CS 118 Project 1

Josh Camarena 00454718
Kuan Xiang Wen 004461554

1. Description of Server Design

We used a single file "server.c" that, when run, starts a webserver. It is non-persistent – the server only handles a single HTTP request and then returns. We have also provided "index.html" and "404.html" to handle the default and File DNE cases.

The server awaits a HTTP GET request, and returns a single HTTP response. To request a file from the server, type "(IP ADDRESS):(PORT NO.)/(FILENAME)" into an up-to-date Firefox browser. It can return the following filetypes: *.htm/*.html, *.jpg/*.jpeg file, *.gif file. If the filename is nonempty but has does not have any of the above filetypes, it returns a binary file that prompts a download. If the field is empty, it looks to download "index.html". If the file is not found, it returns "404.html with a 404 error".

After it serves a request, the server closes all sockets and returns/shuts down. To retrieve another file, the server binary should be run again.

High-level functions details:

main(): Sets up socket connection and starts server (based entirely off sample code). Calls response() and closes socket and returns right after.

error (called by every other high-level function): Simple error handling function that prints a message to stderr and exits.

parse_file_req(): (Parse File Request) Parses the HTML GET request and prepares the file to send to client. Also sets string values for parts of the HTTP response header.
Steps:

1) Extracts the string (the filename) between "GET /" and "HTTP/1.1\r\n"
    a. If string is empty, transform string to "index.html"
2) Replaces "%20" in the string with spacebars
3) Finds the file type of that string, turns it into lowercase, then prepare the suitable string for the Content-Type field in the HTTP response header.
4) Try to import the file (with lowercase file type) and prepare the file descriptor.
    a. Also prepares the HTTP response code (eg. 200 OK, 404 Not Found)
5) If import fails, change the file to upload to 404.html
6) Find the size of that file
7) Returns the file descriptor (int)

response(): Handles server input/output. Calls parse_file_req()
Steps:

1) Reads the HTTP GET request from the client into a buffer.
2) Passes that buffer into parse_file_req().
    a. parse_file_req() will set global values content_type, content_length, and content_response_code
3) Writes the HTTP Header using the global variables above.
4) Reads from the opened file to a buffer with size equal the file size

5) Writes the buffer into the socket
6) Returns.

2. Challenges

1. Messages were appearing twice to either the server terminal or to the webpage.
    a. Solution: zero the write buffer after data is used, otherwise the same data possibly would be written twice to the page or terminal.
2. HTML file written to browser as raw HTML code with tags.
    a. Solution: Added code to write the correct HTTP header, and the HTML displayed correctly
3. Special case for if no file is requested (e.g. only a '/' is provided)
    a. Set the default file as 'index.html' to avoid seg faults by using the string 'index.html'. Before arriving at this solution, we tried not processing the empty file name string, but seg faults kept occurring and many if statements had to be added. In our solution, we replace the empty file name string with 'index.html' to make the processing cleaner and avoid segfaulting.
4. Formatting the response header correctly and efficiently
    a. Many dynamic parts to the request string, varying in name and length. We used a lot strcat and sprintf to create the response header
5. Used a fixed buffer to write the file content, but the images / binary were not processing correctly
    a. Replaced the fixed buffer with malloc, using the file size from fstat. This allows all the data to be in one buffer. HTTP payloads can be of any size.

3. Manual

1. Download "004454718.tar.gz" into directory with the test files
2. Open a terminal and *cd* into the directory with the files
3. Untar the submission with *tar -xzf 004454718.tar.gz*
4. Run *make* to compile the "server" executable
5. To run the server, run *./server port* where port is the port number the server will listen to
6. Open a web browser on the same machine and enter *127.0.0.1:port/filename* to open the *filename* in the web browser.
    a. text/HTML, jpg/jpeg, and gif files will be opened in the web browser. Binaries files without extensions will prompt the user to download them
7. The server is not persistent. Loop through steps 5 and 6 with different port numbers to test different files.

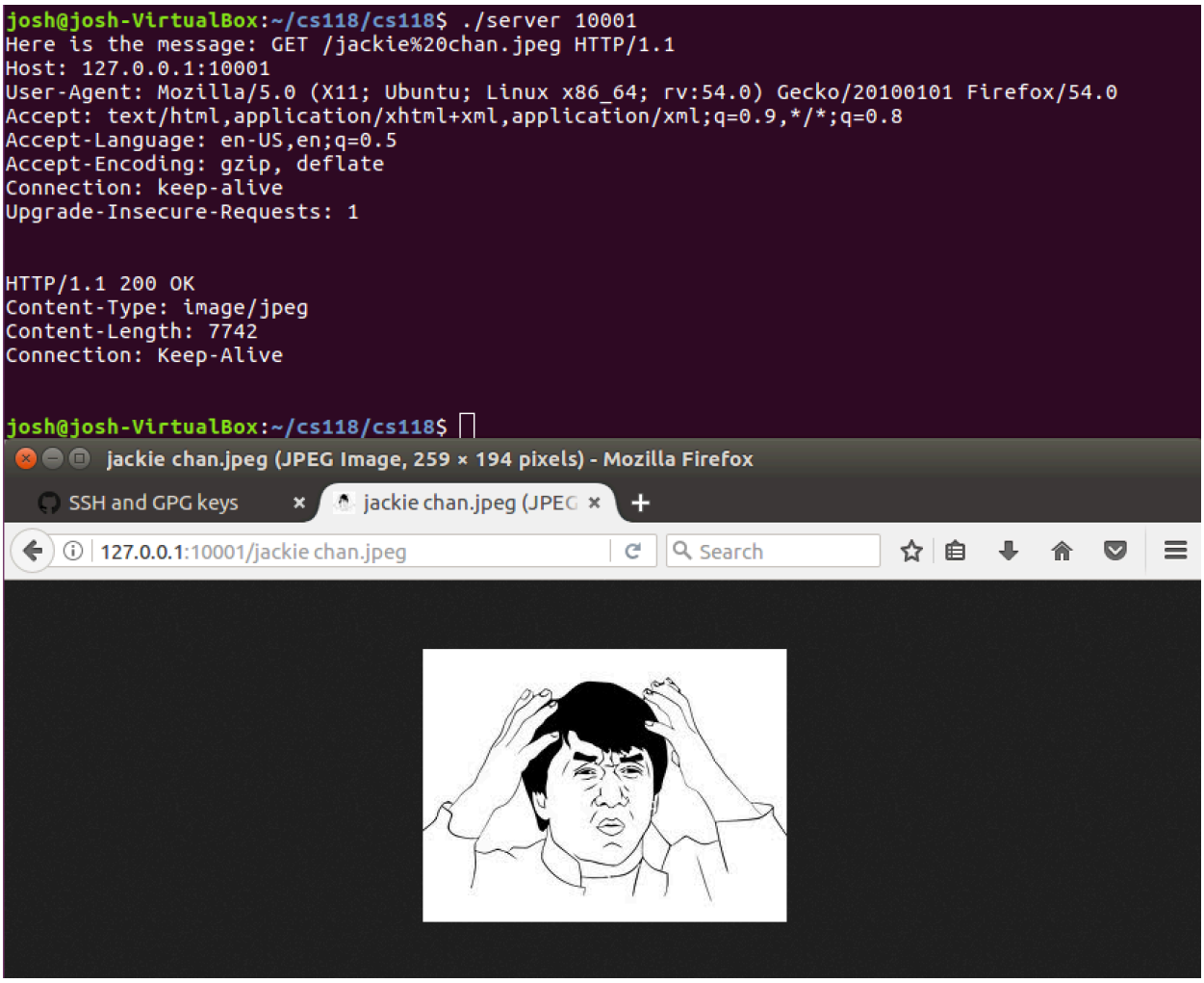
4. Examples

*The HTTP response header was also output to the console to provide another measure for the server working correctly.

File name with a space

./server 10001

127.0.0.1:10001/jackie chan.jpeg

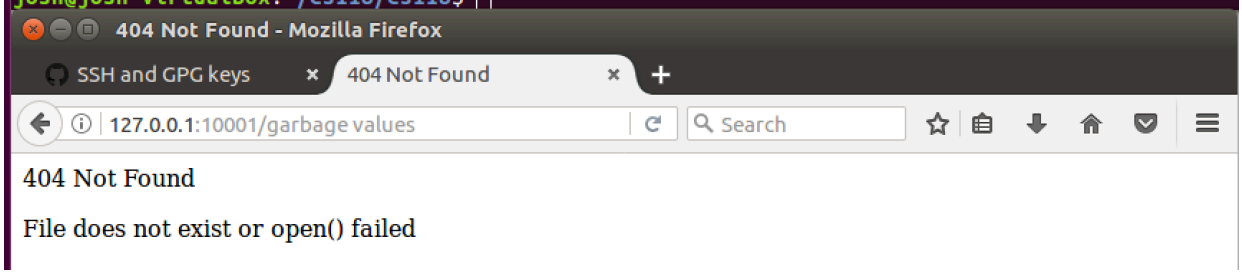


Garbage value file name that does not exist returns 404 page

```
josh@josh-VirtualBox:~/cs118/cs118$ ./server 10001
Here is the message: GET /garbage%20values HTTP/1.1
Host: 127.0.0.1:10001
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:54.0) Gecko/20100101 Firefox/54.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1


HTTP/1.1 404 Not Found
Content-Type: text/html
Content-Length: 141
Connection: Keep-Alive


josh@josh-VirtualBox:~/cs118/cs118$
```



gif file type test (not seen in still image: it animating correctly)

```
josh@josh-VirtualBox:~/cs118/cs118$ ./server 10001
Here is the message: GET /mindblown.gif HTTP/1.1
Host: 127.0.0.1:10001
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:54.0) Gecko/20100101 Firefox/54.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1


HTTP/1.1 200 OK
Content-Type: image/gif
Content-Length: 4584014
Connection: Keep-Alive


josh@josh-VirtualBox:~/cs118/cs118$
```