Computer Networking MCIS 6163
Project 1
Simple Web Server & Client

Instructor: **Sajib Datta**

"What I cannot create, I do not understand." ***Richard P Feynman***

Objectives

(A) To understand Client-Server communication via sockets

(B) To gain exposure to the basic operations of a Web Server and Client

(C) To explore basic structures of HTTP messages

Due Date

Available on BlackBoard[1]

Project Description

(A) You will be developing a multi-threaded **Web server** which interacts with any standard Web Clients (You may use any web browser of your choice to test the functionality however you should also submit the a client as given in (B) below). The Web server and Web client communicate using a text-based protocol called HTTP (Hypertext Transfer Protocol)

(B) Build a single threaded **Web Client** on your own which interacts with your Web Server, and downloads a file from the server

(C) **Display the essential connection parameters** of connection for both the **Web client** (*on the server side*) and for the **Web Server** (*on the client side*)

[1] All Submissions should be completed through BlackBoard

Specification

Specifications - Server

The server being multi-threaded, should be able to handle multiple requests concurrently. The main thread (server), listens to a specified port like the standard port for HTTP (8080). Upon receiving a HTTP request, the server sets up a TCP connection to the requesting client and serves the request in a separate thread using a new port. After sending the response back to the client, it closes the connection. For this exercise you may choose any browser of your choice for testing. (Internet Explorer or FireFox or Chrome)[2]. *However you* ***should*** *submit a client program as per the the Section Specifications - Client.*

The server is assumed to work with **HTTP GET** messages. If the requested file exists at the server, it responds with a *"HTTP/1.1 200 OK"* together with the requested page to the client, otherwise it sends a corresponding error message, *"HTTP/1.1 404 Not Found"* or *"HTTP/1.1 400 Bad Request"*.

- If running the server program using command line, the syntax should be

$$server\_code\_name < port\_number >$$

- You must test your Web server implementation on your local machine using a Web browser. You need to specify the used port number within the URL. If omitting the port number portion, i.e., 8080, the browser should use the default port 8080. To cite an example,

$$http://localhost:8080/index.html$$

- You should display/log the request and header lines of request messages on the server for the purpose of debugging.

Specifications - Client

- The client should be able to initiate a connection to the server, via a socket and request ***any page*** on the server. Upon receipt of the response message from the server, the client extracts and displays/logs the message status[3], and then retrieves the page content from the corresponding message body.
- The requested file need not be HTML, even a text file would suffice [4].
- You may execute the client program using command line, with the following syntax,

[2] Caveat: Some of the browsers need some additional setting changes for enabling complete functionality

[3] *'HTTP 200 OK'* or *'404 Bad Request'*

[4] But the format of the request should strictly be HTTP as discussed in class

$$client\_code < server\_IPaddress >< port\_no >< requested\_file\_name >$$

(a) **Server_IPaddress**: The IP address or name of the Web server, e.g., *127.0.0.1* or *localhost* for the server running on the local machine.

(b) **port_no**: The port on which the server is listening to contnections from clients. If the port number is not entered, the default port 8080 should be used.

(c) **requested_file_name**: The name of the requested file, which may include the path to the file.

Specifications - Connection Parameters

You should be able to extract the following information from the connection objects,

(a) Calculate and Display RTT for the client request[5].

(b) Print the relevant ***server details*** on ***client side***. The examples could be *Host Name of the server, socket family, socket type, protocol, timeout and get peer name* [6].

(c) Print the relevant ***client details*** on ***server side***. The examples could be *Host Name of the client, socket family, socket type, protocol, timeout and get peer name* [7].

Notes

(a) This is an individual project.

(b) You can use the programming language of your choice [8].

(c) You may use the skeleton code for the server provided in the textbook's companion website for reference. You may also want to refer to the textbook, chapter 2, section 2.2.3, for more details on HTTP message format and section 2.7, for socket programming.

(d) The source codes should be well documented to make it easier for the GRADER to follow.

[5] Refer Slide 2 - 25 of class lecture
[6] Print a minimum of 4 out of 6
[7] Print a minimum of 4 out of 6
[8] You may get more help with Java or Python. Our best choice for you will be Python

Submission Guidelines

- Submit a single zipped file with the naming convention,

$$< your\_\text{SAU}\_id > \_ < your\_name > .zip$$

- Your submission should have the following items to be considered for evaluation,
  (a) Source codes of the Web server and client
  (b) Any additional files required to run your codes
  (c) **–Very Important–** ***readme.txt*** file with instructions on how to compile and run your codes. You must mention the IDE as well as any packages that are required to run the codes.
  (d) **–Very Important–** Provide ample amount of comments in the code to make it more readable and sustainable.
- Do **NOT** include any runnable executable (binary) program.
- Make sure your **name** and your **SAU ID** are also listed in the *readme* file and in comments at the beginning of your source files.
- Make sure that submissions of the zipped file is through *BlackBoard*[9].
- No Late submission will be accepted.

Additional Requirements/Instructions

(a) Please email your Instructor for any doubts and clarifications regarding the project 1.

(b) Complete documentation and instructions for running the codes are recommended.

(c) If you are using any code from some external source or book, you MUST mention it explicitly in the codes as well as the *readme* file. Otherwise, it will be considered plagiarism and your project will not be evaluated.

[9] Please strictly follow the naming convention of the zipped file

(d) You can discuss with other classmates on steps/algorithms to implement the project. ***However, the source codes must be written by yourself***.

Grading Rubric **(25 points)**

(i) The server works correctly with requests from a Web browser *(3.5 points)*

(ii) The server can serve multiple requests at the same time (multithreaded implementation) *(4 points)*

(iii) The client sends/receives messages to/from the server correctly *(5 points)*

(iv) The client extracts the status and content of messages from the server correctly *(4 points)*

(v) Extracting and displaying connection parameters *(1.5 points)*

(vi) Calculate and Display Round Trip Time (RTT). *(2.5 points)*

(vii) Proper closing of the ports with exception handling. *(2 points)*

(viii) Display/log of proper messages on the server as well as on the client. *(1.5 points)*

(ix) Code documentation and Readme file. *(1 points)*

Wish you all a good luck