AML140830

|  |  |
| --- | --- |
| **HOMEWORK 6** | **Transaction Processing** |
| **Due Wed, Nov 11 at 11:30 pm** | **Objectives: Introduction to Transaction Processing Concepts** |

**20.16.** Add the operation commit at the end of each of the transactions *T*1 and *T*2 in Figure 20.2, and then list all possible schedules for the modified transactions. Determine which of the schedules are recoverable, which are cascadeless, and which are strict.

(5+3)! / (5! \* 3!) = 8\*7\*6\*5\*4\*3\*2\*1/ 5\*4\*3\*2\*1\*3\*2\*1 = 56.

You don’t need to list all 56 possible schedules; only list 2 strict, 2 recoverable, 2 non-recoverable, and 2 cascadeless schedules.

Below are the 56 possible schedules, and the type of each schedule:

S 1 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; r 2 (X); w 2 (X); C 2 ; strict

S 21 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; w 2 (X); C 2 ; strict

S 2 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); C 1 ; w 2 (X); C 2 ; recoverable

S 3 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); w 2 (X); C 1 ; C 2 ; recoverable

S 9 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); w 1 (Y); C 2 ; C 1 ; non-recoverable

S 10 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); C 2 ; w 1 (Y); C 1 ; non-recoverable

S 22 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); C 1 ; C 2 ; cascadeless

S 23 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); C 2 ; C 1 ; cascadeless

**20.17.** List all possible schedules for transactions *T*1 and *T*2 in Figure 20.2, and determine which are conflict serializable (correct) and which are not.

Below are the 15 possible schedules, and the type of each schedule:

S 1 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); w 2 (X); serial (and hence also serializable)

S 2 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 1 (Y); w 2 (X); (conflict) serializable

S 3 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); w 1 (Y); (conflict) serializable

S 4 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 1 (Y); w 2 (X); (conflict) serializable

S 5 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 2 (X); w 1 (Y); (conflict) serializable

S 6 : r 1 (X); w 1 (X); r 2 (X); w 2 (X); r 1 (Y); w 1 (Y); (conflict) serializable

S 7 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); not (conflict) serializable

S 8 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); not (conflict) serializable

S 9 : r 1 (X); r 2 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); not (conflict) serializable

S 10 : r 1 (X); r 2 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); not (conflict) serializable

S 11 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); not (conflict) serializable

S 12 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); not (conflict) serializable

S 13 : r 2 (X); r 1 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); not (conflict) serializable

S 14 : r 2 (X); r 1 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); not (conflict) serializable

S 15 : r 2 (X); w 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); serial (and hence also serializable)

**20.23.** Consider the three transactions *T*1, *T*2, and *T*3, and the schedules *S*1 and *S*2 given below.

Draw the serializability (precedence) graphs for *S*1 and *S*2, and state whether each schedule is serializable or not. If a schedule is serializable, write down the equivalent serial schedule(s).

*T*1: *r*1 (*X*); *r*1 (*Z*); *w*1 (*X*);

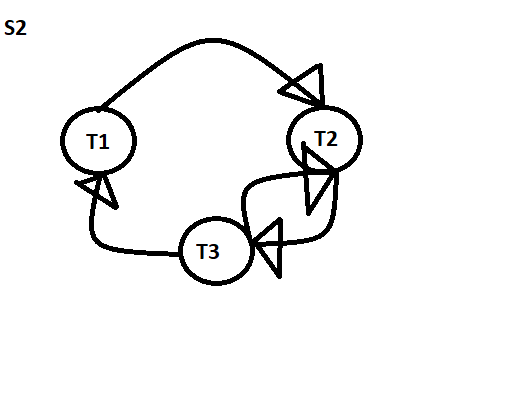
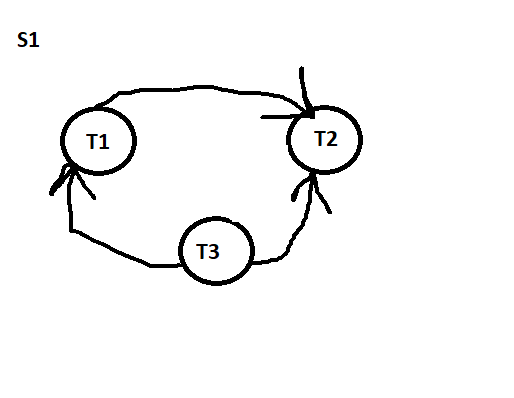
*T*2: *r*2 (*Z*); *r*2 (*Y*); *w*2 (*Z*); *w*2 (*Y*);

*T*3: *r*3 (*X*); *r*3 (*Y*); *w*3 (*Y*);

*S*1: *r*1 (*X*); *r*2 (*Z*); *r*1 (*Z*); *r*3 (*X*); *r*3 (*Y*); *w*1 (*X*); *w*3 (*Y*); *r*2 (*Y*); *w*2 (*Z*); *w*2 (*Y*);

*S*2: *r*1 (*X*); *r*2 (*Z*); *r*3 (*X*); *r*1 (*Z*); *r*2 (*Y*); *r*3 (*Y*); *w*1 (*X*); *w*2 (*Z*); *w*3 (*Y*); *w*2 (*Y*);

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20.23 |  |  |  | S1 |  |  |  | S2 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| T1 | T2 | T3 |  | T1 | T2 | T3 |  | T1 | T2 | T3 |
| R(x) | R(z) | R(x) |  | R(x) |  |  |  | R(x) |  |  |
| R(z) | R(y) | R(y) |  |  | R(z) |  |  |  | R(z) |  |
| W(x) | W(z) | W(y) |  | R(z) |  |  |  |  |  | R(x) |
|  | W(y) |  |  |  |  | R(x) |  | R(z) |  |  |
|  |  |  |  |  |  | R(y) |  |  | R(y) |  |
|  |  |  |  | W(x) |  |  |  |  |  | R(y) |
|  |  |  |  |  |  | W(y) |  | W(x) |  |  |
|  |  |  |  |  | R(y) |  |  |  | W(z) |  |
|  |  |  |  |  | W(z) |  |  |  |  | W(y) |
|  |  |  |  |  | W(y) |  |  |  | W(y) |  |



S1is conflict serializable because there is NOT a cycle.

S2 is NOT conflict serializable because there is a cycle.

**20.24.** Consider schedules *S*3, *S*4, and *S*5 below. Determine whether each schedule is strict, cascadeless, recoverable, or nonrecoverable. (Determine the strictest recoverability condition that each schedule satisfies.)

*S*3: *r*1 (*X*); *r*2 (*Z*); *r*1 (*Z*); *r*3 (*X*); *r*3 (*Y*); *w*1 (*X*); *c*1; *w*3 (*Y*); *c*3; *r*2 (*Y*); *w*2 (*Z*); *w*2 (*Y*); *c*2;

*S*4: *r*1 (*X*); *r*2 (*Z*); *r*1 (*Z*); *r*3 (*X*); *r*3 (*Y*); *w*1 (*X*); *w*3 (*Y*); *r*2 (*Y*); *w*2 (*Z*); *w*2 (*Y*); *c*1; *c*2; *c*3;

*S*5: *r*1 (*X*); *r*2 (*Z*); *r*3 (*X*); *r*1 (*Z*); *r*2 (*Y*); *r*3 (*Y*); *w*1 (*X*); *c*1; *w*2 (*Z*); *w*3 (*Y*); *w*2 (*Y*); *c*3; *c*2;

**Strict schedule**: A schedule is strict if it satisfies the following conditions:

1. Tj reads a data item X ***after*** Ti has written to X and Ti is terminated (aborted or committed)
2. Tj writes a data item X ***after*** Ti has written to X and Ti is terminated (aborted or committed)

S3

* **is not strict** because T3 reads X (r3(X)) ***before*** T1 has written to X (w1(X)) but T3 commits ***after*** T1

S4

* **is not strict** because T3 reads X (r3(X)) ***before*** T1 has written to X (w1(X))but T3 commits ***after*** T1.

S5

* **is not strict** because T3 reads X (r3(X)) ***before*** T1 has written to X (w1(X))but T3 commits ***after*** T1.

**Cascadeless schedule**: A schedule is cascadeless if the following condition is satisfied:

1. Tj reads X only ***after*** Ti has written to X and terminated (aborted or committed).

S3

* is ***not cascadeless*** because T3 reads X (r3(X)) before T1 commits.

S4

* is ***not cascadeless*** because T3 reads X (r3(X)) before T1 commits.

S5

* is ***not cascadeless*** because T3 reads X (r3(X)) ***before*** T1 commits or T2 readsY (r2(Y)) ***before*** T3 commits.

**Recoverable schedule**: A schedule is recoverable if the following condition is satisfied:

1. Tj commits after Ti if Tj has read any data item written by Ti.
2. Abort operations will be used in place of commits, one at a time
3. Notations Used:
   1. Commit Notation: Ci > Cj means Ci happens ***before*** Cj.
   2. Abort Notation: Ai denotes abort Ti.
   3. Strictness:
      1. a transaction neither reads nor writes to a data item, which was written to by a transaction that has not committed yet.

S3.

* If A1>C3>C2, then S3 is ***recoverable*** because rolling back of T1 does not affect T2 and T3.
* If C1>A3>C2. S3 is ***not recoverable*** because T2 read the value of Y (r2(Y)) ***after*** T3 wrote X (w3(Y)) and T2 committed but T3 rolled back. Thus, T2 used non- existent value of Y.
* If C1>C3>A3, then S3 is ***recoverable*** because roll back of T2 does not affect T1 and T3.
* Strictest condition of S3 is C3>C2.

S4

* If A1>C2>C3, then S4 is ***recoverable*** because roll back of T1 does not affect T2 and T3.
* If C1>A2>C3, then S4 is ***recoverable*** because the roll back of T2 will restore the value of Y that was read and written to by T3 (w3(Y)). It will not affect T1.
* If C1>C2>A3, then S4 is ***not recoverable*** because T3 will restore the value of Y which was not read by T2.
* Strictest condition of S4 is C3>C2, but it is not satisfied by S4.

S5

* If A1>C3>C2, then S5 is ***recoverable*** because neither T2 nor T3 writes to X, which is written by T1. If C1>A3>C2, then S5 is ***not recoverable*** because T3 will restore the value of Y, which was not read by T2. Thus, T2 committed with a non-existent value of Y.
* If C1>C3>A2, then S5 is ***recoverable*** because it will restore the value of Y to the value, which was read by T3. Thus, T3 committed with the right value of Y.
* Strictest condition of S3 is C3>C2, but it is not satisfied by S5.