Documentation

Networks Assignment 1

1575286

This documentation serves to guide the user to use the C web server as tasked in Networks Assignment 1. The web server is written in C programming language and websites in HTML.
Two fonts will be used, Calibri which is intended for description, and `Consolas`, which to emphasise code or anything going on inside a computer.

Basic Requirements:

- Server
  - Linux Machine (code uses linux functions)
  - Gcc & make (for compiling)
- Client
  - Linux Machine (to use the client program)
  - Gcc (for compiling client)
  - Modern browser (Chrome/Mozilla/Edge)

File structure

I have used and build up upon Ian's code to complete this web server. // comments present beside file names indicate the things I created or changed for this assignment. The **`src`** directory consists of the following files:

```
-   404.html       // HTML pages, what is going to be served to
-   firstpage.html // the client that connects
-   index.html     //
-   client.c
-   get_listen_socket.c
-   get_listen_socket.h
-   main.c
-   make_printable_address.c
-   make_printable_address.h
-   Makefile // file to assist compilation
-   README
-   service_client_socket.c // thread function that serves the webpage
-   service_client_socket.h
-   service_listen_socket.c
-   service_listen_socket.h
-   service_listen_socket_multithread.c
-   service_listen_socket_multithread.h
```

There is a Makefile provided to compile all the programs. It is always safe to remove all object files before compiling, so do:

```
$ make clean // removes object (.o) files in the folder
$ make       // compiles the code
```

This Makefile generates two binaries for use:

- `web_server`
- `client`

Explanation

The next few sections will describe the usage and description of the implemented web server.

Website

The website consists of three items, index page, firstpage, and a 404 error page.
The index is what the user sees when first connecting to the server, then there is link which directs the user to the first page. The first page has a link that goes back to the index. If the user requests for a page other than these two, the 404 error page will be given as a response.

Server

Run the server by going to terminal in the directory, and do:

```
        $ ./web_server [insert port here from 0-65535]
                // ports 0 – 1023: reserved by the system, need privileges
e.g     $ ./web_server 9999
```

The web_server program takes in one argument, the port, and will host in localhost. In case when the number of arguments is invalid, the program will print to stdout the usage hint.

Upon successful, the server will print e.g. “`binding to port 9999`”.
At this state, the server is listening and ready for any client to connect to it. For every connection made, the server will print to stdout the HTTP Request Header from the client. The server is also robust and can handle arbitrary long inputs. A client connection may be simulated in several ways:

- Open a browser and go to `localhost:port`
  This will create the request and it should route and serve `index.html` page.
- Open a terminal and type `curl localhost:port`
  This returns the webpage as text.
- Run client program `./client localhost port` and manually type the http request for the server. E.g. `GET / HTTP\1.1`

There are many other ways to connect such as `wget` which saves the response as a file too.

Blank URI requests will automatically route to the `index.html` page. An invalid URI will route to the `404.html` page.

Extension

I managed make such that the web server accepts a single byte range. E.g. `Range: bytes=0-1023`
This means that the client requests a certain range of bytes and the server only gives back data that is from the requested range.

- In a valid byte range request, the server will return the data together with a 206 Partial Content HTTP Response Header.
- In an invalid one, however, the server will return no data and a 416 Range Not Satisfiable HTTP Response Header.

Functions

These describes the functions inside `service_client_socket.c` that serves contents and/or errors to connected clients. More information can be found as comments in the code.

- `char *fileToStr(file)`
    - Takes a path to the file and returns the string of it.
- `char *route(page)`
    - Takes the string of a file and returns a server response.
- `char *grabRange(page, start, end)`
    - Returns a range of the page delimited by `start` and end.
- `char *make_header(code, content_length, range_start, range_end, page_length)`
    - Helper method to create a HTTP Response Header based on the `code`.
- `int validRange(length, start, end)`
    - Helper method to determine whether a given range from start to end is valid for the page of size length.
- `int service_client_socket(socket, tag)`
    - This is the method that is called when a client connects.