Programming Project

Simple File Server

<u>Due</u>: March 8, 2021, 11:59pm (as instructed below)

In this programming project, you will be writing (1) a simple file server that responds to client requests, and (2) a client that requests the file(s) from the server. You are given a sample file server and client in C from Figure 6-6 of the Tanenbaum & Wetherall textbook. The codes in Figure 6-6 will not work in a typical system. We have updated the codes and made them workable. Also, as an alternative, you are given the codes implemented in Java. Your task is to update the file server and client codes (whichever you choose – C or Java) to satisfy the features detailed below. The code package with a Makefile in it is available via Webcourses in a file named `file-server-files.zip`. Be sure to download that zip file first.

The Basic Assignment

For the basic part of the assignment, improve the file client and server codes as follows:

- Make the client write the received data bytes to a file with the name identified by the client's user. Your client must write the bytes to the file in order and without introducing any other characters or bytes. It must be able to successfully write a binary (not just text) file received from the server.
- Make the server print out debugging messages if a DEBUG flag is set to 1. Define this DEBUG flag as a command line argument and its default value should be 0 – indicating no debugging messages to be printed. When the server is run with this flag set to 1 (i.e., "`server DEBUG=1`"), the followings must be printed out by the server:
  - Before starting to transfer the file:
    ```
    Sending <filename> to <client's-IP-address>
    ```
    Hint: For obtaining the client's IP address, look at manual pages of the `accept()` socket API or Java Socket class documentation.
  - During the file transfer at every 10% progress:
    ```
    Sent 10% of <filename>
    Sent 20% of <filename>
    Sent 30% of <filename>
    …
    ```
    Note: Every 10% progression must be displayed even if sending of a single chunk of the file causes more than 10% progress.
  - When the transfer is complete:
    ```
    Finished sending <filename> to <client's-IP-address>
    ```
- Allow the client to add arguments that specify a byte range instead of requesting to download the whole file. The usage of byte range should be as: "`client <server-name> <file-name> -s START_BYTE -e END_BYTE`". Here, `START_BYTE` and `END_BYTE` will be positive integers and the user wants to retrieve this particular block of bytes of the entire file from the server. The `START_BYTE` and `END_BYTE` values must start from 1 and must be inclusive. For instance, if the user wants the range from the third byte to the 11th byte, `START_BYTE` and `END_BYTE` must be 3 and 11, respectively. Remember, byte flags are not mandatory for the user, and hence this command should still return the entire file: "`client <server-name> <file-name>`".
- Add error detection such that the server responds back to the client with an error if the requested file does not exist. Show error message if user requests for an invalid byte range.
- Add a client flag "`-w`" that allows the file to be written to the server. When the client is being executed from the command prompt, the flag will optionally be available to the user. The usage of the client process should be as: "`client <server-name> [-w] <file-name>`". If the "`-w`" flag is given, the server should interpret that as the file will be uploaded/written to the server side. Note that the server should check if file exists in its local directory first before it allows the client to send the file to the server side.
- Show help message if client gives wrong argument or fewer arguments. It's essentially printing some sample commands you're expecting from user.

The Advanced Assignment: Extending to a Web Server (extra credit)

Note that the following features are supposed to be additional. That means, after you add the Web server features, the server process should still be able to serve the file client's requests as described in the Basic Assignment above.

- (10 points) Make the server capable of responding to GET requests according to the HTTP 1.1 (http://www.rfc-editor.org/rfc/rfc1945.txt). Your server will then be able to properly respond to a regular web browser's file download request.
- (10 points) Make your server capable of handling two simultaneous GET requests.
- (15 points) Make your server capable of handling more than two simultaneous GET requests. There should not be any limit on the number simultaneous requests. You must use threads to do so.

Important Notes and Requirements

There will be a zipped directory (named as "file-server-files.zip") posted on Webcourses for this programming project. In that directory, you will find

(i) sample executables of the file client and file server (named as named as "client" and "server") which can be run on eustis.eecs.ucf.edu;
(ii) C codes of the client, server, and some includes (named as "client.c", "server.c", "file-server.h");
(iii) Java implementations of the client and server (named as "Client.java", "Server.java"); and
(iv) a sample makefile that will compile all the codes at once.

This is an individual assignment and you are expected to turn in your own work. Discussion with other students is allowed, but sharing your work is strictly not allowed.

You are supposed to do this assignment in UNIX environment. For developing as well as testing your code, you are given remote access to two Ubuntu machines: eustis.eecs.ucf.edu and eustis3.eecs.ucf.edu. You can use your own Linux/Unix box for developing the code, but we will test your code on these two machines. If your source code does not work on these machines, you will lose points. Your code must compile without any errors or warnings. There will be a heavy penalty if it doesn't. Your source code will be assessed and marked. Commenting is expected in your source code.

You can use programming languages other than C or Java, but you are on your own in implementing the sockets in the client and the server. Also, your code must still compile on the eustis machines. This means that you are limited to using the programming languages available on the eustis machines. For instance, C++ and Python are available on these machines.

Submission Requirements

A `makefile` is required. All files in your submission will be copied to the same directory, therefore do not include any paths in your `makefile`. The `makefile` should include all dependencies that build your executable(s). If a library is necessary, your `makefile` should also build the library. Unless you are using a fancy library, the sample `makefile` we provided you should suffice.

To make this clear: do not hand in any binary or object code (in java, .class) files. All that is required is your source code, a `makefile`, and other necessary files as stated in the assignment description. Do test your code by copying it into an empty directory and then compile it by entering the command `make`.

1. Go through the following steps to make your submission file ready to submit:
2. Go to your home directory and make a folder named with your name in format LastName_FirstName. Please DON'T name the folder as Project1 or something else. Strictly follow the format.
3. Put your source code and all other necessary files in this folder.
4. Write a ReadMe.txt file detailing the instructions to run your code and any other specific

requirements that your code might need.

5. In your home directory type and enter:
   ```
   tar -cvf <LastName_FirstName>.tar <LastName_FirstName>
   ```
6. Finally, type and enter:
   ```
   gzip <LastName_FirstName>.tar
   ```

These above steps will yield a new file named <LastName_FirstName>.tar.gz in your home directory.

To submit your file, log in to Webcourses and select the EEL 4781 course. Go to “Assignments” and choose “Project”. Attach your submission file and submit.

If you have any questions please post them on the Discussions board at Webcourses. Good luck!