|  |  |
| --- | --- |
|  | **BAHRIA UNIVERSITY**  **(Karachi Campus)**  *Department of Software Engineering*  **ASSIGNMENT#03 – Spring 2023**  **COMPLEX ENGINEERING PROBLEM**  Based on CLO-3 |

|  |  |  |  |
| --- | --- | --- | --- |
| COURSE TITLE: | **Database Management System** | COURSE CODE: | **CSC-220** |
| Class: | **BSE – 4A & B** | Shift: | **Morning** |
| Course Instructor: | **ENGR. LARAIB SIDDIQUI** | Max. Marks: | **06 Marks** |

Assignment Date : 6th **JUNE 2023** Due Date : 13th **JUNE 2023**

This assignment is based on the following CEP attributes:

* Cannot be resolved without in-depth engineering knowledge.
* Have no obvious solution and require abstract thinking and originality in analysis to formulate suitable models.

Question

Consider three transactions given below to create one serial schedule and two concurrent schedules with different time slices. Also, assume different orders for the transaction’s timestamps e.g. the order T1 < T2 < T3 < T4 (T1 is the oldest). Then check for view serializability and conflict serializability.

* + T1 : I1 R(Y) I2 R(Z) I3 R(B) I4 B:=B+(Y\*.1)+Z\*.2 I5 Y:=Y-Y\*.1 I6 Z:=Z-Z\*.2 I7 W(B) I8 W(Y) I9 W(Z)
  + T2 : I1 R(D) I2 R(Y) I3 D:=D-50 I4 W(D) I5 Y:=Y+50 I6 W(Y)
  + T3 : I1R(Z) I2 R(D) I3 Z:=Z-5 I4 W(Z) I5 D:=D+5 I6 W(D)

a) Apply shared lock (Ls), exclusive lock (Lx) and unlock (Us or Ux) on any one of the schedules produced in part a) for T1, T2 and T3 on database items B, D, Y and Z. Then simulate how these transactions are executed under the wait-die and wound-wait prevention strategy. For the simulation, assume that these transactions are executed in a round-robin fashion.

When it is a transaction’s turn, it executes its next lock or unlock step if it can, and otherwise dies or waits or wounds the holder of the lock, as appropriate. When a waiting

transaction is restarted (due to the action of some other transaction) it becomes eligible to run at its very next turn.

Make a table to show the sequence of steps that these transactions make under each policy. Use Lx(Y) to denote exclusive locking an object Y, Ux(Y) to denote unlocking Y, similarly Ls(Y) and Us(Y) for locking and unlocking shared lock on Y, “Die” to denote the transaction dying, “Wound” being wounded, and “Wait” to denote the transaction waiting. If transaction A is wounded by some other transaction B, transaction A’s next action is ’Die’. In the grid, the cell for [i, j] (row i column Tj ) represents the i th action of Tj . We have filled out a few cells to illustrate how you should fill out the rest of the table if cycle is for two instructions. Simulate actions until all three transactions complete. If a transaction is already complete, its turn is skipped.

|  |  |  |  |
| --- | --- | --- | --- |
| T1 | T2 | T3 | T4 |
| Ls(Y) |  |  |  |
| R(Y) |  |  |  |
|  | Ls(D) |  |  |
|  | R(D) |  |  |
|  |  | Lx(Z) |  |
|  |  | W(Z) |  |

Execute for both schedules and apply both wait-die and wound-wait strategy. Also explain which is better and why.

Solution:

**Wait Die for Timestamp 3**

|  |  |  |
| --- | --- | --- |
| **T1** | **T2** | **T3** |
| **Lx(Y)** |  |  |
| **R(Y)** |  |  |
| **Lx(Z)** |  |  |
|  | Lx(D) |  |
|  | R(D) |  |
|  | LX(Y) |  |
|  | Die |  |
|  |  | LX(Z) |
|  |  | Die |
| **R(Z)** |  |  |
| **LX(B)** |  |  |
| **R(B)** |  |  |
|  | Lx(D) |  |
|  | R(D) |  |
|  | LX(Y) |  |
|  | Die |  |
|  |  | LX(Z) |
|  |  | Die |
| **W(B)** |  |  |
| **UX(B)** |  |  |
| **W(Y)** |  |  |
|  | Lx(D) |  |
|  | R(D) |  |
|  | LX(Y) |  |
|  | Die |  |
|  |  | LX(Z) |
|  |  | Die |
| UX(Y) |  |  |
| W(Z) |  |  |
| UX(Z) |  |  |
| Commit |  |  |
|  | Lx(D) |  |
|  | R(D) |  |
|  | LX(Y) |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | W(Y) |  |
|  | Ux(D) |  |
|  | UX(Y) |  |
|  | Commit |  |
|  |  | LX(Z) |
|  |  | W(Z) |
|  |  | LX(D) |
|  |  | W(D) |
|  |  | UX(Z) |
|  |  | UX(D) |
|  |  | Commit |

**Wound Wait for Timestamp 3**

|  |  |  |
| --- | --- | --- |
| **T1** | **T2** | **T3** |
| **Lx(Y)** |  |  |
| **R(Y)** |  |  |
| **Lx(Z)** |  |  |
|  | LS(D) |  |
|  | R(D) |  |
|  | LX(Y) |  |
|  | Wait |  |
|  |  | LX(Z) |
|  |  | Wait |
|  |  |  |
|  |  |  |
| **R(Z)** |  |  |
| **LX(B)** |  |  |
| **R(B)** |  |  |
|  | wait |  |
|  |  | Wait |
|  |  |  |
| **W(B)** |  |  |
| **UX(B)** |  |  |
| **W(Y)** |  |  |
| **US(Y)** |  |  |
| **W(Z)** |  |  |
|  | W(Y) |  |
|  | US(D) |  |
|  | UX(Y) |  |
|  | Commit |  |
|  |  | Wait |
| **Ux(Z)** |  |  |
| **Commit** |  |  |
|  |  | W(Z) |
|  |  | LX(D) |
|  |  | W(D) |
|  |  | UX(Z) |
|  |  | UX(D) |
|  |  | Finish |

**Wound wait for Timestamp 4 AND order T3 < T2 < T1**

|  |  |  |
| --- | --- | --- |
| **T3** | **T2** | **T1** |
| LX(Z) |  |  |
| W(Z) |  |  |
| LX(D) |  |  |
| W(D) |  |  |
|  | LS(D) |  |
|  | Wait |  |
|  |  | Lx(Y) |
|  |  | R(Y) |
|  |  | Lx(Z) |
|  |  | Wait |
| UX(Z) |  |  |
| UX(D) |  |  |
| Commit |  |  |
|  | R(D) |  |
|  | LX(Y) |  |
|  | Wounds T1 | Wounded |
|  | W(Y) |  |
|  | Us(D) |  |
|  |  | Lx(Y) |
|  |  | Wait |
|  | UX(Y) |  |
|  | Commit |  |
|  |  | R(Y) |
|  |  | Lx(Z) |
|  |  | R(Z) |
|  |  | LX(B) |
|  |  | R(B) |
|  |  | W(B) |
|  |  | UX(B) |
|  |  | Ux(Y) |
|  |  | Ux(Z) |
|  |  | Commit |
|  |  |  |

The method to avoid a deadlock is far more effective than waiting. The younger transactions usually fail because the necessary resource is unavailable or is being consumed by the older transactions. The younger or newer transaction can, however, wait until the older transaction releases the resource in wound wait. The older transaction may cause issues for the younger ones when the latter needs a resource that is already in use by the former. However, the wound wait standstill prevention strategy is more successful since this happens less frequently.