Serializability in Practice

- It is difficult to test for the serializability of a schedule
- Interleaving is determined by Operating system and we have no control
- In practice, methods are used which ensure serializability without testing the schedule.
- Protocols are defined, which if followed, will ensured serializability
- Some of them are: Lock based, Timestamp based, multiversion and optimistic protocol.

Concurrency Control Through Locks

- **Lock:** variable associated with each data item
  - Describes status of item wrt operations that can be performed on it
- Binary locks: Locked/unlocked
  - Enforces mutual exclusion
- Multiple-mode locks: Read/write
  - a.k.a. Shared/Exclusive
- Three operations
  - read_lock(X)
  - write_lock(X)
  - unlock(X)
- Each data item can be in one of three lock states

Implementation

- Maintain lock table
- Keep track of locked items and their locks

  <data item, LOCK, no_of_reads, locking_transaction>
- For read locks, keep track of the number of transactions that hold a read lock on an item

Locking Rules

1. T must issue read_lock(X) or write_lock(X) before any read_item(X) op is performed in T
2. T must issue write_lock(X) before any write_item(X) op is performed in T
3. T must issue unlock(X) after all read_item(X) and write_item(X) ops are completed in T
4. T will not issue a read_lock(X) if it already holds a read lock or write lock on X (may be relaxed)
5. T will not issue a write_lock(X) if it already holds a read lock or write lock on X (may be relaxed)

Lock Conversions

- Sometimes beneficial to relax locking rules 4 and 5
- Upgrade read lock on X to a write lock (by issuing a write_lock(X))
  - Only possible if T is the only transaction holding a read lock on X
- Downgrade a write lock by issuing a read_lock(X)
- Must be noted in lock table

Granting of Locks

- Suppose T2 has read-lock on item X
- T1 is requesting write-lock on item X; needs to wait for T2 to release
- T3 requests read-lock on X; request is granted
  - Assume shortly thereafter T2 relinquishes read-lock
  - Continue scenario through a sequence of transactions all requesting read-lock on X
  - T1 will never make progress
- T1 is said to be *starved*

Granting of Locks

- How do you avoid starvation in the presence of locks?
- Assume T_i requesting lock on Q
- Grant lock provided that
  - No locking conflict with lock requested by T_i, OR
  - No other transaction waiting for lock and made request before T_i

Two Transactions

T1	T2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
X:=X+Y;	Y:=X+Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Let’s assume serial schedule S1: T1;T2

Initial values: X=20, Y=30 Result: X=50, Y=80

Locks Alone Don’t Do the Trick!

Let’s run T1 and T2 in interleafed fashion

Schedule S

T1	T2
read_lock(Y);	
read_item(Y);	
unlock(Y);	
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	Y:=X+Y;
	write_item(Y);
	unlock(Y);
write_lock(X);	
read_item(X);	
X:=X+Y;	
write_item(X);	
unlock(X);	

unlocked too early!

Non-serializable!

Result: X=50, Y=50

Two-Phase Locking (2PL)

- Def.: Transaction is said to follow the *two-phase-locking protocol* if all locking operations precede the *first* unlock operation
  - Expanding (growing) = first phase
  - Shrinking = second phase
- During the shrinking phase no new locks can be acquired!
  - Downgrading ok
  - Upgrading is not

Example

T1'	T2'
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
write_lock(X);	write_lock(Y);
unlock(Y);	unlock(X);
read_item(X);	read_item(Y);
X:=X+Y;	Y:=X+Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

- Both T1' and T2' follow the 2PL protocol
- Any schedule including T1' and T2' is guaranteed to be serializable
- Limits the amount of concurrency

Variations to the Basic Protocol

- Previous technique known as *basic 2PL*
- *Conservative* 2PL (static) 2PL: Lock all items needed BEFORE execution begins by predeclaring its read and write set
  - If any of the items in read or write set is already locked (by other transactions), transaction waits (does not acquire any locks)
  - Deadlock free but not very realistic

Variations to the Basic Protocol

- *Strict* 2PL: Transaction does not release its write locks until it aborts/commits
  - No other transaction read item X untill this transaction committs: endure recoverability.
  - Not deadlock free
  - Most popular variation of 2PL

Variations to the Basic Protocol

- *Rigorous* 2PL: No lock is released until abort/commit
  - Transaction is in its expanding phase until it ends
  - Conservative 2PL is always in shrinking phase where as Rigorous 2PL is always in expanding phase.

Concluding Remarks

- Concurrency control subsystem is responsible for inserting locks at right places into your transaction
  - Strict 2PL is widely used
  - Requires use of waiting queue
- All 2PL locking protocols guarantee serializability
- Does not permit all possible serializable schedules
  - Conservative and rigorous 2PL charge a high price for serializability
- However, deadlock-based algorithms may suffer from starvation and *deadlock* (see next lecture)