Project 1:

Web Server Implementation using BSD sockets

CS118

Juan Hernandez

UID: 904476103

Compiling and Running the Server

In order to get the server running, untar the tarball and use make to compile the server into an executable file named “server.” This directory should be the directory containing the files that you want to give access over the specified port. In order to start the server use “$ ./server portno” where “portno” is the port number that the server will be listening for incoming HTTP requests. This server will continue responding to HTTP requests until it receives a SIGINT signal.

High Level Description

The server program will set up a socket using AF_INET protocol and sets a boundary of at most 5 concurrent requests. The server will accept a connection and read an entire HTTP GET request. If the HTTP protocol is not followed, we will not service the request. If it is a valid GET request, we will output the GET request message on stdout, extract the filename from the first header, call a function to sanitize the filename, then attempt to open the requested file if it exists in the current directory, and if it exists, it will read the binary data of the file into a buffer. The server then uses the gathered data to format a valid HTTP response to a GET request using the appropriate HTTP headers. Depending on the file extension(if any) the server will format the respond message to provide the right “Content-Type:” header and will default to “text/html” in the case that an extension is not recognized or if there is no extension. The buffer with the binary data is then appended to the formatted response message and written to the socket. By default, a connection is closed after a single GET request is satisfied. The server then finishes the request and waits for another connection to service.

<u>Project Difficulties</u>

1) Sanitize the Filename Requested

The file name specified will be received in a GET request message through HTTP protocol. This means that all spaces are converted to “%20” in a request message. The server must convert this 3 byte value into a one byte value ‘ ‘ with the space character. To do this, this implementation stores a temporary c string containing the full name of the file name. First we iterate through each character and convert any alphabetic characters to the lower case equivalent. Then we iterate through again and if we find a matching 3 byte sequence of “%20” we replace it with a space character and shift the characters leftwards accordingly.

2) Read Binary Data Into Buffer(vs. Into a C string)

Since we are reading file data in binary, we cannot use the c string library to concatenate and manipulate the data. This is because valid file data may contain the null byte. Instead we use fseek to determine the length of the file in bytes and read that many bytes into a buffer. Both the buffer data and the length of the file size must be passed to helper functions because there is no other way of determining the length of the data.

3) Determining the File Extension

We find the file extension by iterating through a temporary file name string searching for a “.” character. If it is not found, the default extension will be “text/html” and if it is found, we continue iterating until the end of the string. We compare this new substring with common extensions and format the “Content-Type” header in the response message accordingly.

4) Convert int to string

In order to write the “Content-length” header we must know the number of bytes that our file takes up. We cannot use the string library to do so so we use fseek to determine this value. This value is a long int that must be converted into a c string. In order to do so, we can write a function that adds the numerical value of each decimal place digit to the character representing the “0” character. This will return the ascii/etc value for the corresponding digit. We do this by dividing and using the mod of the number until we reach 0.

5) Get 404 Response

If the file is not found by the system call “fopen” then we will format a new respond message that has the appropriate header “HTTP/1.1 404 Not Found” as well as a stream of characters that represent an html file that displays a 404 web page on the responding client’s browser. This stream of data does not need to exist in a separate file and is instead hard coded into the server program.