University of Central Florida
CGS 2545
Database Concepts

DEPARTMENT OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE
COMPUTER SCIENCE DIVISION

Deadlock

- In a multi-process system, deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.
- Deadlocks are not healthy for a system.
- In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

Deadlock

- For example
  - assume a set of transactions {T_0, T_1, T_2, … T_n}
  - T_0 needs a resource X to complete its task
  - Resource X is held by T_1 and T_1 is waiting for a resource Y, which is held by T_2
  - T_2 is waiting for resource Z, which is held by T_0
  - Thus, all the processes wait for each other to release resources.
  - In this situation, none of the processes can finish their task.
  - This situation is known as a deadlock.

Deadlock

- Deadlock Prevention
  - To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute.
  - The DBMS inspects the operations and analyzes if they can create a deadlock situation.
  - If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.
  - There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

Deadlock

- Deadlock Prevention
  - Wait-Die Scheme
    - In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur
      - If $TS(T_i) < TS(T_j)$ that is T_i which is requesting a conflicting lock, is older than T_j then T_i is allowed to wait until the data-item is available

Deadlock

- Deadlock Prevention
  - Wait-Die Scheme
    - If $TS(T_i) > TS(T_j)$ that is younger than T_j then T_i dies. T_i restarted later with a random delay but with the same timestamp.
      - This scheme allows the older transaction to wait but kills the younger one

Deadlock

- Deadlock Prevention
  - Wound-Wait Scheme
    - In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by some another transaction, one of the two possibilities may occur
      - If $TS(T_i) < TS(T_j)$ then T_i forces T_j to be rolled back, that is T_i wounds T_j. T_j is restarted later with a random delay but with the same timestamp

Deadlock

- Deadlock Prevention
  - Wound-Wait Scheme
    - If $TS(T_i) > TS(T_j)$ then T_i is forced to wait until the resource is available
      - This scheme, allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item.
      - In both the cases, the transaction that enters the system at a later stage is aborted.

Deadlock

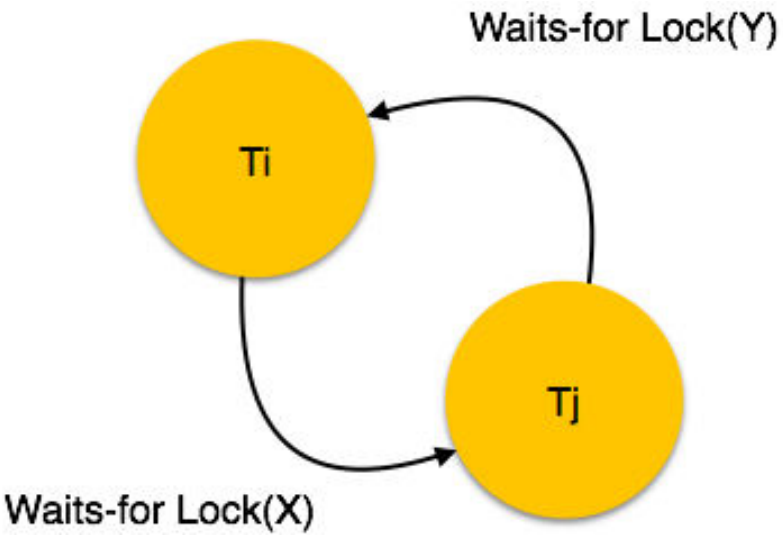
- Deadlock Avoidance
  - Aborting a transaction is not always a practical approach.
  - Instead, deadlock avoidance mechanisms can be used to detect any deadlock situation in advance.
  - Methods like "wait-for graph" are available but they are suitable for only those systems where transactions are lightweight having fewer instances of resource. In a bulky system, deadlock prevention techniques may work well.

Deadlock

- Deadlock Avoidance
  - Wait-for Graph
    - This is a simple method available to track if any deadlock situation may arise.
    - For each transaction entering into the system, a node is created.
    - When a transaction T_i requests for a lock on an item, say X, which is held by some other transaction T_j a directed edge is created from T_i to T_j
    - If T_j releases item X, the edge between them is dropped and T_i locks the data item

Deadlock

- Deadlock Avoidance
  - Wait-for Graph
    - The system maintains this wait-for graph for every transaction waiting for some data items held by others.
    - The system keeps checking if there's any cycle in the graph.



Deadlock

- Deadlock Avoidance
  - Wait-for Graph
    - Here, we can use any of the two following approaches –
      - First, do not allow any request for an item, which is already locked by another transaction.
      - This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.

Deadlock

- Deadlock Avoidance
  - Wait-for Graph
    - The second option is to roll back one of the transactions.
    - It is not always feasible to roll back the younger transaction, as it may be important than the older one.
    - With the help of some relative algorithm, a transaction is chosen, which is to be aborted.
    - This transaction is known as the **victim** and the process is known as **victim selection**.