International Journal of Academic Information Systems Research (IJAISR)
ISSN: 2643-9026
Vol. 4, Issue 5, May – 2020, Pages: 11-15

Deadlock Avoidance Strategy in Concurrent Processing

Khin Po[1], May Thu Naing[2]

[1]Faculty of Computer Science, University of Computer Studies, Mandalay, Myanmar
trkhinpo@cumandalay.edu.mm
[2]Faulty of IT Support and Maintenance, University of Computer Studies, Mandalay, Myanmar
maythunaing@cumandalay.edu.mm

Abstract*: Deadlock can be defined as the permanent blocking of a set of processes. This processes are either competes for the system resources or communicate with each other. All deadlocks involve conflicting needs for resources by two or more processes. The most common example of deadlock is the traffic deadlock. The blockage is permanent unless the OS takes some extraordinary action, such as killing me or more processes or forcing one or more processes to backtrack. There are three general approaches to dealing with the deadlock: prevention, detection and avoidance. This paper presents the banker's algorithm that used in deadlock avoidance strategy.*

Keywords—Deadlock, Deadlock prevention, Deadlock detection, Deadlock avoidance, Banker's algorithm

1. INTRODUCTION

Deadlock may involve reusable resources or consumable resources. A reusable resource is not depleted or destroyed by use, such as I/O channel or a region or memory. As an example of deadlock involving reusable resources, consider two processes that compete for exclusive access to a disk file D and a tape drive T. Deadlock occurs if each process holds one resource and requests the other. When a consumable resource is acquired by a process, it is destroyed. The examples include messages and information in IO buffers. As an example of deadlock involving consumable resources, they consider the pair of processes. Each process attempts to receive a message from the other process. And then, it send a message to the other process. Deadlock occurs if the receiving process is blocked [8].

There is no single effective strategy that can have deal with all types of deadlock. Three most important approaches that have been developed: prevention, avoidance and detection. A useful tool in characterizing the allocation of resources to processes is the resource allocation graph. Three conditions of policy must be present for a deadlock to be possible: Mutual exclusion, Hold and wait and No preemption and for deadlock to actually take place, a fourth condition is required, Circular wait.

Deadlock prevention guarantees that deadlock will not occur, by assuring that one of the necessary conditions for deadlock is not met. Deadlock detection is needed if the OS is always willing to grant resource requests; periodically, the OS must cheek for deadlock and take action to break the deadlock. 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 possible.[7]

2. RELATED WORKS

This paper focus on the problem of deadlock avoidance. There are quite a lot research papers devote to deadlock avoidance. A deadlock avoidance algorithm is proposed for a class of Petri net models formed for flow shop manufacturing where a set of sequential processes are executed without alternating the order of using resources in each case [2]. The algorithm controls the input flowing of new tokens in a local area, ensuring that token evolutions in system are always possible. Abdallah in [4] use structure theory of Petri nets to develop efficient deadlock prevention and deadlock avoidance methods for FMS. Naiqi Wu et al point out that, if an Automated Manufacturing System(AMS) operates at the deadlock boundary, i.e., under the maximally permissive control policy, it will not be deadlocked but a blocking(the process is stopped temporarily, and will go on after a period of time) may occur more likely. Wu presents an AMS that works near but not at the deadlock boundary in order to gain the highest productivity [5]. For the first time Wu presents such a policy: Liveness-policy. Without being too conservative, it can effectively reduce or even eliminate the blocking possibility that exists under a maximally permissive control policy.

Ajoy K.Datta Sukumar GhoshTair-Shian Chou [1] present an algorithm for deadlock avoidance in a resource sharing environment with multiple types of resources, where the maximum claims of the individual processes are unknown *a priori*. A new copy of *process ordering* is introduced in place of *resource ordering*.

3. DEADLOCK

Four conditions of policy must be present for a deadlock:
1. Mutual exclusion: Only one process may use a resources at a time. No process may access a resource unit that has been allocated to another process.
2. Hold and wait: A process may hold allocated resources while awaiting assignment of other resources.
3. No preemption : No resource can be forcibly removed from a process holding it.

4. Circulate wait: A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain.[7]

Three general approaches exist for dealing with deadlock. First, the deadlock is made unreachable. It is done by locking the resource. The allocated resource will be blocked and none other can use that resource. The prevention can be done if the resources availability prior is known. Second, different protocols or algorithms are used to avoid deadlock. All protocols are on assumption that circular dependency is absent. And also, one can avoid deadlock by making the appropriate dynamic choices based on the current state of resource allocation. [6] Third, one can attempt to detect the presence of deadlock (conditions, through 4 hold) and take action to recover.

For example in gaming system if deadlock may come once in year it may hang it will not be a serious problem by rebooting system it can be overcome.

Different approaches to avoid deadlock:

1. Statically break circular wait.

This will reduce the resource use and concurrency will be reduced. The programmer has to check all the situation of circular wait and he has to eliminate it.

2. Release some resources and go back to previous state. This can be done in databases. It is not applicable to real time embedded systems. If done then timely response can't be provided.

3. Dynamically allocating the resources which are free will not give efficient resource allocation [3].

Different deadlock avoidance models are:

1. Dijkstra's Banker's algorithm
2. Adequate Protocol
3. Basic Protocol
4. Efficient Protocol
5. K-Efficient Protocol
6. Live Protocol

In deadlock avoidance, employs to access the possibility Banker's algorithm that deadlock could occur and acting accordingly. Thus this approach differs from deadlock prevention. With deadlock detection, requested resources are granted to processes whenever possible. Periodically, the OS performs an algorithm that allows it to detect the circular wait condition described earlier in condition (4).

1. Banker's algorithm

The strategy of resource allocation denial, referred to the banker's algorithm. The name was chosen because the algorithm would be used in a banking system to ensure that the bank never allocates its available case such that it can no longer satisfy the needs. A deadlock avoidance policy refuse to start a new process if its resource requirements might lead to deadlock. A process is only started if the maximum claims of all current processes plus those of a new process can be met.

Thus, after processing, one can define the system whether the system is safe or safe state. If the state of the system avoids the deadlock, this state is called a safe state and deadlock state is an unsafe state. The state consists of the two vectors, Resource and Available, and the two matrices, claim and Allocation. A safe state is one in which there is at least one sequence of resource allocations to processes that does not result in a deadlock (i.e all of the process can be run to completion). An unsafe state is, of course a state that is not safe [7].

The following example illustrates these concepts.

(a) Determination of a safe state.

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim Matrix C

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation Matrix A

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C-A=Needs

R1	R2	R3
9	3	6

Resources vector R

R1	R2	R3
0	1	1

Available vector V

Figure 1. (a) Initial State

	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim Matrix C

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation Matrix A

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C-A=Needs

R1	R2	R3
9	3	6

Resources vector R

R1	R2	R3
6	2	3

Available vector V

Figure 1. (b) P2 runs to completion

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim Matrix C

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation Matrix A

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C-A=Needs

R1	R2	R3
9	3	6

Resources vector R

R1	R2	R3
7	2	3

Available vector V

Figure 1. (c) P1 runs to completion

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Claim Matrix C

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Allocation Matrix A

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	0

C-A=Needs

R1	R2	R3
9	3	6

Resources vector R

R1	R2	R3
9	3	4

Available vector V

Figure 1. (d) P3 runs to completion

Is this a safe state? To answer this question, we ask an intermediate question: Can any of the four processes be run to completion with the resource available? Clearly, this is not possible for P1, which has only 1 unit of R1 and requires more units of R1, 2 units of R2 and 2 units of R3. However, by assigning one unit of R3 to process P2, P2 has its maximum required resources allocated and can run to completion. Let us assume that this is accomplished. Thus P2 runs to completion and its resources can be returned to the available vector V. This state is shown in Figure 1(b). According to this stage, each of the remaining processes could be completed. For example, process P1 which needs 2 units of R1, 2 units of R2 and 2 units of R3. There are in available vector 6 units of R1, 2 units of R2 and 3 units of R3. Therefore, P1 can run be completion. This state is shown in Figure 1 (c). Next we can complete P3 (Figure 1(d)). Finally, we can complete all processes. The state is safe state.

(b) Determination of an unsafe state.

Now consider the state defined in figure 1. Suppose P1 makes a request for one additional unit of R1 and one additional unit of R3. If we assume that the request is granted, we are left in the figure. Is this a safe state? The answer is no, because each process unit need at least are additional unit of R1, and there are none available. Thus, on the basis of deadlock avoidance, the request by P1 should be denied and P1 should be blocked.

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim Matrix C

	R1	R2	R3
P1	1	0	0
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation Matrix A

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C-A=Needs

R1	R2	R3
9	3	6

Resources vector R

R1	R2	R3
1	1	2

Available vector V

Figure 2. (a) Initial State

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim Matrix C

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

Allocation Matrix A

	R1	R2	R3
P1	1	2	1
P2	1	0	2
P3	1	0	3
P4	4	2	0

C-A=Needs

R1	R2	R3
9	3	6

Resources vector R

R1	R2	R3
0	1	1

Available vector V

Figure 2. (b) P1 request one unit

4. Conclusion

Deadlock is permanent because none of the events is ever triggered. Unlike other problems in concurrent process management , there is no efficient solution in the general case. There is no single effective strategy that can deal with all types of deadlock. If the detection and prevention will take more time then it will not be efficient. This paper presented Banker's algorithm that one of the solution for deadlock avoidances with examples.

6. References

[1] Ajoy K.Datta Sukumar Ghosh Tair-Shian Chou, A New algorithm for deadlock avoidance, Information Sciences, volume 46, issues 1–2, October–November 1988, pages 47-72
[2] Banaszak, Z.A.; Krogh, B.H. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows[J]. IEEE Transactions on Robotics and Automation, 1990, 6(6), 724 –734.
[3] Cesar Sanchez, Henny B. Sipma, and Zohar Manna, "A Family of Distributed Deadlock Avoidance Protocols and their Reachable State Spaces", Computer Science Department.
[4] I.BenAbdallah ; H.ElMaraghy. Deadlock Prevention and Avoidance in FMS:A Petri Net-Based Approach[J]. International Journal of Advanced Manufacturing Technology, 1998, 16(1).
[5] Naiqi Wu, MengChu Zhou. Avoiding deadlock and reducing starvation and blocking in automated manufacturing systems[J]. IEEE Transactions on Robotics and Automation, 2001, 17(5), 658 –669.
[6] Prashant.H, Raju.Hand Santosh.K, "A Study on Different Deadlock Avoidance Strategies in Distributed Real Time Embedded Systems", International Journal on Emerging Technologies , ICRIET-2016,pp. 244-247
[7] William Stallings, Operating Systems internals and design principles, seventh edition.
[8]https://en.m.wikipedia.org/wiki/Deadlock