[redacted] Introduction to Computer Networks

Programming Assignment #2
Due Sunday, end of Week 9, by 11:59 pm.
Submit the source files, Makefile, and README in a .zip file to Canvas.

Objectives:

1. Implement 2-connection client-server network application
2. Practice using the *sockets* API
3. Refresh programming skills

The Program:

Design and implement a simple file transfer system, i.e., create a file transfer server and a file transfer client. Write the *ftserver* and the *ftclient* programs. The final version of your programs must accomplish the following tasks:

1. *ftserver* starts on Host A, and validates command-line parameters (<SERVER_PORT>).
2. *ftserver* waits on <PORTNUM> for a client request.
3. *ftclient* starts on Host B, and validates any pertinent command-line parameters. (<SERVER_HOST>, <SERVER_PORT>, <COMMAND>, <FILENAME>, <DATA_PORT>, etc…)
4. *ftserver* and *ftclient* establish a TCP control connection on <SERVER_PORT>. (For the remainder of this description, call this connection *P*)
5. *ftserver* waits on connection *P* for *ftclient* to send a command.
6. *ftclient* sends a command (`-l` (list) or `-g <FILENAME>` (get)) on connection *P*.
7. *ftserver* receives command on connection *P*.

   If *ftclient* sent an invalid command

   - *ftserver* sends an error message to *ftclient* on connection *P*, and *ftclient* displays the message on-screen.

   otherwise

   - *ftserver* initiates a TCP data connection with *ftclient* on <DATA_PORT>. (Call this connection *Q*)
   - If *ftclient* has sent the `-l` command, *ftserver* sends its directory to *ftclient* on connection *Q*, and *ftclient* displays the directory on-screen.
   - If *ftclient* has sent `-g <FILENAME>`, *ftserver* validates `FILENAME`, and **either**
     - sends the contents of `FILENAME` on connection *Q*. *ftclient* saves the file in the current default directory (handling "duplicate file name" error if necessary), and displays a "transfer complete" message on-screen

     or

     - sends an appropriate error message ("File not found", etc.) to *ftclient* on connection *P*, and *ftclient* displays the message on-screen.
   - *ftserver* closes connection *Q (don't leave open sockets!)*.
8. *ftclient* closes connection *P (don't leave open sockets!)* and terminates.
9. *ftserver* repeats from 2 (above) until terminated by a supervisor (SIGINT).

Program Requirements:

- *ftserver* must be written in C.
- *ftclient* must be written in Java or Python.
- Of course, your program **must be well-modularized and well-documented**.
- Your programs must run on a `flip` server: (*flip1*, *flip2*, *flip3*).*engr.oregonstate.edu*
    - Probably the best way to do this is to use SSH Secure Shell, Putty, or another terminal emulator to log onto *access.engr.oregonstate.edu* using your ENGR username/password and note which *flip* you get.
    - It will be easiest if you bring up two instances of the shell on the same flip server and use one to run the server, and the other to run the client (this is how I will be testing!).
- You may not use *sendfile* or any other predefined function that makes the problem trivial.
- Your program should be able to send a complete text file. You are not required to handle an "out of memory" error. Separate grading for short text files and long text files.
- Use the directories in which the programs are running. Don't hard-code any directories that might be inaccessible to the graders.
- If you use additional include-files or ***make-files***, be sure to hand them in with your program.
- Create a README containing detailed instructions on how to compile and run your server and client.
- Don't ZIP things up, you should only be submitting 5 files, maximum. I don't want your whole directory structure (Mac people).
- Be *absolutely* sure to cite any references and credit any collaborators. I'm sick of giving failing grades for people not doing this.

Options:

There are many possibilities for extra credit. All extra credit must be documented and referenced in your program description and README.txt. Here are a few ideas to get you started:

- Make your server multi-threaded.
- Implement username/password access to the server.
- Allow client to change directory on the server.
- Transfer files additional to text files (e.g. binary files) (a text file with a non-.txt extension doesn't count.
- etc…

Notes:

- *Beej's Guide* will be helpful. It has many things you'll need for this assignment.
- Don't hard-code the port numbers
- Don't use the well-known FTP port numbers, or 30021 or 30020, as these will be probably in use (by network services or other students).
- We will test your system with text files only (unless your README specifies additional file types), one very large and one small.
- If you implement extra credit features, be sure to fully describe those features, and how to use them, in your README, or you won't receive any extra credit.
- Programs will be accepted up to 48 hours late with a 10% penalty per 24-hour period.

Example Execution:

SERVER (flip1)		CLIENT (flip2)	
Input to console	*Output*	*Input to Console*	*Output*
`> ftserver 30021`			
	`Server open on 30021`		
		`> ftclient flip1 30021 –l 30020`	
	`Connection from flip2.`		
	`List directory requested on port 30020.`		
	`Sending directory contents to flip2:30020`		
			`Receiving directory structure from flip1:30020`
			`shortfile.txt longfile.txt`
		`> ftclient flip1 30021 –g shortfile.txt 30020`	
	`Connection from flip2.`		
	`File "shortfile.txt" requested on port 30020.`		
	`Sending "shortfile.txt" to flip2:30020`		
			`Receiving "shortfile.txt" from flip1:30020`
			`File transfer complete.`
		`> ftclient flip1 30021 –g longfileee.txt 30020`	
	`Connection from flip2.`		
	`File "longfileee.txt" requested on port 30020.`		
	`File not found. Sending error message to flip2:30021`		
			`flip1:30021 says FILE NOT FOUND`
		`>`	