Maiah Pardo
Cruz ID: mapardo

Writeup: Asgn1: httpserver.c

Testing:

- I tested my code using GET, PUT, and HEAD requests through curl.
- I tested with massively large binary and text files, smaller binary and text files, and mixed text files.
- I tested with incorrect header names, unaccessable files, files with names that are not permitted.

Questions:

- What fraction of your design and code are there to handle erros properly? How much of your time was spent ensuring that the server behaves "reasonably" in the face of errors?
    - I would say that probably 75% of my code is there to handle errors properly. The job of the handle_connection() function is almost purely to make sure that the client's request is valid, and if it is not, sends error messages. Then before sending or receiving data (for PUT, GET, or HEAD) I check for errors with opening, reading, or writing to the current file we are working with. Most of time on this project was spent making sure that my server behaves correctly when dealing with errors. I made sure that it sends the appropriate error number and message depending on the error.

- How does your design distinguish between the data required to coordinate/control the file transfer (GET or PUT files), and the contents of the file itself?
    - (Note: I take this question to mean that that the data we receive in our request content is: CONTROL_DATA\r\n\r\nCONTENT_DATA, where CONTENT_DATA is an optional body)
    - My design itself makes the distinction between the control data and content data: after I have made a connection with a client, I enter my handle_connection() function, where I handle the connection by checking that the request line and all headers (the control data) is valid. If it is valid, I move to the appropriate function (handlePut(), for example) to handle the content data. So, in my design I handle control data and content data in different functions (breaking the code into modules). I chose this design so that my code would be more readable, easier to find errors (if I am having a problem with only PUTS, for example, I can narrow that down to the handlePut() function) and to improve the manageability of my code. This project was very large (in terms of lines of code) so it was necessary to modularize the code so that I could have an easier time debugging and modifying it.

- What happens in your implementation if, during a PUT, the connection is closed, ending the communication early?
    - If, during a PUT, when we are receiving data, if the result of the recv() function returns -1, this means an error has occurred during the receive (possibly the client closed the connection) and I chose to close the connection and go back to listening for a new connection. I chose to do this because if the server is receiving data and writing it to a file, and suddenly receive never received any bytes and we try to write, we will have an error.

- Does endianness matter for the HTTP protocol? Why or why not?
    - Endianness does not matter for the HTTP protocol. Endianness refers to byte order and ascii values are not affected by endianness. We only send, receive, read and write ascii when we use the HTTP protocol (made up of a request, a response, and a possible body). Since we do not do any computations on numbers, the endianness shouldn't matter.