HTTP Server Documentation

This document will explain the code to serverMain.c, in chronological order as it is executed.

Build the program with make.
To run the program, run:

```
$ serverMain <portnum>
```

See page 3 for additional useful information when using the server.

Main Method

The first lines in main are for parsing the argument. The program requires a legal portnum to proceed.

Main() creates an IPv6 sockaddr structure for the server, and binds a socket to it.
Every step in socket creation is error tested, and exits on failure.

After a successful socket initialisation, the listen loop kicks in.
The listen loop allocates a socket which is bound to the sockaddr structure which contains client information. In hindsight I would use a thread control block to pass arguments into accept() as this method is a lot more thread safe.

The pthread is created with it’s routine being defined as the function “handleConnection”.

void *handleConnection(void *args)

This subroutine is very simple, its purpose is to decode an HTTP request.
In psuedocode:

```
while(1){
    read data from newSocket
    Separate first line of incoming data.
    Check if this first line matches a HTTP GET request by strstr.
    If request is HTTP GET{
        Use ‘strtok’ and a temporary char buffer to separate the requested address
        into it’s own string
        Serve the file via serveFile()
    }
    Else{
        Error that only GET requests are supported.
        Send 501
    }
    Free allocated memory, cleanup thread
}
```

void serveFile(char *filePath, int *newSock)

This function is what fetches a file, reads the data into a buffer and writes the buffer downstream, with an appropriate HTTP header.

filePath is the raw file requested by the HTTP GET request, e.g. “/”.
All filePath inputs are prepended with “www” such as to redirect all file requests to the www subdirectory.
A request of “/” is redirected to open “www/index.html” otherwise a request of “/request” attempts to serve “www/request”.

fopen tries to open the requested file. If no such file exists then a 404.html is opened.
If 404.html is missing then the server closes the connection, sending an empty 404 code.

The size of the serving file is discerned via fseek-ing the end of file, tracking the number of bytes traversed and then rewinding the pointer back to the head of the file.

The file is read to a string htmlContents via the fread() method.

If a blank 404 error code is sent to the client, a dumb buffer of 1024 bytes is used to write the output; if however, a file is being served to the client, a *longOutput* buffer is constructed with a size enough to contain data from an arbitrarily long file.

The data is written to the client.

int getIntLen(int value)

A helper function to calculate the number of bytes that a decimal representation of a number uses. This function is called when allocating a buffer to contain the HTTP response.

Known Bugs & Useful information

The only main bug to concern yourself with is that, depending on the nature of serverMain's exit, the socket's port is not immediately/always released, and re-running the server on the same port may return an "Error on binding".
The bug is infrequent, but if it occurs, please run serverMain on a different port, after which the original port will become free to use again. I obviously haven't solved this bug yet!