Faculty of Engineering and Architecture

American University of Beirut

Fall 2010-2011

**EECE 450 – Computer Networks**

**Term Project**

***http web server***

**Objective**

The objective of the project is to design and implement an HTTP web server; it should accept HTTP requests from web clients and respond back with the corresponding HTTP responses.

**Requirements**

1. The web server must conform to the HTTP 1.1 specifications in RFC 2616: it must support only the GET method and should be able to properly respond to any (valid or invalid) HTTP request. The status codes to be supported are:
   1. 200 [OK]: to respond to a valid GET request
   2. 304 [Not Modified]: to respond to a conditional GET request asking for a file not modified since a specified time
   3. 400 [Bad Request]: to respond to a malformed request (error in syntax)
   4. 401 [Unauthorized]: to respond to a request requiring user authentication
   5. 404 [Not Found]: to respond to a GET request asking for a non-existing file

Refer to RFC 2616 for further explanation of the status codes. You can get from:

* [www.w3.org/Protocols/rfc2616/rfc2616.html] or
* [www.faqs.org/rfcs/rfc2616.html]

1. The web server must serve all clients’ requests for *dynamic* pages that include HTML markup, images, and links.
2. The web server should be able to run continuously (with no runtime errors) to serve concurrent and consequent HTTP GET requests.
3. In order to serve simultaneous requests and reduce the response time for each web client, the web server should be implemented as a single, multi-threaded process. The example posted on Moodle is a good starting point for creating client connections in threads.
4. The screen output of the web server should provide runtime information about the status of the server (creating socket, listening for requests, etc), as well as what port it is listening on.
5. All client requests should be logged to a text file and to screen. Each log entry must contain the following items:
   1. The date and time of the request
   2. The client’s IP address
   3. The browser type (if possible to determine)
   4. The URI (Uniform Resource Identifier) in the request header
   5. The web server’s response status code

In the log file, each entry must be a single line and the items must be separated by a single [tab] character. On the screen, you may choose to format the items in any way as long as they are clearly readable (since log entries may fill more than one line on console applications for example).

1. The web server must start up with a pre-defined home directory, specified either in a file or as a command-line argument, but not hard-coded. For example, if the home directory is defined to be **c:\web**, then the web server must, upon a URI reference to **/page.html** in the client’s request, send the file **c:\web\page.html** to the client, and another reference to **/images/picture.jpg** must be served with the file **c:\web\images\picture.jpg**. If a URI reference to **/** is made in the request, the web server should send the file **c:\web\index.html** to the client.
2. The home directory of the web server should include a protected file that the client cannot retrieve unless it provides the correct credentials for authentication, which could be stored in a simple text file on the server.
3. The port number on which the web server is listening must be specified either in a file or as a command-line argument, but not hard-coded.
4. The web server can be a simple console application (no need for a graphical interface).
5. For dynamic web pages, only very simple ones are required. They may be specified by the client user as follows (refer to the Power Point (ppt) document “Overview of HTML”) :
   1. If the URI specifies /basic, then output a basic page like the one on Slide 2 of the ppt document.
   2. If the URI specifies /[category]?links=m, then output a page like the one on Slide 5 of the ppt document. Here, [category] is a topic, like food, games, movies, etc. the value of m is an integer in the range of 1 to 10. In the server implementation, have a finite number of categories. If the client requests a non-supported category, then return an appropriate error code.
   3. If the URI specifies /email?to=x@y.z, then output a page like the one on Slide 7 of the ppt document. Here, x@y.z is an e-mail that is specified by the client.
   4. If the URI specifies /images?number=n, then output a page like the one on Slide 11 of the ppt document. Here, the client specifies the desired number of images. The server can choose any images.
   5. If the URI specifies /mix?paragraphs=m,links=m,images=n, then output a page that has a mix of paragraphs, links, and images.

**Implementation**

You can implement the web server using any programming language you like under any operating system. Make sure your code is neat, well structured and well documented (properly commented for ease of reading and understanding).