Project 1(CSE5331 Summer 2019)

Read Me

-By

Rachana Naganagouda Patil- 1001644227
Asita Prakash Satpute – 1001600126

Summary:
In this project, we will implement a program that simulates the behavior of the two-phase (2PL) locking protocol (**rigorous 2PL**) for concurrency control. We will be using the **wound-wait** method for dealing with deadlock.

This folder contains below files:

- ✓ main.py – It has the python source code for this project. This code can be run on **pycharm**. (Added comments in the code)
- ✓ Output file (1 to 10)
- ✓ ReadMe

- **Pseudo-code:**

We will implement this code using **python language**. First, we will **import OS module**. OS module in python provides functions for interacting with the operating system.

For **transaction table and lock table** we will use **array list data structure** and for that we will be creating two classes of each and these classes will have some functions to update the array list used by calling functions.

To read the input, we will first save input in a txt file using function **read_file()** and read it from the same through code and add it to an array list. Then we will read each operation and call specific function matching with that operation. Below is the list of some functions called depending on the operation read from file. (before calling these functions it will check the status of transaction whether it is waiting or aborted etc. using **check_status()** function)

b; begin_transaction()

This will add transaction record into the transaction table and start the transaction.

r; readLock_item()

This function is used to put item under read lock by that particular transaction. It will also check if there are any conflicting locks before giving the read lock and in case of deadlock it will call wound wait algorithm (**wound_wait()** function). Also, will add relevant information to the lock table.

w; writeLock_item()

This function is used to put item under write lock by that particular transaction. It will also check if there are any conflicting locks before assigning the write lock and in case of deadlock it will call wound wait algorithm(**wound_wait()** function). Also, will add relevant information to the lock table.

e; end_transaction()

This function is called when the schedule shows that the transaction is ending. It will commit the transaction and free all items locked by that transaction. It will also add relevant information to the lock table.

Besides these functions, we will implement some more function mentioned below which will be called by the above functions for more information and validations.

unlock_item()

This will be called when the transaction ends or if the transaction is aborted. This function will free all items held by that particular transaction.

wound_wait()

This will implement the wound wait algorithm. It will check if the transaction requesting the item is younger or older. If the requesting transaction is older, it will abort/wound the transaction holding the lock else will wait till the transaction holding the lock frees the item.

Output will be printed to the console. Output shows the actions taken for each operation in the schedule given as input using Rigorous 2PL protocol and wound wait algorithm in case of deadlock.

The process of simulating program:

- At the very first step the function **read_file()** is called to access the input file in which transaction schedule is present. The file is read in the read mode and assigned the whole schedule in a separate list.
- To differentiate what each transaction in a schedule means and does, we extract out each transaction from the schedule and depending on their transaction IDs, transaction operation and data item, they are assigned and updated in the lock and transaction table accordingly.
- If the transaction is b(transcation_ID)(data item name), it is allowed to go through the function **begin_transaction().** Here, ID is extracted and the values are appended to the Transaction table.

- If the transaction is r(transaction_ID)(data item name), it is allowed to go through the function **readLock_item(transaction)**. Here, it is checked whether particular ID is present in lock table or not. If not, its values are appended directly to the Lock Table. If yes, it checks whether the particular data item is being processed current or not. If that data item is processed currently by write operation of another transaction it will lead to conflicting operations and hence **wound_wait()** function is called to resolve the priority transaction operation or else if it is read operation of other transaction, it is allowed to access that particular data item.
- If the transaction is w(transaction_ID)(data item name), it is allowed to go through the function **writeLock_item(transaction)**. Here, it is checked whether particular ID is present in lock table or not. If not, its values are appended directly to the Lock Table. If yes, it is checked whether particular data item is locked under multiple transactions or not. If it is held by only one read operation and that too same transaction ID as current requesting transaction, it is upgraded to write_operation. If multiple transactions are present, then conflicting situations are formed and **wound_wait()** function is called to give priority to the one among transactions.
- If the transaction is e(transaction_ID), it is allowed to go through the function **end_transaction(transaction)**. Here it will check whether the transaction is aborted or not. If not, it will assign the transaction state as "Commit" and unlock the respective resource items through **unlock_item()** function.

- **Transaction Table:**
  For creating this Transaction table, we will use **array list data structure** in our program.

Below are short names and description given for each transaction table column:
Transaction Id (Tx_id): The unique Id is assigned to each transaction.
Transaction Timestamp (Tx_TS): Transaction timestamp is a unique identifier created by DBMS when the transaction begins to identify a transaction.
Transaction State (Tx_state): Transaction state is the current state of the transaction like active, blocked (waiting), aborted (cancelled) or committed.
List of items read locked by the transaction (Tx_itemsReadLocked): List of items read locked by each transaction.
List of items write locked by the transaction (Tx_itemsWriteLocked): List of items write locked by each transaction.

We will maintain below relevant information about each transaction

Tx_id	Tx_TS	Tx_state	Tx_itemsReadLocked	Tx_itemsWriteLocked

- **Lock Table:**
  For creating this lock table, we will use **array list data structure** in our program.

Below are short names and description given for each Lock table column:
Item name (Item_name): This is showing the list of data items that are locked currently. If the item is not in this list that means the item is unlocked for now.
Lock state (Lock_state): This will tell us if that data item is read locked/write locked.
Transactions holding read lock (ReadLock_Tid): Transaction ids of transactions that have read locked that item.
Transaction holding write lock (WriteLock_Tid): Transaction ids of transactions that have write locked that item.
Transactions waiting for read lock (ReadWaiting_Tid): List of transaction ids of transactions waiting for the item to be unlocked so that they can acquire read lock.
Transactions waiting for write lock (WriteWaiting_Tid): List of transaction ids of transactions waiting for the item to be unlocked so that they can acquire write lock.

We will maintain below relevant information about each locked item.

Item_name	Lock_state	ReadLock_Tid	WriteLock_Tid	ReadWaiting_Tid	WriteWaiting_Tid

- **The actions the simulation will do for the two given inputs:**

Assumptions:

1. Initially all the items are unlocked.
2. Relevant information about each locked data item will be updated in lock table.

First Input:

b1; This is the beginning of transaction 1(T1). Create record for Transaction 1 in the transaction table.

r1 (Y); T1 acquires read lock on the data item Y.

w1 (Y); T1 upgrades from read lock to write lock on item Y.

r1 (Z); T1 acquired read lock on the data item Z.

b3; This is the beginning of transaction 3(T3). Create record for Transaction 3 in the transaction table.

r3 (X); T3 acquires read lock on the data item X since no other transaction has locked this item before.

w3 (X); T3 upgrades from read lock to write lock on item X since no other transaction has locked this item in read or write mode before.

w1 (Z); T1 upgrades from read lock to write lock on item Z since no other transaction has locked this item in read or write mode before.

e1; The transaction T1 will commit and unlock the read lock and write lock on items Y, Z.

r3 (Y); T3 acquires read lock on the data item Y since T1 has released its lock on Y after committing.

b2; This is the beginning of transaction 2(T2). Create record for Transaction 2 in the transaction table.

r2 (Z); T2 acquires read lock on the data item Z since T1 has released its lock on Z after committing.

w2 (Z); T2 upgrades from read lock to write lock on item Z since no other transaction has locked this item in read or write mode currently.

w3 (Y); T3 upgrades from read lock to write lock on item Y since no other transaction has locked this item in read or write mode currently and only T3 itself has read locked this item.

e3; The transaction T3 will commit and unlock the read lock and write lock on items X, Y.

r2 (X); T2 acquires read lock on the data item X since T3 has released its lock on X after committing.

w2 (X); T2 upgrades from read lock to write lock on item X since no other transaction has locked this item in read or write mode currently.

e2; The transaction T2 will commit and unlock the read lock and write lock on items X, Z.

Conclusion: In this schedule no dead lock ever happens and all the three transaction (T1, T2. T3) successfully finish all operation and commit when each transaction ends.

Second Input:

b1; This is the beginning of transaction 1(T1). Create record for Transaction 1 in the transaction table.

r1(Y); T1 acquires read lock on the data item Y.

w1(Y); T1 upgrades from read lock to write lock on item Y.

r1(Z); T1 acquired read lock on the data item Z.

b2; This is the beginning of transaction 2(T2). Create record for Transaction 2 in the transaction table.

r2(Y); Transaction 2(T2) will be waiting for T1 to release lock on Y since T1 has a write lock on Y and write lock is exclusive. Also, T2 will go in waiting mode instead of wounding T1 because timestamp of T2 is larger than T1 as it started after T1.
Based on wound wait algorithm:
If TS(T2) < TS(T1))

Then wound T1;
Else T2 waits;

b3; This is the beginning of transaction 3(T3). Create record for Transaction 3 in the transaction table.

r3(Z); T3 acquires read lock on the data item Z since read operations do not conflict.

w1(Z); T1 will wound/abort T3 as T3 has read lock on Z. Using wound wait algorithm timestamp of T1 is less than timestamp of T3. T3 unlocks all locks it holds (in this case, read lock on Z).
Based on wound wait algorithm:
If TS(T1) < TS(T3)
Then wound T3;
Else T1 waits;
Finally, T1 upgrades its lock from read to write on Z.

e1; The transaction T1 will commit and unlock the read lock and write lock on items Y, Z. T2(which was waiting for T1 to unlock item Y) will now change its state from waiting to active as it acquires read lock on Y.

w3(Z); Transaction T3 already aborted.

e3; Transaction T3 already aborted and when it was aborted it unlocked all the item it had locked.

Conclusion: In this schedule the dead lock condition took place twice and was resolved using wound wait algorithm. Transaction T3 was aborted during wound wait and T1 committed successfully. T2 goes in waiting mode but resumes as soon as T1 committed and unlocked its locks at e1;

Thank You!