Lab 1: HTTP Web Client (10 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 that you can use to download a web page like this:

$ `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 the first part of this lab, your task is to write a limited version of "`wget`", which we will call "`http_client`", that can download a ***single*** file. You will do all your implementation inside the file "`http_client.c`" inside the "`http_client`" directory. To build and run the code, do the following:

First, go inside the "`http_client`" directory,

$ `cd http_client`

Next, compile the code,

$ `make`

Finally, run the code,

$ `./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 (which will always be 80 for HTTP), and (3) the file path. The program will download the given file and save it in the current directory. So in the example above, it should produce "`make.html`" in the current directory. It should overwrite an existing file.

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!

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 in 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", the program should **print only the first line of the response to the terminal (stdout) and exit**.

   You will need to extract the file name from file path (for example, extract `make.html` from file path `/software/make/manual/make.html`), create a new file with extracted file name in the current directory, and write the received file content into that file.

   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. You 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, e.g., non-ASCII files such as pdf and image files. Functions like "`fprintf()`" might not work! **To verify correctness, also download the file using "`wget`" and make sure that it exactly matches the file downloaded by your program.**

3 Grading

We will run your submitted code to download a set of test files from one or more web servers. For each test file, you will receive a percentage of the total points if your code generates the desired output, otherwise a 0. Your final grade will be the sum total of the points received across all test files.

4 Socket Resources

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.

5 Submission

You are required to submit **one** file "`http_client.c`" on Brightspace.

Do not submit a compressed (e.g., .zip) file.