CS118
Project 1: Web Server Implementation using BSD sockets

Catherine Lin 104440877
Samir Tabriz

Description:

The purpose of this project is to introduce the basics of socket programming and the HTTP protocol. We were able to implement this using a server and client. The client in this case is a browser that uses URLs to create and send HTTP requests to the server. The server processes received HTTP requests, attempts to retrieve the requested data specified by the URL, and subsequently sends a response back to the client. The client then parses the response to determine the header properties and write the requested data to a file.

We used the Linux OS (Ubuntu) to develop the server.

1. Build Instructions

In order to build the executable files we included a Makefile to make compiling the webserver quick and easy. First, navigate to the directory that contains the source files. Then run the "make" command. This will generate the server executable that can be used to test the program.

2. Server Design

The design of our web server is based off of the skeleton code from Bee's Guide to Programming. The web server takes three arguments: a hostname, a port number, and a file-directory. The server first resolves the IP address from the hostname and port number using DNS. It then uses the *bind*, *listen* and *accept* functions to set up a socket to listen for any connection requests. When the connection is established, it receives the HTTP request, and processes the request to determine the file name and path of the requested object, starting at the directory specified by the given file directory. Another TCP socket is created to facilitate data transmission. The purpose of the serveFile function is to ensure that a valid file is requested, and it also handles the case when no file is requested. If no file is requested, a "File Not Found" page is displayed in the browser. Otherwise, it uses fopen and fread to convert the requested file's data into a c string buffer. An HTTP response message is transmitted to the client via the send method and then subsequently the data stored in the buffer is also transmitted via send to the client.

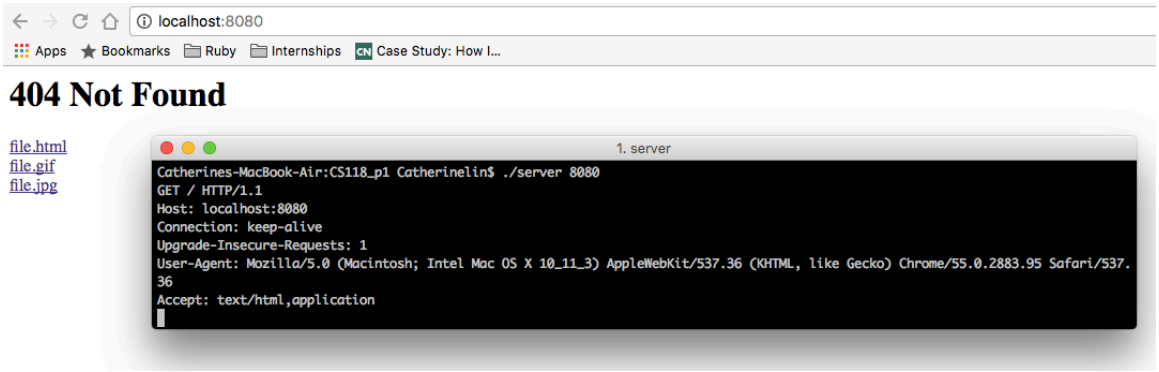
Our implementation of the client accepts multiple HTTP requests until forcefully closed.

3. Difficulties faced

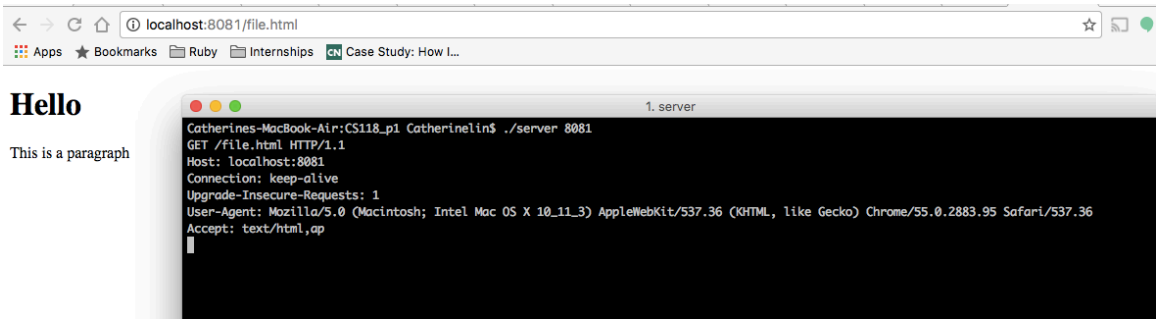
We had trouble with implementing a server that would server multiple HTTP requests from the server. We were able to fix this problem by using a while loop that accepts new connections each time an old connection is closed.

We also had segmentation fault issues. Small errors in generating the HTTP headers would cause an error in the HTTP response message as well as corruption in the file data.

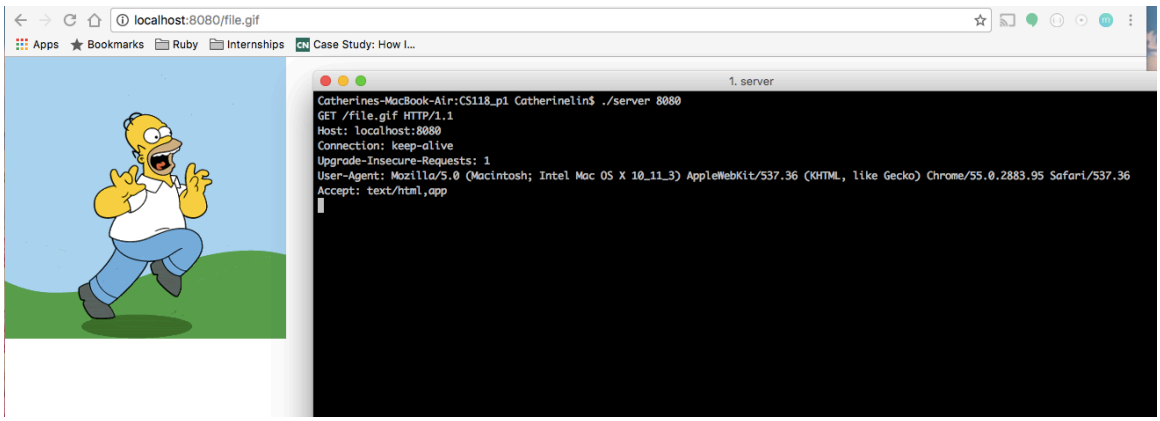
Sample Outputs of client server:



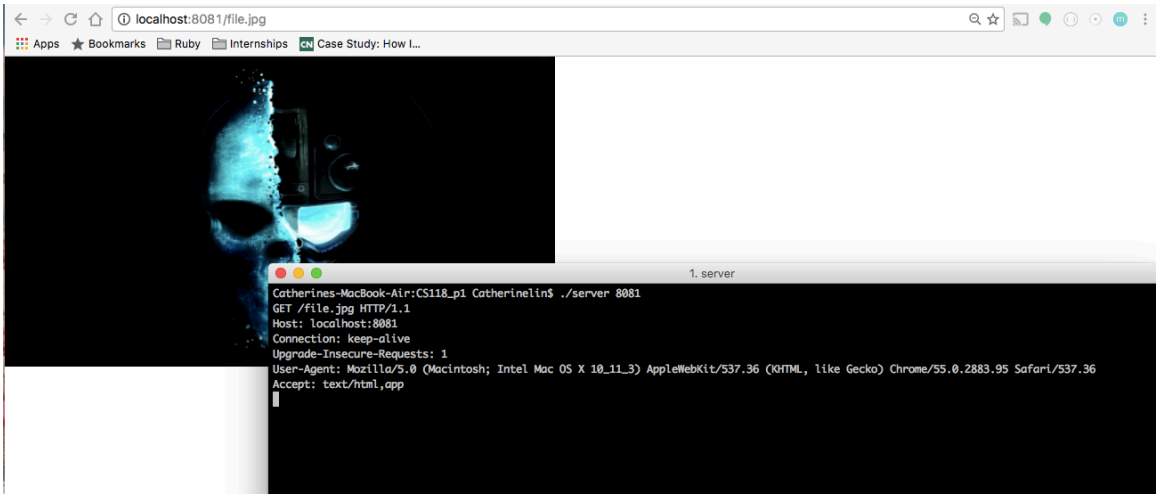
Above is an image of the output when no file specified.



Above is the output of when an html file is requested.



Above is the output of when a gif file is requested



Above is an image of the output when a jpeg file is requested.