CMPE 156/L, Winter 2019

Programming Assignment 1
Due: Wednesday, Jan 16

The purpose of this project is to develop the client side of a simple HTTP-based download application similar to `wget`. The user of the application can send requests with GET or HEAD methods to any public HTTP server and retrieve the corresponding document or just response headers as appropriate.

Requirements

- The client must accept as argument a valid IP for the server hosting the document and the URL of the document. The URL would have the the hostname (e.g. `www.example.com`), the port number (80), and the document itself (e.g. `index.html`). As an optional parameter the user can input -h after all other parameters to only get the response and not the documents. Example:

  `./myrequester 93.184.216.34 www.example.com:80/index.html`

  Which would be equivalent to:

  `wget 93.184.216.34:80/index.html --header="Host:  www.example.com"`

  The corresponding basic HTTP request (using C string notation for CR and LF) would be:

  ```
  GET /index.html HTTP/1.1\r\nHost: www.example.com\r\n\r\n
  ```

  Example with `-h`:

  `./myrequester 93.184.216.34 www.example.com index.html -h`

  Equivalent to:

  ```
  wget 93.184.216.34/index.html --header="Host:  www.example.com"
  --server-response --spider
  ```

  And:

  ```
  HEAD /index.html HTTP/1.1\r\nHost: www.example.com\r\n\r\n
  ```

- The client **must** output the document to the file `output.dat` unless the option `-h` is present, in which case there is no file output.
- If the client receives the option `-h`, it must output the complete server response onto `stdout` exactly as received. If the option is not present, then the client should output **nothing** on `stdout`.
- You are free to output whichever information/debug messages you want on `stderr`.
- The client must be robust and not crash if there are network errors, or if it received invalid parameters, it should instead close itself properly. An explanation of the error delivered on `stderr` is optional. It can, however, generate a partially completed (or even empty) `output.dat` file if it is interrupted while downloading the contents.
- The memory use of the client should not depend on the size of the file being downloaded, furthermore the client should be able to work with documents of any size, as long as the computer has enough free space.

What to submit?

You must submit all files in a single compressed tar file (with `tar.gz` extension). The files should include

1. A README file including your name, student ID, and a list of files in the submission with a brief description.
2. Documentation of your design in plain text or pdf. Do not include any Microsoft Word files. The documentation should describe how to use your application and internal design, as well as any shortcomings it might have.
3. A `Makefile` that can be used to build the client program and the required source. Source files should be in the `src` directory and the compiled program should be in the `bin` directory.
4. Organize the files into directories (`src`, `bin`, `doc`, etc.)

Grading

Each submission will be tested to make sure it works properly and can deal with errors. Grades are allocated using the following guidelines:

Basic Functionality:	50%
Dealing with Errors:	20%
Documentation:	20%
Style/Code structure, etc.:	10%

Note that 20% of the grade will be based on how well your code deals with errors. Good practices include checking all system calls for errors and avoiding unsafe situations such as a buffer overflow.

The files must be submitted before midnight on the due date above.

Honor Code

All the code must be developed independently. All the work must be your own.

Useful references

- HTTP requests: `https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html`
- HTTP responses: `https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html`
- Some websites for testing
  - `http://www.linuxhowtos.org/`
  - `http://scratchpads.eu/explore/sites-list`
  - `http://www.caaspp.org/`
  - `http://pudim.com.br/`