**CS 118 Network Fundamentals**

**Webserver Part 1**

Chih Chin Chang

304541390

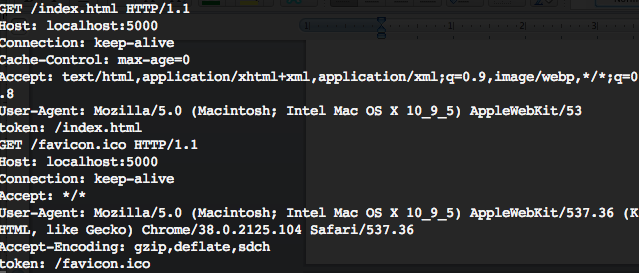
Joseph Cox

703956821

**Problem Overview:**

The project is split into two separate parts, where the first part analyzes the HTTP request sent to a web server by a client browser and the second part modifies the web server to return the requested resources. A convenient caveat provided by the problem is that the web server, for now, only needs to server html files.

**Part A:**



For the first part, we are interested in analyzing the elements of the request message sent by the client browser to the server. For our simple server, we print out the request message to the console and the example given by the console screenshot.

The first line states that the request asks the server to GET a specific file at path "/index.html" using the prototype "HTTP/1.1". The file path is relative to the directory of the server listener. But the first backslash can be ignored when traversing the path because it simply is a part of the URL delimiting the server location and its file path.

The second line describes the server host location. In this case, the client is requesting the localhost at port 5000.

The third line asks the server to keep the connection between the server and the client alive so that the RTT overhead can be reduced since the handshakes are not needed for subsequent requests.

The fourth line gives option for the cache, setting the max age of the cache to 0.

The fifth line gives acceptable content types to be returned by the server. It's an expression of the content types of message the client expect to be returned by the server.

The sixth line describes the information of the client. The client is using the Mozilla browser version 5.0. It is running on an Intel Mac OSX 10.9.5.

The token is not a part of the request message. It is the file path that the web server will use to find the file to return to the client in part B.

**Part B:**

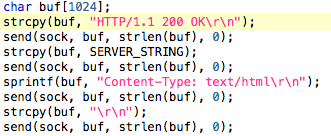
Server Overview:

To start, the server must accept and open a socket to the client. This simple server will create a new process for each client.

Initially, when the client sends a request, the server first prints the request message to the console. The server assumes that the request will be a GET and will extract the file path part of the header.

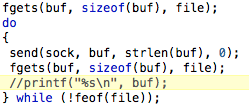
If no path is given, no file is returned. Standard and more robust web servers will automatically seek the index files; however, handling multiple possible extensions for the index file is not supported because our web server only need to serve html files.

The server then opens the file at the path, and if the file is there, the server will perform two operations. First, the server will send the header to the client. The header is provided by the following screenshot.



In essence, this returns a 200 "ok" message using the HTTP protocol. It suggests that the content type in the message is to be interpreted as text/html.

Next, the server will open the file, check the file is available and opened, and then send the file to the client, as illustrated by the following code.



Now the message is complete and the file can be closed.

Difficulties

There are two main difficulties when making this web server, and they are apparent given the overall design of the server. First, we must extract the file path from the message received by the server. To do this, we tokenize the message string by whitespaces, and choose the second token, which contains the file path. However, since the path string in the message will seek the file in the subdirectory of the current directory by default, we ignore the first "/" so that the file can be located in the directory of the server file.

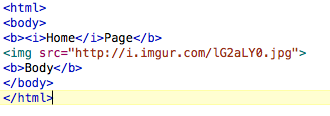
The second main difficulty is in thinking of all the exception cases that may return behaviors that is not expected. These include empty file paths, different request method other than GET, wrong file extensions, and files that do not exist.

Our simple solution is to find account for these exceptions, but also ignore them. If files are not found for any reason, simply return nothing. If the client tries to use other methods than GET, just treat it as if it is using GET

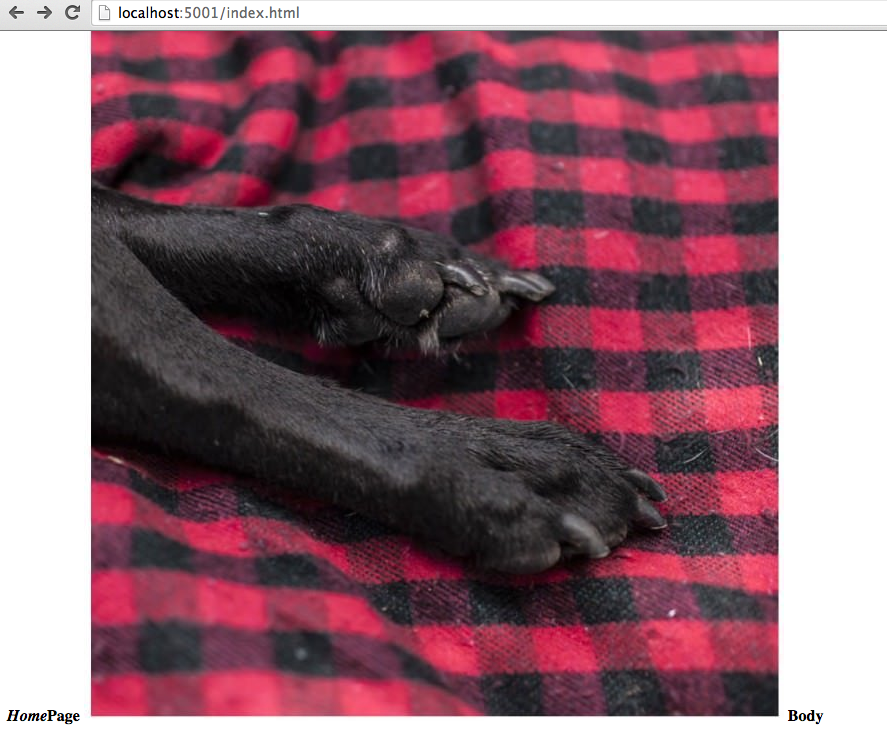
Because our server only need to handle html files, the server can simply ignore requests that does not conform to that.

Samples

We've created a simple html file with an image file, font, and some simple html styling.

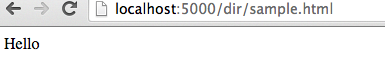


Next, we make the request on the browser. And the browser successfully returns the page.



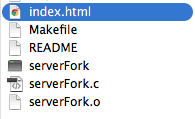
*Example 2:*

The file path can also be within a subdirectory. Also the port number can be changed.



How to Run

Here's a sample of the directory with a test index.html page.



The index files should be in the same directory as the server file.

To start the server, call "make".

Then call "./serverFork *portnumber*" where *portnumber* can be any user specified open port.

Now you can visit the server by visiting a web browser and use the url "*hostname:portnumber/filepath"* where hostname is the host of your webserver, which can be localhost if your hosting the server at the same client as the client. The portnumber is the same port number as before. And the filepath is the path to the file.