Statutory note

- This course is on a no plagiarism diet.
- All parties involved in plagiarism harakiri will be penalized to the maximum extent.
- `The moss detective agency` has agreed to conduct all investigations.
  https://theory.stanford.edu/~aiken/moss/
  Byomkesh, Sherlock and Hercule are on standby.

Context

This assignment is inspired by Flask — a python based web application framework.
more info: https://palletsprojects.com/p/flask/
https://flask.palletsprojects.com/en/2.2.x/
https://flask.palletsprojects.com/en/2.2.x/quickstart/#

Example #1

The following code block demonstrates a simple Flask example. It exposes the hello_world() function to the web which can be accessed at the ‘/’ URL path. Flask, by default, spins a server on localhost listening for requests on port 5000 (default port number, but configurable).

```
# Importing flask module in the project is mandatory
# An object of Flask class is our WSGI application.
from flask import Flask

# Flask constructor takes the name of
# current module (__name__) as argument.
app = Flask(__name__)

# The route() function of the Flask class is a decorator,
# which tells the application which URL should call
# the associated function.
@app.route('/')
# ‘/’ URL is bound to the hello_world() function.
def hello_world():
              return 'Hello World'

# main driver function
if __name__ == '__main__':

# run() method of Flask class runs the application
# on the local development server.
      app.run()
```

Example #2_

The following code block demonstrates how to deal with GET query parameters. The code snippet below is able to parse a URL similar to ***/hello?name=Ashwin***, and stores ***‘Ashwin’*** in the variable ***name***

```
from flask import Flask
app = Flask(__name__)

@app.route('/hello')
def hello_name():
```

```
          name = request.args.get('name')
        return 'Hello %s!' % name

  if __name__ == '__main__':
        app.run()
```

In this assignment, we will recreate some of Flask’s functionality of adding new functions and exposing them to the web via a URL using **libHTTP** — an open source HTTP library in C, of which the web server and HTTP parser is our interest.
LibHTTP: https://www.libhttp.org
LibHTTP download: https://github.com/lammertb/libhttp/archive/v1.8.zip
Download and extract source. Study examples in the examples dir, build and try them.

Part1

The tasks of the first part of the assignment are to spin up a simple web server using the libHTTP library, parse HTTP requests (using the libHTTP library), and then finally execute the function corresponding to each request.
Your program should be designed to accept the following inputs …

I. Port number on which the server accepts requests
II. Number of threads in the server thread pool to serve requests

You are expected to submit 4 files to implement the above mentioned functionality:

1. **functions.h -** This file will contain the list of all URLs to be exposed and will associate each URL to an integer which will act as an index to access the actual function associated with the URL.
   Example listing:
   ```
   /           0
   /square     1
   /cube       2
   /hello      3
   /prime      4
   /pingpong   5
   ```
2. **functions.c -** This file will contain implementations of all the functions which are to be exposed on the Web.
3. **functionslist.h -** This file will contain a function pointer array which will place each function implemented in the file **functions.c** at the index specified in the file **functions.h**
4. **cflask.c -** This file will contain code to instantiate the web server using the libHTTP library and will also code to parse the URL and parameters (if any) of the incoming request, and code to call the actual requested functionality.

Note: All request URLs will only be of length 1 for this part of the assignment. (e.g.,: `http://127.0.0.1:8080/square`), and will not contain any query parameters.

Part2

In the second part of the assignment, extend the implementation to handle arguments passed in the URL as either query parameters in a GET request or inside a request body of a POST request. You can choose any of the above techniques to handle HTTP arguments.
HTTP requests will require careful parsing to extract the arguments from the request and then be supplied to the function corresponding to the HTTP request.

E.g., **`http://127.0.0.1:8080/square?num=3`**
The URL request beyond '?' specifies the argument variable **num** and its value is **3**. The corresponding function already expects this format.

Part3

In the third part of the assignment we will increase the complexity of parsing request URLs by increasing the length of the request URLs to two.
An example of a request URL of length two along with query parameters is …
`http://127.0.0.1:8080/arithmetic/square?num=3`

A corresponding entry in the **functions.h** file for the above example as follows

```
/arithmetic/square        x  (x is the function id for this
function)
```

Part4 (optional)

Build an in-memory representation (a search tree, or a hash-based store) to store the URL to function ID mappings … instead of reading this information from a file on each HTTP request.

Part5

(a) The sample list of URLs are to be used for testing your implementation …

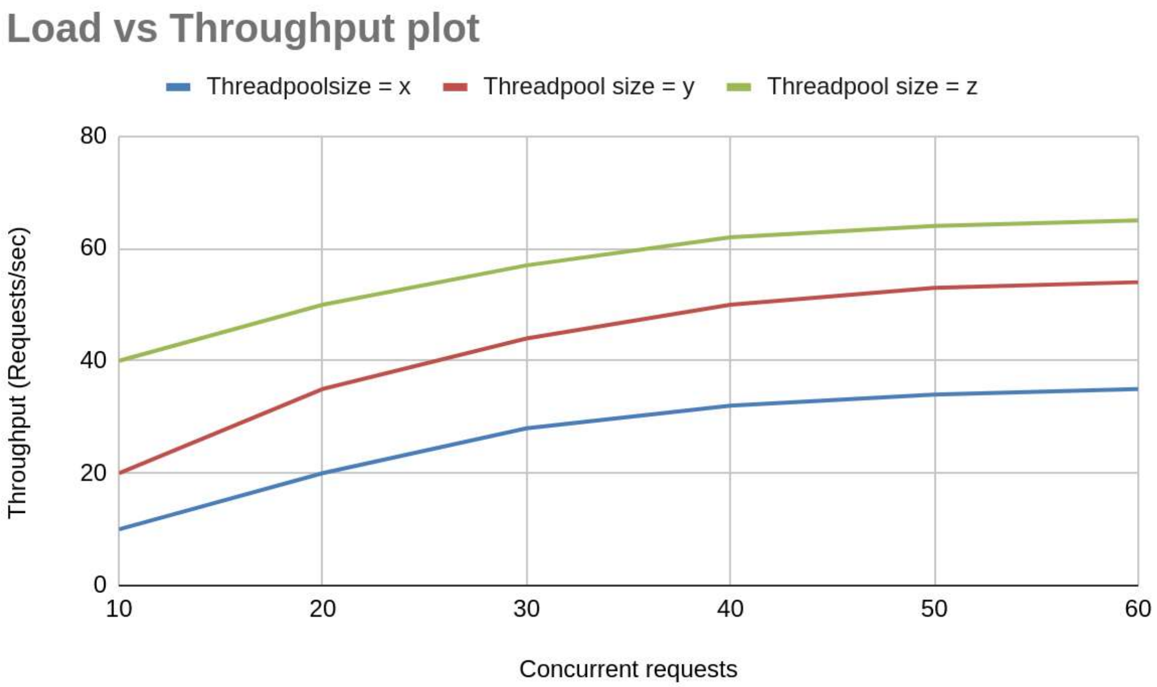
1. **`/`**
   This just returns a hello world message back to the user.
   This URL will not have any arguments.
2. **`/square`**
   This URL requests computing the square of a number which can be passed along with the URL as a query parameter like **/square?num=3** (return the square of 3). If no query parameter is present then this function should return the value 1.
3. **`/cube`**
   This URL requests computing the cube of a number which can be passed along with the URL as a query parameter like **/cube?num=7** (return the cube of 7). If no query parameter is present then this function should return the value 1.
4. **`/helloworld`**
   This URL maps to a function that takes a string argument as a query parameter, and returns the same string prefixed by hello back to the user. For example, **/helloworld?str=Ashwin,** returns back **"Hello, Ashwin".** In case of no query parameter, it simply returns **"Hello"** back to the user.
5. **`/pingpong`**
   This URL maps to a function that takes a string argument as a query parameter, and returns the same string back to the user. For example, **/pingpong?str=cs695,** returns back **"cs695".** In case of no query parameter, it simply returns "**PingPong"** back to the user.
6. **`/arithmetic/prime`**
   This URL maps to a function that takes a number as a query parameter and returns **'True'** to the user if the number is a prime and **'False'** if the number is not prime. In case of no query parameter, the function returns **'False'** back to the user.
7. **`/arithmetic/fibonacci`**
   This URL maps to a function that takes a number **k** as a query parameter and returns **the kth fibonacci number** to the user. In case of no query parameter, the function returns **1** back to the user.

(b)

Perform a load test of the server+function(s).
Pick a function (URL request) from the list above and increase the request rate and measure the average response time and throughput (#requests/second that were

served).

- Plot these metrics as a function of Request rate on a graph for at least three different settings of the thread pool size.
  Make sure the plots have x and y axes labels and units.
  Hypothesis: throughput increases and then flattens at a certain load value.
- Repeat the above load test for at least 3 different functions of your choice and summarize your observations in a README file.



You can use the tool **Apache Benchmark** to load test the cflask implementation. More info:
https://www.techrepublic.com/article/load-test-website-apache-bench/

To change load, fix total requests to a high value (e.g., 100000 requests or more) and change the concurrency factor (the -c option).

Reporting of experiments
Experiment with **ab** as the default load generator and additionally, use any other load-generators that you know of or are familiar with or want to explore.
In any and all of the above cases the following is required for experimentation (and its reporting).

I. aim/purpose of the experiment
II. setup and execution details, metrics and independent parameter
III. hypothesis/expectation
IV. observations from the data/plots
V. explanation of behavior and inferences

... repeat above for all individual experiments and for reasoning about a collection of experiments.

Submission guidelines

Add all files in a directory with the following format …
/<rollno-cs695-a1>
    |_ functionslist.h
    |_ functions.h

```
      |_ functions.c
      |_ cflask.c
      |_ README (describe instructions to execute Part 5a and Part 5b and any other
specifics).
      |                attach additional files if required for scripts for generation load etc.
      |_ plots
           |_ loadtest1.data      (load test output data and command used from the client)
           |_ loadtest1.jpg       (the image showing the graph)
           |_ loadtest2.data
           |_ loadtest2.jpg
           |_ loadtest3.data
           |_ loadtest3.jpg
```

Upload the zipped file `<rollno-cs695-a1>.tar.gz` of the submission folder on Moodle.

Submission deadline: 18th Jan 2023, Wednesday, 5 pm. (no extensions)