CSE 5306 **Lab # 1** **Spring – 2019**

Sockets and Thread Management

Objectives:

1. Understanding two communication protocols for distributed processing, HyperText Transport Protocol - HTTP, and Sockets.
2. Exposure to multithreading.
3. Using an Integrated Development Environment (IDE)

Due: February 27th, *before* 11:59 pm via Blackboard

Project Specification:

These will be individual projects. You may write the program in any language that is supported under any Integrated Development Environment (IDE). Keep in mind that more help may be available to you in some languages than in others. Furthermore, available controls, objects, libraries etc. may make some of these tasks easier in one language than in another. Finally, because of the lack of restrictions on IDEs, you will have to have that IDE available to demo to the TA. For example, you might bring a laptop to demo the program. Socket programming is so universal that you can probably find major portions of this part of the program with searching on Google. Using code you find on the Internet is fine, but be sure to document the source in the writeup and in the program source code!

You will write a system consisting of a server and three client processes. Each client process will connect to the server over a socket connection and register a user name at the server. The server should be able to handle all three clients concurrently and display the names of the connected clients in real time. The server should be a passive component that acts only to facilitate message delivery.

Each client will allow a user to enter a brief text message. The user will enter a text message and indicate one of two delivery options:

1. Deliver the message to a single, designated client (1-to-1); or,
2. Deliver the message to all clients (1-to-N).

If the user selects option 1, the client should prompt the user to designate which client the message should be delivered to. How the user indicates which client should receive the message is left to the developer's discretion.

If the user selects option 2, the message should be delivered to all clients. Whether the client receives its own message is left to the developer's discretion.

When a client receives a message from a peer, the message should be printed to the screen and include the name of the originating peer, as well as whether the message was 1-to-1 or 1-to-N.

The server and the client should each be managed with a *simple* GUI. The GUI should provide a way kill the process without using the 'exit' button on the window. Messages exchanged between server and client should use HTTP formats and commands.

The HTTP tags must use, at minimum, `Host`, `User-Agent`, `Content-Type`, `Content-Length`, and `Date`. If you are polling the server, use `GET`. If you are sending data to the server, use `POST`.

The required actions are summarized as follows:

Client

The client will execute the following sequence of steps:

1. Connect to the server via a socket.
2. Provide the server with a unique user name. The user name may be:
    a. A string provided by the user; or,
    b. Some value associated with the process.
3. Proceed to send and receive messages until killed by the user.

Sending messages:

1. Accept a brief text message from a user.
2. Prompt the user to indicate the delivery preference. If delivering the message to a single client is indicated, the user should be prompted to specify which client.
3. Encode the message in HTTP and upload to the server.
4. Indicate that the message has been successfully sent to the server.

Receiving messages:

1. Wait for a message to be received from the server.
2. Parse the HTTP metadata and print the message to the GUI.
3. Show which client sent the message.
4. Indicate whether the message was 1-to-1 or 1-to-N.

Server

The server should support three concurrently connected clients and display a list of which clients are connected in real-time. The server will execute the following sequence of steps:

1. Startup and listen for incoming connections.
2. Print that a client has connected and fork a thread to handle that client.
3. Accept and forward messages received from users.
4. Print the **unparsed** HTTP message received from a client to the screen.
5. Begin at step 3 until connection is closed by the client.

The server must correctly handle an unexpected client disconnection without crashing. When a client disconnects from the server, the server GUI must indicate this to the user in real time. The server should print messages both received from, and sent to, the client in unparsed HTTP so that the grader can verify the format.

Your program must operate independently of a browser. Date/time on the messages should be encoded according to HTTP.

References:

1. http://www.w3.org/Protocols/ This is the standard.

2. http://www.jmarshall.com/easy/http/ HTTP Made Really Easy.

3. http://tangentsoft.net/wskfaq/ Winsock Programmer's FAQ

4. http://www.eggheadcafe.com/articles/20020323.asp VB.net Single thread Telnet client & server.

Submission Guidelines:

FAILURE TO FOLLOW ANY OF THESE DIRECTIONS WILL RESULT IN DEDUCTION OF SCORES.

Submit your assignment via the submission link on Blackboard. You should zip your source files and other necessary items like project definitions, classes, special controls, DLLs, etc. and your writeup into a single zip file. No other format other than zip will be accepted. The name of this file should be your **lastname_loginID.zip**. Example: If your name is John Doe and your login ID is jxd1234, your submission file name must be "Doe_jxd1234.zip".

Be sure that you include everything necessary to unzip this file on another machine and compile and run it. This might include forms, modules, classes, config. files, etc. DO NOT INCLUDE ANY RUNNABLE EXECUTABLE (binary) program. The first two lines of any file you submit must contain your name and student ID. Include it as comments if it is a code file.

You may resubmit the project at any time. Late submissions will be accepted at a penalty of 10 points per day. This penalty will apply regardless of whether you have other excuses. In other words, it may pay you to submit this project early. If the TA can not run your program based on the information in your writeup then he will email you to schedule a demo. The TA may optionally decide to require all students to demonstrate their labs. In that case we will announce it to the class. It is your responsibility to keep checking blackboard for announcements, if any.

If your program is not working by the deadline, send it anyway and review it with the TA for partial credit. Do not take a zero or excessive late penalties just because it isn't working yet. We will make an effort to grade you on the work you have done.

Writeup:

Your write-up should include instructions on how to compile and run your program. Ideally it should be complete enough that the TA can test your program without your being there. Your writeup should include any known bugs and limitations in your programs. If you made any assumptions you should document what you decided and why. This writeup can be in a docx or pdf format and should be submitted along with your code.

Grading:

Points – element

10 – Client Process works correctly
10 – Server Process works correctly
05 – Server shows HTTP message format in full
10 – HTTP message formats are valid
10 – Client provides user name to server
10 – Clients correctly uploads messages to server
10 – Server correctly sends messages to client(s)
10 – Client parses HTTP message received from server
10 – Client and server handle disconnections correctly
10 – Server displays connected clients in real time.
05 – Comments in code

Deductions for failing to follow directions:

-10 Late submission per day.
-05 Including absolute/ binary/ executable module in submission.
-02 Submitted file doesn't have student name and student Id in the first two lines.
-05 Submitted file has a name other than student's lastname_loginID.zip.
-05 Submission is not in zip format.
-05 Submitting a complete installation of the java virtual machine.
-10 Per instance of superfluous citation.
-20 Server and/or clients run exclusively from a command line.

To receive full credit for comments in the code you should have **brief** headers at the start of every module/ subroutine/ function explaining the inputs, outputs and function of the module. You should have a comment on every data item explaining what it is about. (Almost) every line of code should have a comment explaining what is going on. A comment such as /* Add 1 to counter */ will not be sufficient. The comment should explain what is being counted.

Important Note:

You may discuss the problem definition and tools with other students. You may discuss the lab requirements. You may discuss or share project designs. All coding work must be your own. You may use any book, WWW reference or other people's programs (but not those of other students in the class) as a reference as long as you cite that reference in the comments. If you use parts of other programs or code from web sites or books YOU MUST CITE THOSE REFERENCES. If we detect that portions of your program match portions of any other student's program it will be presumed that you have collaborated unless you both cite some other source for the code. You must not violate University of Texas at Arlington regulations, laws of the State of Texas or the United States, or professional ethics. Any violations, however small, will not be tolerated.

DO NOT POST YOUR CODE ON PUBLICLY ACCESSIBLE SECTIONS OF WEBSITES UNTIL AFTER THE DEADLINE. SHOULD YOU DO SO, THIS WILL BE CONSIDERED COLLUSION,

AND YOU WILL BE REFERRED TO THE OFFICE OF STUDENT CONDUCT AND RECEIVE A FAILING GRADE IN THE COURSE