Deadlock Prevention in Process Control Computer System

Manisha Mohanty
Gandhi Institute For Education & Technology
Baniatangi, BBSR, 752060

Prerna Kumara
Gandhi Institute For Education & Technology
Baniatangi, BBSR, 752060

ABSTRACT
Deadlock can occur wherever multiple processes interact. System deadlock is a serious problem in a multiprogramming environment. The approaches to this problem can be divided into three categories: (1) prevention, (2) detection and recovery, and (3) avoidance. This paper proposes a variation of the first approach, partially applying ideas developed in the second and third approaches. This is an approach that is especially effective in process control computer systems, in which the application programs are usually fixed once designed. Using four predetermined application program parameters obtained in the program development stage, a directed graph model and a 'restriction' matrix model are introduced representing the usage of common resources. Conditions sufficient for system deadlock prevention are presented along with algorithms for checking to see that the models meet these conditions. By using this approach, if a deadlock possibility is detected the causes can also be detected. The deadlock can thus be prevented during the program development stage. As the algorithms are not used in the real-time mode, there is no negative effect on the responsiveness of the system. A higher utilization rate of common resources is also ensured because the usage of resources is restricted only when the possibility of a deadlock is detected.

Keywords
Deadlock, prevention, recovery, detection, avoidance.

1. INTRODUCTION
In the non-distributed case, all the information on resource usage lives on one system and the graph may be constructed on that system. In the distributed case, the individual subgraphs have to be propagated to a central coordinator. A message can be sent each time an arc is added or deleted. If optimization is needed, a list of added or deleted arcs can be sent periodically to reduce the overall number of messages sent.

What is Deadlock: A deadlock is a state where a set of processes request resources that are held by other processes in the set. A deadlock is a condition in a system where a process cannot proceed because it needs to obtain a resource held by another process but it itself is holding a resource that the other process needs.

A computer system which allows more than one process to be simultaneously active, holding and requesting resources, may encounter the phenomenon of deadlock (sometimes called deadly embrace).“Deadlock occurs so infrequently that it is not worthwhile to degrade system performance by executing prevention algorithms.”When deadlock situation arise in an online computer system, the system cannot respond within an acceptable period of time. This is particularly true in process control applications, where a very quick response is required of computer systems. According to Coffman approaches to this problem can be classified in three categories.(1)prevention (2)detection and recovery and (3)avoidance.

Prevention approach: The uses of resources are restricted so that system deadlock will never occur .However this is a disadvantages of degrading system performance, because of severe constraints in resources usage.

Deadlock avoidance: Pre-claim strategy used in operating system. And not effecting in database environment.

Deadlock detection: If transaction is blocked is blocked due to another transaction make sure that transaction is not blocked on the first transaction, either directly or indirectly via another transaction. Using four predetermined application program parameters obtained in the program development stage, a directed graph model and a restriction matrix model are introduced representing the usage of common resources.

2. ASSUMPTIONS AND DEFINATION

2.1 Assumptions
Four predetermined parameter relating to the application program must be given in the development stage

a) *Mode of usage for resource in each task.*

Information must be given as to whether each task request common resource in shared usage or exclusive usage.

b) *Sequence of common resources usage in each task.*

c) *Non-concurrent task.*

In multiprogramming environment, a number of task are activated and run simultaneously.

However there are pair o task which are never activated concurrently with one another

d) *Type and number of common resourses.*

In process control computer system it is not difficult to obtain these parameters in the development stage of application program.

2.2 DEFINATION - 1:

Exclusive usage and Shared usage

i) **Exclusive Usage:**
A resource is said to be exclusive usage state, when being used for only one task.

ii) **Shared usage:**
A resource is said to be shared usage state when being used in more than one task.

2.3 DEFINATION - 2:

System deadlock

When a task request a resource ,the system is used to be in a deadlock state if the following two conditions.

1. Either one of the two following situation is encountered.

a) A task request exclusive usage of a resource but the resource is already being used for exclusive or shared usage by one or more other tasks. The request is then suspended.

b) A task requests shared usage of resources, but resource already being used for exclusive usage by another task .The request then suspended.

2. The request for the resource can't be accepted unless the requesting task release at least one of the resources which the task is holding.

3.'wait' relation between resources :

If task T request resource R_j while holding resources R_i (i!=j) it is said that resources Ri is waiting for resource R_j in task T . This situation is denoted by Ri-> R_j. This relation is referred as 'wait' relation between resource.

4. Propagation of wait relation:

Let T_1 and T_2 are two task. If the following two wait relation $R_i \xrightarrow{T1} R_j$ and $R_j \xrightarrow{T2} R_k$ are true and the usage of resource R_j by T_1 and T_2 is in an exclusive usage state, it is said that resource R_i is waiting for resource R_k via resource R_j. This situation is denoted by $R_i \underset{T1}{\rightarrow} R_j \underset{T2}{\rightarrow} R_k$. This is known as propagation of wait relation.

3. DEADLOCK WITH REUSABLE AND CONSUMABLE RESOURCES:

Secondary storage, I/O devices, or processors, and software components, such as data files, tables, or semaphores, are all considered reusable resource
classes. Consumable resources are produced and consumed dynamically, and have the following properties:

1. The number of units within a class varies at runtime; it may be zero and is potentially unbounded.

2. A process may increase the number of units of a resource class by releasing one or more units into that class; i.e., processes create new resources at runtime without acquiring them first from the class.

3. A process may decrease the number of units of a resource class by requesting and acquiring one or more units of that class.

Many types of data generated by either hardware or software have the above characteristics of consumable resources.

4. DIRECTED GRAPH FOR WAIT RELATION AND RESTRICTION MATRIX:

4.1 Directed Graph

Directed graph T_i(i=1,2,….N) represents the i^{th} task and R_j(j=1,2,..M) the j^{th} common resource

After developing the application programs,

A Gantt chart is drawn representing the sequence and the mode of resource usage For resource in each task as shown in fig1.

Next, a directed graph representing the wait relations described in definition 3is constructed from Gantt Chart in the following manner.

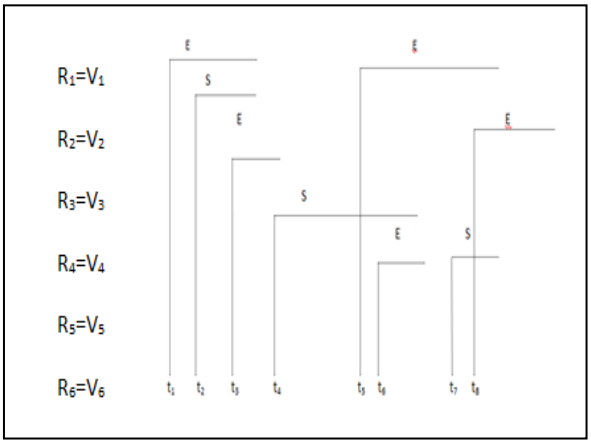


Fig 1: E=Exclusive usage, S=Shared usage, $t_1,t_2....t_8$ = time point resourse requests

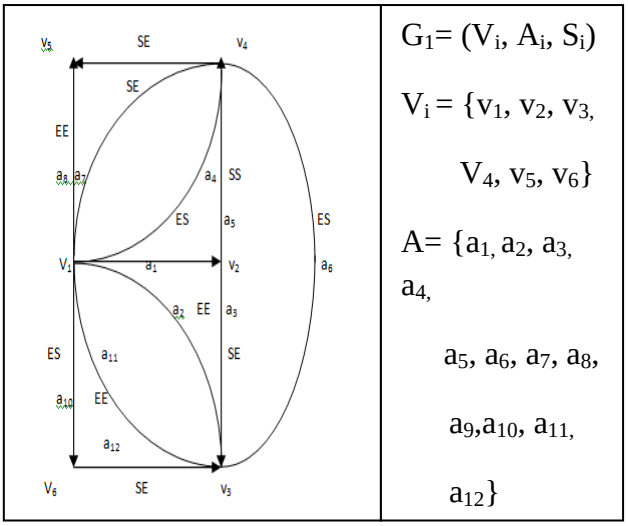
The non propagation matrix is defined as Q where (l,m) are elements of Q :

q_{lm} =
- 1:If arc a_1 is incident into vertex v_k,arc a_m is Incident out of vertex v_k and a_1. a_1 is an ES
- Or SS arc a_m is an SE or ss arc.
- 0: If otherwise.

The non propagation matrix defined above have the same properties that they describe the condition for non-propagation of wait relations in graph G. Thus a **restriction matrix**

R=(r_{lm}) is obtained as follows:

R= P+Q, $r_{lm}=P_{lm}$ V q_{lm}



$G_1 = (V_i, A_i, S_i)$

$V_i = \{v_1, v_2, v_3, V_4, v_5, v_6\}$

$A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12}\}$

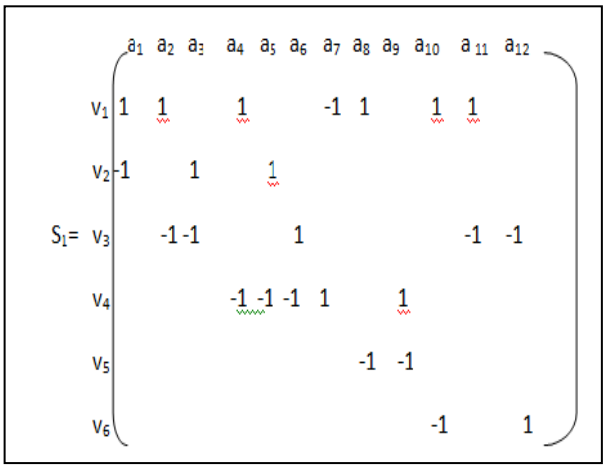


Fig 2: Directed graph representation of wait relations between resources in a task.

4.2 Resource-Allocation Graph Algorithm

Claim edge Pi → R_j indicated that process Pj may request resource R_j; represented by a dashed line.
Claim edge converts to request edge when a process requests a resource.
When a resource is released by a process, assignment edge reconverts to a claim edge.
Resources must be claimed a priori in the system.

4.3 Methods for Handling Deadlocks

- Ensure that the system will never enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

4.4 Deadlock Prevention

Mutual Exclusion – not required for sharable resources; must hold for non-sharable resources.

Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources.

Require process to request and be allocated all its sources before it begins execution, or allow process to request resources only when the process has none. Low resource utilization; starvation possible. Restrain the ways request can be made.

No Preemption – If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.

Preempted resources are added to the list of resources for which the process is waiting.

4.4.1 Safe State

When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
System is in safe state if there exists a safe sequence of all processes.
Sequence <P_1, P_2, …, P_n> is safe if for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with j<I.

1. If P_i resource needs are not immediately available, then Pi can wait until all P_j have finished.

2. When P_j is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate.

3. When Pi terminates, P_{i+1} can obtain its needed resources, and so on.

Properties of Safe States:

• A safe state is not a deadlock state
• An unsafe state may lead to deadlock
• It is possible to go from a safe state to an unsafe state
• Example: A system with 12 units of a resource
  – Three processes
• P1: max need = 10, current need = 5
• P2: max need = 4, current need = 2
• P3: max need = 9, current need = 2
  – This is a safe state, since a safe sequence <P2, P1, P3> exists
  – P3 requests an additional unit. Should this request be granted?
  – No, because this would put the system in an unsafe state

• P1, P2, P3 will then hold 5, 2, and 3 resources (2 units are available)
• P2's future needs can be satisfied, but no way to satisfy P1's and P3's needs
• Avoidance algorithms prevent the system from entering an unsafe state.

5. BASICS OF PETRI NETS

Petri nets are a graphical and mathematical modelling tool applicable to many systems. It is a promising tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic. The main definitions related to Petri net models are introduced in a very compact way.

A Petri net is a 3-tuple N=(P, T, F) where P and T are finite, nonempty, and disjoint sets. P is the set of places and T is the set of transitions. F⊆(P×T)∪(T×P) is called the flow relation. The preset of a node x□P□T is defined as □x={y∈P∪T|(y, x) ∈F}. The postset of a node x∈P∪T is defined as x={y ∈P∪T|(x, y)∈F}.

ELEMENTARY SIPHONS

We distinguish the strict minimal siphons (SMS) in a Petri net by elementary and dependent ones. In the sequel, Π is used to denote the set of strict minimal siphons, while ΠE and ΠD the sets of elementary and dependent ones, respectively. Unless otherwise stated, we refer to a strict minimal one when mentioning a siphon.

6. SUFFICIENT CONDITIONS FOR SYSTEM DEADLOCK PREVENTION:

In this section, sufficient conditions for systems deadlock prevention are discussed for two cases. First, where there is only one resource of each type. Second , where there is more than one resource of each type.

Sufficient conditions for system deadlock prevention where there is only one resource of each type.

Let C= (C_{ij}) directed be a circuit matrix in graph G= (V, A,S).

The given theorem below holds true where only one resource of each type is.

Theorem 1:

Given graph G=(V,A,S) and restriction matrix R, sufficient conditions for system deadlock prevention are:
a) No directed circuit exists in graph G.
b) If directed circuits exists, $C\times R\times C_K^T \neq 0$ for each vector c_k of directed circuit matrix C ($C^T{}_K$: column vector of C_k).

Sufficient conditions for system deadlock prevention where there is more than one resource of each type:

Let us consider

$E=(e^{(1)},e^{(2)},\ldots e^{(m)})$ showing the number of resource available in the system;
$e^{(m)}$= number of type of m resources (m=1,2,……M) .

Theorem 2:

When graph G, restriction matrix R, number of resources of each type E , u_{ij} ,r_{ij} are given, the sufficient conditions for system deadlock prevention are: (1)G and R meet the conditions in Theorem 1 or 2 if G and R do not meet the conditions in Theorem 1 for directed circuits $c_1,c_2,\ldots,ck\ldots ck$,there is a full sequence α(k) where

$$\exists_m\ 0 < r_k^{(m)} \le e^{(m)} - u^{(m)}{}_{\alpha(k)}{}^{(l)}\ k=1,2,3\ldots..$$

Where

$$U^{(l)}{}_{\alpha(k)} = \sum \left\{ \bigcup_{x(n)\le x(l)} ij \,\middle|\, Aij \sqcap \bar{w}n \neq \emptyset \right\}^{u}{}_{ij}$$

$$= (\cup_{\alpha}^{1}, (k), \cup_{\alpha(k)}^{l} \ldots\ldots \cup_{\alpha(k)} (m),)$$

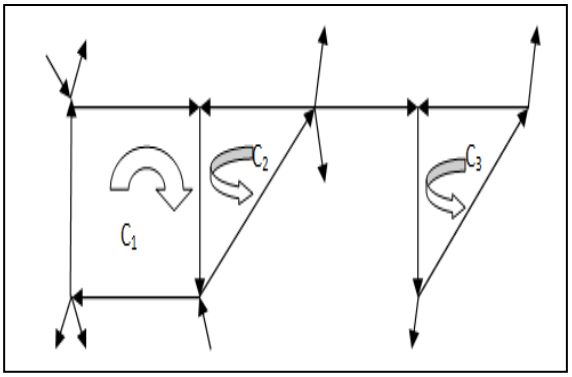
$$\gamma_k = \sum \{ij | A_{ij} \cap \bar{w}_k = \emptyset\} v^{ij}$$

$$= (\gamma_k{}^{(l)}, \gamma_k{}^{(2)}, \ldots. \gamma_k{}^{(m)})$$

$$\bar{\omega}_k = \{a_i | c_{lk} = 1\}$$

When the total number of resources allocated to the task in the circuit is

$$\left\{ \bigcup_{i=1}^{k} ij / Aij \cap^{\Sigma} Wk \neq \emptyset \right\}^{\cup}{}_{ij}$$



Situation when 'wait' relations in circuit occur simultaneously

Theorem 3

Where a full sequence α(k) which meets the conditions in theorem 2 does not exist, the sufficient conditions for system deadlock prevention are as follow . There exists a full sequence $\alpha^{(l)}(k)$ which meets the following conditions for each subset C(l)∈C such that

$$C^{(l)} \times p \times C^{(l)\,T} = 0$$

$$\exists_m\ 0 < r_k^{(m)} \le e^{(m)} - u^{(m)}{}_{\alpha(k)}{}^{(l)}\ k=1,2,3\ldots..$$

Where

$$U^{(l)}{}_{\alpha(k)} = \sum \left\{ \bigcup_{x(n)\le x(l)} ij \,\middle|\, Aij \sqcap \bar{w}n \neq \emptyset \right\}^{u}{}_{ij}$$

$=(U_{\alpha}^{1}, (k), U_{\alpha(k)}^{l} \ldots\ldots U_{\alpha(k)}(m),)$

$\gamma_k = \sum$

$\{ij \mid A_{ij} \cap \overline{w}_k = \emptyset\}\ v^{ij}$

$=(\gamma_k^{(l)}, \gamma_k^{(2)}, \ldots \gamma_k^{(m)})$

$\overline{\omega}_k = \{a_l \mid c_{lk} = 1\}$

And the subscripts of each element in $C^{(1)}$ are reassigned as:

C(l)={$\hat{C}_1^{(l)}$,$\hat{C}_2$(l)} and $W_n^{(l)}$ is a

set of arcs which make up $\hat{C}_n^{(l)}$.

7. DEADLOCK AVOIDANCE

• Main idea:

– request additional information about how resources are to be requested

– before allocating request, verify that system will not enter a deadlock state
(resources currently available,
resources currently allocated,
future requests and releases)

• if no: grant the request

• if yes: block the process

• Algorithms differ in amount and type of information

– simplest (also most useful) model: maximum number of resources

– other choices

• sequence of requests and releases
• alternate request paths

7.1 Limitations of Deadlock Avoidance

• Deadlock avoidance vs. deadlock prevention

– Prevention schemes work with local information.

• What does this process already have, what is it asking

– Avoidance schemes work with global information.

• Therefore, are less conservative.

• However, avoidance schemes require specification of future needs.

– not generally known for OS processes

– more applicable to specialized situations

• programming language constructs (e.g., transaction-based systems)

• known OS components (e.g., Unix "exec")

APPLICATION

The algorithm for checking deadlock possibility was developing by introducing three theorems mentioned above.

8. CONCLUSION AND FUTURE SCOPE:

In computer science, deadlock refers to a specific condition when two or more processes are each waiting for the other to release a resource, or more than two processes are waiting for resources in a circular chain. Deadlock is a common problem in multiprocessing where many processes share a specific type of mutually exclusive resource known as a software lock or soft lock. The future work includes developing some effective and efficient deadlock prevention policies based on elementary siphons in a Petri net. The problem of deadlock detection in distributed systems has undergone extensive study. In this paper we have tried to get rid of on distributed deadlock by studying the performance representative algorithms. Using four predetermined application program parameters, a directed graph model representing the usage of common resources by tasks and a restriction matrix model were presented Sufficient conditions for system deadlock prevention were mentioned. The methods in this paper is useful in process control computer systems because the algorithms introduced do have to be executed in real time mode, and the restrictions on the usage of common resources are applied when deadlock possibility is found. Thus additional side effects on the responsiveness of the system will not arise and the proper and higher utilization rate of common resources is guaranteed.

9. ACKNOWLEDGEMENT

The authors would like to acknowledge valuable discussions with Prof. Binayak Sahu, Prof. Amarnath Singh, Prof. Tanushree Mohapatra and Prof. Neelamani Samal for their effort and cooperation in making this paper successful. This work was supported by Gandhi Institute for Education and Technology (GIET), Bhubaneswar.

10 REFRENCES

[1] E.G. Coffman, M.J. Elphick . System deadlock and computing surveys.

[2] Operating system concepts ,Greg Gagne, Peter B. Galvin.

[3] J.W. Murphy, Resource allocation with interlock detection in a multitasking system.

[4] Dijkstra N.W, and Scholten C.S., Termination detection for diffusing computations, Inf. Process. Lett., 11(1), 1-4 (1980)

[5] Goldman B., Deadlock detection in computer networks. Tech. Rep. MIT-LCS-TR185, Massachusetts Institute of Technology, Cambridge, Mass., (1977)

[6] Gray J.N., Notes on database operating systems. In Operating Systems: An Advanced Course, Lecture Notes in Computer Science, Springer-Verlag, New York, 60, 393-481 (1978), Science Dept., Univ. of Texas at Austin, July (1981).

[7] HOARE, C.A.R. Communicating sequential processes Commun. ACM21, 8, 666-677 (1978)

[8] B. Abdallah and H. A. ElMaraghy, "Deadlock Prevention and Avoidance in FMS: a Petri net based approach", *Int. J Manuf. Tech.*,

[9] Li, Z. W., Uzam, M., and Zhou, M. C., "Comments on 'Deadlock prevention policy based on Petri nets and siphons' ", *Int. J. Prod. Res*., Vol. 42, No. 24, 2004, pp. 5253–5254.

[10] Murata, T., "Petri nets: properties, analysis, and applications".