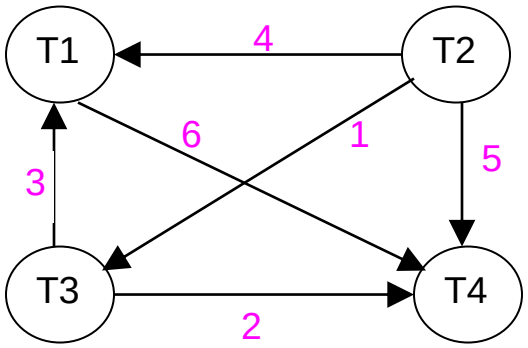
COP 4710 – Database Systems – Fall 2013

Homework #4 – 40 points – due November 26, 2013 **KEY**

Problem #1 (10 points)

Shown below is a concurrent schedule S of four transactions operating under a locking version of concurrency control, determine if S is serializable. If S is serializable produce a serial schedule which is equivalent to S. If a cycle exists, list the nodes in the cycle.

S = [(T2: Slock A) (T3: Slock A) (T2: Xlock B) (T2: Unlock A) (T3: Xlock A)
(T2: Unlock B) (T1: Slock B) (T3: Unlock A) (T4: Slock B) (T1: Slock A)
(T4: Unlock B) (T1: Xlock C) (T1: Unlock A) (T4: Xlock A) (T4: Unlock A)
(T1: Unlock B) (T1: Unlock C)]



Edges in graph

1: (T2: Slock A) - - - (T3: Xlock A)

2: (T3: Xlock A) - - - (T4: Xlock A)

3: (T3: Unlock A) - - - (T1: Slock A)

4: (T2: Unlock B) - - - (T1: Slock B)

5: (T2: Unlock B) - - - (T4: Slock B)

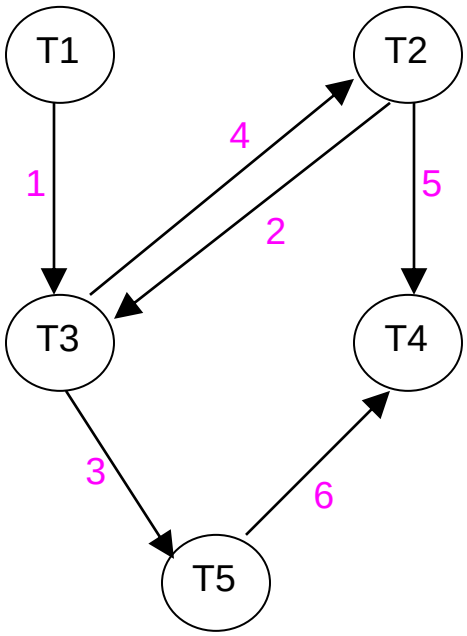
6: (T1: Slock A) - - - (T4: Xlock A)

No cycle in the graph. A serially equivalent schedule is:
(T2, T3, T1, T4)

Problem #2

Shown below is a concurrent schedule S of five transactions operating under a locking version of concurrency control, determine if S is serializable. If S is serializable produce a serial schedule which is equivalent to S. If a cycle exists, list the nodes in the cycle.

S = [(T1: Xlock A) (T2: Xlock B) (T1: Unlock A) (T3: Xlock A) (T4: Xlock C)
(T4: Xlock D) (T2: Unlock B) (T3: Unlock A) (T3: Xlock B) (T4: Xlock E)
(T5: Xlock A) (T3: Unlock B) (T2: Xlock B) (T2: Unlock B) (T4: Unlock C)
(T4: Unlock D) (T4: Unlock E) (T5: Unlock A) (T4: Xlock A) (T4: Unlock A)
(T4: Xlock B) (T4: Unlock B)]



Edges in graph

1: (T1: Unlock A) - - - (T3: Xlock A)
2: (T2: Unlock B) - - - (T3: Xlock B)
3: (T3: Unlock A) - - - (T5: Xlock A)
4: (T3: Unlock B) - - - (T2: Xlock B)
5: (T2: Unlock B) - - - (T4: Xlock B)
6: (T5: Unlock A) - - - (T4: Xlock A)

Cycle (T2 – T3 – T2), not serializable

Problem #3

Shown below is a concurrent schedule S of three transactions operating under a timestamp version of concurrency control, the timestamp of each transaction, and the initial read and write timestamp values for all of the objects referenced in S. Using the timestamping protocol, determine the action at each time instance of S that the protocol will take with respect to the transaction attempting the operation.

ts(T1) = 20	ts(T2) = 30	ts(T3) = 10
rts(A) = 10	rts(B) = 15	rts(C) = 0
wts(A) = 5	wts(B) = 0	wts(C) = 0

S = [(T1: read A) (T3: read C) (T1: write B) (T2: write A) (T2: write C) (T3: write C) (T3: write A) (T1: write B)]

(T1: read A) – OK, sets rts(A) = 20
(T3: read C) – OK, sets rts(C) = 10
(T1: write B) – OK, sets wts(B) = 20
(T2: write A) – OK, sets wts(A) = 30
(T2: write C) – OK, sets wts(C) = 30
(T3: write C) – ignore write since ts(T3) < wts(C)
(T3: write A) – rollback T3 since ts(T3) < rts(A)
– remainder of schedule is dependent on T3's rollback
(T1: write B) – OK, sets wts(B) = 20

Final timestamp values:

	A	B	C
rts	20	15	10
wts	30	20	30

Problem #4

Using a timestamping technique to prevent deadlock from occurring we presented two different protocols: "wait or die" and "wound or wait". Given the transaction timestamps: ts(T1) = 10, ts(T2) = 5, ts(T3) = 12, and ts(T4) = 6, determine the action of both protocols for each of the following scenarios.

(a) T1 requests an object whose lock is held by T3
(b) T2 requests an object whose lock is held by T1
(c) T4 requests an object whose lock is held by T2
(d) T4 requests an object whose lock is held by T3

Wait or Die protocol

(a) T1 is older than T3, so T1 waits
(b) T2 is older than T1, so T2 waits
(c) T4 is younger than T2, so T4 dies
(d) T4 is older than T3, so T4 waits

Wound or Wait protocol

(a) T1 is older than T3, so T1 “wounds” T3 and T3 dies
(b) T2 is older than T1, so T2 “wounds” T1 and T1 dies
(c) T4 is younger than T2, so T4 waits
(d) T4 is older than T3, so T4 “wounds” T3 and T3 dies