Despina Patronas
CSE 130

Write Up Document Assignment 1: httpserver

Testing

httpserver, should be similar to an http protocol server.

i. When Client sends curl requests (w/ GET, HEAD, or PUT) to *httpserver*, *httpserver* should perform the operation and respond back to client properly.

Testing Code for the spec:

Tests were done using Scripts provided by ***Christopher Lim*** and ***Donovan Henry***.

These tests include various curl commands for *httpserver* and compares expected outputs to *httpserver outputs*.

Example Client Sends:

1. curl -T file1 http://localhost:8080/file2 *(PUT)*

   diff file1 file2 should be identical

2. curl -s http://localhost:8080/file3 -o file4 *(GET)*

   diff file3 file4 should be identical

3. curl -I file1 http://localhost:8080/file5 *(HEAD)*

   should send appropriate response for file5

1) **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?**

Most of the time was spent making sure the program flow is correct for handling errors according to the spec.
Isolating portions of the program to check individually:

Request line checking
Put command
Get command
Head command

Then consolidated modules to clean up the code once everything seemed to run in order, appropriately, and handling errors before doing any processing.

2) **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?**

Short answer:
Based on the command from request line.

Long answer:
I have stored the request line parameters and use the method command to determine processing.

Processing IO:
1) (PUT). Infile is the body of connection fd socket.
Outfile as file requested to hold contents.

2) (GET). Infile is the file stored in server to read.
Outfile is connection fd socket.

Notice the PUT and GET infile and outfile are basically inverse of each other but process is the same.

Sending Request based on command:
For PUT the response is sent afterwards unless there is a bad request.
For GET the response is sent before the processing once request is validated.
For HEAD, it functions the same as GET but there is no need to do any processing of files, merely require validating the request.

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

If the communication ends from the client end during a PUT, the server will exit the processing loop and continue taking requests. The file will be partially filled with contents for the file creation / truncation requested. The Client will need to redo the request to get an accurate processing.

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

The endianness does not matter! ASCII based characters looks the same on both Client and Server ends regardless of hardware endianness. This is perfect for HTTP protocol since it is ASCII based.