Homework 4

Introduction to Networks and Their Applications
Due: April 11th, 2017

You are to develop a simple web server in Python. The web server must be started with a two arguments specifying the IP address of the interface on which the server will bind to and the port number on which the server will run. The server should handle http requests such as (assuming the server is running on port 8000):

- http://127.0.0.1:8000/index.html
- http://127.0.0.1:8000/cars/ford.html

The server will store html static content in a local directory called **static**. Accordingly, the above URLs will be resolved to the following files on the server:

URL	file location
http://127.0.0.1:8000/index.html	static/index.html
http://127.0.0.1:8000/cars/ford.html	static/cars/ford.html

The server should implement the following logic:

- **Connection Setup:** Create a socket to wait for incoming connections. The server should be able to listen to 5 pending connections. Upon receiving a connection, you should accept it and process the incoming http request as discussed in the next step.
- **Multiple Connections:** You should support multiple connections and multiple between them using select.
- **HTTP Request:** Your server should only handle HTTP GET requests. A complete description of how an http request is formatted may be found in RFC 2616. For the purposes of this assignment it is safe to assume the following restricted format:

```
GET SP Request-URI SP HTTP/1.1 CRLF
(HTTPKey:HTTPValue CRLF)*
CRLF
```

In the above, the following notation is used: the text in blue will appear as it is in the header packet. SP stands for space and CRLF for RequestURI is the request URI. The HTTPKey:HTTPValue are header fields that may appear. The ()* indicates that there could be zero or more header fields. Your server should read these fields and make sure the header is formatted correct but it can safely ignore them (i.e., you do not need to take any actions on them) The server must make sure that the incoming request is consistent with the above format. It is safe to assume that the Request?URI will not exceed 255 bytes. This functionality should be **process_http_header(socket)** that returns the URI that is requested in the case when the request is correctly formatted or None otherwise.

- **Responding to an HTTP request:** There are three possible outcomes when handling an HTTP request:

  1. Bad Request: the HTTP request was invalid (i.e., it does not copy with the above format). In this case your server should respond with a Bad Request reply.
  2. Not Found: if the HTTP request is valid, then your program the process_http_header should store the RequestURI into the memory location pointed by uri variable. The RequestURI must be translated into a file location as indicated in the URL to file mapping table. After performing this translation, the server must check if the request file exists. If it does not, the server should respond with a Not Found error.
  3. No errors: in the case when the file exists, the server shall read the contents of the file and generate an HTTP reply.

- **Generating an HTTP reply:** In the case of finding the appropriate file to server on the web server, you need to send the content. However, you need to generate an appropriate HTTP header. In our case the minimal HTTP header is:

  ```
  HTTP/1.1 SP 200 SP OK CRLF
  Content-Type: text/html
  CRLF
  CRLF
  ```

  After sending the header, you can send the contents of the file.

As usual, you have to submit your homework through ICON. The following resources may be useful in solving the homework: HTTP RFC (`https://tools.ietf.org/html/rfc2616`).