THERE IS A ZERO TOLERANCE CHEATING POLICY

ANY HONOR CODE VIOLATION – NO MATTER HOW SLIGHT – WILL RESULT IN AN F IN THE COURSE AND REFERRAL TO THE OFFICE OF STUDENT CONDUCT

CSE 5306 **Lab # 1** **Summer 2020**

Objectives:

1. An introduction to sockets.
2. Exposure to thread management.

Due: July 12th, *before* 11:59pm via Canvas

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 available controls, objects, libraries, et cetera, 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 (e.g., you will demo the program on your own laptop).

All components should be managed with a *simple* GUI. The GUI should provide a way to kill the process without using the 'exit' button on the window.

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 username at the server. The server should be able to handle all three clients simultaneously and display the names of the connected clients in real time.

Two or more clients may not use the same username simultaneously. Should the server detect a concurrent conflict in username, the client's connection should be rejected, and the client's user should be prompted to input a different name.

Every ten seconds, the server will randomly select a connected client and send that client an integer between 3 and 9. Upon receiving the integer, the client will pause (e.g., sleep or otherwise suspend) the thread managing the connection to the server for a period equaling the value received from the server, in seconds. The client's GUI will maintain a decrementing countdown timer indicating when the thread will resume, as well as a button to skip the wait and resume the thread's operation immediately.

When the client thread is finished waiting, it will reply to the server with a message stating, "Client <name> waited <#> seconds for server." The server will display this message on its GUI. This sequence will be repeated until the components are manually terminated by the user.

Client

Startup:

1. Prompt the user to input a username.
2. Connect to the server over a socket and register the username.
    a. When the client is connected, the user should be notified of the active connection.
    b. If the provided username is already in use, the client should disconnect and prompt the user to input another username.
3. Proceed to accept commands received from the server until manually terminated by the user.

Receiving Pause Commands:

1. The client will wait for 'Pause' commands from the server.
2. Upon reception of a 'Pause' command, the client will suspend the thread managing the connection to the server for the indicated period.

ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE

3. The client will display a counter on its GUI which decrements in accordance with the number of seconds remaining until the thread resumes.
4. The thread will resume when either:
    a. The timer expires; or,
    b. The user manually resumes the thread by pressing a button on the GUI.
5. Return to **Receiving Pause Commands: Step 1** until manually killed by the user.

Server

The server should support three concurrently connected clients. The server should indicate which of those clients is presently connected on its GUI. The server will execute the following sequence of steps:

1. Startup and listen for incoming connections.
2. Print that a client has connected, and:
    a. If the client's username is available (e.g., not currently being used by another client), fork a thread to handle that client; or,
    b. If the username is in use, reject the connection from that client.
3. Every ten seconds, the server should randomly select a connected client, then:
    a. Randomly choose and integer between 3 and 9; and,
    b. Send that integer to the selected client.
4. The server will then wait for a reply from that client.
5. Upon receiving a reply from a client, the server will print the reply to its GUI.
6. Return to **Step 1** until manually killed by the user.

Notes:

- All three clients and the server may run on the same machine.
- 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 program must operate independently of a browser.

Citations:

You may use open source code found on the Internet in your program. When citing this code:

YOU MUST CITE THE EXACT URL TO THE CODE IN THE METHOD / FUNCTION / SUBROUTINE HEADER WHERE THE CODE IS UTILIZED.

Failure to cite the exact URL will result in a twenty (20) point deduction on the grade of your lab

A full list of your source URLs should be included in your writeup file. Including generic citations (for instance, simple listing "StackOverlow.com" without additional details) will result in a ten (10) point deduction in your lab grade, per instance.

Submission Guidelines:

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

ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE

CSE 5306 **Lab # 1** **Summer 2020**

Submit your assignment via the submission link on Canvas. You should zip your source files and other necessary items like project definitions, classes, special controls, DLLs, et cetera and your writeup into a single zip file. No other format other than zip will be accepted. The name of this file should be **lab#_lastname_loginID.zip**. Example: If your name is John Doe and your login ID is jxd1234, your submission file name must be "lab#_doe_jxd1234.zip" where # is the number of the lab assignment.

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, configuration files, et cetera. DO NOT INCLUDE ANY RUNNABLE EXECUTABLE (binary) program. The first two lines of any file you submit must contain your name and student ID.

You may resubmit the project at any time. Late submissions will be accepted at a penalty of 10 points per day. 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:

15 – Client process works correctly.
15 – Server process works correctly.
05 – Client provides username to server.
05 – Server rejects conflicted usernames.
05 – Server indicates which usernames represent currently connected clients.
10 – Server sends 'Pause' command to randomly selected client every ten seconds.
10 – Client correctly pauses thread when instructed by the server.
15 – Client correctly displays countdown timer on GUI.
15 – Client skips 'Pause' period if instructed by user.
05 – Client and server handle disconnections correctly.
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.

ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE

To receive full credit for comments in the code you should have headers at the start of every module / function / subroutine explaining the inputs, outputs, and purpose 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 or web reference as long as you cite that reference in the comments. If we detect that portions of your program match portions of any other student's program, it will be presumed that you have colluded unless you both cite a specific 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 COURSE CONCLUDES. 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