CSE 5306 **Lab # 2** **Fall – 2018**

Centralized Mutual Exclusion

Objectives:

1. Understanding mutual exclusion.
2. Further experience with HTTP, threads, and sockets.
3. Using an Integrated Development Environment (IDE)

Due: Wednesday, October 31st, *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!

This lab is intended to be built on top of the program created for Lab 1. If Lab 1 was completed successfully, much of the baseline functionality of this program (i.e., server process, client process, timers, et cetera) should already be in place.

Terminology used in this lab is explained in Distributed Systems: Principles and Paradigms (2 ed), Tanenbaum and van Steen, **6.3.2**.

You will write a system consisting of a server (coordinator) 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.

Each client will generate a random integer and upload that integer to the server. The server will queue the client-generated integers in the order they were received. The server will then pop the first integer queued and pause all client-handling threads (e.g., sleep or otherwise wait) for the number of seconds equal to that integer.

When the server is finished waiting, it will reply to the client that originated that integer stating, "`Server waited <#> seconds for client <name>.`" The client will print this message to the user, as well as display the total time it spent waiting for the server to respond. The server will repeat this process until killed by the user.

The server and the clients 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`.

Centralized Mutual Exclusion

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.
    a. May be a string provided by the user; or,
    b. Some value associated with the process.
3. Generate a random integer between 3 and 10.
4. Upload that integer to the server.
5. Wait until response received from the server.
6. Parse the HTTP message and print response from the server in normal text.
7. Inform the user of the total time spent waiting for the server response.
8. Repeat at step 3 until the process is killed by the user.

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. Queue the integers received from clients.
4. Pop the queued integers in the order they were received from clients.
5. Print the current integer to the GUI and announce that it is waiting for that period of time.
6. Pause (sleep or otherwise wait) all client-handling threads for the number of seconds equal to that integer.
7. After waiting, return a message to client stating, "`Server waited <#> seconds for client <name>.`"
8. 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 disconnected, the user must be notified 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. Time on the messages should be encoded according to HTTP.

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 handles wait intervals
10 – Server queue operates correctly
10 – Server correctly handles wait intervals
05 – Client parses HTTP message received from server
05 – 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