COMP1521 18s2

# Week 09 Lab Exercise
## A Minimal Web Server

Computer System Fundamentals

## Objectives

- to learn about HTTP
- to practice working with sockets

## Admin

| | |
|---|---|
| **Grades** | A+=outstanding, A=very good, B=adequate, C=sub-standard, D=hopeless |
| **Demo** | in the Week09 Lab or at the start of the Week03 Lab |
| **Submit** | `give cs1521 lab09 server.c` or via WebCMS |
| **Deadline** | must be submitted by 11:59pm Sunday 23 September |

Note: you need to do something *truly* outstanding, above and beyond the "call of duty" to get A+. Doing the exercise well and correctly as specified will get you an A grade. An A grade gets you full marks; an A+ grade gives a small bonus.

## Background

## Setting Up



Make a new directory for this lab, change into that directory, and run the command:



```
$ unzip /home/cs1521/web/18s2/labs/week09/lab09.zip
```

To work on this at home, download the ZIP file and unzip it on your own machine. You should be able to get a web server working on most implementations of Unix. It works on Linux and MacOS, at least, but no guarantees about other systems.



The `unzip` command will place the following files in you directory

| | |
|---|---|
| `Makefile` | a set of dependencies used to control compilation |
| `server.c` | a small skeleton program for the warm-up exercise |

The main task for this lab is to build a minimal web server that can run on your local machine (e.g. a CSE workstation) and can interact with a web server running on the same machine.

You should read the code in the`server.c` file to work out what needs to be done (look for `TODO`). You might recognise much of the code in the `main()` function from one of the examples in the Week08 Lectures. See if you can remember what all of the parameters and data types were for the various socket functions; or at least recall the general method for setting up a socket on a server: create a socket, bind it to an address (host+port), start it listening, accept connections.

You can use the `Makefile` to compile the server (and it will give compile errors until you fix it). When it's executable, you run the server in one terminal window and send requests to it either from another terminal window (using the `curl` command) or from a web browser running on the same machine as the server. You can use the output appearing in the terminal where the server is running for debugging.

An example of running the server, and the kind of output you might see:

```
$ ./webserver
WebServer: waiting for connections...
   ... the server will wait here ...
```

```
   ... until it receives a request ...
   ... at which point it will display ...
WebServer: got connection
Request: GET /hello HTTP/1.1
   ... after processing the request ...
   ... the server will wait here ...
   ... until it receives a another request ...
   ... etc. etc. etc. ...
```

Note: if you leave your webserver running and someone else comes to use the workstation, or even if you try to run it again, you will receive the error message:

```
$ ./webserver
WebServer: binding socket: Address already in use
```

and your server won't start. If it's your webserver that's causing this, simply kill it. If some other inconsiderate person has left their server running, you can try to get around this by using a different port number. You will need to change it in the code, then recompile, and then use different URLs to the ones below (using the new port number).

## Exercise

The `server.c` files contains a partially complete implementation of a web server. The server runs on the local host and writes logging/debugging information on its standard output. You need to complete it so that it can respond to the following HTTP requests:



| Request | Response |
|---|---|
| `http://localhost:3490/` | Server running ... |
| `http://localhost:3490/hello` | Hello |
| `http://localhost:3490/hello?John` | Hello, John! |
| `http://localhost:3490/date` | Sat Sep 15 16:35:44 2018 |
| `http://localhost:3490/xyz` | 404 Page Not Found |

Obviously, we don't always want the same date as above. You need to extract the current date/time and render that as a character string. The library functions `time()` and `ctime()` will help; read the `man` entries to find out how they work.

Similarly, we don't always want the name `John`. The server should return an appropriate response whatever name is supplied. You can use the `sscanf()` function to extract the name, or do it some other way if you wish.

Note that if you test the server using `curl`, you will see the logging and debugging information intermingled with the HTTP response on your terminal. If you send requests through a web browser, you will see the responses (once your server is working) in the browser's window.

**Challenge exercises**

Make the webserver handle other HTTP operations, such as POST and PUT.

## Submission

You need to submit one file: `server.c`. You can submit this via the command line using `give` or you can submit it from within WebCMS. After submitting the code, show your tutor, who'll give you feedback on your work and award a grade.

Have fun, *jas*