CS118 Computer Network Fundamentals

Project 1: Web Server Implementation using BSD Sockets

Name: Jeannie Chiem
ID: 504-666-652

For this project, we needed to implement a server that can receive HTTP requests and send back HTTP responses.

Server Outline

First, the server program receives a socket descriptor and then fills its socket address structure with the correct info about the host machine and port. Afterwards, it binds that info to the socket descriptor and then waits for a client to connect to it using listen(). Once a client is found, the program obtains the client's socket descriptor by using accept(). From here, the server can then receive the client's request message.

To get the name of the file requested, the server program will parse through the request message by looping through each character in the string. During this process, it will check for the characters "%20" and convert them into spaces and will also isolate the filetype into its own separate string. After parsing for the filename, the program will then parse the message for the HTTP version used to put into the return header file. After the parsing the request message, the program will then check the filetype to create the correct Content-Type header.

Once that is complete, the server will attempt to find and open the requested file. If the file is not found within the same directory as the server program, it will instead open the included 404.html file. Once a file is opened and its file descriptor is obtained, the program will then pass the file descriptor, client socket descriptor, and relevant header strings into the SendtoClient() function. This function will handle all the necessary steps to send an HTTP response to the client. It will first find the complete size of the file in bytes and then allocate space in memory to hold the file. It then reads the contents of the file into memory and then create the correct headers for it. Afterwards, it will send the header and file contents to the client through the send() function. If the server cannot send the whole file in one send, it will loop and send more parts of the file to the client until the whole file is sent. After this is completed, the program will return from SendtoClient() back to main() where it will then close the connection between the server and client and end the program.

The difficulties faced in this program were figuring out how to parse through the request message and how to send the file correctly.

First, we used the provided sample server to see what the request messages from a web browser looked like. Using that, we were able to figure out how to separate the filename from the request message.

Next was finding how to send data to the client. Using the provided test client, we were able to see what exactly was being sent to the client in text. We figured out that the file needed to be opened in binary in order to comply with the browser's HTTP response format. After that, we needed to find a way to send larger files that could not be sent with one send request. Since send() returns the amount of bytes sent, we used that to check if the whole file was sent and made it keep looping to send more of the file until the amount needed to send left was 0. After that the server was able to send the files to the browser correctly. A problem we encountered when testing for different file names was requesting files with spaces contained in them. We found that the browser would replace all spaces with the characters "%20" so we had to take that into account when parsing the request for the filename.

When testing our program, we made files with odd names in order to check the parser. We also made the program output what filename it got from the request message to see if it worked correctly. Here is one such output:

Here is the request: GET /hEyWoo%20%20%20fkd%20f.hTm HTTP/1.1
Host: localhost:3000
Connection: keep-alive
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36
Upgrade-Insecure-Requests: 1
Ac
Here is the requested file: hEyWoo fkd f.htm
Here is the HTTP version: HTTP/1.1
Here is the filetype and Content-Type: htm, text/html

When checking for the filetype, we made sure to use tolower() and toupper() so the program would check files in both cases.

USER MANUAL

To compile the program, simply use cd to go into the program directory and then use the make command as shown.

$ make

The makefile will compile server.c and create the server program in the same directory.
To use the server, in the same directory type a command in this format into the command line:
./server <portnumber>

For example one valid command is this:

$ ./server 3000

Once this is done the server will do nothing but wait until a client connects to it.
To connect to the server using a web browser(e.g. firefox, chrome), put a request in this format into the address bar:
http://<machinename>:<port>/<filename>

Where <machinename> is the name or IP address of the server machine, <port> is the port number of the server, and <filename> is the name of the file requested.

For example, when testing the server on the same machine, you can use:

http://localhost:3000/hello.html

If the filename requested exists on the same directory as the server and is one of the filetypes listed here (.jpeg, .jpg, .gif, .html, .htm), the browser should show the file correctly on the screen. If the file does not exist on the server directory, the browser should instead show a 404 error page.

Once you are finished using the program, you can use the make clean command to remove all extraneous files created during compilation and return the program to its original state as shown.

$ make clean