**Implementation of a Transaction Manager**

Submitted by: Team 01 – Sameer Moses Murala (1001556494)

– Sameena (1001679154)

Under Supervision of: Prof. Sharma Chakravarthy

Department of Computer Science,

University of Texas at Arlington

**Contents**

**Overall Status**

Begin Implementation

Read Implementation

Write Implementation

Commit Implementation

Abort Implementation

**File Description**

**Division of Labor**

**Logical Errors Faced**

**Overall Status**

We first understood the Transaction Manager Structure provided in blackboard. With the help of Professor and TA (Teaching Assistant), we got understood the structure of a Transaction Manager and how we could implement a simple Transaction Manager.

We have successfully implemented the following functions:

* **Begin Implementation**

In this method first in the zgt\_tm.C we created a structure node and passed all the required arguments and create a thread and passed the node and begintx present in zgt\_tx.C as arguments. In the begintx method we get the node and count information. Call the start\_operation to maintain the order of operations in the transaction. Create a new transaction object, lock the transaction manager and add that node to the transaction list. We print the result to the log file and release the transaction manager.

* **Read Implementation**

In this method first in the zgt\_tm.C we created a structure node and passed all the required arguments and create a thread and passed the node and readtx present in zgt\_tx.C as arguments. In the readtx method we get the node and count information. Call the start\_operation to maintain the order of operations in the transaction.

We lock the transaction manager and get the node information and store in a transaction object. We check the status of the transaction and if the transaction is present and is in a active mode we release the transaction manager object and call the set\_lock method to request a lock for the object for the read operation. If the lock is granted we call the perform\_readWrite method to perform the read operation. For the read operation we first lock the transaction manager, decrement the object value by 1 and run a for loop to simulate the read operation time and print the result in the log file. We then release the transaction manager.

* **Write Implementation**

In this method first in the zgt\_tm.C we created a structure node and passed all the required arguments and create a thread and passed the node and writetx present in zgt\_tx.C as arguments. In the writetx method we get the node and count information. Call the start\_operation to maintain the order of operations in the transaction.

We lock the transaction manager and get the node information and store in a transaction object. We check the status of the transaction and if the transaction is present and is in a active mode we release the transaction manager object and call the set\_lock method to request a lock for the object for the write operation. If the lock is granted we call the perform\_readWrite method to perform the write operation. For the write operation we first lock the transaction manager, increment the object value by 1 and run a for loop to simulate the write operation time and print the result in the log file. We then release the transaction manager.

The read and write implementation uses the setlock method to get the lock on the object:

* **set\_lock Implementation**

In the setlock method implementation we first lock the transaction manager. Get the transaction information and store it in a transaction object. We get the transaction that is holding the object for which the transaction is requesting a lock for, from the hash table. If the object is not present in the hash table that means that there is no lock on the object and the lock can be given to the transaction and it is given and added to the hash table. If the object is present in the hash table and the transaction holding the object is same as the transaction requesting the lock for the object, no change is required and the program continues. If the transaction holding the object is not as same as the one requesting it then a couple of checks are done. If the transaction holding the object has a shared lock on it and transaction requesting lock on the object is also a shared lock and the number of transactions waiting on this object is zero then the lock is given. If the number of transactions waiting on the object are not zero or else if the transaction holding the object has an exclusive lock on it then in those scenarios the transaction requesting the lock waits on the transaction holding the object. The current transaction status is changed to waiting and the setTx\_semno method sets the semno and makes transaction wait on the transaction holding the object. A p operation is done on the transaction holding the object to make the transaction requesting the lock sleep on that transaction. The transaction manager object is released. Once the transaction is woken up then the set\_lock method is called again recursively to check for all the conditions to get the lock.

* **Commit Implementation**

In this method first in the zgt\_tm.C we created a structure node and passed all the required arguments and create a thread and passed the node and committx present in zgt\_tx.C as arguments. In the committx method we get the node and count information. Call the start\_operation to maintain the order of operations in the transaction.

We lock the transaction manager and get the node information and store in a transaction object. We check whether the transaction object is present or not. If it is present we call the do\_commit\_abort() method to do the commit operations. In this method get the node information and store in a transaction object. We free all the locks on the objects held by that transaction. We get the semaphore number of that transaction and use it to remove transaction from the transaction list. Using the semaphore number of the transaction we get the number of transactions waiting on that transaction. We do a v operation that many number of times to wake up all the transactions that were sleeping on this transaction. We then print the result to the log file.

* **Abort Implementation**

In this method first in the zgt\_tm.C we created a structure node and passed all the required arguments and create a thread and passed the node and aborttx present in zgt\_tx.C as arguments. In the aborttx method we get the node and count information. Call the start\_operation to maintain the order of operations in the transaction.

We lock the transaction manager and get the node information and store in a transaction object. We check whether the transaction object is present or not. If it is present we call the do\_commit\_abort() method to do the abort operations. In this method get the node information and store in a transaction object. We free all the locks on the objects held by that transaction. We get the semaphore number of that transaction and use it to remove transaction from the transaction list. Using the semaphore number of the transaction we get the number of transactions waiting on that transaction. We do a v operation that many number of times to wake up all the transactions that were sleeping on this transaction. We then print the result to the log file.

**File Description**

No additional files were created for our project. All the files we have used is present in the skeleton code given.

**Division of Labor**

We met a couple of times to understand the transaction manager structure and studied the information and transaction manager data structure given in the pdf files. Figured out the use of various methods and found out how to use those functions and variables in our implementation .Then we decided on how to distribute our work.

As working of the read and write functions were similar and commit and abort were similar, we decided to work on write and commit first. We followed the pdf given in class started working on the code. We divided the work as follows:

Sameer Moses Murala-\_readtx() and wrtitetx method

Sameena-committx() and aborttx() method

Each of us spent around 15 hours each for the implementation of the project.

**Logical Errors**

1. During the implementation of all the methods we were unsure of the locking and unlocking of the transaction manager and other transactions this lead to our code hanging a lot of times.
2. During the implementation of the set\_lock() method we had confusion regarding the order of p and v operations on the transaction object and transaction manager object. The wrong order of operations lead to segmentation fault.
3. During the implementation of the set\_lock() method we faced challege regarding the conditions to check for granting the lock. The wrong conditions led to the code hanging.
4. We faced difficulty understanding the order of execution of operations in a transaction. For example if a transaction is waiting on another transaction for an object and if the next operation in the current transaction can be executed should it wait for waiting operation or continue with next operation.