COM SCI 118 Spring 2019
Computer Network Fundamentals

Project 1: Web Server Implementation using BSD Sockets

Chengyao Zhang 405231343
Chengshun Zhang 905061060

1. High-level description

Our server has two functionalities, one is echoing the request back and printing it to the console; the other one is serving static files. If the user does not specify a path in the request URL, the server will just echo the request back. If the file the user requests does not exist, he will get a 404 Not Found error. If the file exists, the server will send the file to the user and displayed in the browser, supporting common extensions. The file names are case insensitive and tolerant of spaces in them.

Our codebase consists of 4 files: server.cc, request_parser.cc, request_parser.h and request.h.

The server.cc is the main entrance of our program. In main function, we first set up the server following the slides in the first discussion. We create a socket, set the server's address information, bind the socket with the address and call listen() function to have the server listen on incoming requests from clients. We make the port number as a parameter to be passed into the program by the user. In the while loop, the server receives one request and processes it and makes the response accordingly. Then it sends back the response to the client and waits to accept the next request.

The request.h file defines the request structure. It contains all the information of the request with the following fields:

- method: The method in request-line such as GET, POST, etc
- uri: The URL in request-line like /, /file.name, etc
- http_version_major: number in HTTP version before the '.'
- http_version_minor: number in HTTP version after the '.'
- headers: request headers with name, value
- body: request content

There are also two methods in request struct: clear() which make the request struct empty and rawString() which transforms the struct to string representation.

The request_parser.h and request_parser.cc together define and implement a request parser that parses a raw request and stores the parsed request in request struct. It works as a finite state machine that consumes the request one character by one character.

There are also other util functions and a mime_type mapping struct in server.cc.

- extension_to_type: map an extension of a file to a corresponding content type in request header
- get_file_names: get the file name in request URL
- to_lower: transform a char array to lowercase

- get_all_files: get the names of files in some directory and return a map that maps from the lowercase version of a file name to its original name
- url_decode: input the filename from the URL with special characters like space (%20) and output the decoded string (%20 -> '\ ')

The function server_static_file first constructs a map from lowercase filename to original name, decodes the filename from URL and opens the file if exists. Then it reads the file content to a buffer with certain size and then appends to the response string. If the file is large, the process may take several times. Then it adds Content-type header with proper value based on file extension. If no problems occur, we add the header line with status code 200; if the file does not exist, 404; if other errors happen, 400. At last, we return the response string.

2. Difficulties

1. When we issue a request in the browser, sometimes there will be multiple outputs to the console that show the request. The problem is that we do not empty the request struct at the end of each iteration. So we add a clear() method that clears all current information in request struct and call the method at the end of each iteration.
2. When we send multiple requests to the server, only the first one is handled correctly and sends expected response. By observing the output to the console, we find that the return value of request parser is bad for requests after the first one, while it should be good. The problem is similar that we do not reset the state of the request parser, as mentioned before, the request parser works as a state machine. We solved the problem by calling reset() method in request parser at the end of while loop. An alternative way to solve these two problems is to initialize the request struct and request parse at the beginning of while loop.
3. We test our server by visiting it with a browser but get an empty page. There is request printed out to console and we get 200 OK status code. But we do not receive the content of the response. It is because we forget to add one more CRLF, i.e. \r\n between the last response header and the response message body. After adding this, we are able to get the content as expected.

3. Manual

Go to the root directory of the project.
Run:

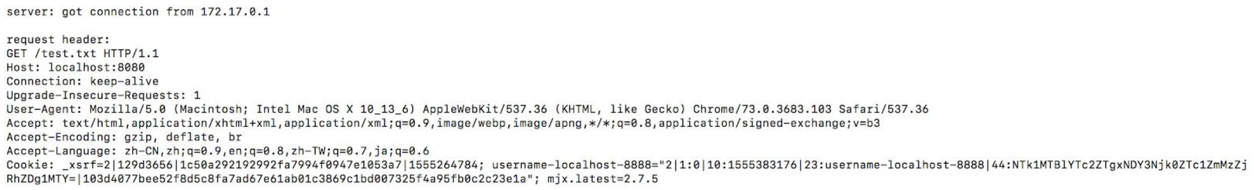
```
$ make
$ ./server 8080
```

4. Sample output

Meaning of fields in the request header:

The first line is ***Request line***. It begins with a method token (e.g. GET), followed by the Request-URI and the protocol version, and ending with CRLF.
Host: A legal Internet host domain name or IP address.
Connection: keep-alive: It indicates that the connection established is persistent.
User-Agent: It contains information about the user agent originating the request.
Accept: It indicate a list of media ranges which are acceptable as a response to the request. The asterisk "*" character is used to group media types into ranges, with "*/*" indicating all media types and "type/*" indicating all subtypes.
Accept-Encoding: It is similar to Accept, but restricts the content-coding values which are acceptable in the response.
Accept-Language: It is similar to Accept, but restricts the set of natural languages that are preferred as a response to the request.
Cookie: It shows the stored cookies included by the user agent.

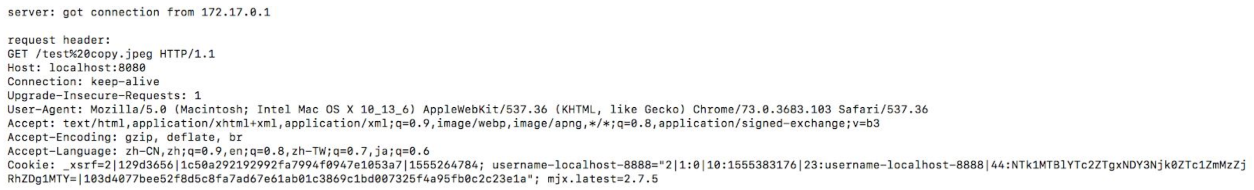
This is the console output of the web server when the browser requests a txt file:



```
server: got connection from 172.17.0.1

request header:
GET /test.txt HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 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: zh-CN,zh;q=0.9,en;q=0.8,zh-TW;q=0.7,ja;q=0.6
Cookie: _xsrf=2|129d3656|1c50a292192992fa7994f0947e1053a7|1555264784; username-localhost-8888="2|1:0|10:1555383176|23:username-localhost-8888|44:NTk1MTBlYTc2ZTgxNDY3Njk0ZTc1ZmMzZj
RhZDg1MTY=|103d4077bee52f8d5c8fa7ad67e61ab01c3869c1bd007325f4a95fb0c2c23e1a"; mjx.latest=2.7.5
```

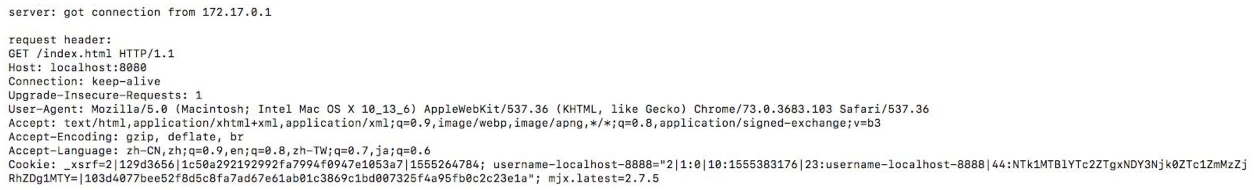
This is the console output of the web server when the browser requests a png file with space in the file name:



```
server: got connection from 172.17.0.1

request header:
GET /test%20copy.jpeg HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 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: zh-CN,zh;q=0.9,en;q=0.8,zh-TW;q=0.7,ja;q=0.6
Cookie: _xsrf=2|129d3656|1c50a292192992fa7994f0947e1053a7|1555264784; username-localhost-8888="2|1:0|10:1555383176|23:username-localhost-8888|44:NTk1MTBlYTc2ZTgxNDY3Njk0ZTc1ZmMzZj
RhZDg1MTY=|103d4077bee52f8d5c8fa7ad67e61ab01c3869c1bd007325f4a95fb0c2c23e1a"; mjx.latest=2.7.5
```

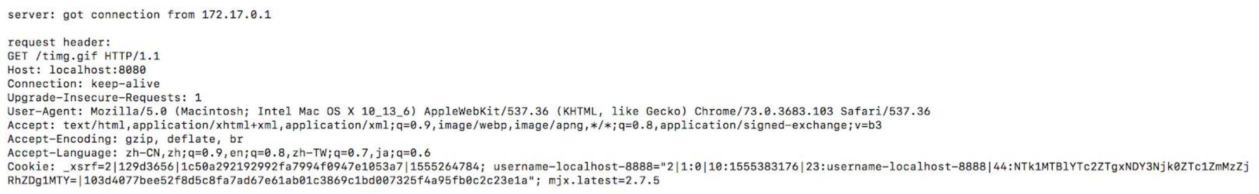
This is the console output of the web server when the browser requests an HTML file:



```
server: got connection from 172.17.0.1

request header:
GET /index.html HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 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: zh-CN,zh;q=0.9,en;q=0.8,zh-TW;q=0.7,ja;q=0.6
Cookie: _xsrf=2|129d3656|1c50a292192992fa7994f0947e1053a7|1555264784; username-localhost-8888="2|1:0|10:1555383176|23:username-localhost-8888|44:NTk1MTBlYTc2ZTgxNDY3Njk0ZTc1ZmMzZj
RhZDg1MTY=|103d4077bee52f8d5c8fa7ad67e61ab01c3869c1bd007325f4a95fb0c2c23e1a"; mjx.latest=2.7.5
```

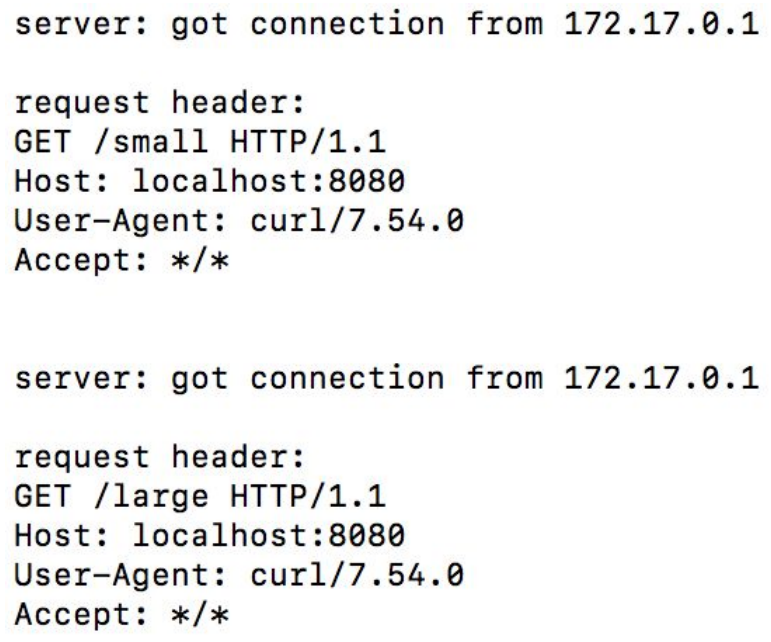
This is the console output of the web server when the browser requests a gif file:



```
server: got connection from 172.17.0.1

request header:
GET /timg.gif HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 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: zh-CN,zh;q=0.9,en;q=0.8,zh-TW;q=0.7,ja;q=0.6
Cookie: _xsrf=2|129d3656|1c50a292192992fa7994f0947e1053a7|1555264784; username-localhost-8888="2|1:0|10:1555383176|23:username-localhost-8888|44:NTk1MTBlYTc2ZTgxNDY3Njk0ZTc1ZmMzZj
RhZDg1MTY=|103d4077bee52f8d5c8fa7ad67e61ab01c3869c1bd007325f4a95fb0c2c23e1a"; mjx.latest=2.7.5
```

This is the console output of the web server when using curl to download binary files:



```
server: got connection from 172.17.0.1

request header:
GET /small HTTP/1.1
Host: localhost:8080
User-Agent: curl/7.54.0
Accept: */*


server: got connection from 172.17.0.1

request header:
GET /large HTTP/1.1
Host: localhost:8080
User-Agent: curl/7.54.0
Accept: */*
```