Testing:

For testing, I first started out by testing if I could even make a connection to the server through the socket with curl. I mostly ran curl on my desktop directory while the server was ran inside of ankho/asgn1.

After I confirmed that the connection was established, I tested if I was parsing the data correctly. I did this by using the write() system call with file descriptor 1 in order to print the received header to STDOUT. After receiving the header, I began to split up the data and I also used the write() system call to print out the individual parts to STDOUT to check for correctness.

After testing that I parsed the header correctly, I tested my GET functionality. I tested this by using curl to send GET requests to my server and checking what my server sends back. I also wrote to STDOUT during this step in order to check the data that I was receiving from the client.

After testing the GET functionality, I tested my PUT functionality with similar steps to my GET tests. I used curl to send PUT requests to my server and used cat to check what was in the files that were created/overwritten. I also wrote to STDOUT in order to check issues with the data I was receiving from the client.

After all the individual parts were all tested, I did a full test of the entire workflow by putting my server in a loop so that it can accept multiple requests without closing. After opening the server, I ran a series of PUT requests to populate the server with files and to test error handling. After the series of PUT requests, I ran a series of GET requests to check the data in the files that my PUT requests created and updated. I also tested error handling for the GET requests during this phase.

Question:

Q: 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?

A: I think I spent around 15% of the time in my design and code deciding where to place the errors. Then I spent an additional 20% of my time ensuring that the server behaves “reasonably” in the face of errors. This included the time spent designing, implementing, testing, and debugging the errors.

Q: List the “errors” in a request message that your server must handle. What response code are you returning for each error?
A:
400 - Bad Request: When the resource name is not in valid 27-character format with only upper and lowercase letters, numbers, -, and _.
403 - Forbidden: When the client is trying to access a file that cannot be accessed.
404 - Not Found: When the client tries to GET a file that does not exist.
500 - Internal Server Error: When the client tries any request other than GET or PUT requests.

Q: What happens in your implementation if, during a PUT with a Content-Length, the connection is closed, ending the communication early?
A: In my implementation, if the connection is closed, ending the communication early, the file is created but nothing is written into the file. The curl receives an empty response from my server.

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

A: Yes, endianness matters for the HTTP protocol because the client and the server have to agree. If the server is in big endian, and the client is in little endian, then communication wont work correctly. I believe that in HTTP protocol, the client and server both agree to use big endian.