Testing:

The testing during this assignment was done periodically throughout the duration of the assignment in a form of print statements and if statements making sure the flow of code is right. The program was first coded to handle small files, this was because of the way recv() works. To test the code, I used curl to send HTTP requests. When the code was running right at that point, I just had to adapt it so that it can use recv() in a loop for the PUT request.

When I completed the code to handle bigger files, I used the scripts that were given through the discord. This was where big-sized files were pushed into the program using curl and checked with diff to see the difference in files that were uploaded or requested.

After running the scripts, I adapted my code to check for bad requests, this is where I started to implement the error handling for the program.

After I also tested the code for memory leaks using Valgrind. There were no memory leaks reported.

Questions:

- 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?

  About half of the code is to handle errors properly which are mainly within the handle_connection function. This would be the same for the time spent about half of the time. I spent checking if the server behaves reasonably in the face of errors.

- 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?

  My design for GET requests distinguishes between data required to coordinate the file transfer and the contents of the file itself by first reading over the file to get the number of files and then going back and reading the file again sending the actual bytes over.
  And my design for PUT requests distinguishes between data required to coordinate the file transfer and the contents of the file itself by first getting the bytes needed to read from the client's header and then parsing till the bytes that are written to the file exceeds the total bytes given.

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

  If the connection is closed, the code will close the file and remove the file. After this, the program will return to the listening position waiting for the next request.

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

  Endianness matters because for the HTTP protocol because the internet uses a different type of endianness than that your computer uses. So when having a connection using the HTTP protocol you have to change the endianness to the one used by the internet, this is so other computers can expect the type of endianness that they receive and how to convert it to their own type of endianness.