Assignment 5

CS 301 - Operating Systems

Due date: 18th November, 2021

1 Introduction

The Hypertext Transport Protocol (HTTP) is the most commonly used application protocol on the Internet today. Like many network protocols, HTTP uses a client-server model. An HTTP client opens a network connection to an HTTP server and sends an HTTP request message. Then, the server replies with an HTTP response message, which usually contains some resource (e.g. file, text, binary data) that was requested by the client.

In this assignment, you will implement a HTTP server that handles HTTP GET requests. You will provide functionality through the use of HTTP response headers, add support for HTTP error codes, create directory listings with HTML, and create a HTTP proxy. The request and response headers must comply with the HTTP 1.0 protocol found at `http://www.w3.org/Protocols/HTTP/1.0/spec.html`.

2 Background

2.1 HTTP Request

The format of a HTTP request message is:

- A HTTP request line containing a method, a query string, and the HTTP protocol version
- Zero or more HTTP header lines
- A blank line (i.e. a CRLF by itself)

The line ending used in HTTP is CRLF, which is represented as
r
n in C. Below is an example HTTP request message sent by the Google Chrome browser to a HTTP web server running on localhost (127.0.0.1) on port 8000 (the CRLF's are written out using their escape sequences):

```
GET /hello.html HTTP/1.0\r\n
Host: 127.0.0.1:8000\r\n
```

```
Connection: keep-alive\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
User-Agent: Chrome/94.0.4606.81\r\n
Accept-Encoding: gzip,deflate,sdch\r\n
Accept-Language: en-US,en;q=0.8\r\n
\r\n
```

Header lines provide information about the request. Here are some HTTP request header types:

- Host: contains the hostname part of the URL of the HTTP request (e.g. `iitgn.ac.in` or 127.0.0.1:8000)
- User-Agent: identifies the HTTP client program, takes the form Program-name/x.xx, where x.xx is the version of the program. In the above example, the Google Chrome browser sets User-Agent as Chrome/94.0.4606.81

2.2 HTTP Response

The format of a HTTP response message is:

- An HTTP response status line containing the HTTP protocol version, the status code, and a human-readable description of the status code
- Zero or more HTTP header lines
- A blank line (i.e. a CRLF by itself)
- The body (i.e. content) requested by the HTTP request

Here is an example HTTP response with a status code of 200 and a body consisting of an HTML file (the CRLF's are written out using their escape sequences):

```
HTTP/1.0 200 OK\r\n
Content-Type: text/html\r\n
Content-Length: 84\r\n
\r\n
<html>\n
<body>\n
<h1>Hello World</h1>\n
<p>\n
Let's see if this works\n
</p>\n
</body>\n
</html>\n
```

Typical status lines might be `HTTP/1.0 200 OK`, `HTTP/1.0 404 Not Found`, etc. The status code is a three-digit integer, and the first digit identifies the general category of response:

- 1xx indicates an informational message only
- 2xx indicates success
- 3xx redirects the client to another URL
- 4xx indicates an error in the client
- 5xx indicates an error in the server

Header lines provide information about the response. Here are some HTTP response header types:

- Content-Type: the MIME type of the data attached to the response, such as `text/html` or `text/plain`
- Content-Length: the number of bytes in the body of the response

3 Your Task

3.1 HTTP Server Outline

From a network standpoint, your basic HTTP web server should implement the following:

1. Create a listening socket and bind it to a port
2. Wait for a client to connect to the port
3. Accept the client and obtain a new connection socket
4. Read in and parse the HTTP request
5. Serve a file from the local file system, or yield a 404 Not Found
6. Send the appropriate HTTP response header and attached file/document back to the client (or an error message)

The skeleton code already implements steps 2-4. Part 1 of this assignment is to bind the server socket to an address and to listen for incoming connections. This will be done with the `bind()` and `listen()` syscalls. Parts 2 and 3 of this assignment are to serve files and directories to the client. You will then implement a variety of methods to handle client requests (Parts 4-5).

3.2 Usage of `httpserver`

Below is a description of how to invoke `httpserver` from the shell. The argument parsing step has already been implemented for you:

```
./httpserver --help
Usage:  ./httpserver --files any_directory_with_files/ [--port 8000
--num-threads 5]
```

The available options are:

- `--files` — Selects a directory from which to serve files. You should be serving files from the `hw5/` folder (e.g. if you are currently `cd`'ed into the `hw5/` folder, you should just use `"--files www/"`.
- `--port` — Selects which port the HTTP server listens on for incoming connections. Used in both files mode and proxy mode. If a port number is not specified, port 8000 is the default.
- `--num-threads` — Indicates the number of threads in your thread pool that are able to concurrently serve client requests. This argument is initially unused and it is up to you to use it properly.

The `--num-threads` argument is used to specify the amount of worker threads in the thread pool. This will only be used in Part 7 of the assignment.

If you want to use a port number between 0 and 1023, you will need to run your HTTP server as root. These ports are the "reserved" ports, and they can only be bound by the root user. You can do this by running `"sudo ./httpserver --port PORT --files www/"`.

3.3 Assignment

3.3.1 Finish setting up the server socket in the `serve_forever()` function

- Bind the socket to an IPv4 address and port specified at the command line (i.e. `server_port`) with the `bind()` syscall.
- Afterwards, begin listening for incoming clients with the `listen()` syscall. At this stage, a value of 1024 is sufficient for the backlog argument of `listen()`.
- After finishing Part 1, access to the server should output "Empty reply from server".

3.3.2 Implement `handle_files_request(int fd)` to handle HTTP GET requests for files. You will need to call `serve_file()` accordingly. You should also be able to handle requests to files in sub-directories of the files directory (e.g. `GET /images/....`)

- If the file denoted by path exists, call `serve_file()` on it. Read the contents of the file and write it to the client socket.
  - Make sure you set the correct `Content-Length` HTTP header. The value of this header should be the size of the HTTP response body, measured in bytes. For example, `Content-Length: 7810`. You can use `snprintf()` to convert an integer into a string.
  - You must use the `read()` and `write()` syscalls for this assignment

- Else, serve a `404 Not Found response` (the HTTP body is optional) to the client. There are many things that can go wrong during an HTTP request, but we only expect you to support the 404 Not Found error message for a non-existent file.
- After finishing Part 2, looking up `index.html` should output the contents of the file `index.html`.

3.3.3 Implement `handle_files_request(int fd)` to handle HTTP GET requests for both files and directories.

- You will now need to determine if path in `handle_files_request()` refers to a file or a directory. The `stat()` syscall and the `S_ISDIR` or `S_ISREG` macros will be useful for this purpose. After finding out if path is a file or a directory, you will need to call `serve_file()` or `serve_directory()` accordingly.
- If the directory contains an `index.html` file, respond with a `200 OK` and the full contents of the `index.html` file. (You may not assume that directory requests will have a trailing slash in the query string.) The `http_format_index()` function in `libhttp.c` may be useful.
- If the directory does not contain an `index.html` file, respond with an HTML page containing links to all of the immediate children of the directory (similar to `ls -1`), as well as a link to the parent directory. The `http_format_href()` function in `libhttp.c` may be useful. To list the contents of a directory, good functions to use are `opendir()` and `readdir()`
- If the directory does not exist, serve a `404 Not Found` response to the client.
- After finishing Part 3, looking up the root directory `/` should output the contents of the file `index.html`