Tentative Rubric for the File Server Project

1. Successful submission — 15 points
2. Code compile without error (This includes running the code on "eustis" machines as mentioned in assignment description and use of proper Makefile)—15 points
3. Code compile without warning—10 points
4. Correctly write the received data to the file—15 points
5. If DEBUG is set to 1, print out debugging messages—5 points
6. Correctly shows the progress while sending file (10%, 20%, 30%, …)—10 points
7. Correctly give the argument that specifies the byte range—10 points
8. Correct error detection—10 points
9. Correct use of flag "w"—10 points
   Bonus (as mentioned in assignment description)
10. Correctly responding to GET requests—10 points
11. Correct handling of two simultaneous GET requests—10 points
12. Correct handling of more than two simultaneous GET requests—15 points

Rubric Explained

The basic premise of the project is to create a file server that lets the user download files from the server and upload files to the server. There will be two sets of code that need to be completed, a client and a server, that will be written in the language of your choice and run on the eustis machines. The followings explain the reasoning behind the above rubric:

Steps 1-3: The code was submitted correctly with the makefile and can be compiled and run without errors or warnings.

Step 4: The client side can successfully download the exact file from the server side. This should mean that additional bytes are not added or removed from the downloaded file.

Step 5: The code should implement a debugging feature that can be turned on or off. If debugging is set, then it will print out the progress of the file transfer on the server side.

Step 6: The server side will correctly print out the percentage of the file that has been sent while it is sending it to the client. For example, 10% of file sent, 20% of file sent and etc. This will be shown when DEBUG is set to 1 (per Step 5). Those messages will be printed regardless of the size of the file, so if a specific byte range is selected (per Step 7), then it would still print the messages from 10% to 100% on the server side.

Step 7: The client side can request only a specific segment of a file. For example, if a file is 160 bytes and a client requested the byte range from 50–77. Then the server would only send that specific byte range of the file to the client.

Step 8: The client and server sides should implement error detection to protect against incorrect requests or unexpected errors. For example, if the requested file did not exist on the server side or the attempt to connect to the server did not succeed, an appropriate error message must be printed. Note that this error detection is independent of the DEBUG flag, i.e., even If DEBUG is 0 the error messages are supposed to be printed.

Step 9: The client should be able to specify that it is seeking to upload a file to the server. This is to be accomplished by using the flag "-w." And, the file must be properly uploaded to the server side.