PROJECT 2

TRANSACTION MANAGER IMPLEMENTATION

SUBMITTED BY:

ANKITA DATTA - 1001774282 , HARSH CHALUDIA – 1001744551

Abstract

In this project, we implement a simple transaction manager using C/C++. The transaction manager manages concurrency control using strict two phase locking protocol with shared locks for read and exclusive locks for write. The project does not implement upgrading of locks. The project will complete the implementation of the classes zgt_tm and zgt_tx.

Table of Contents

Overall Status

In zgt_tx, we implement the following functions:

1. Void *readtx(void *arg):

   In this, we are looking to acquire the 'S' lock and finding the object that the current tx needs. No further operations are performed if the tx is set to abort. Else the tx is sent to the set_lock method. And we look for the object in the hashtable. After the set_lock method returns a true value, we know that now we have shared lock and can perform read operation.

2. Void *writetx(void *arg):
   Does the writing operation quite similar to readtx. Here, we acquire the critical lock and find the object that the current tx needs. No further operations are performed if the tx is set to abort. Else the tx is sent to the set_lock method. And we look for the object in the hashtable. After the set_lock method returns a true value, we know that now we have shared lock and can perform read operation.

3. Void *aborttx(void *arg):
   Calls the function do_commit_abort, that passes the tid and the status that is 'A' for abort. Here we change the status of the tx to abort and start a method to initialize do_commit_abort which deals with freeing of locks held by txs and orgranize (removing)them.

4. Void *committx(void *arg):
   Calls the function do_commit_abort, that passes 'C' for commit in status. Very much similar to abort tx method. Here,

we modify the status of tx and start the do_commit_abort method which deals with freeing of locks and remove the transactions.

5. Void *do_commit_abort():
   This function when invoked, frees the locks and removes a transaction from the transaction manager. It is used by both commit and abort method. Here, it deals with retrieving of transactions that are waiting on semaphores. It also deals with checking of wait-queue. It releases all the semaphores 1-by-1 using zgt_v(). After this the released objects can be used again by the txs.

6. Int set_lock():
   Checks if the object is present in the hash table or not and grants locks to the transaction on a particular object. Also, if the tx already holds a lock and is requesting for it, then it is already granted to its status. Every read/ write has an object assigned and this cannot be shared by neither. There could be a chance, if the particular tx is waiting for X mode, it has to wait infinitely in case more shared lock enquiries coming in.

7. Void perform_readwrite():
   It is used to write in the logfile after everything is sorted out from the set_lock method. Updates the objarray[] value by incrementing 1 for write and decrementing 1 for read.

The following outputs were obtained:

COMPILING THE FILES:

```
  OIT Help Desk Self Service is available: https://go.uta.edu/sn
******************************************************************************
[hxc4551@omega ~]$ cd tx-manager
[hxc4551@omega tx-manager]$ cd src
[hxc4551@omega src]$ make clean
rm -f *.o *~ zgt_test
[hxc4551@omega src]$ make
/usr/bin/g++  -I../include -I. -L/usr/lib -DTX_DEBUG -DTM_DEBUG -DHT_DEBUG -c zg
t_test.C
/usr/bin/g++  -I../include -I. -L/usr/lib -DTX_DEBUG -DTM_DEBUG -DHT_DEBUG -c zg
t_tm.C
/usr/bin/g++  -I../include -I. -L/usr/lib -DTX_DEBUG -DTM_DEBUG -DHT_DEBUG -c zg
t_tx.C
/usr/bin/g++  -I../include -I. -L/usr/lib -DTX_DEBUG -DTM_DEBUG -DHT_DEBUG -c zg
t_ht.C
/usr/bin/g++  -I../include -I. -L/usr/lib -DTX_DEBUG -DTM_DEBUG -DHT_DEBUG -c zg
t_semaphore.C
/usr/bin/g++ -lpthread  -DTX_DEBUG -DTM_DEBUG -DHT_DEBUG -I../include -I. zgt_te
st.o zgt_tm.o zgt_tx.o zgt_ht.o zgt_semaphore.o -o zgt_test
[hxc4551@omega src]$ ./zgt_test ../test-files/S2T.txt
```

INPUT FILE:

zgt_tx.C | S2T.txt

```
1   // serial history
2   // 2 transactions
3   // same object accessed
4   // multiple times
5   Log S2T.log
6   BeginTx 1 W
7   Read    1 1
8   Read    1 2
9   Write   1 3
10  Write   1 4
11  read    1 1
12  write   1 2
13  write   1 4
14  write   1 4
15  commit 1
16  begintx 2 W
17  read    2 5
18  write   2 5
19  write   2 6
20  read    2 6
21  commit  2
22
```

OUTPUT FILE:

```
1  ------------------------------------------------------------
2  TxId    Txtype  Operation   ObId:Obvalue:optime LockType    Status      TxStatus
3  T1  W   BeginTx
4  T1          readTx      1:-1:277        ReadLock    Granted      P
5  T1          readTx      2:-1:277        ReadLock    Granted      P
6  T1          writeTx         3:1:277         WriteLock   Granted      P
7  T1          writeTx         4:1:277         WriteLock   Granted      P
8  T1          readTx      1:-2:277        ReadLock    Granted      P
9  T1          writeTx         2:0:277         WriteLock   Granted      P
10 T1          writeTx         4:2:277         WriteLock   Granted      P
11 T1          writeTx         4:3:277         WriteLock   Granted      P
12 T2  W   BeginTx
13 T2          readTx      5:-1:235        ReadLock    Granted      P
14 T2          writeTx         5:0:235         WriteLock   Granted      P
15 T2          writeTx         6:1:235         WriteLock   Granted      P
16 T2          readTx      6:0:235     ReadLock    Granted      P
17 T2         CommitTx
18
19 T1         CommitTx
```

Difficulties Encountered

- Initial problem was how to compile the file using different compilers which then finally we decided to go for g++ compiler in windows. Also, understanding the semaphores was a huge task for us. We get to know later that for every transaction we have a semaphore allotted. And this keeps on incrementing until the transaction is desired.

- set_lock method really took a lot of time which was because of simple problems like not releasing and locking the txs.

- In the initial stages, we were kind of debugging to why the output was not having those perform read/write operations. And then we later on got to know that we need to pass those through the set_lock method.

- The idea of threads was new to us and we invested a lot of time understanding it.

- Common errors like segmentation we encountered.

- Took a little time to understand how the object value really works, and how we can verify it with the output if it is correctly working or not.

File Description

No new files were created in this projects. All files used were provided in the assignment attachment.

Division of Labor

The initial days were given to understand the working of a transaction manager. Later, we together formulated the logic that we would use to implement the functions. The code for readtx, writetx, committx and aborttx functions was done by Ankita and the code for set_lock, perform_readwrite, do_commit_abort and the rest was done by Harsh.

Logical Errors

1) We encountered a lot of logical errors while writing code for acquiring locks and we did invest a lot of time to the information provided to us.

2) Initially we had problems like blank output because we did make a lot of mistakes in the set_lock method which kind of skipped the lockmode conditions leading to perform the perform read/write operations. And this made us realise that the most important was to release and lock the tx wherever and wherever needed.

3) Dealing with semaphores was a little painstaking process. When the tx ends, we got to realize later that we need release all the semaphores again so that the new txs can use it.

4) Simple mistakes like not setting the g++ compiler path in the system variables led to wrong compilation technique. Also, the makefile paths had to be edited while compiling in windows which we got to know about it later on.

5) Problems like no errors in our code led us to an illusion that our output was working fine but thanks to TA that he showed a sample input file/output file to ensure our version was correct.

6) We did make a lot mistakes using pointer because we were not quite familiar with c/c++ tools.