CS 118
Project 1:
Concurrent Web Server using BSD Sockets

Alvin Lim, 804011675, alvinl
Tremaine Eto, 904171710, tremaine

I. High Level Description

In thinking about how to design our server, we first broke down the cases accordingly. For our purposes, the HTTP status codes possible would be 403 (bad permission), 404 (file not found), and 200 (OK). In our main() function, we thus listen and allow five simultaneous connections at most. We then define a buffer for reading in the client's message and then our own response buffer, which we can re-size later with realloc().

```
char buffer[256];
char *response_buffer, *file_name;
response_buffer = (char*) calloc(buffer_length, sizeof(char));
```

Once we accept, we then create the response accordingly by calling our own parse() function.

```
parse(buffer, &response_buffer, &buffer_length);
```

Within the parse function, we thus account for, firstly, the case in which the filename that is requested is not valid or found (thus, a 404 HTTP status code). Before we do this, however, we first tokenize the response by spaces and then by individual lines by looking for the carriage return character '\r'.

```
char *req_type, *file_name, *html_version;
req_type = strtok(buffer, " ");      // tokenize by spaces
file_name = strtok(NULL, " ");
file_name++;     // Get rid of the slash in first char
html_version = strtok(NULL, "\r");  // look for carriage returns and tokenize by them
```

At this point, we first included the case where we have identified that the file name is not found via:

```
if (access(file_name, F_OK) == -1)
```

We then explicitly write the "HTTP/1.1", 404 status code, and server information to the response buffer. To write the date and time, we utilize the time.h C library in order to access the system's (1) day of the week, (2) date of the month, (3) month, (4) year, (5) hour, (6) minutes, and (7) seconds. We had to account for the fact that the day of the week returns an integer response that corresponds to the number of days after Sunday as well as the date of the month returning an integer that represents the number of months after January.

```
char int_string[5];  // To convert integers to strings (year = need 5)
int n;
n = sprintf(int_string, "%d", date_and_time->tm_mday);
strcat(*response_buffer, int_string);  // change second argument to string
// Month
switch (date_and_time->tm_mon) {                    // converts integer (months after January) to strings
case 0:
strcat(*response_buffer, " Jan ");
break;
case 1:
strcat(*response_buffer, " Feb ");
break;
// more months...
strcat(*response_buffer, " Dec ");
break;
}
```

After this, we explicitly write the time zone, content-type, content-length, and 404 HTML error message to the buffer.

If the filename *was* valid, however, we have our other case. Firsthand, however, we account for a permission error (403) in the case that the server could be reached but refuses to take any further action. If that is not true, however, then we have the valid 200 OK case. We explicitly write the connection info and server info to the buffer and then write the date and time using the same logic as in the file not found case.

However, since the file was found, we have to understand what type of file it is. We thus determine if the file is text (‘t’) or an image (‘i’). If it’s an image, then we check the first character of the file extension—if it’s a ‘j’, then we know it is a .jpeg. If it’s a ‘g’, then we know it is a .gif. Finally, we acknowledge if there is no extension; in that case, it simply is text/html.

```
int i;
char cont_type = 't';
for (i=0; i < strlen(file_name); i++) {      // checking to see what type of file the extension is
if (file_name[i] == '.') {
switch (file_name[i+1]) {
case 'j':                   // checking for the j in jpeg
            cont_type = 'i';        // "image"
            strcat(*response_buffer, "Content-Type: image/jpeg\r\n");
            break;
case 'g':                   // checking for the g in gif
            cont_type = 'i';        // "image"
            strcat(*response_buffer, "Content-Type: image/gif\r\n");
            break;
default:                    // default is HTML
            cont_type = 't';        // "text"
            strcat(*response_buffer, "Content-Type: text/html; charset=utf-8\r\n");
            break;
}
```

```
break;
}
```

As in the other case, we then write to the response buffer the content-length and date information. We also then check if the length of the response buffer is large enough to accommodate the size of the response itself; if not, we re-allocate more space in the response buffer.

```
*buffer_length = attrib.st_size+strlen(*response_buffer)+2; // +2 for '\r\n' between header and content
*response_buffer = (char*) realloc(*response_buffer, (*buffer_length)*(sizeof(char)));
strcat(*response_buffer, "\r\n");
```

Finally, we interpret the data differently for images and text. If it's an image, we map the file into memory with mmap and then use memcpy in order to display the image. If it's text, then we take a different approach and use getline() in order to read it in line-by-line. After all this has been added to our response buffer, we then send out the response to the client.

II. Difficulties

In coding the server, we faced numerous difficulties but were able to work through all of them. One issue we had at first was figuring out how exactly we should parse through the response itself. We fixed that by determining how to correctly use the strtok() function and then looking for spaces and carriage return characters to get the text we needed. Additionally, we had trouble at first figuring out how to handle date and time until we figured out how to correctly utilize the time.h library; we also needed to interpret the integers we got back and had to troubleshoot to find out that we needed to convert the integers to strings in order to strcat() them to our response_buffer.

Figuring out a solution for the file name extension was difficult, but we were glad to find a novel solution by checking the first character to determine it; for example, 'j' for .jpeg, 'g' for .gif. Another issue was the fact that we could not read line by line to display images like we did for HTML text, so we had to figure out to map the data with mmap() and then use memcpy() for images. Finally, we had to fix the problem where our buffer would fill up in the case that the file was too big; to fix this, we get the size of the file and then check it against our buffer. If it is too big, then we use realloc() to allocate more memory for our buffer.

III. Manual on Running Source Code

The code we worked on was server.c. We also included a Makefile for easy compilation. In order to run our project, simply type 'make' which will then generate 'server'. To run

server, simply type ‘./server’ and then the port number you want to specify. For instance, ‘./server 8081’. Then simply request the server by going to a browser (i.e. Google Chrome or Mozilla Firefox’ and typing in the IP address with the port number (i.e. localhost:8081) along with a file (i.e. localhost:8081/test.html). The page will then display accordingly and you can check the response header in the terminal.

IV. Explaining Sample Outputs of Client-Server

For an invalid (404 file not found) filename:

```
HTTP/1.1 404 Not Found
Server: CS118 Project
Date: Sun, 25 Oct 2015 23:08:17 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 89
Connection: close

<HTML><HEAD><TITLE>404 Not Found</TITLE></HEAD><BODY><H1>404 Not
Found</H1></BODY></HTML>
```

We first have the 1990 HTTP 1.1 protocol displayed along with the status response message for a file not found: 404. We then have the server information, which we just simply named “CS118 Project”. We then display the current date and time information according to the system the project is run on. After, we have the content-type (i.e. “text/html” or “image/gif”) as well as the HTTP standard character set of UTF-8. We then display the content length (counted by characters) of the header itself, 89. We then define the connection as close in order to signify that the connection will be closed after completion of the response. Lastly, we output the HTML code that is displayed in the browser.

For an forbidden (403 permissions error) filename:

```
HTTP/1.1 403 Forbidden
Connection: close
Server: CS118 Project
Date: Mon, 26 Oct 2015 10:29:39 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 89

<HTML><HEAD><TITLE>403 Forbidden</TITLE></HEAD><BODY><H1>403 Forbidden</H1></BODY></HTML>
```

We first have the 1990 HTTP 1.1 protocol displayed along with the status response message for a forbidden file: 403. We then have the server information, which we just simply named “CS118 Project”. We then display the current date and time information according to the system the project is run on. After, we have the content-type (i.e.

“text/html” or “image/gif”) as well as the HTTP standard character set of UTF-8. We then display the content length (counted by characters) of the header itself, 89. We then define the connection as close in order to signify that the connection will be closed after completion of the response. Lastly, we output the HTML code that is displayed in the browser.

For a valid (200 OK) filename (text HTML):

```
HTTP/1.1 200 OK
Connection: close
Server: CS118 Project
Date: Mon, 26 Oct 2015 04:50:17 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 39
Last-Modified: Fri, 23 Oct 2015 02:02:53 GMT

<h1>Hello world</h1>

<h2>CS 118</h2>
```

We have the HTTP 1.1 protocol along with the 200 OK status code indicating that the request is fulfilled. Again, we have the connection as close after every request, the server information, the system time and date in the correct format, the content-type (text/html in this case), the UTF-8 character set, the length of the content (in number of characters), and the date that the file was last modified. Finally, we output the raw HTML code.

For a valid (200 OK) filename (image .jpeg and .gif):

```
HTTP/1.1 200 OK
Connection: close
Server: CS118 Project
Date: Mon, 26 Oct 2015 04:54:31 GMT
Content-Type: image/jpeg
Content-Length: 39792
Last-Modified: Sun, 25 Oct 2015 06:11:15 GMT

HTTP/1.1 200 OK
Connection: close
Server: CS118 Project
Date: Mon, 26 Oct 2015 04:48:25 GMT
Content-Type: image/gif
Content-Length: 11397
Last-Modified: Mon, 26 Oct 2015 04:43:48 GMT
```

As in the previous cases, we again have the same information with the exception being that the content-type being image/jpeg and image/gif, respectively.