Writeup

I tested each and every one of my functions individually and ensured they worked before moving onto the next function. I first had to test to make sure that my client and server were connecting to each other. After that was done, I implemented the parsing of a http request. This took hours of testing in order to make sure that every part of the request was split up and stored in the proper place for use later. I tested every small change I made in this step, using lots of print statements in order to determine where I was facing bugs and errors. The same applies to the function I made to process the requests. Every time I made a change to the functionality of a certain part of the code, I would test it and make sure it worked before I progressed. If I got stuck on one part while testing, I would not stop until it started working. Obviously, when I added more things to these functions they would cause other parts of the functions to break. I would then go back and change it to accommodate for the newer changes I made and then test those changes as well until everything was working. I only tested my whole program together a few times, but since I knew that every other part worked as intended, I recognized that for this assignment it would be fine since I tested it heavily prior to finishing it.

What fraction of your design and code are there to handle errors properly? How much of your time was spent ensuring that the server behaves "reasonably" in the face of errors?

- Probably around 40% (maybe more) of my design and code is towards checking for errors. I have probably spent over 15 hours alone just checking for errors because I wanted the functionality to be complete. Every time I learned of an edge case to account for, I had to try and implement a way to error check for it. I also had to make sure my program wouldn't crash randomly. Every type of error had to be handled, or attempted at.

List the "errors" in a request message that your server must handle. What response code are you returning for each error?

- In a request message there can be tons of errors. The header may contain parts of the body which is an invalid request. The resource name cannot be more than 28 characters and it must be between the ASCII characters a-z, A-Z, 0-9, -, _. The header cannot be larger than 4096 bytes. Header can be sent separately in multiple parts. These types of errors would all return a 400 Bad Request.
- 403 Forbidden
    - don't have permission to access resource name in request message
- 404 Not Found
    - file with resource name does not exist
- 500 Internal Server Error
    - server fails because of syscall failure

What happens in your implementation if, during a PUT with a Content-Length, the connection is closed, ending the communication early?

- If the connection closed early, then the file that the client wanted to send to the server would not be received and saved by the server. This means that the contents of the file would not be written to the server (so an existing file would not be overwritten with new data, or a new file would not be created with that data).

Does endianness matter for the HTTP protocol? Why or why not?

- Yes, endianness matter for the HTTP protocol. We had to use htons() to convert the port number. It converts the unsigned short integer from host byte order to network byte order (man page for htons()), which implies that endianness does matter since the bytes needed to be altered.