An Overview on Deadlock Resolution Techniques

Sumika Jain[1]
[1]Assistant Professor, Department of Computer Science, Shobhit University Gangoh (U.P)

Nitin Kumar[2]
[2] Assistant Professor, Department of Computer Science, Shobhit University Gangoh (U.P)

Kuldeep Chauhan[3]
[3] Assistant Professor, Department of Computer Science, Shobhit University Gangoh (U.P)

Abstract-- **A deadlock occurs when there is a set of processes waiting for resource held by other processes in the same set. The processes in deadlock wait indefinitely for the resources and never terminate their executions and the resources they hold are not available to any other process. The occurrence of deadlocks should be controlled effectively by their detection and resolution, but may sometimes lead to a serious system failure. After implying a detection algorithm the deadlock is resolved by a deadlock resolution algorithm whose primary step is to either select the victim then to abort the victim. This step resolves deadlock easily. This paper describes deadlock detection using wait for graph and some deadlock resolution algorithms which resolves the deadlock by selecting victims using different criteria**

Keywords: Deadlock, Resources, Processes, Release.

1. INTRODUCTION

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Consider an example when two trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other. Similar situation occurs in operating systems when there are two or more processes hold some resources and wait for resources held by other(s). There is a variant of deadlock called lovelock. This is a situation in which two or more processes continuously change their state in response to changes in the other processes without doing any useful work. This is similar to deadlock in that no progress is made but differs in that neither process is blocked or waiting for anything. A human example of livelock would be two people who meet face-to-face in a corridor and each moves aside to let the other pass, but they end up swaying from side to side without making any progress because they always move the same way at the same time. Deadlocks can be avoided by avoiding at least one of the four conditions, because all this four conditions are required simultaneously to cause deadlock.

Mutual Exclusion Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process.

Hold and Wait In this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

No Preemption Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.

Circular Wait Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing (or decreasing) order.

The above points focus on preventing deadlocks. But what to do once a deadlock has occurred. Following three strategies can be used to remove deadlock after its occurrence.

Preemption We can take a resource from one process and give it to other. This will resolve the deadlock situation, but sometimes it does causes problems.

Rollback In situations where deadlock is a real possibility, the system can periodically make a record of the state of each process and when deadlock occurs, roll everything back to the last checkpoint, and restart, but allocating resources differently so that deadlock does not occur.

Kill one or more processes this is the simplest but it works.

A process in operating systems uses different resources and uses resources in following way.

(1) Request a Resource
(2) Use a resource
(3) Release a resource

Generally speaking there are three ways of handling deadlocks:

(1) Deadlock prevention or avoidance - Do not allow the system to get into a deadlocked state.
(2) Deadlock detection and recovery - Abort a process or preempt some resources when deadlocks are detected.
(3) Ignore the problem all together - If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take.

In order to avoid deadlocks, the system must have additional information about all processes. In particular, the system must know what resources a process will or may request in the future. (Ranging from a simple worst-case maximum to a complete resource request and release plan for each process, depending on the particular algorithm.)

Deadlock detection is fairly straightforward, but deadlock recovery requires either aborting processes or preempting resources, neither of which is an attractive alternative. If deadlocks are neither prevented nor detected, then when a deadlock occurs the system will gradually slow down, as more and more processes become stuck waiting for resources currently held by the deadlock and by other

waiting processes. Unfortunately this slowdown can be indistinguishable from a general system slowdown when a real-time process has heavy computing needs.
Mono-processing systems do not have to worry about deadlock. The reason is that deadlock involves resource allocation, and if there is only one process, it has uncontested access to all resources. Only certain types of resources are associated with deadlock, and they are of the exclusive-use, non-preemptible type. That is to say, only one process can use the resource at any given time, and once allocated the resource cannot be unallocated by the operating system, but rather the process has control over the resource until it completes its task. Excellent examples of such resources are printers, plotters, tape drives, etc.. Resources that do not fit the criteria are memory and the CPU. While it is convenient to discuss deadlock in terms of hardware resources, there are software resources that are equally good candidates for deadlock, such as records in a data base system, slots in a process table, or spooling space. Hardware or software, all that matters is that the resources are non-preemptible and serially reusable. The four conditions that must exist for deadlock are as follows: The first two are described above - mutually exclusive use of resources by the processes and non-preemption (resources cannot be removed from the processes). The third condition is called the 'hold and wait' or 'wait for' condition.

2. DEADLOCK

In concurrent computing, a deadlock is a state in which each member of a group is waiting for another member, including itself, to take action, such as sending a message or more commonly releasing a lock.[1]

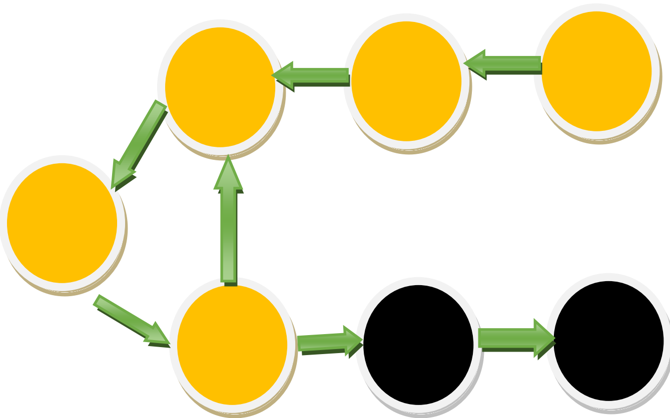

Fig. A Few Process in Deadlock

Deadlock is a common problem in multiprocessing systems, parallel computing, and distributed systems, where software and hardware locks are used to arbitrate shared resources and implement process synchronization.[2]

In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock.[3]

In a communications system, deadlocks occur mainly due to lost or corrupt signals rather than resource contention

3. BACKGROUND

Deadlock can occur when the permanent blocking of a set of processes compete for the same system resources. A set of processes is deadlocked when each process in the set is blocked awaiting an event that can only be triggered by another blocked process in the set. Deadlock is permanent because none of the events are ever triggered. Three conditions must take place for a deadlock to take place. The first one is Mutual exclusion, which is a single process that uses one resource at a time. No process may access a resource unit that is being utilized by another process. Hold and wait is the second condition, it can be described as one process that holds assigned resources while waiting for another assignment. Finally, No preemption occurs when no resource can be forced or removed from a process holding it. These three conditions are necessary for a deadlock to exist. However, a fourth condition is required for an actual deadlock to take place. Circular wait occurs when a closed chain of processes exists, and each process holds one or more resources needed by the next process in the chain. This condition is an immediate result of the first three. Below is a self-explanatory illustration of Deadlock. Deadlock blocks a set of processes that competes for system resources. This can be permanent unless the OS takes action, such as forcing one or more processes to backtrack. Deadlock may involve consumable or reusable resources. A reusable resource is one that is not depleted by use. A consumable resource is one that is destroyed when it is obtained by a process. There are three approaches to dealing with deadlock: prevention, detection, and avoidance. Prevention guarantees that deadlocks will not happen. Detection is required if the OS is willing to grant resource requests; the OS checks for deadlocks and takes action to break the deadlock. Avoidance involves the analysis of each new resource request to determine if it could lead to deadlock, and granting it only if deadlock is not an option. For Windows you can run driver verifier to scan for any corrupted drivers, which may be causing problems, this program works by running various stress tests on drivers, in order to produce a BSOD, which will locate the driver. To the best of my knowledge deadlocks are ignored by Linux operating systems.
Under the deadlock detection, deadlocks are allowed to occur. Then the state of the system is examined to detect that a deadlock has occurred and subsequently it is corrected. An algorithm is employed that tracks resource allocation and process states, it rolls back and restarts one or more of the processes in order to remove the detected deadlock. Detecting a deadlock that has already occurred is easily possible since the resources that each process has locked and/or currently requested are known to the resource scheduler of the operating system.

4. DEADLOCK PREVENTION

Deadlock prevention works by preventing one of the four conditions from occurring.

1. By Removing the mutual exclusion condition means that no process will have exclusive access to a resource. This proves impossible for resources that cannot be spooled. But even with spooled resources, deadlock could still occur. Algorithms that avoid mutual exclusion are called non-blocking synchronization algorithms.

2. The hold and wait or resource holding conditions may be removed by requiring processes to request all the resources they will need before starting up (or before embarking upon a particular set of operations). This advance knowledge is frequently difficult to satisfy and, in any case, is an inefficient use of resources. Another way is to require processes to request resources only when it has none. Thus, first they must release all their currently held resources before requesting all the resources they will need from scratch. This too is often impractical. It is so because resources may be allocated and remain unused for long periods. Also, a process requiring a popular resource may have to wait indefinitely, as such a resource may always be allocated to some process, resulting in resource starvation.(These algorithms, such as serializing tokens, are known as the all-or-none algorithms.)

3. The no preemption condition may also be difficult or impossible to avoid as a process has to be able to have a resource for a certain amount of time, or the processing outcome may be inconsistent or thrashing may occur. However, inability to enforce preemption may interfere with a priority algorithm. Preemption of a "locked out" resource generally implies a rollback, and is to be avoided, since it is very costly in overhead. Algorithms that allow preemption include lock-free and wait-free algorithms and optimistic concurrency control. If a process holding some resources and requests for some another resource(s) that cannot be immediately allocated to it, the condition may be removed by releasing all the currently being held resources of that process.

4. The final condition is the circular wait condition. Approaches that avoid circular waits include disabling interrupts during critical sections and using a hierarchy to determine a partial ordering of resources. If no obvious hierarchy exists, even the memory address of resources has been used to determine ordering and resources are requested in the increasing order of the enumeration.[3] Dijkstra's solution can also be used.

5. FUTURE WORK

In the future any other new prevention techniques can be originated .so that it can give more efficient result.

6. REFERENCES

[1] Terekhov, T. Camp, “Time efficient deadlock resolution algorithms”, June 1998

[2] D. Manivannan and Mukesh Singhal, “An Efficient Distributed Algorithm for Detection of Knots and Cycles in a Distributed Graph”, IEEE Transactions on Parallel And Distributed Systems, Vol. 14, No. 10, October 2003.

[3] Lee, S., “Fast, Centralized Detection and Resolution of Distributed Deadlocks in the Generalized Model”, IEEE Trans. On Software Engineering, Vol. 30, NO. 9, 561-573, 2004

[4] Yibei Ling, Shigang Chen and Cho-Yu Jason Chiang, “On Optimal Deadlock Detection Scheduling”, IEEE TransactionsOn Computers, Vol. 55, No. 9, September 2006.

[5] Pallavi Joshi, Mayur Naik, “A randomized Dynamic Program Analysis Technique for Detecting Read Deadlocks”, 2009 ACM, June 2009.

[6] Prodromos Gerakios, Nikolaso Papaspyrou, Konstantinos, Vekris,”Dynamic Deadlock Avoidance in System Code Using Statically Inferred Effects”, 2011 ACM, October 2011

[7] Iryna Felko, TU Dortmund, “Simulation based Deadlock Avoidance and Optimization in Bidirectional AGVS”, 2011 ACM, march 2011.

[8] Selvaraj Srinivasan, R. Rajaram, “A decenreralized deadlock detection and resolution algorithm for generalized model in distributed systems”, January 2011.

[9] Kunwar Singh Vaisla, Menka Goswami, Ajit Singh, “VGS Algorithm -an Efficient Deadlock Resolution Method”, International Journal ofComputerApplications (0975 – 8887),Vol.44-No. 1April 2012.

[10] Jeremy D.Buhler, Kunal Agrawal, Peng Li, Roger D. Chamberlain, “Efficient Deadlock Avoidance For Streaming Computation With Filtering”, 2012 ACM, February 2012.