CSE 5306 **Lab # 2** **Spring – 2018**

Two-phase Commit (2PC)

Objectives:

1. Familiarity with two-phase commit (2PC) protocol.
2. Additional exposure to two communication protocols for distributed processing, HyperText Transport Protocol - HTTP, and Sockets.

Due: 2 April, *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.

Your program will consist of three components:

- A passive server to relay commands between systems;
- Three clients acting as transaction participants; and
- One client acting as the transaction coordinator.

Server

Your server facilitates relaying correspondence between your client processes. It will have a GUI text box that displays message between clients as they transit the server. It should have no other functionality than relaying messages and printing the message contents.

Participant

Three of your previous clients will now act as transaction participants. They will receive an arbitrary string from the coordinator in the INIT phase. Each client should have two GUI buttons that allow the user to vote to either ABORT or COMMIT that transaction (in this lab, COMMIT consists of writing the arbitrary string to a file).

If the user votes to ABORT, the participant should immediately relay that command to the coordinator, and the coordinator should initiate a GLOBAL ABORT accordingly. If the user votes to COMMIT, the participant should enter the READY state and prepare to COMMIT the operation if instructed to do so by the coordinator.

Participants should set timers according to the 2PC protocol. If the coordinator times out, participants should check with one another on how to proceed according to 2PC.

If the participants receive the GLOBAL COMMIT command from the coordinator, they should save the arbitrary string to non-volatile storage. Anything saved into permanent storage should be loaded into the process and displayed to the user upon the process starting. Nothing should be done with the contents of permanent storage other than printing them to the GUI.

Coordinator

One of your clients from the first lab should now act as the transaction coordinator. The client should accept an arbitrary string from a simple GUI that is then passed to the three participants. The participants will then be given a finite amount of time to vote on whether to commit or abort writing that string to a file.

The coordinator should handle votes according to 2PC. The coordinator should also implement timeouts according to 2PC (e.g., if a participant does not respond within a finite amount of time, the coordinator should initiate a GLOBAL ABORT command).

Messages

The messages 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 a process, use `GET`. If you are sending data to a process, use `POST`.

Your program must operate independently of a browser. Time and content-length on the messages should be encoded according to HTTP.

Two-phase Commit (2PC)

References:

1. I strongly recommend reading the pseudo-code included in the textbook for guidance on how to implement 2PC.

2. There are Python source code examples available from the author's website: https://www.distributed-systems.net/index.php/books/distributed-systems-3rd-edition-2017/

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

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

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 – Participants correctly transition states according to 2PC
15 – Coordinator correctly transitions states according to 2PC
15 – Participants handle timeouts according to 2PC
10 – Coordinator handles timeouts according to 2PC
10 – Server shows HTTP message format
10 – HTTP message formats are valid
10 – Participants save arbitrary string to non-volatile storage
10 – Participants load arbitrary string from non-volatile storage on startup.
05 – Comments in code

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.

Deductions for failing to follow directions:

-10 Late submission per day.
-5 Including absolute/ binary/ executable module in submission
-2 Submitted file doesn't have student name and student Id in the first two lines.
-5 Submitted file has a name other than student's lastname_loginID.zip.
-5 Submission is not in zip format.
-5 Submitting a complete installation of the Java Virtual Machine.

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 UTA, state of Texas or US laws or professional ethics. Any violations, however small, will not be tolerated.