**PROGRAMMING ASSIGNMENT – 1**

**Due: 09/10 11:59 pm**

**NOTE: This assignment is NOT a group assignment, but an individual assignment.**

**Objective:**

To create a HTTP-based web server that handles various requests from users.

**Background:**

What is a web-server?

A web server is a program that receives requests from a client and sends the processed result back to the client as a response to the command. A client is usually an user trying to access the server using a web browser (chrome, firefox).

The HTTP request consists of three substrings – request method, request URL, and request version. All three parts should be separated by one or more white spaces.

The request method should be capital letters like “GET”, “HEAD”, and “POST”.

The request URL is a set of words which are case insensitive and separated by “/” and the server should treat the URL as a relative path of the current document root directory.

The request version follows the rule like “HTTP/x,y” where x and y are numbers.

Here is an example to make things more clear:

If you type an URL like http://www.w3.org/Protocols/rfc1945/rfc1945 in a web browser, the web browser will send an HTTP GET command to the server http://www.w3.org/ after establishing a TCP connection with port 80 on the server. In this scenario, the format for the HTTP GET command received at the web server is as follows:

```
GET /Protocols/rfc1945/rfc1945 HTTP/1.1
```

Based on the rules, we can identify three substrings like the following:

Request Method: GET

Request URI: /Protocols/rfc1945/rfc1945

Request Version: HTTP/1.1

**Deliverables:**

In this assignment, the Web server will have a document root which would contain files with the extension ".html", ".txt", ".gif" and ".png". The web-server when receives a HTTP request from a client (from a web-browser such as chrome) for a particular file, it should open the file from this document root and then send the file back to the client with proper header information. Header information is the required headers which should be sent along with the file so the web-browser would understand that the HTTP request was successful and the result is being received. In all cases, the names of files in the Request URL should be interpreted in the context of the current document root. If the server interprets the request (HTTP GET) and the requested file exists on the specified location, the server needs to send both Content-Type and Content-Length follows by an appropriate status code. The file content should be separated by a carriage-return-line-feed ("\n") from the header. For example, when the client requests an existing ".html" file on the server, the server replies with a 200 status code.

The Status Code "200" indicates that the requested file exists and the server is going to send the processed data (the requested file) back to the client. The string next to Status Code "200" conveys the information that the requested document follows in this reply. Usually, many Web servers use "Document Follows" and "Ok" for that field.

```
HTTP/1.1 200 Document Follows

Content-Type: <>

Content-Length: <>

<file contents>
```

**Default Page:**

Also, if the Request URL is the directory itself, the web server tries to find a default web page such as "index.html" or "index.htm" on the requested directory. What this means is that when no file is requested in the URL and just the directory is requested in the URL, then a default web-page should be displayed. This should be named either "index.html" or "index.htm" and it should be present in the document root. The default web page and document root directory should be searched in the server configuration file which is shown in later in this document.

**Handling Multiple Connections:**

When the client receives a web page that contains a lot of embedded pictures, it repeatedly requests an HTTP GET message for each object to the server. In such a case the server should be capable of serving multiple request at same point of time. This can be done using PThread.

**Pipelining:**

A client that supports persistent connections may "pipeline" its request (i.e., send multiple requests without waiting for each response). A server must send its responses to those requests in the same order that the requests were received. So, if there is "Connection: keep-alive" header in the request then a timer has to start. If within the timer period any other request from the same client is not received by the web-server then the socket will be closed for that client. This timer will have a value of 10 seconds. If there is no "Connection: keep-alive" in the request header then the socket will be closed immediately. For example if the request is something like the following:

```
GET /index.html HTTP/1.1
Host: localhost
Connection: keep-alive
```

The response for this request should something like the following:

```
HTTP/1.1 200 OK
Content-Type: <>
Content-Length: <>
Connection: keep-alive
<file contents>
```

Also, in this case the web-server after receiving this request triggers a timer of 10 seconds. If any other request from the same client is received by the server then the timer is reset back to 10 seconds otherwise the socket for this client will be closed if no other request arrives within that 10 second period. If the request doesn't contain "Connection: keep-alive" in its header then the socket will be closed immediately after the web-server completes processing the request.

Example Scenario:

Consider a request is received by the server with the keep-alive header. At this point a timer is triggered. Now if another request from the same client is received after 2 seconds of the first request, then the timer will now be reset to 10 seconds. If no other request is received by the web-server within the 10 second timeout period, the connection is closed by the web-server.

**Error Handling:**

When the HTTP request results in an error then the web-server should respond to the client with an error code. In this assignment, the following status codes should be used:

1. The response Status Code "400" represents that the request message is in a wrong format and hence it cannot be responded by the server. It sends the client one of following messages depending on the reason:

| |
|---|
| HTTP/1.1 400 Bad Request: Invalid Method: <requested method> |
| HTTP/1.1 400 Bad Request: Invalid URI: <requested URI> |
| HTTP/1.1 400 Bad Request: Invalid HTTP-Version: <requested HTTP version> |

2. The response Status Code "404" informs the client that the requested URL doesn't exist (file does not exists in the document root). It results in the following message.

| |
|---|
| HTTP/1.1 404 Not Found: /csc573/img/logo.jpg |

3. The response Status Code "501" represents that the requested file type is not supported by the server and thus it cannot be delivered to the client. It results in the following message.

| |
|---|
| HTTP/1.1 501 Not Implemented: /csc573/video/lecture1.mpg |

4. All the other error messages can be treated as the "500 Internet Server Error" indicating that the server experiences unexpected system errors; it results in the following messages.

| |
|---|
| HTTP/1.1 500 Internal Server Error: cannot allocate memory |

These messages in the exact format as shown above should be sent back to the client if any of the above error occurs.

**Configuration File:**

As mentioned earlier, the Web server configuration file named “ws.conf” stores the initial parameters of the server which must be read by the server when the server starts running. Also, the Content-Type associated with a filename extension is stored in this file. The example of “ws.conf” file is as follows:

```
#service port number
Listen 8004
#document root
DocumentRoot "/home/sha2/www"
#default web page
DirectoryIndex index.html index.htm index.ws
#Content-Type which the server handles
.html    text/html
.txt     text/plain
.png     image/png
.gif     image/gif
```

This file should be used to modify all configuration parameters of the web server. The server should read from this file to get all the required configuration for its operation.

**Submission Requirements:**

1. Please submit the code for web server, ws.conf file, README and the document root folder in the format your_identity_key_PA1.tar.gz after you tar-gzip the files.

2. Include short comments in your web server code and explain its proper usage in the README file.
3. The web server should handle error code 404, the three 400 Bad request errors, 501 Not Implemented error and error 500 along with the 200 success code.
4. The code should serve the four file formats ".html",".txt" ,".png" and ".gif" and the client should receive these files when requested.
5. The select module should be used to handle multiple request from a client.
6. Pipeline support should be present in the code with a 10 second timer.
7. Usage of any modules for HTTP server is not accepted.

**Testing your Web server:**

In order to check the correctness of the server, you have to test your server with any of commercial Web browsers such as FireFox or Internet Explorer. It is very helpful if you test the server after making a sample web page that contains some pictures and text.

**Testing pipelining on the Webserver:**

***Command to be typed on terminal:***

```
(echo -en "GET /index.html HTTP/1.1\n Host: localhost \nConnection: keep-alive\n\nGET
/index.html HTTP/1.1\nHost: localhost\n\n"; sleep 10) | telnet localhost 80
```

***Idle Response:***

```
Trying 127.0.0.1...
Connected to localhost.lan.
Escape character is '^]'.
HTTP/1.1 200 OK
Date: Sun, 27 Oct 2013 17:51:58 GMT
Content-Length: 62
Connection: Keep-Alive
Content-Type: text/html

<html>
    <body>
        <h1>test</h1>
    </body>
</html>
HTTP/1.1 200 OK
Date: Sun, 27 Oct 2013 17:51:58 GMT
Content-Length: 62
Connection: Keep-Alive
Content-Type: text/html

<html>
    <body>
```

```
        <h1>test</h1>
    </body>
</html>
```

**Helpful Links:**

1. https://www.youtube.com/watch?v=eVYsIolL2gE : Basics of socket programming.
2. http://www.csc.villanova.edu/~mdamian/sockets/echoC.htm : Echo server-client code with threads