CSE 5306 **Lab # 3** **Spring – 2018**

Online Advising Simulation Using Message Queuing

Objective:

1. Exposure to push and pull protocols.
2. Understanding message queuing systems.

Due: Wednesday, April 24th *before* 11:59pm via Blackboard

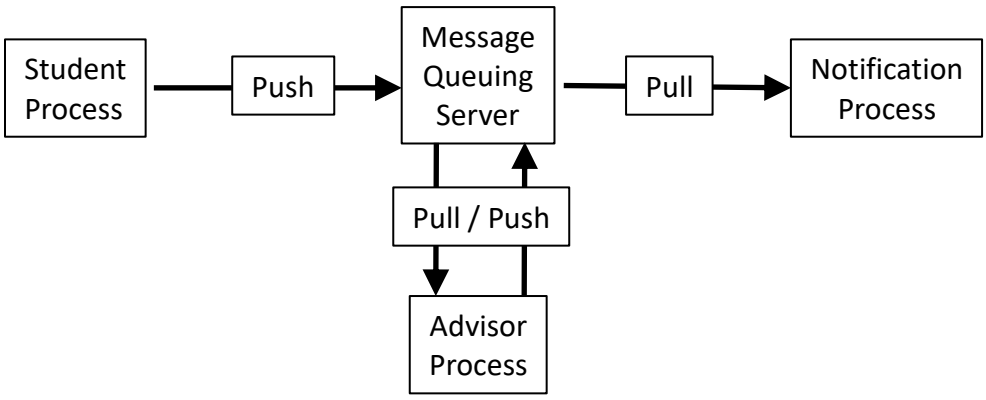
Project Specification:

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!

All processes 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.

In this lab, you will simulate an online advising system which is similar to what you might use to register for classes. Students request clearance for a course from the advisor. At some other time, the advisor approves or disapproves the request, and then the student is notified of the advisor's decision.

Four processes are required to simulate this: a Student Process, an Advisor Process, a Notification Process, and a Message Queuing Server (MQS). The Student Process, Advisor Process, and Notification Process communicate through the Message Queuing Server (message-oriented middleware). Communication from these processes to the Message Queuing Server utilizes both push and pull protocols, as depicted in Figure 1:



Communication Model
Figure 1

Student Process

The Student Process works on behalf of the student. It takes as input the name of the student and the course for which the student wants clearance, as a string. The Student Process should then push that message to the MQS. It should keep doing this (under control of the user) until it is killed.

Advisor Process

When it is started, the Advisor Process should poll the MQS for any messages from the Student Process. If there are any messages, it should pull those messages and approve or disapprove those requests based on a random probability. It should then push its approval or disapproval back to the MQS.

If there are no messages for the Advisor Process in the MQS, then the Advisor Process should sleep for 3 seconds and poll the MQS again. It should keep doing this until is killed. You should print to the GUI whether each message was approved or disapproved.

Notification Process

The Notification Process notifies the Student Process of the Advisor's decision. This involves printing a message to a simple GUI, including the student's name, the course, and the advisor's decision.

The Notification Process should poll the MQS for messages from the Advisor Process. If there are any messages, it should pull them and notify the student as stated above. If there are no messages, then it should sleep for seven seconds and contact the MQS again. It should continue doing this until is killed.

Message Queuing Server

Messages held by the MQS should be persistent, e.g., even if the MQS is shut down, messages should not be lost. There should be one queue (array, linked list, vector, et cetera) to store messages from the Student and the Advisor Process. Once messages are retrieved from their respective processes, the message should be physically deleted from the queue.

The MQS must correctly handle an unexpected process disconnection without crashing. When a process disconnects from the MQS, the GUI must indicate this to the user in real time.

Other Specifications:

1. The MQS will be started first. The other three processes can be started or stopped in any order. Your application must work correctly independent of the order in which these three processes are started and stopped.
2. For the Advisor and Notification Process, print a message (such as "no message found") on the console when it polls the MQS and there are no messages available.
3. Your program must work independently of a browser.

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

15 – Student Process works correctly
15 – Advisor Process works correctly
15 – Notification Process works correctly
15 – MQS works correctly
15 – MQS stores messages persistently
10 – MQS handles connections / disconnections correctly
10 – MQS displays processes connected 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