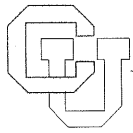
Static Detection of Deadlocks *

Ashok R. Saxena

CU-CS-122-77



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

* This work supported by NSF grant DCR 75-09972.

STATIC DETECTION OF DEADLOCKS

Ashok R. Saxena
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309

CU-CS-122 -77 December 1977

This work supported by NSF grant DCR 75-09972

ABSTRACT

The problem of deadlocks is an important factor in the design of multiprogramming computer systems. Current techniques for preventing deadlocks, without run-time overhead, are based on violating one of the necessary conditions for deadlocks to occur. One such technique is the linear ordering of resources $R_1,\ldots,R_M$ and the restriction that a process cannot request resource R_j if it currently holds a resource R_i such that $j \leq i$. In this paper we present an algorithm to detect if any such ordering exists for a given set of flow graphs representing concurrent processes. Even if no such ordering exists it can sometimes be shown that there can be no deadlocks. A graph model for resource requests in a system of processes is developed. It is shown that if this graph is acyclic then there can be no deadlocks. Further, the notions of exclusive and reducible components of the resource graph are defined. It is shown that if the resource graph consists only of these two type of components then there can be no deadlock. Algorithms to detect these components are also given. It should be noted that these are static algorithms and do not require run-time overhead.

I. INTRODUCTION:

In a computer system allowing concurrent processes to share resources, situations may develop such that the progress of one or more processes will be blocked forever. Such situations are called deadlocks. For example, consider a system with two concurrent processes P1 and P2, each requiring exclusive access to resources R1 and R2 (R1 and R2 may be files, for example). If process P1 requests resource R2 while holding resource R1 and process P2 requests resource R1 while holding resource R2, and neither process can release the resources they hold without first acquiring the requested resources, the system will be in a deadlock state as the progress of processes P1 and P2 will be blocked forever.

Necessary conditions for existence of deadlocks have been enunciated [Coffman et al. 1971]. Algorithms for dynamic detection [Shoshani and Coffman 1970a, Holt 1971] and prevention [Havender 1968, Shoshani and Coffman 1970b, Habermann 1969, Holt 1971] of deadlocks under various assumptions have also been reported. It is well known that if a system has resource types $R_1,\ldots,R_M$, and they can be partitioned into classes $k_1,\ldots k_\ell, \ell \leq m$, such that if a process holds a resource in class k_i, it can only request resources in classes k_j, $1 \leq i \leq j \leq \ell$, then there will be no deadlock.

In this paper an algorithm is presented to detect if an ordering of resources can be obtained that satisfies the above condition, given the flow graphs for the processes in the system. These flow graphs can be derived easily if the processes are specified in a language like concurrent PASCAL. Even when there is no ordering of resources that

satisfies the above condition it can sometimes be shown that there can be no deadlocks. Algorithms to detect these conditions are also given. These algorithms are applied to the flow graphs of the processes and analyze the system statically. It is assumed that resources are requested one unit at a time and are granted whenever resources are available. It is also assumed that the resource allocation policy is "fair" (such as first come first served) and does not discriminate among competing processes.

II. GRAPH MODEL

Consider a system of N concurrent processes, $Q_1,\ldots,Q_N$ and M resources, $R_1,\ldots,R_M$. A process consists of a series of acquisitions and releases of these resources. $R_1,\ldots,R_M$ are types of resources. There may be more than one unit of each type of resource. In the programs of $Q_1,\ldots,Q_N$ these resources will be represented by variables $r_1,\ldots,r_m$; r_i initialized to count(r_i), the number of units of resource R_i in the system. Acquisition of resource R_i is represented by the statement "$r_i \leftarrow r_i-1$" and release of resource R_i by "$r_i \leftarrow r_i+1$". Resources may be acquired and released only in single units. The statements "$r_i \leftarrow r_i-1$" and "$r_i \leftarrow r_i+1$" are indivisible. A release operation, $r_i \leftarrow r_i+1$, can always be executed. An acquisition operation, $r_i \leftarrow r_i-1$, can only be executed when the resource variable r_i is greater than zero. Thus resource variables are nonnegative integer variables. When an acquisition operation cannot be executed by a process because the resource variable, say r_j, is zero, the process is <u>blocked</u>. Further, it is <u>blocked on resource</u> r_j. A blocked process cannot continue its execution until it is unblocked: it becomes unblocked when some other process in the system executes a release operation on r_j causing r_j to become greater than zero, thus permitting execution of the acquisition operation.

A blocked process is deadlocked if it can never be unblocked, that is, if it can never execute the acquisition operation that caused it to be blocked. A system of processes is deadlocked if one or more processes are deadlocked. In the following it is assumed that if a resource is continually acquired and released, then no process can be deadlocked on that resource, that is, the resource allocation policy is "fair" (such as first come first served) and does not discriminate among competing processes.

It is assumed that each process can be considered to execute a nondeterministic sequential program that can be represented by a finite directed acyclic graph G = (N,E) where nodes represent operations (acquire or release) and edges represent transitions to next operations.

An acquisition operation $r_i \rightarrow r_i - 1$, is written as $P(r_i)$ and a release operation, $r_i \rightarrow r_i + 1$, is written as $V(r_i)$. The graph G contains special nodes S and F that represent a null operation. S is an entry (root) node and no edge can be incident on S. F is a terminating node and no edge leaves it. If S,...,a is any path in a graph G, then the notation $|S,...,a|_{P(r_i)}$ represents the number of acquire operations on the resource r_i in the path S,...,a; and the notation $|S,...,a|_{V(r_i)}$ represents the number of release operations on the resource r_i in the path S,...,a. A flowgraph G is an SR-graph if and only if it satisfies the following three properties.

i) For all a such that S,...,a is a path in G, and for all r_i, $1 \leq i \leq M$:

$$|S,...,a|_{P(r_i)} \geq |S,...,a|_{V(r_i)}$$

ii) For all a such that S,...,a is a path in G, and for all r_i, $1 \leq i \leq M$:

$$|S,...,a|_{P(r_i)} - |S,...,a|_{V(r_i)} \leq Count(r_i)$$

iii) For all paths S,...,F in G and for all r_i, $1 \leq i \leq M$:

$$|S,...,F|_{P(r_i)} = |S,...,F|_{V(r_i)}$$

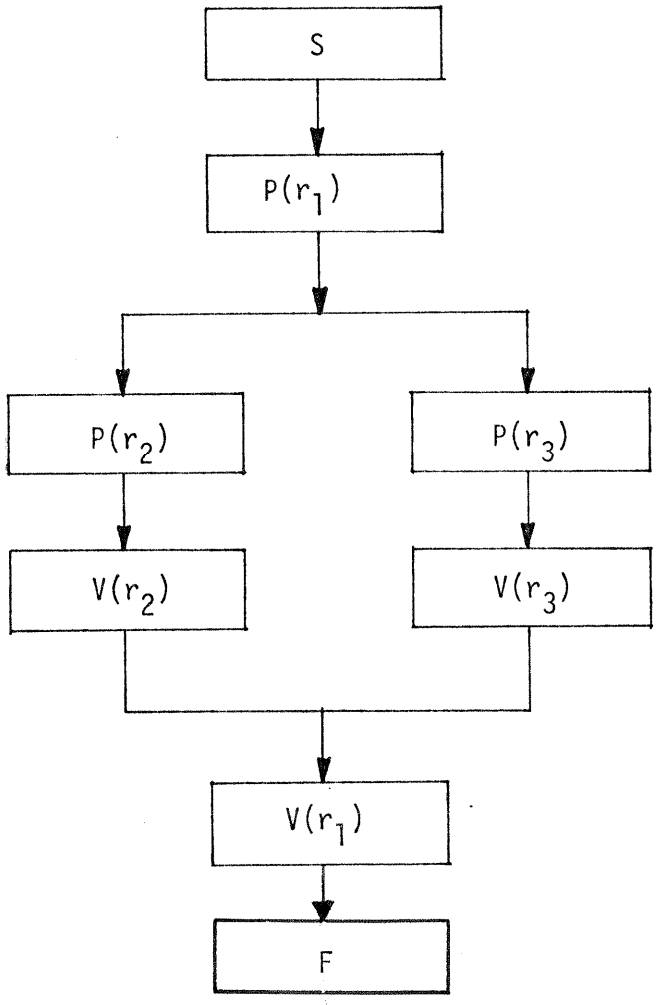


Figure 1

Flowgraph of a process using resources r_1, r_2, r_3.

These restrictions imply that in an SR-graph a resource can never be released unless it was acquired earlier and that all resources acquired are released when node F is reached. Further, no process can request more resources than are available in the system. The SR-graph G is a model for a process that acquires resources one unit at a time, in a system with serially reusable resources and with a resource allocation policy that grants requests whenever resources are available and does not discriminate among competing processes. An example of an SR-graph is given in Figure 1. All flow graphs considered in this paper are assumed to be SR-graphs.

Generally, the processes of interest will be cyclic and execute forever. However, it can be easily seen that if they satisfy the above conditions then considering them to be acyclic does not affect any of the results obtained in this paper, for before a process starts a new iteration it would have released all resources acquired and the set of system states considered will be identical. In fact, we can model any loop in the process as an acyclic segment in the flow graph if the following condition is satisfied in the loop: all paths from an entry to an exit in the loop are such that the resources held by the process at entry to the loop are the same as the resources held by it at exit from the loop. That is if $x,\ldots,y$ is any such path, then

$$\forall\, r_i(|x,\ldots,y|_{P(r_i)} = |x,\ldots,y|_{V(r_i)})$$

An example is given in Figure 2.

Given the graphs $G_i = (N_i,E_i)$ for each process Q_i, $1 \le i \le N$ in a system $S_{N,M}$ with N processes and M resources, construct a directed graph $G_s = (N_s,E_s)$ where

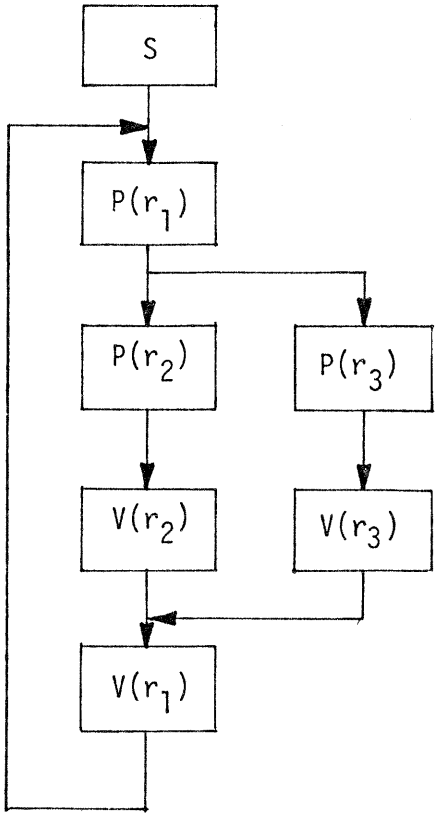
$N_s = \{r_1,\ldots,r_M\}$ and

$(r_j,r_k) \in E_s$ if and only if $\exists\, G_i$ and a path $S,\ldots,a,b$ in G_i such that

$b = P(r_k)$ and $|S,\ldots,a|_{P(r_j)} > |S,\ldots,a|_{V(r_j)}$.

That is, in the graph G_s there is an edge from r_j to r_k if, in any process, resource r_k is acquired while holding resource r_j. An example is given in Figure 3 for the flow graphs of Figure 2.

Process P1:



Process P2:

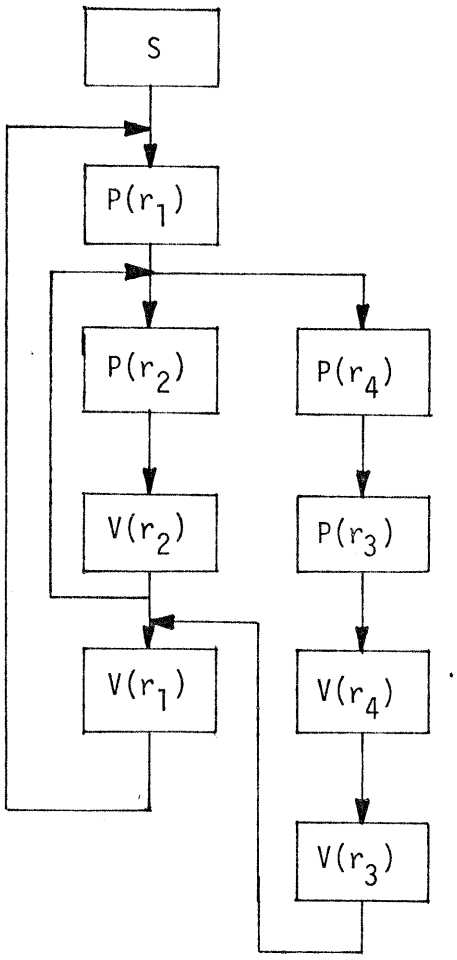
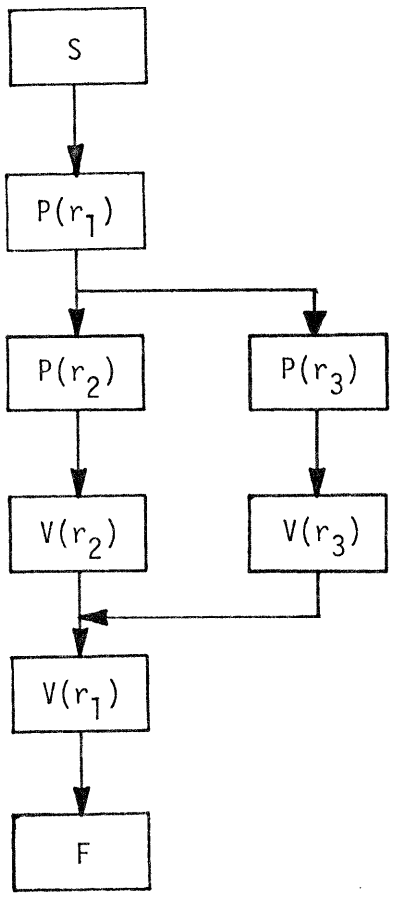


Figure 2 part 1

Acyclic graph for process P1



Acyclic graph for process P2

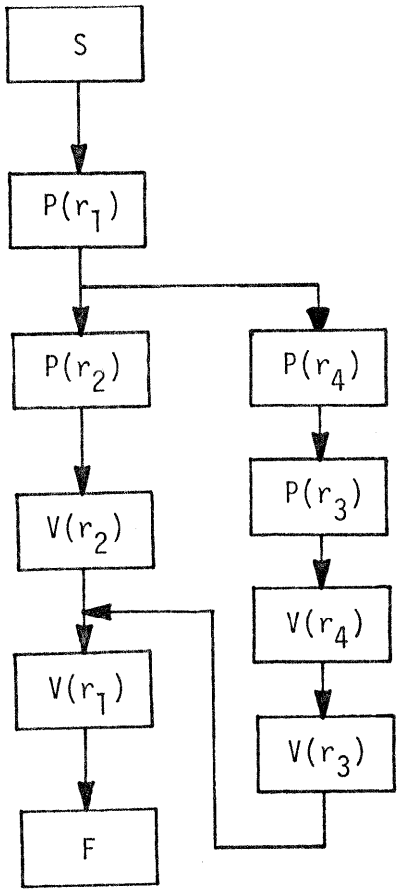


Figure 2 part 2

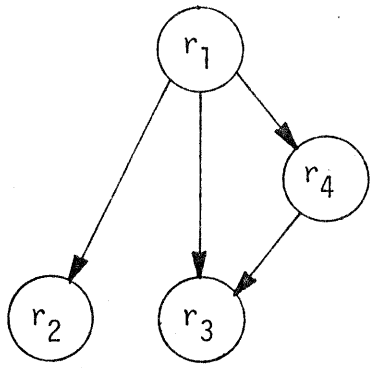


Figure 3

Graph G_s for the system of Figure 2.

Theorem 1: if G_s is acyclic then there is no deadlock in the system $S_{N,M}$.

Proof: G_s is acyclic implies that the nodes of G_s can be arranged in a sequence $a_1,\ldots,a_M$ such that for $1 \leq i, j \leq m$ and $j \geq i, (a_j,a_i) \notin E_s$. Hence no graph $G_k (1 \leq k \leq N)$ contains a path $S,\ldots,a,b$ such that:

$$b = P(a_i) \text{ and } |S,\ldots,a|_{P(a_j)} > |S,\ldots,a|_{V(a_j)}.$$

That is, in every path in every process that resource a_M is acquired, it is released before the acquisition of any other resource $a_i, 1 \leq i \leq M$.

Hence no process can forever be blocked on the resource a_M. By induction no process can forever be blocked on resources $a_{M-1},\ldots,a_1$. Hence there is no deadlock in the system $S_{N,M}$.

■

For example, the graph in Figure 3 is acyclic and its nodes can be arranged in the sequence r_1, r_4, r_2, r_3. No process requests a resource while holding either resource r_2 or r_3. No process requests resources r_1 or r_4 while holding r_4 and no process requests resource r_1 while holding r_1. Thus there is no deadlock in the system represented by the graph of Figure 3. It should be noted that if G_s is acyclic then only one unit of a resource can be held by a process.

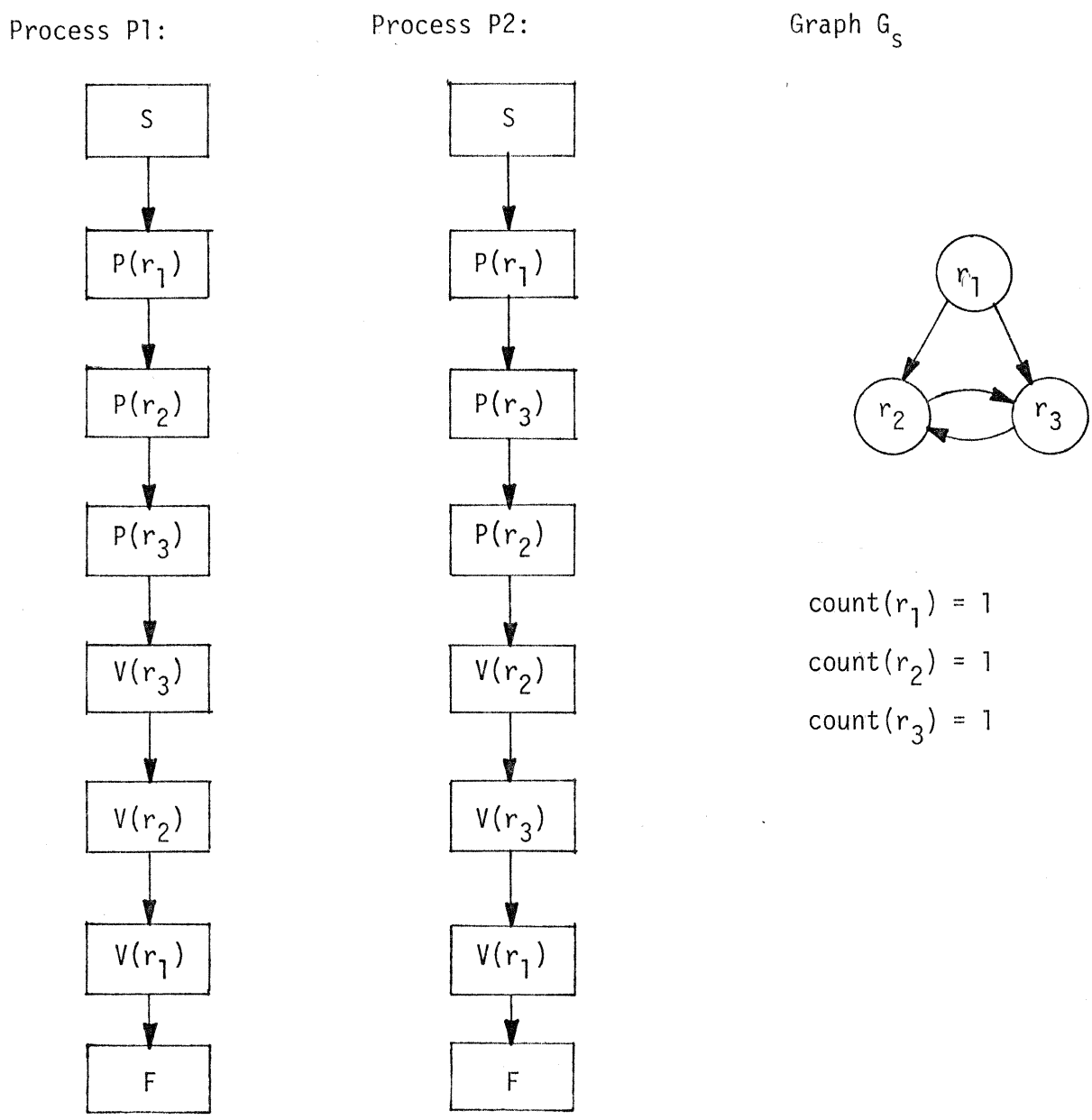


Figure 4

If the graph G_s has cycles it does not necessarily mean that there are deadlocks in the system. In fact,under certain conditions we can demonstrate that there will be no deadlocks. In the example of Figure 4, the cycle in G_s arises because of resources r_2 and r_3. However, observe that both processes, P1 and P2, attempt to acquire r_2 and r_3 only after acquiring resource r_1. Hence, the process that acquires r_1 will be able to acquire r_2 and r_3 without any competition from the other process and

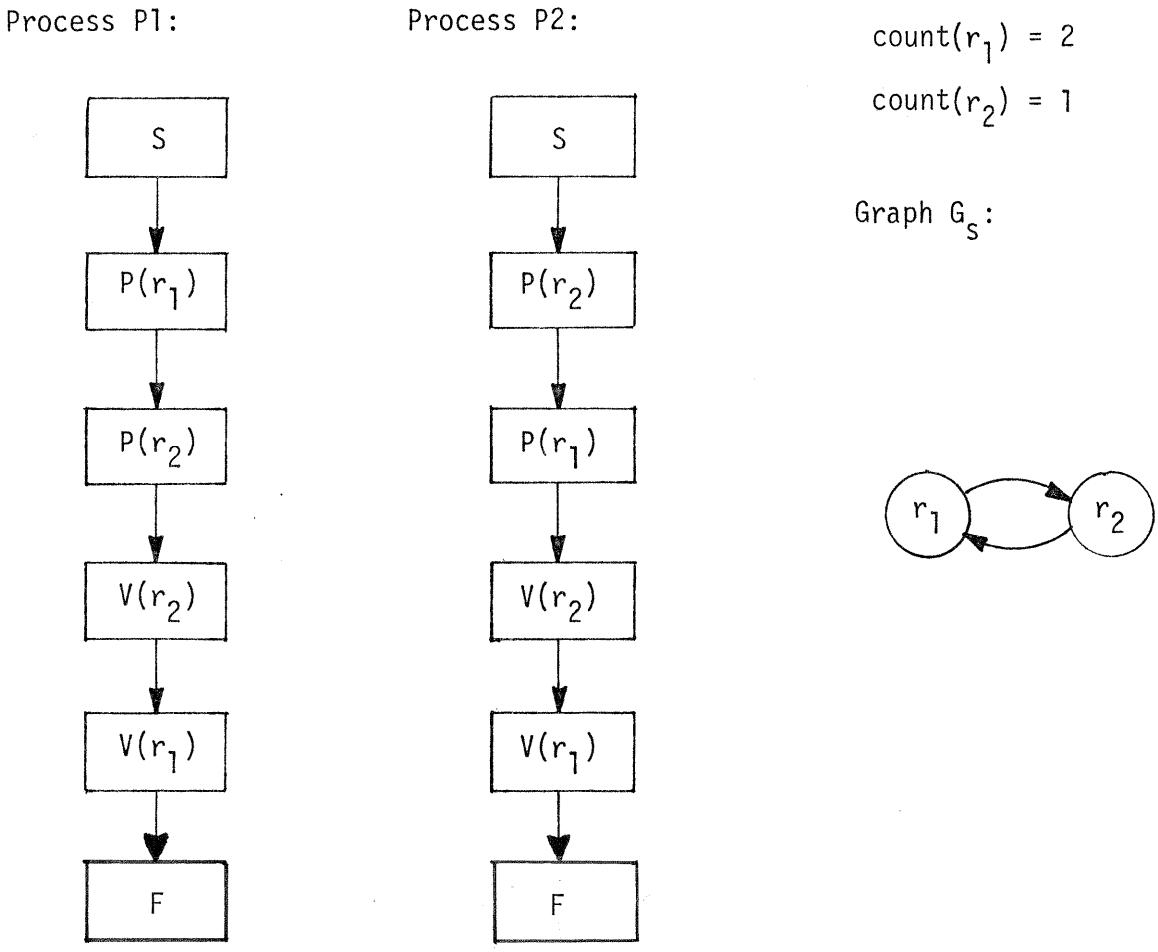


Figure 5

there can be no deadlock. An algorithm to detect such conditions is given in the next section. In the example of Figure 5, process P1 needs to acquire resource r_2 while holding resource r_1 while process P2 needs to acquire r_1 while holding r_2. If there were only single units of r_1 and r_2 in the system, we would have a potential deadlock. However, in the example there are two units of resource r_1; hence if P2 acquires resource r_2 it will be able to acquire one unit of resource r_1 and eventually release resource r_2 for use by process 1. In this case there is no deadlock even though there is a cycle in G_s. An algorithm to detect such conditions is given in a following section.

III. DETECTION OF EXCLUSIVE COMPONENTS OF THE GRAPH G_s

Let $c_1,\ldots,c_n$ be the strongly connected components of the graph $G_s = (N_s E_s)$. Let C_D be the set of resources r_i such that r_i is a single unit resource. Let $g_k(r_j,r_i)$ be the minimum number of resources of type r_j held by process k while requesting a resource of type r_i.

A component c_i of the graph G_s is an exclusive component if and only if $\exists r' \forall k \forall r (r \in c_i \wedge r' \in C_D \wedge 1 \leq k \leq N \wedge g_k(r',r) > 0)$.
That is, a component c_i is an exclusive component if and only if there exists at least one single unit resource, say r', that is held by every process k, $1 \leq k \leq N$, before acquiring any resource, say r, of the component c_i. The resource r' dominates the component c_i. Note that resource r' cannot be a member of the component c_i. The only access to resources of an exclusive component c_i is through its dominating resource r'. If the dominating resource r' is released by a process, then it must also release all currently held resources of the exclusive component before reacquiring the resource r'.

It is clear that in a system represented by the graph G_s, only one process has access to resources in an exclusive component at any time. As long as there are sufficient resources to satisfy the maximum demands of each process, which is true for any SR-graph, the resources in an exclusive component cannot lead to deadlock situations, as proved in the Lemma below. Thus, the resources in an exclusive component can be ignored. For the example of Figure 4, the components are: (r_1) and (r_2,r_3). The component (r_2,r_3) is an exclusive component, dominated by resource r_1, and there is no deadlock.

Lemma 1:

The resources of an exclusive component cannot cause deadlock.

Proof:

The only access to resources in an exclusive component is through a single unit resource, by definition of an exclusvie component. That is, any process acquiring a resource in an exclusive component must first acquire a particular single unit resource and if it releases the single unit resource, then it must also release all resources of the exclusive component before reacquiring the single unit resource. Thus a process is either blocked on the single unit resource dominating the exclusive component orit can have all its requests for the exclusive component resources satisfied. Note that no process can demand more units of a resource type than are available in the system.

∎

IV. REDUCIBLE COMPONENTS

A strongly connected component c_i of the graph G_s is reducible if no process can be blocked forever on the resources in the component $r_1,\ldots,r_\ell$, provided no process can be blocked forever on resources in other components requested while holding resources in c_i. A procedure to determine a reducible component is given below.

Let $r_1,\ldots,r_\ell$ be the resources in a strongly connected component of G_s.

Let $h_k(r_i)$ be the maximum number of resources of type r_i held by process k while requesting some resource r_j; r_i, $r_j \varepsilon (r_1,\ldots,r_\ell)$. That is, $h_k(r_i) = \max_{(S,\ldots,b) \text{ in } G_k\left(|S,\ldots,b|_{P(r_i)} - |S,\ldots,b|_{V(r_i)}\right)}$

where $b = P(r_j)$ and r_i, $r_j \varepsilon (r_1,\ldots,r_\ell)$

Let $H(r_i) = \sum_{k=1}^{N} h_k(r_i)$.

Algorithm R:

0. Let C be the set of nodes $(r_1,\ldots,r_\ell)$.
1. Compute the functions $H(r_i)$ for all $r_i \varepsilon C$.
2. Remove all nodes r_i from the set C such that $H(r_i) < \text{count}(r_i)$. If any nodes were removed go to step 3 else the component is not reducible.
3. If the set C is empty then the component is reducible. If the set C is not empty then in the flow graphs of all processes, replace all operations on the resources represented by the nodes removed in step 2 by "null" operations (that is, in the flow graphs of all processes, eliminate the nodes that represent operations on the removed resources by making an edge incident on such a node incident on all nodes that had an edge from the node to be eliminated) and go to step 1.

Lemma 2:

Algorithm R determines a reducible component.

Proof:

Let $r'_1,\ldots,r'_\ell$ be a sequence in which the nodes $r_1,\ldots,r_\ell$ can be removed from the set C. Then initially $H(r'_1) < count(r'_1)$ hence the maximum number of r'_1 units that can be acquired while waiting for another resource in the set C, is less than the total number of units of r'_1 in the system. So no process can be blocked forever on resource r'_1. Given that no processes can be blocked forever on resources $r'_1,\ldots,r'_i$ it follows that if the recomputed $H(r'_{i+1}) < count(r'_{i+1})$ then no process can be blocked forever on r'_{i+1}. Hence algorithm R determines a reducible component. For the example of figure 5, $\{r_1,r_2\}$ is a reducible component, since

$H(r_1) = H(r_2) = 1$

$count(r_1) = 2,\ count(r_2) = 1$

Initially, $H(r_1) < count(r_1)$

after removing r_1, $H(r_2) = 0$

and $H(r_2) = 0 < count(r_2) = 1$

The ordering r'_1, r'_2 is r_1,r_2.

■

Theorem 2:

There are no deadlocks in the system $S_{N,M}$ if all strongly connected components of the corresponding graph G_s are either

(i) exclusive components or

(ii) reducible.

Proof:

The components of the graph G_s $(c_1,\ldots,c_n)$ can be ordered such that if component c_j appears after component c_i then there is no edge from any node in c_i to any node in c_j. That is, no process requests a resource in the component c_j while holding resources in the component c_i. Let one such ordering be $c_1,\ldots,c_n'$. No process can be blocked forever on the resources in c_1, for if c_1 is an exclusive component then by Lemma 1 it can be ignored, and if c_1 is a reducible component then, since no resources are requested in other components while holding resources of c_1', it follows from the definition of a reducible component that no process can be blocked forever on its resources. Similarly, given that no process can be blocked forever on the components $c_1',\ldots,c_i'$, it can be shown that no process can be blocked forever on the resources of component c_{i+1}'. Hence by induction there is no deadlock in the system $S_{N,M}$.

∎

V. CONCLUSION

The flowgraphs for the processes in the system can be constructed automatically from their specification in a language like PASCAL. It is proposed that a system be built to construct G_s from these flowgraphs. If after applications of theorems 1 and 2 it cannot be shown that there is no deadlock, that is, there are strongly connected components in G_s that are neither exclusive nor reducible, the exercise will help identify the resources involved in such components. We can then either attempt to redesign the system to eliminate the cycles or increase the available resource units in the system so as to make the components reducible.

VI. ACKNOWLEDGEMENTS

The author would like to thank Lloyd D. Fosdick and William E. Riddle for their comments on earlier drafts of this paper. Thanks are also due to Lloyd D. Fosdick for his encouragement of the work reported here. The support of National Science Foundation for this work is gratefully acknowledged.

REFERENCES

[Coffman et al. 1971]

Coffman, E. G., Jr., Elphick, M. J., and Shoshani, A. "System Deadlocks." ACM Computing Surveys 3,2 (June 1971), 67-78.

[Habermann 1969]

Habermann, A. N. "Prevention of System Deadlocks." Comm. ACM 12,7 (July 1969), 373-377, 385.

[Havender 1968]

Havender, J. W. "Avoiding deadlocks in multi-tasking systems." IBM Systems Journal 2 (1968), 74-84.

[Holt 1971]

Holt, R. C. "On deadlock in computer systems." (Ph.D. Dissertation) Department of Computer Science, Cornell University, Ithaca, N. Y., Jan. 1971.

[Shoshani and Coffman 1970 a]

Shoshani, A. and Coffman, E. G. "Prevention, detection, and recovery from system deadlocks." In Proc. 4th Annual Princeton Conf. on Information Sciences and Systems, March 1970.

[Shoshani and Coffman 1970 b]

Shoshani, A. and Coffman, E. G. "Sequencing tasks in multi-process systems to avoid deadlocks." In Proc. 11th Annual Symposium on Switching and Automata Theory, Oct. 1970, 225-233.