DBMS Deadlock

In a multi-process system, deadlock is a situation, which arises in shared resource environment where a process indefinitely waits for a resource, which is held by some other process, which in turn waiting for a resource held by some other process.

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 processes wait for each other to release resources. In this situation, none of processes can finish their task. This situation is known as 'deadlock'.

Deadlock is not a good phenomenon for a healthy system. To keep system deadlock free few methods can be used. In case the system is stuck because of deadlock, either the transactions involved in deadlock are rolled back and restarted.

Deadlock Prevention

To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations which transactions are about to execute. DBMS inspects operations and analyze 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, which uses time-stamp ordering mechanism of transactions in order to pre-decide a deadlock situation.

WAIT-DIE SCHEME:

In this scheme, if a transaction request to lock a resource (data item), which is already held with conflicting lock by some other transaction, 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, T_i is allowed to wait until the data-item is available.
- If $TS(T_i) > TS(t_j)$, that is T_i is younger than T_j, T_i dies. T_i is restarted later with random delay but with same timestamp.

This scheme allows the older transaction to wait but kills the younger one.

WOUND-WAIT SCHEME:

In this scheme, if a transaction request to lock a resource (data item), which is already held with conflicting lock by some other transaction, 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, T_i forces T_j to be rolled back, that is T_i wounds T_j. T_j is restarted later with random delay but with same timestamp.
- If $TS(T_i) > TS(T_j)$, that is T_i is younger than T_j, T_i is forced to wait until the resource is available.

This scheme, allows the younger transaction to wait but when an older transaction request an item held by younger one, the older transaction forces the younger one to abort and release the item.

In both cases, transaction, which enters late in the system, is aborted.

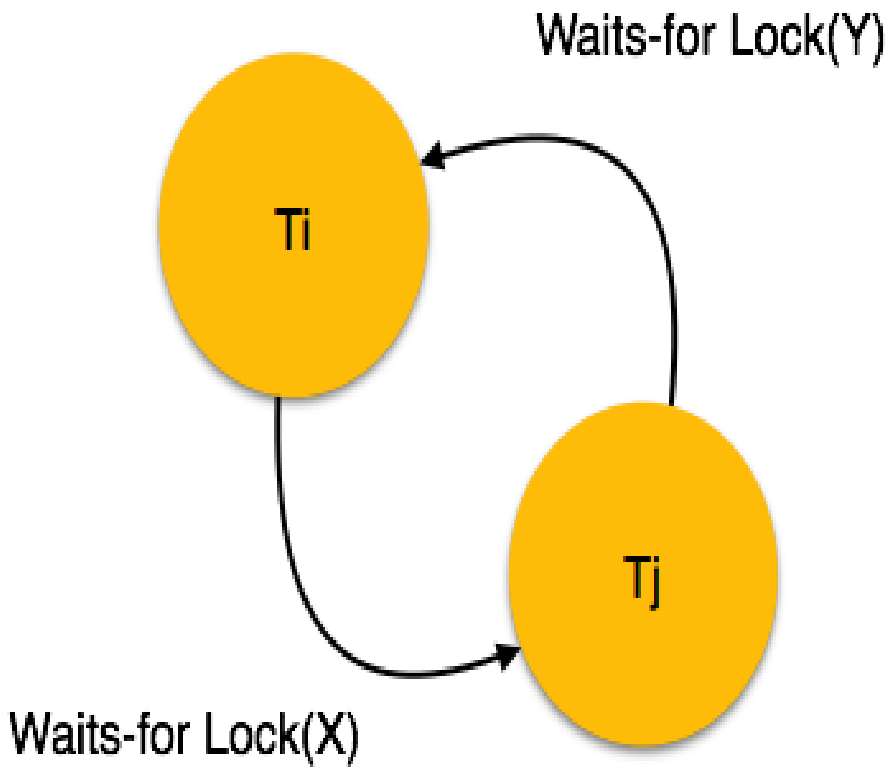
Deadlock Avoidance

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 for the system where transactions are light in weight and have hold on fewer instances of resource. In a bulky system deadlock prevention techniques may work well.

WAIT-FOR GRAPH

This is a simple method available to track if any deadlock situation may arise. For each transaction entering in the system, a node is created. When transaction T_i requests for a lock on 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.

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



[*Image: Wait-for Graph*]

Two approaches can be used, first not to allow any request for an item, which is already locked by some other transaction. This is not always feasible and may cause starvation, where a transaction indefinitely waits for data item and can never acquire it. Second option is to roll back one of the transactions.

It is not feasible to always roll back the younger transaction, as it may be important than the older one. With help of some relative algorithm a transaction is chosen, which is to be aborted, this transaction is called victim and the process is known as ***victim selection.***

Source:

http://www.tutorialspoint.com/dbms/dbms_deadlock.htm