**CSE 5331-Fall 2016**

**DBMS Models and Implementation**

**Project 2: Implementation of a Transaction Manager**

Submitted by :

Team No. 22

Parth Shah(1001246682)

Shivesh Ranjan Singh(1001231391)

Department of Computer Science,

University of Texas at Arlington

**Overall Status**

This project implements Transaction Manager that manages concurrency control using locking. We have implemented Strict Two Phase protocol which works on principle of exclusive lock for Write and shared lock for Read operation. Transaction manger is responsible for successfully handling locking and releasing objects as per the need. Transaction manager uses hash tables and data structure concepts for implementation. The Transaction Manager has been implemented using input files and different methods from zgt\_tx.c and zgt\_tm.c. For any input respective thread is created which calls base methods in zgt\_tx.c. Kindly find the details as below:

readtx() : the read tx function is called by the TXread function using pthread. This function is used to get the node information, fetch the transaction using the get\_tx function and set a shared lock on the the particular object usin set\_lock().

writetx() similar to the read tx it is called by the Txwrite function using pthread. This function is used to get the node information, fetch the transaction using the get\_tx function and set an exclusive lock on the the particular object using set\_lock().

aborttx(): this is called by the AbortTx function and is used to get the transaction which we need to abort. We then call the do\_commit\_abort function with the status as TR\_ABORT.

committx() : this is called by the CommitTx function and is used to get the transaction which we need to commit. We then call the do\_commit\_abort function with the status as TR\_END.

Current transaction is retrieved and its status is verified. For non NULL values of the pointer do\_commit\_abort is called.

do\_commit\_abort (): first we open log file for appending. Depending on the status of the transaction we commit or abort the transaction. Then we retrieve the transaction using get\_tx() and free all semaphores waiting on that transaction.

set\_lock(): in this function we make use of the hash table. We first try to find the transaction that owns the object. Whenever there is there is read/write request then Transaction Manager verifies that whether it’s an existing transaction or not .if it’s an existing one then it’s not held by any other transaction. If no one owns the object the we try to find if the object is already on the hash table, if not then we add it to the hast table. Otherwise we continue regular operation and call the perfor\_readWrite() function to print in our log file. We have also tried dealing with trying to get an exclusive lock on an already locked object but it did work.

Perform\_readWrite(): in this function we are getting the object value accesed by the transaction. Then we open the log fil for appending. Lastly we are printing the appropriate format in the log file with its transaction id, object number, lock mode, granted etc. Then we sleep off for a random amount of time.

We tried to implement for deadlock but it did not work.

**Where I encountered difficulty**

This project has been very challenging. Few of the challenges were

Understanding data structure of Transaction Manager was difficult as it was quite cumbersome.

Coding on Terminal turned out to be quite time consuming process as we need to upload it even if we made a trivial change.

Semaphore has been the real trouble maker for us. It was difficult to understand them at first and there implementation was also difficult.Lack of documentation on the existing code make me take a lot of time to understand functionalities only by using

**File Descriptions**

No new file was created. We added code in existing files to implement Transaction Manager.

**Division of Labor**

Me and Parth worked together as a team to complete this project. The algorithm is very well explained in the document provided and it was very helpful in developing the logic of the code. The project was given to us in first week of Oct and we had 4 weeks to complete the project. We started with getting an understanding of the structure of Transaction Manager and how is it implemented. Understanding it was not a cakewalk and we had to devote a week time in getting complete understanding of it. The algorithm had to be implemented in C++ and involved looking it through different angles .We devoted 3-4hours religiously in 2nd and 3rd week for Read/Write code. All in all we have spent roughly 50-55 hrs in completing the project.

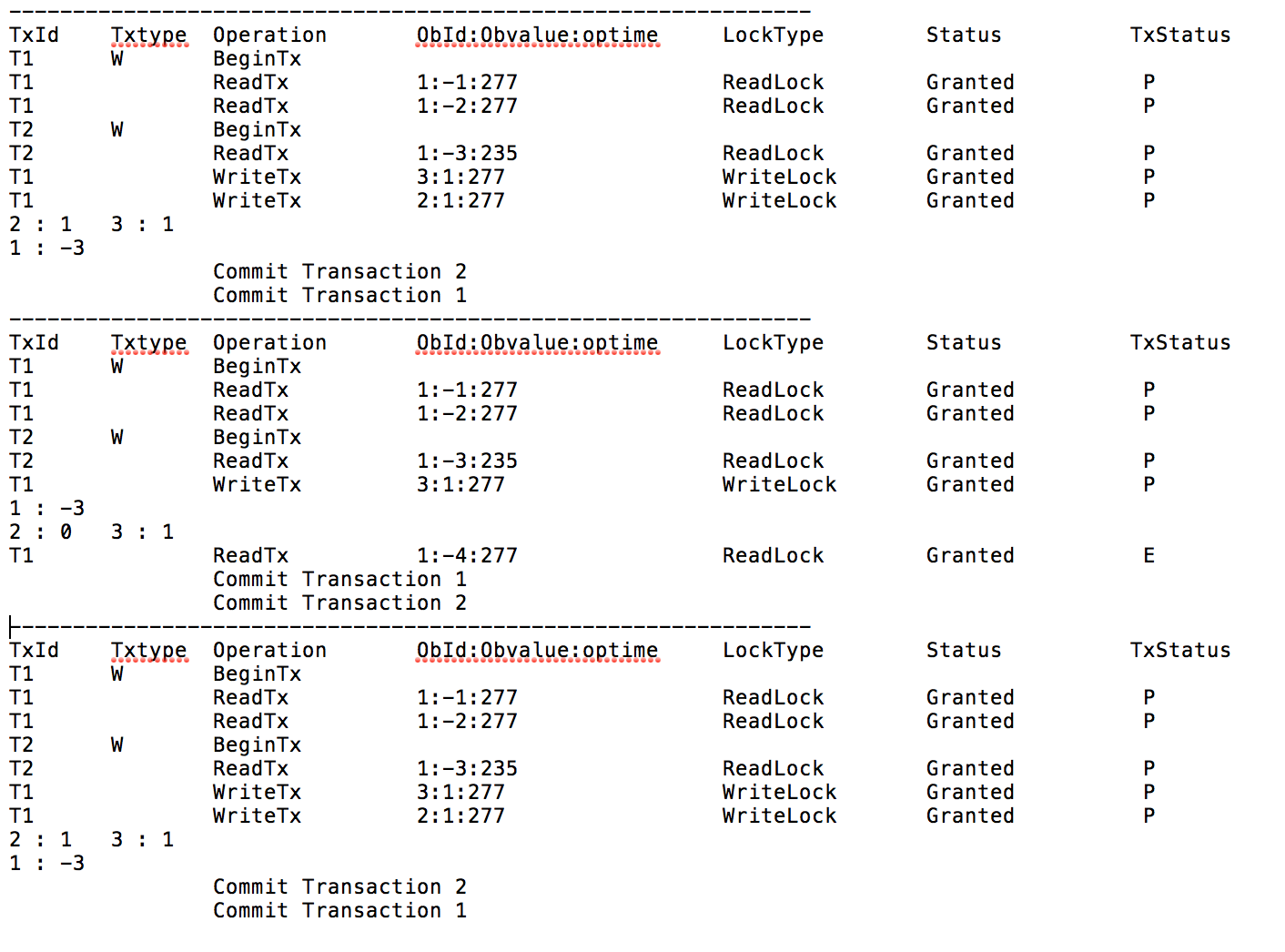
**Logical Errors**

Semaphores ->We encountered numerous issues related to semaphores. Incorrect handling of P and V operations on semaphores resulted in errors. While calling for P operations and assigning semaphores we were not handling V operations correctly. One of major issue was in ending the transactions. We failed to release all the semaphores waiting for completed/aborted transaction.

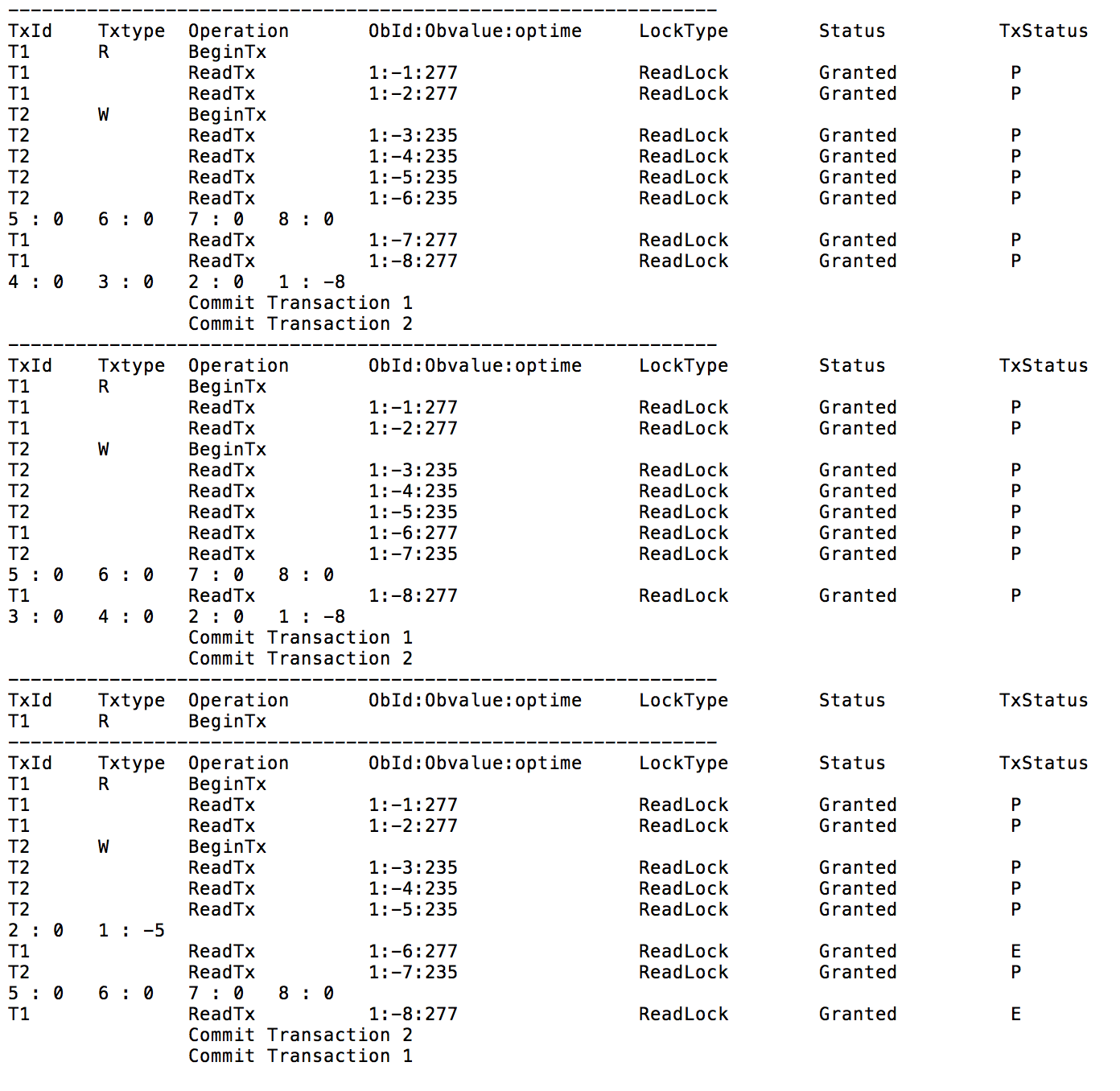
Segmentation Fault -> Segmentation Fault has been most frequent error we encountered. This aroused in traverse method(). We were not checking for already visited node and whose level is 0. In this case we called traverse method recursively from node’s head and in case of visited node its level is reduced by 1. So basically nodes with level (-1,0,1) were not been handled which resulted in errors.

Linker error for 64 bit mac -> It gave us a lot of trouble for linking of files together. Sometimes the error said that it was a problem with the architecture of the hardware of the laptop but it would run on omega which was time consuming.

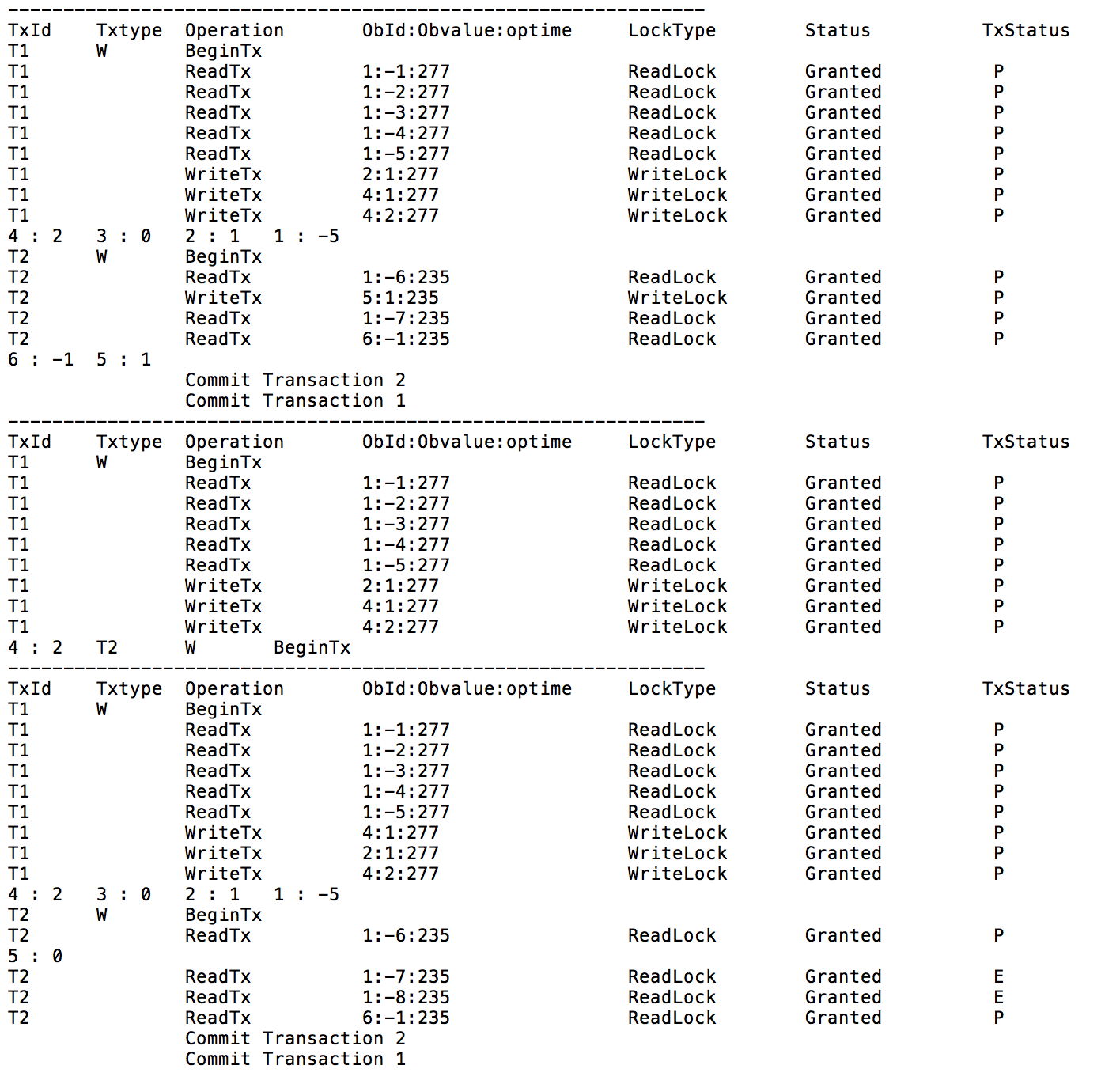
**Output**

C2Tsz.log****

NoC2T.log



S2T.log

****

deadlock.log

