CHAPTER 21: CONCURRENCY CONTROL TECHNIQUES

Answers to Selected Exercises

21.20 - Prove that the basic two-phase locking protocol guarantees conflict serializability of schedules. (Hint: Show that, if a serializability graph for a schedule has a cycle, then at least one of the transactions participating in the schedule does not obey the two-phase locking protocol.)

Answer:
(This proof is by contradiction, and assumes binary locks for simplicity. A similar proof can be made for shared/exclusive locks.)
Suppose we have n transactions T1, T2, ..., Tn such that they all obey the basic two-phase locking rule (i.e. no transaction has an unlock operation followed by a lock operation). Suppose that a non-(conflict)-serializable schedule S for T1, T2, ..., Tn does occur; then, according to Section 17.5.2, the precedence (serialization) graph for S must have a cycle. Hence, there must be some sequence within the schedule of the form:
S: ...; [o1(X); ...; o2(X);] ...; [o2(Y); ...; o3(Y);] ... ; [on(Z); ...; o1(Z);]...
where each pair of operations between square brackets [o,o] are conflicting (either [w, w], or [w, r], or [r,w]) in order to create an arc in the serialization graph. This implies that in transaction T1, a sequence of the following form occurs:
T1: ...; o1(X); ... ; o1(Z); ...
Furthermore, T1 has to unlock item X (so T2 can lock it before applying o2(X) to follow the rules of locking) and has to lock item Z (before applying o1(Z), but this must occur after Tn has unlocked it). Hence, a sequence in T1 of the following form occurs:
T1: ...; o1(X); ...; unlock(X); ... ; lock(Z); ...; o1(Z); ...
This implies that T1 does not obey the two-phase locking protocol (since lock(Z) follows unlock(X)), contradicting our assumption that all transactions in S follow the two-phase locking protocol.

21.21 - Modify the data structures for multiple-mode locks and the algorithms for read_lock(X), write_lock(X), and unlock(X) so that upgrading and downgrading of locks are possible. (Hint: The lock needs to keep track of the transaction id(s) that hold the lock, if any.)

Answer:
We assume that a List of transaction ids that have read-locked an item is maintained, as well as the (single) transaction id that has write-locked an item. Only read_lock and write_lock are shown below.

```
read_lock (X, Tn):
B: if lock (X) = "unlocked"
then begin lock (X) <- "read_locked, List(Tn)";
no_of_reads (X) <- 1
end
else if lock(X) = "read_locked, List"
then begin
(* add Tn to the list of transactions that have read_lock on X *)
lock (X) <- "read_locked, Append(List,Tn)";
no_of_reads (X) <- no_of_reads (X) + 1
end
```

```
else if lock (X) = "write_locked, Tn"
(* downgrade the lock if write_lock on X is held by Tn itself *)
then begin lock (X) <- "read_locked, List(Tn)";
no_of_reads (X) <- 1
end
else begin
wait (until lock (X) = "unlocked" and
the lock manager wakes up the transaction);
goto B;
end;
write_lock (X,Tn);
B: if lock (X) = "unlocked"
then lock (X) <- "write_locked, Tn"
else
if ( (lock (X) = "read_locked, List") and (no_of_reads (X) = 1)
and (transaction in List = Tn) )
(* upgrade the lock if read_lock on X is held only by Tn itself *)
then lock (X) = "write_locked, Tn"
else begin
wait (until ( [ lock (X) = "unlocked" ] or
[ (lock (X) = "read_locked, List") and (no_of_reads (X) = 1)
and (transaction in List = Tn) ] ) and
the lock manager wakes up the transaction);
goto B;
end;
```

21.22 - Prove that strict two-phase locking guarantees strict schedules.

Answer:
Since no other transaction can read or write an item written by a transaction T until T has committed, the condition for a strict schedule is satisfied.

21.23 – No solution provided.

21.24 - Prove that cautious waiting avoids deadlock.

Answer:
In cautious waiting, a transaction Ti can wait on a transaction Tj (and hence Ti becomes blocked) only if Tj is not blocked at that time, say time b(Ti), when Ti waits. Later, at some time b(Tj) > b(Ti), Tj can be blocked and wait on another transaction Tk only if Tk is not blocked at that time. However, Tj cannot be blocked by waiting on an already blocked transaction since this is not allowed by the protocol. Hence, the wait-for graph among the blocked transactions in this system will follow the blocking times and will never have a cycle, and so deadlock cannot occur.

21.25 - Apply the timestamp ordering algorithm to the schedules of Figure 21.8 (b) and (c), and determine whether the algorithm will allow the execution of the schedules.

Answer:
Let us assume a clock with linear time points 0, 1, 2, 3, ..., and that the original read and write timestamps of all items are 0 (without loss of generality).
read_TS(X) = read_TS(Y) = read_TS(Z) = 0
write_TS(X) = write_TS(Y) = write_TS(Z) = 0
Let us call the schedules in Figure 17.8(b) Schedule E or SE, and that in Figure 17.8(c) Schedule F or SF. The two schedules can be written as follows in shorthand notation:
SE:
r2(Z); r2(Y); w2(Y); r3(Y); r3(Z); r1(X); w1(X); w3(Y); w3(Z); r2(X); r1(Y); w1(Y); w2(X);
1 2 3 4 5 6 7 8 9 10 11 12 13
SF:
r3(Y); r3(Z); r1(X); w1(X); w3(Y); w3(Z); r2(Z); r1(Y); w1(Y); r2(Y); w2(Y); r2(X); w2(X);
1 2 3 4 5 6 7 8 9 10 11 12 13
Assume that each operation takes one time unit, so that the numbers under the operations indicate the time when each operation occurred. Also assume that each transaction timestamp corresponds to the time of its first operations in each schedule, so the transaction timestamps are as follows (Note: These values do not change during the schedule, since they are assigned as unique identifiers to the transactions):
Schedule E Schedule F
TS(T1) = 6 TS(T1) = 3
TS(T2) = 1 TS(T2) = 7
TS(T3) = 4 TS(T3) = 1

(a) Applying timestamp ordering to Schedule E:
Initial values (new values are shown after each operation):
read_TS(X)=0,read_TS(Y)=0,read_TS(Z)=0,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
TS(T1)=6, TS(T2)=1, TS(T3)=4 (These do not change)
T2: read_item(Z)
TS(T2) > write_TS(Z)
Execute read_item(Z)
Set read_TS(Z) <- max(read_TS(Z),TS(T2)) = 1
read_TS(X)=0,read_TS(Y)=0,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T2: read_item(Y)
TS(T2) > write_TS(Y)
Execute read_item(Y)
Set read_TS(Y) <- max(read_TS(Y),TS(T2)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T2: write_item(Y)
TS(T2) = read_TS(Y) and TS(T2) > write_TS(Y)
Execute write_item(Y)
write_TS(Y) <- max(write_TS(Y),TS(T2)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=1,write_TS(Z)=0
T3: read_item(Y)
TS(T3) > write_TS(Y)
Execute read_item(Y)
read_TS(Y) <- max(read_TS(Y),TS(T3)) = 4
read_TS(X)=0,read_TS(Y)=4,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=1,write_TS(Z)=0
T3: read_item(Z)

TS(T3) > write_TS(Z)
Execute read_item(Z)
read_TS(Z) <- max(read_TS(Z),TS(T3)) = 4
read_TS(X)=0,read_TS(Y)=4,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=1,write_TS(Z)=0
T1: read_item(X)
TS(T1) > write_TS(X)
Execute read_item(X)
read_TS(X) <- max(read_TS(X),TS(T1)) = 6
read_TS(X)=6,read_TS(Y)=4,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=1,write_TS(Z)=0
T1: write_item(X)
TS(T1) = read_TS(X) and TS(T1) > write_TS(X)
Execute write_item(X)
write_TS(X) <- max(write_TS(X),TS(T1)) = 6
read_TS(X)=6,read_TS(Y)=4,read_TS(Z)=1,write_TS(X)=6,write_TS(Y)=1,write_TS(Z)=0
T3: write_item(Y)
TS(T3) = read_TS(Y) and TS(T3) > write_TS(Y)
Execute write_item(Y)
write_TS(Y) <- max(write_TS(Y),TS(T3)) = 4
read_TS(X)=6,read_TS(Y)=4,read_TS(Z)=1,write_TS(X)=6,write_TS(Y)=4,write_TS(Z)=0
T3: write_item(Z)
TS(T3) > read_TS(Z) and TS(T3) > write_TS(Z)
Execute write_item(Z)
write_TS(Z) <- max(write_TS(Z),TS(T3)) = 4
read_TS(X)=6,read_TS(Y)=4,read_TS(Z)=1,write_TS(X)=6,write_TS(Y)=4,write_TS(Z)=4
T2: read_item(X)
TS(T2) < write_TS(X)
Abort and Rollback Y2, Reject read_item(X)

Result: Since T3 had read the value of Y that was written by T2, T3 should also be aborted and rolled by the recovery technique (because of cascading rollback); hence, all effects of T2 and T3 would also be erased and only T1 would finish execution.

(b) Applying timestamp ordering to Schedule F:
Initial values (new values are shown after each operation):
read_TS(X)=0,read_TS(Y)=0,read_TS(Z)=0,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
TS(T1)=3, TS(T2)=7, TS(T3)=1 (These do not change)
T3: read_item(Y)
TS(T3) > write_TS(Y)
Execute read_item(Y)
Set read_TS(Y) <- max(read_TS(Y),TS(T3)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=0,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T3: read_item(Z)
TS(T3) > write_TS(Z)
Execute read_item(Z)
Set read_TS(Z) <- max(read_TS(Z),TS(T3)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T1: read_item(X)
TS(T1) > write_TS(X)
Execute read_item(X)
read_TS(X) <- max(read_TS(X),TS(T1)) = 3
read_TS(X)=3,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T1: write_item(X)
TS(T1) = read_TS(X) and TS(T1) > write_TS(X)

Execute write_item(X)
write_TS(X) <- max(write_TS(X),TS(T1)) = 3
read_TS(X)=3,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=3,write_TS(Y)=0,write_TS(Z)=0
T3: write_item(Y)
TS(T3) = read_TS(Y) and TS(T3) > write_TS(Y)
Execute write_item(Y)
write_TS(Y) <- max(write_TS(Y),TS(T3)) = 1
read_TS(X)=3,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=3,write_TS(Y)=1,write_TS(Z)=0
T3: write_item(Z)
TS(T3) = read_TS(Z) and TS(T3) > write_TS(Z)
Execute write_item(Z)
write_TS(Z) <- max(write_TS(Z),TS(T3)) = 1
read_TS(X)=3,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=3,write_TS(Y)=1,write_TS(Z)=1
T2: read_item(Z)
TS(T2) > write_TS(Z)
Execute read_item(Z)
Set read_TS(Z) <- max(read_TS(Z),TS(T2)) = 7
read_TS(X)=3,read_TS(Y)=1,read_TS(Z)=7,write_TS(X)=3,write_TS(Y)=1,write_TS(Z)=1
T1: read_item(Y)
TS(T1) > write_TS(Y)
Execute read_item(Y)
Set read_TS(Y) <- max(read_TS(Y),TS(T1)) = 3
read_TS(X)=3,read_TS(Y)=3,read_TS(Z)=7,write_TS(X)=3,write_TS(Y)=1,write_TS(Z)=1
T1: write_item(Y)
TS(T1) = read_TS(Y) and TS(T1) > write_TS(Y)
Execute write_item(Y)
write_TS(Y) <- max(read_TS(Y),TS(T1)) = 3
read_TS(X)=3,read_TS(Y)=3,read_TS(Z)=7,write_TS(X)=3,write_TS(Y)=3,write_TS(Z)=1
T2: read_item(Y)
TS(T2) > write_TS(Y)
Execute read_item(Y)
Set read_TS(Y) <- max(read_TS(Y),TS(T2)) = 7
read_TS(X)=3,read_TS(Y)=7,read_TS(Z)=7,write_TS(X)=3,write_TS(Y)=3,write_TS(Z)=1
T2: write_item(Y)
TS(T2) = read_TS(Y) and TS(T2) > write_TS(Y)
Execute write_item(Y)
write_TS(Y) <- max(write_TS(Y),TS(T2)) = 7
read_TS(X)=3,read_TS(Y)=7,read_TS(Z)=7,write_TS(X)=3,write_TS(Y)=7,write_TS(Z)=1
T2: read_item(X)
TS(T2) > write_TS(X)
Execute read_item(X)
Set read_TS(X) <- max(read_TS(X),TS(T2)) = 7
read_TS(X)=7,read_TS(Y)=7,read_TS(Z)=7,write_TS(X)=3,write_TS(Y)=3,write_TS(Z)=1
T2: write_item(X)
TS(T2) = read_TS(X) and TS(T2) > write_TS(X)
Execute write_item(X)
write_TS(X) <- max(write_TS(X),TS(T2)) = 7
read_TS(X)=7,read_TS(Y)=7,read_TS(Z)=7,write_TS(X)=7,write_TS(Y)=7,write_TS(Z)=1

Result: Schedule F executes successfully.

21.26 - Repeat Exercise 22.25, but use the multiversion timestamp ordering method.

Answer:
Let us assume the same timestamp values as in the solution for Exercise 18.22 above. To refer to versions, we use X, Y, Z to reference the original version (value) of each item, and then use indexes (1, 2, ...) to refer to newly written version (for example, X1, X2, ...).

(a) Applying multiversion timestamp ordering to Schedule E:
Initial values (new values are shown after each operation):
read_TS(X)=0,read_TS(Y)=0,read_TS(Z)=0,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
TS(T1)=6, TS(T2)=1, TS(T3)=4 (These do not change)
T2: read_item(Z)
Execute read_item(Z)
Set read_TS(Z) <- max(read_TS(Z),TS(T2)) = 1
read_TS(X)=0,read_TS(Y)=0,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T2: read_item(Y)
Execute read_item(Y)
Set read_TS(Y) <- max(read_TS(Y),TS(T2)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T2: write_item(Y)
TS(T2) = read_TS(Y)
Execute write_item(Y) (by creating a new version Y1 of Y)
write_TS(Y1) <- TS(T2) = 1,
read_TS(Y1) <- TS(T2) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Y1)=1,read_TS(Z)=1,
write_TS(X)=0,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0
T3: read_item(Y)
Execute read_item(Y) by reading the value of version Y1
read_TS(Y1) <- max(read_TS(Y1),TS(T3)) = 4
read_TS(X)=0,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Z)=1,
write_TS(X)=0,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0
T3: read_item(Z)
Execute read_item(Z)
read_TS(Z) <- max(read_TS(Z),TS(T3)) = 4
read_TS(X)=0,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Z)=4,
write_TS(X)=0,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0
T1: read_item(X)
Execute read_item(X)
read_TS(X) <- max(read_TS(X),TS(T1)) = 6
read_TS(X)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Z)=4,
write_TS(X)=0,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0
T1: write_item(X)
Execute write_item(X) (by creating a new version X1 of X)
write_TS(X) <- TS(T1) = 6,
read_TS(X) <- TS(T1) = 6
read_TS(X)=6,read_TS(X1)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Z)=4,
write_TS(X)=0,write_TS(X1)=6,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0
T3: write_item(Y)
Execute write_item(Y) (by creating a new version Y2 of Y)
write_TS(Y2) <- TS(T3) = 4,
read_TS(Y2) <- TS(T3) = 4
read_TS(X)=6,read_TS(X1)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Y2)=4,
read_TS(Z)=4,

write_TS(X)=0,write_TS(X1)=6,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=4,
write_TS(Z)=0
T3: write_item(Z)
Execute write_item(Z) (by creating a new version Z1 of Z)
write_TS(Z1) <- TS(T3) = 4,
read_TS(Z1) <- TS(T3) = 4
read_TS(X)=6,read_TS(X1)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Y2)=4,
read_TS(Z)=4,read_TS(Z1)=4,
write_TS(X)=0,write_TS(X1)=6,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=4,
write_TS(Z)=0,write_TS(Z1)=4
T2: read_item(X)
Execute read_item(X) by reading the value of the initial version X
read_TS(X) <- max(read_TS(X),TS(T3)) = 6
read_TS(X)=6,read_TS(X1)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Y2)=4,
read_TS(Z)=4,read_TS(Z1)=4,
write_TS(X)=0,write_TS(X1)=6,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=4,
write_TS(Z)=0,write_TS(Z1)=4
T1: read_item(Y)
Execute read_item(Y) by reading the value of version Y2
read_TS(Y2) <- max(read_TS(Y2),TS(T3)) = 6
read_TS(X)=6,read_TS(X1)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Y2)=6,
read_TS(Z)=4,read_TS(Z1)=4,
write_TS(X)=0,write_TS(X1)=6,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=4,
write_TS(Z)=0,write_TS(Z1)=4
T1: write_item(Y)
Execute write_item(Y) (by creating a new version Y3 of Y)
write_TS(Y3) <- TS(T3) = 4,
read_TS(Y2) <- TS(T3) = 4
read_TS(X)=6,read_TS(X1)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Y2)=6,
read_TS(Y3)=6,read_TS(Z)=4,read_TS(Z1)=4,
write_TS(X)=0,write_TS(X1)=6,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=4,
write_TS(Y2)=6,write_TS(Z)=0,write_TS(Z1)=4
T2: write_item(X)
Abort and Rollback T2 since read_TS(X) >TS(T2)

Result: Since T3 had read the value of Y that was written by T2, T3 should also be aborted and rolled by the recovery technique (because of cascading rollback); hence, all effects of T2 and T3 would also be erased and only T1 would finish execution.

(b) Applying timestamp ordering to Schedule F:
Initial values (new values are shown after each operation):
read_TS(X)=0,read_TS(Y)=0,read_TS(Z)=0,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
TS(T1)=3, TS(T2)=7, TS(T3)=1 (These do not change)
T3: read_item(Y)
Execute read_item(Y)
Set read_TS(Y) <- max(read_TS(Y),TS(T3)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=0,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T3: read_item(Z)
Execute read_item(Z)
Set read_TS(Z) <- max(read_TS(Z),TS(T3)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T1: read_item(X)
Execute read_item(X)

read_TS(X) <- max(read_TS(X),TS(T1)) = 3
read_TS(X)=3,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T1: write_item(X)
Execute write_item(X) by creating a new version X1 of X
write_TS(X1) <- TS(T1) = 3, read_TS(X1) <- TS(T1) = 3
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=1,read_TS(Z)=1,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Z)=0
T3: write_item(Y)
Execute write_item(Y) by creating a new version Y1 of Y
write_TS(Y1) <- TS(T3) = 1, read_TS(Y1) <- TS(T3) = 1
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=1,read_TS(Y1)=1,read_TS(Z)=1,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0
T3: write_item(Z)
Execute write_item(Z) by creating a new version Z1 of Z
write_TS(Z1) <- TS(T3) = 1, read_TS(Z1) <- TS(T3) = 1
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=1,read_TS(Y1)=1,read_TS(Z)=1,
read_TS(Z1)=1,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0,
write_TS(Z1)=1
T2: read_item(Z)
Execute read_item(Z) by reading the value of version Z1
Set read_TS(Z1) <- max(read_TS(Z1),TS(T2)) = 7
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=1,read_TS(Y1)=1,read_TS(Z)=1,
read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0,
write_TS(Z1)=1
T1: read_item(Y)
Execute read_item(Y) by reading the value of version Y1
Set read_TS(Y1) <- max(read_TS(Y1),TS(T1)) = 3
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=3,read_TS(Y1)=1,read_TS(Z)=1,
read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0,
write_TS(Z1)=1
T1: write_item(Y)
Execute write_item(Y) by creating a new version Y2 of Y
write_TS(Y2) <- TS(T1) = 3, read_TS(Y2) <- TS(T1) = 3
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=3,read_TS(Y1)=1,read_TS(Y2)=3,
read_TS(Z)=1,read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=3,
write_TS(Z)=0,write_TS(Z1)=1
T2: read_item(Y)
Execute read_item(Y) by reading the value of version Y2
Set read_TS(Y2) <- max(read_TS(Y2),TS(T2)) = 7
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=3,read_TS(Y1)=1,read_TS(Y2)=7,
read_TS(Z)=1,read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=3,
write_TS(Z)=0,write_TS(Z1)=1
T2: write_item(Y)
Execute write_item(Y) by creating a new version Y3 of Y
write_TS(Y3) <- TS(T2) = 7, read_TS(Y3) <- TS(T2) = 7
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=3,read_TS(Y1)=1,read_TS(Y2)=7,
read_TS(Y3)=7,read_TS(Z)=1,read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=3,

write_TS(Y3)=7,write_TS(Z)=0,write_TS(Z1)=1
T2: read_item(X)
Execute read_item(X) by reading the value of version X1
Set read_TS(X1) <- max(read_TS(X1),TS(T2)) = 7
read_TS(X)=3,read_TS(X1)=7,read_TS(Y)=3,read_TS(Y1)=1,read_TS(Y2)=7,
read_TS(Y3)=7,read_TS(Z)=1,read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=3,
write_TS(Y3)=7,write_TS(Z)=0,write_TS(Z1)=1
T2: write_item(X)
Execute write_item(X) by creating a new version X2 of X
write_TS(X2) <- TS(T2) = 7, read_TS(X2) <- TS(T2) = 7
read_TS(X)=3,read_TS(X1)=7,read_TS(X2)=7,read_TS(Y)=3,read_TS(Y1)=1,
read_TS(Y2)=7,read_TS(Y3)=7,read_TS(Z)=1,read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(X2)=7,write_TS(Y)=0,write_TS(Y1)=1,
write_TS(Y2)=3,write_TS(Y3)=7,write_TS(Z)=0,write_TS(Z1)=1

Result: Schedule F executes successfully.

21.27 – 21.29: No solutions provided.