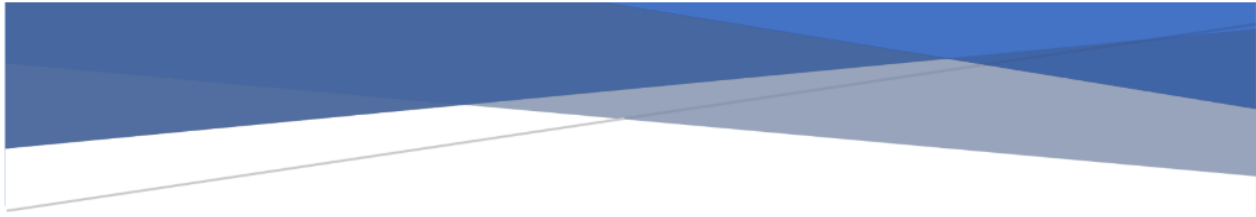


AustinH Networked Applications Report

by Austin Hooper

FILE	936_AUSTIN_HOOPER_AUSTINH_NETWORKED_APPLICATIONS_REPOR T_252504_337563990.DOCX (293.88K)		
TIME SUBMITTED	30-APR-2020 01:51PM (UTC+0100)	WORD COUNT	1146
SUBMISSION ID	125668725	CHARACTER COUNT	6212



Networked Applications (ACCA6015)

Networked Application Development

University of Wales Trinity Saint David

Austin Hooper (1700481)

Lecturer: Dr Tim Bashford

1st May 2020

Table of Contents

1. Introduction	1
2. Networked Application Development	2
3. Conclusion	6
References	7

Table of Figures

Figure 1 - Index PHP page	3
Figure 2 - Web server directory handling	3
Figure 3 - Images served from the server	4
Figure 4 - Server reading POST data.....	4

1. Introduction

This report details the development of a C++ web server. Features, design decisions and development good practice are discussed, followed by a conclusion and references list.

2. Networked Application Development

The application developed for this assignment was a HTTP web server. The list below describes the application's functionality:

- Receives requests from clients, currently supporting GET, HEAD and POST. GET requests are responded to with the file they requested along with headers such as content type and content length, and by saving any GET parameters to a vector of pairs (each pair contains key => value). POST requests are handled similarly, reading data from the body, and saving it to a vector, which is then presented as a HTML page. HEAD requests are handled similarly to GET requests, except the response has no body. The request type is stored as an enumeration value which is then used in a switch statement to run appropriate class methods.
- On receiving a HTTP request, the request's headers are stored in a vector of pairs, similarly to the GET and POST values, in the form of header => value. One example would be Connection: close. This was done to access them later, as well as providing room for additional functionality in the future.
- PHP is supported by running a command on the included PHP executable when a client requests a .php file. The output of this command is then used as the body of the response.
- Reading and serving of a variety of content types based on the request (only content types with media available to demonstrate were added). Requests for unimplemented file types will be responded to with "Error 415: Unsupported Media Type," in line with good practice. The supported file types are listed below:
 - JPG / JPEG
 - PNG
 - GIF
 - BMP
 - ICO
 - SVG
 - HTML
 - CSS
 - TXT
 - PHP
 - AVI
- Responding with a variety of HTTP response codes [1], [2, pp. 166-167]:
 - 200 OK
 - 201 Created
 - 400 Bad Request
 - 404 Not Found
 - 415 Unsupported Media Type
 - 500 Internal Server Error
 - 501 Not Implemented
- Responding with a custom-generated page when a directory is requested, showing its contents (Figure 2)

The functionality of the server is demonstrated with the following pages:

index.php (localhost, localhost/index.php)

This page demonstrates both the PHP functionality and the index default. If the address is input with no page, the server searches for index.html or index.php. This page is shown below using PHP (finding directories and performing arithmetic) in Figure 1:



Figure 1 - Index PHP page

localhost/media-tests/

When a directory is entered, the server responds with a list of the contents of the directory, with icons representing sub-directories and files, as seen below in Figure 2:

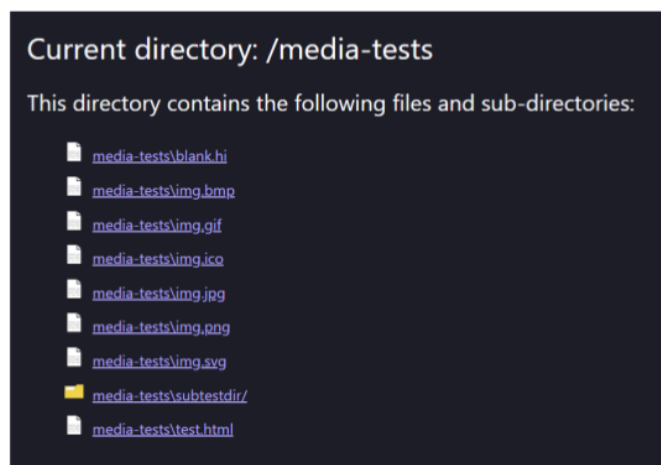


Figure 2 - Web server directory handling

localhost/media-tests/test.html

This page demonstrates the server's ability to respond with a variety of content types, and is shown below in Figure 3:

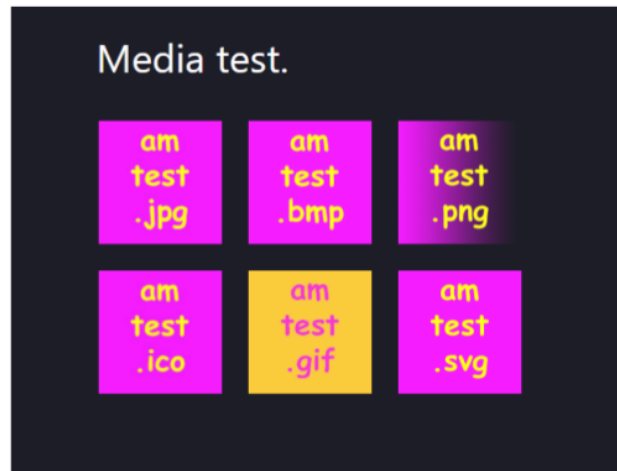


Figure 3 - Images served from the server

localhost/form-test.html

This page provides a short form to demonstrate the server's ability to capture POST data, which is shown in Figure 4:

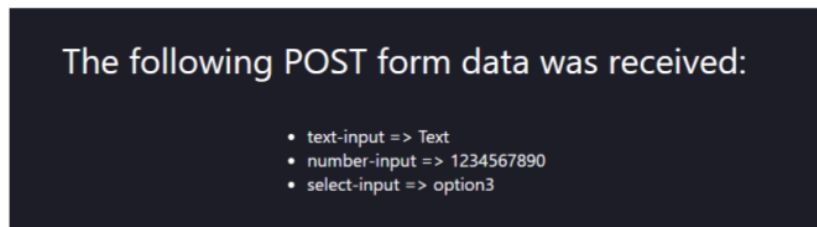


Figure 4 - Server reading POST data

The application uses the network code from in-class exercises as a starting point. This has been modified, primarily the Readn method in the ConnectionInstance class. As HTTP does not send a message length prior to sending the message, the approach to reading data was changed. It now checks for a content-length field when reading HTTP headers, and if found, will read until it encounters two instances of a new line and carriage return ("

"), indicating the body. From the end of the last

new line, it then reads the specified number of bytes, to capture the body. The application serves files from the html directory within the project. As part of the learning process, an attempt was made to use newer and more advanced features from the C++ STL throughout the application's development.

Three new classes have been implemented to facilitate HTTP server functionality:

- HTTP – Handles HTTP requests, holding an instance of ServerFileReader which reads files which it then serves. This class generates HTTP responses including their response code, content type, content length and body. Currently, it reads and stores entire HTTP requests, but only uses some of their headers.
- ServerFileReader – Reads files when requested by the HTTP class holding it. Reads in different ways dependent on file type, reading text files as plain text, and media files as binary data. This class holds a PhpHandler object which it uses if it is requested to read a .php file.
- PhpHandler – Runs a command to execute the included PHP executable with given parameters, parsing the files requested by ServerFileReader. Currently, this class has little functionality but was moved to a separate class to allow for future expansion and also adhere to software engineering good practice.

Where possible, enumeration types have been used, allowing for use of switch statements – this allows for easier expansion in future for different file or request types. The class layout is also such that expansion should be simple, although the ServerFileReader and HTTP classes could be further separated. Where possible, RFCs 2616 [3], 7231 [4] and 7230 [5] have been followed regarding formatting, content types, headers, and status codes. The application also follows guidance from the Mozilla Developer Network, specifically [6]–[8]. Wherever possible, the code has been made as simple to read as possible, primarily by separating functionality into classes and methods. This separation of code also increases its testability. As the application expands, abstract classes could be included, as polymorphism becomes necessary. The implemented functionality was tested and verified using the Postman software [9]. The server has also been tested in the latest versions of Chrome, Firefox and Edge for Windows 10 and no issues were found.

3. Conclusion

This report has discussed development of a C++ web server, including design decisions, features, and good practice.

References

- [1] D. Gourley and B. Totty, *HTTP The Definitive Guide*. Sebastopol, CA: O'Reilly, 2002.
- [2] W. R. Stevens, *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX@Domain Protocols*, vol. 10, no. 5. Boston: Addison Wesley, 1996.
- [3] Network Working Group, "Hypertext Transfer Protocol -- HTTP/1.1," RFC 2616, 1999.
- [4] Internet Engineering Task Force, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," RFC 7231, 2014.
- [5] Internet Engineering Task Force, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," RFC 7230, 2014.
- [6] Mozilla, "HTTP Messages - HTTP | MDN," 2019. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>. [Accessed: 26-Apr-2020].
- [7] Mozilla, "HTTP headers - HTTP | MDN," 2020. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>. [Accessed: 26-Apr-2020].
- [8] Mozilla, "MIME types (IANA media types) - HTTP | MDN," 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types. [Accessed: 26-Apr-2020].
- [9] Postman, "Postman v7.23.0." 2019.

AustinH Networked Applications Report

GRADEMARK REPORT

FINAL GRADE

97 / 100

GENERAL COMMENTS

Instructor

This is a really phenomenal piece of work, Austin, vastly beyond the scope I anticipated for the brief. You've clearly put a huge amount of effort into the completion of this work. The mark is reflective of the effort and skill which went into the completion of your code and the accompanying report. Extremely well done.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9