Project 2 : Transaction Manager Implementation

Overall Status :

Project is implemented using the strict 2PL protocol and a shared lock is assigned for read operations and an exclusive lock is assigned for write operations so that the transaction manager can be able to handle the locking and as well as releasing of objects. The main thread will create the transaction manager and the corresponding hash table acts as a lock table. Code is written in the zgt_txt.c file and operation statements are converted to thread in zgt_tm.c file. (Note : Lock Upgrades are not taken into consideration in implementation of this project)

*readtx: We find the object in the hash table that a current transaction needs and the transaction acquires the shared lock on the object. Set_lock method returns a true value so that the requested lock on the object will be provided to the transaction. If transaction is aborted, no operation is performed.

Writetx: This method is also similar to the above read method, but here, the transaction tries to acquire the exclusive lock on an object. Rest of the process is same as the above and in write operation, object value will be incremented by 1.

Aborttx: We just change the status of the current transaction to abort and do_commit_abort method will be initiated.

committx: We just change the status of the current transaction to commit and do_commit_abort method will be initiated.

do_commit_abort: This method will free the lock and remove the lock from the transaction manager. Waiting transactions in the queue can also be invoked in this method using zgt_nwait() and semaphores can be released using zgt_v().

set_lock: This method grants the lock for a transaction and it also checks what type of lock is being requested by the transaction for a given particular object. Nodes of current transactions are added to the hash tables and it will point out till the last object node until the lock is granted after which it returns to readtx() or writetx().

Difficulties Encountered:

Writing the semaphore logic and understanding the flow of threads was a bit difficult task and writing the coding part to set the locks was a tedious job as we have took more time and much efforts in understanding the connection between the objects, transactions and the pointers, type of operation that is requested by a particular transaction which can be either in the waiting or active state.

File Descriptions:

We haven't created any new files. Code is implemented using the functions that are provided to us.

Division of Labor:

Before we actually worked on this project, we spent some time on brushing the C++ concepts. Then we have gone through the slides and pdf which are shared to us and we understood the basic concepts of a transaction, objects, locks, semaphores, threads. Once when we are done with the initial part, we started working on the project for implementing the transaction manager. Over all, we have spent around 20 hours each for writing the code zgt_tx.c and zgt_tm.c files

Mahesh - readtx, writetx, do_commit_abort

Harsha - aborttx, committx, set_lock

Error Handling:

a) We were getting errors in the pointing of the objects in the set_lock function. After thorough debugging, we came to know that we are pointing the next pointer to the incorrect object node and then after understanding the underlying logic, we have got the correct result

b) We got stuck in setting the directory path of where the Cygwin is installed and accordingly setting the path of the gcc. Even execution of the Cygwin environment has took us a bit of time to properly understand it. Referring to the pdf again has helped in resolving this issue.

c) As this project involves a lot of test cases , execution was being stopped in the middle and debugging has become tough, then we made sure that all the threads are closed before executing the code.