|  |
| --- |
|  |
| CS 118 Project 1: Concurrent Web Server using BSD Sockets |
| |  | | --- | | **Brian Soumakian** | | **UID: 003554865** |   **SEASnet login: soumakia**   |  | | --- | | **Kevin Greene** | | **UID: 903670945** |   **SEASnet login: keving** |
|  |

# Overview

This report is designed to give a high-level description of the web server design. It will also detail any obstacles encountered during the project and how they were resolved. Information regarding how to compile the source code and run the web server will also be provided. Finally, sample output from the web server, along with a brief description of the output, will be provided.

# Server design

After the web server has been setup, it waits for client to connect to the server. The server is setup to listen on port 3400. Once a client has connected, a new process is forked to parse and respond to the HTTP request sent by the client. The parsing process starts by extracting the HTTP method and requested URI from the first line of the request. Each of the subsequent lines of the request is parsed for the HTTP field name and field data.

The method of the request is used to determine the course of action the server should take. The simple web server is designed to only respond to HTTP GET requests. For GET requests, the requested URI is used to look for the requested file. This was done by appending the requested URI to the directory the web server is running from.

If the requested file is not found, the HTTP status code is set to 404 and a failure message is added to the HTTP response body. If the file is found, the file contents are copied into the HTTP response body and the “Content-Length” field is set to the file length. The HTTP status code is then set to 200. The web server is capable of displaying the following files to the client:

* html
* gif
* jpg
* png
* tiff
* txt

Finally, the complete HTTP response is sent back to the client.

# Obstacles encountered during development

After implementing support for basic html files, we encountered an issue attempting to display MIME files. Our method of sending the byte-stream to the host from the file was causing encoding issues, likely due to erroneous bits being introduced into the buffer. The problem was solved by using the std::stringstream::write() method instead of the overloaded << operator. In addition, we only write how many bytes were just read from the previous std::stringstream::read() call, denoted by std::stringstream::gcount().

# Building/Running the webserver

### Compiling webserver source code

Open a terminal and change to the project1 directory (same directory where this report lives). Run the command “make”. The source code should compile and produce the web server executable file named “webserver”.

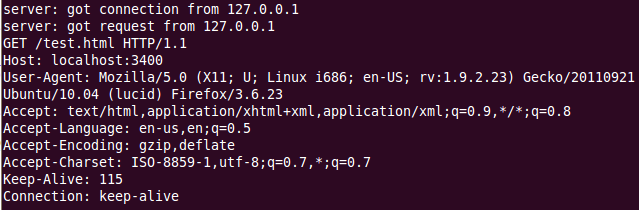
### Running the webserver

After compiling the web server source code, launch the outputted executable named “webserver”.

# Sample Output

### GET request for HTML file

Shown below is the server output for the URL “localhost:3400/test.html”. The server prints a message when a client has connected to the server and when the server has received a request from the client. Both messages contain the address of the connected client. Following the two messages, the server prints the exact request the server received from the client. The response is formulated based on the request and sent directly to the client.



### GET request for JPG file

Shown below is the request sent by the client for the URL “localhost:3400/test.jpg”.

