Documentation

# Server Overview

The purpose of this project was to demonstrate an understanding of HTTP/1.1 web servers (defined in RFC 2616). When hosted, it allows clients to connect via standard web browsers and request files in the server’s root directory. It was tested with jpg, txt, html, and pdf files, although the raw nature of the ByteBuffer used to transfer the data should support most file types. A typical selector is used, registering channels that connect to the server socket channel. Connections will be maintained unless the header “connection: close” is found in which the client channel will close after that transmission. The server always sends the status code, status message, date header, and when actual resources are sent, the last-modified, content-type, and content-length headers. It also supports if-modified-since requests, returning a 304 response if the resource has not been modified after the requested time.

This program requires the server to be started first. There are three flags it accepts:

## -p: port

Allows a user to specify a port for the server to run on. If not specified, will default to port 8080 as it does not require root privileges to run on.

## -docroot: root directory

Allows a user to specify a root directory. The server may only access files that are directly in this directory, and all parent directories are protected.

## -logfile: log

Allows a user to specify a log file. All output from the program (general status messages of what’s happening, and headers of all requests received and responses sent) will be written to both standard out and this log file. If not specified, output will only stream to standard out.

# Functions

The print function is used to print all output from the function. If the user has established a log file, it will print output to standard out, and that file via a print stream. Otherwise, the output is simply written to standard out. To handle this, a shutdown hook is added early in the program which closes the file output stream, print stream, and also the server socket channel.

The getString functions build and return the fairly static (aside from date which is taken at the time of creation) 501, 404, and 304 responses. This includes the HTML required to display the error pages on the web browser.

# Major Challenges

## HTTP Specification

Files weren’t being picked up by Wireshark as HTTP packets. Upon closer investigation of RFC 2616, I realized I was just using new line characters between each header, when I actually needed a carriage return each time as well.

## Bytes to String

There were some issues getting the data bytes on everything but .txt files, until I realized I was appending the requested file bytes to a String Builder object. After moving the data to the ByteBuffer directly after the header string, transmissions worked for every file type I tested.