Part A: Rigorous 2-phase Locking using Monitors

In this assignment, we will be implementing 2-phase locking that we studied in the database course. Assume atleast 5 database state variables eg. X, Y, ..

A transaction will involve operations Read (R), Write (W) & Commit (C) / Abort (A).

eg. R(X) W(Y) C. Assume that each transaction runs in a separate thread. To perform R(X), we need to acquire a read lock on X from the lock mgr. and to perform W(X), we need to acquire a write lock on X. Suppose the tx. contains R(X) W(X), then we need to upgrade the read lock to write lock for X.

When a transaction commits/aborts, then all the locks acquired by the transaction will be released. If transaction requests a lock that is currently held by another transaction, then the requesting transaction waits.

Design:

Parse Input file → Store transactions in an array → Execute the transactions

- Create a Transaction class which stores transaction id, and the sequence of operations <op. type, variable Name> and outcome Commit / Abort. Optype corresponds to Read / Write.

- Parse the input file and Store the transactions in an array.

- Iterate through the transaction array and create a new thread for executing each transaction. Write a function execute transaction (Transaction) that will be run by each thread. Depending on the sequence of operations, each transaction will issue requests to acquire appropriate lock from the LockMgr.

- If the lock is held by another transaction, then the current thread waits until the lock is released.

- The design of the Lock Mgr is shown in the next page.

- Create a LockMgr class which has methods such as
  - acquireReadLock(txId, var.Name)
  - acquireWriteLock(" , ")
  - upgradeToWrite(" , ")
  - releaseLock(txId, var.Name)

It should use condition variables, one for each variable in the database to allow waiting of transactions. LockMgr is the monitor. Identify its state variables & condition variables [hint: 1 condition variable for each database state variable, 1 Queue for each database state variable to keep track of all the transactions that are waiting for a lock on that variable. 1 Lock for the monitor]

```
class LockMgr {
    Lock lock;
     CV [] cvs;
}    Queue[] qs;        methods defined above -
```

Input:

- Number of transactions : N
- Database state variables : u = 100, v = 1000, x = 50, y = 20, z = 100 (atleast 5)
- Specify each transaction :

transaction Id

R, X

X = X + 100 | X = X - 50 | X = X + (earlier variable read)

W, X

C

the operation can be one of these

Output:

- final values of the database state variables
- Order in which locks are actually acquired / released -

  eg. R-lock [TxId, Var.Name]

  unlock [TxId, Var.Name]
- if transaction has to wait, then output wait_R-lock [TxId, Var.Name] or wait_W-lock [" , "]
- "Successfully executed all the transactions" - printed by the main thread after all transactions are completed.

Test Cases

1) Transactions have only read operations. All transactions commit

2) Transactions have read/write operations but they dont interfere with each other. All transactions commit.

3) Same as 2) but few transactions abort.

4) Transactions have conflicting read/write operations but the 2PL will exeute successfully without deadlocks.

   (a) All transactions commit
   (b) Few transactions abort

5) Same as 4) but the 2PL schedule results in a deadlock.

Bonus: (using signals to output transactions involved in a deadlock)

When there is a deadlock the main thread will not exit. Thus, "Successfully completed - ." line wont be printed. At that time, the user can press "Ctrl-C" this signal is caught by the main thread which then outputs the transactions involved in the deadlock.

Submission Instructions:

1) Please donot copy. If you have any difficulty, please ask over email. If you are caught copying, you will get an 'F' grade in the whole course.

2) You should submit your src. code, input/output for each test case eg. input1.txt, output1.txt correspond to test case 1 Without the src code, your submission wont be considered.

3) Programming language: C++