COM SCI 118: COMPUTER NETWORK FUNDAMENTALS

Project 1: Web Server Implementation using BSD Sockets

Yash Choudhary, 704630134
Sparsh Arora, 804653078

Introduction

In this project our purpose was to get a deep understanding of how BSD sockets and HTTP protocols work together to implement a server-client model over the internet. The overall design was based on a browser(client) sending HTTP requests to the server, which then processes them and gets the data at the specified directory and then finally sends a HTTP response, that includes the requested data, back to the client. The client can now parse the response message to get essential details and data messages which will be displayed on the browser (client).

1. High Level server design description

The web server is implemented by prematurely giving it a port number (random number) and giving it a default hostname ("localhost"). The server then sets up TCP sockets and uses bind(), listen() and accept() to process and accept connection requests from clients. The files requested by the client are limited to being only in the directory of the server itself.

We use a while loop with forking to provide for functionality of the socket while continuing to accept other requests. In this way, our user can request for multiple files at once without having to run it again and again.

The server then processes the HTTP request it receives, parses it and tries to get the data requested by the client. The server reads the requested data as a stream of bytes regardless of the file extension type.

This method still supports multiple requests from the same client (opening multiple tabs in Chrome, Firefox) because of child process implementations. However, our program/server does not terminate until forcefully done by the user using Control+c (SIGINT). This is expected in a server as it is supposed to be "on" at all times to process client requests and only on user/administrator demand will it shut down.

2. Difficulties faced

One of the major difficulties we faced was in the forking process. Initially, we were unable to send multiple requests but forking and creating a child process fixed that issue. We had to go through API for forking.

Another one we faced was to take care of spaces in the filename. It took us time to realize how space was represented in an HTTP request (%20). However, once we realized that spaces were encoded in this way, we easily implemented code that replaces the substring %20 in a string with a space.

Finally, we faced some minor difficulties with implementing and understanding socket API. However, by looking at their syntaxes and descriptions on man pages, we were easily able to overcome them.

3. Compiling and Testing

To compile, simply use the command:

tar -xzvf 804653078.tar.gz

Then, to compile the code use,

make

And then to run the code:

./server

OUR PORT NUMBER THAT WE USED: 2010

IMPORTANT NOTE: Please keep in mind that our code can handle multiple HTTP requests (5 at one time). So, to end the code, the user has to use the CTR-C (SIGINT) option. Otherwise, the connection stays open, ready to accept requests.

Also, sometimes the TCP connection might be busy and show a binding error. Simply refresh your web browser and run the code again.

4. Sample Outputs

Given below are some outputs for sample cases.

The browser is the client requesting to access certain files (.html, .jpg or .gif) and the server sends the requested data so that it is displayed on the browser.

In this screenshot, user simply requests for a simple .jpg file that is present in our server directory. The code correctly fetches the file and presents it to the web browser as seen.

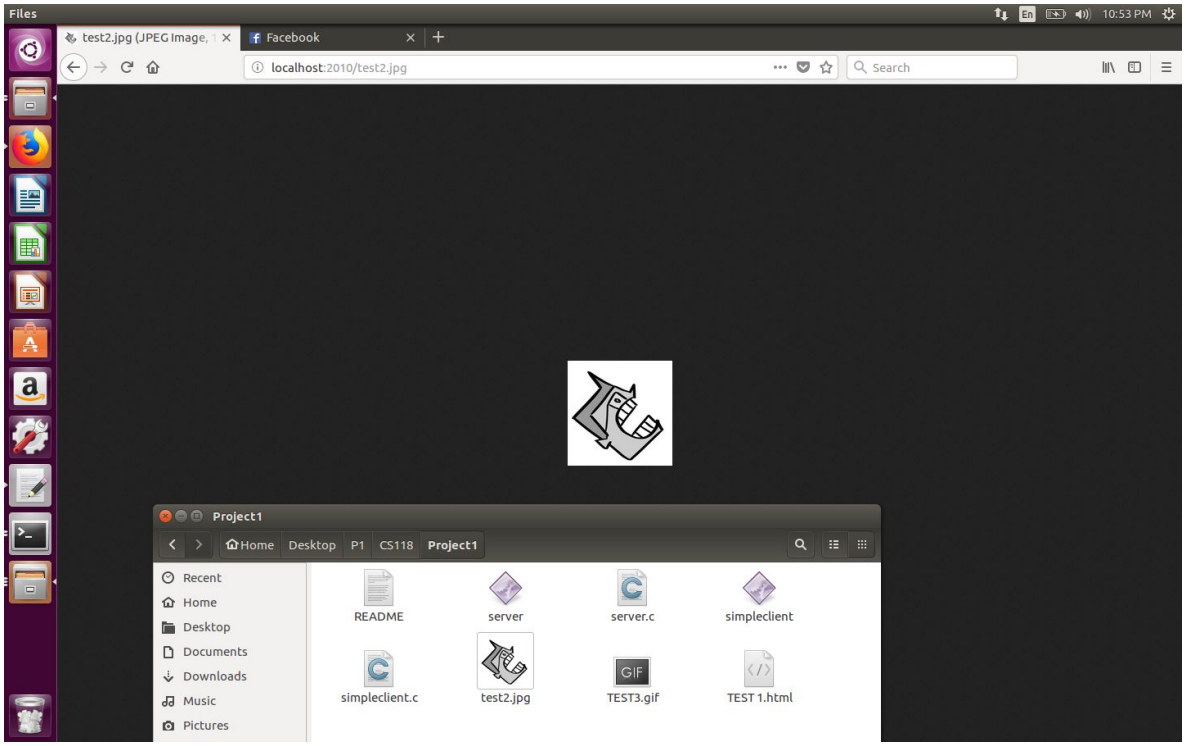

Figure 2: Output for request to access a .jpg file (image). Shown are the directory of the server on the bottom and the requested web page with the image on the top (Mozilla Firefox).

In this screenshot, the client requests an HTML page with a space ‘TEST 1.html’, as we can see in our server directory, such a file exists. Hence, our code correctly handles spaces in filenames and gives the right file

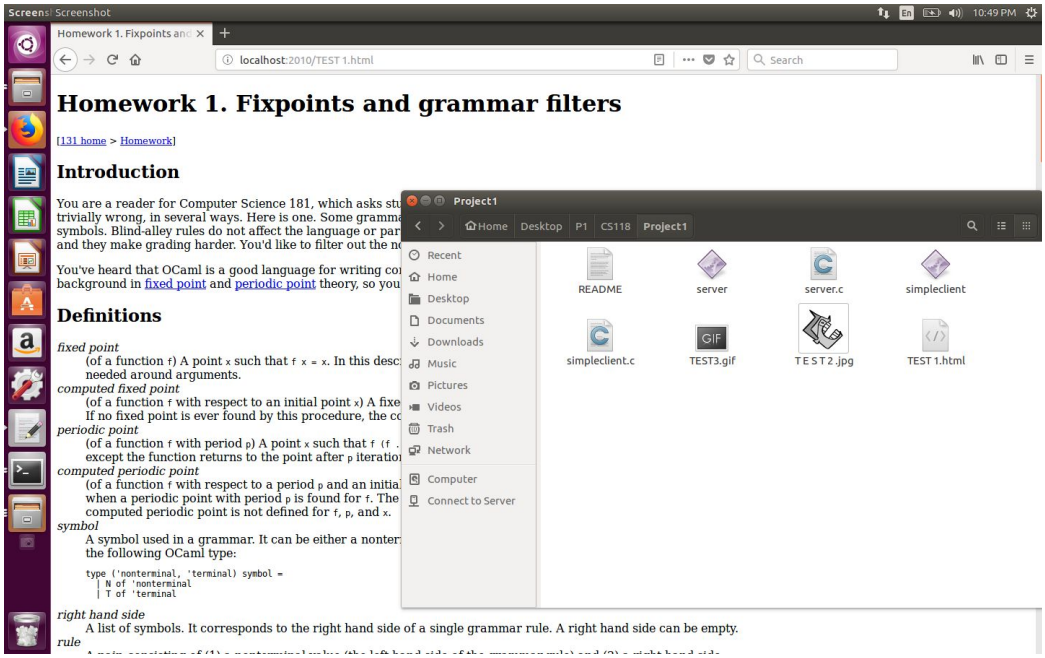

Figure 1: Output for request to access a html webpage with a space in its name. Shown are the directory of the server on the right and the requested web page on the left. (Mozilla Firefox).

```
parsharora@arora:~/Desktop/P1/CS118/Project1$ ./server

GET /test2.jpg HTTP/1.1
Host: localhost:2010
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:58.0) Gecko/20100101 Firefox/58.0
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
If-Modified-Since: Sat Feb  3 02:24:52 2018
Cache-Control: max-age=0
```

Figure 3: HTTP Request dump on the console. Shown above is the request message sent by the client (browser) to the server.