an invalid filename or if the directory the user is trying to navigate into does not exist, the client/server must respond with a "550 No such file or directory."
- If the user issues commands in an invalid sequence, the server must respond with a "503 Bad sequence of commands."

Your submission **must** be working successfully on a Linux Operating System (either the GUI of the remote Linux Server shown in the lab or using VirtualBox with an Ubuntu image). You should follow the guidelines for the file structure and expected output format demonstrated in the lab session.

5 Example of an FTP Session

```
220 Service ready for new user.
ftp> USER bob
331 Username OK, need password.
ftp> PASS donuts
230 User logged in, proceed.
ftp> CWD test
200 directory changed to /Users/bob/test
ftp> RETR vanilla_donut.txt
200 PORT command successful.
150 File status okay; about to open data connection.
226 Transfer completed.
ftp> LIST
200 PORT command successful.
150 File status okay; about to open data connection.
vanilla_donut.txt
choco_donut.txt
226 Transfer completed.
ftp> QUIT
221 Service closing control connection.
```

6 Grading

Description	Score (/20)
Successful compiling of the programs using a Makefile	1
Properly displaying and formatting of the command line user interface	1
Properly handling user authentication	1
Handling concurrent connections and supporting multiple simultaneous users	3
Properly implementing PORT command	2
Properly executing FTP commands and sending correct reply	5
Properly transferring files (binary, plaintext) of any type and size	3
Properly closing the TCP connections (control and data channel)	2
Coding style and usage of meaningful comments	2

7 Submission Details and Policy

Submission Deadlines:

1. Group formation due: April 21, 2025 (-10% penalty if not met)
2. Final submission due: May 7, 2025

Submission Format and System: You can directly submit your files as a zip file on Brightspace (<https://brightspace.nyu.edu/>). Due to technical limitations, submissions via email are not accepted.

Late Submissions: Late submissions will be penalized by 10% per 24 hours, with a maximum of 3 days late. In case of a late submission, please upload your zip file to Brightspace.