# Computer Science 118
# Computer Network Fundamentals
# Project 1: Web Server Implementation using BSD Sockets

Robert Geil, *UCLA Computer Science Department,* UID: 104916969
Levi Weible, *UCLA Computer Science Department,* UID: 104934015

## Abstract

Utilizing the TCP sockets available within the C++ library, a simple server to send and receive HTTP/1.0 messages can be built. When the server is run, a TCP connection is opened, and once the TCP connection is opened, a browser can send an HTTP/1.0 request for a file within the server directory. Once this request is received and processed, the server outputs the request to the console, opens the requested file, outputs a response to the console, and sends the file to the browser to be displayed over the TCP connection. Then, the server waits for the next request.

## 1. Summary

### 1.1. Project Design

When the server is run, an address and port number are specified in the command line. When the server first starts running, a dictionary vector is made containing the case insensitive file names of each file in the current directory with the actual file name. The server then takes the specified arguments and opens a TCP connection with the specified address and port number. Once the TCP connection is opened, the server then goes into a loop listening on that port number for a browser to make an HTTP request. When a request is received, the HTTP request is sent to the HTTPRequest and printed to the console containing all of the information about the request. The module searches converts the requested file name to a case insensitive version and looks to see if the file exists in the dictionary that was made earlier.. If the file is found in the dictionary, the file is sent to the HTTPResponse module, and an HTTP response message is sent to the console containing all of the information about the opened file. If the file is not found, a “404 Not Found” message appears in the browser that is a 404 html file in the current directory, and a corresponding response message is sent to the console. After the response message is printed to the console, the file is sent over the TCP connection in pieces in a loop sending one byte at a time. Once the loop completes, the connection closes, and the file is closed as well. The server then returns to the main server module, and the listening loop continues, waiting for the next request from a browser.

### 1.2. Difficulties and Resolutions

One difficulty that was run into was obtaining and formatting the current time and the last modified time for the HTTP response header. When gmtime was called a second time to format the current date, the data from the last modified time of the file was overridden. Storing both values after calling gmtime() prevented this override from happening.

Another issue that we ran into was periodic segmentation faults when testing our server. By running the server through a debugger, it was found that these segmentation faults were caused by sending pointers to memory instead of locally storing the data. This was particularly a problem when formatting the HTTP request and response messages through a string stream. The string stream was not saved locally, but instead a pointer to the memory was sent. The faults were fixed by saving the string stream locally before accessing that data. This also fixed an issue that was found with random characters being printed to the console from the response message.

When creating string streams, another issue was faced. Storing the string streams locally created a large amount of data on the stack. This large amount of data on the stack essentially meant that the stack was running out of storage space even though it appeared that the string streams were valid when running, creating a bad_alloc error in the HTTPRequest module. The solution to this issue was moving the string streams to the heap by using new and delete to allocate memory specifically for the string streams. This removed the allocation errors and allowed the string streams to be printed without any errors.

## 2. Operation Instructions

To operate the server, a relatively simple series of steps needs to be carried out.

### 2.1. Make

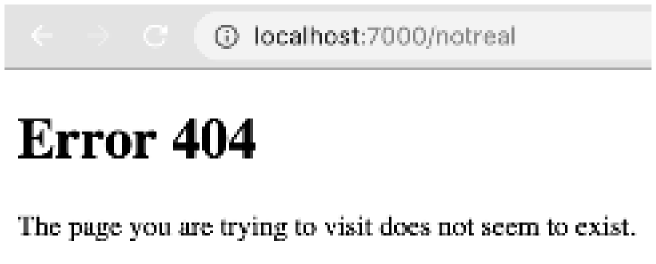
Within the directory for the server, use the `make' command to generate all required files for server operation. Other `make` targets include *clean* to remove all built files from the working directory and return it to a newly opened state, *check* to run a series of tests on integral portions of the program, and *dist* to build a tarball distribution of the server program.

### 2.2. Run

Once the server has been compiled, it can be run from the command line. The program requires exactly two parameters: a host address and a port number. The host address should typically be 127.0.0.1 for local hosting, and the port can be any available port, typically above 1024. If the port cannot be bound to, the server will send an error message and exit with non-zero status. Once the server has been started, with an example command such as `./server 127.0.0.1 7071` the port specified can be visited in the local browser (Chrome, Firefox, etc) with the address *localhost:7071*. Upon visiting this page, a landing index.html page will be served, and other pages can be navigated to utilizing the URL bar. The server supports sending .html, .htm, .txt, .cpp, .jpg, .jpeg, .png and .gif files, while all other file types will be delivered as application/octet binary data, to be downloaded by the browser. Visiting a page that doesn't exist will return a 404 error and corresponding error page. Once connected, multiple requests to the server for files can be accepted. The server and corresponding connection can be terminated utilizing CTRL-C from the command line.

## 3. Sample Outputs

### 3.1 404 Error



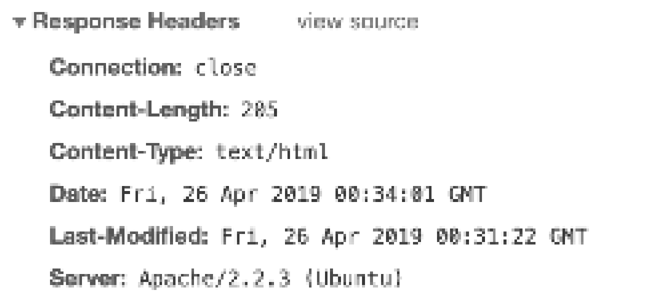
localhost:7000/notreal

Error 404

The page you are trying to visit does not seem to exist.

*Figure 1: 404 Error in Browser*

Figure 1 shows that when requesting the file "notreal", "404.html" file from the project directory is displayed in the browser.



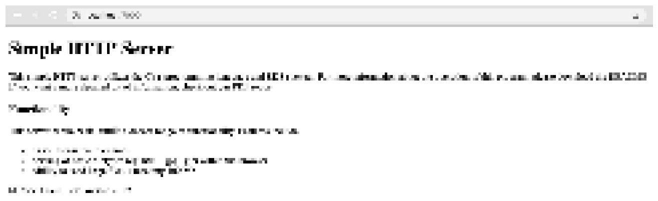
```
▾ Response Headers     view source
   Connection: close
   Content-Length: 285
   Content-Type: text/html
   Date: Fri, 26 Apr 2019 00:34:01 GMT
   Last-Modified: Fri, 26 Apr 2019 00:31:22 GMT
   Server: Apache/2.2.3 (Ubuntu)
```

*Figure 2: 404 Error HTTP Response Header*

Figure 2 shows the HTTP response header that is sent from the server to the browser to display the contents of "404.html". The server console shows the following output when the HTTP request was received:

```
GET /notreal HTTP/1.1
Host: localhost:7000
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 6.0; Nexus
5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/73.0.3683.103 Mobile Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=
0.9,image/webp,image/apng,*/*;q=0.8,application/si
gned-exchange;v=b3
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
```

### 3.2 Index Page

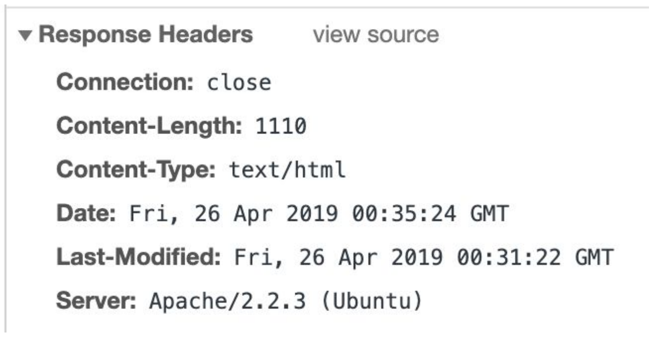


Simple HTTP Server

[illegible]

Figure 3: Index Page in Browser

Figure 3 shows the index page that is shown by default if the url path does not specify a file. From here, the report.pdf and README file can be downloaded. Information about the project and server are also included. This html file exists in the project directory as “index.html”.



```
▾ Response Headers      view source
    Connection: close
    Content-Length: 1110
    Content-Type: text/html
    Date: Fri, 26 Apr 2019 00:35:24 GMT
    Last-Modified: Fri, 26 Apr 2019 00:31:22 GMT
    Server: Apache/2.2.3 (Ubuntu)
```

Figure 4: Index Page HTTP Response Header

Figure 4 shows the HTTP response header that is sent from the server to the browser to display the index page. The server console displays the following HTTP request header:

```
GET / HTTP/1.1
Host: localhost:7000
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 6.0; Nexus
5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/73.0.3683.103 Mobile Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=
0.9,image/webp,image/apng,*/*;q=0.8,application/si
gned-exchange;v=b3
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
If-Modified-Since: Fri, 26 Apr 2019 00:31:22 GMT
```

### 3.3 Displaying an Image

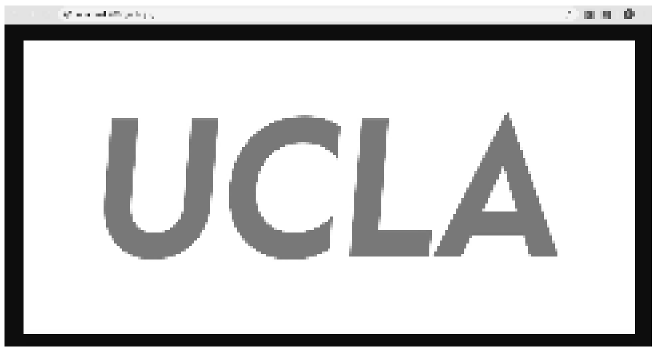
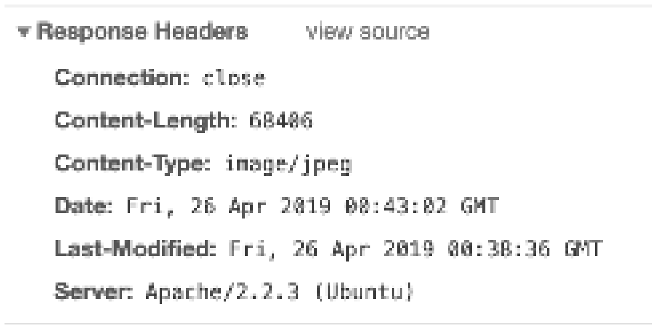


Figure 5: JPG Image in Browser

Figure 5 shows a JPG image, “ucla.jpg”, being shown in the browser. This image existed in the project directory when testing and was requested by specifying the path name of the image.



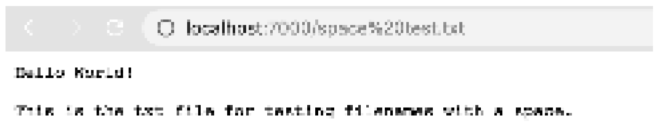
```
▾ Response Headers      view source
    Connection: close
    Content-Length: 68406
    Content-Type: image/jpeg
    Date: Fri, 26 Apr 2019 00:43:02 GMT
    Last-Modified: Fri, 26 Apr 2019 00:38:36 GMT
    Server: Apache/2.2.3 (Ubuntu)
```

Figure 6: JPG Image HTTP Response Header

Figure 6 shows the HTTp response header sent from the server when “ucla.jpg” was requested by the browser. The server displayed the request message in the console:

```
GET /ucla.jpg HTTP/1.1
Host: localhost:7000
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 6.0; Nexus
5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/73.0.3683.103 Mobile Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=
0.9,image/webp,image/apng,*/*;q=0.8,application/si
gned-exchange;v=b3
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
If-Modified-Since: Fri, 26 Apr 2019 00:38:36 GMT
```

## 3.4 Displaying a Filename with Space



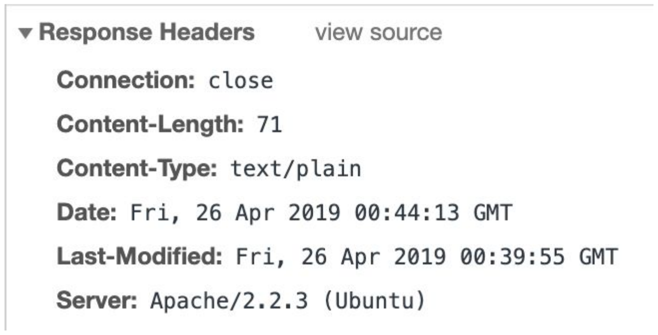
```
localhost:7000/space%20test.txt

Hello World!

This is the txt file for testing filenames with a space.
```

*Figure 7: Filename with Space in Browser*

Figure 7 shows a filename being requested that contains a space. The filename is “space test.txt” in the project directory. It can be seen that the file is displayed correctly even with the space in the filename.



```
▾ Response Headers    view source
    Connection: close
    Content-Length: 71
    Content-Type: text/plain
    Date: Fri, 26 Apr 2019 00:44:13 GMT
    Last-Modified: Fri, 26 Apr 2019 00:39:55 GMT
    Server: Apache/2.2.3 (Ubuntu)
```

*Figure 8: Filename with Space HTTP Response Header*

Figure 8 shows the HTTP response header sent back to the browser by the server for “space test.txt”. The server displayed the HTTP request in the console:

```
GET /space%20test.txt HTTP/1.1
Host: localhost:7000
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Mobile Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
If-Modified-Since: Fri, 26 Apr 2019 00:39:55 GMT
```