CS472 Computer Networks
Fall 2015-2016
Homework Assignment #2
Due Date: Monday, October 19th, 2015 at 5:59pm (CLASS TIME)

NOTE: Assignments must be submitted in electronic format via Drexel Learn. The work must be original, NO TEAM WORK. Late assignments will NOT be accepted. Please submit your assignment as your firstname and lastname as a zip file with all files needed (e.g. mine would be mkain_hw2a.zip)

Objective

This assignment asks you to implement a popular network protocol. The idea is for you to be able to interpret and 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 client which is to interact with a server which has already been written to adhere to the protocol.

Problem

In Part A of the assignment, you are to implement a FTP (File Transfer Protocol) client that can login, list directory information, and retrieve information from a server hosting the FTP service.

You are to write the client code in C, C++, or Java 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++ and the package java.net are the ONLY permissible libraries that you can use for socket programming. In case you do need to use some other library, 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.

It is your responsibility to ensure that the code runs on tux.cs.drexel.edu or a suitable environment (like eclipse or java). 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.

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 client program should accept the following command line arguments:

The first argument is a REQUIRED argument and is the name of the FTP server host.

The second argument is required and is the filename that logs all the messages generated by the client and received from the server. Each message is to be logged in a single line. The message should include a full timestamp and the data sent or received. Data transfers do not have to be logged (but it would be nice), but some indication of the data transfer should be logged (opening and closing port). This file should be appended to with each run of the client.

A sample log file could look like:

> 9/25/12 20:00:00.0002 Connecting to tux.cs.drexel.edu (129.25.8.226).
> 9/25/12 20:00:00.0011 Received: 220 FTP Server ready.
> 9/25/12 20:00:00.0031 Sent: USER cs472
> 9/25/12 20:00:01.0099 Received: 331 Username received, please send the Password.
> <and so on>

The last argument is optional, and if specified, denotes the port number to connect to other than the default. If not specified, the default port number of 21 should be used.

A sample command line would be:

> Ftpclient tux.cs.drexel.edu mylog 2121

This would run the ftpclient connecting to tux.cs.drexel.edu at port 2121 rather than port 21.

Another sample command line would be:

> Ftpclient landsend.cs.drexel.edu mylog

This would run the ftpclient connection to landsend.cs.drexel.edu at port 21.

Protocol details

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

Your client should implement the following features:

- It must parse the command line and check all parameters.
- It must set up an appropriate connection with the server.
- Correctly handle ALL error conditions, non-supported commands, and cases like unreachable servers and errors in the client.

- Your client need only implement the following commands (and any responses that these commands generate):
  USER, PASS, CWD, CDUP, QUIT, PASV, EPSV, PORT, EPRT, RETR, PWD, LIST, HELP.
- When the client initiates, it should ask the user for his usercode and password and send those to the server (via the USER and PASS commands). If any errors are found, they should be noted and the client should terminate. If successful, a prompt should be shown which allows the other commands to be entered.
- The commands PASV/EPSV and PORT/EPRT and RETR are used for retrieving files. FTP uses a separate data connection to transfer files. You'll be telling the FTP server to use another port and then connect to that and send the RETR command to get the data. It will send the data through the other connection and you are to create the file locally.
- The client executes until the user enters the QUIT command.
- Your client must handle any errors returned from these commands and generate the appropriate error text to the local screen.

IMPORTANT: The user interface is up to you, but similarity to the current FTP client would be easy to understand. I DO NOT want the user to be entering the actual FTP commands, so I'd like to you have a simple UI and the program constructing/deconstructing the FTP commands/responses. Look at what the current ftp client does for an idea of what you should be doing.

Testing

I will be putting up a cloud instance for use for students who do not have access to another FTP server. I'll post the IP address and the usercode/password that work.
NOTE: You can use this instance for debugging, but you can also use any FTP server. You can even host you own.

Questions to be answered and turned in with the project.

1. Think about the conversation of FTP – how does each side validate the other (on the connection **and** data ports – think of each separately)? How do they trust that they're getting a connection from the right person?
2. Think about the conversation as a 1-1 client/server connection – does it scale to one to many or not?
3. How does your client know that it's sending the right commands in the right order? How does it know the sender is trustworthy?

Your submission

Your submission MUST contain the following:
- Well documented code (VERY, VERY, VERY WELL documented code) that should compile correctly. My rules are that code should be documented to the point that someone can pick up the code and understand it completely – procedure headings are recommended, and code comments are recommended for each code block.

- A Readme file detailing instructions to compile your code or the use of your makefile and how to run your code.
- Sample run and results
- Answers to the questions above.
- Any other information you deem important (like your name, etc.)

Point sheet

Points below are for the maximum for the concept – given to an excellent implementation.

Command line processing	5 points
Connection with server	5 points
PDU delineation	5 points
Commands & responses (4@)	44 points
All errors handled correctly	5 points
Well documented code	5 points
Readme included	3 points
Sample run and results	3 points
Logging implemented and submitted	10 points
Questions (5 @)	15 points
Total:	100 points