



Assessed Coursework

| | | | | |
|--|---|--------|-------|------------------|
| Course Name | Networked Systems (H) | | | |
| Coursework Number | Summative exercise 1 | | | |
| Deadline | Time: | 4:30pm | Date: | 13 February 2017 |
| % Contribution to final course mark | 10% | | | |
| Solo or Group ✓ | Solo | ✓ | Group | |
| Anticipated Hours | 10 | | | |
| Submission Instructions | Submit via Moodle, following instructions in the lab 2 handout. | | | |
| Please Note: This Coursework cannot be Re-Assessed | | | | |

Code of Assessment Rules for Coursework Submission

Deadlines for the submission of coursework which is to be formally assessed will be published in course documentation, and work which is submitted later than the deadline will be subject to penalty as set out below.

The primary grade and secondary band awarded for coursework which is submitted after the published deadline will be calculated as follows:

- (i) in respect of work submitted not more than five working days after the deadline
 - a. the work will be assessed in the usual way;
 - b. the primary grade and secondary band so determined will then be reduced by two secondary bands for each working day (or part of a working day) the work was submitted late.
- (ii) work submitted more than five working days after the deadline will be awarded Grade H.

Penalties for late submission of coursework will not be imposed if good cause is established for the late submission. You should submit documents supporting good cause via MyCampus.

Penalty for non-adherence to Submission Instructions is 2 bands

You must complete an "Own Work" form via <https://studentltc.dcs.gla.ac.uk/> for all coursework

Networked Systems (H) laboratory exercise 2: Further Network Programming in C

Dr Colin Perkins
School of Computing Science
University of Glasgow

<https://csperkins.org/teaching/2016-2017/networked-systems/>

18 January 2017

Introduction

The laboratory exercise last week introduced you to network programming using the Berkeley Sockets API, by implementing a simple networked “Hello, world” application. In this exercise, you will extend and debug a simple web server application that was previously implemented in C using Berkeley Sockets. This is a summative exercise, that is worth 10% of the marks for the course.

Background

In this lab you will debug and extend a simple web server. A web server is a program that listens for, accepts, and responds to requests made by web clients (i.e., browsers) using the hypertext transport protocol (HTTP) to deliver web pages. To complete this exercise you must understand the basic operation of the HTTP protocol, and the way it’s implemented in the sample web server.

The Hypertext Transport Protocol

A web browser uses the Hypertext Transport Protocol (HTTP) to retrieve pages from a web server. The browser makes a TCP/IP connection to the web server, sends an HTTP request for the desired web page over that connection, reads the response back, and then displays the page. HTTP requests and responses are text-based, making the network protocol human-readable, and straight-forward to understand.

In version 1.1 of HTTP, a request comprises a single line command (the “method”), followed by one or more header lines containing additional information. To retrieve a page, a web browser uses the GET method, specifying the page to retrieve and the version of the HTTP protocol used (this exercise uses the HTTP/1.1 protocol). For example, a browser would send the method `GET /index.html HTTP/1.1` to retrieve the page `/index.html` from a server. Following the GET method line will be a sequence of header lines, giving more information about the request and the capabilities of the browser. One of these header lines will be a `Host:` header, giving the name of the site, for example `Host: www.gla.ac.uk`, used to allow a single servers to host more than one site. After the headers is a blank line, indicating end of request.

For example, to fetch the main University web page (<http://www.gla.ac.uk/index.html>), a browser would make a TCP/IP connection to `www.gla.ac.uk` port 80, and send the following request:

```
GET /index.html HTTP/1.1
Host: www.gla.ac.uk
```

Note that each line ends with a carriage return ('\r') followed by a new line ('\n'), and the whole request ends with a blank line (i.e., a line containing nothing but the \r\n end of line marker). The example above is a minimal HTTP request. A web browser will usually include other headers, in addition to the Host: header, to control the connection, indicate support for particular file formats and languages, to convey cookies, and so on.

When it receives an HTTP GET request for a web page that exists, a web server will reply with a HTTP/1.1 200 OK response, followed by more header lines providing information about the response, a blank line, and then the contents of the page to be displayed. The header lines should include a Content-Length: header, which specifies the size of the page in bytes, and a Content-Type: header that describes the format of the page. The server can also include other header lines, to specify additional information about the response. As with the request, each header line ends with a carriage return followed by a new line. Finally, a blank line separates the headers from the page content. An example of a response follows ("..." indicates omitted text):

```
HTTP/1.1 200 OK
Date: Tue, 12 Jan 2010 11:18:30 GMT
Server: Apache/1.3.34 (Unix) PHP/4.4.2
Last-Modified: Tue, 12 Jan 2010 09:59:31 GMT
ETag: "1a-3d4e-4b4c4803"
Accept-Ranges: bytes
Content-Length: 15694
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
  <title>University of Glasgow :: Glasgow, Scotland, UK</title>
</head>
<body>
  ...
</body>
</html>
```

In this example, the "Content-Length:" is 15694 bytes and the "Content-Type:" is text/html, meaning that there are exactly 15694 bytes of HTML text in the body of the response (starting with the "<" of the "<!DOCTYPE" line, after the blank line indicating end-of-header, and finishing with the ">" of the "</html>" line).

The "Content-Type:" header will take different values depending on the type of file returned. Commonly used values are:

| Filename: | Content-Type: |
|---------------|--|
| *.html, *.htm | Content-Type: text/html |
| *.css | Content-Type: text/css |
| *.txt | Content-Type: text/plain |
| *.jpg, *.jpeg | Content-Type: image/jpeg |
| *.gif | Content-Type: image/gif |
| *.png | Content-Type: image/png |
| (unknown) | Content-Type: application/octet-stream |

The IANA maintains the master list of standard content type values. It is available from their website at <http://www.iana.org/assignments/media-types>.

If the browser requests a non-existing file, the server will respond with a HTTP/1.1 404 Not Found response. In this case, the body of the response is the error page to display, and the headers give information about the error. An example might be:

```
HTTP/1.1 404 Not Found
Date: Tue, 20 Jan 2009 10:31:56 GMT
Server: Apache/2.0.46 (Scientific Linux)
Content-Length: 300
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head>
<title>404 Not Found</title>
...
</body>
</html>
```

Other types of response are possible, distinguished by the numeric code in the first line of the response.

A browser will open one or more connections to a server for each web page. On each connection, the browser will send a request, then wait for and process the response from the server. It may then send another request on the same connection, perhaps after waiting some time, or close the connection. If the server wants the browser to close the connection immediately, it can include a `Connection: close` header in its response to the request.

A Sample Web Server

The zip archive associated with this lab exercise contains the source code for a simple web server, as an example of a networked application using TCP/IP. The server is written in C, using the Berkeley Sockets API, and runs on the Linux machines in the lab. The zip archive also contains a Makefile, and a small sample website.

The web server implementation uses a single thread to `accept()` connections from new clients. It transfers each new connection to a worker thread from a thread pool for processing, by passing the file descriptor to the thread. Once a worker thread gets the file descriptor, it enters a loop where it reads the request, sends the response, and repeats until the browser closes the connection. The thread then blocks until it's passed a new connection. The web server is a minimal implementation, that contains only the bare minimum features needed to serve a simple website.

Download the file `lab02.zip` from the course website. Extract the zip archive, to create a directory called `lab02`. Open a shell in that directory, and run `make` to build the web server. Then, run `./wserver` to start the server. The server listens for connections on port 8080 of the host on which it's running. For example, if you run the server on host `bo720-1-01u.dcs.gla.ac.uk`, it will be reachable by the URL `http://bo720-1-01u.dcs.gla.ac.uk:8080/`. Browse the sample website using the server you have just built, by running a browser and opening the appropriate URL. Review the provided code, and familiarise yourself with its operation. You will notice that the web server provided is buggy and incomplete, and will not correctly serve image files. If you don't understand the HTTP protocol, or its implementation in the server, then ask one of the demonstrators.

Summative Exercise 1: Web Server

In Summative Exercise 1 you will debug and extend the web server provided in the `lab02.zip` file. There are three parts to the exercise, as follows:

1. Debug the web server, by modifying the file `wserver.c` so it correctly sends JPEG format images back to the browser. The sample website provided with the server contains some JPEG format images. You have successfully solved this part of the exercise when you can browse this sample

site, served by the web server, and have your browser display all the images, and indicate that it has finished downloading the page.

2. Modify the server code, in `wserver.c`, so that if a browser requests a URL representing a directory, and the file `index.html` exists in that directory, the server returns a redirect asking the browser to fetch the `index.html` file instead. This is done by sending a **307 Temporary Redirect** HTTP response. For example, if the browser requests `http://example.com/foo/`, the server would redirect it to `http://example.com/foo/index.html` by sending a response like:

```
HTTP/1.1 307 Temporary Redirect
Location: /foo/index.html
Content-Length: 143
Content-Type: text/html
```

```
<html>
  <head>
    <title>Redirected</title>
  </head>
  <body>
    <p>Redirecting ...</p>
  </body>
</html>
```

The **Location:** header gives the location to which the browser is being redirected. The HTML body is for backwards compatibility, and is displayed by old browsers that don't understand the redirect response. Ensure the redirect works whether or not the URL representing the directory ended in a `/` character.

3. Modify the server code, in `wserver.c`, so that if a browser requests a URL representing a directory, and the file `index.html` does not exist in that directory, it returns instead a dynamically generated HTML page (a **200 OK** response) that contains a listing of the contents of the corresponding directory. Each entry in the listing should be a link to the corresponding file or directory. Be careful to avoid race conditions when generating the directory listing.

Hint: use the `opendir()`, `readdir()`, and `closedir()` functions from the Linux standard library to read a directory listing.

To complete this exercise you will need to modify the `wserver.c` file included in the zip archive. Do not modify any of the other files in the archive, and make only the minimum number of changes required. Ensure any modifications you make to the `wserver.c` file match the code style of the surrounding code.

Your modified `wserver.c` file must compile cleanly, without any warnings or errors, using the provided `Makefile`, and run on the Linux machines in the labs.

Submission Instructions

Once you have completed the assignment, make a copy of the `wserver.c` file – containing your modifications – under a new name, with a dash and your 7-digit numeric matriculation number before the extension (note: 7-digits only, *do not* include the first letter of your surname). You can do this using the command `cp wserver.c wserver-matric.c` (replacing `matric` with your matriculation number). Submit the resulting `wserver-matric.c` file via Moodle. Do not submit any other files.

This is an assessed exercise, worth 10% of the marks for this course. The deadline for submissions is 4:30pm on Monday 13 February 2017. The Code of Assessment allows late submission up to 5 working days beyond this deadline, subject to a penalty of 2 bands for each working day, or part thereof, the

submission is late. Submissions received more than 5 working days after the due date will receive an H (band value of 0). *These penalties will be strictly enforced.*

Submissions that are not made via Moodle, that have the wrong filename, or that otherwise do not follow the submission instructions will be subject to a 2 band penalty. *This penalty will be strictly enforced* and will be applied in addition to any late submission penalty.

Marking Scheme

Marks will be awarded for:

- Modifying the `wserver.c` file so it correctly send JPEG format images back to the browser:
 - [1 mark] for code that works; and
 - [1 mark] for correctly identifying and fixing the underlying problem rather than working around the bug.
- Modifying the `wserver.c` file so it correctly redirects browsers that request a URL representing a directory to the `index.html` file in that directory, if it exists:
 - [1 mark] for a response with the correct syntax;
 - [1 mark] for including the correct URL in the `Location:` header;
 - [1 mark] for including a correct `Content-Length:` header;
 - [1 mark] for including an appropriate body containing valid HTML;
 - [2 marks] for correct code to generate and send the response;
 - [2 marks] for making appropriate use of existing functions in the `wserver.c` file; and
 - [2 marks] for matching the style of the existing code.
- Modifying the `wserver.c` file so it correctly displays a directory listing if the browser requests a URL representing a directory and the corresponding `index.html` file does not exist:
 - [4 marks] for correct use of `opendir()`, `readdir()`, and `closedir()`;
 - [4 marks] for correctly building up a response while reading the directory listing, with dynamic allocation of memory as needed;
 - [2 marks] for correctly formatting the HTTP headers, including a correct `Content-Length:` header;
 - [2 marks] for generating valid HTML in the response;
 - [2 marks] for correct code to generate and send the response;
 - [2 marks] for making appropriate use of existing functions in the `wserver.c` file; and
 - [2 marks] for matching the style of the existing code.

Marks will be deducted for the following general issues occurring in code you write or modify:

- up to [2 marks] for compiler errors or warnings;
- up to [4 marks] for excessive abstraction, over-engineering, or poor design;
- up to [2 marks] for use of global variables;
- up to [2 marks] for abuse of the type system;

- up to [2 marks] for reinventing standard library functions;
- up to [2 marks] for use of unallocated memory/using memory after free()/memory leaks;
- up to [2 marks] for buffer overflows, race conditions, or other security problems;
- up to [2 marks] for missing error handling; and
- up to [2 marks] for code duplication and repetition.

The result will be a numeric mark out of 30. This numeric mark will be converted to a percentage, then the percentage will be converted to a band on the 22-point University of Glasgow scale using the standard translation table for the School of Computing Science. Any applicable penalty for late submission and/or for not following submission instructions will then be applied, and a band will be returned. A brief written justification for the band will also be supplied.