CSC742 Homework 7
Solution

Ex. 1
21.28  b  
21.29  b  
21.30  b  
21.31  a  
21.32  c  
21.33  b  
21.34  c  
21.35  c  
21.36  b  
21.37  b

Ex. 2
• S1: T1:R(X), T2:W(X), T2:W(Y), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

1). Strict 2PL with timestamps used for deadlock prevention.

\[\text{Wait\_die:}\]

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read_lock (X)</td>
<td>Read_item (X)</td>
<td>TS (T2) &gt; TS (T1): die</td>
</tr>
<tr>
<td></td>
<td>Read_item (X)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TS (T1) &lt; TS (T3): wait</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{Wound\_wait:}\]
### 2). Strict 2PL with deadlock detection. (Show the waits-for graph if a deadlock cycle develops).

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read_lock (X)</td>
<td>TS (T2) &gt; TS (T1): wait</td>
<td>Write_lock (Y)</td>
</tr>
<tr>
<td>Read_item (X)</td>
<td></td>
<td>Write_item (Y)</td>
</tr>
<tr>
<td></td>
<td>TS (T1) &lt; TS (T3): wound T3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write_lock (Y)</td>
<td>Abort</td>
</tr>
<tr>
<td></td>
<td>Write_item (Y)</td>
<td>Unlock (Y)</td>
</tr>
<tr>
<td></td>
<td>Commit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unlock (X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unlock (Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write_lock (X)</td>
<td>Restart</td>
</tr>
<tr>
<td></td>
<td>Write_item (X)</td>
<td>Write_lock (Y)</td>
</tr>
<tr>
<td></td>
<td>Write_lock (Y)</td>
<td>Write_item (Y)</td>
</tr>
<tr>
<td></td>
<td>Write_item (Y)</td>
<td>Commit</td>
</tr>
<tr>
<td></td>
<td>Commit</td>
<td>Unlock (Y)</td>
</tr>
<tr>
<td></td>
<td>Unlock (X)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unlock (Y)</td>
<td></td>
</tr>
</tbody>
</table>
Unlock (Y)

3). Conservative (and strict, i.e., with locks held until end-of-transaction) 2PL.

T1 T2 T3
Read_lock (X) Read_lock (X) Write_lock (X)
Write_lock (Y) Write_lock (Y) Write_lock (Y)
Read_item (X) Write_item (X) Write_item (X)
Write_item (Y) Commit Commit
Unlock (X) Unlock (X) Unlock (X)
Unlock (Y) Unlock (Y) Unlock (Y)

4). Optimistic concurrency control.

T1 T2 T3
Read Phase Read Phase Read Phase
Read_item (X) Write_item (X) (Local Copy) Write_item (Y) (Local Copy)
Write_item (Y) (Local Copy) Write_item (Y) (Local Copy)
Validation (Success) Validation (Success)
Write Phase Write Phase Write Phase
Commit Commit Commit

5). Timestamp concurrency control with buffering of reads and writes (to ensure recoverability) and the Thomas Write Rule.

T1 T2 T3
Read_item (X) (Read_TS (X)=1)
Write_item (X) (Write_TS (X)=2)
Write_item (Y) (Write_TS (Y)=2)
Write_item(Y) (Write_TS(Y)=3)
(Ignore Write (Y))
Commit

Commit
Commit

6). Multiversion concurrency control.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read_item (X)</td>
<td>Write_item (X)</td>
<td>Write_item (Y)</td>
</tr>
<tr>
<td>(Read_TS (X1)=1)</td>
<td>(Read_TS (X2)=Write_TS (X2)=2)</td>
<td>(Read_TS (Y1)=Write_TS (Y1)=2)</td>
</tr>
<tr>
<td>Write_item (Y)</td>
<td>Write_item (Y)</td>
<td>Write_item (Y)</td>
</tr>
<tr>
<td>(Read_TS (Y3)=Write_TS (Y3)=1)</td>
<td>(Read_TS (Y2)=Write_TS (Y2)=3)</td>
<td></td>
</tr>
<tr>
<td>Commit</td>
<td>Commit</td>
<td>Commit</td>
</tr>
</tbody>
</table>

- S2: T1:R(X), T2:W(Y), T2:W(X), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

1. Strict 2PL with timestamps used for deadlock prevention.

Wait-die

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read_lock (X)</td>
<td>Write_lock (Y)</td>
<td>Write_lock (Y)</td>
</tr>
<tr>
<td>Read_item (X)</td>
<td>Write_item (Y)</td>
<td>Write_item (Y)</td>
</tr>
<tr>
<td>T1:R(X)</td>
<td>T2:W(Y)</td>
<td>T3:W(Y)</td>
</tr>
<tr>
<td>Abortion (T2)</td>
<td>Unlock (Y)</td>
<td>Unlock (Y)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS (T1) = TS (T2): die</td>
<td>TS (T2) = TS (T1): wait</td>
<td></td>
</tr>
</tbody>
</table>

TS (T1) < TS (T2): wait
Write_lock (Y)

Commit
Unlock (Y)

TS (T2) < TS (T1): wait
Write_lock (Y)

Commit
Unlock (Y)
Write_item (Y)
Commit
Unlock (X)
Unlock (Y)

Restart
Write_lock (Y)
Write_item (Y)
Write_lock (X)
Write_item (X)
Commit
Unlock (Y)
Unlock (X)

Wound-wait

T1
Read_lock (X)
Read_item (X)

T2
Write_lock (Y)
Write_item (Y)
TS (T2) > TS (T1): wait
TS (T3) > TS (T2): wait

T3
Write_lock (Y)
Write_item (Y)

TS (T1) < TS (T2): wound T2
Abort (T2)
Unlock (Y)

Write_lock (Y)
Write_item (Y)
Commit
Unlock (X)
Unlock (Y)

Write_lock (Y)
Write_item (Y)
Commit
Unlock (Y)
Unlock (X)

2. Strict 2PL with deadlock detection. (Show the waits-for graph if a deadlock cycle develops).

T1
Read_lock (X)
Read_item (X)

T2
Write_lock (Y)
Write_item (Y)
Wait

T3
Write_lock (Y)
Write_item (Y)
(Deadlock detected)

wait

Abort (T2)
Unlock (Y)

Write_lock (Y)
Write_item (Y)
Commit
Unlock (Y)

Write_lock (Y)
Write_item (Y)
Commit (T1)
Unlock (X)
Unlock (Y)

Restart
Write_lock (Y)
Write_item (Y)
Write_lock (X)
Write_item (X)
Commit
Unlock (Y)
Unlock (X)

3. Conservative (and strict, i.e., with locks held until end-of-transaction) 2PL.
Unlock (X)  
Write_lock (Y)  
Write_item (Y)  
Commit  
Unlock (Y)

4. Optimistic concurrency control.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Phase</td>
<td>Read Phase</td>
<td>Read Phase</td>
</tr>
<tr>
<td>Read_item(X)</td>
<td>Write_item (Y) (Local Copy)</td>
<td>Write_item (Y) (Local Copy)</td>
</tr>
<tr>
<td></td>
<td>Write_item (X) (Local Copy)</td>
<td>Write_item (X) (Local Copy)</td>
</tr>
<tr>
<td></td>
<td>Write_item (Y) (Local Copy)</td>
<td>Write_item (Y) (Local Copy)</td>
</tr>
<tr>
<td>Validation (Success)</td>
<td>Validation (Success)</td>
<td>Validation (Success)</td>
</tr>
<tr>
<td>Write Phase</td>
<td>Write Phase</td>
<td>Write Phase</td>
</tr>
<tr>
<td>Commit</td>
<td>Commit</td>
<td>Commit</td>
</tr>
</tbody>
</table>

5. Timestamp concurrency control with buffering of reads and writes (to ensure recoverability) and the Thomas Write Rule.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read_item (X) (Read_TS (X)=1)</td>
<td>Write_item (Y) (Write_TS (Y)=2)</td>
<td>Write_item (Y) (Write_TS(Y)=3)</td>
</tr>
<tr>
<td>Write_item (X) (Write_TS (X)=2)</td>
<td>(Ignore Write (Y))</td>
<td>Commit</td>
</tr>
<tr>
<td>Commit</td>
<td>Commit</td>
<td>Commit</td>
</tr>
</tbody>
</table>

6. Multiversion concurrency control.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read_item (X)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Ex. 3

1. Read record P1200: 5.
   - D: IS
   - F2: IS
   - P1200: IS
   - P1200: 5: S.

2. Read records P1200: 98 through P1250: 2.
   - D: IS
   - F2: IS
   - P1200: IS
   - P1250: IS
   - P1201 through P1249: S

3. Read all (records on all) pages in file F1.
   - D: IS
   - F1: S.

4. Read pages P500 through P520
   - D: IS
   - F1: IS
   - P500 through P520: S.

5. Read pages P10 through P980
   - D: IS
   - F1: S.

6. Read all pages in F1 and modify about 10 pages, which can be identified only after reading F1.
   - D: IS and IX
   - F1: SIX.

7. Delete record P1200: 98. (This is a blind write.)
   - D: IX
   - F2: IX
8. Delete the first record from each page. (Again, these are blind writes)
   D: IX
   F1: X
   F2: X.

9. Delete all records.
   D: IX
   F1: X
   F2: X

Ex. 4 (20.23)

In wait-die, transactions only wait on younger transactions. So no cycle is created in the wait-
for graph. Thus, there is no deadlock. Since a younger transaction requesting an item held by an
older transaction is aborted and restart with the same timestamp, starvation is avoided.

In wound-wait, transactions only wait on older transactions so no cycle is created. Thus,
there is no deadlock. An older transaction requesting an item held by an younger transaction
preempts the younger transaction by aborting it. The younger transaction restart with the same
timestamp, and starvation is avoided.

Ex. 5 (20.25)

For 19.8 (b)

Initial: 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

1. Read_TS (Z) = 1
2. Read_TS (Y) = 1
3. Write_TS (Y) = 1
4. Read_TS (Y) = 4
5. Read_TS (Z) = 4
6. Read_TS (X) = 6
7. Write_TS (X) = 6
8. Write_TS (Y) = 4
9. Write_TS (Z) = 4
10. Abort T2

The schedule is NOT allowed.

For 19.8 (c)

Initial: Read_TS (X) =0  Read_TS (Y) =0  Read_TS (Z) =0
The schedule is allowed.

Ex. 6 (20.26)

For 19.8 (b)

1. Read_TS (Z1) = 1
2. Read_TS (Y1) = 1
3. Write_TS (Y2) = 1
   Read_TS (Y2) = 1
4. Read_TS (Y2) = 4
5. Read_TS (Z1) = 4
6. Read_TS (X1) = 6
7. Write_TS (X2) = 6
   Read_TS (X2) = 6
8. Read_TS (Y3) = 4
   Write_TS (Y3) = 4
9. Read_TS (Z2) = 4
   Write_TS (Z2) = 4
10. Read_TS (X1) = 6
11. Read_TS (Y3) = 6
12. Write_TS (Y4) = 6
   Read_TS (Y4) = 6
13. Abort (T2)

The schedule is NOT allowed.

For 19.8 (c)

1. Read_TS (Y1) = 1
2. Read_TS (Z1) = 1
3. Read_TS (X1) = 3
4. Write_TS (X2) = 3
   Read_TS (X2) = 3
5. Write_TS (Y2) = 1
   Read_TS (Y2) = 1
6. Write_TS (Z2) = 1
   Read_TS (Z2) = 1
7. Read_TS (Z2) = 7
8. Read_TS (Y2) = 3
9. Write_TS (Y3) = 3
   Read_TS (Y3) = 3
10. Read_TS (Y3) = 7
11. Write_TS (Y4) = 7
    Read_TS (Y4) = 7
12. Read_TS (X2) = 7
13. Write_TS (X3) = 7
    Read_TS (X3) = 7

The schedule is allowed.