# CS472 Computer Networks
# Fall 2020 – 2021
# Homework Assignment #3
# Due Date: Tuesday, November 3rd, 2020 at 6:29pm (CLASS TIME)

NOTE: Assignments must be submitted in electronic format via Drexel Learn. All the work must be original, NO TEAM WORK. Late assignments will not be accepted. Please submit your assignment as your first initial and last name as a zip file with all files needed (e.g. mine would be mkain_hw3.zip)

## Objective

This assignment asks you to implement a popular network protocol as a server, rather than a client. The idea is for you to be able to implement the specifications of an RFC and identify the significance of the different states the protocol transits through as the task being achieved by the protocol is being performed. RFCs are definitions of protocols that all clients and servers must follow. You will be writing a server which is to interact with the client you have already written to adhere to the protocol.

## Problem

In this assignment, you are to implement a FTP CONCURRENT server (any of the three models discussed in class are acceptable).

You are to write the client code in C, C++, Java or Python ONLY. Other languages must be approved by the Professor before implementation. Any questions – ASK!

Libraries such as the libnsl, libresolv, and libsocket with C/C++, the package java.net (javax.net), and the Python socket library are the ONLY permissible libraries that you can use for socket programming. In case you do need to use some other library, PLEASE CONFIRM WITH THE PROFESSOR FIRST. **Using other libraries without permission will cause a significant loss of points on the assignment.** This is to assure that you get the right goals out of the assignment (same rules as the client).

It is your responsibility to ensure that the code runs on tux.cs.drexel.edu or in a suitable environment (like eclipse, java or Microsoft Visual Code). Code that doesn't compile/run will be graded as a zero. ALL source code must be available for inspection as well as any makefiles/build scripts. You must also specify where the program will run (tux, etc.).

Please consult the syllabus about plagiarism. You must write the code from scratch. You may not "port" other code that you find on the Internet. If we find the code that you use, you will get a zero on the assignment. **IF YOU CAN FIND IT, SO CAN WE.**

## Input / Command Line

The server program should accept the following command line arguments:

The first argument is REQUIRED and is the filename that logs all the messages received by and generated by the server. You should also log any server actions (including initialization, authentications, and errors). Each message is to be logged in a single line. The message should include some timestamp, client identification (IP address and port number) and the data to be sent to or received from the client. Take the log format from the first part of the assignment and enhance it to include the client identification.

The last argument is REQUIRED and denotes the port number for the server to run on (since there already is an FTP server running on port 21, you will have to specify a different number).

A sample command line would be:

```
Ftpserver serverlog.txt 2121
```

This would run the ftpserver offering its service at port 2121 and log all events into the file serverlog.txt.

NOTE: Your port number may also conflict with other students developing this assignment – so you may have to try several port numbers until you find one that's not being used (above 1024).
ANOTHER NOTE: Tux is a cluster (a group of hosts), so it will act weirdly with FTP (because of the addresses being used).

## Protocol details

Refer to RFC 959 (http://www.rfc-editor.org/pdfrfc/rfc959.txt.pdf) for protocol messages and semantics. Refer to RFC 2428 (http://www.rfc-editor.org/pdfrfc/rfc2428.txt.pdf ) for details about EPRT and EPSV command details.

Your server should be able to respond to all the commands from your client program by implement the following features:

- It must set up a listening socket to accept connections.
- Correctly handle ALL error conditions, non-supported commands and generate a message to the server log as well as send the appropriate message back to the client.
- Catch all exceptions generated by your code and calls to the system. FAULTING IS A BAD THING.
- Your server need only to respond to commands generated by the client including:
  USER, PASS, CWD, CDUP, QUIT, PASV, EPSV, PORT, EPRT, RETR, STOR, PWD, LIST.
- Your server must handle any errors returned from these commands and generate the appropriate error text to the server log.
- There IS NOT a real user interface, since once the server is started, all data is logged to the logfile or sent back to the client via the socket (real servers are started by the system).

- Authentication (what usercodes and passwords work) should be simulated by using a separate file (e.g. accounts) which your program reads when it starts up [the format is up to you].
- **You MUST maintain the state of each connection (and check it to ensure that the command received is valid for the current state) of each client. You must check to make sure that the client is talking to the server correctly. This will stop hacking and fuzzing problems. You will have to write a DFA to implement this part.**

**STILL ANOTHER NOTE: The "state diagrams" in RFC 959 on pages #54-58 are NOT DFAs. Refer to the class notes and examples for a real DFA.**

## Testing

You should test your server with both your client (from homework #2A) as well as the standard ftp client to make sure that BOTH run successfully.

## Questions to also be turned in

1. Consider a peer-to-peer file transfer protocol. Do you think it would be better or worse? Why? Give details to backup your argument.
2. Do you think that someone could hack into your FTP server? Why or why not? Through what methods?
3. EXTRA CREDIT: Critique how FTP solves the problem of file transfer – what is good? What is bad? What is weird?
4. EXTRA CREDIT: How hard would it be to make FTP use one channel (both command and data over the same socket) instead of two?

## Your submission

Your submission MUST contain the following:

- Well documented code (VERY WELL documented code) that should compile correctly – see guidance in Hw2 for what I expect.
- A Readme.txt file detailing instructions to compile your code or the use of your makefile and how to run your code.
- Server log from a sample run using both your client and the released FTP client.
- The answer to the questions above.
- Any other information you deem important (like your name, etc.)

## Point Sheet

The goal of this assignment is a server which never fails, not “just works”

| **Item** | **Points** |
|---|---|
| Command line parsing / Server algorithm | 5 |
| Check and handle all error conditions | 5 |
| Simple commands and responses (2 pts @) | 12 |
| Data transfer commands & responses (5 @) | 30 |
| Logging done and sample turned in. | 10 |
| Well documented code | 10 |
| Compiles correctly (if you miss here, you’ll probably also miss other points) | 5 |
| README with instructions | 5 |
| Answers to above 2 questions | 8 (4@) |
| **Maintain and check state (and submit DFA)** | 10 |
|  | 100 |
| Extra Credit = 10 (5@) | 10 |
| Total | 110 |