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 # 2** **Spring 2021**

Objectives:

1. Exposure to polling.
2. Using queues.
3. Additional experience with thread management.

Project Specification:

This lab is intended to be built upon Lab #1. While it is not a requirement that this lab utilize your code from Lab #1, all of the functionality from Lab #1 must be included in Lab #2.

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.

Lab #1 Infrastructure

You will write a client/server system that will check for commonly misspelled words in a block of text. The system will be demonstrated with 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.

A client will connect to the server over a socket and upload a user-supplied text file. The server will have a lexicon of commonly misspelled words that it will read from a file upon startup. The server will scan the user-supplied text file uploaded by the client and check each word against the lexicon. Any word in the user-supplied text file found in the lexicon will be surrounded by brackets. When the server is finished identifying words, the text file will be returned to the client and the connection will be closed.

For example, assume a text file with this text was uploaded to the server:

```
the quck brown fox jumpz over the lazy dg
```

If the server had a lexicon with the following contents:

```
quck jumpz dg
```

The server should return a a text file with the following contents:

```
the [quck] brown fox [jumpz] over the lazy [dg]
```

ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE

Words in the lexicon will be delimited with white space and character case should be ignored. The files should be plain .txt files and the contents may be created and examined with any basic text editor (for instance, Notepad on Windows or TextEdit on macOS).

Lab #2 Additions

Each client will have the ability to make additions to the server's lexicon. The client GUI will present the user with the ability to enter words for inclusion into the lexicon. What GUI components are used to facilitate this are left to the developer's discretion.

Words entered by the user will be placed into a queue and presented on the client's GUI. Every 60 seconds, the server will poll each client's queue. If the queue is not empty, the server should retrieve the contents of the queue and add those to its lexicon.

Once the client has been polled, the contents of the queue should be purged. The server should remove any duplicate entries in the lexicon. Subsequent comparison between user-supplied text files and the lexicon should reflect the updated contents of the lexicon.

The functionality of the client and server are summarized as follows.

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 a different username.
3. Simultaneously handle 'File Upload' and 'Lexicon Additions' until manually killed by the user.

File Upload

1. Upload the user-supplied text file to the server and notify the user of upload completion.
2. Wait until the server has completed checking the file for spelling errors.
3. Receive the updated text file from the server and notify the user that the spell check sequence has completed.
4. Return to Step 1 of 'File Upload' until manually terminated by the user.

Lexicon Additions

1. Present the user with the ability to enter additions to the lexicon via the GUI.
2. Place the entered text into a queue. There should not be an upper bound on the size of the queue.
3. When polled by the server, the client should indicate that a poll was received and print the contents of the queue retrieved by the server (if any).
4. After polling, the client should clear the contents of the queue (if any).
5. Return to Step 1 of 'Lexicon Additions' until manually terminated by the user.

Server:

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

CSE 5306 **Lab # 2** **Spring 2021**

Startup

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. Simultaneously handle 'Spell Check' and 'Polling' until manually killed by the user.

Spell Check

1. Receive a user-supplied text file from the client.
2. Check all words in the user-supplied text file against the lexicon and identify matches.
3. Return the updated file to the client.
4. Return to Step 1 of 'Spell Check' until manually killed by the user.

Polling

1. Every 60 seconds, poll clients for the status of their queues.
2. Retrieve contents from queue, if any.
3. Compare retrieved contents against present contents of lexicon and remove duplicates, if any.
4. Apply retrieved contents to lexicon.
5. Turn to Step 1 of 'Polling' until manually killed by the user.

Notes:

- How the server tracks clients is left to the developer's discretion.
- 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 # 2** **Spring 2021**

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:

20 – All elements of Lab #1 present and functioning correctly.
10 – Client accepts additions to lexicon via GUI.
10 – Client correctly maintains FIFO queue.
10 – Clients indicate they have been polled.
10 – Client prints contents of queue retrieved by server to GUI
05 – Client purges contents of queue after polling.
15 – Server correctly polls clients.
05 – Server removes duplicate entries in lexicon.
10 – Files received from clients compared against updated lexicon.
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.
-30 – Server and/or clients run exclusively from a command line.

ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE

CSE 5306 **Lab # 2** **Spring 2021**

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

ANY HONOR CODE VIOLATION WILL RESULT IN FAILING THE COURSE