Networking Report

180009917

1 Introduction

This report is to serve as review and guide for the extensions written for the CS5001-P3-Networking practical. For the logging extension (see Section 3), the Images extension (see Section 4) and the multi-threading (see Section 5). For comments on the basic requirements of this practical please see Section 2.

The general experiences of developing the Java Web Server application can be found in the Conclusion section (see Section 6).

2 Basic Requirements

Creating a server that would take two command line arguments, one the root directory and the other the port, was a trivial matter. However, providing the correct responses to the HEAD and GET requests was far more challenging.

After research into the HEAD and GET requests within the HTTP/1.1 protocol [Berners-Lee, T., Fielding, R., Frystyk, H., et al., 1999], the author of this paper was able to work out how to respond to such requests in a manner that browsers would understand.

After the GET and HEAD requests were passing through the *stacscheck* automated testing tool, the 404 (file not found) and 501 (unknown request) error handling responses were implemented to the server. A full stacscheck report for this practical can be seen in Figure 1.

Extensive use was made of Configuration.java and argument checking so all the server actually needs to run is the root directory of the files to be served.

```
pc5-013-l:~/Documents/OOP/Coursework/CS5001-p3-networking cai3$ stacscheck /cs/studres/CS5001/Practicals/p3-networking/Tests
Testing CS5001 Practical 3 (Web Server)
- Looking for submission in a directory called 'src': found in current directory
* BUILD TEST - basic/build : pass
* COMPARISON TEST - basic/Test01_no_arguments/progRun-expected.out : pass
* COMPARISON TEST - basic/Test02_GET_Correct_Header_Status_Line_for_Existing_Text_File/progRun-expected.out : pass
* COMPARISON TEST - basic/Test03_GET_Correct_Header_Content-Type_Line_for_Existing_Text_File/progRun-expected.out : pass
* COMPARISON TEST - basic/Test04_GET_Correct_Header_Content-Length_Line_for_Existing_Text_File/progRun-expected.out : pass
* COMPARISON TEST - basic/Test05_GET_Correct_Header_Status_Line_for_Non-existing_Text_File/progRun-expected.out : pass
* COMPARISON TEST - basic/Test06_GET_Correct_Content_for_Existing_Text_File/progRun-expected.out : pass
* COMPARISON TEST - basic/Test07_GET_Correct_Content2_for_Existing_Text_File/progRun-expected.out : pass
* COMPARISON TEST - basic/Test08_GET_Correct_Content_from_SubDirectory/progRun-expected.out : pass
* COMPARISON TEST - basic/Test09_HEAD_Correct_Header_Status_Line_for_Existing_Text_File/progRun-expected.out : pass
* COMPARISON TEST - basic/Test10_HEAD_Correct_Header_Content-Type_Line_for_Existing_Text_File/progRun-expected.out : pass
* COMPARISON TEST - basic/Test11_HEAD_Correct_Header_Content-Length_Line_for_Existing_Text_File/progRun-expected.out : pass
* COMPARISON TEST - basic/Test12_HEAD_Correct_Header_Status_Line_for_Non-existing_Text_File/progRun-expected.out : pass
* COMPARISON TEST - basic/Test13_HEAD_Request_Should_Not_Return_Response_Body/progRun-expected.out : pass
* COMPARISON TEST - basic/Test14_Correct_Header_Status_Line_for_Unsupported_Method/progRun-expected.out : pass
* INFO - basic/TestQ_CheckStyle/infoCheckStyle : pass
--- submission output ---
Starting audit...
Audit done.
---

16 out of 16 tests passed
```

Figure 1: Full *stacscheck* report for the basic requirements.

3 Logging

The first extensions attempted for this assignment to log the client requests and responses that were passed in and out of the server. To do this an abstract class called **logging** was implemented, which allowed logging to happen during the flow of execution for a regular request. The implementation was done in such a way so that the delay to the client from the additional

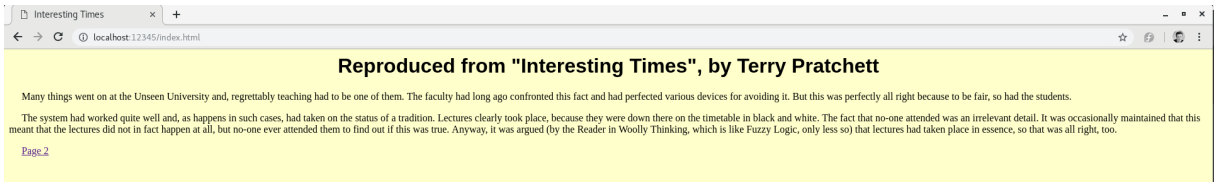

Figure 2: Index.html rendered via a GET request to the server.

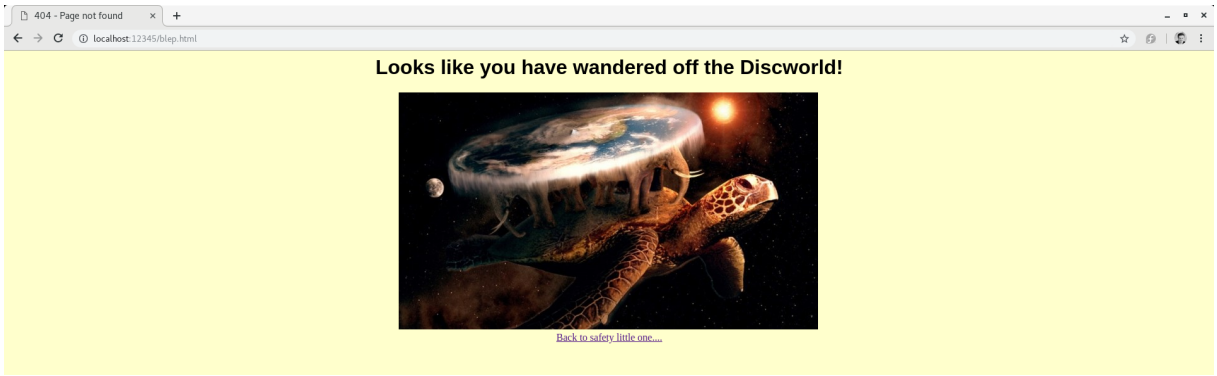

Figure 3: Example of the 404 page rendering on the browser. To text the 404 simply put any nonsense html file after localhost:12345/ into a browser when running the server.

actions by the server was kept to a minimum, whilst allowing the server to log the time delay between receiving the request and responding to it, see Figure 4 for an example of the log file after receiving a GET request from the browser.

To test the logging simply make any number of requests and then check the log.txt file within the root directory of this project.

4 Images

In order to have images processing on the browser, the requested file had to be translated into an Array of bytes and then sent over the socket to the browser, where the bytes would be decoded back into Strings and processed accordingly. In Figure 5 we can see an example of the server rendering an image into the browser, and in Figure 3, we can see an example of an image rendering inside an HTML page.

5 Multi-threading

The final extension that was attempted was to implement multi-threading into the server so that multiple requests can be ran simultaneously. This was the hardest extension to implement and it caused a large refactoring of the server (after a brief discussion with Martin Schimmer and Ricardo Araujo), in order to close the output streams at the correct moment. The issue was that after the files were served to the window the entire socket was closing, an error that was undetected because the server was only handling one request at a time.

Post-refactoring the server was able to multi-thread seamlessly, see Figure 6 for evidence of this.

To test the multi-threading, please open the Firefox browser application, open localhost:12345 /index.html 6 times, start the server on port 12345 and then right click on the header of the Firefox window to then select “Refresh All Tabs” to run 6 requests simultaneously.

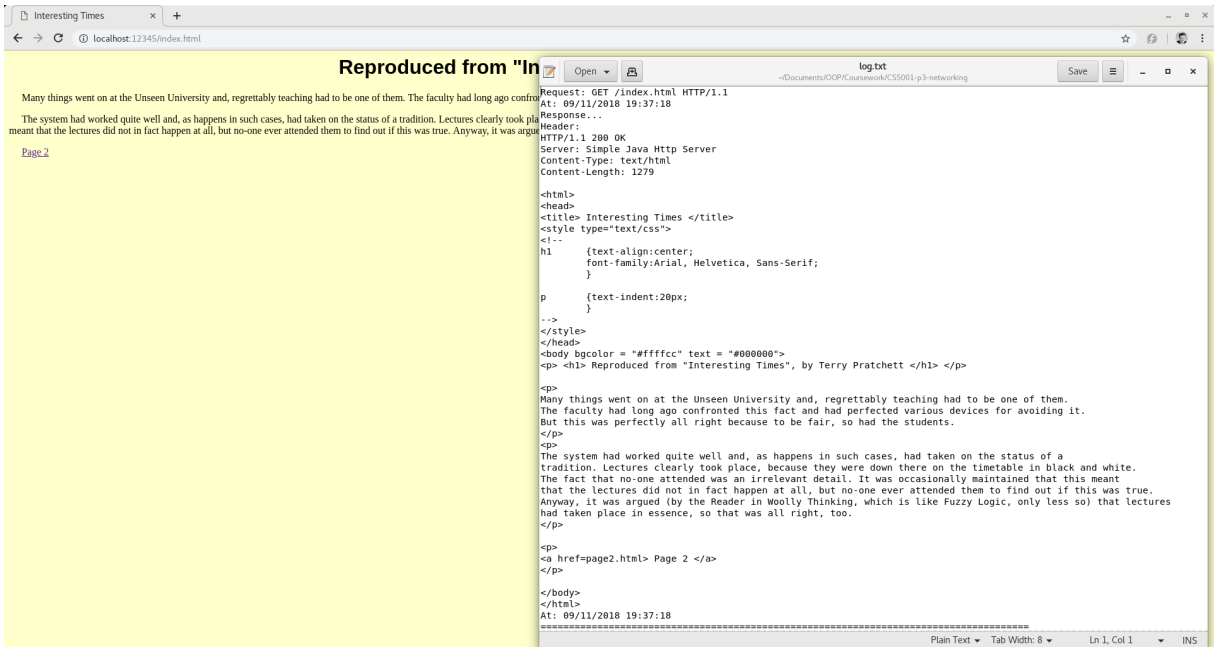

Figure 4: Example of the log file after a GET request has been made to the server from the browser.

6 Conclusion

In conclusion this was a pleasantly challenging assignment for the author of this paper, providing insight and teaching into the HTTP/1.1 response codes and the inner workings of a server, as well as providing invaluable practice into multi-threading and concurrency. The project was particularly challenging initially when first starting to tackle the GET requests appropriately, then it was very easy until the Threading extensions was reached.

Version control proved very valuable allowing the author of this paper to constantly switch between different feature branches so that development would not be stunted when a particular bug appeared within the code. The author of this paper believes that if a bug is not solved in an efficient manner it is best to revisit the error after a brief respite involving other programming based tasks, *clear the head*[1].

References

Berners-Lee, T., Fielding, R., Frystyk, H., et al. Hypertext Transfer Protocol – HTTP/1.1. https://www.w3.org/Protocols/rfc2616/rfc2616.html, 1999.

[1]Evidence of this would be provided, however that would mean supplying the GitHub commit list, therefore providing personal identifying information, which is against the submission procedure for St Andrews

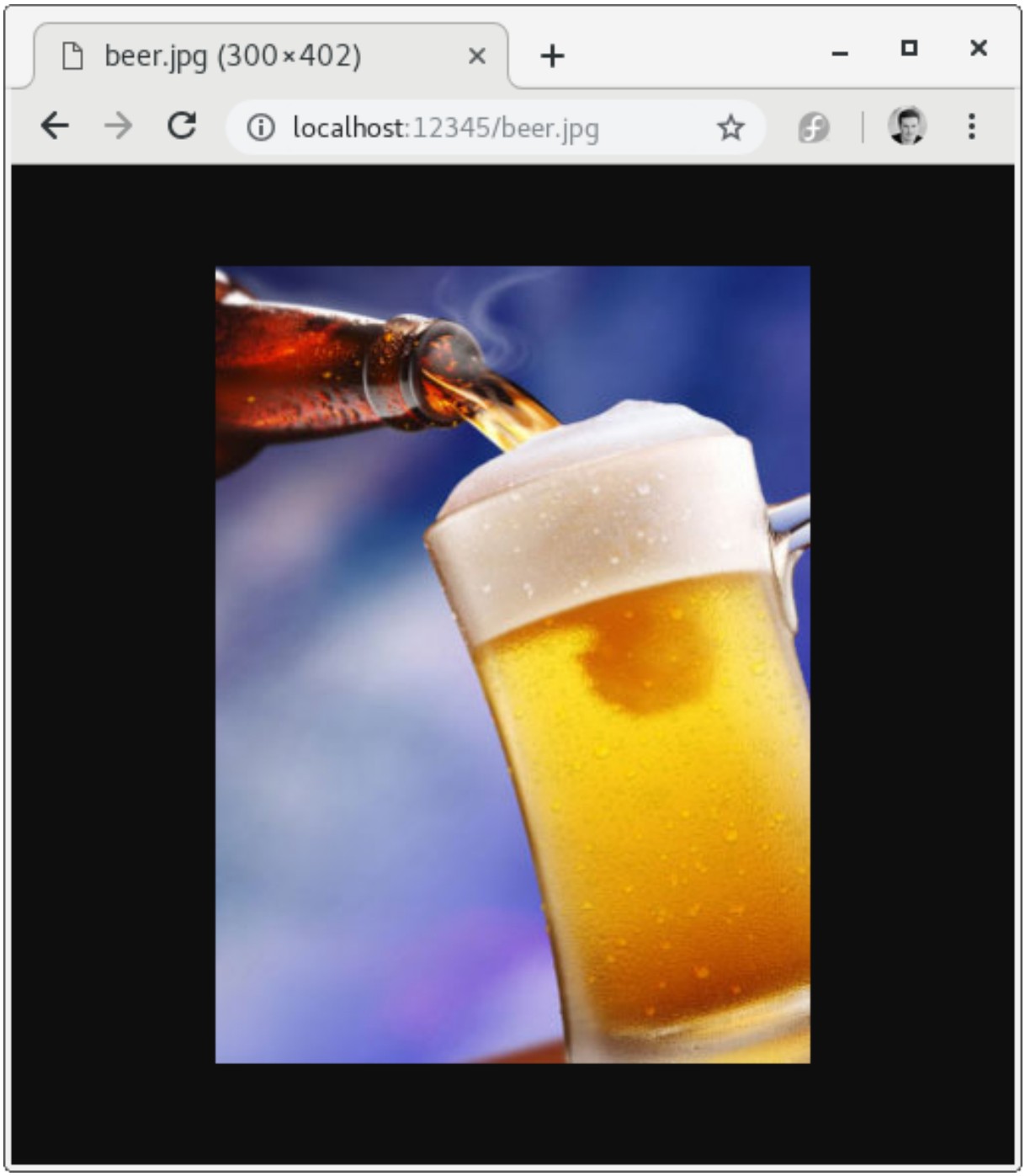

Figure 5: An image rendering inside a browser window, after it was requested via GET.

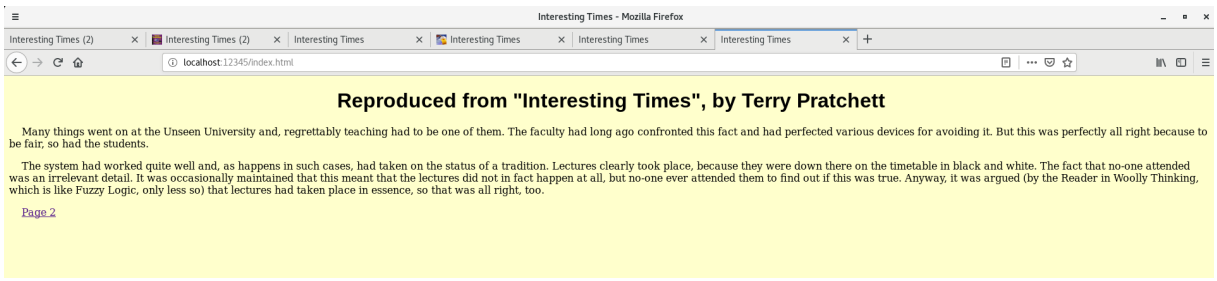

Figure 6: Several threads that were ran simultaneously (see Figure 7 for the terminal window output).

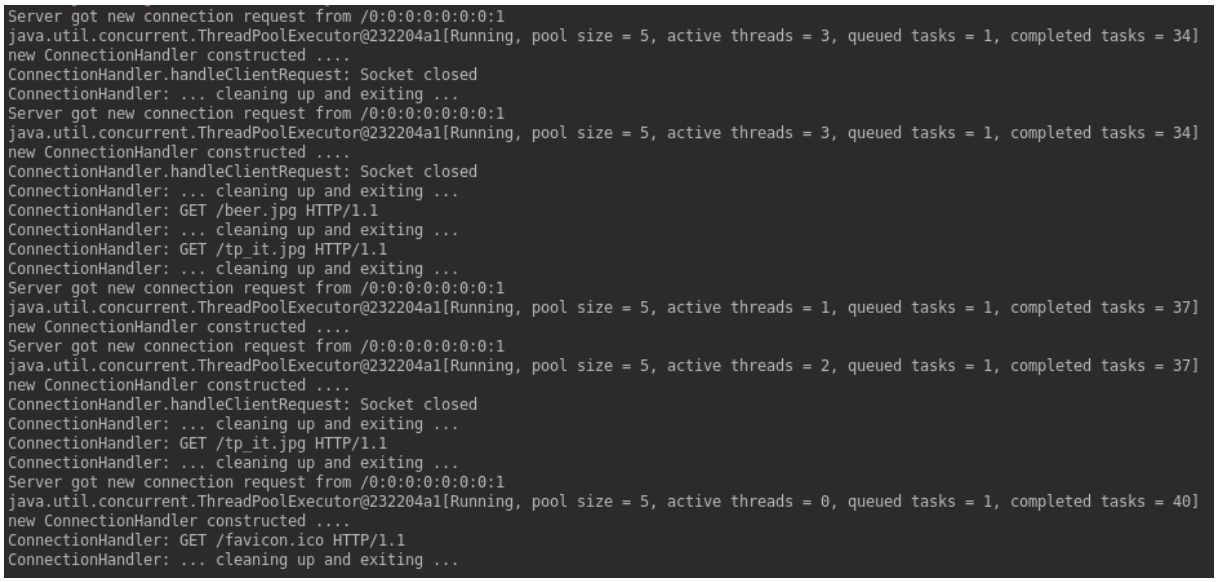

Figure 7: Terminal window inside IntelliJ IDE showing the maximum number of threads (5) serving 6 client requests simultaneously (see Figure 6 for browser window).