Lab 1: HTTP Web Client (5 Points)

1 Download Lab

Download the Lab 1 files from Brightspace. The code files will be inside the directory "`Lab1/`". You must use the environment from Lab 0 to run and test your code. Next, open a terminal and "`cd`" into the "`Lab1/`" directory. Now you are ready to run the lab!

2 HTTP Web Page Downloader

There is a very useful program called "`wget`". It is a command line tool to download a web page:

$ `wget http://www.gnu.org/software/make/manual/make.html`

This will download the make manual page, "`make.html`", and save it in the current directory. "`wget`" can do much more (e.g., downloading a whole web site). See the manual for "`wget`" for more info.

For this lab, your task is to write a limited version of "`wget`", that downloads a ***single*** file at a time. You will do your implementation in the "`http_client.c`" file inside the "`http_client`" directory. **You must <u>not</u> include any other C libraries except the ones provided in "`http_client.c`" file.**

To build the code, first "`cd`" into the "`http_client`" directory and compile using the "`Makefile`",

$ `make`

Finally, run the code using the following command,

$ `./http_client [host] [port number] [filepath]`

For example, `./http_client www.gnu.org 80 /software/make/manual/make.html`

In the above command, you give the components of the URL separately in the command line — (1) the server host, (2) the server port number (always 80 for HTTP), and (3) the file path. The program should download the given file and save it in the current directory. So in the example above, the program should produce "`make.html`" in the current directory. It should overwrite any existing file with the same name.

You must build, run, and test your code on `eceprog` using the environment from Lab 0. If your code does not run in that environment, you will not get any credit!

<u>Resources</u>: We provided you with the sample socket code in Lab 0 for both the TCP (SOCK_STREAM) and UDP (SOCK_DGRAM) connections. You are free to copy or re-use the provided code as you wish.

Some useful hints:

1. The program should open a TCP socket connection to the host and port number specified in the command line, and then request the given file using HTTP/1.x protocol.
   (See http://www.jmarshall.com/easy/http/ for the details of HTTP/1.x protocol).
2. An HTTP `GET` request looks like this:

   ```
   GET /path/file.html HTTP/1.0\r\n
   [zero or more headers]\r\n
   [blank line]\r\n
   ```

   Include the following header in your request:

   ```
   Host: <the_host_name_you_are_connecting_to>:<port_number>
   ```

3. The response from the web server will look something like this:

```
HTTP/1.0 200 OK\r\n
Date: Fri, 31 Dec 2020 23:59:59 GMT\r\n
Content-Type: text/html\r\n
Content-Length: 1354\r\n
[blank line]\r\n
[file content]
```

   There might be slight variations in the formats of responses from different servers. Go through the document from Step 1 (in particular, this section) to ensure that your parser is robust.

   The code "200" in the first line indicates that the request was successful. If it's not "200", your program should **print *only* the first line of the response to the terminal (stdout) and exit; you must not print any other strings or characters to the terminal, or else you will fail the test.**

   Further, you must extract the file name from the file path specified in the command line (e.g., extract "`make.html`" from "`/software/make/manual/make.html`"). Next, create a new file with the extracted file name in the current directory, and write the received file contents into that file. **If the downloaded file name does not match the file name provided in the command line, you will fail the test.**

   You should use the "`Content-Length`" value to figure when to stop receiving data from the web server and close the TCP connection. If the "`Content-Length`" field is not present in the response header, print the following error message to the terminal (stdout) and exit:

```
Error: could not download the requested file (file length unknown)
```

4. Some useful C library functions for parsing–"`strchr()`", "`strrchr()`", "`strtok()`", "`strstr()`".

5. Be very careful while using "`strlen()`" to calculate the length of a string. "`strlen()`" will stop counting as soon as it encounters the first NULL character. Instead use the return value from "`fread()`" / "`read()`" / "`fwrite()`" / "`write()`".

6. Your program should be able to download *any* type of file, not just HTML files. Test your code by downloading all the different files on the web site http://www.gnu.org/software/make/manual/.

   Use "`write()`" or "`fwrite()`" to write to the file in byte chunks. This is important to make your solution work for all different file formats, especially non-ASCII files such as pdf and image files. Functions like "`fprintf()`" might not work!

   To verify correctness, download the original file using "`wget`" and use the "`diff`" command to check whether the original file exactly matches the file downloaded by your program.

3 Grading

We will run your code through several test cases, each trying to download a file from a web server. For each test case, you will receive a percentage of the total points if your code generates the correct output, else a 0. We may also run the same test case multiple times to test the robustness of your code. To receive full points for a test case, your code must succeed in every run. Your final grade will be the sum total of the points obtained across all test cases.

4 Submission

You are required to submit **one** file "`http_client.c`" on Brightspace. **Do not submit a .zip file.**