Fall 2004 CS 186 Exercise Questions

Week 12 – ending 11/19

Transactions

1. Yes. For example consider the following schedule deadlocks under 2PL

T1: X-Lock(A) W(A) X-Lock(B)...

T2: X-Lock(B) W(B) X-Lock(A)...

Strict 2PL also has the deadlock problem, while conservative 2PL avoids it by requesting all the locks upfront.

2.

a) i. T1->T2, T2->T3, T1->T3.

ii. Yes equivalent schedules: T1 -> T2 -> T3.

b) i. T2->T1, T3->T1, T1->T2, T4->T2

ii. No there are cycles in the precedence graph (T2 -> T1, T1 -> T2)

3.

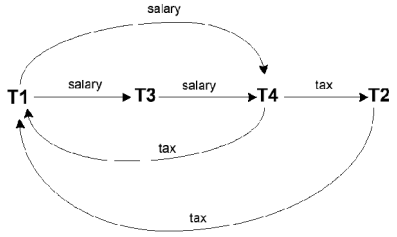
	2PL	**Necessarily conflict Serializable**	**Necessarily recoverable**	**Necessarily ACR**	**Necessarily Strict Schedule**	**Necessarily Serial Schedule**	**May Result in Deadlock**
a)	Y	Y	Y	Y	Y	N	Y
b)	Y	Y	Y	Y	Y	Y	N
c)	Y	Y	N	N	N	Y*	Y

*Any non-serial schedule will result in deadlock. Notice that a schedule like <L1(C); L2(B); ...L2 executes to the end; L1(A); ...L1 executes to the end> is (of course) legal but also serial since the actions of T1 never started. The locks are not part of the transaction, only the scheduler.

The schedule <L1(C); ... ; U1(B); L2(B); ... ;U2(B); CommitT1> (T1 executes but does not commit until after T2 is done) was considered for this question to be serial for a similar reason - we only asked you to look at the reads/write actions (i.e., un-committed reads were allowed), so a commit does not change the serializeability of the transactions.

4.

a)



b) None, the conflict graph has a cycle.
c) Same as above with t4 removed.
d) T2 T1 T3