CS 118: Computer Network fundamentals

Project 1: Web server Implementation

William Lee 904814655
Sangjoon Lee 004749872

1. Give a high-level description of your servers design.

   Our server first creates and bind socket. It then listens and accept client. It reads client's message, parses the get line and extracts file path Our server first creates the first socket that is only responsible for listening for connections (we allowed several pending connections to wait in queue). The server continuously loops through listening and accepting connections to make a new socket for that request non-persistently. Then server parses the GET request for file path, reads the file if it exists, and sends appropriate response header to the client socket and sends either the file or some other data depending file type, and its existence.

2. What difficulties did you face and how did you solve them?

   Some of the difficulties we faced include:
   - Substituting %20 with space in C instead of Python
     - ***Solution: We used temporary buffer to recreate substitution function***
   - Storing arbitrary sized request into a buffer
     - ***Solution: For reading the request, we keep track of the state until we have an empty line signaling end of request.***
   - Formatting response in to variable sized buffer by finding file size.
     - ***Solution: Create the header first by determining file size with stat method. Then write file to client socket.***
   - Parsing through the sections of the request for the file name and allocating/reallocating memory
     - ***Solution: Keep track of sections of the request as a "state" and get the file path when in its corresponding "state"***

3. Include a brief manual in the report to explain how to compile and run your source code (if TAs cant compile and run your source code by reading your manual, the project is considered not to be finished).

   1. Untar the tar file
   2. Run "make" - returns an executable "http_server"
   3. ./http_server <port_num> (substitute <port_num> with actual port number)

4. Include and briefly explain some sample outputs of your client-server (e.g. in Part A you should be able to see an HTTP request). You do not have to write a lot of details about the code, but just adequately comment your source code. The cover page of the report should include the name of the course and project, your partners name, and student id.

Sample output:

After running ./http_server 8000
Type http://127.0.0.1:8000/cat.gif in web server (i.e firefox, chrome)
this outputs (on terminal):

```
GET /cat.gif HTTP/1.1
Host: localhost:8000
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6)
AppleWebKit/604.4.7 (KHTML, like Gecko) Version/11.0.2 Safari/604.4.7
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive

GET /favicon.ico HTTP/1.1
Host: localhost:8000
Connection: keep-alive
Accept: */*
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6)
AppleWebKit/604.4.7 (KHTML, like Gecko) Version/11.0.2 Safari/604.4.7
Accept-Language: en-us
Referer: http://localhost:8000/cat.gif
Accept-Encoding: gzip, deflate
```

The browser made multiple requests one for the file itself, then for favicon.ico.
Note this is so that web browser can display small icon next to http address (i.e. G for google)

Our sample output of the cat.gif file shows following:

Line 0: GET followed by URL and HTTP version
Line 1: destination ip and port
Line 2: client's preference for an encrypted and authenticated response,
Line 3: file types that are accepted
Line 4: web browser that sent this request

Line 5: language choice (en-us)
Line 6: the type of connection browser is requesting for

Note: each line except the request body ended in carriage return and line feed.

Since we have cat.gif, our server returns that file, but since we don’t have favicon.ico, we return 404 to the browser for favicon.ico