Implementation of Transaction Manager

MS-CS FALL 2016

COURSE NUMBER: 5331

Instructor: Prof. Upendranath Sharma Chakravarthy

NAME: Aman Bassi, 1001393217

NAME: Aditya Pandey, 1001405034

Overview:

In this project, we have implemented a transaction manager that manages concurrency control using locking. We have implemented strict 2-phase locking protocol with shared locks for read and exclusive locks for write. The transaction manager handles locking and releasing of objects. It consists of a main thread that handles the input, initializes the necessary data structures and a pool of mutex and condition variables to ensure that operations belonging to the same transaction are executed in proper order. The main thread creates a transaction manager object and the associated hash table that acts as a lock table. In this project, we have implemented the following functions: -

1.void *readtx (void *arg): - This is the method to handle readtx action in test file. In this method, we are creating a node that will return the transaction id, count and the object number. We will then call the start_operation and pass transaction id and count. After this, a zgt_tx pointer is created which will store the respective transaction which we get from get_tx method. Finally, the setlock function is called which will set the lock if successful, else put on wait. Later on, the finish_operation is called.

2. void *writetx(void *arg) :- This method is similar to readtx. In this method, we will first create a node that will return the transaction id, count and the object number. We will then call the start_operation and pass transaction id and count. After this, a zgt_tx pointer is created which will store the respective transaction which we get from get_tx method. Finally, the setlock function is called which will set the lock if successful, else put on wait.

3. void *committx(void *arg) :- we will first create a node that will return the transaction id, count and the object number. We will then call the start_operation and pass transaction id and count. After this, a function do_commit_abort is called, we pass two parameters transaction id and character 'Q' which represents Commit.

4. void *aborttx(void *arg) :- we will first create a node that will return the transaction id, count and the object number. We will then call the start_operation and pass transaction id and count. After this, a function do_commit_abort is called, we pass two parameters transaction id and character 'p' which represents abort.

5. int zgt_tx::set_lock() :- In this method, a zgt_tx pointer is created which will store the respective transaction which we get from get_tx method. After this, p operation is called on the semaphore indicated and we will create a zgt_hlink pointer which will store the value return by find operation. Then , v operation is executed on semaphore indicated. If find operation returns null(object is not in hash table), first p operation is executed, then add function is called and finally v operation is executed on the semaphore indicated. If not, We then checked whether the transactionid is same as the objectid. If same, perform_readWrite is called. If not, some other transaction is using the object and has locked it and we will create a zgt_hlink pointer which will store the value return from findt operation. If this pointer is not equal to null then perform_readWrite is called. Moreover, this method sets lock on objno1 with lockmode1 for a transaction in this.

File Descriptions:

There were no new files added. We just needed to implement the above functions in zgt_tm.C and zgt_tx.C file and our goal was to implement those functions and execute all the test cases that were there in the these files.

Division of Labour:

Most of the Implementation in this project is done by both of us. We first tried to understand the basic schema. We studied the information provided to us for the project and the code given to us in our first week.
We then hardcoded the algorithm on paper and checked if our algorithm is working or not. After that, we started to code in the file where we encountered numerous difficulties. To get the final code running, we almost utilized 40-45 hours.
The total project took approximately 55-60 hours.

Logical errors

We encountered a lot of logical errors in our code. Some of them are as follows: -

Logical error 1:

It was difficult to implement setlock function as it is incorporating a lot of conditions and we were not able to figure out the findt function and other headers, which needs to be used in setlock function.

Logical error 2:

As code is involving a lot of pointers, so we were stuck at some places due to pointers.

Logical error 3:

Sequence of p and v operation was not in correct order at some places which we resolved during debugging.

Logical error 4:

We encounter lots of problem regarding file handling. Some of them were unsolved until the very moment. It took approximately 20 hours to code the programs with file handling.
For this we did a brute force method and tried various combinations.