© 2014 IJIRT | Volume 1 Issue 5 | ISSN : 2349-6002

DEADLOCK DETECTION AND PREVENTION

Manish Gahlot, Jatin Dadhwal
Department of Information Technology
Dronacharya College of Engineering, Khentawas, Farukh Nagar, Gurgaon, Haryana, India

Abstract-- **A deadlock is a situation in which two or more actions are each waiting for the other to finish, and thus neither ever does. Deadlock is a result of some uncontrolled sequence and request of resources among processes in a distributed system. This paper presents some system models and deadlock handling techniques to deal with the problem. Also there are various algorithms presented to see how deadlocks can be detected.**

Index Terms- **Deadlock Local Transaction Structure, Global transaction Structure, Distributed database system**

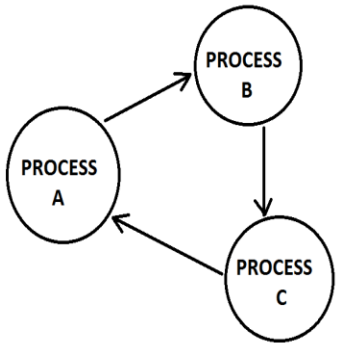
I. INTRODUCTION

A deadlock is a circumstance in which two or more activities are holding up for one another to complete, and consequently not one or the other ever does.

In a transaction database, a deadlock happens when two or more methods inside it transaction overhauls two lines of data yet in the inverse request. Case in point, first process named as process A redesigns column 1 then line 2. In the same time period second process named as methodology B upgrades push 2 then column 1. Process A can't get done with redesigning line 2 until procedure B is done, furthermore transform B can't get done with upgrading column 1 until methodology A completions. Regardless of the amount time is permitted to pass, this circumstance can't illuminate itself and this will regularly kill the transaction of the procedure which ever has done the minimum measure of work.

In Working Framework, a deadlock is a circumstance which happens when a procedure enters a holding up state on the grounds that an asset asked for is continuously held by an alternate holding up methodology, which thus is sitting tight for an alternate asset.

In Operating System, a deadlock is a circumstance which happens when a methodology enters a holding up state on the grounds that an asset asked for is continuously held by an alternate holding up procedure, which thus is sitting tight for an alternate asset. In the event that a procedure is not able to transform its state inconclusively in light of the fact that the assets asked for by it are continuously utilized by an alternate holding up methodology, then the framework is said to be in a stop.



The above diagram shows the deadlock situation

As we can see in the above diagram that Process B wants a resource from Process A. Similarly Process C wants resource from Process B and Process A from Process C.

II. NECESSARY CONDITIONS FOR DEADLOCK

There are four conditions are known as the **Coffman conditions.** They were first described in a 1971 by Edward G. Coffman, Jr. All the following conditions are necessary for a deadlock to occur

1. **Mutual Exclusion:** No less than one asset must be held in a non-shareable mode. Stand out methodology can utilize the asset at a specific time.
2. **Hold and Wait** or **Resource Holding:** A methodology is presently holding no less than one asset and solicitations for an alternate assets which are held by different procedures.

3. **No Preemption:** a resource can be released only voluntarily by the process which is holding it, whenever the process completes its task.
4. **Circular Wait:** A process must be sitting tight for an asset which is constantly held by an alternate process, and the following process is holding up for the first process to leave or say discharge the asset. Case in point, there is a situated of holding up processes, P = {p1, P2, ..., Pn}, such that process P1 is sitting tight for an asset held by process P2, process P2 is sitting tight for an asset held by process P3 along these lines on until last process PN is holding up for an asset held by P1.

III. DEADLOCK DETECTION

It is fundamentally the process of really discovering that a deadlock exists and recognizing the processes and assets included in the deadlock.

The fundamental thought is to check allotment against asset accessibility for all conceivable portion groupings to figure out whether the framework is in deadlocked state.

For Example:

Chandy-Misra-Hass Detection Algorithm

This is viewed as an edge-pursuing, test based algorithm. This algorithm is considered as one of the best deadlock location algorithms.

On the off chance that a process makes a solicitation for an asset which falls flat or times out, the process creates a message and sends it to each of the processes holding one or a greater amount of its asked for assets.

Each message contains the following information:

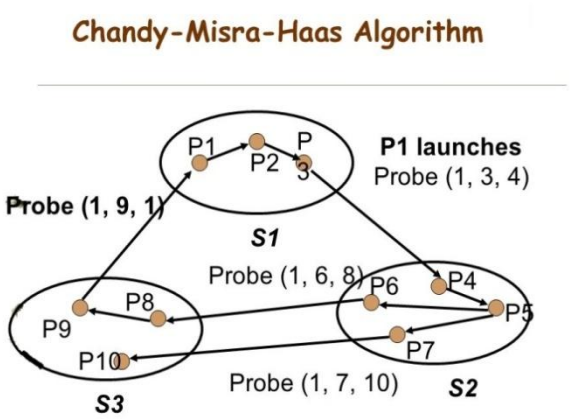
1. the id of the process that is blocked (the one that initiates the probe message);
2. the id of the process is sending this particular version of the probe message; and
3. the id of the process that should receive this probe message.

At the point when a process gets a test message, it verifies whether it is additionally holding up for resources. If not, it is presently utilizing the required resource and will in the long run complete and discharge the resource.

On the off chance that it is holding up for resources, it passes on the test message to all processes it knows to be holding resources it has itself asked.

The process first adjusts the test message, changing the sender and recipient ids.

On the off chance that a process gets a test message that it perceives as having started, it knows there is a cycle in the framework and consequently, gridlock.



[1] Chandy-Misra-Haas Algorithm

IV. ALGORITHM

Controller sending a message

if Pb is locally dependent on itself
    then declare deadlock
else for all Pb, Pc such that
    (i) Pa is locally dependent on Pb,
    (ii) Pb is waiting for 'Pc and
    (iii) Pb, Pb are on different controllers.
send probe(a, b, c).

Controller receiving a probe

if
    (i)Pc is idle,
    (ii) dependent c(a) = false, and
    (iii)requests responded by Pc to Pb
then begin
    "dependents""c"(a) = true;
    if c == a
    then declare that Pa is deadlocked
    else for all Pr,Ps such that
        (i) Pc is locally dependent on Pr,
        (ii) Pr is waiting for 'Ps and
        (iii) Pr, Ps are on different controllers.
    send probe(a, r, s).
end

V. DEADLOCK PREVENTION

It is the process of making it logically impossible for one of the 4 conditions to hold. Deadlock can be prevented by the methods given below:

Elimination of "Mutual Exclusion" Condition: The shared rejection condition must hold for non-sharable assets. That is, a few processes can't at the same time impart a solitary asset. This condition is troublesome in light of the fact that a few assets, for example, the tap drive and printer, are non-shareable.

Elimination of "Hold and Wait" Condition: There are two methods for this condition. Initially is that a process appeal be conceded the greater part of the assets it needs without a moment's delay, before execution. Second is not to permit a process from asking for assets at whatever point it has formerly apportioned assets.

Elimination of "No-preemption" Condition: The non-preemption condition might be minimized by compelling a methodology sitting tight for an asset that can't instantly be designated to give every last bit of its as of now held assets, so different courses of action may utilize them to complete.

Elimination of "Circular Wait" Condition: The last condition, the circular wait, could be denied by forcing an aggregate requesting on the majority of the asset sorts and afterward compelling, all procedures to ask for the assets in place.

VI. CONCLUSION

Deadlock control policies should no longer be classified as prevention and avoidance. All of the policies typically listed as deadlock control directly negate a precondition of resource deadlock, including those classified as "avoidance" schemes. Whenever possible in computer science, and elsewhere, simplicity is a desirable goal.

REFERENCES

[1] http://images.slideplayer.us/1/252903/slides/slide_69.jpg
[2] http://en.wikipedia.org/wiki/Deadlock
[3] http://www.isi.edu/~faber/cs402/notes/lecture9.html
[4] http://www.personal.kent.edu/~rmuhamma/OpSystems/Myos/deadlockPrevent.htm