Project 1 – CSE 4344 (Spring 2018)

Building a Simple Web Client and a Multithreaded Web Server

Objectives

- To understand client-server communication via sockets.
- To gain exposure to the basic operations of a Web server and client.
- To explore basic structures of HTTP messages.

Due: Tuesday, March 6, 2018 11:59pm

Project Description

In this project, you will be developing a multithreaded Web server and a simple web client. The Web server and Web client communicate using a text-based protocol called HTTP (Hypertext Transfer Protocol).

Requirements for the Web server

- The server can handle multiple requests concurrently. This means the implementation is multithreaded. In the main thread, the server listens to a specified port, e.g., 8080. Upon receiving an HTTP request, the server sets up a TCP connection to the requesting client and serves the request in a separate thread. After sending the response back to the client, it closes the connection.
- The server is assumed to work with HTTP GET messages. If the requested file exists, the server responds with "HTTP/1.1 200 OK" together with the requested page to the client, otherwise it sends corresponding error message, e.g., "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>]

  where the optional <port_number> is the port on which the server is listening to connections from clients. If the port number is not entered, the default port 8080 is used.
- You can test your Web server implementation on your local machine using a Web browser, e.g., Internet Explorer, Firefox, or Chrome. You need to specify the used port number within the URL, for example, **http://localhost:8080/index.htm**

  If omitting the port number portion, i.e., 8080, the browser will use the default port 80.
  - The server should response with a default page when users do not enter a specific page in the URL, for example, http://localhost:8080/ . It should also work when the request includes a path to the requested file, for example, http://localhost:8080/path/to/file/example.htm
- You should display/log the request and header lines of request messages on the server for the purpose of debugging.

Requirements for the simple Web client

- The client can connect to the server via a socket and to request a page on the server.
- Upon receipt of the response message from the server, the client extracts and displays/logs the message status, and then retrieves the page content from the message body.
- If running the client program using command line, the syntax should be:

  client_code_name <server_IPaddress/name> [<port_number>] [<requested_file_name>]

  where the <server_IPaddress/name> is 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. The optional <port_number> is the port on which the server is listening to connections from clients. If the port number is not entered, the default port 8080 is used. The optional <requested_file_name> is the name of the requested file, which may include the path to the file. If the file name is not entered, the default file "index.htm" is used.

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.

(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 peer name.

(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 peer name.

Notes:

- This is an individual project.
- You can use the programming language of your choice. (You may get more help with Java or Python.)
- 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.
- **Code documentation**: The source codes should be well documented to make it easier for the TA to follow.

<u>Submission Guidelines</u>

- Submit a single zipped file with the naming convention **<your_UTA_id>_<your_name>.zip** which consists of:
  - Source codes of the Web server and client,
  - Any additional files required to run your codes,

- **Readme file:** It should contain 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.

- Do NOT include any runnable executable (binary) program.
- Make sure your name and your UTA ID are also listed in the readme file and in comments at the beginning of your source files.
- ***Make sure your project submissions are through BlackBoard.***
- Late submission will be accepted with a 10-point deduction for each extra day.

Additional Requirements/Instructions

- Complete documentation and instructions for running the codes are recommended, otherwise you may be asked to come give the TA a demo if he is not able to run your programs from the instructions provided.
- 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.
- You can discuss with other classmates on steps/algorithms to implement the project. However, the source codes must be written yourself.

Grading (100 points)

- The server works correctly with requests from a Web browser and command line (20 points)
- The server can serve multiple requests at the same time (multithreaded implementation) (20 points)
- The client sends/receives messages to/from the server correctly (20 points)
- The client extracts the status and content of messages from the server correctly (15 points)
- Display connection parameters from the connection objects (5 points)
- Display/log of proper messages on the server (5 points)
- Display/log of proper messages on the client (5 points)
- Code documentation (5 points)
- Readme file (5 points)

Late submission: 10-point deduction for each extra day.