CMPT 454 Assignment 5: Transactions

1 - Definitions

a) Define these terms: [**1** mark each]

1. deadlock – two (or more) transactions are in *deadlock* when they hold locks on objects required by other transactions such that all the transactions are suspended waiting for each other to release locks
2. rollback – a process to reverse the effects (i.e. writes) of a transaction when it aborts
1. blind write – a write to an object by a transaction without a preceding read to the same object
3. unrecoverable schedule – a schedule where if a transaction aborts then rolling back that transaction would also entail rolling back a second transaction that has already committed
4. phantom problem –
5. dirty read – a read of a DB object that has been modified by another, not yet committed, transaction
6. unrepeatable read – a read of an object by a transaction where the same value would not be read again, even though the object has not been modified by the transaction
7. view equivalent schedule – two schedules are *view equivalent* if they contain the same transactions and each transaction reads the same data and the schedules result in the same ending DB state
8. serializable schedule – a schedule that is equivalent (i.e. has the same results as) some serial schedule
9. conflict equivalent schedule – two schedules are *conflict equivalent* if they contain the same transactions and the order of each pair of conflicting actions is the same

2 - Transaction Schedules

Write a schedule for parts (**a**) to (**e**) that satisfies the listed requirements. When reading a transaction the first letter indicates the action (**R**ead or **W**rite), the subscript indicates the transaction that the action belongs to (transaction 1, 2 or 3), and the letter in brackets indicates the data object that the transaction affects (A, B or D). [**2** marks each, there could be multiple different answers, not necessarily just the one (or two) I've provided]

a) For the transactions given below write a non-serial schedule that is *conflict serializable*.

T1: $R_1(A)$; $R_1(B)$; $W_1(A)$; $W_1(B)$
T2: $W_2(A)$; $R_2(B)$; $W_2(B)$

$R_1(A)$; $R_1(B)$; $W_1(A)$; $W_2(A)$; $W_1(B)$; $R_2(B)$; $W_2(B)$
$W_2(A)$; $R_1(A)$; $R_2(B)$; $W_2(B)$; $R_1(B)$; $W_1(A)$; $W_1(B)$

I'm pretty sure these are the only solutions

b) For the transactions given below write a non-serial schedule that is neither conflict serializable nor view serializable.

T1: $R_1(A)$; $R_1(B)$; $W_1(A)$; $W_1(B)$
T2: $W_2(A)$; $R_2(B)$; $W_2(B)$

$R_1(A)$; $W_2(A)$; $R_1(B)$; $W_1(A)$; $W_1(B)$; $R_2(B)$; $W_2(B)$
$W_2(A)$; $R_1(A)$; $R_2(B)$; $R_1(B)$; $W_1(A)$; $W_1(B)$; $W_2(B)$
$R_1(A)$; $R_1(B)$; $W_2(A)$; $R_2(B)$; $W_2(B)$; $W_1(A)$; $W_1(B)$

There are others

c) For the transactions given below write a non-serial, but *conflict serializable*, schedule, where each transaction processes at least one action before any other transaction's final action. In addition, each write of an object by a transaction should immediately follow the read of that object by the same transaction – i.e. there should be no other actions between a read and a write of the same object by the same transaction.

T1: $R_1(A)$; $W_1(A)$; $R_1(B)$; $W_1(B)$
T2: $W_2(B)$; $R_2(D)$; $W_2(D)$;
T3: $R_3(A)$; $W_3(A)$; $W_3(D)$;

$R_3(A)$; $W_3(A)$; $W_2(B)$; $R_1(A)$; $W_1(A)$; $W_3(D)$; $R_2(D)$; $W_2(D)$; $R_1(B)$; $W_1(B)$
$W_2(B)$; $R_1(A)$; $W_1(A)$; $R_3(A)$; $W_3(A)$; $R_2(D)$; $W_2(D)$; $W_3(D)$; $R_1(B)$; $W_1(B)$
$R_1(A)$; $W_1(A)$; $W_2(B)$; $R_3(A)$; $W_3(A)$; $R_1(B)$; $W_1(B)$; $R_2(D)$; $W_2(D)$; $W_3(D)$

d) For the transactions given below write a non-serial schedule that is *not* conflict serializable but *is* view serializable.

T1: $R_1(A)$; $R_1(B)$; $W_1(A)$; $W_1(B)$
T2: $W_2(A)$
T3: $W_3(A)$

$R_1(A)$; $R_1(B)$; $W_2(A)$; $W_1(A)$; $W_1(B)$; $W_3(A)$
$R_1(A)$; $R_1(B)$; $W_3(A)$; $W_1(A)$; $W_1(B)$; $W_2(A)$

One blind-write between R1(A) and W1(A) makes it non-CS, the other at the end fixes it

e) Re-write the transaction schedule given below by adding commit (C) actions for each transaction such that the schedule would be recoverable should any of the commit actions be replaced by aborts.

$R_1(A)$; $W_2(A)$; $R_3(A)$; $W_3(A)$; $W_1(A)$

$R_1(A)$; $W_2(A)$; C_2; $R_3(A)$; $W_3(A)$; C_3; $W_1(A)$; C_1

There are others, but C3 must come after C2

f) Re-write the transaction schedule given below by adding commit (C) actions for each transaction such that the schedule would *not* be recoverable if (at least) one of the commit actions is replaced by an abort.

$R_1(A)$; $W_2(A)$; $R_3(A)$; $W_3(A)$; $W_1(A)$

$R_1(A)$; $W_2(A)$; $R_3(A)$; $W_3(A)$; $W_1(A)$; C_1; C_3; C_2

There are others. but C3 should precede C2

3 - Locking

Consider the schedule shown below, which is not *conflict serializable*.

$R_1(A)$; $R_3(A)$; $R_2(B)$; $W_2(B)$; $R_2(C)$; $R_1(C)$; $W_1(C)$; $W_2(C)$; $R_1(B)$; $W_1(B)$

a) Draw a precedence graph for the schedule. [**1** mark] T1 ←→ T2

b) Explain why the schedule is not conflict-serializable, by describing how actions cannot be swapped to result in a serial schedule. [**1** mark] T1's read and write of C cannot be moved to either the left or the right past T2s read or write of C. In addition, T1's read and write of B cannot be swapped to move to the left of T2's write and read of B, such that T1 cannot be moved to the front of the schedule.

c) Rewrite the schedule to show the schedule that would result under a *2PL* (not *Strict*) regime using shared and exclusive locks. Show shared and exclusive lock and unlock actions with *S*, *X* and *U*, a subscript that indicates the transaction, followed by the data object to which the lock applies in parentheses. For example, $S_1(A)$ indicates that transaction 1 is applying a shared lock to object *A*. Assume that objects are locked immediately before the first action on the object and unlocked as soon as possible. [**2** marks, -½ mark for each error, up to a maximum of 2]

$S_1(A)$; R1(A); ***$S_3(A)$***; R3(A); ***$U_3(A)$***;
$X_2(B)$; R2(B); W2(B); ***$X_2(C)$***; ***$U_2(B)$***; R2(C); **$S_1(C)$** not granted, ***T1 suspended***; W2(C); ***$U_2(C)$***;
$X_1(C)$; R1(C); W1(C); ***$X_1(B)$***; ***$U_1(A)$***; ***$U_1(C)$***; R1(B); W1(B); ***$U_1(B)$***;

d) Give an interleaved (i.e. non-serial) schedule of the transactions given below under a *Strict 2PL* regime that would *not* result in deadlock. Include the lock and unlock actions in your schedule. [**2** marks, -½ for each error, up to a maximum of 2]

T1: $R_1(A)$; $W_1(A)$; $R_1(B)$; $W_1(B)$
T2: $R_2(C)$; $W_2(C)$; $R_2(A)$
T3: $R_3(B)$; $R_3(C)$

Your schedule *must* begin with these actions:

$X_1(A)$; $R_1(A)$; $W_1(A)$; $X_2(C)$; $R_2(C)$; $W_2(C)$

$X_1(A)$; $R_1(A)$; $W_1(A)$; **$X_2(C)$**; $R_2(C)$; $W_2(C)$
$X_1(B)$; $R_1(B)$; $W_1(B)$; **$U_1(A)$**; **$U_1(B)$**; **$S_3(B)$**; $R_3(B)$;
$S_2(A)$; $R_2(A)$; **$U_2(C)$**; **$U_2(A)$**; **$S_3(C)$**; $R_3(C)$; **$U_3(B)$**; **$U_3(C)$**

There could be others

e) Give an interleaved schedule of the transactions shown in part (**d**) that *would* result in deadlock under a Strict 2PL regime. Include the lock actions in your schedule, ending with the lock actions that result in deadlock. [**2** marks, -½ for each error, up to a maximum of 2]

Your schedule *must* begin with these actions:

$X_1(A)$; $R_1(A)$; $W_1(A)$; $X_2(C)$; $R_2(C)$; $W_2(C)$

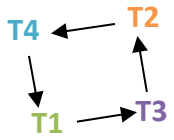
$X_1(A)$; $R_1(A)$; $W_1(A)$; $X_2(C)$; $R_2(C)$; $W_2(C)$
$S_3(B)$; $R_3(B)$; $X_1(B)$ not granted, ***T1 suspended***;
$S_2(A)$ not granted, ***T2 suspended***;
$S_3(C)$ not granted, ***T3 suspended***

There could be others

f, g and h) Consider the schedule shown below, which results in *deadlock*.

$R_1(A)$; $W_1(A)$; $W_2(B)$; $R_3(C)$; $R_4(D)$; $W_4(D)$; $W_2(D)$; $R_3(B)$; $R_4(A)$; $W_4(A)$; $R_1(C)$; $W_1(C)$

f) Draw the waits-for graph for the schedule [**2** marks]



g) Which transaction(s) would be aborted in a *wound-wait* scheme? Assume that priority is given to lower numbered transactions and that transactions commit as soon as they have performed their last action. [**1** mark]
T4
T2 wounds T4, completes and unlocks B, T3 completes, T1 completes

h) Which transaction(s) would be aborted in a *wait-die* scheme? Assume that priority is given to lower numbered transactions and that transactions commit as soon as they have performed their last action. [**1** mark]
T3 and T4
T2 waits, T3 dies, T4 dies, T1 completes, T2 completes

4 – Optimistic Concurrency Control

Describe what happens for each of the schedules (**a**) to (**d**) shown below. Note whether each transaction completes, is aborted or is suspended. Where a transaction is waiting (i.e. is suspended) explain what the transaction is waiting for and what will happen to the transaction based on results of the action it is waiting for. Assume that the commit bit for all data objects is true before the schedule commences. When reading a schedule, the first letter indicates the action (**S**tart-time, **R**ead, **W**rite or **C**ommit), the subscript indicates the transaction that the action belongs to (transaction 1, 2 or 3), and the letter in brackets indicates the data object that the transaction affects (A, B or D) [**3** marks each]

a) S_1; S_2; $R_1(A)$; $W_1(A)$; $R_2(B)$; $W_2(A)$; $W_2(B)$; $W_1(B)$

When processing W1(B), TS1 is less than RT(B) (which was read by T2) so T1 is rolled back [**3**]

b) S_1; S_2; $R_1(A)$; $W_1(A)$; $W_2(B)$; $R_1(B)$; $W_1(B)$

When processing R1(B), TS1 is less than WT(B) (which was written by T2) so T1 is rolled back [3]

c) S_1; S_2; $R_1(A)$; $W_1(A)$; $R_2(A)$; $W_2(A)$; $R_1(B)$; $W_1(B)$; $R_2(B)$; $W_2(B)$

When processing R2(A), C(A) is false (as T1 wrote A and has not committed). T2 is suspended [1]. T1 completes [1] and T2 is allowed to continue [1]

d) S_1; S_2; $R_1(A)$; $W_2(A)$; C_2; $W_1(A)$

When processing W1(A), TS1 is equal to RT(A) (which was read by T1) but less than WT(A) (which was written to by T2). Since C(A) is true W1(A) is ignored by the Thomas Write Rule [2] and both transactions complete [1]

Assessment

The assignment is out of 46. Marks are assigned as follows:

- Question 1 – 10
- Question 2 – 12
- Question 3 – 12
- Question 4 – 12

CMPT 454 Home

John Edgar (johnwill@sfu.ca)