International Journal on Emerging Technologies (Special Issue on ICRIET-2016) **7**(2): 244-247(2016)

ISSN No. (Print) : 0975-8364
ISSN No. (Online) : 2249-3255

A Study on Different Deadlock Avoidance Strategies in Distributed Real Time Embedded Systems

Prashant Hebbale*, Raju Hebbale** and Santosh Kolaki*

**Asst. Professor, Department of Electronics & Communication Engineering, VSMIT, Nipani*
***Asst. Professor, Department of Electronics & Communication Engineering, KLECET, Chickodi*

(Corresponding author: Prashant Hebbale)
(Received 28 September, 2016 Accepted 29 October, 2016)
(Published by Research Trend, Website: www.researchtrend.net)

ABSTRACT: In this paper we discuss about the resource management in distributed real time embedded systems. The deadlocks, missed deadlines, priority inversion problems are due to the incorrect handling of resources. Deadlock avoidance will provide the correct handling of resources. All the protocols will first check the availability of resource. If resources are available then only RTOS will allocate resource else it will not. In avoidance technique at any time the resource will be in safe state only. The general solutions may not give the efficient deadlock avoidance strategies. In this paper different protocols are presented that will give efficient deadlock avoidance methods. The condition to use these different protocols will be the system should not have circular dependencies.

In deadlock prevention the deadlock is made unreachable. The resource is blocked and none of process can access it but it will reduce performance and it will restrict execution. But in avoidance technique deadlock is avoided by different protocols. Avoidance of deadlock will provide the efficient resource handling.

Detection of deadlock and preventing it will be offline process. During offline deadlocks are detected and allocation of resources will be calculated and according to requirements available resources will be given. In RTOS offline process will not be effective because resource requirements cannot be known prior. Deadlock avoidance technique will be effective technique for RTOS. Different deadlock avoidance protocols will provide different level of concurrency and less execution time. All protocols in deadlock avoidance will ensure deadlock free system.

Key words: Basic P, Live P, Deadlock.

I. INTRODUCTION

A real-time system is a system that must satisfy explicit (bounded) response-time constraints or risk severe consequences including failure.

In real-time systems logical correctness is based on both the correctness of output and their timeliness [1][5] .

Deadlock:

Necessary conditions for deadlock:

1. Mutual exclusion
2. Circular wait
3. Hold and wait
4. No preemption

The resources which can't be shared with others will create mutual exclusion. The process which has cyclic processing will create circular wait. The process which request resources and lock those resources until all its required resources are filled will create hold on wait. Partially allocated semaphores may create no preemption problem.

Different methods to deal with deadlock:

1. Prevention
2. Avoidance
3. Detection

In first method 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 we know the resources availability prior. But in real time embedded system this will not be possible and it will restrict the execution.

In second method different protocols or algorithms are used to avoid deadlock. All protocols are on assumption that circular dependency is absent.

In third method deadlock condition will be calculated in offline and it will be corrected.

If the detection and prevention will take more time then it will not be efficient. 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 [2].

Different deadlock avoidance models are:

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

1**. Dijkstra's Banker's algorithm**:

The Bankers algorithm is based on the maximum resources available by each task and resource availability at current instant. It is based on static and dynamic knowledge of resource availability. Banker's algorithm will satisfy the need of resources available to processes satisfactorily. This algorithm will ensure that the number of resources required will never exceed the number of resources available for the system.

Algorithm example:

Consider system having 4 processes P, Q, R and S and total of 15 resources. The max requirement of each resource is 7, 7, 4 and 5 respectively. The resource requirements are as shown in table 1. Here requirements of all processes can be met. The system will be in safe state.

Table 1: Resource requirements of different process.

Process	Max req	Requirement at this instant	Used	Possibly need
P	7	0	0	7
Q	7	0	0	7
R	4	0	0	4
S	5	0	0	5
Total availability				15

At each time resources required the OS will update this table to ensure deadlock free system.

Let resource required by processes at first instant be 2,3,1,1 respectively. All these resources can be allocated so system is in safe state. The allocation of resources is as shown in table 2. Then the table will be updated as

Table 2: Usage of resources.

Process	Max req	Requirement at this instant	Used	Possibly need
P	7	2	2	5
Q	7	3	3	4
R	4	1	1	3
S	5	1	1	4
Total availability				8

Let resource required by processes at this instant be 2,2,1,1 respectively, this is as shown in table 3. All these resources can be allocated so system is in safe state.

Then the table will be updated as

Table 3: After allocation of resources.

Process	Max Req	Requirement at this instant	Used	Possibly need
P	7	2	4	3
Q	7	2	5	2
R	4	1	2	2
S	5	1	2	3
Total availability				2

In the above table only the requirements of P,Q or R can be met. The requirement of S can't be met. For the Process S it will be unsafe state for other processes it is safe state [5][6].

Let resource required by processes at this instant be 1,0,0,0 respectively, this is shown in table 4.

Table 4: Resource allocation.

Process	Max req	Requirement at this instant	Used	Possibly need
P	7	1	5	2
Q	7	0	5	2
R	4	0	2	2
S	5	0	2	3
Total availability				1

At this stage none of the process can get possible needed resources so the system is unsafe state. During unsafe state the system will not allocate resources to any of the processes. If any system releases the resources at that time the table will be updated and again check for safe state and allocate resources.

The Bankers algorithm will ensure the safe state of system it is slow for real time embedded system. In this the resources needed by process must be known priori. But in real time system the availability of resources needed may not be known priori.

Protocols: There are different protocols available to avoid deadlock. These protocols will check the availability of resources and ensure dead lock free system. The simple protocol schema is given bellow.

When En_n (Vr) In_n(Vr,Vr')	Entry section
f( ) ;	Function
Out_n(Vr,Vr')	Resource release

Fig. 1. Protocol Schema.

Let
In- is entry section of process,
Out- is exit section of process,
En- is enabling condition of process,
Vr- is resources available,
Vr'- resources available after execution of process,
n - is method of function execution.

En_n(Vr) be the enabling condition for process to execute. the process is executing in method n and checking for resource availability Vr if the resource is available then the that process will be allocated resources and further execution of process will take place. Soon after the enabling condition next step will be to update the resources available after allocating resource to method n. This will be indicated by Vr'. After this the process execution will take place. After executing, process must give resources back which it has allocated. The resource Vr' will become Vr. If at the entry section only the resource has not available then further process will not be done.

Adequate P: Adequate protocol is one in which the number of resources allocated in every task should never exceed the total number of available resources[2]. Adequacy must be there for each and every protocol. The non available resources can't be allocated if allocated then that will lead to deadlock. At each time if resources available are greater than zero then the resource will be allocated to process and remaining resource availability will be updated.

Computational Model:
Example of deadlock

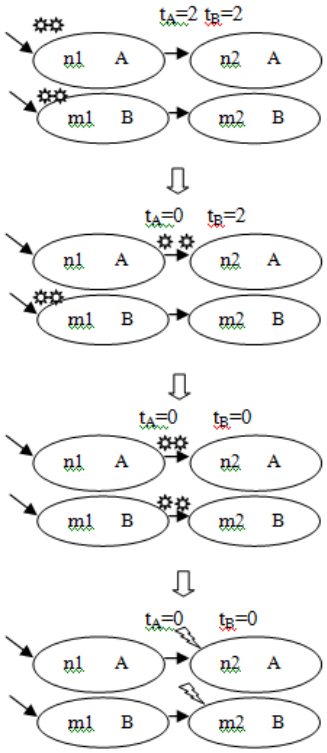

Fig. 2. Computational model.

=> This symbol indicates deadlock
=> This symbol indicates resource allocation

In above example each task has 2 resources. At last step both the task will not be satisfied so they are deadlocked. In this situation both the tasks t_A and t_B will not get resources and they will be deadlocked.

Basic P: The correctness of Basic P protocol is based on the acyclicity of the process. When resources are request then the protocol will check availability of resources and then allocate it. After allocation all the resources are not given to that process immediately. Once the process is complete the resource will be given back that is it will be released and Tr will be increased. The Basic P algorithm is as shown in fig 3.

When i ≤ tr Tr --	Entry section
f( ) ;	Function
Tr ++	Resource release

Fig. 3. Protocol Basic P.

At entry section Tr will be updated and after execution of process Tr is updated again so that resource available can be used by other process's.

Efficient P: The Basic P can be improved by increasing the annotation levels which will lead to Efficient P protocol. Efficient P uses another local variable to track thread availability [4].

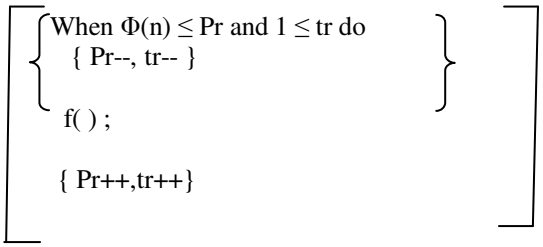

Fig. 4. Protocol Efficient P.

Both the potentially available threads and actually available threads are used in this protocol. During resource allocation both the conditions are checked, the efficient P is as shown in fig 4. If both conditions are satisfied then the resource will be provided and further function will proceed. Pr and Tr are the two variables. Concurrency will be improved as compared to Basic-P protocol.

K Efficient P:

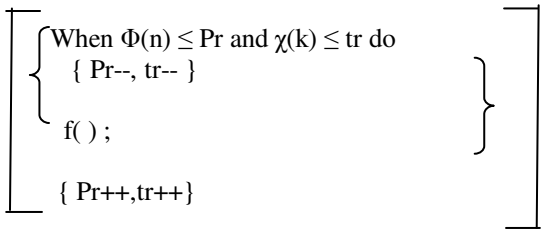

Fig. 5. Protocol K Efficient P.

The k is strengthening point. Figure 5 shows the scheme of K efficient P protocol. Lower the value of k will indicate the less computation needed to check enabling condition. If k = 0 then the protocol will same as Basic P protocol. K will provide the strength to the resource allocation. In Basic P protocol single counter is used where as in K Efficient P many counters are used. The increment or decrement of resource is done independent of tr.

Live P: Resource allocation is controlled in Live P protocol. The correct implementation of Live P will ensure the deadlock situation and also liveness. Implementation of Live P will prevent starvation. Every waiting process is enabled if it satisfies the entry condition of Live P. Figure 6 shows the protocol schema Live-P that controls resource allocation for a method n.

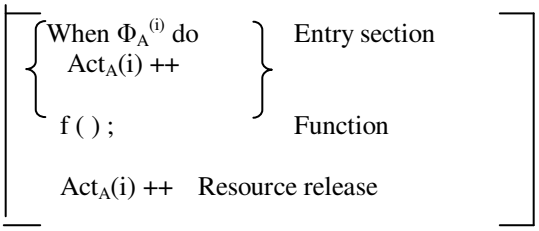

Fig. 6. Protocol Live P.

IV. CONCLUSION

All the Protocols explained above are based on the principle that processes are not cyclic. All the protocols will ensure the process doesn't have deadlock situation. These will prevent deadlock efficiently.

In this paper we present an algorithm to compute optimal annotations that is annotations that maximize parallelism while satisfying the condition of acyclicity. Moreover, we show that the condition of acyclicity is in fact tight and exhibits a rather surprising anomaly: if a cyclic dependency is present in the annotation of the call graph and a certain minimum number of threads is provided, deadlock is reachable. Thus, in the presence of cyclic dependencies, increasing the number of threads may introduce the possibility of deadlock in an originally deadlock free system.

REFERENCES

[1]. Cesar Sanchez, "Deadlock Avoidance for *Distributed Real-Time and Embedded Systems",* Computer Science And The Committee, Stanford University, May 2007.

[1]. Cesar Sanchez, Henny B. Sipma, and Zohar Manna, "*A Family of Distributed Deadlock Avoidance Protocols and their Reachable State Spaces"*, Computer Science Department.

[1]. Kai Han, "*Scheduling Distributed Real-Time Tasks in Unreliable and Untrustworthy Systems",* Virginia Polytechnic Institute and State University, September 18, 2007.

[1]. Cesar Sanchez, Henny B. Sipma, Christopher D. Gill, and Zohar Manna1, "*Distributed Priority Inheritance for Real-Time and Embedded Systems",* Stanford University.

[1]. Raj Kamal, *"Embedded systems Architecture, Programming and Design*", TMH.

[1]. Philip. A. Laplante, *"Real-Time Systems Design and Analysis- an Engineer's Handbook"*-Second Eition, PHI Publications.