# Programming Assignment

It is required that you use socket programming and multi-threading/multi-processing techniques to make the following assignment which is divided into multiple parts

1) Multi-threaded web server:

Your web server should accept incoming connection requests. It should then look for the GET request and pick out the name of the requested file. If the request is POST then it sends OK message and wait for the uploaded file from the client. Note that a GET request from a real **WWW** client may have several lines of optional information following the GET. These optional lines, though, will be terminated by a blank line (i.e., a line containing zero or more spaces, terminated by a '\r\n' (carriage return then line feed characters). Your server should first print out the received command as well as any optional lines following it (and preceding the empty line).
The server should then respond with the line, this is a very simple version of the real HTTP reply message:

```
HTTP/1.0 200 OK\r\n
then in case of GET command only:
{data, data, ...., data}
\r\n
```

waiting for new requests from the same client. If the document is not found (in case of GET), the server should respond with(as would a real http server) :

```
HTTP/1.0 404 Not Found\r\n
```

**Server Side Pseudo Code**

**while** true: **do**

- Listen for connections
- Accept new connection from incoming client and delegate it to worker
  thread/process
- Parse HTTP/1.0 request and determine the command (GET or POST)
- Determine if target file exists (in case of GET) and return error otherwise

```
    - Transmit contents of the file (reads from the file and
    writes on the socket) (in case of
       GET)
    - Close the connection
```

**end while**

*notes: you are required to handle GET, POST, and status 200, status 404, HTML files, TXT files, and png files. Which would you choose for your implementation multi-threaded or multi-process? Justify.*

**2) HTTP Web Client**

Your web client must read and parse a series of commands from an input file. For this assignment, only the GET and POST commands are required to be handled. The commands syntax should be as follows:

```
GET file-name host-name (port-number)
POST file-name host-name (port-number)
```

Note that the port number is optional. If it is not specified, use the default HTTP port
number, 80. In response to the specified operation (GET or POST), the client must open
a connection to an HTTP server on the specified host listening on the specified (or default) port number. The receiver must display the file and then store it in the local directory (i.e., the directory from which the client or server program was run). The client should shut down when reaching the end of the file.

**Client Side Pseudocode**

```
while more operation exists do
   Create a TCP connection with the server
   Wait for permission from the server
   Send next requests to the server
   Receives data from the server (in case of GET) or sends
   data (in case of POST)
   Close the connection
end while
```

*note: your client program should use the reliable stream protocol and the internet domain protocols*

### 3) HTTP 1.1

You are required to add simple HTTP/1.1 support to your web server, consisting of persistent connections and pipelining of client requests to your web browser. You will also need to add some heuristic to your webserver to determine when it will close a "persistent" connection. That is after the results of a single request are returned (e.g., index.html), the server should
by default leave the connection open for some period of time, allowing the client to reuse that connection to make subsequent requests. This timeout needs to be configured in the server and ideally should be dynamic based on the number of other active connections the server is currently supporting. That is, if the server is idle, it can afford to leave the connection open for a relatively long period of time. If the server is busy, it may not be able to afford to have an idle connection sitting around (consuming kernel/thread resources) for very long.

### 4) bonus

Add a simple in place client side caching functionality to your client. You do not need to implement any replacement or validation policies. Your implementation, however, will need to be able to write responses to the disk (i.e., the cache) and fetch them from the disk when you get a cache hit. For this, you need to implement some internal data structure in your client to keep track of which objects are cached and where they are on the disk. You can keep this data structure in the main memory; there is no need to make it persist across shutdowns.

- **Policies**

Teams of three

Submit with any programming language (C/Java/Python).

Your code must support bash arguments.

If you are using C there would be some differences if you used Linux based distributions or Microsoft windows os