AN ABSTRACT OF THE THESIS OF

Effiong James Akpan Edemenang for the degree of Doctor of Philosophy

in Computer Science presented on November 10, 1982

Title: On-line Deadlock Detection in Distributed Computer Systems

Redacted for Privacy

Abstract approved: ______________________
Theodore G. Lewis

A new algorithm, the Horizontal and Vertical Algorithm, for on-line detection of deadlocks in distributed computer systems, is presented. Two protocols for implementing the algorithm are given. The first protocol, the centralized protocol, is based on the assumption that one site in the network acts as the controller for global resource allocation and deadlock detection. The second protocol, the distributed protocol, distributes the responsibilities of resource allocation and deadlock detection among the sites where the requested resources reside.

The new deadlock detection protocols have two important features. Both protocols are characterized by their simplicity in implementation as compared to most published protocols. The storage requirement needed to run the distributed protocol is considerably reduced. The distributed protocol is also characterized by a significant reduction of communication messages passed around the different sites in the network.

The new algorithm is compared with the distributed algorithm proposed by Barry Goldman and the preemption method of deadlock prevention on a ring network. The comparison was made by means of simulation models. Simulation models are developed for both the centralized and distributed control of the new algorithm, Goldman's

algorithm and the preemption technique.

The performances of the algorithms are measured in terms of process response time--average delay per process, and process throughput--the number of processes completed per unit time. Resource request response time--average time to process a resource request and throughput--the number of requests processed per unit time are also measured. Communication overhead associated with the use of each algorithm and frequency of deadlock occurrence are also measured.

The simulation results, for the distributed Horizontal and Vertical algorithm, are used to develop an M/M/z queueing model to measure the request response time of the algorithm. This is done by a regression technique. The results of the analytical model show a very close fit with the results of the simulation model.

On-Line Deadlock Detection in Distributed Computer Systems

by

Effiong James Akpan Edemenang

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philoscphy

Completed November 10, 1982

Commencement June 1983

APPROVED:

Redacted for Privacy

Professor of Computer Science
in charge of major

Redacted for Privacy

Head of Department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis is presented November 10, 1982

Typed by Donna Lee Norvell-Race for Effiong James Akpan Edemenang

DEDICATION ....

This dissertation is dedicated to the memory of my dear mother who died in September, 1981.

She toiled all her life to see me achieve this goal, but did not live long enough to reap the fruits of her labor.

May Almighty God grant her eternal rest.

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to my father, Mr. J. A. Edemenang, for his love, patience and full moral and financial support, without which this goal may not have been achieved.

My sincere thanks and appreciation are extended to my major professor, Dr. T. G. Lewis, for his guidance and contributions throughout this work. I thank him for his tremendous support in times of need.

I also want to thank the other members of my committee, Dr. C. Cook, Dr. M. Freiling and Dr. B. Bose, for their contributions to this work.

I also wish to express my gratitude to my friends and relatives who, at one time or another, contributed morally, financially and otherwise toward the achievement of this goal.

I wish to take this opportunity to mention my two sons, Ubong and Ini. They have introduced a new sense of happiness and responsibility into my life. I want them to know that I love them very much.

To my brother, Samuel, and my sisters Nkoyo, Arit and Adiaha, I am highly indebted for their support and love, without which this goal may not have been reached.

My appreciation and love are extended to my dear wife, Comfort, for her love, patience and encouragement throughout this work. I wish to congratulate her for her ability to withstand my occasional frustration, and her steadfastness, even in the face of extremely difficult situations.

Finally, I thank the Almighty God for making it all possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ON-LINE DEADLOCK DETECTION IN DISTRIBUTED COMPUTER SYSTEMS

I. INTRODUCTION

The growing importance of distributed computer systems has increased the importance of on-line deadlock detection in such systems. On-line detection of deadlocks in distributed computer systems is the recognition of an occurrence of deadlock as requests for resources are made or granted by both local and remote resource managers, with minimum amount of communication among the different sites in the network. The detection mechanism may involve running a detection algorithm every time a resource requested for is not free for immediate allocation to determine if it is safe for the requesting process to wait for the resource. Alternatively, the detection algorithm may be run periodically. Whichever method is used depends on the installation's implementation. The algorithms and simulation models developed in this thesis address the former problem.

The concept of on-line detection was introduced by Isloor and Marsland [40], [42], and [57]. Many researchers have addressed deadlock problems in both centralized and distributed systems. Solutions and counterexamples to some of the solutions have been published. But very few researchers have taken time from theoretical studies to measure the performance of the proposed solutions, and the relative probability of interference and deadlock. "A comprehensive probabilistic model for computer deadlocks of large systems has not yet appeared in the literature" [42].

Since distributed systems are not widely available, experimental data cannot be gathered in practical environments to measure these performances. Models have to be devised to do this. This thesis provides some simulation data, on the operational behavior of the new algorithms proposed and the distributed algorithm proposed by Goldman [28], in a distributed computer system. Goldman's algorithm was chosen because of its relative simplicity in implementation as compared to most of the other published algorithms.

Firstly, some basic information and previous work on deadlock will be reviewed. Deadlock, or "deadly embrace" according to Dijkstra [19], is an important concept in the design and operation of any distributed computer system. An often cited example of a deadlock is the case where a process, P1, has control of resource R1, but cannot proceed until it obtains control of resource R2. At the same time a second process, P2, which has control of resource R2 must wait until it obtains control of R1 to proceed. P1 and P2 are assumed to be running concurrently either on two different sites on a network or on the same site. It is apparent that none of the two processes will ever run to completion unless something is done to break the wait. Deadlock involves circular waiting. Each process is waiting for a condition which can only be satisfied by one of the others. But since each process expects one of the others to resolve the conflict, they are unable to continue.

Deadlock problem was first recognized and analyzed by Dijkstra [19]. Before then the problem was not very well understood and many deadlocks were programmed into some operating systems. Lynch [53] said about EXEC II:

> Several problems remained unsolved with EXEC II operating system and had to be avoided by an ad hoc means or another. The problem of deadlocks was not at all understood in 1962 when the system was designed. As a result several annoying deadlocks were programmed into the system.

Even after the problem was recognized, some installations did nothing during the design of their operating systems to resolve the problem because of the cost involved. Since deadlock occurred infrequently, it appeared the cheapest way of resolving it was by removing one or more processes. Hansen [31] maintains that

> the difficulty with this point of view is that no methods are available at the moment for predicting the frequency of deadlocks and evaluating the costs involved. In this

> situation, it seems more honest to design systems in which deadlocks cannot occur.

The recognition of the deadlock problem resulted in many papers on the solution being published in the literature. Among the earlier ones are the works reported by Coffman et al. [16], Habermann [30], Havender [32], Holt [34], [35], [36], Howard [38], Hutchison [39], Murphy [62], and Russell [67]. The deadlock detection algorithm by Murphy [62] is basically an exhaustive search of all processes and resources to determine deadlocks by locating circular waits in the process-resource graph. For a system with a large number of processes and resources, the execution time of an exhaustive search would be too long. Havender [32] proposed a method that requires resources to be requested and released in some specific order. Habermann [30] requires a prior knowledge of the maximum number of resources that each process will use. A central processor uses this knowledge to determine if any subsequent request and allocation of resources is deadlock free. Holt [34], [35], [36] provides a more extensive work on deadlocks in computing systems. He uses large matrices or their equivalent graph representations to check for deadlocks. Hutchison et al. [39] improved on Holt's work by using a recursive algorithm to remove unnecessary nodes from the precedence graph or adjacency matrix. The technique speeds up execution time by reducing the graph.

However, most of these early solutions are mainly for single location systems, where all processes and resources are available locally, thereby making the resolution of the problem much simpler. These solutions become impractical in distributed computer systems.

Coffman [16] lists the following conditions as necessary for the occurrence of a deadlock:

1. Mutual Exclusion - a resource can only be acquired by one process at a time.
2. Non-preemptive Scheduling - a resource can only be released by the process which acquired it.
3. Partial Allocation - a process can acquire its resources piecemeal.

4. Circular Waiting - the previous conditions allow concurrent processes to acquire part of their resources and enter a state in which they wait indefinitely to acquire each others' resources.

Deadlock can be prevented by ensuring that one or more of these conditions never hold. Although in many practical situations some of these conditions are quite necessary. As an example, in a database environment, it is very desirable for an exclusive access to a resource for update purposes to maintain consistency in the database. The subject of database consistency will not be pursued much further in this thesis. Consistency control in database using two-phase locking technique is discussed by Eswaran et al. [22].

Deadlock became a very serious concern with the coming of multiprogramming operating systems, that is, operating systems which allow several processes to run concurrently. In early systems of this kind, requests for mechanical devices, such as tape drives, sometimes resulted in deadlocks which were treated as special cases or errors. Typically, such deadlocks were either prevented by requiring the user to specify the maximum quantity of such resources when submitting his job or eliminated when they occurred by eliminating the job. Deadlocks on files were generally very infrequent and were typically handled in the latter fashion, that is, eliminating the job. But as more powerful multiprogramming operating systems were designed, the problem of deadlock became a major concern for many of these systems, and resulted in more research on the subject [20], [23] and [63].

1.1 Deadlock Elimination Techniques

There are three basic techniques for resolving deadlock problem: (1) Prevention, (2) Detection, and (3) Avoidance.

1.1.1 Deadlock Prevention

This is the process of designing a deadlock free system. A necessary condition for deadlock is the existence of a circular chain of processes, each of which holds exclusive and non-preemptable control of some resources and each of which is requesting for the resource held by the next process in the chain. This situation can be prevented by:

1. Having each process declare all the resources it will need at once [16], [23], [32]. All requests must be granted before the process can start. This technique is used on OS/360 [54] for device allocation. A slight variation of this technique is having the process specify all the resources needed in advance except that the resource scheduler starts the process even when all the resources are not immediately available [19], [30]. This approach has the following disadvantages:

    a. Some processes may not know what resources they will need until they are at the point of using them.

    b. Resources may be held for an extended period during which they are not needed. They could be released to some other processes.

    c. Delaying of process initiation. Process initiation begins only after the process has acquired all the resources it will need during its execution.

    d. It is wasteful for the system to commit a resource to a process when there is only a small likelihood that the process will use that resource.

    e. Even with the slight variation, the process must still know in advance its maximum resource needs.

2. Preemption. Whenever a process's request for a new resource cannot be granted immediately, other resources held by the process are preempted and the process rolled back. Usually if this approach is used more processes than are necessary will be preempted. A simulation model is developed for this technique in this dissertation.

    Alternatively a process can be forced to release resources temporarily in favor of other processes [31]. This approach may not be feasible if the resource was being updated. However, in present computers preemption is used to multiplex central processors and storage between concurrent processes.

3. Resource Ordering [31], [32]. This is a more sophisticated method of deadlock prevention. There are two basic types of ordering that can be employed:

   a. Sequential ordering.

      Resource requests are ordered sequentially to prevent circular waiting. The "banker's algorithm" [31] uses this approach by finding a sequence in which concurrent processes can be completed one at a time if necessary. The algorithm, however, requires each process to indicate its maximum resource needs in advance. It assumes that each process may request all its resources at once and keep them throughout its life time. The main problem with the bankers algorithm is that it is too expensive to implement.

   b. Hierarchal ordering.

      Resources are grouped into ordered classes $R_1, \ldots, R_k$. If a process holds a resource of class R_j then it may request for another resource of class R_i only if $i > j$. This ordering makes circular chain impossible.

      The lack of flexibility in request sequences can lead to a process requesting and holding a resource unnecessarily early. The process must still know in advance the resources it will need and the class it belongs to. The latter means the user must be well-educated on the system. Also, a mechanism that checks and enforces the ordering must be designed into the system. This means more system overhead.

      Generally, these prevention techniques are not acceptable in a distributed system. It is not feasible to design a deadlock free distributed system, since it is impossible to predict the order requests for resources will be made. If preemption technique is used more processes than are necessary will be preempted. The results of the simulation in this thesis support this fact. Also in a distributed database, the next resource needed by a process may depend on the result of the current action. So declaring all resources needed in advance is not possible.

1.1.2 Deadlock Detection

This technique involves a periodic use of a detection algorithm which inspects the current resource allocations and oustanding requests, to produce an indication of whether a deadlock currently exists, and if it does exist, what processes and resources are involved. The approach is also equipped with the ability to back-up processes in order to break the deadlock. In order to break the deadlock some processes must be preempted. Therefore, detection does not only involve the overhead of running the detection algorithm, but also the loss of processing time spent by the preempted resources. It may result in loss of valuable data and inconsistency in the state of the data. If a more sophisticated back-up technique is used, it will result in high overhead for the system in saving the states of the processes before preemption. The method takes no action until a deadlock actually occurs. Thus a process may be blocked for a long time before it is noticed, unless there exists a mechanism in the system which automatically starts the detection algorithm any time a deadlock is suspected.

1.1.3 Deadlock Avoidance

An avoidance algorithm projects detection into the future in order to keep the system from committing itself to an allocation which will eventually lead to a deadlock. The algorithm must be provided with information about future data requirements for each process. This implies resource requirements forecasting.

Habermann [30] proposed what he called a "maximum claims strategy" to control the future resource requirements of each process. Deadlock avoidance is achieved by testing each possible allocation and granting those which lead to "safe" states.

A problem arises with avoidance schemes when the system is heavily loaded. In this case there will be very few available resources, so new requests will be denied, thus blocking the processes that made the new requests. These processes may be blocked for a long time, thereby tying up those resources they had already acquired. Also, the technique is time-consuming because the algorithm is run every time a request for a resource is made.

1.1.4 Mixed Solution

Howard [38] maintains that prevention, detection, or avoidance alone is inappropriate for the solution of the deadlock problem. A method based on the concept of hierarchical operating system is suggested. The solution combines the three basic techniques while allowing the selection of the optimal one for each class of resources in a system.

1.2 Database and Deadlock Problem

Efficient implementation of a database depends on the amount of concurrency it can support. Sharing of a database creates many problems such as file allocation [13], and deadlock. The high concurrency involved in a database system makes deadlock problem more serious in such systems. The concern here is not only avoiding or detecting deadlocks but also doing so such that the consistency of the database is maintained. Most of the deadlock prevention and avoidance schemes in operating systems mentioned earlier become less feasible in a database system. This is because of the dependency of the next action on the previous data item retrieved. It appears a detection approach with a good rollback and recovery technique is best for a database environment. The rollback and recovery problem, which is of great importance from a data viewpoint in maintaining the consistency of the database, is addressed by Chandy and Ramamoorthy [12], Chandy et al. [11], Maryanski and Fisher [60] and Russell [66].

1.2.1 Centralized Database

Many studies on deadlock protection schemes for centralized databases have been reported in the literature. Among them are works of Berstein and Shoshani [3], Chamberlin et al. [9], Collmeyer [17], Eswaran et al. [22], Frailey [24], King and Collmeyer [47], Schlageter [69], Shemer and Collmeyer [71], and Stearns [72].

Berstein and Shoshani [3] models a database using graphs, with nodes representing a collection of information. They present algorithms to overcome the conflicts and avoid deadlock as concurrent access at the same node takes place. Lomet [51] presents a scheme in which processes are required to pre-declare their anticipated resource requirements. The algorithm is tailored to the needs of a database system, unlike the approaches presented by Havender [32] and Holt [36]. A series of graph representations for database interactions are developed. From these, necessary and sufficient conditions for the existence of a deadlock are derived, and a deadlock avoidance scheme devised. A refinement of this scheme is given by Lomet [52], in which the problem of indefinite delay, that is, the possibility that a process will not run to completion, is eliminated. This approach partitions the resource system into subsystems, each of which can be scheduled independently. Indefinite delay is avoided by the construction of subsystems that guarantee the completion of a process or the granting of a resource request. Although the possibility of indefinite delay is not completely eliminated by this latter algorithm, it is considerably reduced.

Chamberlin's technique [9] is a very shrewd modification and combination of the following steps: (a) try to preclaim needed resources; (b) if preclaiming resources leads to a deadlock, preempt resources; and (c) impose a presequencing mechanism for processes by time stamping to avoid deadlock due to indefinite delay.

King and Collmeyer [47] describes the "LOCK-UNLOCK" mechanism of the Codasyl approach to database management [15], which enables incremental allocation of data resources to processes. The status of all accesses to the database is maintained in an access state graph. The scheme models each of the operations "LOCK", "ALLOCATE" and "DEALLOCATE" and derives a necessary and sufficient condition for the existence of a deadlock in terms of the effect of the "ALLOCATE" function. A detection scheme is derived using this, and a recovery technique in the event of a deadlock is suggested.

Schlageter [69] discusses one-level and two-level lockout mechanisms for access synchronization. In the one-level lockout scheme, shared access to the database is allowed at any time, but exclusive accesses are required to lock the data resources before using them. The presence of a cycle in the state graph is a necessary and sufficient condition for a deadlock. An algorithm is presented for detecting deadlock by traversing the graph from a blocked process node in an attempt to return to that blocked node. In the two-level lockout scheme, shared accesses are split into two classes: those which are insensitive to concurrent updates and those which prevent exclusive access users from concurrently accessing the data. The deadlock detection scheme proposed also starts at the blocked process node and tests if a path returns to the process node. But this scheme is no longer simple since a resource may be held by several processes simultaneously and each of these may be regarded as blocking any waiting processes.

1.2.2 Distributed Database

Fry and Sibley [25] pointed out that distributed database management systems share numerous problems with both database management systems and computer networks as well as introducing several fresh dilemmas, such as locating and updating redundant data. Potential major problems facing designers in this area have been identified by Maryanski [59]. In addition to these concerns is the deadlock problem. Relatively few papers have been published on deadlock resolution in distributed database. Most of the techniques published have some drawbacks. Counter examples to some of the proposed techniques have been reported. An overview of deadlock problem and a summary of deadlock handling techniques in distributed systems can be found in [42].

Chu and Ohlmacher [14] propose two approaches for handling deadlock in a distributed database. The first approach requires the allocation of all needed resources before process initiation. The second approach is based on the concept of process sets, which is a collection of processes with access to common data resources. A process is allowed to proceed only if all data resources required by the process and the members of its process sets are available.

Maryanski [58] gives a prevention algorithm which requires each process to communicate its shared data resource list to all other processes before it can proceed. The resource list is conceptually similar to the process set in [14]. The shared data resource list is determined by using a process profile which contains data resources that can be updated by the process. However, communication and computation of process sets [14] or shared data resource lists [58] which are performed continually as processes enter or leave the system require substantial system overhead.

A centralized approach for deadlock detection in distributed databases is also suggested by Gray [29]. In this approach there is a centralized deadlock detector which is responsible for constructing a global graph. This graph is built from information received from all the participating sites in the network. Rypka and Lucido [68] give a model of resource sharing using access modes. It allows access relationships which can increase concurrency of processes and yet preserve the consistency of data.

They present detection, avoidance and prevention techniques that permit increased multiprogramming.

Mahmoud and Riordan [55], [56] report both centralized and distributed approaches to deadlock detection. The centralized approach detects deadlock by creating an overall global picture of the network status by using information received from other sites in the network. In the distributed approach each site sends identical messages to every other site, and receives different messages from each one, so that a deadlock may be detected at any particular site. Chandra, Howe and Karp [10] propose a scheme that requires maintaining a resource table at each site, containing information on the activities of all processes and resources in the network. They claim the existence of well-known algorithms to detect deadlocks in a single-site facility using the tables, and that the same algorithms can be used in a distributed environment provided the resource tables are expanded to include useful information from remote sites. However, the schemes proposed in [10] and [55], [56] have been shown to be incorrect by Goldman [28], as deadlock may go undetected.

Menasce and Muntz [61] propose hierarchically organized and distributed protocols for deadlock detection in distributed databases. Gligor and Shattuck [27] give a counter example and possible remedies to their scheme. The impracticality of the algorithm is also shown in [27], as condensations of "transaction-wait-for" graphs make it difficult to perform graph updates.

The detection algorithm proposed by Isloor and Marsland [40], [41] and [57] has, as the main features: (a) the significant reduction of communication requirements between sites which usually follow the invocation of a detection mechanism, and (b) allowing a process to have as many outstanding requests as possible. The algorithm maintains a complete process-resource graph for the whole network at each site. Thus, all information needed to detect a deadlock is available at each site at all times, thereby making early detection possible. The algorithm uses the idea of reachable set [36], which is the set of all nodes traversed by a directed path by a given node, to detect an occurrence of a deadlock. A process will be deadlocked if and only if the process belongs to its reachable set. Reachable sets for all nodes in the network are maintained, along with the system graph, as resources are allocated, freed and waited upon at each site. The frequency of graph maintenance characterized by this algorithm will lead to a high communication overhead. Also in large systems communication delays will result in inconsistency in the state of the tables. A deadlock can be detected and removed in one site but not in the others.

Other contributions to the deadlock problem in distributed database are due to Goldman [28], Le Lann [50] and Peebles and Manning [64]. Goldman's distributed algorithm is discussed in Chapter II.

1.3 Deadlock in Packet Switch Networks

Deadlock also manifests itself in congestion control in packet switch networks. This type of deadlock is called "store-and-forward" deadlock and is reported by Gerla [26], Kamoun [46], Schwartz [70] and Vinton [73]. If a routing algorithm used in a packet switch network causes traffic flowing in opposite directions to flow through two adjacent packet switches, and each switch fills to capacity with packets destined for the other, the two switches become deadlocked. Some of the earliest investigations in this area were reported by Kahn and Crowther [45] in their work on the ARPANET. Solutions to this kind of deadlock are reported in [26], [44] and [65].

1.4 Deadlock--A Game between Operating System and Processes

Devillers [18] defines deadlock avoidance problem as determining safe situations which may be realized without endangering the smooth running of the system from some information about the processes, resources and the Operating System. A global approach to the deadlock phenomenon is taken, and the evolution of the system is interpreted as a game between the Operating System and processes. He proposes a method in which a "state" is defined safe if and only if a strategy exists for the resource manager which ensures its success whatever operation the processes in that state choose. A state will lose if an operation exists for the processes such that the resource manager will lose the game whatever strategy it chooses. This approach throws new light on the deadlock problem by providing a way to construct the set of unsafe states and, hence, providing a basis for a systematic study of the properties of the safe states.

1.5 Definitions

Distributed Computer System: A distributed Computer System is a network of loosely coupled processor and resource sites. A processor site consists of a central processing unit, private main memory, peripheral devices and communication channels to other sites in the network. A resource site may be a physical object such as input/output device or abstract object such as a database system.

Resource Manager: A resource manager is a software module that schedules access to resources by competing processes. Each process requests for resources through a resource manager.

Process: A process is the passage of control through an ordered set of instructions that performs some computation.

Resource: A resource is any passive object that can be requested, acquired and released by user processes.

Controller Site: A controller site in a distributed computer network is a site in the network dedicated to controlling accesses to all resources in the network. Requests for resources by all processes in the network are sent to the controller site.

Process-Resource Graph: A process-resource graph is a bipartite directed graph whose disjoint set of nodes are called process nodes and resource nodes. An edge directed from a resource node to a process node means that the resource identified by the resource node is being held by the process identified by the process node. Conversely, an edge from a process node to a resource node means that the process identified by the process node is requesting access to the resource identified by the resource node.

1.6 Statement of the Thesis

The problem solved by this thesis is the design of a good online deadlock detection algorithm for a distributed computer system. Also simulation models are developed to measure the performance of the new algorithm, Goldman's detection algorithm and preemption technique of deadlock prevention on a distributed ring network. A good deadlock detection algorithm should minimize the amount of messages passed between the different sites in the network. It should be simple to implement. The storage requirement needed to run the algorithm should be minimal.

A mathematical model is developed for the new algorithm.

The results of the simulation and mathematical models for different numbers of sites, from 3 to 12 sites, show that the new algorithm improves process throughput when compared to the preemption technique and Goldman's algorithm. Also, the new algorithm gives lower intersite messages than Goldman's algorithm.

II. GOLDMAN'S DISTRIBUTED ALGORITHM

The deadlock detection scheme proposed by Goldman [28] requires the construction and expansion of an "ordered blocked process list" (OBPL) every time deadlock detection is initiated. An OBPL is a list of processes, each of which, with the exception of the last one in the list, is waiting for access to a resource that is held by the next process in the list. The algorithm allows a process to have only one outstanding request at a time. It assumes the existence of a resource manager at each site. The resource manager handles resource allocation and deadlock detection. It maintains local state tables containing information about resources located locally and processes running at its site.

To detect a possible deadlock the resource manager creates an OBPL and inserts the network unique name of its blocked process as the first entry in the OBPL. The requested resource name is inserted in the identification portion of the OBPL. The resource manager then starts to expand the OBPL, until there is not enough information available for further expansion. The OBPL is then sent to other sites for further expansion. Multiple copies of OBPL are made whenever a process waits for or accesses a shared resource, thus introducing inconsistency problem in the different copies. Breaking of deadlock within some OBPLs may not be reflected in the other copies of OBPL soon enough to prevent false deadlocks. This also increases communication overhead in the network. The algorithm is given below in the author's own words. PX and RX are assumed to be names of variables whose contents represent processes and resources, respectively. PMM referred to in the algorithm means process management module or resource manager.

1. Set RX to the value contained in the resource identification portion of the OBPL. If RX represents a resource which is local to the node expanding the OBPL, then go to step 2, otherwise go to step 8.

2. Verify that the last process added to the OBPL is still waiting for RX. If it isn't then discard the OBPL and halt, otherwise go to step 3.

3. Let PX be the process controlling RX. (If there are J shared readers of RX, then repeat this step once for each reader.) If PX already has a process entry in the OBPL, then there is a deadlock and the PMM must take the appropriate action. If PX is not in the OBPL then go to step 4.

4. If PX represents a process which is local to the node expanding the OBPL, then go to step 5, otherwise go to step 7.

5. If PX is active, there is no deadlock, so discard the OBPL and halt. Otherwise, go to step 6.

6. Append PX as a process entry in the OBPL and go to step 10.

7. Append PX as a process entry in the OBPL. Place RX into the resource identification portion of the OBPL and send the OBPL to the PMM in the node in which PX resides. Halt.

8. Verify that the last process added to the OBPL still has access to RX. If it doesn't, discard the OBPL and halt. Otherwise go to step 9.

9. If the last process added to the OBPL is active, there is no deadlock, so discard the OBPL and halt. Otherwise go to step 10.

10. Get the name of the resource for which the last process added to the OBPL is waiting and call it RX. If RX represents a resource which is local to the node expanding the OBPL, go to step 3, otherwise go to step 11.

11. Place RX into the resource identification portion of the OBPL and send the OBPL to the PMM in the node in which RX resides. Halt.

The explanation and verification of the algorithm are given in the reference [28]. The resource manager starts expanding a newly created OBPL in step 10. When a resource manager receives an OBPL from another site, it starts expanding it in step 1. The proposal does not address rollback problem in detail. However, in the simulation model developed for the algorithm in this dissertation, when a deadlock is detected the process whose request caused the deadlock is rolled back to the beginning. All its resources are released. It is delayed a random number of simulated time and then restarted.

2.1 Example

Consider a three-site network, and assume the following state in the network:

Processes P1 and P2; P3, P4, P5, P6 and P7; and P8, P9, P10 and P11 run at sites S1, S2 and S3, respectively. Resources R1 and R2; R3, R4 and R5; and R6, R7, R8 and R9 are located at sites S1, S2 and S3, respectively. Process-resource interactions are as shown in Figure 1. All requests and accesses are exclusive. An arrow from a resource node to a process node means that the process identified by the node had gained access to the resource identified by the resource node. An arrow from a process node to a resource node means that the process is waiting for the resource. Assume that the new request is for R1 by P10. The states of the tables maintained by the resource manager at each site before the request is made are shown in Table 1.

S3 updates its process table and sends the request out, since R1 is not a local resource. S1 receives the request, and updates the waiting list for resource R1, since R1 is not free for immediate allocation.

Now S3 decides to check for deadlock. It creates the OBPL,

R1	P10

and starts expanding. THe expansion starts at step 10 of

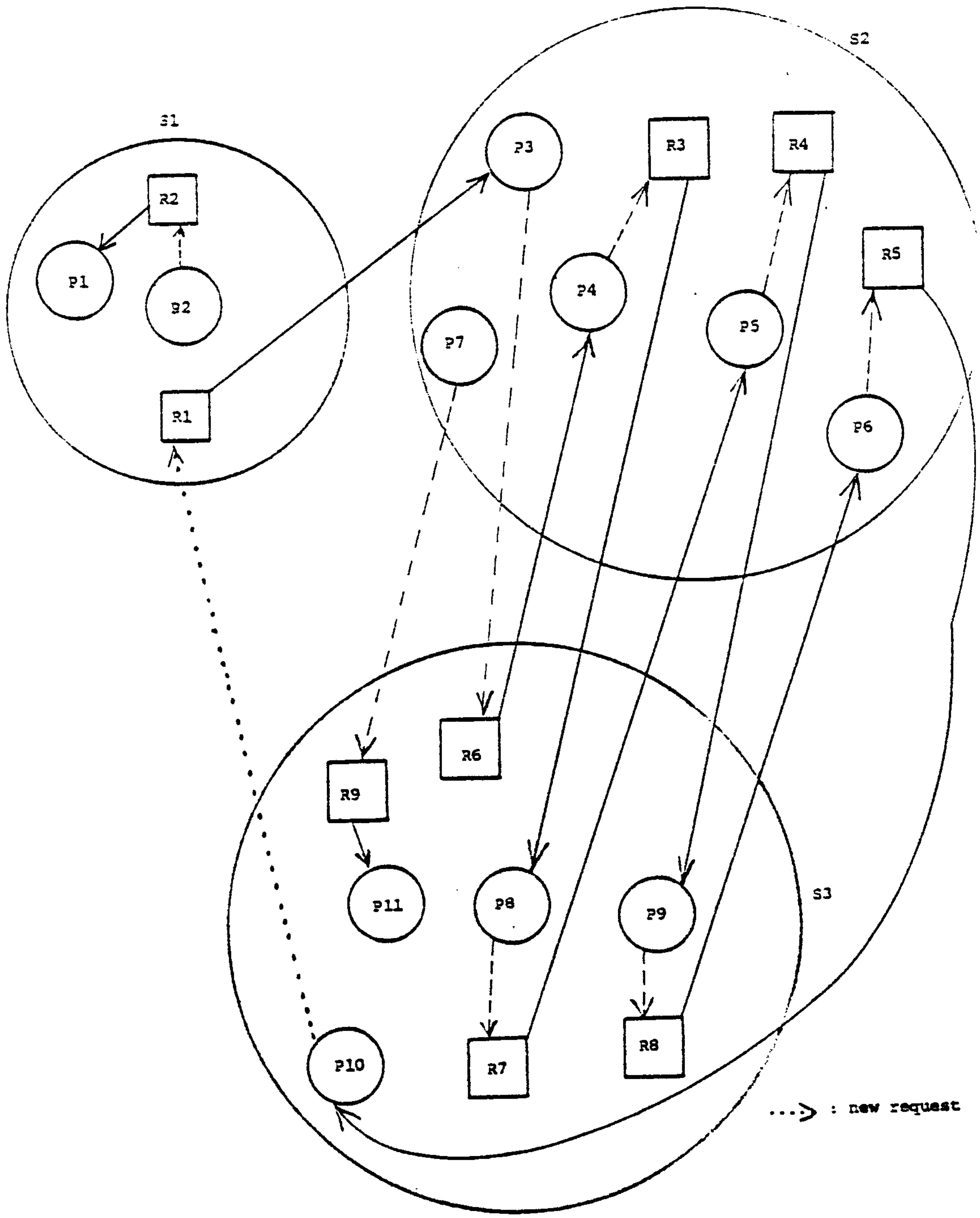


FIGURE 1. Process-Resource Graph for a distributed network with three sites S1, S2, S3, with a set of concurrent processes {P1, P2, ..., P11} and a set of resources {R1, ..., R9}.

TABLE 1. Process and Resource Tables for Goldman's Distributed Algorithm Example

SITE S1

a) Process Table

PROCESS NAME	RESOURCES HELD	NEW REQUEST
P1	R2 @ S1	--
P2	--	R2 @ S1

b) Resource Table

RESOURCE NAME	PROCESSES ACCESSING	PROCESSES WAITING
R1	P3 @ S2	--
R2	P1 @ S1	P2 @ S1

SITE 2

a) Process Table

PROCESS NAME	RESOURCES HELD	NEW REQUEST
P3	R1 @ S1	R6 @ S3
P4	R6 @ S3	R3 @ S2
P5	R7 @ S3	R4 @ S2
P6	R8 @ S3	R5 @ S2
P7	--	R9 @ S3

b) Resource Table

RESOURCE NAME	PROCESSES ACCESSING	PROCESSES WAITING
R3	P8 @ S3	P4 @ S2
R4	P9 @ S3	P5 @ S2
R5	P10 @ S3	P6 @ S2

SITE 3

a) Process Table

PROCESS NAME	RESOURCES HELD	NEW REQUEST
P8	R3 @ S2	R7 @ S3
P9	R4 @ S2	R8 @ S3
P10	R5 @ S2	--
P11	R9 @ S3	--

b) Resource Table

RESOURCE NAME	PROCESSES ACCESSING	PROCESSES WAITING
R6	P4 @ S2	P3 @ S2
R7	P5 @ S2	P8 @ S3
R8	P6 @ S2	P9 @ S3
R9	P11 @ S3	P7 @ S2

Notations: Ri @ Sj means resource Ri located at site Sj.

Pi @ Sj means process Pi located at site Sj.

the algorithm. R1 is not local to S3, so go to step 11. In step 11 the OBPL is sent out.

S1 receives the OBPL, and starts at step 1. R1 is local to S1, so go to step 2. Assume P10 is still waiting for R1, go to step 3. P3 is controlling R1. P3 has no entry in the OBPL, so go to step 4. P3 is not local to S1, go to step 7. Append P3 to the OBPL.

R1	P10	P3

Send the OBPL to S2.

S2 starts at step 1. R1 is not local to S2, go to step 8. P3 still has access to R1, so go to step 9. P3 is not active go to step 10. P3 is waiting for R6. R6 is not local to S2, go to step 11. Place R6 in the resource identification, and send the OBPL to S3.

R6	P10	P3

S3: R6 is local, go to step 2. P3 is still waiting for R6, go to step 3. P4 is controlling R6. P4 is not in the OBPL go to step 4. P4 is not local to S3, go to step 7. Append P4 to the OBPL.

R6	P10	P3	P4

Send the OBPL to S2.

S2: R6 is not local to S2, go to step 8. P4 still has access to R6, go to step 9. P4 is not active, go to step 10. P4 is waiting for R3. R3 is local, go to step 3. P8 is controlling R3. P8 has no entry in the OBPL, go to step 4. P8 is not local, go to step 7. Append P8 to the OBPL. Place R3 in resource identification.

R3	P10	P3	P4	P8

Send OBPL to S3.

S3: R3 is not local to S3, go to step 8. P8 still has access to R3, go to step 9. P8 is not active, go to step 10. P8 is waiting for R7. R7 is local, go to step 3. P5 is controlling R7. P5 has no entry in the OBPL go to step 4. P5 is not local to S3, go to step 7. Append P5 to the OBPL. Place R7 in resource identification.

R7	P10	P3	P4	P8	P5

Send the OBPL to S2.

S2: R7 is not local to S2, go to step 8. P5 still has access to R7, go to step 9. P5 is not active, go to step 10. P5 is waiting for R4. R4 is local to S2, go to step 3. P9 is controlling R4. P9 has no entry in the OBPL, go to step 4. P9 is not local to S2, go to step 7. Append P9 to the OBPL, and place R4 in the resource identification.

R4	P10	P3	P4	P8	P5	P9

Send the OBPL to S3.

S3: R4 is not local to S3, go to step 8. P9 still has access to R4, go to step 9. P9 is not active, got to step 10. P9 is waiting for R8. R8 is local to S3, go to step 3. P6 is controlling R8. P6 has no entry in the OBPL, go to step 4. P6 is not local to S3, go to step 7. Append P6 to the OBPL, and place R8 in the resource identification.

R8	P10	P3	P4	P8	P5	P9	P6

Send the OBPL to S2.

S2: R8 is not local to S2, go to step 8. P6 still has access to R8, go to step 9. P6 is not active, go to step 10. P6 is waiting for R5. R5 is local to S2, go to step 3. P10 is controlling R5. P10 already has entry to the OBPL, therefore a deadlock exists and is detected at step 3 by site S2.

A simulation model is developed for this algorithm on a unidirectional ring network. Its performance is compared with that of the new algorithm proposed in this thesis.

III. THE HORIZONTAL AND VERTICAL ALGORITHM

3.1 Basic Assumptions

The Horizontal and Vertical (H&V) algorithm assumes the following:

1. The existence of a resource manager at each site to handle resource allocation and deadlock detection,
2. a process may have only one outstanding resource request at a time, which means that a process can only wait for one resource at any instant,
3. a resource may be any uniquely identifiable portion of a data object, whole data object or collection of data objects which are requested as an entity and released as an entity by all processes,
4. a process is any identifiable user program that runs on a computer,
5. a process can access as many resources as desired, but they are seized one at a time,
6. during a life cycle of a process it is allowed to seize a resource, release it and later on request the same resource again, and
7. a process can request for exclusive (read/write) or shared (read only) access to a resource. Since a process is not allowed to request for one type of access and while still holding the resource, request for another type of access on the same resource, a process must make the type of access known at the time the request is made.

3.2 Cycles in Process-Resource Graph

Consider a computer system with a set of processes P1, P2, ..., Pn, running concurrently, and holding or waiting for a set of resources R1, R2, ..., Rm. The state of the system

can be represented graphically by a process-resource graph, with nodes corresponding to each process, Pi, 1<=i<=n, and each resource Rj, 1<=j<=m, and with edges representing process interactions in the system. Formally, the process-resource graph is a bipartite directed graph, G = (V,E), where
V = {P1,P2,...,Pn} U {R1,R2,...,Rm} and E are edges either from process nodes to resource nodes or from resource nodes to process nodes.

NOTATIONS: The following notations and convention will be used throughout this thesis:

1. Circles will be used to represent process nodes.
2. Squares will be used to represent resource nodes.
3. A solid arrow from a resource node to a process node means that the resource corresponding to the resource node is being accessed by the process corresponding to the process node.
4. A dashed arrow from a process node to a resource node means that the process corresponding to the process node is waiting for the resource corresponding to the resource node.

As stated in Chapter I, one of the necessary conditions for a deadlock is when two or more processes acquire part of their resources and then wait in a circular chain for each other's resources. In terms of the process-resource graph, this means that a deadlock exists if it is possible to reach a starting node by traversing through the system graph. Therefore, a resource deadlock is a cycle in a process-resource graph.

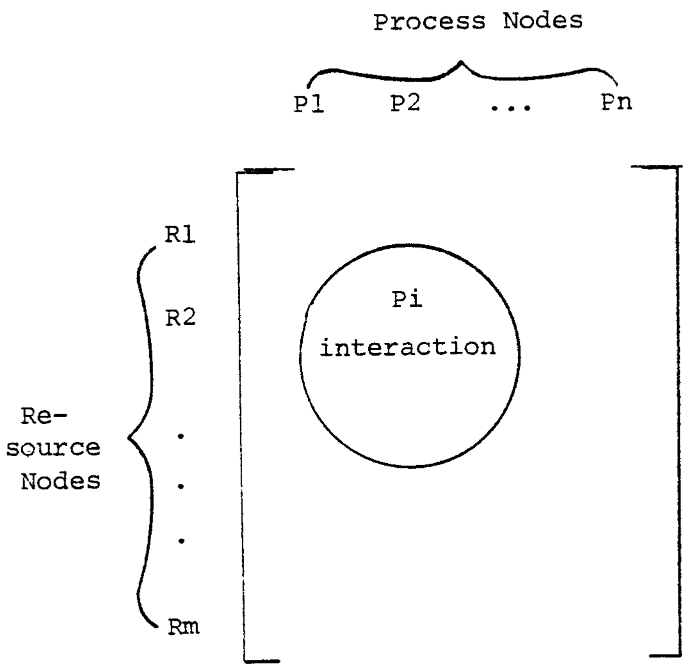
Example 3.1: Figure 1 shows a process-resource graph for a computer system with three sites, S1, S2, and S3. {P1,...,P11} and {R1,...,R9} form the process nodes and resource nodes, respectively. {P1,P2}, {P3,...,P7}, {P8,...,P11} run at sites S1, S2 and S3, respectively, while {R1,R2}, {R3, R4, R5} and {R6,R7,R8,R9} are located at sites S1, S2 and S3, respectively. At S1, P1 is holding R2 while P2 is waiting for R2. P3 at site S2 is holding R1 while P10 at S3 is waiting for R1. At site S2, P8, P9 and P10 are holding R3, R4, and $5, respectively, while P4, P5 and P6 are waiting for R3, R4 and R5, respectively. At S3, P4, P5, P6 and P11 are holding R6, R7, R8 and R9, respectively, P3, P8, P9 and P7 are waiting for R6, R7, R8 and R9, respectively. All accesses and requests are assumed to be exclusive. There is a deadlock in the system because the process-resource graph contains a cycle. The cycle is made up of processes P10, P3, P4, P8, P5, P9 and P6.

3.3 Formal Model of the Horizontal and Vertical Deadlock Detection Scheme

This section introduces the necessary notation and formalism upon which the Horizontal and Vertical algorithm is based. The algorithm is modelled from a process-resource graph. There are two basic structures for representing graphs: adjacency matrix and adjacency list. The method we use to represent the process-resource graph resembles the adjacency matrix.

Let G = (V,E) be a process-resource graph, where V = {P1,...,Pn} U {R1,...,Rm}. This graph will be represented by what we call a process-resource matrix. The columns of the matrix are identified by the process nodes while the rows are identified by the resource nodes. The matrix entries represent process interactions. Table 2 shows the form of the matrix.

TABLE 2. Process-Resource Matrix



Formally, the process-resource matrix is maintained in the form of a table called "Process-Resource Table." Each row of the process-resource table is identified by a resource name, while each column is identified by a process name. The table entries indicate the state of the processes with respect to the resources. A process can be in two different states, namely, active and blocked. A process is blocked if its execution cannot proceed because it is waiting for a resource which is being held by another process, and a process is active otherwise. Thus, the column of the table forms a pattern of requests by the process identified by the column, while the row forms a queue of requests for the resource identified by the row.

An entry in the table is called a RANK of a process identified by the column, for the resource identified by the row of the table. This indicates a process's relative position in the waiting queue of a resource. A rank has two components: (1) the process's relative position, and (2) the type of access required. The symbol ρ is used to represent a rank. Thus ρ [Pi,Rj] = (0,e(exclusive)) means Pi has gained exclusive access to resource Rj and ρ [Pi,Rj] = (0,s(shared)) means Pi has gained shared access to Rj. [Pi,Rj] = (j,e) and [Pi,Rj] = (j,s) means Pi is j^{th} in line for exclusive and shared access, respectively, to the resource. A null entry (blank) means that there is no request by the process identified by the column for the resource identified by the row.

The Process-Resource table is built dynamically as resources are seized and released. To facilitate the maintenance of this table, the resource manager maintains two other tables--the resource table containing the status and name of all local resources, and a process table containing the names and location information of processes using its local resources.

Example 3.2: Figure 2 shows a process-resource graph for a system with six processes and six resources. All requests and accesses are for exclusive use. Table 3 gives the process-resource table for this graph.

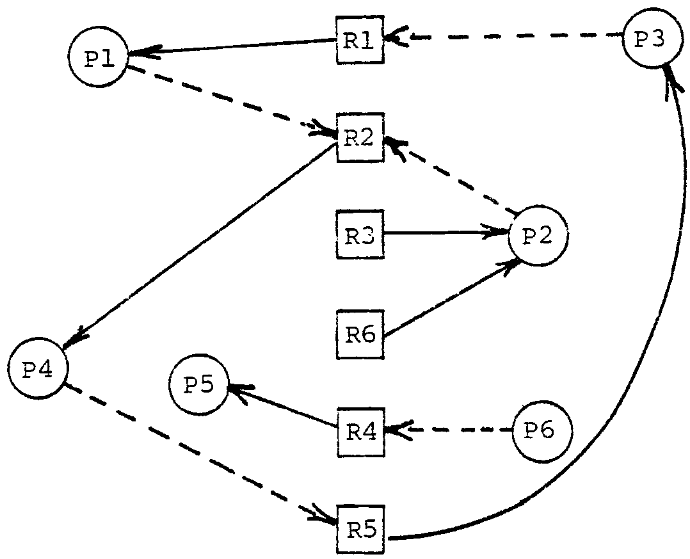


FIGURE 2. Process-Resource Graph.

TABLE 3. Process-Resource Table

	P1	P2	P3	P4	P5	P6
R1	(0,e)		(1,e)			
R2	(1,e)	(2,e)		(0,e)		
R3		(0,e)				
R4					(0,e)	(1,e)
R5			(0,e)	(1,e)		
R6		(0,e)				

Ranking

Each resource request is ranked to prevent process starvation. Without the ranking, it is quite possible for a process requesting for exclusive access to a resource to wait indefinitely for the resource as long as requests for shared access keep coming in.

The ranking of each request by the resource manager is based on the Readers/Writer problem concept [37]. A resource is FREE for immediate allocation if

1. no process is using it.
2. request is for shared access, and the resource is being held under shared access, and no process is waiting for exclusive access.

   a. If request is for exclusive access then rank of new request = highest rank + 1.
   b. If request is for shared access then

      (1) If the resource is held exclusively and there is a waiting process for exclusive access, and a waiting process for shared access then rank of new request = rank of waiting shared request.
      (2) If resource is held exclusively and there is a waiting process for shared access, and no waiting process for exclusive access, then rank of new request = rank of waiting request.
      (3) If the resource is held under shared access and there is a waiting process for exclusive access then the rank of new request = rank of waiting shared request, if any, or highest rank + 1 if no waiting shared request.
      (4) All other cases, rank of new request = highest rank + 1.

Process Pi releases resource Rj

(1) change rank of Pi for Rj to null.

(2) if Pi has no outstanding request and is not holding any other resource locally, then remove Pi from Process-Resource table and Process table.

(3) If Rj is not being held by any other process and no process is waiting for it, then remove Rj from Process-Resource table and update the status of Rj in Resource table.

(4) If Rj is free after the release and there are waiting processes then

```
    for i := 1 to n do
     if PRTABLE [Rj,Pi] · Rank > 0 then
        PRTABLE [Rj,Pi] · Rank := PRTABLE [Rj,Pi] · Rank - 1;
```

Allocate Rj to processes with rank of zero.

Deadlock Detection Approach

Deadlock detection involves building and maintaining the process-resource table and searching for the existence of a cycle, which corresponds to a cycle in the process-resource graph.

To find a cycle using the process-resource table, we need only repeatedly perform a horizontal search followed by a vertical search, until returning to the starting entry. Every time a request is made the resource manager enters the rank of the request in the process-resource table. If the resource is free, the request is immediately granted and a rank of zero is entered. If the resource is not free, a rank greater than zero for the request is entered.

To check for deadlock we start from the current request entry. A horizontal search finds zero rank entries while a vertical search finds a rank greater than zero entry. Since a process is allowed only one outstanding request at a time, there can only be one greater than zero entry in each column. And since shared access is allowed on the resource, there can be more than one zero entry for each row.

Example 3.3 demonstrates how to find a cycle using the process-resource table of Table 3.

Example 3.3

TABLE 4. Process-Resource Table Showing Search Paths

	P1	P2	P3	P4	P5	P6
R1	(0,e)		$\boxed{(1,e)}$			
R2	(1,e)			(0,e)		
R3		(0,e)				
R4					(0,e)	(1,e)
R5			(0,e)	(1,e)		
R6		(0,e)				

starting entry

Assume that we want to check whether P3's request for R1 causes a cycle in the process-resource graph. Using the process-resource table the search starts at location [R1,P3]. A horizontal search finds a zero at location [R1,P1]. A vertical search finds a one at location [R2,P1]. A horizontal search finds a zero at location [R2,P4]. A vertical search finds a one at location [R5,P4]. A horizontal search finds a zero at location [R5,P3] and, finally, a vertical search returns the search to the starting entry. Thus, the search path is [R1,P3], [R1,P1], [R2,P4], [R5,P4], [R5,P3].

In Chapter IV, we shall prove that the algorithm to do the search resembles the algorithm for performing a breadth-first search on the process-resource graph. Table 4 shows the process-resource table with the search path.

The nuclei of the horizontal and vertical algorithm are the horizontal and vertical search. In the next section formal descriptions will be given for them.

3.4 Semantics of the Horizontal and Vertical Routines

3.4.1 Horizontal

The horizontal algorithm takes as its input a resource, R, Process-Resource table and the Process table. It performs a horizontal search on the Process-Resource table along the row identified by R and returns all processes accessing R, that is, all processes Pi, such that ρ [Pi,R] = 0. The routine is given below.

```
Procedure Horizontal (R,h,P);

     // P   = list of processes accessing R                  //
     // PRT = Process-Resource table                         //
     // PT  = Process table                                  //
     // h   = number of processes accessing R                //
     // n   = current number of processes in Process Table  //
        Begin
          h := 0;
          for  i := 1 to  n  do
           if  PRT  [R,i] = 0 then           // if rank = 0 //
               begin
                 h := h + 1;
                 P [h] := PT [i]
               end;
        end:          // Horizontal //
```

3.4.2 Vertical

The Vertical algorithm takes as its input a process, P, Process-Resource table and Resource table. It returns a resource R, such that ρ [P,R] > 0 if it exists and a flag such that the flag is true if such an entry exists and false otherwise. To avoid repetition in the vertical search, that is, returning a resource that was previously returned, each entry is marked when the resource corresponding to the rank is returned. Further vertical search in the column will return false. The routine for Vertical search follows.

```
Procedure Vertical (P,R,v);

  // P   = Process returned by Horizontal                       //
  // R   = Resource which P is waiting for, if waiting          //
  // v   = flag, v is true if P is waiting, false otherwise     //
  // PRT = Process-Resource table                               //
  // RT  = Resource table                                       //
  // m   = current number of resources                          //
  // Mark= n dimensional boolean array,                         //
  // n is the number of processes, each entry corresponding     //
  // to a process                                               //

   Begin
     v := false;
     for i := 1 to  m  do
      if (PRT [i,P] > 0) and (not Mark [P]) then
        begin
          v := true;
          R := RT[i];
          Mark [P] := true;
          return
        end;
   end;   // Vertical //
```

IV. CENTRALIZED APPROACH TO ON-LINE DEADLOCK DETECTION USING THE HORIZONTAL AND VERTICAL ALGORITHM

A centralized approach to on-line deadlock detection in a distributed computer network is based on the assumption that one site in the network acts as the controller for global resource allocation and deadlock detection. All requests for resources from all sites in the network are sent to the controller which allocates resources and detects deadlock. The Centralized Horizontal and Vertical Algorithm is designed to run on the controller site only. No other site in the network may allocate resources. All available resources in the network are directly controlled by the controller site. User processes may run on the controller site since the resource manager is a separate module at the site dedicated to resource management. The resource manager maintains a table of all resources available in the network, and their location information. The Process-Resource table and the Process table are maintained dynamically as resources are requested for and released. The procedure given below describes the Centralized protocol for deadlock detection, assuming process P# requests for resource R#.

Centralized H&V Algorithm

```
Procedure H&V (P#,R#);

  // Process P# requests for resource R#                          //
  // Stack used to store process names returned by Horizontal     //
  // P is an array which contains process names returned by       //
  // Horizontal                                                   //
  // h is the number of process names returned by Horizontal      //
  // V is a flag which is true if Vertical returns any resource   //
  // deadlock is a flag indicating whether a deadlock exists      //
```

```
//  Mark is an n dimensional boolean array used by Vertical,   //
//  n = current number of processes in process table           //

Begin
  Initialize Mark to false;
  deadlock := false;
  done := false;   stackptr := 1;    Rk := R#;
  While not done do
     begin
       Horizontal [Rk,h,P);
       if Pk ∈P, 1 <= k <= h, such that Pk = P# then
         begin
           deadlock := true;
           done := true
                                        // deadlock detected //
         end else
     begin
       while h >= 1 do
         begin
           stack [stackptr] := P [h];
           stackptr := stackptr + 1;
           h := h - 1
         end;
       V := false;
       while (stackptr > 1) and (not V) do
         begin
           stackptr := stackptr - 1;
           Pk := stack[stackptr];
           Vertical (Pk,Rk,V);
           //                                //
           end;
```

```
            if (stackptr = 1) and (not V) then done := true
          end;
        end;
    end; // H&V //
```

Note that the algorithm may also be written recursively.

The procedure given above for the centralized deadlock detection requires that the algorithm be run each time a request is made for a resource which is not free for immediate allocation. Thus, the network is dealock free prior to each initiation of the algorithm. The advantage of centralized control is that the resource manager is able to encapsulate all critical control information needed for the algorithm and thereby eliminate system wide race conditions between competing processes. All the tables are maintained by the resource manager only. A process is blocked as soon as its request is denied and the process given a rank greater than zero. Thus, every column of the process-resource table can contain no more than one entry greater than zero. Each process is given access to a resource either exclusively or shared; a zero is entered in the process-resource table to designate this. Thus, each resource can have as many zero entries in its row of the process-resource table as the number of processes in the system.

4.1 Verification of the Centralized Horizontal and Vertical Algorithm

Let G = (V,E) be a process-resource graph, where V = {P1,P2,...,Pn} U {R1,R2,...,Rm}. Consider a graph G′ = (V′,E′) constructed from G as follows. Let Pi and Rj be vertices in G such that there is an edge from Pi to Rj.

Combine the pair Pi ---> $\boxed{\text{Rj}}$ into a single vertex P′i and let P′i be a vertex in G′. Call all such vertices blocked vertices, or blocked nodes. Let Pk be an active process vertex in G, such that there is no out-degree from vertex Pk, that is, a process vertex in which the process is not waiting for any resource. Add such vertices to the vertices of G′. Call all such vertices active vertices, or active nodes, or leaf nodes. Therefore,

V′ = {P′i | Pi ∈ V is blocked waiting for Rj} ∪ {P′k | Pk ∈ V is active}

Denote all blocked nodes in G′ with the symbol ◯ and all active nodes with the symbol { }.

Let V′i and V′k be blocked vertices in G′, where V′i ≡ Pi-->Rj and V′k ≡ P k-->Rl. Add an edge from V′i to V′k if Rj is accessed by P k. Add an edge from a blocked vertex Pi-->Rj to an active vertex {Pk} if Rj is being accessed by Pk. The graph G′, so constructed, will be called H&V transformed graph.

<u>Example 4.1</u>: Figure 3 shows the H&V transformed graph, constructed as described above, for the graph of Figure 2.

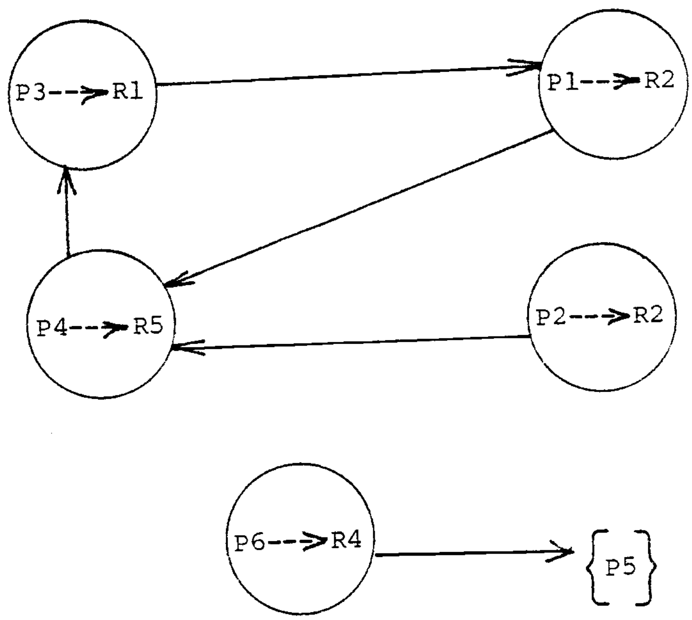


FIGURE 3. H&V Transformed Graph.

Lemma 4.1 A cycle in the H&V transformed graph is produced by a cycle in the process-resource graph.

Proof: The lemma is obvious from the construction of the H&V transformed graph.

Lemma 4.2 The H&V transformed graph has no more than n blocked vertices, where n is the total number of processes.

Proof: Each process is allowed to wait for only one resource at a time. Therefore, there can only be one out-degree from each process vertex in the process-resource graph. Hence, a maximum of n blocked vertices in the transformed graph.

Lemma 4.3 The largest cycle in the H&V transformed graph is of length n, where n is the number of processes.

Proof: Let $G' = (V',E')$ be the transformed graph. Only the blocked vertices in G' will contribute to any cycle in G', since there is no out-degree from the active nodes. Every blocked node in G' corresponds to one out-degree arc belonging to one process node in the process-resource graph. From Lemma 4.2 there are at least n blocked nodes in G'. Therefore, there are at most n out-degree arcs belonging to process vertices in the process-resource graph. Therefore, the largest cycle in G' is of length n.

[ ]

Lemma 4.4 The centralized H&V algorithm finds a cycle in the H&V transformed graph, if there is any.

Proof: Let $G' = (V',E')$ be the H&V transformed graph, and let $v' \in V'$. We shall prove that the algorithm visits all vertices reachable from v'.

Input to the algorithm are process P# and resource R#, where P# is requesting for R#. Hence, an arc from process node P# to resource node R# in the process-resource graph. Therefore, the vertex v' is a blocked vertex in G'.

We shall prove the lemma by induction on the length of the paths from v' to all reachable vertices $w' \in V'$. Let us denote the length, that is, the number of edges, of the path from v' to a reachable vertex w' by $L(v',w')$. Now line 7 of the algorithm identifies all edges from v' and lines 15-20 stacks all these edges. Lines 22-28 visit all vertices adjacent to v'. Therefore, all vertices w' with $L(v',w') <= 1$ get visited. Now assume that all vertices w' with $L(v',w') <= d$ get visited. It will be shown that all vertices w' with $L(v',w') = d + 1$ also get visited. Let w' be a vertex in V' such that $L(v',w') = d + 1$. Let u' be a vertex that immediately precedes w' on a path v' to w'. Therefore, $L(v',u') = d$, and hence u' is visited by the algorithm. Assume $u' \neq v'$ and $d >= 1$. Therefore, immediately u' gets visited, line 7 identifies all edges from u' and lines 15-20 place them on the stack.

The algorithm terminates either when the stack is emptied and all the vertices reachable from v' have been visited, or a cycle is identified in line 8 of the algorithm. Hence, all the edges from u' are removed from the stack and either the vertices reachable from u' are live nodes or one of them is the blocked vertex v'. In the former case, the algorithm will terminate because there are no out-degree arcs from live nodes. And in the later case, $w' = v'$ and a cycle will be identified. Therefore, the algorithm finds a cycle if there is one.

[ ]

Theorem 4.1 The Centralized H&V algorithm detects a deadlock, if one exists.

Proof: A deadlock in the system implies a cycle in the process-resource graph. From Lemma 4.1 a cycle in the H&V transformed graph is produced by a cycle in the process-resource graph. Also from Lemma 4.4 the algorithm finds a cycle in the transformed graph if one exists. Therefore, the centralized H&V algorithm detects a deadlock if one exists. Also from Lemmas 4.2 and 4.3, the algorithm will terminate.

[ ]

4.2 Example

Assume the state of a system at a particular instance is represented by the Process-Resource graph of Figure 4.

The Process-Resource table as maintained by the resource manager is given in Table 5.

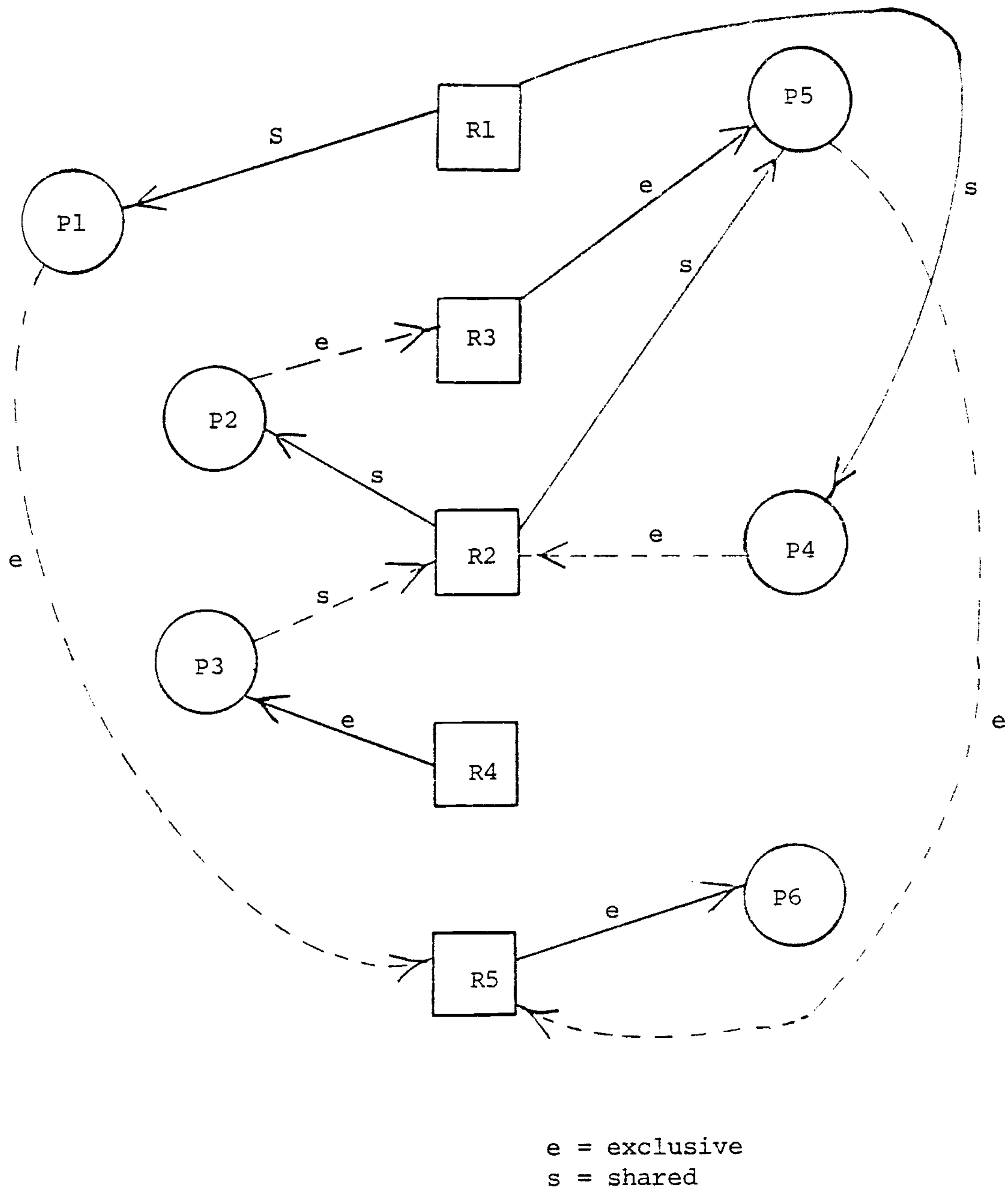


FIGURE 4. Process-Resource Graph for Centralized H&V Example

TABLE 5. Process-Resource Table for Centralized H&V Example

	P1	P2	P3	P4	P5	P6
R1	Os			Os		1e
R2		Os	2s	1e	Os	
R3		1e			Oe	
R4			Oe			
R5	1e				2e	Oe

e exclusive access

s shared access

Let P6 request exclusive access of R1. Since R1 is not free a rank of 1 is entered in the Process-Resource table for P6's request. This is shown surrounded with square dashes in the table. Using the algorithm on the table produces the following search paths, shown in table with dashed lines:

Row Search : R1 : P6 → P4
P6 → P1

Column Search : P4 : R1 → R2

Row Search : R2 : P4 → P5
P4 → P2

Column Search : P5 : R2 → R5

Row Search : R5 : P5 → P6 → deadlock detected.

Note that the algorithm terminates immediately a deadlock is detected. Since the main purpose of the algorithm is to check whether it is safe for a process to wait for a non-available resource it serves no purpose to continue the search once a deadlock is detected.

V. DECENTRALIZED APPROACH TO ON-LINE DEADLOCK DETECTION USING THE HORIZONTAL AND VERTICAL ALGORITHM

The centralized control concept of the Horizontal and Vertical algorithm is very easy to implement since the Process-Resource table is centralized and controlled by only one process module. But the drawback in centralized control is very obvious. Failure in the central controller means failure in the whole system, so the centralized control reduces the reliability of the whole system. Secondly, in a large network, having all processes direct their requests to one site may cause a message bottleneck, thereby reducing the performance of the system.

A distributed approach to on-line deadlock detection is based on the assumption that there is no central resource controller. All sites in the network share the responsibilities of resource allocation and deadlock detection. Each site manages its own resources, runs the deadlock detection algorithm, and allocates its own resources to requesting processes.

The distributed H&V algorithm assumes a kind of site ordering in the network. Messages arrive in the order sent--no reordering of messages. The resource manager at each site maintains a resource table for all the resources local to it, and a process table for all processes using or requesting for its resources. It maintains the Process-Resource table for the algorithm. Since the resource manager is the only process running the detection algorithm, there is no concurrency problem in accessing the tables. Each user process makes a request to the resource manager at its site. The resource manager then determines whether the resource is local or not. If the request is for a local resource the resource manager can determine if the desired resource is available for immediate allocation or not.

If it is a request for an external resource, the resource manager sends the request out. If it is for a local resource that is not free or for an external resource, the requesting process is blocked. In either case, when the resource manager receives a request for its resource, it first checks the status of the resource. If it is not free for immediate allocation, it ranks the request and initiates its own detection algorithm. If no deadlock is detected locally, it sends the detection Path to the next site in the order. The detection Path consists of process names. When the Vertical routine returns false, the process name which was input to the routine is added to the list of processes in the Path.

When a site receives a detection Path, the message is passed to the resource manager at that site. Using the information in the Path and Disjoint Path, the resource manager runs its detection algorithm, producing a new Path. Now, before the site that initiated the detection algorithm sends out the Path, it sets the message origin to itself and the process name entry in the message identification to the process that made the request. If a deadlock is detected at the current site, the site sets the message destination to the site that initiated the detection, and "deadlock" to true. The message is then sent directly to the originator. If no deadlock is detected, the detection message is sent to the next site in the order.

When the site that initiated the detection receives the message back, it first checks the process-resource table to see if the process is still waiting for the resource. Note that it is possible for a resource to be free before the final detection message arrives. Since the resource manager continues processing other messages after sending out the detection Path, it is possible for the resource to be released before the detection Path arrives back. If the process had been allocated the resource, or "deadlock" is false, the message is discarded. If "deadlock" is true, then the request causes a deadlock. The resource manager must initiate its roll-back mechanism. The roll-back mechanism used depends on a particular implementation and is beyond the scope of this thesis. However, in the simulation, the waiting process is rolled back, releasing all the resources it acquired. It is later restarted after a random amount of simulated time. A formal description of the algorithm is given below.

DECENTRALIZED H&V ALGORITHM

Sample Detection Message Format

Message Type : Detection

Message Origin: Site initiating

Message Destination:

Process Name: Requesting process name

Resource Name: Resource being requested

Deadlock: Flag indicating whether deadlock exists

Path: List of process names in the search path, starting at the requesting process node

Disjoint Path: Sets of processes, $\{Pi,Pk,\ 1 <= k < n,\ i \neq k,$ Such that there is a path from Pi to Pk, Pi has rank $> 0\}$. Pi is the identification of the set and n is the number of processes in the system.

The Decentralized H&V Algorithm uses the routine given below.

```
Procedure Detect (Pi,Rj,PP);
  Begin
    done := false;  stackptr  := 1;  Rk := Rj;
    While not done do
      begin
        Horizontal (Rk,h,P);
        if Pk ∈ P, 1 <= k <= h, such that Pk = P# then
           begin
             deadlock := true;
             done := true
           end else
           begin
             while h >= 1 do
               begin
                 stack [stackptr] := P[h];
                 stackptr := stackptr + 1;
                 h := h - 1
               end;
             v := false;
             while (stackptr > 1) and (not v) do
               begin
                 stackptr := stackptr - 1 ;
                 Pk := stack [stackptr];
                 Vertical (Pk,Rk,v);
                 if not v then add Pk to PP
               end;
             if (stackptr = 1) and (not v) then done := true
           end;
      end;
   end; // Detect //
```

Let site St receive a request for resource R# by Process P#. St enters the rank of the request in its local process-resource table.

The Algorithm

STEP A // Site initiating the detection algorithm runs this step //

1. Set Message origin to St, process name to P#, resource name to R# and deadlock to false.
2. Initialize "MARK" to false. (MARK is as explained in the Centralized algorithm)
3. Call Procedure Detect with P# and R# as arguments. This performs the horizontal and vertical search using the partial process-resource table contained within the local site.
4. If deadlock is detected locally, then stop, and resolve it.
5. Set Path to PP. PP is the output returned from Detect.
6. Mark all entries greater than zero in the R# row of the process-resource table.
7. Let {Rk,Pk} be an unmarked entry greater than zero in the process-resource table. Call procedure Detect with Pk and Rk as arguments and do not check for deadlock in Detect. The procedure returns the path P1,P2,...Pj, j >= 1, in PP. Append the set {Pk,P1,P2,...,Pj} as entry in the Disjoint Path. Pk is the identification of this set. Mark the {Rk,Pk} entry in the process-resource table.

   Repeat step 7 until all entries greater than zero in the process-resource table have been marked.
8. Remove duplicate process names in Path.
9. Enter the next site in the order, in the Message destination portion of the detection message, and send the message to this site.

STEP B // Other sites receiving the detection message run this step //

1. Initialize MARK to false, a variable, P#, to the entry in the process name of the message.
2. If there is a process name in Path but not in the local process table, or the process name is in the local process table but is not currently waiting for any local resource, append the process name to new Path.
3. Let process Pi be a process name in Path that is waiting for resource Rj at this site. Call procedure Detect with Pi and Rj as arguments. Mark the [Rj,Pi] entry in the local process-resource table.
4. If deadlock is detected go to step 10, else append process names returned from Detect in PP to the new Path.
5. Repeat step 3 for all processes in Path that are waiting for resources at this site. Check for deadlock each time as in step 4.
6. Remove duplicate entries in new Path.
7. Let {Rk,Pi} be an unmarked entry greater than zero in the process-resource table. Call procedure Detect with Pi and Rj as arguments, and do not check for deadlock. The procedure returns the path P1,P2,...,Pk, k >=1, in PP. Append the set {Pi,P1,P2,...,Pk} as entry in the Disjoint Path. Mark the [Rk,Pi] entry in the process-resource table.

   Repeat step 7 until all entries greater than zero have been marked. Note that this step is similar to step 7 of STEP A.
8. Let Pk be an entry in the new Path. If there is a set {Pi,P1,P2,...,Pj} in the Disjoint Path, such that Pk = Pi, then if there exists Pl in {P1,P2,...,Pj} such that Pl = P# there is a deadlock, go to step 10, else replace Pk in the new Path with P1,P2,...,Pj. Delete the set {Pi,P1,P2,...,Pj}

from the Disjoint Path. Note that Pi is unique within the Disjoint Path since a process can only wait for one resource at a time.

Repeat step 8 until there is no Pk in the new Path equal to Pi in the Disjoint Path.

9. The new Path becomes the Path in the detection message. Remove duplicate entries in Path. Set the message destination to the next site in the order, and send the message to this site.
10. Set deadlock in the message to true, message destination to the message origin. Drop the Path and Disjoint Path portions from the message and send the message to the site that initiated the detection algorithm.

STEP C // Site that initiated the detection receives message //

1. Check if P# is still waiting for R#. If it is not, then discard message.
2. If "deadlock" is true, then P#'s request for R# causes a deadlock. The request must be denied and P# advised to roll back. If "deadlock" is false then there is no deadlock.

* * * * * * end of algorithm * * * * * *

The Decentralized H&V algorithm, as described above, requires running the algorithm every time a resource is not free for immediate allocation. Hence, a deadlock is detected and removed immediately there is one. Also the roll-back mechanism at each site is simplified. There will be no messages generated by the roll-back mechanism as the sites do not have to coordinate their roll-back activities. Also the algorithm assumes that all inter-site messages eventually get received by the proper sites, therefore no detection Path is lost in transmission between sites.

5.1 Verification of the Decentralized Horizontal and Vertical Algorithm

Before giving the proof of the algorithm, let us review the following basic graph definitions.

DEFINITION 5.1 A Directed Graph G = (V,E) is a finite nonempty set V of vertices, together with a set E of edges, disjoint from V, of ordered pairs of distinct elements of V.

DEFINITION 5.2 A Directed Acyclic Graph is a directed graph with no cycles.

DEFINITION 5.3 A Forest is a directed graph consisting of a collection of directed acyclic graphs.

DEFINITION 5.4 A Directed Path in a directed graph is a sequence of ordered edges of the form (V1,V2), (V2,V3), ..., (Vn-1,Vn). It may be represented by the sequence V1,V2,...,Vn of vertices on the path. The length of the path is the number of edges on it. A path is simple if all the edges and all the vertices on the path, except possibly the first and the last vertices, are distinct. If the first and the last vertices are the same then the path is a cycle.

DEFINITION 5.5 The Union G1 U G2 of two directed graphs G1 and G2 is that directed subgraph with vertex set V(G1) U V(G2) and edge set E(G1) U E(G2).

Let G1 = (V1,E1) and G2 = (V2,E2) be two directed acyclic graphs. The following are stated from the definitions.

1. G = G1 U G2 is a forest if $V(G1) \cap V(G2) = \emptyset$
2. G = G1 U G2 combine into one directed graph if G1 and G2 contain at least one vertex in common, assuming G1 and G2 are connected.

Lemma 5.1 Let $G1 = (V1,E1)$ and $G2 = (V2,E2)$ be two directed acyclic graphs with directed paths $V1,V2,\ldots,Vi$, $i >= 3$, and $U1,U2,\ldots,Uj$, $j >= 3$, respectively. If the two paths contain one vertex, W, in common, then $G = G1 \cup G2$ has a path containing W.

Proof: Let $Vp = V1,V2,\ldots,Vi$ and $Up = U1,U2,\ldots,Uj$. Then $G1 \cup G2$ will contain the paths $V1,V2,\ldots,Vi \cup U1,U2,\ldots,Uj$.

Case 1: $W = Vi = U1$. Then the path of G containing W will be $V1,V2,\ldots,W,U2,\ldots,Uj$.

Case 2: $W = V1 = Uj$. Then the path containing W will be $U1,U2,\ldots,W,V2,\ldots,Vi$.

Case 3: $W = Vk = Ul$, $1 < k < i$ and $1 < l < j$. G will contain the paths

$V1,V2,\ldots,W,\ldots,Vi$; $V1,V2,\ldots,W,\ldots,Uj$;

$U1,U2,\ldots,W,\ldots,Uj$; $U1,U2,\ldots,W,\ldots,Vi$.

Therefore, G has at least one directed path containing W. [ ]

Lemma 5.2 Let $G1 = (V1,E1)$ and $G2 = (V2,E2)$ be two directed acyclic graphs with directed paths $Vp = V1,V2,\ldots,Vi$, $i >= 3$ and $Up = U1,U2,\ldots,Uj$, $j >= 3$, respectively. $V1,V2,\ldots,Vi$ and $U1,U2,\ldots,Uj$ are distinct vertices of G1 and G2, respectively. If there are two common vertices X and Y to both graphs such that there is a path from X to Y in G1, and a path from Y to X in G2, each of length at least 2, contained in Vp and Up, respectively, then $G = G1 \cup G2$ contains a cycle.

Proof: Let G = G1 U G2. Then G contains the paths Vp U Up = V1,V2,...,Vi U U1,U2,...,Uj.
Let ℓ be the smallest integer such that $V\ell$ = Uk. Then V1, ..., $V\ell$, Uk+1, ..., Uj is a path in G and contains a cycle. The vertices X and Y are contained in the path.

Observation 5.1
A process-resource graph is a directed graph. It may be acyclic, or it may contain a cycle. In the former case, the system represented by the graph contains no deadlock, and in the later case the system contains a deadlock. If a process-resource graph is partitioned into subgraphs, then the union of all the subgraphs will be the original process-resource graph. We shall call all such subgraphs "process-resource subgraphs."

Observation 5.2
The process-resource table maintained at each site is a representation of the process-resource subgraph(s) at that site. The union of all the process-resource subgraphs from each site is the global process-resource graph.

Lemma 5.3 Let process Pi1,Pi2,...,Pik wait for resource R#. Assume the system is deadlock free. If another process P# later request for R#, then Pi1,Pi2,...,Pik will not contribute to the fact whether P#'s request causes a deadlock or not.

Proof: Assume P#'s request causes a deadlock consisting of processes P#,P1,P2,...,Pn, as shown in Figure 5.

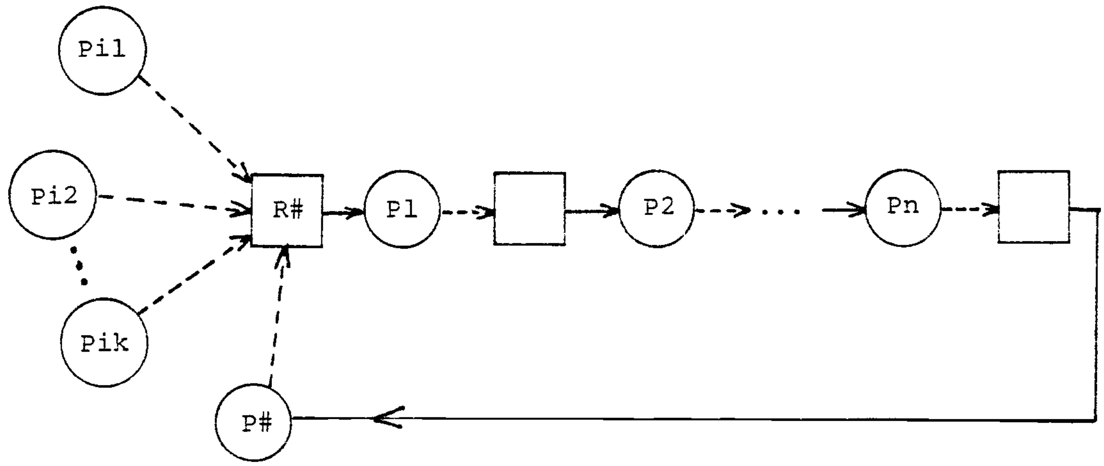


FIGURE 5. Deadlock cycle involving Processes P#,P1,P2,...,Pn

From the figure it is apparent that none of the processes Pi1,Pi2,...,Pik can be in the deadlock path, since Pi1,Pi2,...,Pik and P# all have directed edges into R#. Therefore, Pi1,Pi2,...,Pik will not contribute to a deadlock state caused by process P#. [ ]

Theorem 5.1 The decentralized H&V algorithm described above detects a deadlock, if one exists.

Proof: The proof of the algorithm is based on the definitions, lemmas, and observations given above. Let process P1 request for resource R#. Consider a global deadlock cycle, as shown in Figure 6, consisting of processes P1,P2,...,Pn. Assume that none of the processes involved in the deadlock is aborted or rolled back while the search induced by P1's request is in progress. The former would break the deadlock before it is detected, while the later implies that the deadlock had been detected by a search induced by one of the other processes in the cycle.

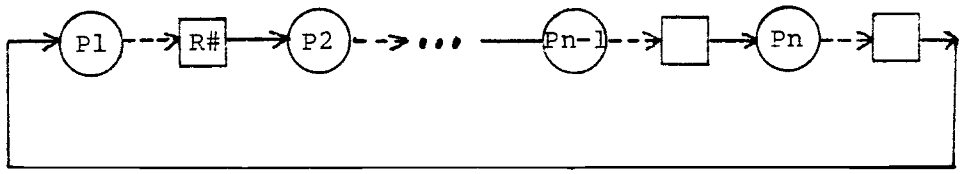


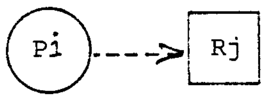
FIGURE 6. Global deadlock cycle involving Processes P1,P2,...,Pn

First, it will be shown that, if a deadlock exists locally at the site where R# resides, it will be detected at step 3 of step A of the algorithm. Observe that procedure Detect is similar to procedure H&V which is used for the Centralized algorithm. The only difference is that if Vertical routine, called at line 26 of procedure H&V, does not find an entry greater than zero, it drops the current search path, whereas procedure Detect will add the path to PP to be returned (line 24).

Therefore, Theorem 4.1 holds for procedure Detect also. Therefore, if a local deadlock exists it will be detected by the decentralized H&V algorithm.

Assume a deadlock involving more than one site. The search begins at step 3 of step A of the algorithm with a process-resource subgraph beginning at node P1. By theorem 4.1, procedure Detect will traverse all paths starting from node P1, producing the Path Pi1,Pi2,...,Pik. Note that processes Pi1,Pi2,...,Pik consist of active nodes in the search path from P1, as seen by the current site. If a deadlock exists then at least one of these processes will be in the deadlock cycle.

Step 6 of step A is justified by Lemma 5.3. Step 7 is a search of the remaining unsearched subgraphs, beginning at each blocked node, namely



.

By Theorem 4.1, procedure Detect will produce the search path Pi---> Rj--->••••--->Pk. Only the beginning blocked process, Pi, and the ending active process, as seen by the current site, will make up the set {Pi,Pk} in the Disjoint Path which is to be sent to the next site. Pk may be in the deadlock cycle, but it is not known at this stage. Therefore, step 7 will search all possible process-resource subgraphs, that may contribute to a deadlock situation.

Let site k receive the detection message. Site k runs step B of the algorithm. As far as the previous site was concerned, the processes in Path are active, but they may be blocked at current site k. Step 2 saves all the processes in Path that are really active, as viewed by current site. Let Pi be in Path. Assume Pi is blocked at site k. This means there is a process-resource subgraph induced by Pi at this site. By Lemma 5.1, a path exists from P1, containing Pi. Step 3 to step 5 searches all such paths. The resultant search path will therefore be P1,...Pi,...P1. If P1 = P1 then by Lemma 5.2 a cycle will be detected. And therefore, the pending deadlock will be detected at step 4.

Assume P1 $\neq$ P1.

Step 7 is similar to step 7 of step A. Therefore, all subgraphs induced by processes blocked at current site, which were not in Path will be searched, and the paths entered in Disjoint Path. Let {Pj,Pg} be a set in Disjoint Path. This implies a path beginning at Pj and ending at Pg exists in the global process-resource graph. Also, a path exists beginning from P1 and ending at P1, as shown above. Now, step 8 performs a union of P1,...,P1 and Pj,...,Pg, for all sets in which P1 = Pj. Therefore, by Lemma 5.1, step 8 will produce the Path P1,...,P1 U Pj,...,Pg = P1,...,Pm,...,Pg, if P1 = Pj, where P1 = Pj = Pm. After this operation, Pg will be in Path. Step 8 also checks if Pg = P1. If it is, then by Lemma 5.2, a cycle exists. Therefore, a deadlock will be detected.

If no deadlock is detected at site n, assuming an n-site network, then P1's request does not cause a deadlock, since at this point all possible process-resource subgraphs have been searched, and their union performed if possible. Therefore, the decentralized protocol will not detect a deadlock, if none exists.

To complete the proof of the decentalized algorithm, we shall prove Corollary 5.1 and then make some observations.

Corollary 5.1 Let process Pi request for resource Ri at site Si, and at the same point in time process Pj requests for resource Rj at site Sj. The distributed H&V algorithm initiated simultaneously at Si and Sj will detect a deadlock if there is one.

Proof: Two cases will be considered in proving this corollary.

Case 1: A search path containing both Pi and Pj.

Case 2: Different search paths for process Pi and process Pj.

Case 1: We shall assume that neither Pi nor Pj is rolled back or aborted, while the searches initiated by Si and Sj, respectively, are in progress.

Assume that at the instance Si and Sj initiate their H&V algorithm, a deadlock cycle exists in the network containing both processes Pi and Pj. Since a deadlock exists, processes Pi and Pj will remain blocked waiting for their requested resources. Therefore, any changes in the process-resource tables in which either process has entry will not change the state of these processes. Now, the search paths initiated by sites Si and Sj will be expanded independently at the different sites in the network. But, effectively, the sites will be searching the same path. One of the paths will be searched before the other, since the process-resource tables at a site is accessed serially by the resource manager at that site. Therefore, by Theorem 5.1, the same deadlock will be detected and reported to both Si and Sj.

Now, assume that there is no deadlock in the system caused by either Pi or Pj. Therefore, by Theorem 5.1, there exists an active process, Pk, in the search path induced by these two processes. Although there will be two identical paths going around (one initiated by Si and the other by Sj), both paths will contain Pk, Pi, and Pj. Therefore, by Theorem 5.1, no deadlock will be reported to either Si or Sj.

<u>Case 2</u>: This case is similar to Theorem 5.1; although there are two different search paths going around, they are independent of each other. Therefore, by Theorem 5.1, a different deadlock cycle will be reported to the respective sites if there is one, caused by the request to the site, and no deadlock will be reported if there is none. [ ]

<u>Observation 5.3</u>

Since a resource is allocated to a waiting process immediately after it is freed, then the protocol does not delay allocation of a freed resource.

<u>Observation 5.4</u>

If a particular request causes a deadlock, then the processes involved will not change their states until the deadlock is broken. Therefore, the states of these processes will not be changed in the individual process-resource tables where they have entries. This means that any change in a process-resource table after the algorithm had been run will not change the outcome.

Observation 5.5

If the network is deadlock free, then neither releasing resources held by completed processes for which there are no waiting access, nor allocating the released resources to the next processes in rank leads to a deadlock. This means that, if Pi and Pj have ranks, say 1 and 2, respectively, for a resource, then changing the ranks to 0 and 1, respectively, when the resource is free, will not lead to a deadlock, if none existed.

Observation 5.6

Corollary 5.1 holds for any number of sites greater than one. In the worst case, the algorithm will report the same deadlock cycle to all the sites that simultaneously initiated their detection routines with processes involved in the same deadlock. Which process in the cycle to roll back will depend on the rollback mechanism in use in the network. But for this study all processes whose request caused the deadlock were simultaneously rolled back by their respective resource managers. This is a case of over-detection. At least it leaves the network deadlock free.

This completes the proof of the decentralized H&V altorithm.□

Highlight of the New Algorithm

1. The algorithm requires looking at each process-resource table only once. There is no passing of detection information forwards and backwards many times as is characterized by Goldman's algorithm. The H&V algorithm will be run in at most n sites (n is the total number of sites in network), whereas in Goldman's algorithm, the number of sites that may run the algorithm, per initiation, may grow much larger than n. Therefore, synchronization problems due to communication delays are reduced to minimum in the H&V algorithm.

2. Goldman's algorithm requires the formation of a different copy of the OBPL for each shared resource. Each copy is expanded independently. In a system with many shared resources the algorithm leads to a heavy overhead in communication and time to run the algorithm. The H&V algorithm does not require any special way of handling shared resources. Each deadlock detection initiation requires only one detection message.

5.2 Example

Consider the configuration of Figure 1, and assume that P10 at site S3 requests exclusive access to resource R1 at site S1. Since all requests and accesses are assumed exclusive, the type of access entry will be omitted in the process-resource tables. The process-resource tables at each site are shown in Table 6.

The resource manager at each site is responsible for detecting any impending deadlock, as a result of a request for a resource at its site. For our example, the resource manager at site S3 sends P10's request to the resource manager at site S1.

TABLE 6. Process-Resource Tables for Distributed H&V Example

SITE S1:

	P1	P2	P3	P10
R1			0	$\boxed{1}$
R2	0	1		

SITE S2:

	P8	P4	P9	P5	P10	P6
R3	0	1				
R4			0	1		
R5					0	1

SITE S3:

	P4	P3	P5	P8	P6	P9	P11	P7
R6	0	1						
R7			0	1				
R8					0	1		
R9							0	1

Site S1:

A rank of 1 is entered for the new request, since R1 is being held by P3. P3 has a rank of zero for R1. The message packet set up by S1 looks like that shown below.

Detection
S1
Destination
P10
R1
False
Path
Disjoint Path

S1 initiates the decentralized Horizontal and Vertical algorithms, producing the path and disjoint path shown in the table. The message to be sent to the next site is as shown below.

Detection
S1
S2
P10
R1
False
P3
{P2,P1}

Site S2:

On receiving the detection message from site S1, the resource manager at S2 initiates its own detection algorithm. It runs step B. Process P3 in Path is not in the process table at S2. So P3 is retained in Path. The paths produced by disjoint path search is shown in the table. The sets {P4,P8}, {P5,P9} and {P6,P10} are the disjoint paths produced, which are appended to the message. Since there is no set in the Disjoint Path that has P3 as its identification, S2 assembles the message as shown below and forwards it to S3.

Detection
S1
S3
P10
R1
False
P3
{P2,P1}, {P4,P8}, {P5,P9}, {P6,P10}

Site S3:

P# <--- P10. P3 is in the process table, and is waiting for R6. A search using P3 produces P4 in Path. The disjoint path produced are {P8,P5}, {P9,P6}, {P7,P11}. After step 7 of the algorithm, Path and Disjoint Path will lock as follows.

~~P4~~ ~~P8~~ ~~P5~~ ~~P9~~ ~~P6~~ P10						
	1	3	5	2	4	
{P2,P1},	~~{P4,P8}~~,	~~{P5,P9}~~,	~~{P6,P10}~~,	~~{P8,P5}~~,	~~{P9,P6}~~,	{P7,P11}

Step 8 of the algorithm expands the sets in the disjoint path in the order shown.

P10 = P#, therefore S3 will detect the deadlock, and forward the message as shown below to S1.

Detection
S1
S1
P10
R1
True

The Path and Disjoint Path portions of the message are dropped since they are no longer needed.

The reader is urged to compare this example with the Goldman's example 2.1.

VI. SIMULATION STUDY OF THE HORIZONTAL AND VERTICAL ALGORITHMS AND GOLDMAN'S ALGORITHM ON A RING NETWORK

In Chapters III, IV and V, two new protocols were presented for detecting deadlocks in distributed computer systems. Two main features were considered in the design of the distributed protocol. First, the reduction of communication overhead resulting from the invocation of the algorithm, and second, limiting the number of sites that are to run the algorithm in order to detect an occurrence of a deadlock, or to verify the nonexistence of a deadlock. We claim that these features will result in an improved response time and throughput over Goldman's algorithm.

In the next section, we shall present some simulation results to support our claim. It must be emphasized that the simulation results are for a unidirectional ring network computer system. Section 6.1 gives a formal definition of the experiment, and the experiment results are discussed in Section 6.2.

6.1 Experiment Definition

TITLE: ON-LINE DEADLOCK DETECTION ALGORITHMS ON A RING NETWORK

TYPE: Performance

OBJECTIVE:

The purpose of the experiment is to gather experimental data to measure:

1. the performance of three deadlock detection algorithms and a deadlock preventive algorithm,
2. the probabilities of occurrence of deadlock, and
3. communication overhead associated with the use of each algorithm, in a distributed computer system environment, where resources are randomly requested and released by processes.

The three detection algorithms studied are the Centralized and Decentralized Horizontal and Vertical algorithms described in Chapters IV and V and Goldman's algorithm discussed in Chapter II. The prevention algorithm studied is preemption, which does not run a deadlock algorithm. A process is rolled back immediately if its request cannot be granted immediately.

RATIONALE:

The growing importance of distributed systems has increased the importance of on-line deadlock detection. Many solutions to the problem have been proposed, but very few researchers have taken time from theoretical studies to measure the performance of the proposed solutions and the probabilities of deadlock occurrence. Since distributed systems are not widely available, experimental data cannot be gathered in a practical environment to measure their performances. Some method has to be devised to do this. It is the purpose of this thesis to provide some simulation data on the operational behavior of detection algorithms in a simulated distributed computer system.

APPROACH:

The simulation programs are written in Path Pascal [4]-[8] [49]. The decision to use Path Pascal was made because no other compiler that supports concurrent processes was available. Since Path Pascal was used only as a tool in the simulation, no discussion of this programming language will be given. However, a brief description and listings of the simulation programs are given in the appendices. The interested reader is referred to the references. Path Pascal provides efficient mechanism for simulating concurrent processes. The Path Pascal was implemented on Cyber at Oregon State University.

The simulation was done on a unidirectional ring network. A study of traffic and message delay in ring networks can be found in [33] and [43]. Both centralized and distributed control environments were assumed. The Centralized Horizontal and Vertical algorithm was used in the implementation of the centralized control. This is based on the premise that one site in the network acts as the controller for global resource allocation and deadlock detection. All requests for resources from all sites in the network are sent to the controller which allocates resources and detects deadlock. The Horizontal and Vertical algorithm runs only on the controller site. No user process runs on the controller site.

The distributed Horizontal and Vertical algorithm and Goldman's distributed algorithm were also implemented in a decentralized control environment. The preemption technique was also run in a decentralized environment, e.g., all sites in the network share the responsibilities of resource allocation and deadlock detection. There is no central control of resources. Each site manages its own resources, runs the deadlock detection algorithm, and allocates its own resources to requesting processes. Although no deadlock detection is run in the preemption technique, each site has a resource manager which checks on the availability of a resource for immediate allocation.

In all the detection simulation models, deadlock detection is initiated every time the requested resource is not free for immediate allocation. When a deadlock is detected, the process involved is rolled back to the beginning, releasing all resources it held, delayed a random number of simulated time units, and then started again from the beginning. Broadcast mode of communication was used in all decentralized control cases. In this mode each site has no knowledge of the locations of the resources, except its own. Requests for external resources are broadcast over the network. Another alternative mode of communication would be a point-to-point mode. In this mode, it is assumed that each site has knowledge of the location of all resources available in the network. Request for external resource is sent directly to the site which owns the resource. This mode would only be meaningful in a fully connected network. Since messages will pass through all the sites in a unidirectional ring network, the performance results obtained would not be affected by whichever mode of communication was used. So the choice of broadcast mode was arbitrary.

The performance of the algorithms was measured in terms of response time and throughput. Response time, sometimes called waiting time or turnaround time, is the length of time from a request for service until the request is completed. Throughput is a measure of the number of requests processed per unit time. Two response time measurements were made for each algorithm.

1. Process response time: This is the average turnaround time for a process in the system. For the purpose of this measurement, each process was subjected to equal number of resource requests.
2. Request response time: This is the average time delay between making a request and getting acknowledgment. This time delay includes the network message delay and the time to run the detection algorithm. An acknowledgment was considered to be a message from the resource manager:
   a. granting the request--in this case, the resource was free for immediate allocation;
   b. informing the requesting process to wait for the resource--in this case, a detection algorithm had been initiated and no deadlock was detected; or
   c. asking the requesting process to roll back--in this case, a deadlock had been detected.

In Goldman's algorithm, multiple copies of OBPL are created if a resource is held under shared access. So it was possible for a process to receive more than one "notfree" message or receive a "rollback" message after it had received a "notfree" message. The request response time was, therefore, the time the requesting process received the last "notfree" message or a "rollback" message. A "notfree" message is a message informing the requesting process that the resource is not free for immediate allocation. A "rollback" message informs the process that its request caused a deadlock. In the former case, the process remains blocked waiting for the resource, while in the latter the process is rolled back.

Two throughput measurements were also made for each algorithm:

1. Process throughput: This is the average number of processes completed per unit time, and
2. Request throughput: This is the average number of requests processed per unit time.

Communication overhead was measured in terms of the expected number of message units passed per request. Frequency of deadlock was measured in terms of the average number of deadlocks detected in the system per request. A Poisson rate of resource request by each process or, in other words, exponential holding times for a process and a random selection of a resource by a process was assumed. The time unit used is the simulated time provided in the Path Pascal interpreter. All messages were processed on a first-come-first-serve basis.

6.2 Results of the Simulation Study

Very few articles have been reported on the analysis of deadlock frequency in computer systems. The only article worth mentioning is a report by Ellis [21] on the probability of increase or decrease of deadlocks as the numbers of processes and resources within a computer system increase. The approach taken in the analysis is to view a state diagram used to represent process-resource interactions as a finite state automation. A probability measure is attached to an occurrence of each possible transition. The analysis is given for small systems only. A random resource allocation model is assumed in the analysis. Results of the analysis show that for fixed numbers of processes the probability of deadlock decreases as the number of resource increases. Conversely, for fixed numbers of resources the probability of deadlock increases as the numbers of processes increase, since more processes now compete for the same number of resources. Since this analysis was done for very small systems, it does not provide any basis for comparison with the results obtained in the simulation reported in this thesis.

Tables 7 through 21 present the simulation quantities of primary interest from the preemption, distributed H&V, distributed Goldman and Centralized H&V models. Tables 7, 9, 14, and 16 list average response times, and Tables 8, 19, 15, and 17 list average throughputs. Their standard deviations and standard errors are also listed.

The standard errors were computed based on a 95 percent confidence limit following a t-Distribution with N-1 degrees of freedom. N was taken as the total number of processes and requests, respectively. The large standard deviations and standard errors for the process average response times were partly caused by the small number of processes used in the calculations and partly by the fact that some processes completed long before the others. Also, in the request average response time, some requests that were granted immediately because the resources were free for immediate allocation had a much smaller response time than those requests that necessitated the invocation of the detection algorithm. Notice that the standard deviations and standard errors of the request response time for the preemption have relatively much smaller values. This is because no detection algorithm was involved. So the deviations of the response time for each request from each other were small. Secondly, in the distributed implementation, requests for local resources had relatively faster response time than requests for external resources, provided the resources were free for immediate allocation.

Tables 11 and 18 list the average message units per request for all the four models, that is, the average number of messages generated in the network by each request. Each message type was considered to be one message unit. No consideration was given to the differences in the length of each message type or the transmission time. Tables 12 and 19 list the frequency of process rollback. These are the probabilities of deadlock occurrences for the distributed and centralized models. Since no detection algorithm is invoked in the preemption model, the values for the preemption technique are the frequency with which a request was denied. Tables 13 and 20 list the frequency of deadlock detection algorithm initiation. These tables apply to the distributed H&V, Goldman's and Centralized H&V models only. Table 21 lists the probability of deadlock occurrence for varying loading factor, rate of resource request/rate of resource release. This table was obtained using the Centralized H&V model on a three-site network.

Some of the information in Tables 7 through 21 can be displayed better by graphs. In the next two sections, the algorithms' performance measures and their comparison with each other will be discussed using such graphs. The comparisons will be made in terms of average values only.

TABLE 7. Process Average Response Time (Average Delay per Process) for All Algorithms with Varying Numbers of Processes, Each with Equal Numbers of Resource Needs, Competing for 6 Resources on a 3-site Network

Number of Processes	PREEMPTION			DISTRIBUTED H&V			DISTRIBUTED GOLDMAN			CENTRALIZED H&V		
	Average Response Time*	Standard Deviation	Standard Error♦	Average Response Time*	Standard Deviation	Standard Error♦	Average Response Time*	Standard Deviation	Standard Error♦	Average Response Time*	Standard Deviation	Standard Error♦
3	29,494	6,744.374	± 16,755.306	17,410.667	2,728.073	± 6,777.456	20,076.667	3,410.602	± 8,473.089	27,953.0	5,581.827	± 13,867.147
5	72,955.2	6,186.310	± 7,680.088	26,182.4	4,694.729	± 5,828.341	41,696.4	7,290.798	± 9,051.270	45,347.8	8,212.554	± 10,195.598
6	106,740	10,971.068	± 11,515.303	33,175.333	6,187.776	± 6,494.729	47,830.833	5,306.688	± 5,569.933	51,513.333	6,561.727	± 6,887.230
7	129,118.333	20,281.519	± 18,757.953	40,040.857	6,538.602	± 6,047.416	73,749.286	18,869.077	± 17,451.614	54,597.429	7,851.473	± 7,261.663
10	159,249	32,235.584	± 23,058.346	64,369.6	17,637.706	± 12,616.378	87,144.7	16,275.974	± 11,642.321	81,288.7	16,196.632	± 11,585.568

* Average response time measured in units of 100.

♦ Standard error computed based on 95% confidence interval following t-Distribution with N-1 degrees of freedom; N = Total number of processes.

TABLE 8. Process Average Throughput (Average Numbers of Processes per Unit Time) for All Algorithms with Varying Numbers of Processes Competing for 6 Resources on a 3-site Network

Number of Processes	PREEMPTION			DISTRIBUTED H&V			DISTRIBUTED GOLDMAN			CENTRALIZED H&V		
	Average Throughput	Standard Deviation	Standard Error♦	Average Throughput	Standard Deviation	Standard Error♦	Average Throughput	Standard Deviation	Standard Error♦	Average Throughput	Standard Deviation	Standard Error♦
3	.00003391	.00000822	±.00002042	.00005744	.00000839	±.00002084	.00004981	.00000814	±.00002023	.00003577	.00000829	±.00002058
5	.00001371	.00000474	±.00000588	.00003819	.00000856	±.00001062	.00002398	.00000464	±.00000576	.00002205	.00000445	±.00000552
6	.00000937	.00000115	±.00000121	.00003014	.00000627	±.000006581	.00002091	.0000023	±.00000241	.00001941	.00000255	±.00000268
7	.00000774	.00000369	±.00000341	.00002497	.000004679	±.000004328	.00001356	.00000461	±.00000426	.00001832	.00000287	±.00000265
10	.00000628	.00000294	±.0000021	.00001554	.000004414	±.000003157	.00001148	.00000281	±.00000201	.0000123	.00000347	±.00000248

* Time measured in units of 100.

♦ Standard error computed based on 95% confidence interval following t-Distribution with N-1 degrees of freedom; N = total number of processes.

TABLE 9. Request Average Response Time (Average Delay per Request) for All Algorithms with Varying Numbers of Processes Competing for 6 Resources on a 3-site Network

Number of Processes	PREEMPTION			DISTRIBUTED H&V			DISTRIBUTED GOLDMAN			CENTRALIZED H&V		
	Average Response Time*	Standard Deviation	Standard Error♦	Average Response Time*	Standard Deviation	Standard Error♦	Average Response Time*	Standard Deviation	Standard Error♦	Average Response Time*	Standard Deviation	Standard Error♦
3	58.821	30.709	±8.822	141.917	189.951	± 63.982	147.972	125.010	±42.316	277.939	359.082	±103.672
5	59.870	32.043	±8.769	189.891	146.510	± 39.511	268.676	374.468	±87.062	354.617	524.504	±117.139
6	60.680	30.462	±3.620	287.085	358.463	± 85.083	392.585	806.757	±182.637	480.364	761.102	±174.339
7	64.144	31.070	±4.144	311.044	269.866	± 90.344	632.747	788.755	±130.556	569.594	746.176	±149.266
10	67.440	30.775	±3.830	588.665	734.812	±113.506	1007.238	1850.826	±160.859	621.448	706.669	±115.825

* Average response time measured in units of 100.

♦ Standard error computed based on 95% confidence interval following t-Distribution with N-1 Degrees of reedom;
N = Total number of requests.

TABLE 10. Request Average Throughput (Average Numbers of Requests per Unit Time) for All Algorithms with Varying Numbers of Processes Competing for 6 Resources on a 3-site Network

Number of Processes	PREEMPTION			DISTRIBUTED H&V			DISTRIBUTED GOLDMAN			CENTRALIZED H&V		
	Average Throughput	Standard Deviation	Standard Error♦	Average Throughput	Standard Deviation	Standard Error♦	Average Throughput	Standard Deviation	Standard Error♦	Average Throughput	Standard Deviation	Standard Error♦
3	.0170	.0014	±.000342	.007046	.00243	±.000818	.006758	.0023	±.000779	.003598	.0009	±.000260
5	.0167	.0027	±.000739	.005266	.00068	±.0004346	.003722	.0014	±.000325	.002820	.0013	±.000289
6	.01648	.0023	±.000273	.003483	.001102	±.000262	.002547	.00091	±.000201	.002082	.0011	±.000251
7	.01559	.0016	±.0002134	.003215	.0008858	±.000198	.001580	.0017	±.000276	.001756	.0012	±.000245
10	.01483	.0021	±.0002614	.001699	.0009623	±.000150	.0009928	.00018	±.0000291	.001217	.00072	±.000118

* Time is measured in units of 100.

♦ Standard error computed based on 95% confidence interval following t-Distribution with N-1 degrees of freedom; N = total number of requests.

TABLE 11. Average Message Units per Request for all Algorithms with Varying Numbers of Processes Competing for 6 Resources on a 3-Site Network

Number of Processes	Preemption	Distributed H&V	Distributed Goldman	Centralized H&V
3	2.896	4.055	4.417	6.469
5	3.315	4.436	5.514	6.519
6	3.428	4.592	5.778	6.792
7	3.613	4.888	6.028	6.865
10	3.884	5.180	6.653	7.130

TABLE 12. Frequency of Rollback for all Algorithms with Varying Numbers of Processes Competing for 6 Resources on a 3-Site Network

Number of Processes	Preemption	Distributed H&V	Distributed Goldman	Centralized H&V
3	0.1940	0.0268	0.02778	0.04082
5	0.48	0.03636	0.05405	0.04938
6	0.5333	0.0423	0.06098	0.05195
7	0.5429	0.05	0.07534	0.05208
10	0.63	0.0683	0.08163	0.05594

TABLE 13. Frequency of Detection Initiation for the Detection Algorithms with Varying Numbers of Processes Competing for 6 Resources on a 3-Site Network

Number of Processes	Distributed H&V	Distributed Goldman	Centralized H&V
3	.25	.2778	.3061
5	.4264	.4459	.4321
6	.4566	.4912	.5125
7	.50	.5274	.5208
10	.5714	.5918	.5804

TABLE 14. Process Average Response Time (Average Delay per Process) for All Algorithms with Varying Numbers of Sites, Each Running One Process and Having One Unique Resource

Number of Sites	PREEMPTION			DISTRIBUTED H&V			DISTRIBUTED GOLDMAN			CENTRALIZED H&V		
	Average Response Time*	Standard Deviation	Standard Error♦	Average Response Time*	Standard Deviation	Standard Error♦	Average Response Time*	Standard Deviation	Standard Error♦	Average Response Time*	Standard Deviation	Standard Error♦
3	51,636.667	7,379.975	± 18,321.571	16,741.667	2,494.803	± 6,197.934	22,543.33	4,754.245	± 11,802.917	19,671.333	4,123.319	± 10,236.576
5	75,680.5	5,796.429	± 7,206.432	28,076.60	4,596.117	± 5,705.918	35,768	8,077.267	± 10,042.093	35,689.2	6,745.214	± 8,386.013
8	117,772.75	14,726.464	± 12,287.555	44,311.625	5,917.64	± 4,948.057	62,425	11,716.176	± 9,775.813	47,318.375	5,267.854	± 4,395.424
10	149,849.2	17,879.693	± 12,778.163	58,414.7	3,743.214	± 2,675.181	86,839.811	13,813.137	± 9,871.900	72,218.186	4,342.739	± 3,103.646
12	182,824.5	16,268.765	± 10,332.053	88,688.5	18,252.198	± 11,591.703	110,421.583	15,389.763	± 9,773.812	98,800.157	6,312.137	± 4,008.745

* Average response time measured in units of 100.

♦ Standard error computed based on 95% confidence interval following t-Distribution with N-1 degrees of freedom; N = total number of processes.

TABLE 15. Process Average Throughput (Average Numbers of Processes per Unit Time) for All Algorithms with Varying Numbers of Sites, Each Running One Process and Having One Unique Resource

Number of Sites	PREEMPTION			DISTRIBUTED H&V			DISTRIBUTED GOLDMAN			CENTRALIZED H&V		
	Average Throughput	Standard Deviation	Standard Error♦	Average Throughput	Standard Deviation	Standard Error♦	Average Throughput	Standard Deviation	Standard Error♦	Average Throughput	Standard Deviation	Standard Error♦
3	.00001937	.00000469	± .00001164	.00005973	.000009442	± .00002346	.00004436	.00001098	± .00002727	.00005084	.00001041	± .00002583
5	.00001321	.00000316	± .00000507	.00003562	.000006362	± .000007898	.00002796	.00000815	± .00001014	.00002802	.00000596	± .00000741
8	.00000849	.00000521	± .00000435	.00002257	.000002807	± .000002429	.00001602	.00000317	± .00000264	.00002113	.00000228	± .0000019
10	.00000667	.00000367	± .00000262	.00001712	.000002672	± .00000191	.00001152	.00000211	± .00000151	.00001385	.00000522	± .00000373
12	.00000547	.00000289	± .00000184	.00001128	.00000341	± .00000217	.00000906	.00000193	± .00000123	.00001012	.00000318	± .00000202

* Time measured in units of 100.

♦ Standard error computed based on 95% confidence interval following t-Distribution with N-1 degrees of freedom; N = total number of processes.

TABLE 16. Request Average Response Time (Average Delay per Request) for All Algorithms with Varying Numbers of Sites, Each Running One Process and Having One Unique Resource

Number of Sites	PREEMPTION			DISTRIBUTED H&V			DISTRIBUTED GOLDMAN			CENTRALIZED H&V		
	Average Response Time*	Standard Deviation	Standard Error♦	Average Response Time*	Standard Deviation	Standard Error♦	Average Response Time*	Standard Deviation	Standard Error♦	Average Response Time*	Standard Deviation	Standard Error♦
3	58.883	1.6147	±0.8896	138.4	95.778	± 32.719	257.6009	107.6333	±40.2454	355.3507	106.2673	±42.0476
5	108.61	9.69	±4.5285	264.0	236.272	± 60.503	463.9586	191.8837	±79.2096	496.54	62.13	±29.79
8	209.01	44.8	±16.1558	826.408	494.571	±101.544	1,971.1405	658.5033	±121.058	842.78	122.92	±51.938
10	318.76	83.78	±26.7583	1,492.7	627.673	±102.520	3,612.616	2,117.6114	±482.6484	1968.27	699.06	±233.2712
12	727.97	345.86	±117.016	2,411.22	2,489.296	±406.585	5,603.231	2,316.131	±784.0103	3521.2	1,195.99	±404.6433

* Average response time measured in units of 100.

♦ Standard error computed based on 95% confidence interval following t-Distribution with N-1 degrees of freedom; N = total number of requests.

TABLE 17. Request Average Throughput (Average Numbers of Requests per Unit Time) for All Algorithms with Varying Numbers of Sites, Each Running One Process and Having One Unique Resource

Number of Sites	PREEMPTION			DISTRIBUTED H&V			DISTRIBUTED GOLDMAN			CENTRALIZED H&V		
	Average Throughput	Standard Deviation	Standard Error♦	Average Throughput	Standard Deviation	Standard Error♦	Average Throughput	Standard Deviation	Standard Error♦	Average Throughput	Standard Deviation	Standard Error♦
3	.01698	.00053	±.00034	.007225	.0003254	±.000112	.003882	.0013	±.00051	.002814	.0012	±.0005
5	.009207	.00082	±.00043	.003788	.00153	±.000394	.002155	.0011	±.00054	.002014	.0008	±.0004
8	.004784	.00091	±.00032	.001377	.0001481	±.0000304	.0005073	.00021	±.0002	.001187	.0002	±.0001
10	.003137	.00087	±.00028	.000718	.0001102	±.000018	.0002768	.00015	±.00004	.000508	.0002	±.0001
12	.001374	.00064	±.00023	.0004147	.00009462	±.00001545	.0001785	.00013	±.000036	.000284	.0001	±.000034

* Time measured in units of 100.

♦ Standard error computed based on 95 % confidence interval following t-Distribution with N-1 degrees of freedom; N = total number of requests.

TABLE 18. Average Message Units per Request for All Algorithms with Varying Numbers of Sites, Each Running One Process and Having One Unique Resource

Number of Sites	Preemption	Distributed H&V	Distributed Goldman	Centralized H&V
3	2.951	4.579	6.5046	6.099
5	2.995	7.557	10.224	9.864
8	3.041	13.688	19.5776	15.103
10	3.213	16.321	21.6783	17.782
12	3.357	19.326	25.093	21.917

TABLE 19. Frequency of Rollback for All Algorithms with Varying Numbers of Sites

Number of Sites	Preemption	Distributed D&V	Distributed Goldman	Centralized H&V
3	.3415	.0286	.0486	.0340
5	.3603	.0328	.0627	.0563
8	.3962	.0538	.0680	.0581
10	.4127	.0631	.0779	.0651
12	.4541	.0712	.0853	.0741

TABLE 20. Frequency of Detection Initiation for the Detection Algorithms with Varying Numbers of Sites, Each Running One Process and Having One Unique Resource

Number of Sites	Distributed H&V	Distributed Goldman	Centralized H&V
3	.351	.4107	.3604
5	.4115	.4318	.4225
8	.4378	.4828	.4494
10	.4521	.5065	.4749
12	.5108	.5504	.5270

TABLE 21. Frequency of Deadlock for Varying Loading Factor for Centralized H&V on a 3-Site Network Running 6 Processes Competing for 3 and 4 Resources

Loading Factor	3 Resources	4 Resources
.2	.0385	.0418
.4	.061	.0661
.6	.0828	.0895
.8	.0833	.1039
.9	.1075	.1193

6.2.1 Comparison of the Algorithms' Performance for Varying Numbers of Processes on a Three-site Network

Figures 7 through 13 are the graphical representations of some of the information contained in Tables 7 through 13.

Figures 7 and 9 show graphs of process average response time and request average response time, respectively, versus the number of processes for all four algorithms--Preemption, Distributed H&V, Distributed Goldman and Centralized H&V. From Figure 7, we observe that the preemption technique has the worst process average response time, hence the lowest average throughput, as Figure 8 shows. The very poor performance of preemption is caused by the high frequency of rollback involved, Figure 12.

But, notice from Figure 9 that preemption has the best request average response time, and subsequently the best request average throughput, Figure 10. Request response time is fast because there is no deadlock detection mechanism involved. Also, the graphs suggest that as the numbers of processes increase, preemption would continue to perform very poorly. Therefore, basedon information from this study, we conclude that preemption method of deadlock resolution is totally unacceptable in a distributed computer system environment.

The performances of Distributed H&V, Goldman and Centralized H&V require very careful study. First, notice from Figure 11 that Centralized H&V has the highest average message units per request. This may not be surprising, since all requests are directed to one site in the network. In distributed implementations a local request for a local resource does not generate any messages, if the resource is available for immediate allocation. But in a Centralized control, such request must be sent out to the controller, thereby increasing the number of messages passed around in the network.

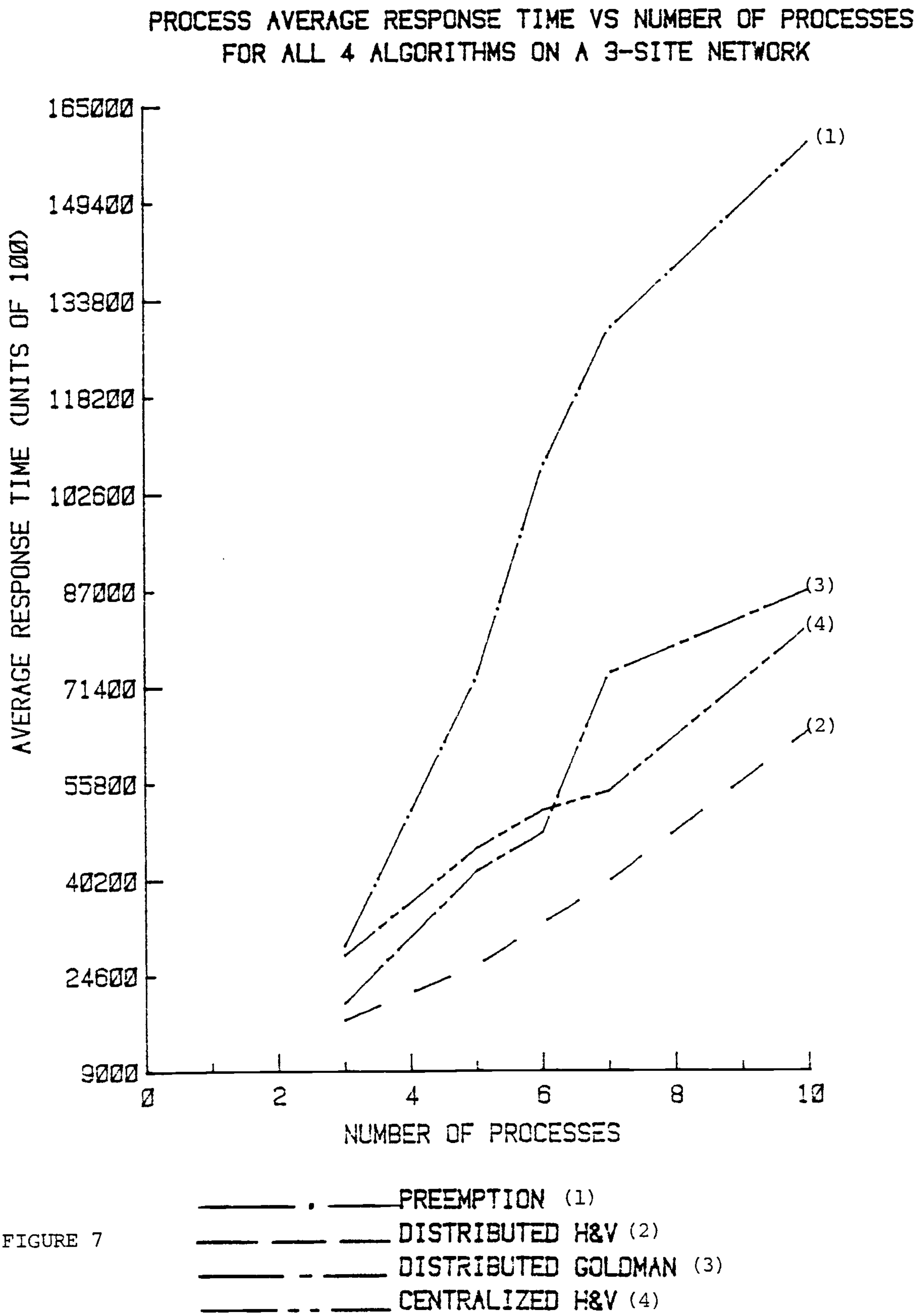

PROCESS AVERAGE RESPONSE TIME VS NUMBER OF PROCESSES
FOR ALL 4 ALGORITHMS ON A 3-SITE NETWORK
AVERAGE RESPONSE TIME (UNITS OF 100)
165000
149400
133800
118200
102600
87000
71400
55800
40200
24600
9000
0
2
4
6
8
10
NUMBER OF PROCESSES
(1)
(2)
(3)
(4)
PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)

FIGURE 7

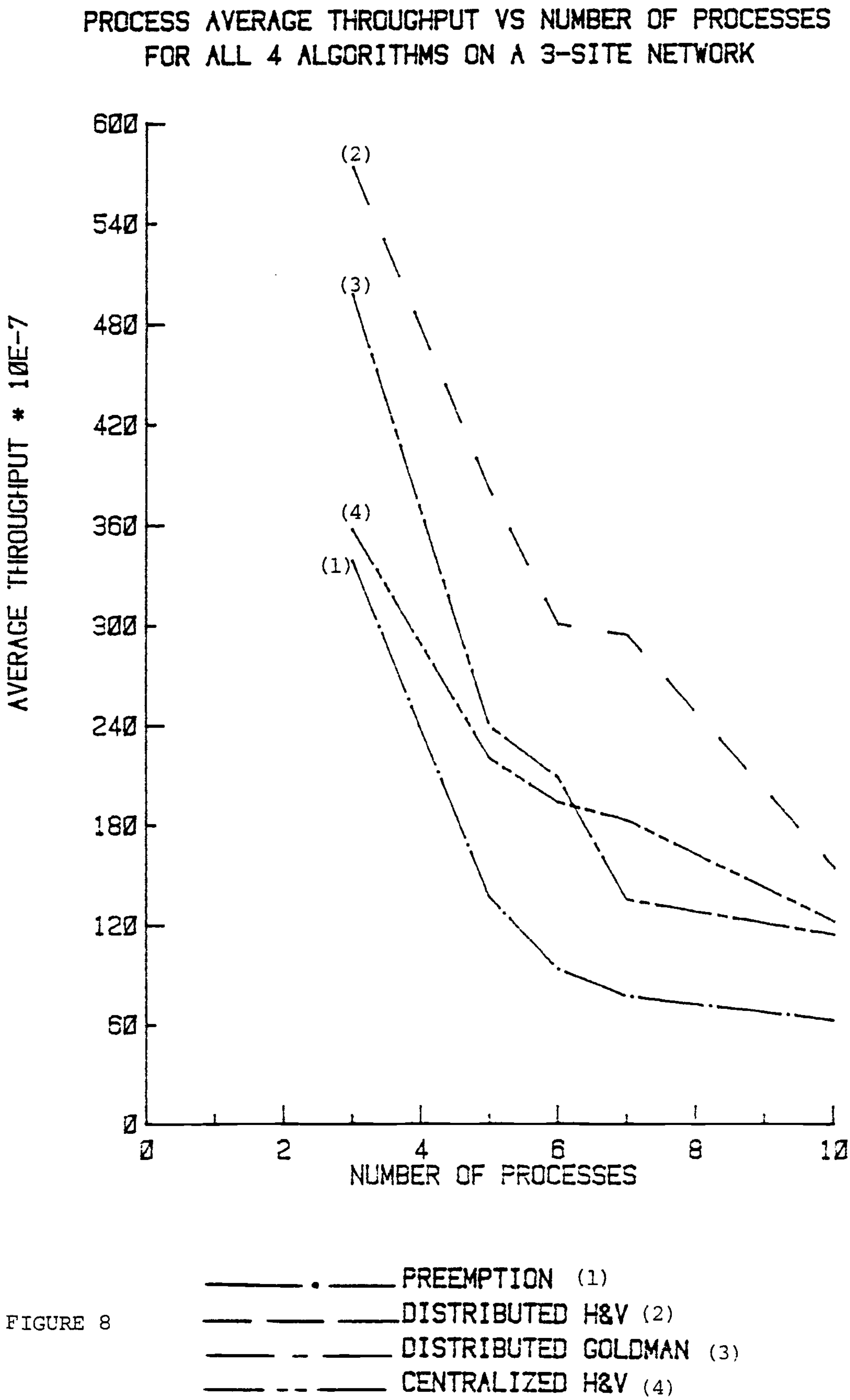

PROCESS AVERAGE THROUGHPUT VS NUMBER OF PROCESSES
FOR ALL 4 ALGORITHMS ON A 3-SITE NETWORK
AVERAGE THROUGHPUT * 10E-7
600
540
480
420
360
300
240
180
120
60
0
0
2
4
6
8
10
NUMBER OF PROCESSES
(1)
(2)
(3)
(4)
PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)

FIGURE 8

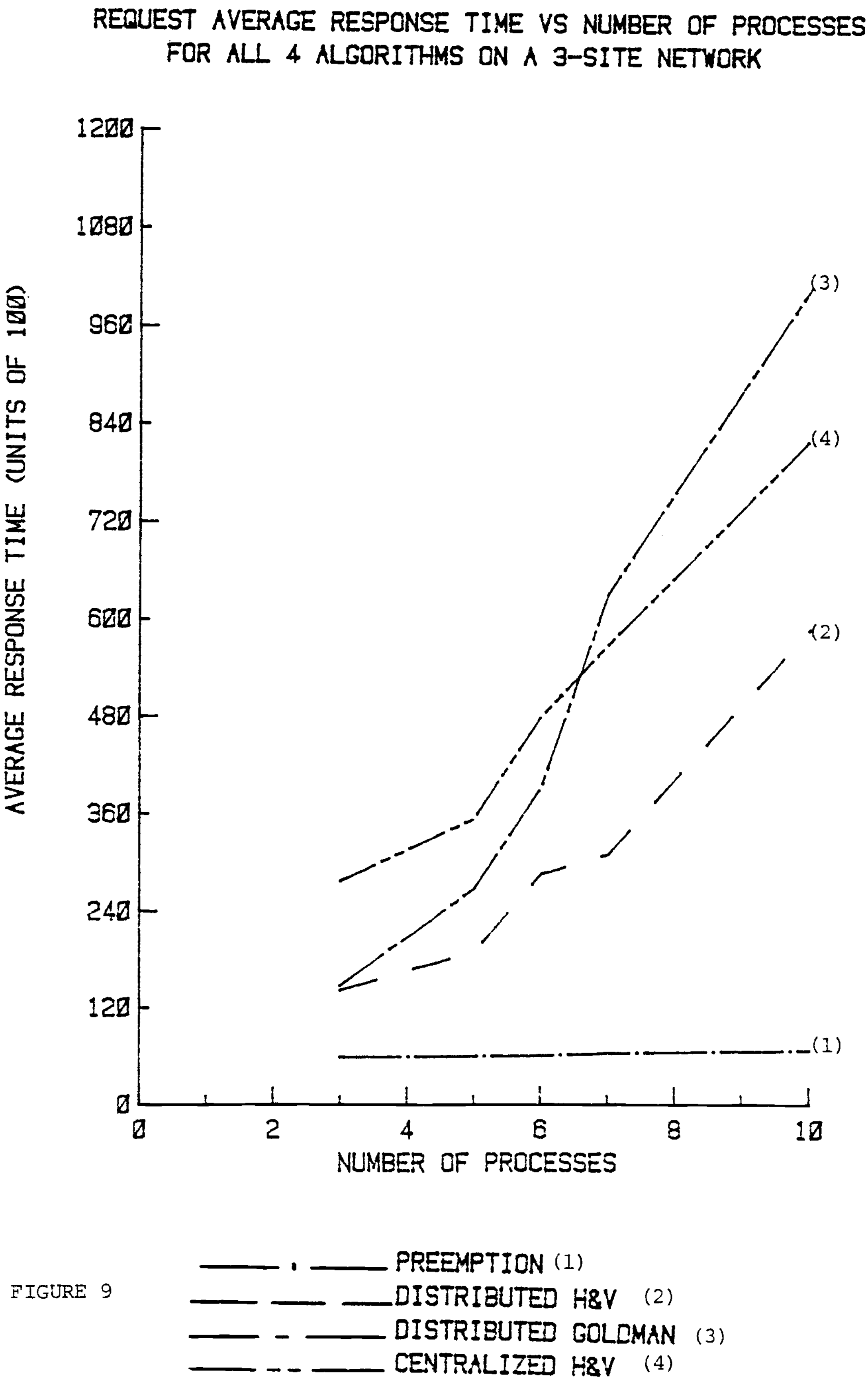

REQUEST AVERAGE RESPONSE TIME VS NUMBER OF PROCESSES
FOR ALL 4 ALGORITHMS ON A 3-SITE NETWORK
AVERAGE RESPONSE TIME (UNITS OF 100)
1200
1080
960
840
720
600
480
360
240
120
0
0
2
4
6
8
10
NUMBER OF PROCESSES
(1)
(2)
(3)
(4)
PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)

FIGURE 9

REQUEST AVERAGE THROUGHPUT VS NUMBER OF PROCESSES
FOR ALL 4 ALGORITHMS ON A 3-SITE NETWORK

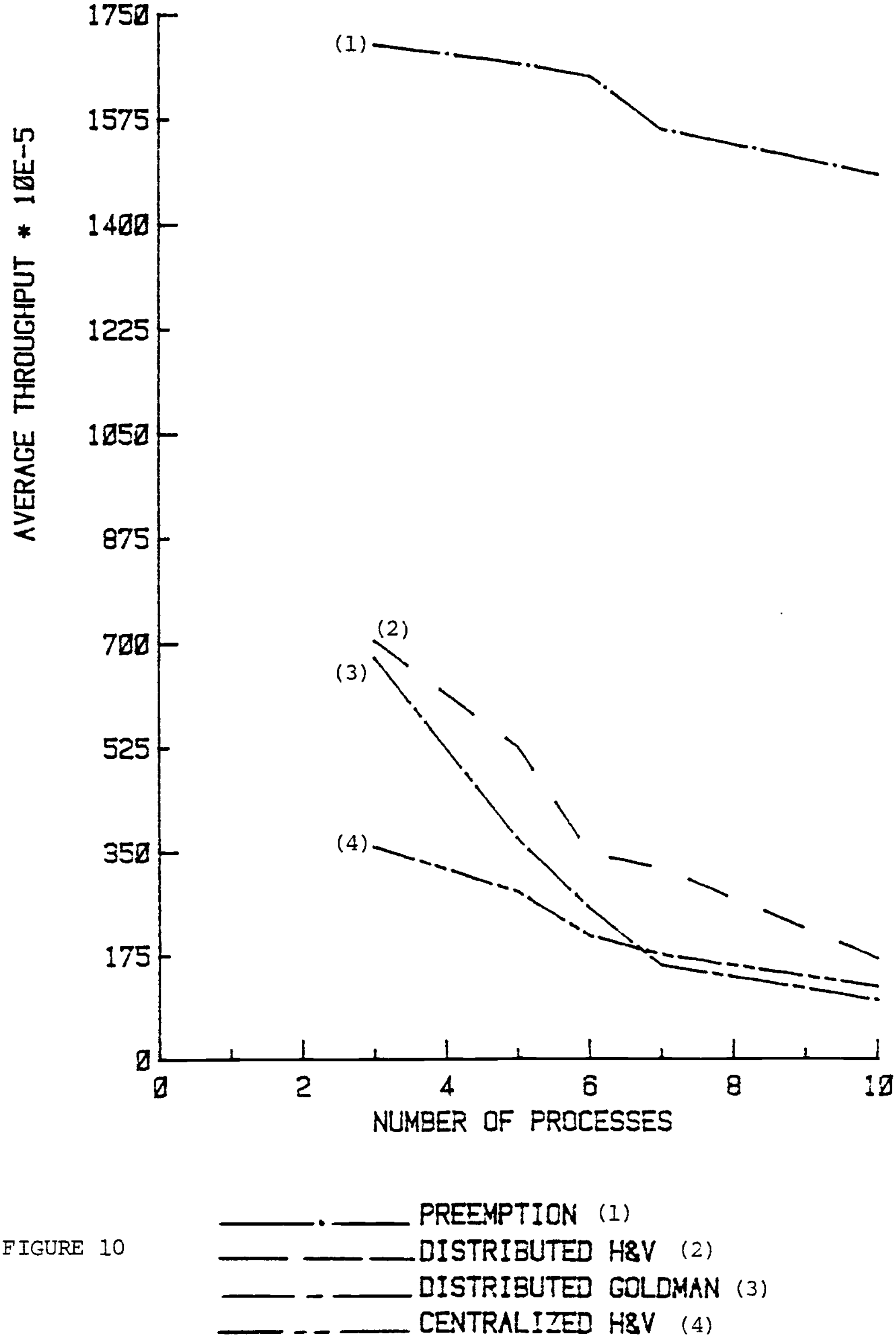


FIGURE 10

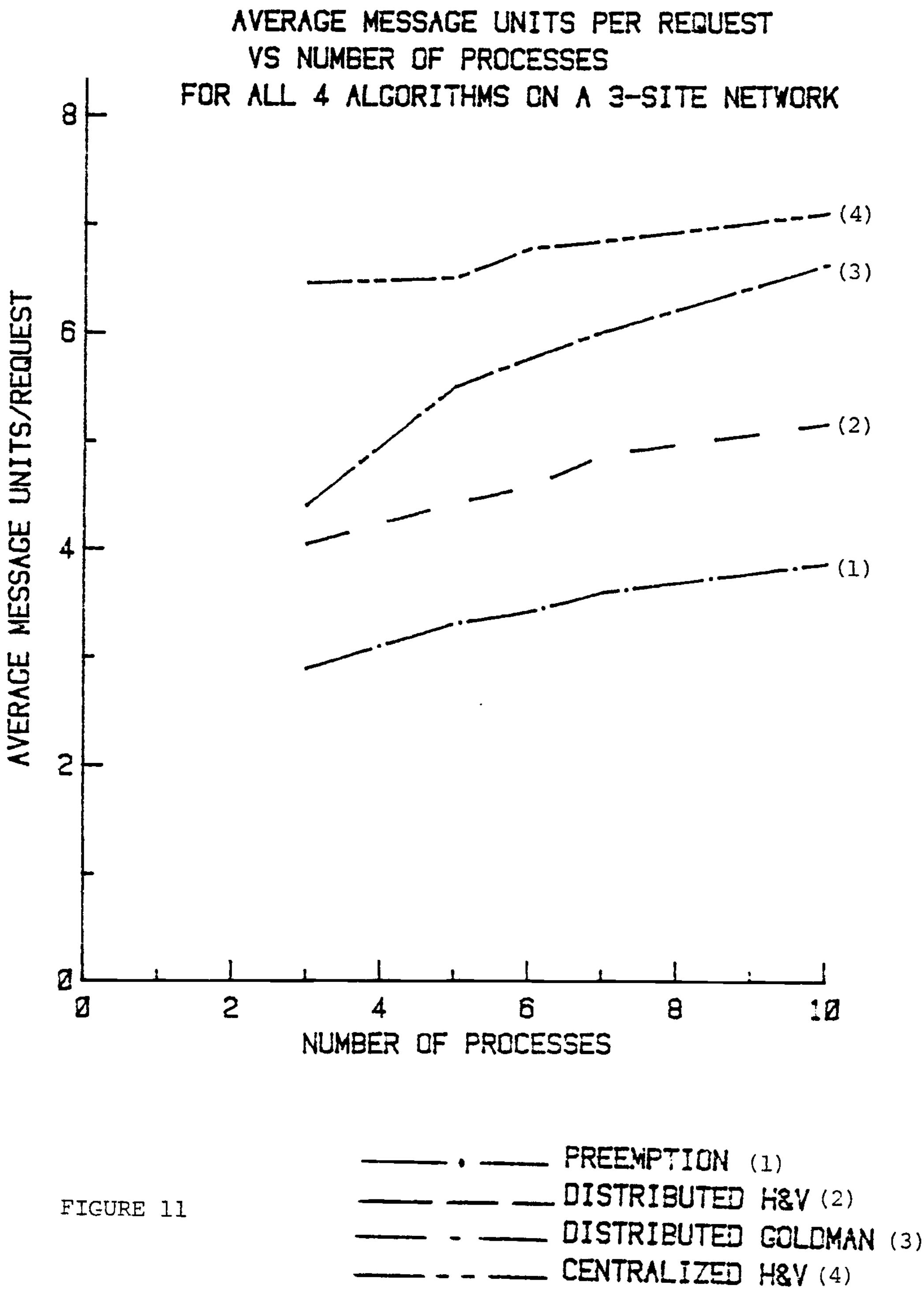

AVERAGE MESSAGE UNITS PER REQUEST
VS NUMBER OF PROCESSES
FOR ALL 4 ALGORITHMS ON A 3-SITE NETWORK
AVERAGE MESSAGE UNITS/REQUEST
NUMBER OF PROCESSES
0
2
4
6
8
10
(1)
(2)
(3)
(4)
PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)

FIGURE 11

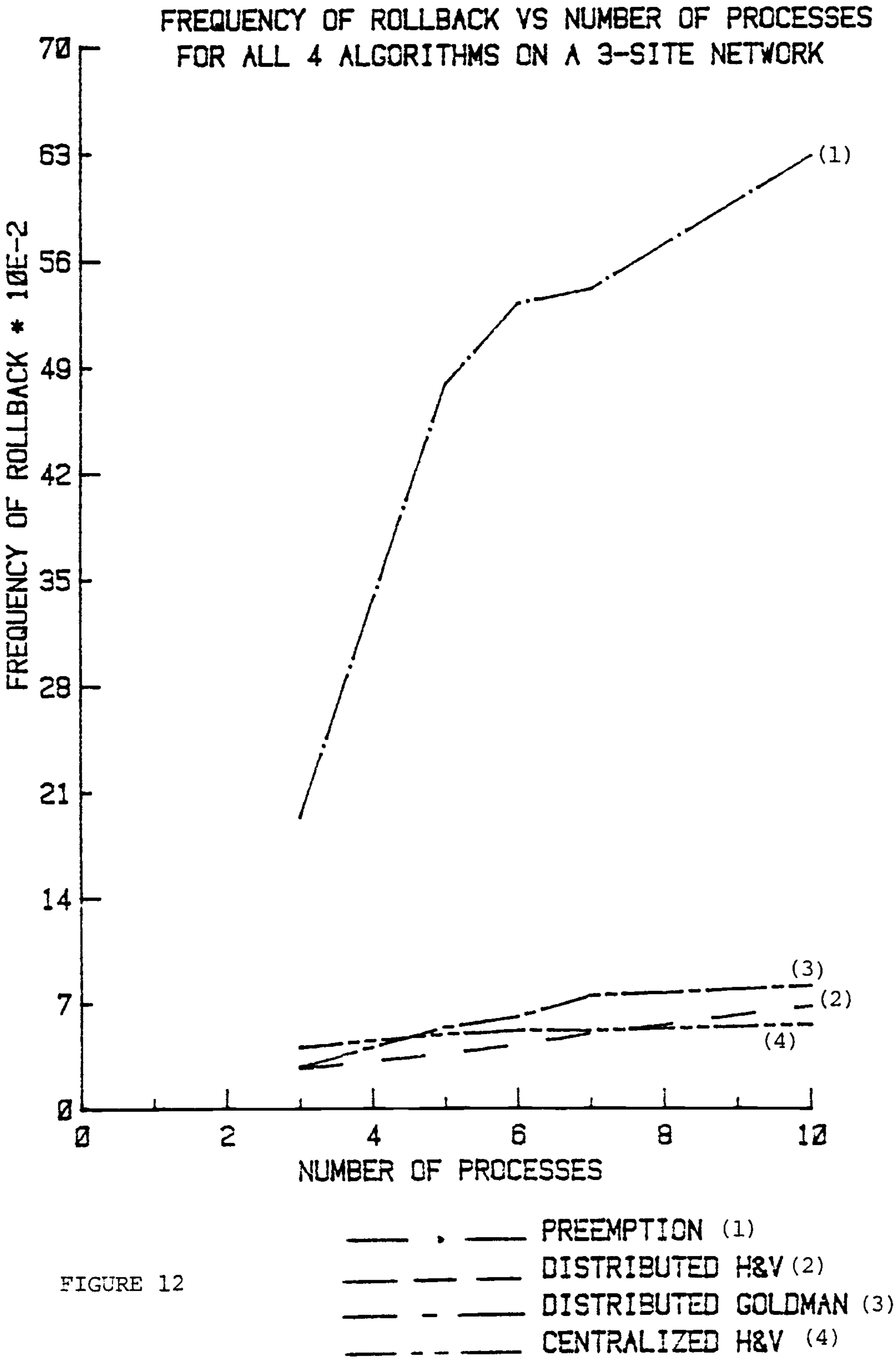

FREQUENCY OF ROLLBACK VS NUMBER OF PROCESSES
FOR ALL 4 ALGORITHMS ON A 3-SITE NETWORK
FREQUENCY OF ROLLBACK * 10E-2
0
7
14
21
28
35
42
49
56
63
70
(1)
(2)
(3)
(4)
0
2
4
6
8
10
NUMBER OF PROCESSES
PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)

FIGURE 12

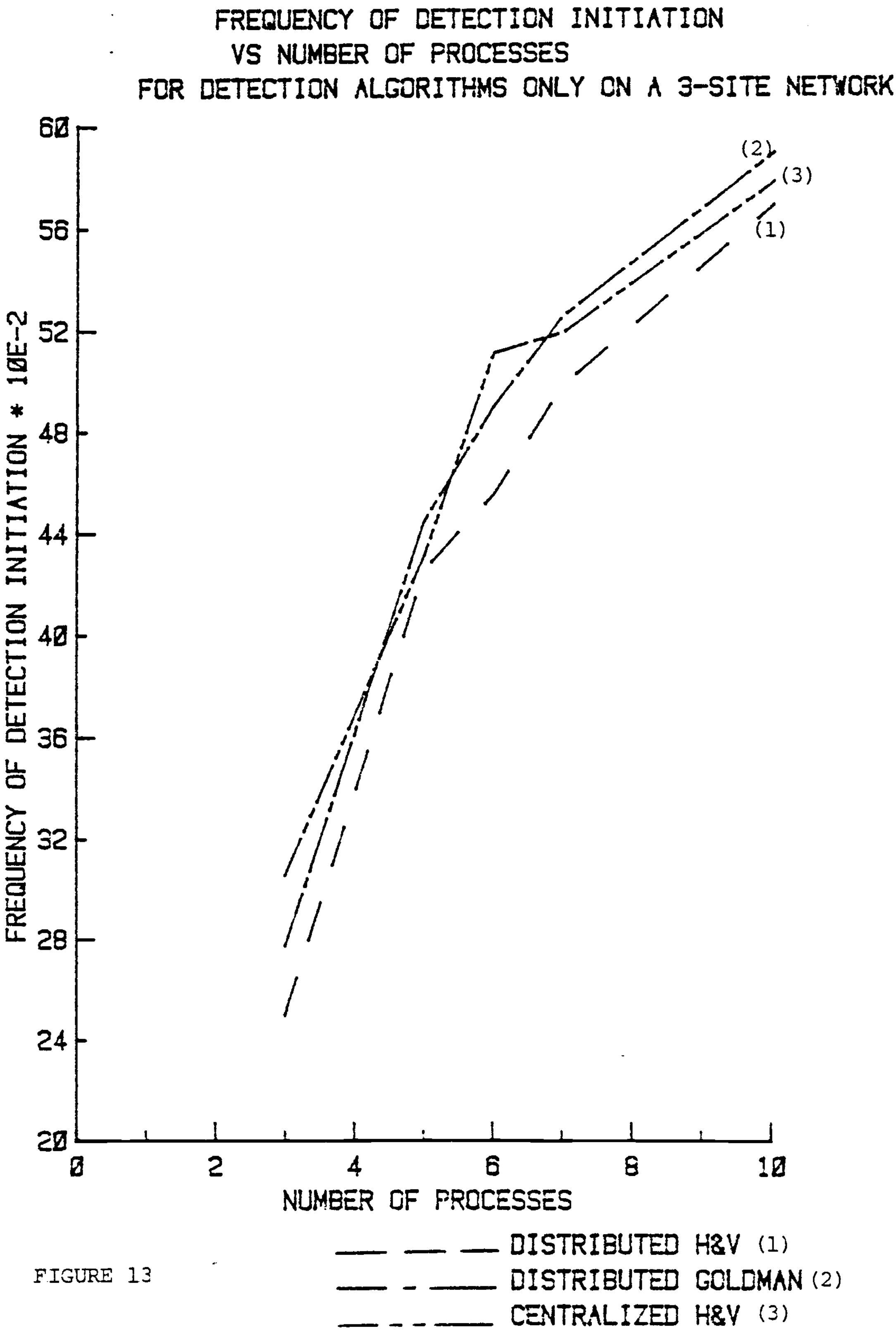

FREQUENCY OF DETECTION INITIATION
VS NUMBER OF PROCESSES
FOR DETECTION ALGORITHMS ONLY ON A 3-SITE NETWORK
FREQUENCY OF DETECTION INITIATION * 10E-2
60
56
52
48
44
40
36
32
28
24
20
0
2
4
6
8
10
NUMBER OF PROCESSES
(1)
(2)
(3)
DISTRIBUTED H&V (1)
DISTRIBUTED GOLDMAN (2)
CENTRALIZED H&V (3)

FIGURE 13

Goldman's algorithm has the second highest message units per request, while Distributed H&V has the lowest, among the three detection algorithms. Remember that in Goldman's algorithm duplicate copies of OBPL are made whenever a resource is held under shared access. Also, it is possible for the same detection message to go around the network more than once, whereas in Distributed H&V, each detection invocation gives rise to only one detection message unit. The message can go around the network only once. Therefore, the Distributed H&V algorithm has a lower average message units per request than Goldman's algorithm. From the graph, Figure 11, this trend is bound to continue for numbers of processes greater than ten.

Table 12 and Figure 12 reveal that for smaller numbers of processes in the network the frequency of deadlock occurrence is highest for Centralized H&V. But as the numbers of processes increase the frequency of deadlock occurrence is least when Centralized H&V is used. Distributed Goldman gives the highest frequency of deadlock among the three detection algorithms, for higher number of processes. Centralized H&V is more attractive, in terms of the frequency of deadlock, because the tables used in the detection algorithm are centralized. Therefore, when a deadlock is detected, it is removed much faster than it is removed in a distributed control environment.

The problem of false deadlock has been mentioned by many researchers [27]. The study performed in this dissertation supports the fact that delayed table updates or graph updates cause more false deadlock in a distributed control environment than the running time of the detection algorithm at each site. Goldman's algorithm results in a higher frequency of deadlock than Distributed H&V because of higher messages in the network. Distributed H&V reports deadlock only once. It is possible for a request to cause more than one cycle in the Process-Resource graph. Distributed H&V will terminate immediately the first cycle is detected and a message is sent directly to the site that initiated the detection. But since any site can detect a deadlock in Goldman's algorithm, it is possible for the same deadlock situation to be reported more than once to the site where the requested resource resides. The latter implies more overhead, and therefore the possibility of more false deadlock. Deadlock removal is, therefore, faster with Distributed H&V than with Goldman's algorithm. Therefore, the frequency of deadlock occurrence depends on how fast a deadlock is detected and removed. From the experiment we conclude that deadlock removal in Distributed H&V is faster than that in Distributed Goldman. The results also confirm the notion that for fixed numbers of resources, the frequency of deadlock increases as the numbers of processes increase, since more processes now compete for the same numbers of resources.

Also from Figure 13 and Table 13 it may be noticed that the frequency of detection initiation is lowest for Distributed H&V. The same reasoning for frequency of deadlock may be applied here. As deadlocks are detected and removed, more resources become available for immediate allocation. Therefore, a detection algorithm that finds a deadlock and removes it much faster will result in more resources being free in the network. So the Distributed H&V, once again, appears to be a better algorithm than Goldman's.

A higher frequency of deadlock and more messages in the network will result in a slower response time. Tables 7 and 9 and Figures 7 and 9 confirm this fact; although, from the results it is the higher amount of rollback that contributes more to a poor process response time, as true in Preemption.

Figure 7 shows that the Distributed H&V algorithm has the lowest process response time than any of the other three algorithms. This translates into a higher process throughput as

Figure 8 shows. The better performance of the Distributed H&V than Goldman's algorithm is not surprising since the Distributed H&V produces less messages in the network. Also as mentioned earlier, in the Distributed H&V, a detection message goes around the network at most once, while in Goldman's algorithm, a detection message can go around the network more than once. Also, the better performance of the Distributed H&V is due to the fact that it has a lower frequency of rollback.

Figures 9 and 10 give the performance of the algorithms with respect to individual resource request. Once again the Distributed H&V algorithm gives a lower request response time and a higher request throughput than Goldman's algorithm and Centralized H&V.

6.2.2 Comparison of the Algorithms' Performance for Varying Numbers of Sites

To evaluate the algorithms' performances for varying numbers of sites, the simulation models were run for networks of 3, 5, 8, 10 and 12 sites. Tables 14 through 20 present the results and Figures 14 through 20 show graphs of some of the values in the tables. Figures 14 and 16 plot graphs of process average response time and request average response time, respectively, versus numbers of sites, while Figures 15 and 17 show average throughput versus numbers of sites. Figure 18 presents the average message units per request. Figures 19 and 20 give the graphs of frequency of rollback and deadlock detection initiation, respectively.

From Figure 19, we see that the frequency of rollback for preemption is extremely very high. And this worsens as the numbers of sites increase. The high frequency of rollback translates into a very high process average response time, Figure 14, for the preemption technique. As in the three-site experiment,

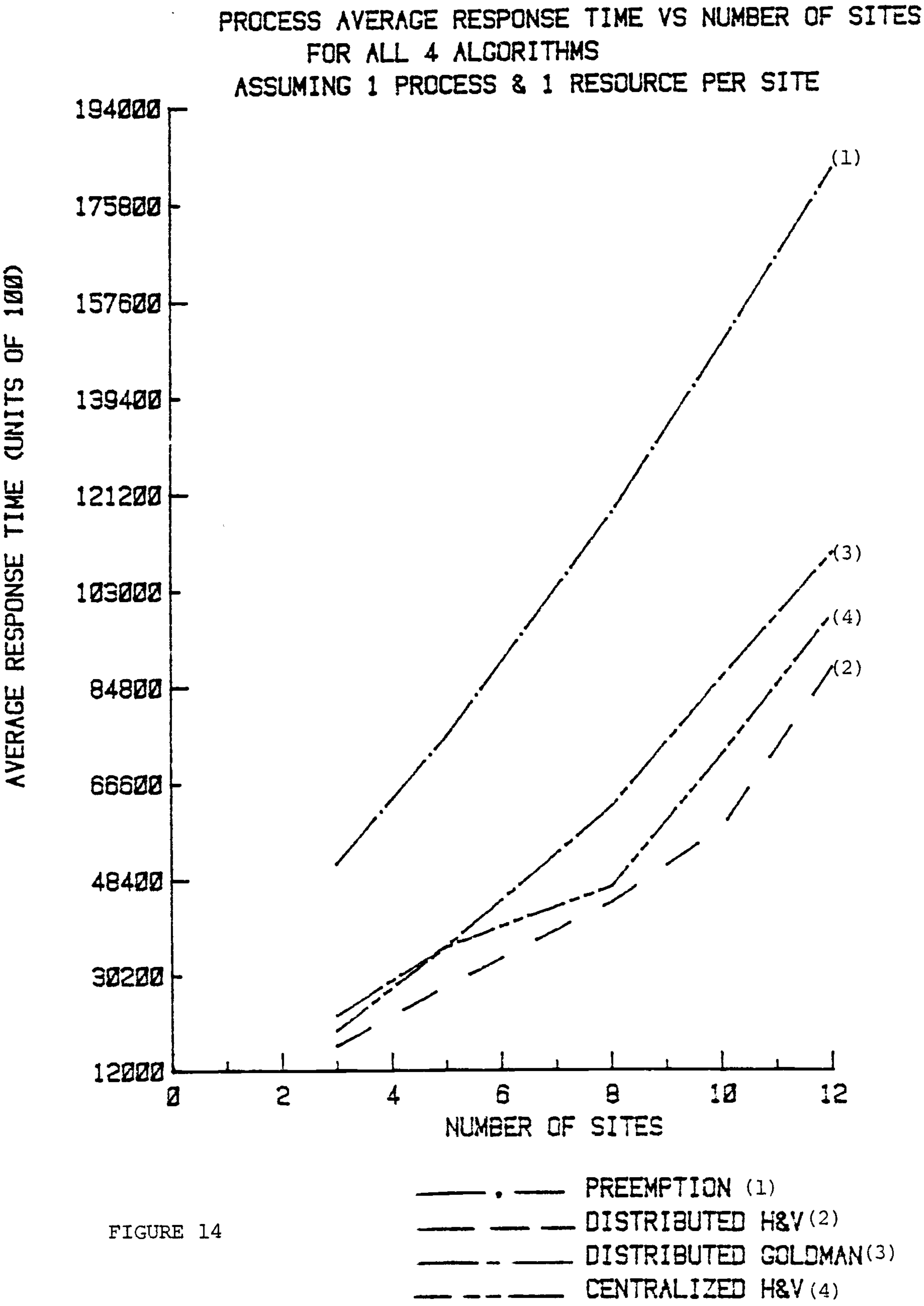

PROCESS AVERAGE RESPONSE TIME VS NUMBER OF SITES
FOR ALL 4 ALGORITHMS
ASSUMING 1 PROCESS & 1 RESOURCE PER SITE
AVERAGE RESPONSE TIME (UNITS OF 100)
194000
175800
157600
139400
121200
103000
84800
66600
48400
30200
12000
0
2
4
6
8
10
12
NUMBER OF SITES
(1)
(3)
(4)
(2)
PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)

FIGURE 14

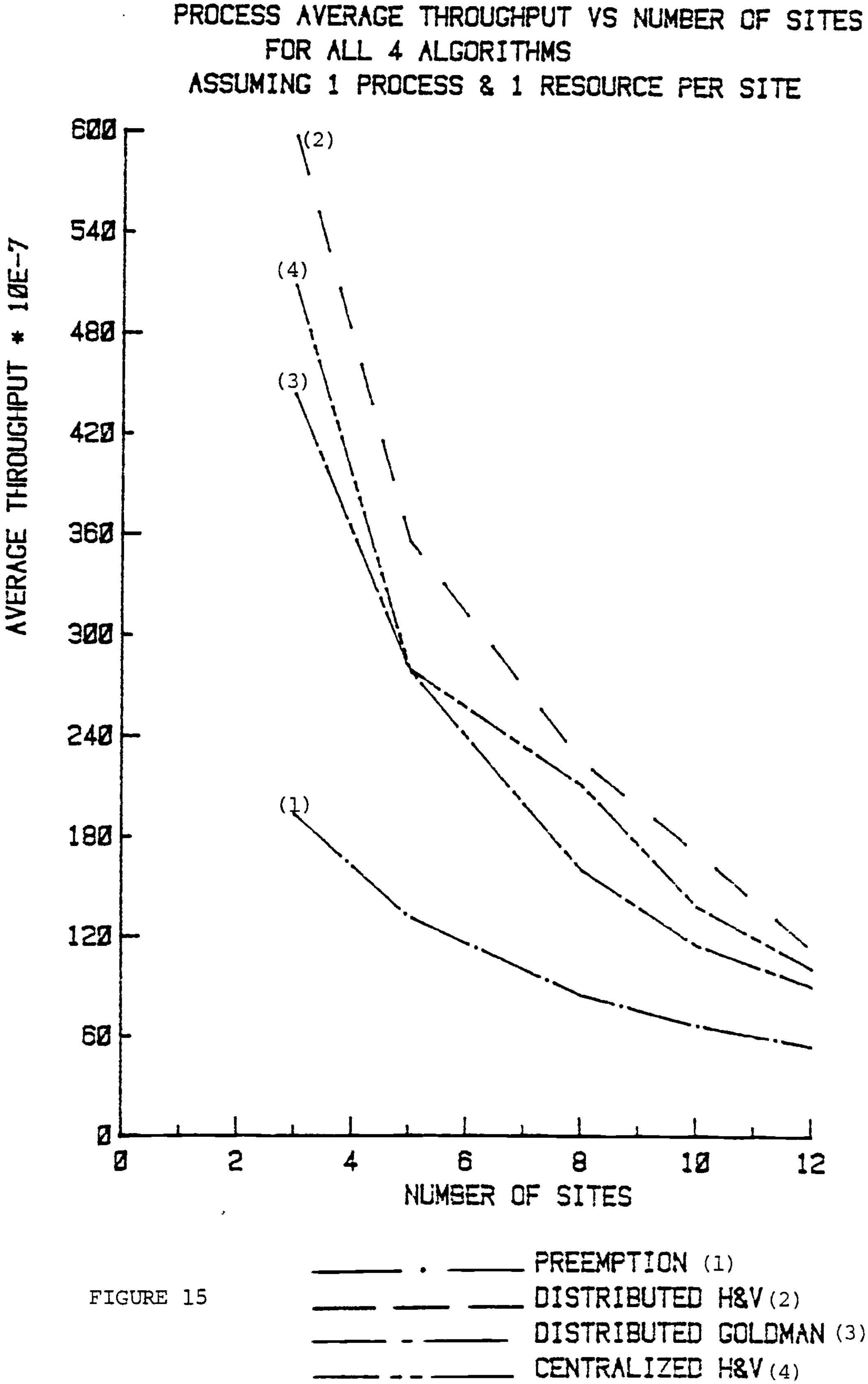
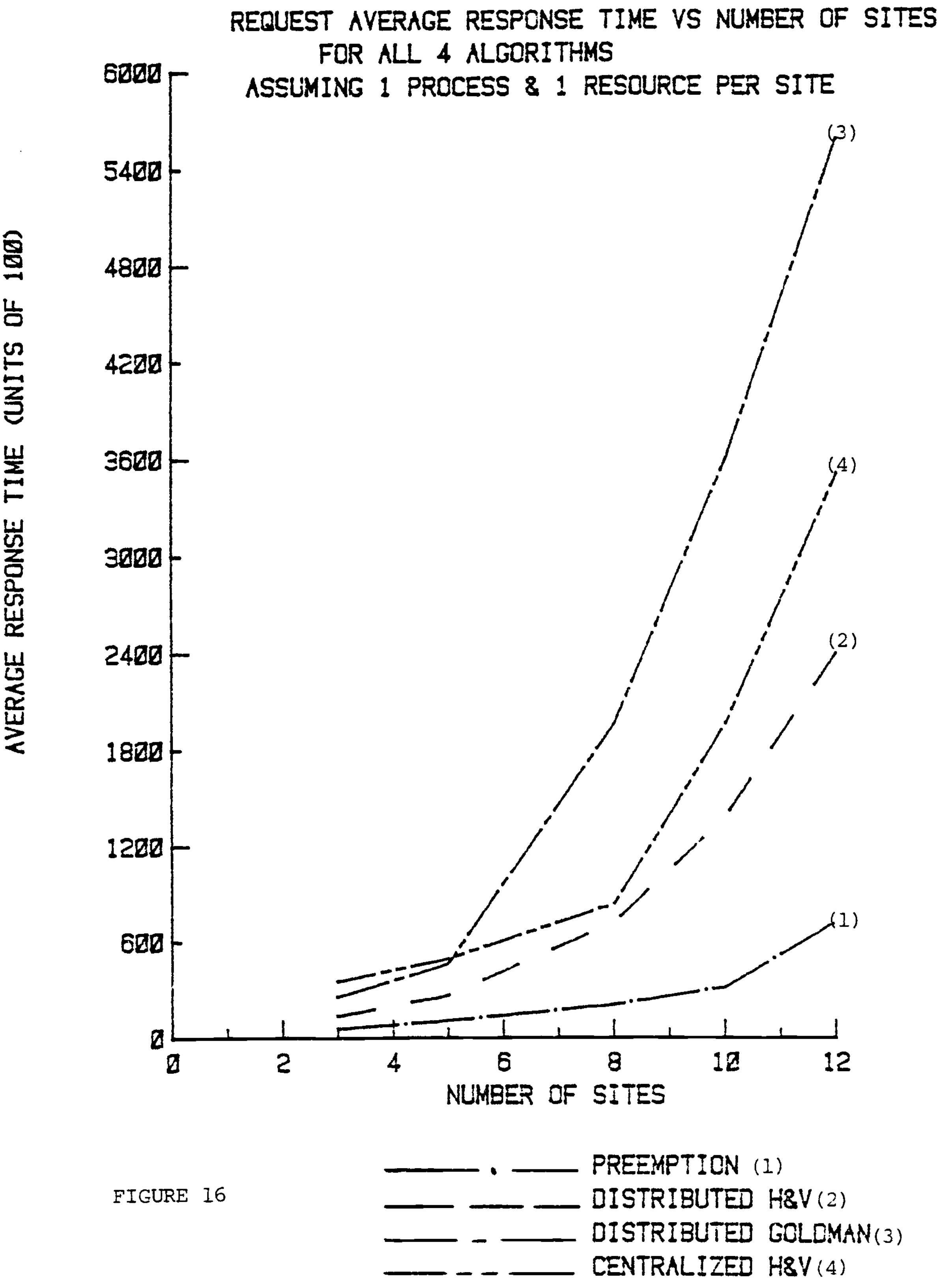

PROCESS AVERAGE THROUGHPUT VS NUMBER OF SITES
FOR ALL 4 ALGORITHMS
ASSUMING 1 PROCESS & 1 RESOURCE PER SITE
AVERAGE THROUGHPUT * 10E-7
0
60
120
180
240
300
360
420
480
540
600
(1)
(2)
(3)
(4)
0
2
4
6
8
10
12
NUMBER OF SITES
PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)

FIGURE 15



REQUEST AVERAGE RESPONSE TIME VS NUMBER OF SITES
FOR ALL 4 ALGORITHMS
ASSUMING 1 PROCESS & 1 RESOURCE PER SITE
AVERAGE RESPONSE TIME (UNITS OF 100)
6000
5400
4800
4200
3600
3000
2400
1800
1200
600
0
0
2
4
6
8
10
12
NUMBER OF SITES
(1)
(2)
(3)
(4)
PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)

FIGURE 16

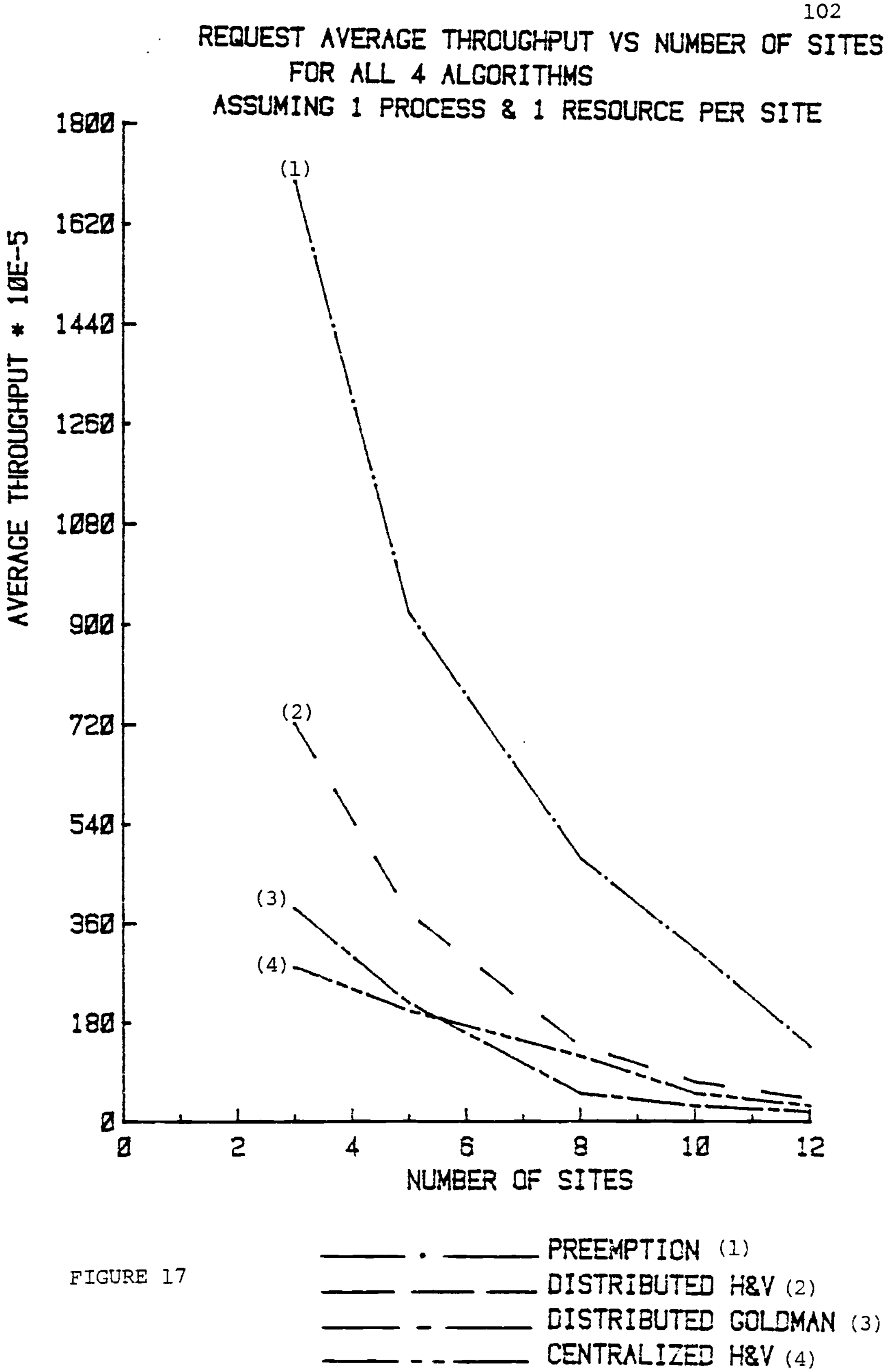

REQUEST AVERAGE THROUGHPUT VS NUMBER OF SITES
FOR ALL 4 ALGORITHMS
ASSUMING 1 PROCESS & 1 RESOURCE PER SITE
AVERAGE THROUGHPUT * 10E-5
1800
1620
1440
1260
1080
900
720
540
360
180
0
(1)
(2)
(3)
(4)
0
2
4
6
8
10
12
NUMBER OF SITES
PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)

FIGURE 17

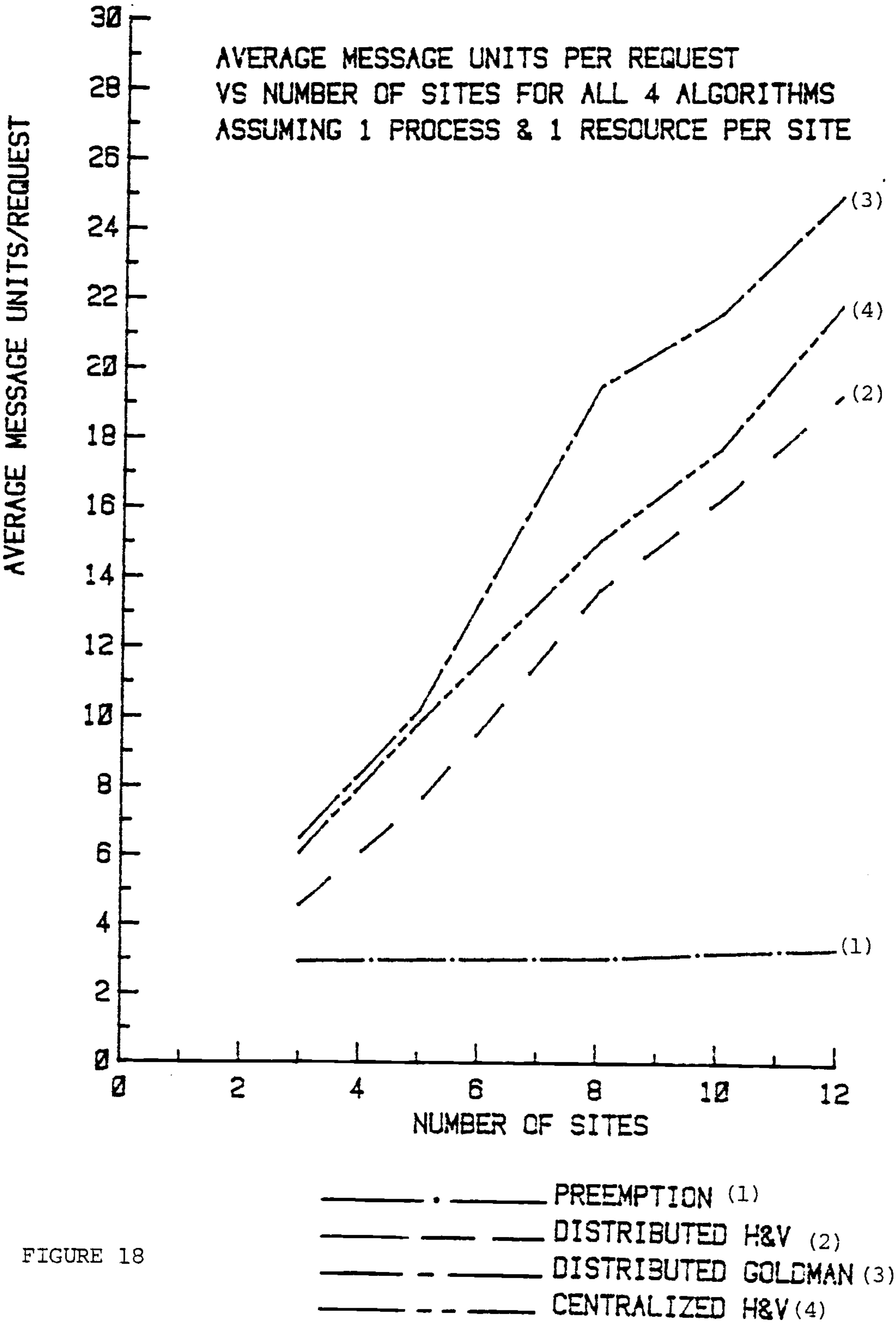

AVERAGE MESSAGE UNITS PER REQUEST
VS NUMBER OF SITES FOR ALL 4 ALGORITHMS
ASSUMING 1 PROCESS & 1 RESOURCE PER SITE
AVERAGE MESSAGE UNITS/REQUEST
30
28
26
24
22
20
18
16
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
NUMBER OF SITES
(1)
(2)
(3)
(4)
PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)

FIGURE 18

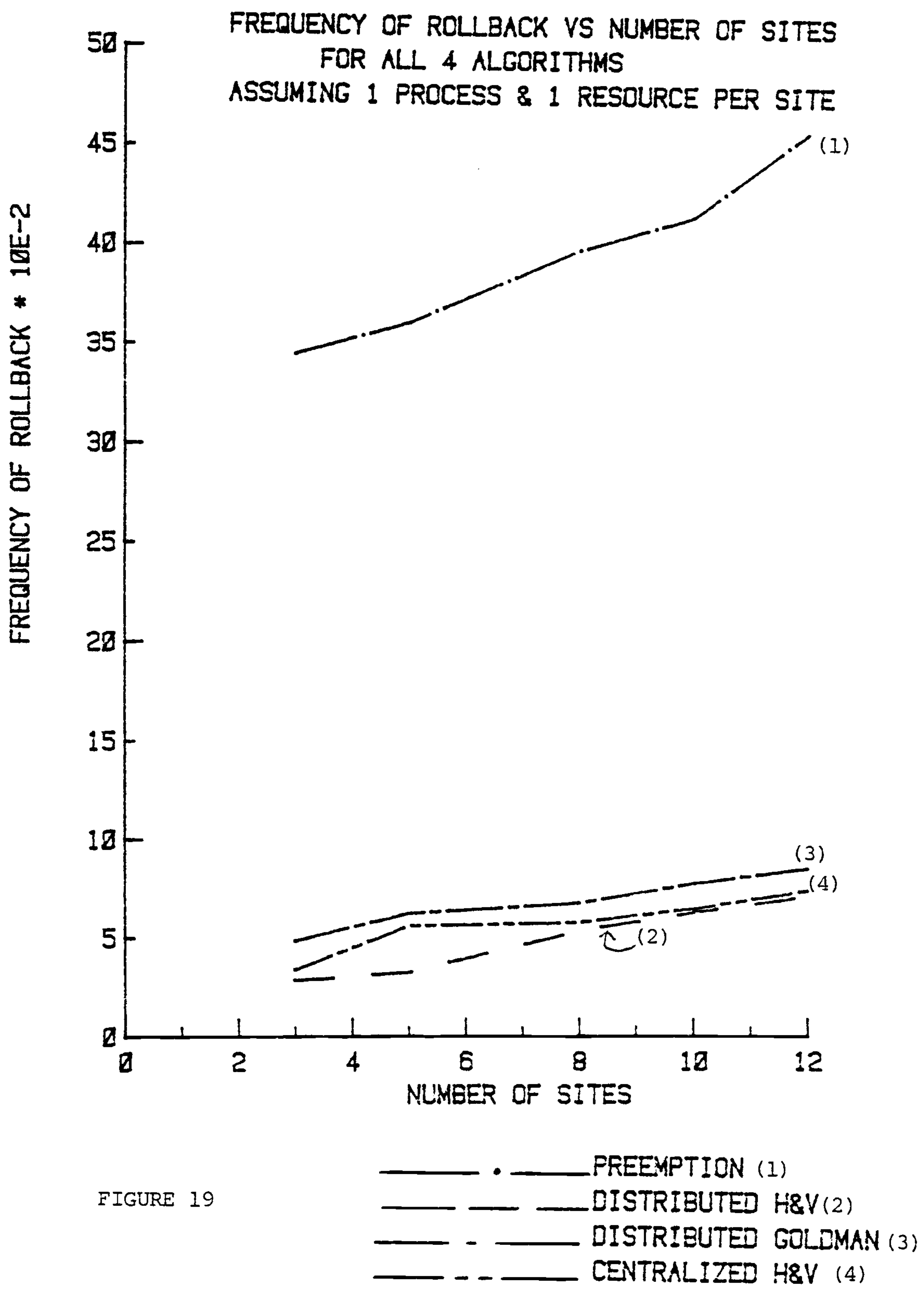

FREQUENCY OF ROLLBACK VS NUMBER OF SITES
FOR ALL 4 ALGORITHMS
ASSUMING 1 PROCESS & 1 RESOURCE PER SITE
FREQUENCY OF ROLLBACK * 10E-2
50
45
40
35
30
25
20
15
10
5
0
0
2
4
6
8
10
12
NUMBER OF SITES
(1)
(3)
(4)
(2)
PREEMPTION (1)
DISTRIBUTED H&V(2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)

FIGURE 19

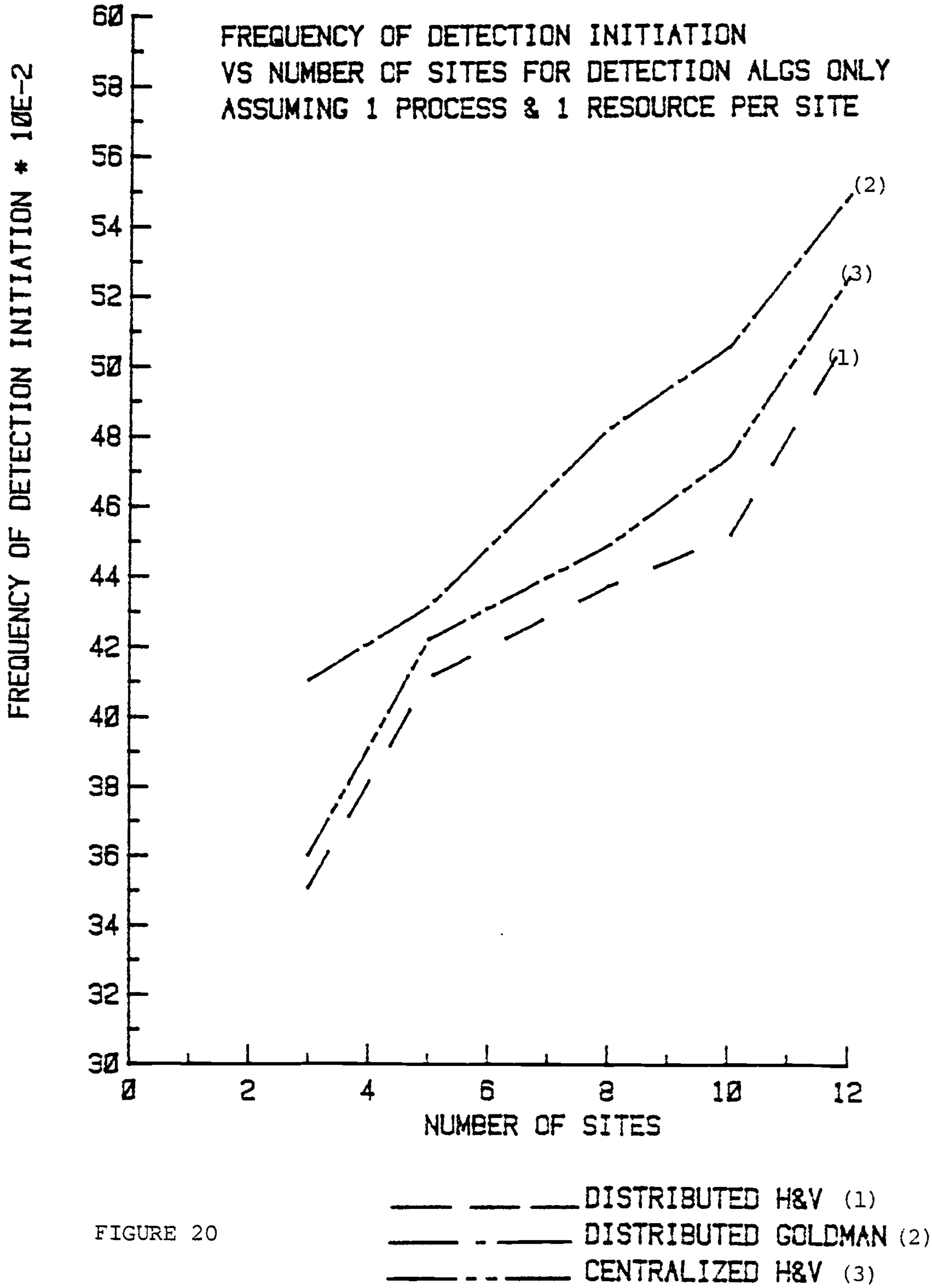

FREQUENCY OF DETECTION INITIATION
VS NUMBER OF SITES FOR DETECTION ALGS ONLY
ASSUMING 1 PROCESS & 1 RESOURCE PER SITE
FREQUENCY OF DETECTION INITIATION * 10E-2
NUMBER OF SITES
60
58
56
54
52
50
48
46
44
42
40
38
36
34
32
30
0
2
4
6
8
10
12
(1)
(2)
(3)
DISTRIBUTED H&V (1)
DISTRIBUTED GOLDMAN (2)
CENTRALIZED H&V (3)

FIGURE 20

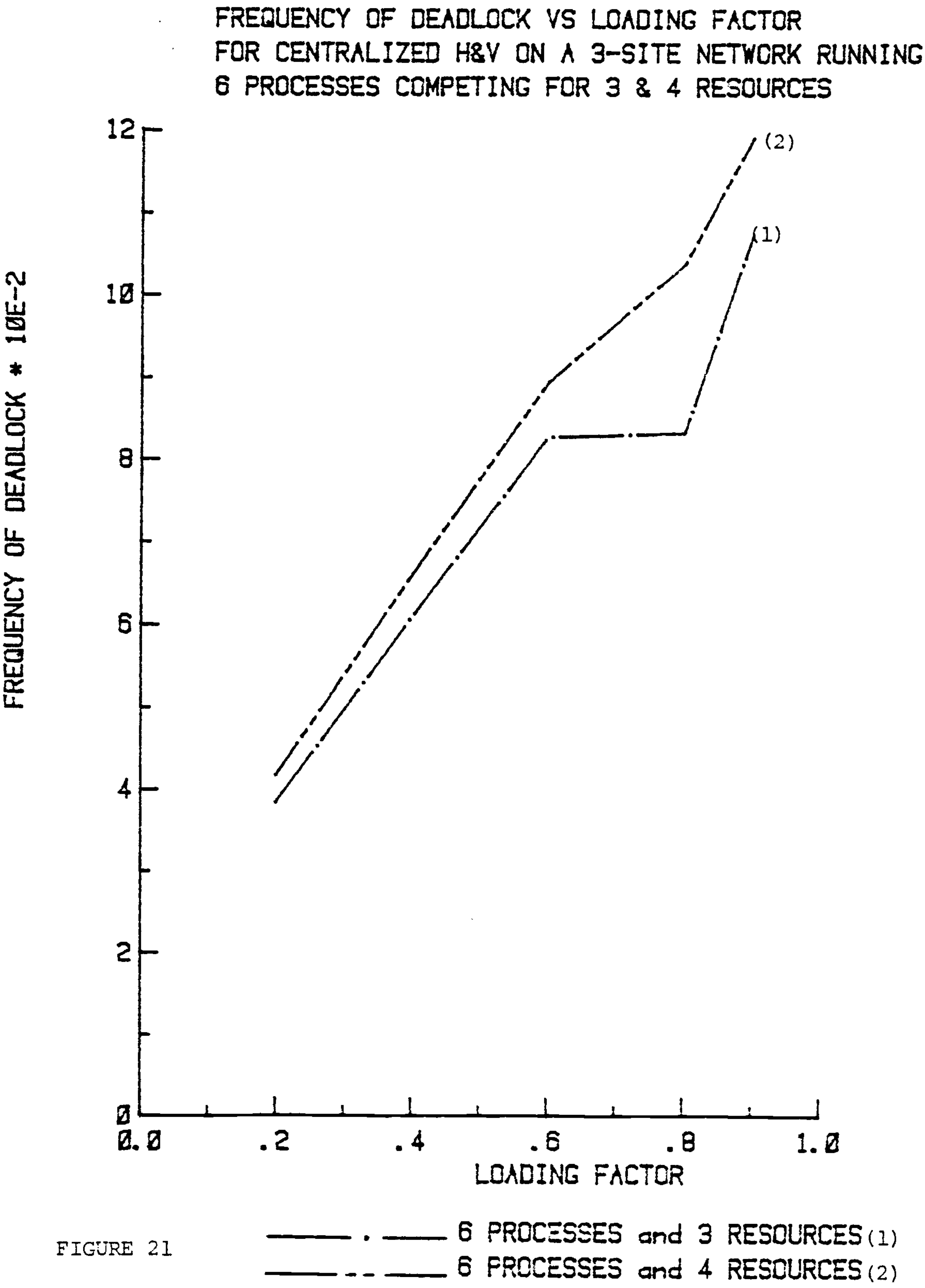

FREQUENCY OF DEADLOCK VS LOADING FACTOR
FOR CENTRALIZED H&V ON A 3-SITE NETWORK RUNNING
6 PROCESSES COMPETING FOR 3 & 4 RESOURCES
FREQUENCY OF DEADLOCK * 10E-2
12
10
8
6
4
2
0
0.0
.2
.4
.6
.8
1.0
LOADING FACTOR
(1)
(2)
6 PROCESSES and 3 RESOURCES (1)
6 PROCESSES and 4 RESOURCES (2)

FIGURE 21

the request average response time is very low, as no detection algorithm is involved. The high process response time means a very low throughput for the preemption method, as Figure 15 indicates, whereas the request average throughput is high, Figure 17. From these we conclude that preemption technique has the worst performance among all the four algorithms considered. As in the three-site experiment we see that it is the frequency of process rollback that causes a particular deadlock resolution technique to perform very poorly. Figure 18 shows that Preemption has the lowest average message units per request. But this is not enough to offset the high frequency of rollback. Preemption would therefore be unacceptable in practical environment, especially in distributed database system.

Figure 18 shows that Goldman's algorithm has the highest average message units per request. This is followed by Centralized H&V, while Distributed H&V has the lowest among the detection algorithms. Therefore, as the numbers of sites increase, Goldman's algorithm would generate more messages than any of the other algorithms.

In the three-site experiment, discussed in Section 6.2.1, the Centralized H&V had the highest amount of messages. This was because there were two resources per site, so the probability that a local process would request for a local resource was higher. This kept the average message units per request for the distributed control experiments lower than the centralized. Now that there is only one resource per site, the average message units per request for the Centralized H&V is lower than that of Goldman's algorithm but still higher than that of the Distributed H&V.

From Table 19 and Figure 19 we see that the Distributed H&V has the lowest frequency of deadlock, while Goldman's algorithm has the highest of the three detection algorithms. For higher numbers of sites we see that the frequency of deadlock for the Centralized H&V compares very well with that of the Distributed H&V, although it has a higher average message units per request. Again, as in Figure 12, this is because the detection tables are centralized, and therefore deadlock is detected and removed faster.

Figures 14 and 16 show that Goldman's algorithm has the highest process and request average response times, respectively, than the Distributed and Centralized H&V. The relatively poor performance of Goldman's algorithm should not be surprising, since it has the highest average messages, the highest frequency of deadlock and the highest frequency of detection initiation. The average throughput measurements of Figures 15 and 17 are another way of looking at the performances of these algorithms. Distributed Goldman's algorithm gives relatively the lowest process throughput while the Distributed H&V gives the highest average throughput. The Centralized H&V performs better than Goldman's algorithm.

The figures suggest that, as the numbers of sites increase, the Distributed H&V will continue to perform better than Goldman's algorithm. From the results, we have seen that the performance of a particular algorithm depends very much on how fast it detects and removes a deadlock. Also, the amount of messages the detection routine sends out contributes a lot to the network congestion. Thirdly, if there is a high probability that the detection algorithm will go around the network more than once, as is the case in Goldman's algorithm, the network performance will be relatively very poor. Hence, Goldman's algorithm performs poorer than the Distributed H&V algorithm.

6.2.3 Frequency of Deadlock for Varying Loading Factor

To measure the frequency of deadlock for varying loading factor (rate of resource request/rate of resource release) a number of simulation runs were carried out on a three-site network, for loading factors of 0.2, 0.4, 0.6, 0.8 and 0.9. Centralized H&V algorithm was used to detect deadlock. The number of processes running on the network was fixed at 6, two processes per site. The experiment was done for 3 and 4 resources. In each run, each process was allowed a maximum of 1000 requests. All assumptions that applied to the Centralized H&V model, discussed ealier, also applied to this experiment.

Table 21 presents the results, and Figure 21 plots the information contained in the table. From the results, we notice that the frequency of deadlock is higher for the system with four resources. Intuitively the number of possible cycles with six processes and four resources is more than that with six processes and three resources. So as the system stabilizes, the number of deadlock occurrences in the latter system would be more than that in the six processes, three resources system. This is confirmed by the results obtained here.

Also in both systems the frequency of deadlock increases as the loading factor increases. This is not surprising, since at higher loading factor more resource requests are sent to the controller site. This increases the congestion in the network thereby slowing down the system. Also, the queue of messages at the controller site will increase. Resource release messages will, therefore, take longer time to get processed by the controller. In practical environment, it may be a good design strategy for the resource manager to give higher priority to resource release messages. Giving higher priority to resource release messages may decrease the frequency of false deadlock in the system, both in centralized and distributed control environment.

VII. QUEUEING ANALYSIS OF THE DISTRIBUTED HORIZONTAL AND VERTICAL ALGORITHM

Queueing network models have been applied by many researchers to the analysis and prediction of computer system performance. The main motivation for performance evaluation of complex systems, such as distributed computer systems, is the fact that such systems are too complex for any human to fully understand. Because of this, researchers have resorted to using simulation models or analytical approaches to study the behavior of such systems.

Advances in queueing theory have provided adequate mathematical tools to attack simplified models of computer systems analytically. However, the more complex a system is, the more difficult it is to provide accurate and precise analytical model. For such systems, one technique for obtaining reasonably accurate performance values is by simulating the model and then using the simulation results along with known mathematical formulas and regression techniques to obtain approximate analytical model.

In this chapter we shall use the simulation results of the distributed Horizontal and Vertical algorithm and basic multi-server M/M/m queueing model to obtain an approximate analytical model called the M/M/z model. A similar approach was used by Jafari [43] to study simulation results obtained for a new loop architecture for a distributed computer network. A detailed study of an M/M/m queueing model is given by Kleinrock [48]. Only relevant formulas will be repeated here.

The fundamental equation relating the average time a process spends in a system (response time) to the average service time and the average waiting time is given by the following [48]:

$$T = \bar{x} + W \tag{1}$$

where T = response time

$\bar{x}$ = average service time

W = average waiting time (queueing time)

The average waiting time is given by

$$W = \frac{P_m}{m\mu(1-\rho)} \tag{2}$$

where P_m = probability that all m servers are busy

m = number of servers

$\mu = \frac{1}{\bar{x}}$ = service rate

ρ = utilization of the system

The utilization of the system is given by

$$\rho = \frac{\lambda}{m\mu} = \frac{\lambda\bar{x}}{m} \tag{3}$$

λ = the arrival rate.

The condition for ergodicity, necessary for equilibrium probabilities to exist is met whenever $\rho = \frac{\lambda}{m\mu} < 1$.

The probability that an arriving customer finds k customers in the system is given by

$$P_k = \begin{cases} P_0 \dfrac{(m\rho)^k}{k!} & \text{for } k \leqslant m \\ \\ P_0 \dfrac{(\rho)^k\, m^m}{m!} & \text{for } k > m \end{cases} \tag{4}$$

where

$$P_0 = \left[\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!\,(1-\rho)}\right]^{-1} \tag{5}$$

From these, the expression for P_m is given by

$$P_m = P_0 \frac{(m\rho)^m}{m!\,(1-\rho)} \tag{6}$$

$$= \frac{\left(\frac{(m\rho)^m}{m!}\right)\left(\frac{1}{1-\rho}\right)}{\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \left(\frac{(m\rho)^m}{m!}\right)\left(\frac{1}{1-\rho}\right)} \tag{7}$$

Therefore, equation (1) becomes

$$T = \frac{1}{\mu} + \frac{P_m}{m\mu\,(1-\rho)} \tag{8}$$

Like the M/M/m model, the M/M/z model is a multiserver queueing model, where z stands for the number of servers. But unlike the M/M/m model, z indicates the mean effective number of servers in the system, which is not necessarily an integer number. The primary motivation for the M/M/z queueing model is the fact that in systems where the number of available servers is time-dependent, it is quite possible to end up with a non-integer average number of servers.

The M/M/z model for z values between one and two was developed by Jafari [43]. For the model, the equation for the average response time, equation (8), becomes

$$T = \frac{1}{\mu} + \frac{P_z}{z\mu(1-\rho)} \tag{9}$$

where

z = average number of servers including non-integer values

$$\rho = \frac{\lambda}{z\mu}$$

$$P_z = \frac{\dfrac{(z\rho)^z}{z!}\left(\dfrac{1}{1-\rho}\right)}{\displaystyle\sum_{k=0}^{z-1} \frac{(z\rho)^k}{k!} + \left(\frac{(z\rho)^z}{z!}\right)\left(\frac{1}{1-\rho}\right)} \tag{10}$$

Since z includes non-integer values, equation (10) is not in a correct form. But for z values between one and two, equation (10) can be approximated to [43]:

$$P_z \cong \frac{z\rho^z}{1 + (z-1)\rho} \tag{11}$$

Equation (11) is a good approximation of equation (10) since for integer values of z, one and two, P_z precisely agrees with P_m. For boundary values of ρ, example $\rho = 0$ and $\rho = 1$, P_z agrees precisely with P_m. Also, Jafari [43] gives plots of P_z for z values between one and two, which are reasonably located between the plots for z values one and two.

Therefore the response time equation (9) becomes:

$$T = \frac{1}{\mu} + \frac{\rho^z}{[1 + (z-1)\rho](1-\rho)\mu} \tag{12}$$

Jafari [43] used this equation to analyze his new loop architecture, since the simulation results he obtained suggested values of z between one and two.

7.1 M/M/z Queueing Model for the Distributed Horizontal and Vertical Algorithm

In this section we shall use the M/M/z queueing model to obtain a mathematical model for the request response time of the Distributed Horizontal and Vertical algorithm. Let us assume a network of N sites, n processes distributed throughout the network, and an average of m resources per site. Assume that all processes are statistically identical and independent. The service facilities are the resource managers located at each site. The resource managers will be modeled as one conceptual global service facility. We shall assume a Poisson rate of resource request by each process at a rate of λ requests per unit time. Therefore, there are n independent sources of requests for the network model as shown in Figure 22.

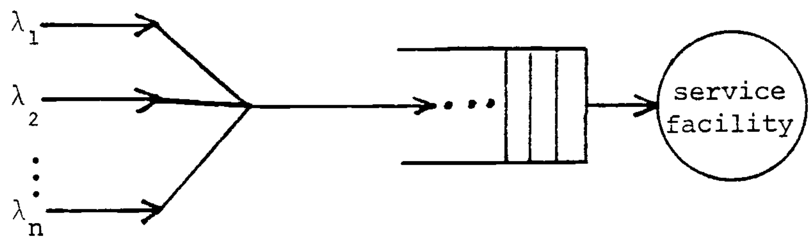


FIGURE 22. The Conceptual Global Queue

Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the arrival rates. Define the Distribution function of the sum of sources:

$$
\begin{aligned}
F(t) &= \Pr\{\text{arrival occurs before time } t\} \\
&= 1 - \Pr\{\text{all arrivals occur after time } t\} \\
&= 1 - \prod_{k=1}^{n} \Pr\{\text{arrival occurs after time } t\} \\
&= 1 - \prod_{k=1}^{n} \Pr\left\{\tilde{t}_k > t\right\} \\
&= 1 - \prod_{k=1}^{n} \left(1 - \Pr\left\{\tilde{t}_k \leqslant t\right\}\right) \\
&= 1 - \prod_{k=1}^{n} \left(1 - (1 - e^{-\lambda_k t})\right) \\
&= 1 - \prod_{k=1}^{n} e^{-\lambda_k t}
\end{aligned}
$$

Therefore,

$$F(t) = 1 - e^{-\sum_{k=1}^{n} \lambda_k t} \tag{13}$$

Assume $\lambda_1 = \lambda_2 = \cdots = \lambda_n = \lambda$

Therefore, the probability density distribution of t is

$$f(t) = n\lambda e^{-n\lambda t} \tag{14}$$

Therefore, the arrival distribution of the conceptual global queue is also a Poisson process with arrival rate of $n\lambda$ requests per unit time.

Assume that the service time for a request by each resource manager is given by an exponentially distributed random variable with mean $\frac{1}{\mu}$. Let $\bar{x}_s = \frac{1}{\mu}$, where $\bar{x}_s$ is the average service time. From Chapters IV and V, some requests will require running the algorithm at more than one site. Therefore, some requests will require service in more than one site. For the purpose of this analysis, we shall assume that such requests will be serviced at all the N sites. This means that in practice each site is a queueing system and the network will consist of a series of queues in tandem. Each site consists of an independent single exponential server at a rate μ. Therefore, each site is an M/M/1 queueing system. When a request leaves one site it queues up for service at the next site, as shown in Figure 23. We shall solve for the arrival process to the next site.

Assume that the algorithm is initiated at site i. Let $d(t)$ be the probability density function of the interdeparture process from site i. Let

$$B(t) = \Pr\{\text{arrival to site i at time } \tilde{t} \leqslant t\} \text{ , and}$$

$$b(t) = \frac{dB}{dt} = \Pr\{\tilde{t} = t\} \quad . \text{ Then}$$

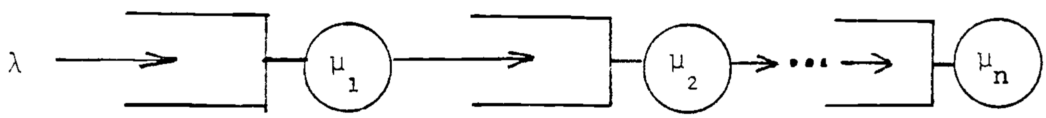


FIGURE 23. N-site Tandem Network

$$d(t) = \Pr\{\text{site i busy}\}\ b(t) + \Pr\{\text{site i empty}\}\int_0^t a(t-x)b(x)dx$$

$$= \rho\mu e^{-\mu t} + (1-\rho)\int_0^t \lambda e^{-\lambda(t-x)}\ \mu e^{-\mu t} dx$$

Since, for an M/M/1 model $\rho = \frac{\lambda}{\mu}$,

$$d(t) = \lambda e^{-\mu t} + (1-\rho)\lambda\mu\int_o^t e^{-\lambda t} \quad e^{-x(\mu-\lambda)} \quad dx$$

$$= \lambda e^{-\mu t} + \frac{(1-\rho)\lambda\mu e^{-\lambda t}}{\mu-\lambda}\left[-e^{-x(\mu-\lambda)}\right]_o^t$$

$$= \lambda e^{-\mu t} + \frac{(1-\rho)\lambda\mu}{\mu-\lambda} \quad e^{-\lambda t} \quad (1-e^{-(\mu-\lambda)t})$$

$$= \frac{(\lambda\mu-\lambda^2)e^{-\mu t} + (\lambda\mu-\lambda^2)(e^{-\lambda t} - e^{-\mu t})}{\mu-\lambda}$$

This expression simplifies to give

$$d(t) = \lambda e^{-\lambda t} \tag{15}$$

Therefore, the interdeparture times at site i, and subsequently the interarrival times at site i+1, are exponentially distributed with the same parameter as the interarrival times at site i. Also, we assume that the detection message length is fixed throughout passage through the network. Therefore, the average service time for a request that requires invocation of the algorithm can be approximated to $N\bar{x}_s$. This is a fairly good approximation to reality.

Therefore, the average service time of the algorithm for the global network is given by

$$\bar{x}_a = N\bar{x}_s \tag{16}$$

where $\bar{x}_s$ = average service time by each resource manager.

In most conventional queueing systems, the average service time is a constant. But in our system, the average service time of the algorithm per site, $\bar{x}_s$, is a variable of n and m. The

H&V algorithm involves maintaining a process-resource table dynamically. Therefore, $\bar{x}_s$ will increase with increased n and m.

Now let $\bar{x}_c$ be the service time due to communication delays. Since we are considering a global network the average service time of a request is given by

$$\bar{x} = \bar{x}_a + \bar{x}_c \tag{17}$$

Therefore, from equation (9) the average request response time for the Distributed H&V algorithm is:

$$T_{H\&V} = \bar{x} + \frac{P_z}{z\mu(1-\rho)} \tag{18}$$

However, the arrival rate, λ, in equation (9) represents network arrival rate. We have already shown that the arrival rate to the global queue is given by $\lambda = n\lambda_n$, where λ_n is per site arrival rate. ρ in equation (18) is the global network utilization and is therefore defined by

$$\rho = \frac{n\lambda_n}{z\mu} = \frac{n\lambda_n\bar{x}}{z} \tag{19}$$

In conventional queueing systems, the arrival rate of requests is often controlled by the customer. But in the Distributed H&V queueing model considered here, the rate of resource request depends on the request response time. Since a process is blocked when it makes a request, it can only make another request when the previous one had been granted and it has used it up to a point that it needs another resource. In some cases, where the request causes a deadlock, the process is rolled back and delayed a random length of time before it can make another request. Therefore the only way we could determine the arrival rates was from the simulation results.

The next problem is to find the amount of concurrency in the network. This will determine the parameter z in equation (18), thus giving us our M/M/z model. We have already mentioned that the value of z may not necessarily be an integer number, as in an M/M/m model, since the services provided by each resource manager is time-dependent. Obtaining a mathematical formula for the amount of concurrency is a complex combinatorial problem. Instead, we can use the results of the simulation model, the M/M/m model, and regression techniques. It was found that the value of z changes between one and two. Therefore, equation (12) holds for the H&V algorithm for the range of network size considered in this dissertation.

This result may not be surprising since the network considered by Jafari [43] is a ring network. And his analysis was for a network of maximum size 15. The maximum network size considered in this thesis is 12. Therefore, the average request response time for the Distributed H&V algorithm is:

$$T_{H\&V} = \bar{x} + \frac{\bar{x}\rho^{z}}{[1+(z-1)\rho](1-\rho)} \tag{20}$$

Tables 22 and 23 present the results of the mathematical model compared with the results of the simulation model for variable numbers of processes on a three-site network and variable numbers of sites, respectively. The simulation results are as listed in Tables 9 and 16. The average request rates and the two components of average service times were measured directly from the simulation model. To illustrate the differences between the two models the average response time versus number of processes and average response time versus number of sites were plotted for both the mathematical and simulation models in Figures 24 and 25. The vertical plots indicate the standard errors, based on 95 percent confidence interval derived from the student t-distribution, for the simulation results. From Tables 22 and 23 and Figures 24 and 25, we can observe that the mathematical and simulation results closely agree with each other.

As a conclusion to this chapter, it should be mentioned that the analytical model was obtained by first using the simulation data and regression techniques to obtain the M/M/z model, for z values between one and two. Although it is not claimed that this is a final performance model for the Distributed H&V algorithm, it seems the results obtained have provided a reasonable model to explain the simulation results.

TABLE 22. Comparison of Mathematical Results and Simulation Results for Varying Numbers of Processes on a 3-site Network for the Request Response Time of the Distributed Horizontal and Vertical Algorithm

# Processes (n)	Mean Inter-arrival Time ($\bar{\lambda}_n$)	Mean Algorithm Service Time ($\bar{x}_a$)	Mean Communication Service Time ($\bar{x}_c$)	Total Request Service Time ($\bar{x}$)	System Utilization (ρ)	Math. Average Request Response Time	Simulation Average Request Response Time
3	984.576	84.954	25.682	110.636	0.3180	157.894	141.917 ± 63.982
5	1693.620	114.515	25.682	140.197	0.3905	223.149	189.891 ± 39.511
6	2167.062	137.342	25.682	163.024	0.4258	275.017	287.085 ± 85.083
7	2762.616	171.137	25.682	196.819	0.4705	359.383	311.044 ± 90.344
10	3866.775	228.075	25.682	253.757	0.6190	639.989	588.665 ± 113.506

Degree of concurrency, z = 1.06

Number of resources per site, m = 2

TABLE 23. Comparison of Mathematical Results and Simulation Results for Varying Numbers of Sites for the Request Response Time of the Horizontal and Vertical Algorithm

Number of Sites (N)	Degree of Concurrency (z)	Total Request Service Time ($\bar{x}$)	System Utilization ρ	Math. Average Request Response Time	Simulation Average Request Response Time
3	1.06	117.3124	0.2188	146.9180	138.400 ± 32.72
5	1.17	189.2927	0.3172	257.9444	264.000 ± 60.50
8	1.47	326.5279	0.7302	894.0713	826.408 ± 101.50
10	1.63	541.6230	0.7646	1544.2256	1492.700 ± 102.52
12	1.68	712.2500	0.8168	2491.4000	2411.222 ± 406.59

Number of processes per site = 1

Number of resources per site = 1

COMPARISON OF MATHEMATICAL RESULTS WITH SIMULATION RESULTS FOR VARYING NUMBERS OF PROCESSES ON A 3-SITE NETWORK FOR THE REQUEST RESPONSE TIME OF DISTRIBUTED H&V ALGORITHM

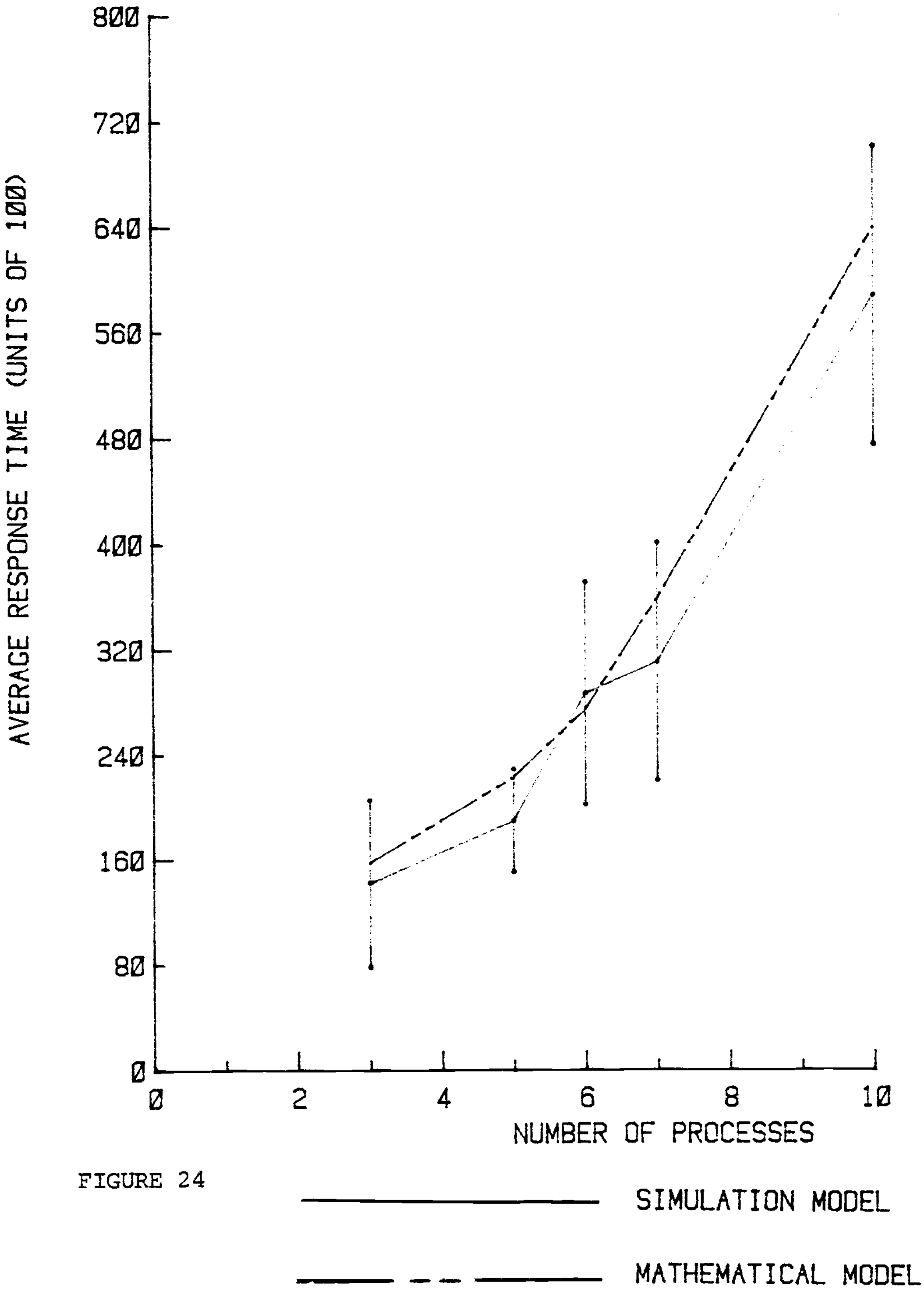


FIGURE 24

COMPARISON OF MATHEMATICAL RESULTS WITH SIMULATION RESULTS FOR VARYING NUMBERS OF SITES FOR THE REQUEST RESPONSE TIME OF DISTRIBUTED H&V ALGORITHM

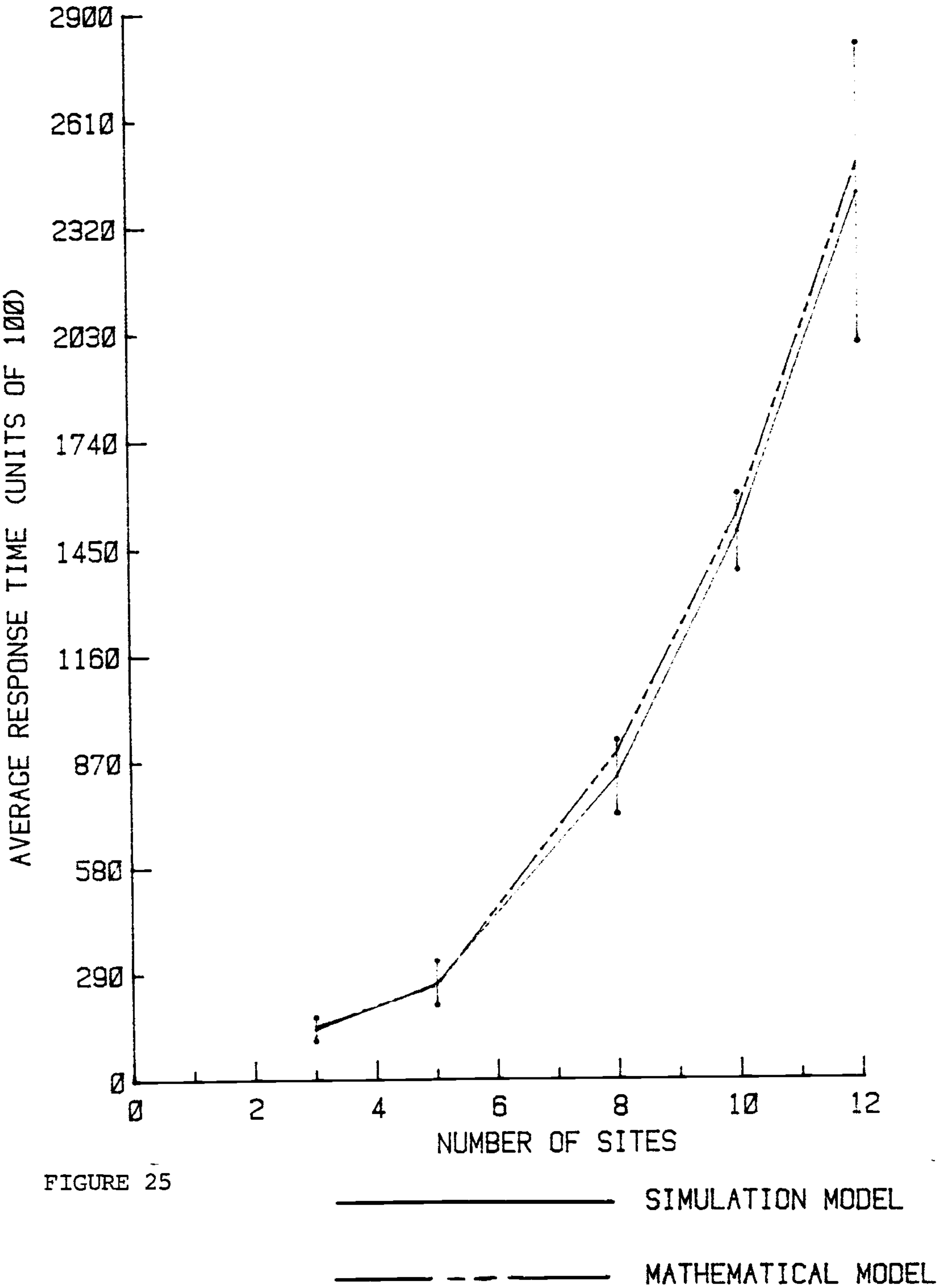


FIGURE 25

VIII. SUMMARY AND CONCLUSION

This dissertation proposes two solutions for on-line deadlock detection in a distributed computer system--the Centralized Horizontal and Vertical algorithm and the Decentralized Horizontal and Vertical algorithm. Simulation models are developed to study the performance of the two algorithms, Goldman's Distributed algorithm and deadlock prevention technique using preemption in a distributed computer ring network.

As in Goldman's distributed algorithm, the two protocols require that processes wait for only one resource at a time. A process is allowed to request for both shared and exclusive access to resources. Although a process is allowed to request for a resource, release it and latter request for another type of access or the same type of access to the same resource, it is not allowed to request for another type of access to a resource it is currently holding. So a process must make the type of access known when it makes a request. Both protocols require the building and maintenance of a Process-Resource table by the resource manager.

The centralized protocol assumes the existence of a controller site whose responsibilities are to allocate resources to competing processes and to check for deadlock. However, the centralized deadlock detection scheme proposed has some major drawbacks. It can result in message bottlenecks at the controller site, and if the controller site fails, it will result in significant delay while a new controller site is established. Also, in a network that is widely distributed over a large area, the delay can be annoying and undesirable if a local process requests for a local resource.

The Decentralized algorithm proposed requires each site to only maintain information on processes using resources located at its site. Thus the storage requirement needed to run the algorithm at each site is considerably reduced. The algorithm assumes a kind of site ordering in the network. Messages arrive in the order sent. There is no reordering of messages. Although a ring network topology is used in the performance evaluation of this algorithm because of the natural ordering of a ring network, any kind of network topology can be used. The ordering can be done by numbering the sites.

The Distributed H&V algorithm requires looking at each process-resource table only once. There is no passing of detection information forwards and backwards many times, as is characterized by Goldman's algorithm. The H&V algorithm will be run in at most N sites (N is the total number of sites in network), whereas in Goldman's algorithm, the number of sites that may run the algorithm, per initiation, may grow much larger than N. Therefore, synchronization problems due to communication delays are reduced to minimum in the H&V algorithm.

Goldman's algorithm requires the formation of a different copy of the OBPL for each process holding a shared resource. Each copy is expanded independently, and may have to go around the network more than once. In a system with many shared resources the algorithm leads to a heavy overhead in communication and time to run the algorithm. The H&V algorithm does not require any special way of handling shared resources. Each deadlock detection initiation requires only one detection message. Therefore, the proposed decentralized algorithm results in significant reduction of messages in the network.

The highlight of this dissertation is the simulation study of the new protocols, Goldman's algorithm and preemption scheme. The results show that preemption gives the lowest system throughput, while the Distributed H&V gives the best performance. Preemption is worst because of the high percentage of rollback involved. The results also show that the performance of any algorithm used depends on the amount of messages the algorithm generates. The more the network is congested, the more there are false deadlocks in the system. This will drive up the percentage of rollback that has to be performed, resulting in lower system throughput.

The performance of the algorithms were measured in terms of process average response time, process average throughput, request average response time, request average throughput, average message units per request, frequency of rollback and frequency of detection algorithm initiation. A unidirectional ring network topology was used in the experiment. The measurements were made on a three-site network with varying numbers of processes, up to a maximum of ten processes. Measurements were also taken for varying numbers of sites up to a maximum of twelve sites.

The Decentralized Horizontal and Vertical algorithm performed much better than Goldman's and the Centralized Horizontal and Vertical algorithms. The good performance of the Decentralized H&V algorithm is due to the lower amount of detection messages it generates. Also, each initiation of the algorithm results in running the algorithm in at most N sites, whereas Goldman's algorithm may be run in more than N sites. The Centralized protocol compares very favorably with the distributed solution. Although the maximum number of sites considered in this experiment was by no means large, the centralized solution seems more attractive for practical purposes because of its simplicity in implementation. Based on the results a centralized scheme may be recommended on a small network consisting of a few sites.

The greatest problem with distributed protocol is the occurrence of false deadlock. The simulation results revealed that this problem is not completely absent in the centralized scheme, because resource release messages take time to reach the controller site. In practical environments, it is recommended that the resource manager should give higher priority to resource release messages. This will free resources much faster, thereby reducing the frequency of detection algorithm initiation and subsequently reducing the frequency of rollback.

The simulation results show the following about the frequency of deadlock occurrence:

(a) For a fixed number of resources the frequency of deadlock increases as the number of processes increase,

(b) As the number of sites, number of processes and number of resources increase the frequency of deadlock increases,

(c) For a fixed number of processes and resources the frequency of deadlock increases as the loading factor increases,

(d) For a fixed number of processes the frequency of deadlock increases with increasing number of resources.

An analytical model was obtained by first using the simulation results and regression techniques to obtain an M/M/z queueing model for the response time of the Distributed H&V algorithm. The model developed in Chapter VII are summarized here:

$$T_{H\&V} = \bar{x} + \frac{\bar{x}\rho^{z}}{[1 + (z-1)\rho](1-\rho)}$$

where

$$\bar{x} = \bar{x}_a + \bar{x}_c \qquad \text{and}$$

$$\lambda = \frac{n\lambda_n \bar{x}}{z}$$

The results obtained from the analytical model were found to agree with the simulation results.

There is still much work to be done in the area of analyzing most of the existing deadlock detection algorithms. Simulation models have to be run for very large network. It was not possible to perform the experiments described in this dissertation for network larger than twelve sites because of limitations in the computing facilities. Analytical models for the frequency of deadlocks are yet to appear in the literature.

The method of rollback used in this dissertation is not recommended for practical purposes, especially in a distributed database environment. Research needs to be performed to determine efficient methods for rolling back processes.

In conclusion, the simulation study of the new detection algorithms, Goldman's algorithm and preemption scheme has helped to answer some questions about the operational behavior of deadlock resolution techniques. It is clear that preemption technique should never be used in any distributed system. Also, two deadlock detection protocols have been proposed. Their simplicity in implementation makes them very attractive for practical purposes. Analytical model has been developed for the new distributed protocol. Certainly, significant contributions have been made in the area of deadlock in distributed systems. Hopefully, future researchers in this area will not concentrate more on the theoretical aspect of the problem, but also on performance evaluation.

IX. BIBLIOGRAPHY

[1] Amman, U., Nori, K. and Jacobi, C., "The Portable Pascal Compiler," Institut Fuer Informatik, EIDG, Technische Hochschule CH-8096, Zurich, 1976.

[2] Bensonssan, A., "Overview of the Locking Strategy in the File System," Multics Systems Programmers Manual, Section BG.19.00, November 1969, Pages 1-10.

[3] Berstein, A. J. and Shoshani, A., "Synchronization in a Parallel Access Data Base," Communications of the ACM, Vol. 12, No. 11, November 1969, Pages 604-607.

[4] Campbell, R. H., "Path Expressions: A technique for specifying process synchronization," Ph.D. Thesis, The University of Newcastle upon Tyne, August 1976; Also, Department of Computer Science Technical Report, University of Illinois at Urbana-Champaign, UIUCDCS-R-77-863, May 1977.

[5] Campbell, R. H. and Habermann, A. N., "The Specification of Process Synchronization by Path Expressions," Lecture Notes in Computer Science, Springer-Verlag, Vol. 16, 1974, Pages 89-102.

[6] Campbell, R. H. and Kolstad, R. B., "Practical Applications of Path Pascal in Systems Programming," Department of Computer Science, University of Illinois, Urbana-Champaign, August 1979.

[7] Campbell, R. H. and Kolstad, R. B., "Path Expressions in Pascal," Fourth International Conference on Software Engineering, Munich, September 17-19, 1979.

[8] Campbell, R. H. and Kolstad, R. B., "A Practical Implementation of Path Pascal," Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, UIUCDCS-R-80-1008, 1980.

[9] Chamberlin, D. D., Boyce R. F. and Traiger, I. L., "A Deadlock-Free Scheme for Resource Locking in a Database Environment," Information Processing 74, Proc. IFIP Congress, North-Holland Publishing Co., Amsterdam, August 1974, Pages 340-343.

[10] Chandra, A. N., Howe, W. G. and Karp, D. P., "Communication Protocol for Deadlock Detection in Computer Networks," IBM Technical Disclosure Bulletin, Vol. 16, No. 10, March 1974, Pages 3471-3481.

[11] Chandy, K. M., Browne, J. C., Dissly, C. W., and Uhrig, W. R., "Analytic Models for Rollback and Recovery Strategies in Database Systems," IEEE Transactions on Software Engineering, Vol. SE-1, No. 1, March 1975, Pages 100-110.

[12] Chandy, K. M. and Ramamoorthy, C. V., "Rollback and Recovery Strategies," IEEE Transactions on Computers, Vol. C-21, No. 2, February 1972, Pages 137-146.

[13] Chu, W. W., "Optimal File Allocation in a Multiple Computer System," IEEE Transactions on Computers, Vol. C-18, No. 10, October 1969, Pages 885-889.

[14] Chu, W. W. and Ohlmacher, G., "Avoiding Deadlock in Distributed Data Bases," Proceedings, ACM National Symposium, Vol. 1, November 1974, Pages 156-160.

[15] CODASYL Database Task Group Report, Conference on Data System Languages, ACM, New York, April 1971.

[16] Coffman, E. G., Elphick, M. J., and Shoshani, A., "System Deadlocks," ACM Computing Surveys, Vol. 3, No. 2, June 1971, Pages 67-78.

[17] Collmeyer, A. J., "Database Management in a Multi-Access Environment," Computer, Vol. 4, No. 6, November/December 1971, Pages 36-46.

[18] Devillers, R., "Game Interpretation of the Deadlock Avoidance Problem," Communications of the ACM, Vol. 20, No. 10, October 1977, Pages 741-745.

[19] Dijkstra, E. W., "Cooperating Sequential Processes," Technological University, Eindhoven, The Netherlands, 1965. (Reprinted in Programming Languages, F. Genuys, ed., Academic Press, New York, 1968).

[20] Easton, W. B., "Process Synchronization without Long-Term Interlocks," Third ACM Symposium on Operating Systems Principles, Stanford University, October 1971, Pages 95-100.

[21] Ellis, C. A., "Probabilistic Models of Computer Deadlock," Report CU-CS-041-74, Department of Computer Science, University of Colorado, Boulder, Colorado, April 1974, 25 pages.

[22] Eswaran, K. P., Gray, J. N., Lorie, R. A., and Traiger, I. L., "The Notions of Consistency and Predicate Locks in a Data Base System," Communications of the ACM, Vol. 19, No. 11, November 1976, Pages 624-633.

[23] Fontao, R. O ., "A Concurrent Algorithm for Avoiding Deadlocks," Third ACM Symposium on Operating Systems Principles, Stanford University, October 18-20, 1971, Pages 72-79.

[24] Frailey, D. J., "A Practical Approach to Managing Resources and Avoiding Deadlocks," Communications of the ACM, Vol. 16, May 1973, Pages 323-329.

[25] Fry, J. P. and Sibley, E. H., "Evolution of Data-Base Management Systems," Computing Surveys, Vol. 8, No. 1, March 1976, Pages 7-42.

[26] Gerla, M., "Routing and Flow Control," University of California, Los Angeles. Printed in "Protocols and Techniques for Data Communication Networks," Franklin F. Kuo, ed., Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

[27] Gligor, V. D. and Shattuck, S. H., "On Deadlock Detection in Distributed Systems," IEEE Transactions on Software Engineering, Vol. SE-6, No. 5, September 1980, Pages 435-440.

[28] Goldman, B., "Deadlock Detection in Computer Networks," Technical Report MIT/LCS/TR-185, Laboratory for Computer Science, MIT, Cambridge, Mass., September 1977, 180 pages.

[29] Gray, J. N., "Notes on Database Operating Systems," in "Operating Systems (An Advanced Course)," in Lecture Notes in Computer Science 60, 1978, Pages 294-481.

[30] Habermann, A. N., "Prevention of System Deadlocks," Communications of the ACM, Vol. 12, No. 7, July 1969, Pages 373-377.

[31] Hansen, Per Brinch, "Operating System Principles," Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.

[32] Havender, J. M., "Avoiding Deadlock in Multitasking Systems," IBM Systems Journal, Vol. 7, No. 2, 1968, Pages 74-84.

[33] Hayes, J. F. and Sherman, D. N., "Traffic Analysis of a Ring Switched Data Transmission System," The Bell System Technical Journal, Vol. 50, No. 9, November 1971, Pages 2947-2978.

[34] Holt, R. C., "On Deadlock in Computer Systems," Cornell University, Ithaca, New York, 1971.

[35] Holt, R. C., "Comments on Prevention of System Deadlocks," Communications of the ACM, Vol. 14, No. 1, January 1971, Pages 36-38.

[36] Holt, R. C., "Some Deadlock Properties of Computer Systems," Computing Surveys, Vol. 4, No. 3, September 1972, Pages 179-196.

[37] Holt, R. C., Graham, G. S., Lazowska, E. D., Scott, M. A., "Structured Concurrent Programming with Operating Systems Applications," Addison-Wesley Publishing Company, Reading, Mass., 1978.

[38] Howard, J. H., Jr., "Mixed Solutions for the Deadlock Problem," Communications of the ACM, Vol. 16, No. 7, July 1973, Pages 427-430.

[39] Hutchison, D. A. et al., "A Recursive Algorithm for Deadlock Preemption in Computer Systems," in Information Processing 1977, Proc. IFIP Congress 77, Toronto, August 1977.

[40] Isloor, S. S. and Marsland, T. A., "Deadlock Detection in Databases Distributed on a Network of Computers," Technical Report TR 78-3, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, May 1978, 40 pages.

[41] Isloor, S. S. and Marsland, T. A., "An Effective "ON-LINE" Deadlock Detection Technique for Distributed Data Base Management Systems," Proceedings COMPSAC 78, Chicago, Illinois, November 1978, Pages 283-288.

[42] Isloor, S. S. and Marsland, T. A., "The Deadlock Problem: An Overview," Computer, September 1980, Pages 58-78.

[43] Jafari, H. "A New Loop Structure for Distributed Microcomputing Systems," Ph.D. Dissertation, Oregon State University, Oregon, December, 1977.

[44] Kahn, R. E. and Crowther, W. R., "A Study of the ARPA Computer Network Design and Performance," Report 2161, Bolt, Baranek and Newman, Inc., August 1971.

[45] Kahn, R. E. and Crowther, W. R., "Flow Control in a Resource-Sharing Computer Network," IEEE Transactions on Communications, Vol. COM-22, No. 6, June 1972, Pages 539-546.

[46] Kamoun, F., "Design Considerations for Large Computer Communications Networks," Ph.D. Dissertation, Engineering Report 7642, UCLA, Los Angeles, California, April 1976.

[47] King, P. F. and Collmeyer, A. J., "Database Sharing--An Efficient Mechanism for Supporting Concurrent Processes," Proceedings AFIPS National Computer Conference, June 1973.

[48] Kleinrock, L., Queueing Systems, John Wiley and Sons, Vol. 1, 1975.

[49] Kolstad, R. B. and Campbell, R. H., "Path Pascal User Manual," Technical Report, Department of Computer Science, University of Illinois, Urbana-Champaign, Urbana, Illinois, February 1980.

[50] Le Lann, G., "Pseudo-dynamic Resource Allocation in Distributed Databases," Proceedings Fourth International Conference on Computer Communications, ICCC-78, Kyoto, Japan, September 1978, Pages 245-251.

[51] Lomet, D. B., "A Practical Deadlock Avoidance Algorithm for Data Base Systems," Proceedings ACM-SIGMOD International Conference on Management of Data, Toronto, Canada, August 1977, Pages 122-127.

[52] Lomet, D. B., "Subsystems of Processes with Deadlock Avoidance," IEEE Transactions on Software Engineering, Vol. SE-6, No. 3, May 1980.

[53] Lynch, W. C., "An Operating System Design for the Computer Utility Environment," International Seminar on Operating System Techniques, Belfast, Northern Ireland, August-September 1971.

[54] Madnick, S. E. and Donovan, J. J., "Operating Systems," McGraw-Hill, New York, 1974, Pages 395-404.

[55] Mahmoud, S. A. and Riordon, J. S., "Protocol Considerations for Software Controlled Access Methods in Distributed Data Bases," Proceedings International Symposium on Computer Performance Modeling, Measurement and Evaluation, Cambridge, Mass., March 29-31, 1976, Pages 241-264.

[56] Mahmoud, S. A. and Riordon, J. S., "Software Controlled Access to Distributed Data Bases," INFOR, Vol. 15, No. 1, February 1977, Pages 22-36.

[57] Marsland, T. A. and Isloor, S. S., "Detection of Deadlocks in Distributed Database Systems," INFOR, Vol. 18, No. 1, February 1980, Pages 1-20.

[58] Maryanski, F. J., "A Deadlock Prevention Algorithm for Distributed Data Base Management Systems," Technical Report CA 77-02, Computer Science Department, Kansas State University, Manhattan, Kansas, February 1977, 24 pages.

[59] Maryanski, F. J., "A Survey of Development in Distributed Data Base Management Systems," IEEE Computer, Vol. 11, No. 2, February 1978, Pages 28-38.

[60] Maryanski, F. J. and Fisher, P. S., "Rollback and Recovery in Distributed Data Base Management Systems," Technical Report CS 77-05, Computer Science Department, Kansas State University, Kansas, February 1977.

[61] Menasce, D. A. and Muntz, R. R., "Locking and Deadlock Detection in Distributed Databases," IEEE Transactions on Software Engineering, Vol. SE-5, May 1979, Pages 195-202.

[62] Murphy, J. E., "Resource Allocation with Interlock Detection in a Multi-Task System," Proceedings of the 1968 Fall Joint Computer Conference, Pages 1169-1176.

[63] Needham, R. M. and Hartley, D. F., "Theory and Practice in Operating System Design," Second ACM Symposium on Operating Systems Principles, Princeton University, October 1969, Pages 8-12.

[64] Peebles, R. and Manning, E., "System Architecture for Distributed Data Management," IEEE Computer, Vol. 11, February 1978, Pages 28-38.

[65] Raubold, E. and Haenle, J., "A Method of Deadlock-free Resource Allocation and Flow Control in Packed Networks," Proceedings of the Third International Conference on Computer Communications, Toronto, August 1976.

[66] Russell, D. L., "Process Backup in Producer-Consumer Systems," Proceedings of Sixth ACM Symposium on Operating Systems Principles, November 1977, Pages 151-157.

[67] Russell, R. D., "A Model for Deadlock-free Resource Allocation," Ph.D. Dissertation, Stanford University, 1972.

[68] Rypka, D. J. and Lucido, A. P., "Deadlock Detection and Avoidance for Shared Logical Resources," IEEE Transactions on Software Engineering, Vol. SE-5, No. 5, September 1979, Pages 465-471.

[69] Schlageter, G., "Access Synchronization and Deadlock in Database Systems: An Implementation-Oriented Approach," Information Systems, Vol. 1, No. 2, 1975, Pages 92-102.

[70] Schwartz, M., "Computer-Communication Network Design and Analysis," Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

[71] Shemer, J. E. and Collmeyer, A. J., "Database Sharing: A Study of Interference, Roadblock, and Deadlock," Proceedings of the 1972 ACM-SIGFIDET Workshop, 1972.

[72] Stearns, R. E., Lewis, P. M. II, Rosenkrantz, D. J., "Concurrency Control for Database Systems," Proceedings IEEE 17th Annual Symposium on Foundations of Computer Science, October 1976, Pages 19-32.

[73] Vinton, G. C., "Packet Communication Technology," Defense Advanced Research Projects Agency, Information Processing Techniques Office. Printed in "Protocols and Techniques for Data Communication Networks," Franklin F. Kuo, ed., Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

APPENDICES

APPENDIX A

Description of the Simulation Programs

As mentioned in Section 6.1, the simulation models were written in Path Pascal. Path Pascal is an extension of Pascal P4 [1]. The extension includes an encapsulation machanism called objects, open path expressions [4], and a process mechanism. Open paths are integrated with the encapsulation mechanism to describe shared data objects. All access to encapsulated data is done by operations synchronized by open paths.

An object specifies the access mode, transformation and synchronization on its shared data. Its data and code are accessible to other parts of the program only by explicit declaration of entry types and entry operations. An object's operations (procedures, functions and processes) are differentiated from other internal operations by prefixing their declaration with the token "entry." The object's path expression specifies the synchronization constraints on a possibly concurrent set of operation executions within the object. A process is a procedure which has an independent execution sequence associated with it. It is differentiated from a standard pascal procedure by using the token "process" instead of "procedure" in its declaration. A process is instantiated dynamically by invoking the process name in a manner similar to a procedure invocation. A detail description of the Path Pascal compiler is beyond the scope of this thesis. However, to fully understand the simulation programs given in the appendices, the reader is advised to read through the Path Pascal User Manual [48].

The general structure of all four simulation programs is the same. The following objects are basically the same for all of them.

1. PROCIO

Procedures in these objects are used to encode the output report. Since there is only one printer to be shared concurrently by many machine objects it was necessary to encode the report to resolve contention for the printer. The encoded report was decoded by a separate pascal program. The explanation of the decode program is given in Appendix F. The path expression for the

PROCIO object allows all the procedures to execute in mutual exclusion. Figure 27a shows the components of this object.

2. LINE

The "LINE" object simulates the physical communication lines between machines. Each machine references two different lines: one for input and the other for output. A unidirectional ring network is assumed, and messages are passed clockwise (see Figure 26 and Figure 28). Each machine has two processes that have access to the line--the "Reader" and the "Writer". Access to the line is synchronized so that a reader is blocked if there is no message on the line. A writer is allowed to put a message to the line any time a message is available to be sent out. Figure 27b shows the structure of the "LINE" object. The shared data is the message buffer ("MESGBUF").

3. MACHINE

The "MACHINE" object simulates a site in the network.

The main components of the machine are the "Writer", the "Reader", the "Kernel" (resource manager) and user processes. The writer receives messages from the Kernel process and puts them on output line. The Reader monitors the input line for all incoming messages. Requests for local resources are put in queue to be processed by the Kernel. The Kernel handles all resource allocation at each site. Resource requests by processes at a site are sent to the Kernel at that site. The Kernel then determines whether the resource requested for is local or not. Requests for external resource are put on line. The Kernel runs the detection algorithm.

Within the machine objects are three main objects: the buffers--the input buffer, output buffer and user process's private buffers. Each process is assigned a private buffer. When a process makes a request, it is blocked, waiting on its private buffer for a response. Figure 27c shows the structure of the machine object. The simulated user process was the same for all the models.

A.1 Distributed Control

Figure 26 shows the network topology assumed for a three-site implementation of the distributed algorithms. The direction of message flow is indicated by the arrows. The structure of each of the programs is given in Figures 27a, 27b, and 27c. The detection algorithms are implemented in the "DETECTION ROUTINES". Since the preemption technique does not implement any algorithm, these procedures do not apply to the preemption program. It must be noted that, since the "KERNEL" is the only process that runs the detection algorithm, there is no inconsistency problem in the tables used by the detection algorithms.

Each message unit is organized into one pascal record construct. The number of message types used by each program depends on the needs of the algorithm.

A.1.1 Distributed Horizontal and Vertical Algorithm

The simulation program for the Distributed Horizontal and Vertical algorithm, Appendix B, uses the following types of messages:

1. Request : External resource request.
2. Response : When the resource manager allocates a resource to a requesting user process it sends this type of message to the process.
3. Completion : When a process releases a resource this type of message is sent to the site that owns the resource.
4. Rollback : Rollback type of message is sent by the "KERNEL", on detecting a deadlock, to the requesting process.
5. Locall : A process makes a request to its resource manager. The message is given a different type from external request type. If the resource requested is not local to the site, the resource manager

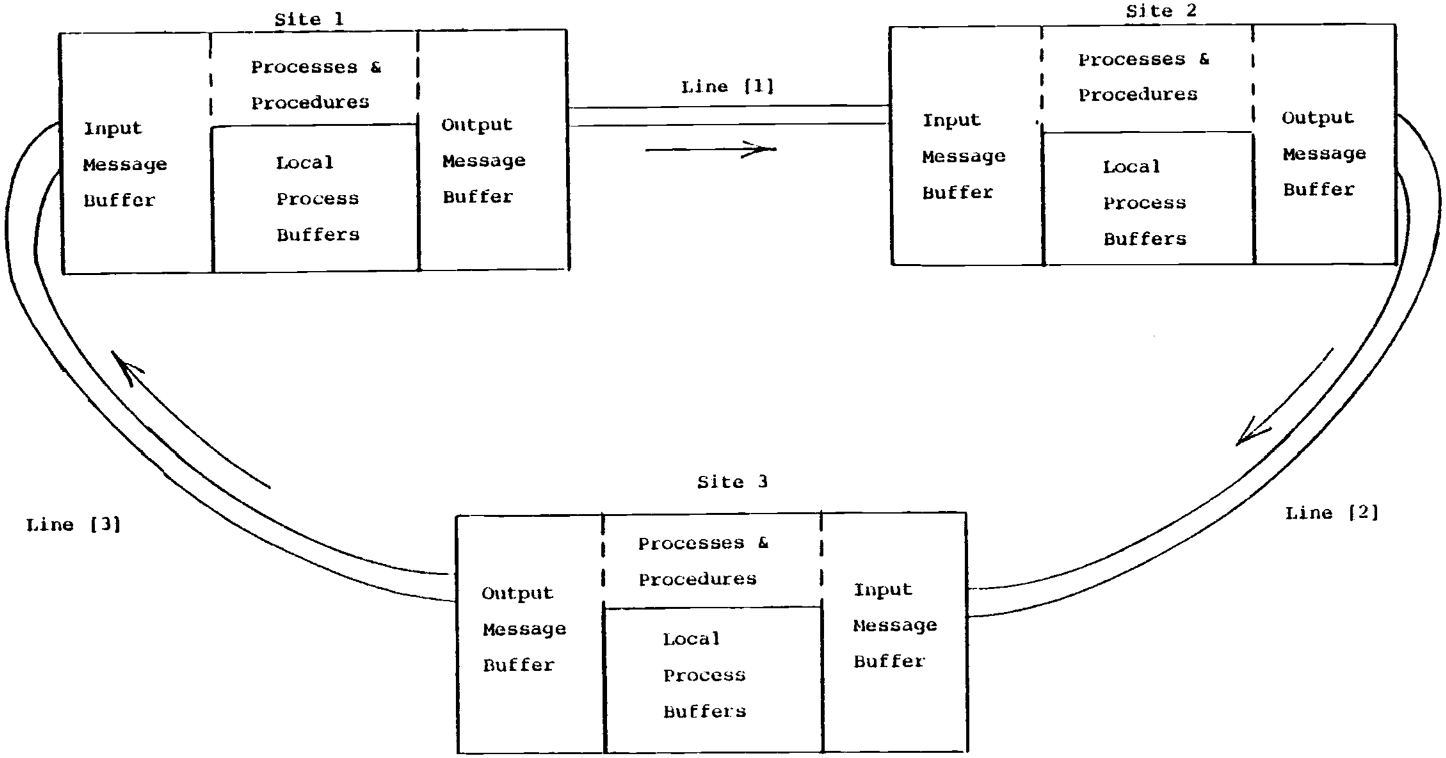


FIGURE 26. 3-Site Network Topology for Distributed Control Model

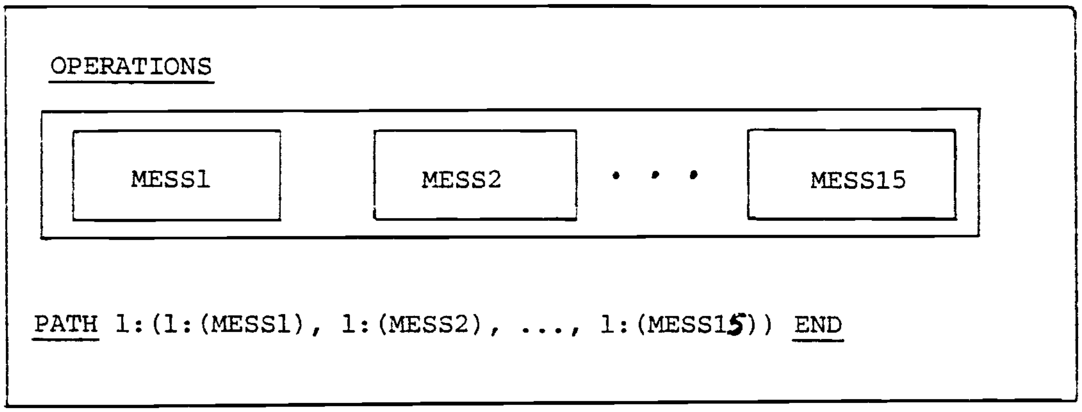


FIGURE 27a. "PROCIO" Object for Distributed Control Model

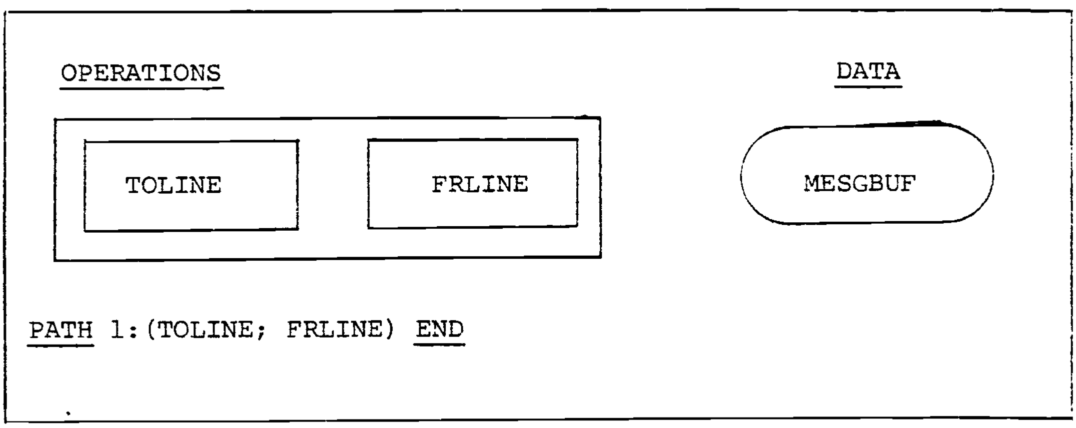


FIGURE 27b. "LINE" Object for Distributed Control Model

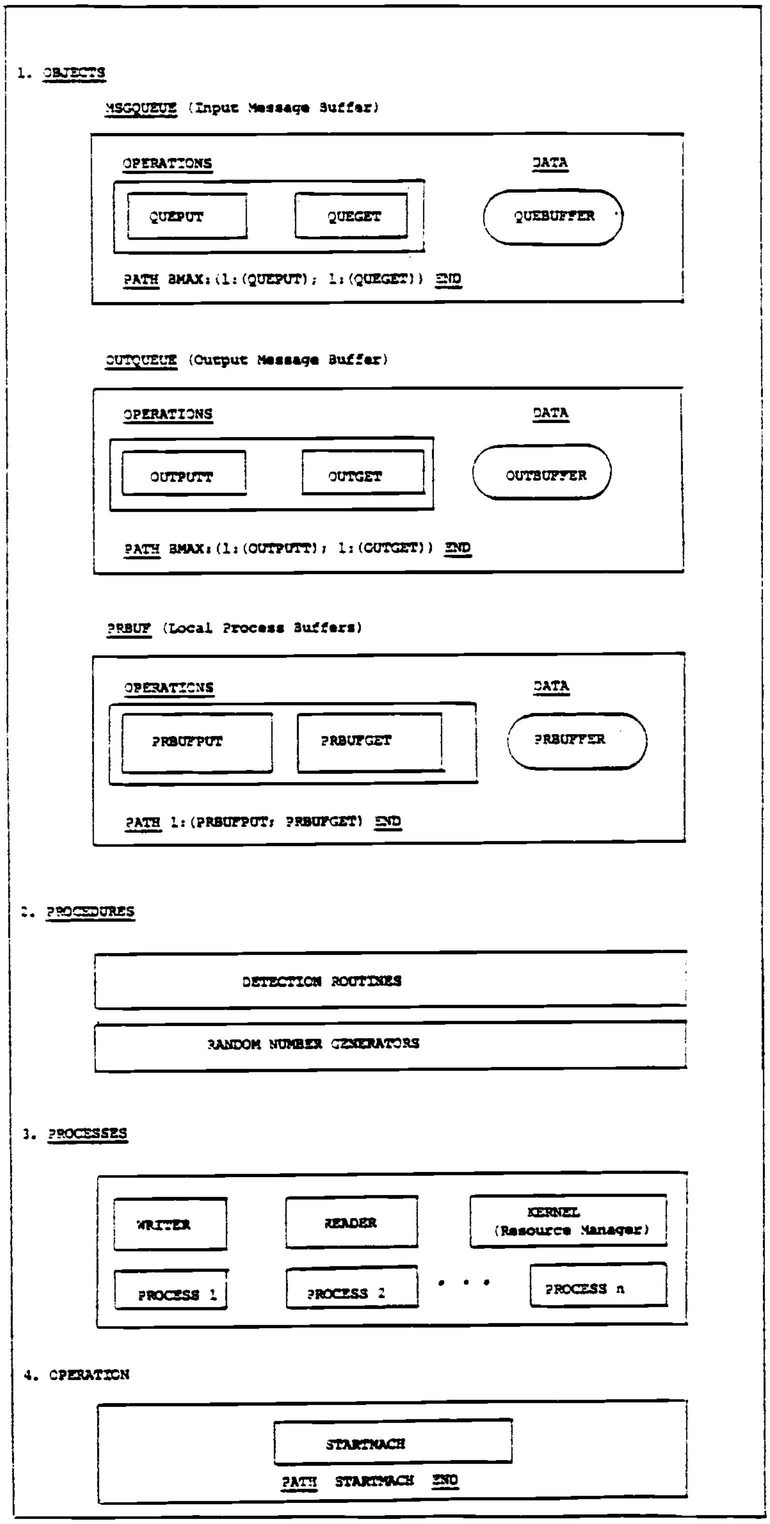


FIGURE 27c. "MACHINE" Object for Distributed Control Model

changes the message type to REQUEST before sending the request out.

6. Notfree : If a resource is not free for immediate allocation, this type of message is sent to the requesting process to wait for the resource. The message is sent after a successful completion of the detection algorithm.

7. Detek : This is the message type generated as a result of the detection algorithm initiation. It consists of the detection Path.

8. Aterminate : When a process runs to completion it must make this fact known to other processes running in the network. It thus sends a terminate message out.

A general pascal record construct was assumed for all message types. The following components made up the record: message type, message origin, message destination, process name, resource name, access type, detection Path and disjoint Path. Detection Path is an array of process names.

Deadlock Detection Initiation and Rollback Handling

Each site is responsible for managing the resources local to it. On receipt of a request, the kernel checks if the resource is free for immediate allocation. If it is not free, it ranks the request and initiates the detection algorithm. Detection is initiated every time the requested resource is not free for immediate allocation. Detection path is sent out only if deadlock is not detected locally. Before sending out the detection path, message type "DETEK", the initiating kernel sets message origin to itself, and message destination to the next site in the order.

When a kernel receives DETEK type of message it first checks if both the message origin and message destination are set to itself. If they are, then it is the detection path it sent out as a result of "process name's" request for "resource name" located at its site. It first checks the Process-Resource table to see if the process is still waiting for the resource. Note that it is possible for the resource to be free before the final detection path arrives. When a kernel sends out the detection path, it continues processing other messages. When a resource is released, the resource is allocated immediately to waiting processes.

If the process had been allocated the resource, the path is discarded, otherwise it checks the "deadlock" flag. If no deadlock is detected the kernel sends a NOTFREE message to the requesting process. In the event of a deadlock, the kernel sends a ROLLBACK message to the process. It then immediately releases all local resources held by the process and allocates them to other processes waiting for them. It also removes the process's name from the waiting list of any other resource at its site. The kernel then re-ranks all requests affected by the rollback.

A process maintains the names of the owner of all resources it acquires and the one it is waiting for if it has received a NOTFREE message. When it receives a rollback message it immediately releases all resources it holds. Since the resources from the site whose latest request caused the deadlock had already been released, the process only sends COMPLETION message (resource release) to other sites whose resources it held. All released messages are sent directly to the "Writer" process to put on line, unless the released resource is local, in which case, the message is given directly to the local kernel.

A.1.2 Distributed Goldman's Algorithm

Goldman's distributed algorithm, as proposed by Goldman [28], requires the site the requesting process resides to initiate the detection process. Also, in the event that no deadlock is detected, no message to this fact is sent to the requesting process. A slight modification was made to conform to our definition of on-line detection. The site where the requested resource resides was made to create the OBPL, and then send it to the site where the requesting process resides to start expansion. Also, when no deadlock is detected, a message was sent to the requesting process to wait for the resource. The following types of messages were considered for the simulation program, Appendix C.

The message types Request, Response, Completion, Rollback, Locall, Notfree and Aterminate saved the same purpose as in the distributed Horizontal and Vertical algorithm. In addition, the following were considered:

DETEK : Message type generated as a result of the detection algorithm. It consists of the OBPL.

INITDEAD : As mentioned earlier, Goldman's algorithm is supposed to be initiated by the site the requesting process resides. So if the resource requested for is not local to the site, the site owning the resource, after determining that the resource is not available for immediate allocation, sends a message to the site owning the process to initiate the detection algorithm. This message type is INITDEAD.

DLOCK : In Goldman's algorithm any site can detect a deadlock. If a deadlock is detected by a site other than that the requested resource resides, a message reporting the deadlock is sent to the site the resource resides. This enables the site to send a rollback message to the requesting process. The message type is DLOCK.

NFREE : Any site can determine that there is no deadlock, and discard the OBPL. Before discarding the OBPL, an NFREE message is sent to the site where the requested resource resides. This site then sends a NOTFREE message type to the requesting process to wait for the resource.

Each message is organized into one pascal record construct as for that of the H&V, with the following components: message type, message origin, message destination, process name, resource name, access type and OBPL. OBPL is in turn a record with components: resource name, location of resource, location of requesting process and array of process names.

Deadlock Detection Initiation and Rollback Handling

Like the H&V, each kernel is responsible for managing the resources local to it. But unlike the H&V, each site also maintains a table of all processes running locally. The Reader does not communicate directly with the processes. All messages for a process are passed to the kernel, who updates its table, before passing the message to the process.

When a kernel receives a request for a resource local to its site, it first checks if the resource is available for immediate allocation. If it is not, the kernel updates its table, creates an OBPL and sends the OBPL, message type INITDEAD, to the site the requesting process resides. When the message is received, the kernel changes the message type to DETEK and starts expanding the OBPL.

When a deadlock is detected by any site, a message reporting this (DLOCK message type) is sent to the site the requested resource resides. Also if no deadlock is detected, an NFREE message type is sent to the site the requested resource resides, before discarding the OBPL.

Like the H&V, before a ROLLBACK or NOTFREE message is sent to the requesting process, a check is made to see if the process is still waiting for the resource.

Unlike the H&V, partial rollback is not performed by the site the requested resource resides, with the exception of refusing waiting access to the resource, and removing the requesting process from the waiting list of the resource. This is because this site does not have enough information about the process, unless it is a local process. This was a program design consideration.

When the process receives the rollback message it immediately releases all the resources it holds. Unlike the H&V, the release message is sent to the kernel of the process. The kernel then updates its tables, before sending the message to the site the released resource resides, if it is not a local resource.

A.1.3 Preemption

The only types of message assumed in the premption model were Request, Response, Completion, Rollback, Locall, and Aterminate. Their meanings are as explained in Appendix A.1.1. The only table maintained by the resource manager is the local resource table. The simulation program is in Appendix E.

A.2 Centralized Control

In the centralized control model one site in the network was dedicated to resource management. No user process was allowed to run on the controller site, although in practical environment this restriction may be lifted. All requests were sent to this site. It was also assumed that all the resources available in the network were directly controlled by the controller site. Figure 28 shows a network topology assumed for the centralized control environment in the case where user processes ran on three sites. The direction of message flow is indicated by the arrows.

The program structure is given in Figures 29a, 29b, 29c and 29d. The "PROCIO" and "LINE" objects are similar to those of the distributed control. The controller object, Figure 29c, simulates the controller site, while the process machine object, Figure 29d, simulates the sites user processes run on. The "START CONTROLLER" and "START MACH" operations activate the controller and process machines, respectively.

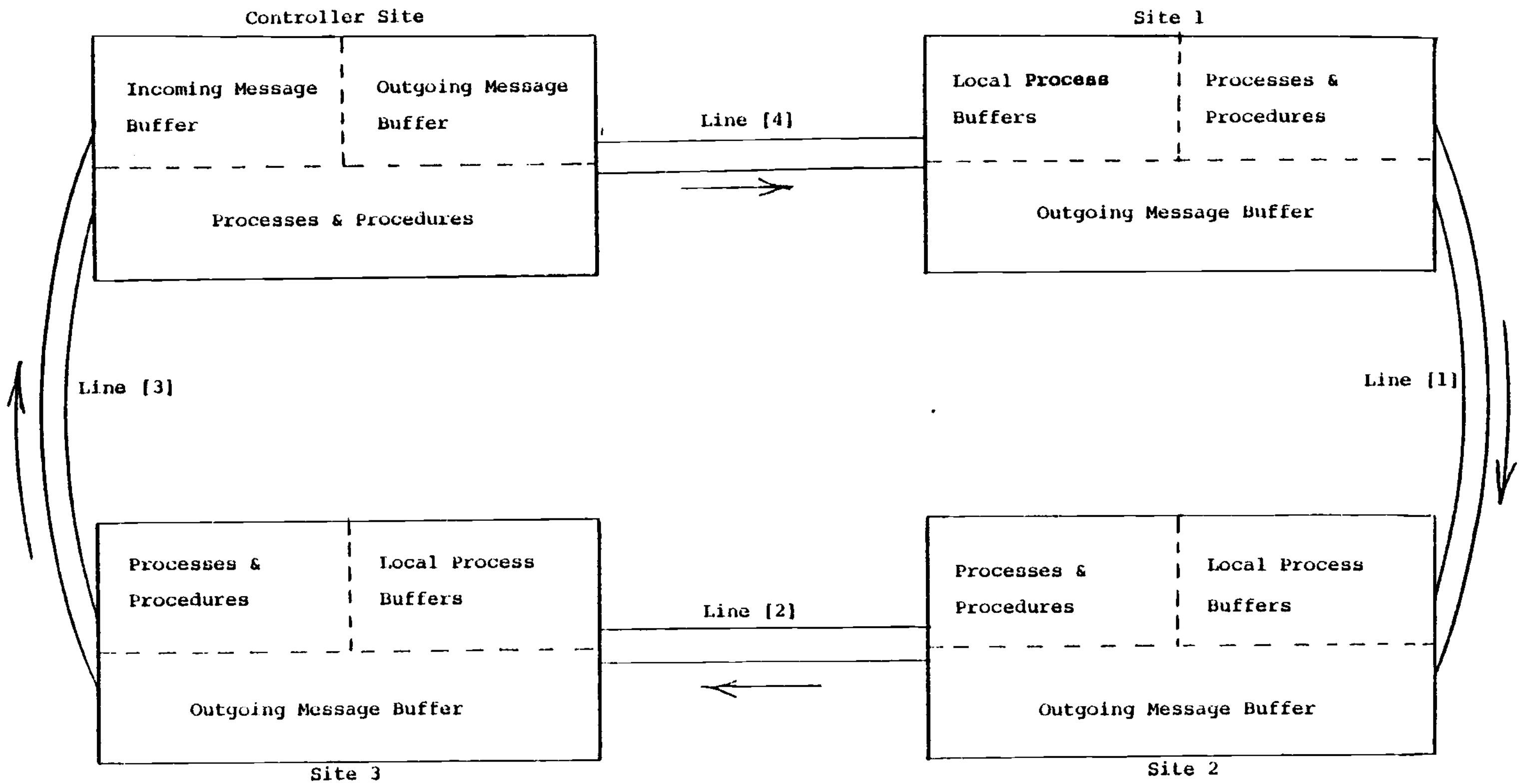


FIGURE 28. Centralized Control Network Topology

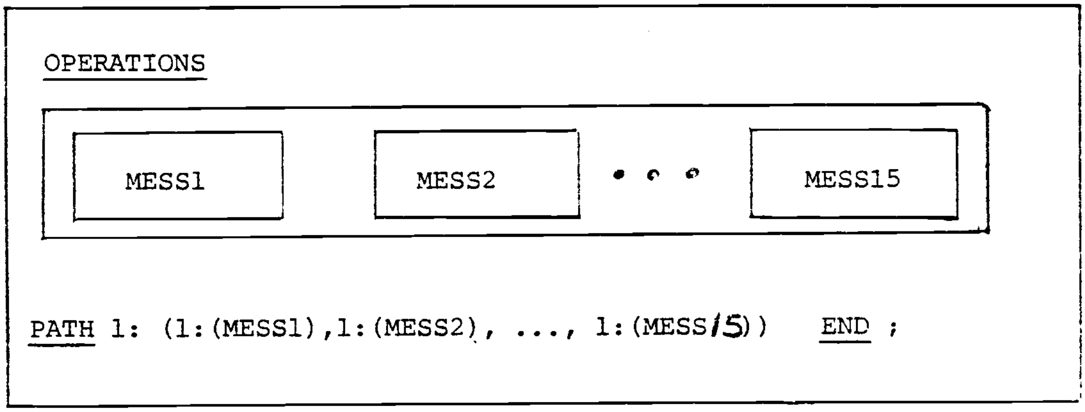


FIGURE 29a. "PROCIO" Object for Centralized Control Model

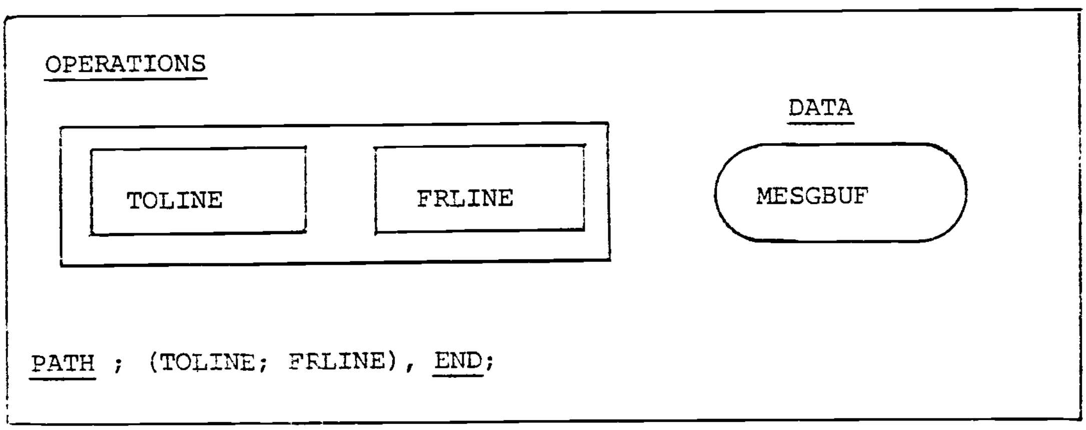


FIGURE 29b. "LINE" Object for Centralized Control Model

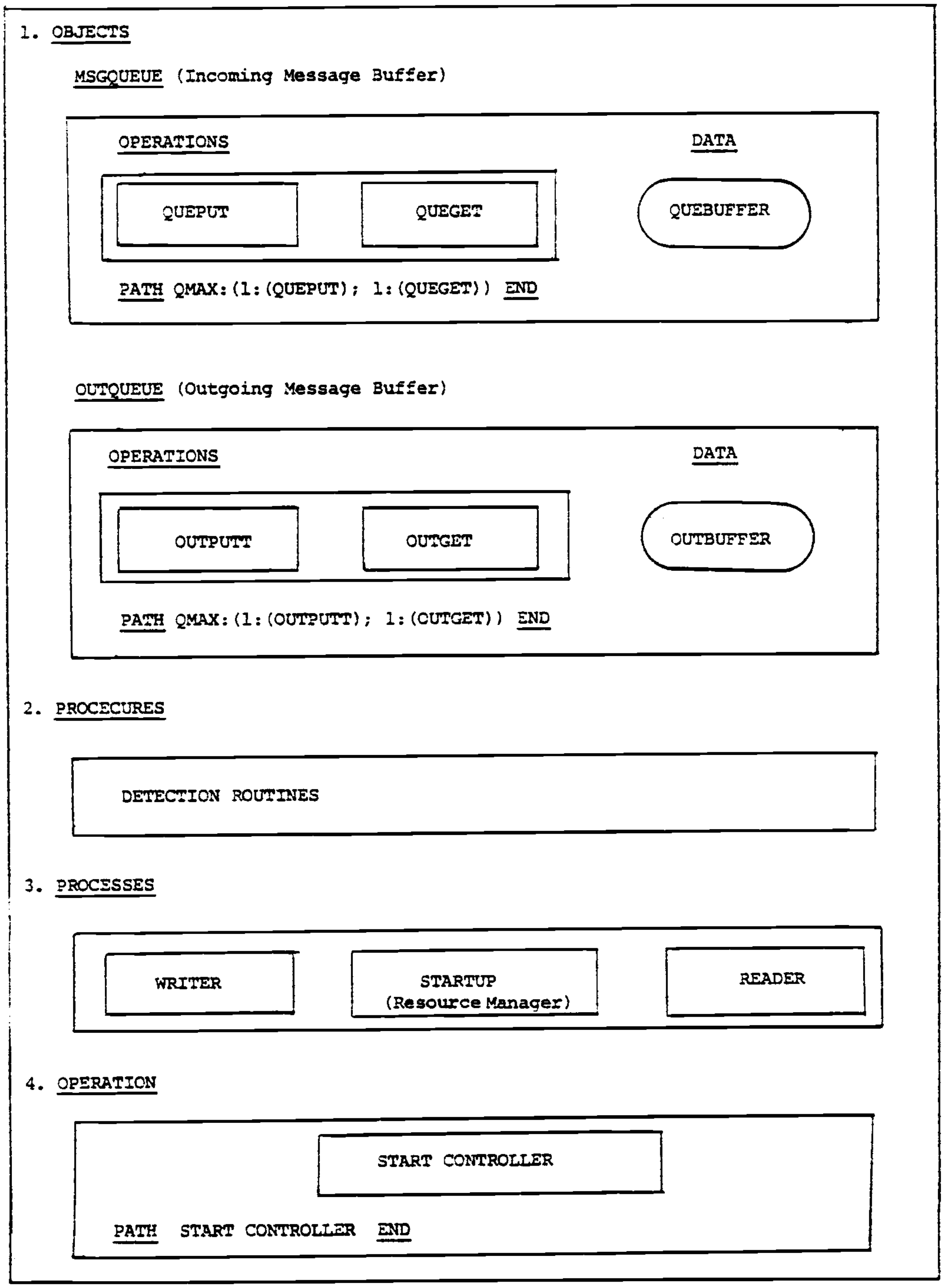


FIGURE 29c. "CONTROLLER" Object for Centralized Control Model

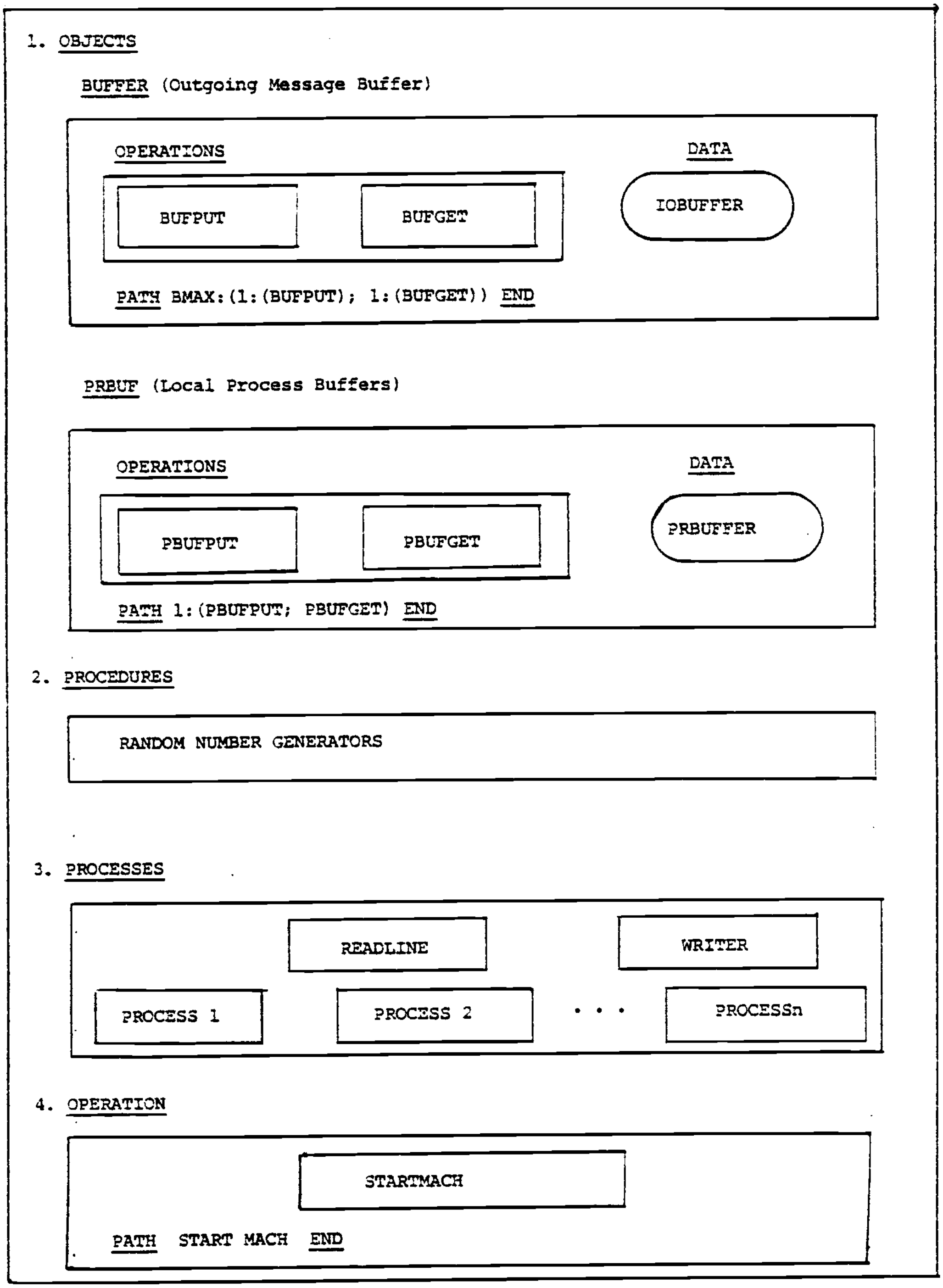


FIGURE 29d. Process "MACHINE" Object for Centralized Control Model

Six types of messages were assumed in the Centralized control model: Request, Response, Completion, Rollback, Notfree and Termination. Their meanings are as discussed in Appendix A.1.1.

When a deadlock was detected by the controller, a rollback message was sent to the requesting process. All the resources held by the process were then released by the controller, and allocated to other waiting processes. The rolled back process did not have to send resource release message to the controller, since the resources had already been released by the controller. The program listing is in Appendix D.

APPENDIX B

Program Listing for Distributed Implementation of the Horizontal and Vertical Algorithm on a 3-Site Network

```
PROGRAM DISBRC(INPUT,OUTPUT);

   (*****************************************************************)
   (*                                                               *)
   (*   DISTRIBUTED IMPLEMENTATION OF THE HORIZONTAL                *)
   (*   AND VERTICAL AGORITHM                                       *)
   (*                                                               *)
   (*****************************************************************)

   CONST
     NSITES=3;          (* 3 SITE NETWORK  *)
     BMAX=6;           (* BUFFER SIZE     *)
     NMAX=10;           (* MAXIMUM # PROCESSES *)
     MMAX=2;            (* MAXIMUM # RESOURCES A EACH SITE *)
     LINES=3;
   TYPE
     MESSTYPE=(AREQUEST,ARESPONSE,COMPLETION,ROLLBACK,LOCALL,
               NOTFREE,DETEK,ATERMINATE);
     SITES=1..NSITES;
     STATUS=(FREE,EXCLUSIVE,SHARED);
     NLINES=1..LINES;
     DSET=RECORD
            DIDENT:INTEGER;
            DPROCS:ARRAY[0..5] OF INTEGER;
     END;
     MESSAGE=RECORD
       MSGTYPE:MESSTYPE;
       MSGORIGIN:INTEGER;
       MSGDEST:INTEGER;
       PROCNAME:INTEGER;
       RESNAME:INTEGER;
       ACESTYPE:STATUS;
       QUESIZE:INTEGER;
       DEADLOCK:BOOLEAN;
       DPATHS:ARRAY[0..NMAX] OF INTEGER;
       DISJOINT:ARRAY[0..NMAX] OF DSET;
     END;

     PROCIO=OBJECT
       PATH 1:( 1:(MESS1),1:(MESS2), 1:(MESS3),1:(MESS4),
                1:(MESS5),1:(MESS6),1:(MESS7),1:(MESS8),
          1:(MESS9),1:(MESS10),1:(MESS11),1:(MESS12),
          1:(MESS13),1:(MESS14),1:(MESS15)) END;

       ENTRY PROCEDURE MESS1(I,J:INTEGER);
         VAR K:INTEGER;
         BEGIN
           K:=(J*100+I)*100;
           WRITELN(K)
         END;   (*  MESS1  *)

       ENTRY PROCEDURE MESS2(I,J:INTEGER);
```

```
  VAR K:INTEGER;
  BEGIN
    K:=(J*100+I)*100+1;
    WRITELN(K)
  END;   (*  MESS2  *)

ENTRY PROCEDURE MESS3(I,J,K:INTEGER;L:STATUS);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100;
    IF L=EXCLUSIVE THEN T:=T+2 ELSE T:=T+3;
    WRITELN(T)
  END;   (*  MESS3  *)

ENTRY PROCEDURE MESS4(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
   T:=I*100000+J*10000+K*100+4;
   WRITELN(T)
  END;   (*  MESS4  *)

ENTRY PROCEDURE MESS5(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+5;
    WRITELN(T)
  END;   (*  MESS5  *)

ENTRY PROCEDURE MESS6(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+10;
    WRITELN(T)
  END;   (*  MESS6  *)

ENTRY PROCEDURE MESS7(I,J,K,L :INTEGER);
  VAR T,T1,T2,T3 :INTEGER;
  BEGIN
    T:=I*1000+J*100+11;
    T1:=K*100000+I*1000+J*100+12;
    T3:=L DIV 100;
    T2:=T3*10000+I*100+J*10+8;
    WRITELN(T,T1,T2)
  END;   (*  MESS7  *)

ENTRY PROCEDURE MESS8(I,J,K:INTEGER);
  VAR T,T1 : INTEGER;
  BEGIN
    T:=J*10000+I*100+13;
    T1:=K*10000+I*100+14;
    WRITELN(T,T1)
  END;   (*  MESS8  *)
```

```
ENTRY PROCEDURE MESS9(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+15;
    WRITELN(T)
  END;   (*  MESS9  *)

ENTRY PROCEDURE MESS10(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+20;
    WRITELN(T)
  END;   (*  MESS10  *)

ENTRY PROCEDURE MESS11(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+21;
    WRITELN(T)
  END;   (*  MESS11  *)

ENTRY PROCEDURE MESS12(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+22;
    WRITELN(T)
  END;   (*  MESS12  *)
  ENTRY PROCEDURE MESS13(I,J,DU,QS :INTEGER);
      VAR T,T1,T3:INTEGER;
      BEGIN
        DU:=DU DIV 100;
        T:=I*100+J*10;
        T1:=DU*10000+T+7;
        T3:=QS*10000+T+9;
        WRITELN(T1,T3)
   END;   (*  MESS13  *)

ENTRY PROCEDURE MESS14(I,J,K :INTEGER);
     VAR T,T1 :INTEGER;
     BEGIN
       T:=J*10000+I*100+23;
       T1:=K*10000+I*100+26;
       WRITELN(T,T1)
     END;   (*  MESS14  *)

ENTRY PROCEDURE MESS15(I,NF,DE,RES.ROL,COM,
   ARE,K1,K2.K3 :INTEGER);
 VAR T1,T2,T3,T4,T5,T6,T7,T8,
     T9,T1000,T100 : INTEGER;
     BEGIN
      T1000:=10000;
      T100:=I*100;
      T1:=NF*T1000+T100+30;
```

```
          IF DE<>99 THEN BEGIN
                          T2:=DE*T1000+T100+31;
                          WRITE(T2)
                        END;
          T3:=RES*T1000+T100+32;
          T4:=ROL*T1000+T100+33;
          T5:=COM*T1000+T100+34;
          T6:=ARE*T1000+T100+35;
          IF K1<>99 THEN BEGIN
                         T7:=K1*T1000+T100+36;
                         T8:=K2*T1000+T100+24;
                         T9:=K3*T1000+T100+25;
                         WRITE(T7,T8,T9)
                       END;
          WRITELN(T1,T3,T4,T5,T6)
   END;   (*  MESS15  *)


END;        (*  ******  PROCIO  ******  *)

LINE=OBJECT

  PATH 1:(TOLINE;FRLINE) END:
  VAR MESGBUF:MESSAGE;

  ENTRY PROCEDURE TOLINE(M:MESSAGE);
    BEGIN
     MESGBUF:=M
    END;       (*  TOLINE  *)

  ENTRY PROCEDURE FRLINE(VAR M:MESSAGE);
    BEGIN
      M:=MESGBUF
    END;       (*  FRLINE  *)

END;       (*  ******  LINE  ******  *)

MACHINE=OBJECT
  PATH STARTMACH END;

  TYPE
   MSGQUEUE=OBJECT  (* INPUT MSGES TO BE PROCESSED  *)
     PATH BMAX :(1:(QUEPUT);1:(QUEGET)) END;
     VAR QUEBUFFER:ARRAY[1..BMAX] OF MESSAGE:
         INQQ,OUTQQ:1..BMAX;

    ENTRY PROCEDURE QUEPUT(M:MESSAGE);
      BEGIN
        QUEBUFFER[INQQ]:=M;
        INQQ:=(INQQ MOD BMAX)+1
      END;   (*   QUEPUT  *)

    ENTRY PROCEDURE QUEGET(VAR M:MESSAGE;
```

```
                VAR QS:INTEGER);
    BEGIN
     M:=QUEBUFFER[OUTQQ];
     IF OUTQQ>INQQ THEN QS:=(BMAX-OUTQQ)+INQQ ELSE
                             QS:=INQQ-OUTQQ;
     OUTQQ:=(OUTQQ MOD BMAX) + 1;
    END;   (*          *)

 INIT; BEGIN
        INQQ:=1;
        OUTQQ:=1
       END;       (*  INIT  *)

END;    (*  ******  MSGQUEUE  ******  *)


OUTQUEUE=OBJECT    (*  MSGES TO BE SENT OUT  *)
  PATH BMAX:(1:(OUTPUTT);1:(OUTGET)) END;
  VAR OUTBUFFER:ARRAY[1..BMAX] OF MESSAGE;
      OUTP,OUTG:1..BMAX;

  ENTRY PROCEDURE OUTPUTT(M:MESSAGE);
     BEGIN
       OUTBUFFER[OUTP]:=M;
       OUTP:=(OUTP MOD BMAX) + 1
     END;   (*  OUTPUTT  *)

  ENTRY PROCEDURE OUTGET(VAR M:MESSAGE);
     BEGIN
       M:=OUTBUFFER[OUTG];
       OUTG:=(OUTG MOD BMAX )+1
     END;     (*  OUTGET   *)

  INIT; BEGIN
          OUTP:=1;
          OUTG:=1
         END;  (*  INIT  *)

END;   (*  ******  OUTQUEUE  ******  *)

PRBUF=OBJECT   (* PRIVATE BUFFER FOR EACH PROCESS*)
  PATH 1:(PRBUFPUT;PRBUFGET) END;
  VAR  PRBUFFER:MESSAGE;

  ENTRY PROCEDURE PRBUFPUT(M:MESSAGE);
    BEGIN
      PRBUFFER:=M
    END;   (* PRBUFFER  *)

  ENTRY PROCEDURE PRBUFGET(VAR M:MESSAGE);
    BEGIN
      M:=PRBUFFER
    END;  (*  PRBUFGET  *)
```

```
END;  (*  ******  PRBUF  ******  *)

PRTBLE=RECORD
  RNKNTEGER;
  TACCES:STATUS
END;
MAT=ARRAY[0..MMAX,0..NMAX] OF PRTBLE;
STATE=(BLOCKED,RUNNING);
RESHELD=RECORD
  RNAM:INTEGER;
  RACC:STATUS
END;
PROCS=RECORD
 PNAME:INTEGER;
 PSITE:INTEGER;
 PSTATE:STATE;
 RHELD:ARRAY[0..MMAX] OF RESHELD
END;
RESRC=RECORD
 RNAME:INTEGER;
 RSTATUS:STATUS
END;
VAR
  MQUEUE:MSGQUEUE;
  OQUEUE:OUTQUEUE;
  PBUF:ARRAY[1..2] OF PRBUF;
  IO:PROCIO;
  PROCTAB:ARRAY[0..NMAX] OF PROCS;
  LRESTAB:ARRAY[0..MMAX] OF RESRC;
  PRTABLE:MAT;
  MARKED:ARRAY[0..NMAX] OF BOOLEAN;
  PPATHS:ARRAY[0..NMAX] OF INTEGER;
  N,M,PP,RR: INTEGER;
  P2:ARRAY[0..NMAX] OF INTEGER;
  REQACCESS:STATUS;
  MSGTEMP:MESSAGE;
  TENTRY:(REQ,REL,DETEC);
  TOTREQ,IFR,JFP,TOTDEAD,NINITD  :INTEGER;
  MYSITE:SITES;
  STK:ARRAY[0..20] OF INTEGER;

 (*****************************************)
 (*                                      *)
 (*  DETECTION ROUTINES                  *)
 (*                                      *)
 (*****************************************)

 PROCEDURE INITIALIZE;
  (* INITIALISES THE PROCESS-RESOURCE TABLE ,THE *)
  (* PROCESS AND THE RESOURCE TABLES             *)
  VAR
      I,J:INTEGER;
```

```
BEGIN
 FOR I:=0 TO NMAX DO
   BEGIN
     PROCTAB[I].PNAME:=-1;
     PROCTAB[I].PSITE:=-1;
     PROCTAB[I].PSTATE:=BLOCKED;
     FOR J:=0 TO MMAX DO
       BEGIN
         PROCTAB[I].RHELD[J].RNAM:=-1;
         PROCTAB[I].RHELD[J].RACC:=SHARED
       END;
   END;
  FOR I:=0 TO MMAX DO
   BEGIN
     LRESTAB[I].RNAME:=-1;
     LRESTAB[I].RSTATUS:=FREE
   END;
  FOR I:=0 TO MMAX DO
    FOR J:=0 TO NMAX DO
      BEGIN
         PRTABLE[I,J].RNK:=-1;
         PRTABLE[I,J].TACCES:=FREE
      END;
 END;  (*  INITIALISE  *)

FUNCTION NEWP(PNTEGER):BOOLEAN;
  (* RETURNS TRUE IF REQUESTING PROCESS IS
                  NOT IN TABLE *)
  VAR I:INTEGER;
  BEGIN
      I:=0;
      WHILE (PROCTAB[I].PNAME<>P) AND (I<=N) DO
        I:=I+1;
      IF I>N THEN NEWP:=TRUE ELSE NEWP:=FALSE;
  END;  (*  NEWP  *)

  FUNCTION FINDP(P:INTEGER):INTEGER;
   (* RETURNS AN INDEX TO A PROCES IN THE PROCES TABLE *)
   VAR I:INTEGER;
   BEGIN
     I:=0;
     WHILE (PROCTAB[I].PNAME<>P) AND (I<=N) DO
       I:=I+1;
    IF I>N THEN
            BEGIN WRITELN(' **ERR**',P);
                  FINDP:=999 END
       ELSE FINDP:=I;
   END;  (*  FINDP  *)

FUNCTION FINDR(R:INTEGER)NTEGER;
  (* RETURNS AN INDEX TO A RESOURCE IN RESOURCETABLE *)
 VAR I:INTEGER;
  BEGIN
```

```
   I:=0;
   WHILE LRESTAB[I].RNAME<>R DO I:=I+1;
   FINDR:=I;
  END;   (*  FINDR  *)


PROCEDURE INSERTP;
   BEGIN
     JFP:=0;
     WHILE (PROCTAB[JFP].PNAME<>-1) DO
       JFP:=JFP+1;
     WITH PROCTAB[JFP] DO
       BEGIN
         PNAME:=PP;
         PSITE:=MSGTEMP.MSGORIGIN
       END;
  END;


  PROCEDURE ALLOCATER;
    (* ALLOCATES RESOURCES TO WAITING  PROCESSES  *)
    VAR ROW,I,JNTEGER;
    BEGIN
      ROW:=FINDR(RR);
      FOR J:=0 TO N DO
       WITH PRTABLE[ROW,J] DO
        IF RNK>0 THEN RNK:=RNK-1;
      FOR J:=0 TO N DO
       IF PRTABLE[ROW,J].RNK=0 THEN
         BEGIN
           (*  ALLOCATE RESOURCE TO PROCESS WITH INDEX J *)
          PROCTAB[J].PSTATE:=RUNNING;
           (* SEND RESPONSE MSG  *)
            WITH MSGTEMP DO
              BEGIN
                MSGTYPE:=ARESPONSE;
                MSGORIGIN:=MYSITE;
                MSGDEST:=PROCTAB[J].PSITE;
                RESNAME:=RR;
                PROCNAME:=PROCTAB[J].PNAME MOD 1000;
                DPATHS[0]:=-1
              END;
            I:=0;
          WHILE PROCTAB[J].RHELD[I].RNAM<>RR DO I:=I+1;
          LRESTAB[ROW].RSTATUS:=PROCTAB[J].RHELD[I].RACC;
          MSGTEMP.ACESTYPE:=LRESTAB[ROW].RSTATUS;
          IF MSGTEMP.MSGDEST=MYSITE THEN
          PBUF[MSGTEMP.PROCNAME].PRBUFPUT(MSGTEMP) ELSE
          OQUEUE.OUTPUTT(MSGTEMP);
          (* IO.MESS9(MSGTEMP.MSGDEST,MSGTEMP.PROCNAME,RR); *)
       END;
END;  (*  ALLOCATER  *)
```

```
PROCEDURE RESREL;
    (* HANDLES RESOURCE RELEASE FOR NORMAL
       COMPLETION  *)
  VAR J : INTEGER;
      SW,SW1 : BOOLEAN;

  BEGIN
    J:=0;
    WHILE (PROCTAB[JFP].RHELD[J].RNAM<>RR)
          AND (J<=M) DO J:=J+1;
    IF J>M THEN WRITELN('**ERRES**');
    WITH PROCTAB[JFP].RHELD[J] DO
        BEGIN
          RNAM:=-1; RACC:=SHARED
        END;
    (* REMOVE ENTRY FROM PRTABLE *)
    WITH PRTABLE[IFR,JFP] DO
         BEGIN
           RNK:=-1;  TACCES:=FREE
         END;
    (* CHECK IF PP STILL HAS A RESOURCE AT
        THIS SITE   *)
    SW:=FALSE;
    FOR J:=0 TO MMAX DO
     IF PROCTAB[JFP].RHELD[J].RNAM<>-1
        THEN SW:=TRUE;
    IF NOT SW THEN (* REMOVE PP *)
     WITH PROCTAB[JFP] DO
        BEGIN
          PNAME:=-1; PSTATE:=BLOCKED
        END;
     (* ANY MORE PROCESS USING RR *)
    SW:=FALSE;
    FOR J:=0 TO N DO
     IF PRTABLE[IFR,J].RNK=0 THEN SW:=TRUE;
    IF NOT SW THEN (* NO PROCESS *)
     BEGIN
       (* ANY PROCESS WAITING FOR RR *)
      SW1:=FALSE;
      FOR J:=0 TO N DO
       IF PRTABLE[IFR,J].RNK>0 THEN
          SW1:=TRUE;
      IF NOT SW1 THEN
         LRESTAB[IFR].RSTATUS:=FREE
         ELSE ALLOCATER
     END;
 END;  (*  RESREL  *)

 PROCEDURE ROLLBREL;
   (* ROLLS BACK A PROCESS *)
  VAR J,TRANK : INTEGER;
      SW,SW1 : BOOLEAN;
```

```
  BEGIN
   J:=0;
   WHILE (PROCTAB[JFP].RHELD[J].RNAM<>RR)
     AND (J<=M) DO J:=J+1;
   IF J>M THEN WRITELN('**ERROL**');
   WITH PROCTAB[JFP].RHELD[J] DO
     BEGIN
       RNAM:=-1;  RACC:=SHARED
     END;
   TRANK:=PRTABLE[IFR,JFP].RNK;
   WITH PRTABLE[IFR,JFP] DO
     BEGIN
       RNK:=-1;  TACCES:=FREE
     END;
   IF TRANK=0 THEN
    BEGIN
     SW:=FALSE;
     FOR J:=0 TO N DO
      IF PRTABLE[IFR,J].RNK=0 THEN
         SW:=TRUE;
     IF NOT SW THEN
       BEGIN
         SW1:=FALSE;
         FOR J:=0 TO N DO
          IF PRTABLE[IFR,J].RNK>0 THEN
             SW1:=TRUE;
         IF NOT SW1 THEN
            LRESTAB[IFR].RSTATUS:=FREE
            ELSE ALLOCATER
       END;
    END  ELSE
    BEGIN
     SW:=FALSE;
     FOR J:=0 TO N DO
      IF PRTABLE[IFR,J].RNK=TRANK
         THEN SW:=TRUE;
     IF NOT SW THEN
      FOR J:=0 TO N DO
       IF PRTABLE[IFR,J].RNK>TRANK THEN
       PRTABLE[IFR,J].RNK:=
       PRTABLE[IFR,J].RNK-1;
    END;
END;   (*  ROLLBREL   *)

PROCEDURE ROLLB;
 (* ABORTS A PROCESS AND ALLOCATES ALL
    RESOURCES TO OTHER WAITING PROCESSES *)
 VAR J,K : INTEGER;

 BEGIN
  K:=-1;
  FOR J:=0 TO MMAX DO
   IF PROCTAB[JFP].RHELD[J].RNAM<>-1 THEN
```

```
     BEGIN
      K:=K+1;
      P2[K]:=PROCTAB[JFP].RHELD[J].RNAM
     END;
   FOR J:=0 TO K DO
    BEGIN
      RR:=P2[J];  IFR:=FINDR(RR);
      ROLLBREL
    END;
   (* REMOVE PP FROM SITE *)
   JFP:=FINDP(PP);
   WITH PROCTAB[JFP] DO
     BEGIN
       PNAME:=-1;  PSTATE:=BLOCKED
     END;
END;

 (**********  H&V ALGORITHM STARTS  *********)


   PROCEDURE HORIZONTAL(VAR R,H:INTEGER);
     (* THE HORIZONTAL ALGORITHM; IT RETURNS IN P2 ALL THE
        PROCESSES WITH RANK OF ZERO ON R ; H INDICATES THE NUMBER OF PROCESSES
        WITH THE RANK     *)
     VAR I,J:INTEGER;
     BEGIN
       H:=-1;
       I:=FINDR(R);
       FOR J:=0 TO N DO
         IF PRTABLE[I,J].RNK=0 THEN
           BEGIN
            H:=H+1;
            P2[H]:=PROCTAB[J].PNAME
           END;
     END;  (*  HORIZONTAL   *)

   PROCEDURE VERTICAL(VAR VP,VR:INTEGER;VAR V:BOOLEAN);
       (* THE VERTICAL ALGORITHM; V IS TRUE IF VR EXISTS SUCH
          THAT VP'S RANK>0   *)
     VAR I,J:INTEGER;
     BEGIN
      I:=FINDP(VP);
      FOR J:=0 TO M DO
        IF (PRTABLE[J,I].RNK>0) AND (NOT MARKED[I]) THEN
           BEGIN
             V:=TRUE;
             VR:=LRESTAB[J].RNAME;
             MARKED[I]:=TRUE
           END;
   END;   (*   VERTIVCAL  *)
```

```
PROCEDURE HVDETECT(VAR PI,RJ,K:INTEGER;
                   DLCHECK:BOOLEAN);
  (* PROCEDURE PERFORMS THE HORIZONTAL AND
    VERTICAL SEARCH USING [RJ,PI] AS
    STARTING ENTRY IN THE TABLE. RETURNS
    PATH INFO IN PPATHS                *)
  VAR SW,DONE,V:BOOLEAN;
      I,H,P1,STKPTR : INTEGER;
  BEGIN
   DONE:=FALSE; STKPTR:=0; K:=0;
   FOR I:=0 TO NMAX DO PPATHS[I]:=-1;
   WHILE NOT DONE DO
    BEGIN
      HORIZONTAL(RJ,H); SW:=FALSE;
      IF DLCHECK THEN
       FOR I:=0 TO H DO
        IF P2[I]=PP THEN SW:=TRUE;
      IF SW THEN
         BEGIN
          MSGTEMP.DEADLOCK:=TRUE;
          DONE:=TRUE
         END ELSE
        BEGIN
         WHILE H>=0 DO
          BEGIN
            STK[STKPTR]:=P2[H];
            STKPTR:=STKPTR+1;
            H:=H-1
          END;
         V:=FALSE;
         WHILE (STKPTR>0) AND (NOT V) DO
          BEGIN
           STKPTR:=STKPTR-1;
           P1:=STK[STKPTR];
           VERTICAL(P1,RJ,V);
           IF NOT V THEN
             BEGIN (* ADD P1 TO PPATHS *)
              K:=K+1;
              PPATHS[K]:=P1
             END;
          END;
         IF (STKPTR=0) AND (NOT V) THEN
            DONE:=TRUE;
       END;
      END;
 END;   (*  HVDETECT  *)

 PROCEDURE DISEARCH;
  (*  SEARCH DISJOINT PATHS  *)
  LABEL 1;
  VAR I,J,K,L,I1,J1,L1,PS,RS:INTEGER;
      DLCHECK,SW:BOOLEAN;
  BEGIN
```

```
   FOR I:=0 TO N DO
    BEGIN
     FOR J:=0 TO M DO
      IF (PRTABLE[J,I].RNK>0) AND (NOT MARKED[I])
        THEN
       BEGIN
        PS:=PROCTAB[I].PNAME;
        RS:=LRESTAB[J].RNAME;
        DLCHECK:=FALSE; J1:=PS;
        MARKED[I]:=TRUE;
        HVDETECT(PS,RS,K,DLCHECK);
        (* SET DISJOINT PATH *)
        L:=1;
        WHILE MSGTEMP.DISJOINT[L].DIDENT<>-1 DO
          L:=L+1;
        MSGTEMP.DISJOINT[L].DIDENT:=J1;
        J1:=1;
        MSGTEMP.DISJOINT[L].DPROCS[1]:=PPATHS[1];
        FOR L1:=2 TO K DO
          BEGIN
           PS:=PPATHS[L1]; SW:=FALSE;
           FOR I1:=1 TO J1 DO
            IF PS=MSGTEMP.DISJOINT[L].DPROCS[I1] THEN
              SW:=TRUE;
           IF NOT SW THEN
             BEGIN
               J1:=J1+1;
               MSGTEMP.DISJOINT[L].DPROCS[J1]:=PS
             END;
          END;
         MSGTEMP.DISJOINT[L].DPROCS[0]:=J1;
         GOTO 1
        END;
    1:
   END;
   MSGTEMP.DISJOINT[0].DIDENT:=L;
 END;   (*  DISEARCH   *)


PROCEDURE HVINIT;
   (* INITIATES DETECTION ALG  *)
 LABEL 1;
 VAR I,J,K,L,PS,RS: INTEGER;
     DLCHECK,SW:BOOLEAN;
 BEGIN
   MSGTEMP.DEADLOCK:=FALSE;
   FOR I:=0 TO N DO
     MARKED[I]:=FALSE;
   FOR I:=0 TO N DO
     MSGTEMP.DISJOINT[I].DIDENT:=-1;
   PS:=PP;  RS:=RR; DLCHECK:=TRUE;
   HVDETECT(PS,RS.K,DLCHECK);
   IF MSGTEMP.DEADLOCK THEN GOTO 1;
```

```
  (*  SET MSGTEMP.DPATHS TO FPATHS *)
 J:=1;
 MSGTEMP.DPATHS[1]:=PPATHS[1];
 FOR L:=2 TO K DO
   BEGIN
    PS:=PPATHS[L]; SW:=FALSE;
    FOR I:=1 TO J DO
      IF PS=MSGTEMP.DPATHS[I] THEN
          SW:=TRUE;
    IF NOT SW THEN
      BEGIN
         J:=J+1;
         MSGTEMP.DPATHS[J]:=PS
      END;
    END;
  MSGTEMP.DPATHS[0]:=J;
  L:=FINDR(RR);
  FOR I:=0 TO N DO
   IF (PRTABLE[L,I].RNK>0) AND (NOT MARKED[I])
     THEN MARKED[I]:=TRUE;
  DISEARCH;
 1:
END;   (*  HVINIT  *)



PROCEDURE DTECTCONT;
    (* OTHER SITES RUN THIS  *)
  LABEL 1;
  VAR I,J,K,L,PS,RS,DL,LAST:INTEGER;
      PTEMP:ARRAY[0..NMAX] OF INTEGER;
      DLCHECK,V: BOOLEAN;

  PROCEDURE INSERTPATH(P:INTEGER);
    VAR DUPLICATE:BOOLEAN;
        MDL:INTEGER;
    BEGIN
      DUPLICATE:=FALSE;
      IF DL<>0 THEN
         FOR MDL:=1 TO DL DO
          IF MSGTEMP.DPATHS[MDL]=P THEN
             DUPLICATE:=TRUE;
      IF NOT DUPLICATE THEN
         BEGIN
          DL:=DL+1;
          MSGTEMP.DPATHS[DL]:=P
         END;
     END;


  BEGIN
   DLCHECK:=FALSE;
   FOR I:=0 TO M DO
```

```
 IF LRESTAB[I].RSTATUS<>FREE THEN
    DLCHECK:=TRUE;
IF NOT DLCHECK THEN GOTO 1;
PP:=MSGTEMP.PROCNAME;
FOR I:=0 TO N DO
   MARKED[I]:=FALSE;
K:=0; DL:=0; LAST:=MSGTEMP.DPATHS[0];
FOR I:=1 TO LAST DO
  BEGIN
   K:=K+1;
   PTEMP[K]:=MSGTEMP.DPATHS[I]
  END;
FOR I:=1 TO K DO
 BEGIN
   PS:=PTEMP[I];
   IF NEWP(PS) THEN
     INSERTPATH(PS) ELSE
   BEGIN
     V:=FALSE;
     VERTICAL(PS,RS,V);
     IF NOT V THEN INSERTPATH(PS) ELSE
       BEGIN
         DLCHECK:=TRUE;
         HVDETECT(PS,RS,L,DLCHECK);
         IF MSGTEMP.DEADLOCK THEN GOTO 1;
         FOR J:=1 TO L DO
          BEGIN
           PS:=PPATHS[J];
           INSERTPATH(PS)
          END;
        END;
    END;
  END;
  MSGTEMP.DPATHS[0]:=DL;
  DISEARCH;
 (*   UNION OF SUPPATHS  *)
 K:=DL;  DL:=0;
 FOR I:=1 TO K DO
   PTEMP[I]:=MSGTEMP.DPATHS[I];
 FOR I:=1 TO K DO
  BEGIN
     PS:=PTEMP[I];
    FOR J:=1 TO N DO
    IF PS=MSGTEMP.DISJOINT[J].DIDENT THEN
     BEGIN
      L:=MSGTEMP.DISJOINT[J].DPROCS[0];
      FOR LAST:=1 TO L DO
       BEGIN
        PS:=MSGTEMP.DISJOINT[J].DPROCS[LAST];
        IF PS=PP THEN
           BEGIN
            MSGTEMP.DEADLOCK:=TRUE;
            GOTO 1
```

```
            END;
           INSERTPATH(PS)
          END;
          MSGTEMP.DISJOINT[J].DIDENT:=-1;
          MSGTEMP.DISJOINT[J].DPROCS[0]:=-1
       END ELSE
       INSERTPATH(PS);
     END;
   MSGTEMP.DPATHS[0]:=DL;
   1:
   IF MSGTEMP.DEADLOCK THEN
     MSGTEMP.MSGDEST:=MSGTEMP.MSGORIGIN ELSE
     MSGTEMP.MSGDEST:=(MYSITE MOD NSITES) +1;
   OQUEUE.OUTPUTT(MSGTEMP);
 END;   (*  DTECTCONT  *)

(***********   END OF H&V ALGORITHM  **********)

   PROCEDURE SENRESPONSE;
      (* SENDS OUT RESPONSE TO REQUESTING PROCESSES  *)
    BEGIN
      IFR:=FINDR(RR); JFP:=FINDP(PP);
      LRESTAB[IFR].RSTATUS:=REQACCESS;
      PRTABLE[IFR,JFP].RNK:=0;
      PRTABLE[IFR,JFP].TACCES:=REQACCESS;
      PROCTAB[JFP].PSTATE:=RUNNING;
      WITH MSGTEMP DO
          BEGIN
            MSGTYPE:=ARESPONSE;
            MSGDEST:=MSGORIGIN;
            MSGORIGIN:=MYSITE;
            DPATHS[0]:=-1
          END;
      IF MSGTEMP.MSGDEST=MYSITE THEN
      PBUF[MSGTEMP.PROCNAME].PRBUFPUT(MSGTEMP) ELSE
      OQUEUE.OUTPUTT(MSGTEMP)
      (* IO.MESS7(MSGTEMP.MSGDEST,MSGTEMP.PROCNAME,RR)  *)
    END;

   PROCEDURE SENDROLLB:
        (*  SENDS ROLLBACK MESSAGE  *)
     BEGIN
       TOTDEAD:=TOTDEAD+1;
       WITH MSGTEMP DO
          BEGIN
            MSGTYPE:=ROLLBACK;
            MSGDEST:=MSGORIGIN;
            MSGORIGIN:=MYSITE:
            DPATHS[0]:=-1
           END;
        IF MSGTEMP.MSGDEST=MYSITE THEN
            PBUF[MSGTEMP.PROCNAME].PRBUFPUT(MSGTEMP) ELSE
        OQUEUE.OUTPUTT(MSGTEMP);
```

```
    IO.MESS10(MSGTEMP.MSGDEST,MSGTEMP.PROCNAME,RR);
    IO.MESS8(MYSITE,TOTDEAD,TOTREQ);
    ROLLB
  END;   (*   SEND ROLLBACK   *)

  PROCEDURE RANK;
    (* ASSIGNS A RANK TO A REQUESTING PROCESS *)
    (*  RERANK WILL REASSIGN THE RANKS IF NECESSARY  *)
   VAR K,L:INTEGER;
   BEGIN
    K:=-1;
    FOR L:=0 TO N DO
      IF PRTABLE[IFR,L].RNK>K THEN K:=PRTABLE[IFR,L].RNK;
    WITH PRTABLE[IFR,JFP] DO
       BEGIN
         RNK:=K+1;
         TACCES:=REQACCESS
       END;
   END;   (*   RANK  *)

PROCEDURE RERANK(THELD:STATUS);
    (* RESOURCE RR IS BEING HELD THELD  *)
     (*  REASSIGNS A RANK TO THE NEW REQUEST IF THE REQUEST IS
          FOR SHARED ACCESS  *)
  VAR WAITEXCL,WAITSHARED,SW:BOOLEAN;
      I,K:INTEGER;
  BEGIN
    WAITEXCL:=FALSE;  WAITSHARED:=FALSE;
    IF REQACCESS=SHARED THEN
       BEGIN
         FOR I:=0 TO N DO
           IF I<>JFP THEN
            WITH PRTABLE[IFR,I] DO
              BEGIN
                IF (RNK>0) AND (TACCES=EXCLUSIVE) THEN
                     WAITEXCL:=TRUE;
                IF (RNK>0) AND (TACCES=SHARED) THEN
                     WAITSHARED:=TRUE
              END;
          SW:=FALSE;
          IF (THELD=EXCLUSIVE) AND (WAITEXCL) AND (WAITSHARED)
               THEN SW:=TRUE ELSE
          IF (THELD=SHARED) AND (WAITEXCL) THEN SW:=TRUE;
          IF SW THEN FOR I:=0 TO N DO
             WITH PRTABLE[IFR,I] DO
                 IF (RNK>0) AND (TACCES=SHARED) AND (I<>JFP) THEN
                    PRTABLE[IFR,JFP].RNK:=RNK;
       END;
  END;   (*  RERANK  *)

PROCEDURE RESFREE(VAR RFREE:BOOLEAN; VAR THELD:STATUS);
   VAR I:INTEGER;
       SW:BOOLEAN;
```

```
    BEGIN
      RFREE:=FALSE;
      THELD:=LRESTAB[IFR].RSTATUS;
      IF THELD=FREE THEN RFREE:=TRUE ELSE
         IF (THELD=SHARED) AND (REQACCESS=SHARED) THEN
         BEGIN
            (* CHECK IF THERE IS ANY PROCESS WAITING ON RR
               FOR EXCLUSIVE ACCESS  *)
            SW:=FALSE;
            FOR I:=0 TO N DO
              IF (PRTABLE[IFR,I].RNK>0) AND (PRTABLE[IFR,I].TACCES=
                   EXCLUSIVE) THEN SW:=TRUE;
            IF NOT SW THEN RFREE:=TRUE
          END;
  END;  (*   RESFREE  *)

PROCEDURE RESREQ;
     (* PROCESS PP REQUEST FOR RESOURCE RR  *)
  VAR I,J:INTEGER;
      RFREE:BOOLEAN;
      THELD:STATUS;
  BEGIN
    TOTREQ:=TOTREQ+1;
    IF NEWP(PP) THEN
      INSERTP ELSE JFP:=FINDP(PP);
    IFR:=FINDR(RR);
    J:=0;
    WITH PROCTAB[JFP] DO
      BEGIN
       WHILE RHELD[J].RNAM<>-1 DO J:=J+1;
       RHELD[J].RNAM:=RR;
       RHELD[J].RACC:=REQACCESS
     END;
    RESFREE(RFREE,THELD);
    IF RFREE THEN SENRESPONSE ELSE
      BEGIN
        PROCTAB[JFP].PSTATE:=BLOCKED;
        NINITD:=NINITD+1;
        RANK;
        HVINIT;
        IF NOT MSGTEMP.DEADLOCK THEN
           BEGIN    (* SEND PATH OUT  *)
             IF REQACCESS=SHARED THEN RERANK(THELD);
             WITH MSGTEMP DO
               BEGIN
                 PROCNAME:=PP;
                 MSGTYPE:=DETEK;
                 MSGORIGIN:=MYSITE;
                 MSGDEST:=(MYSITE MOD NSITES)+1
               END;
             OQUEUE.OUTPUTT(MSGTEMP)
           END  ELSE  SENDROLLB
      END;
```

```
  END;   (*  RESREQ  *)

  PROCEDURE DTECTEND;
(*  SITE THAT INITIATED THE DETECTION ALG
   RECEIVES FINAL MESSAGE  *)
      BEGIN
        WITH MSGTEMP DO
          BEGIN
             RR:=RESNAME;
             PP:=PROCNAME;
             REQACCESS:=ACESTYPE
          END;
        IF NOT NEWP(PP) THEN
     BEGIN
        IFR:=FINDR(RR);
        JFP:=FINDP(PP);
        MSGTEMP.MSGORIGIN:=PROCTAB[JFP].PSITE;
        MSGTEMP.PROCNAME:=PP MOD 1000;
        (*  CHECK IF PP HAD BEEN ALLOCATED RR DUE TO A RELEASE AFTER
            THE DETECTION PATH WAS SENT OUT  *)
        IF PRTABLE[IFR,JFP].RNK>0 THEN
          BEGIN
          IF MSGTEMP.DEADLOCK THEN SENDROLLB ELSE
             BEGIN (* SEND WAIT FOR RESOURCE MSG *)
               WITH MSGTEMP DO
                BEGIN
                  MSGTYPE:=NOTFREE;
                  MSGDEST:=MSGORIGIN;
                  MSGORIGIN:=MYSITE
                END;
               IF MSGTEMP.MSGDEST=MYSITE THEN
                   PBUF[MSGTEMP.PROCNAME].PRBUFPUT(MSGTEMP) ELSE
               OQUEUE.OUTPUTT(MSGTEMP);
               (* IO.MESS11(MSGTEMP.MSGDEST,MSGTEMP.PROCNAME,RR);  *)
            END;
         END;
      END;
  END;   (*  DTECTEND  *)

  PROCEDURE MANAGER;
    VAR I,J:INTEGER;
    BEGIN
      CASE TENTRY OF
         REQ: RESREQ;
         REL:BEGIN
              JFP:=FINDP(PP);
              IF RR=-1 THEN ROLLB
                       ELSE BEGIN
                     IFR:=FINDR(RR);
                     RESREL  END;
             END;
         DETEC: IF MSGTEMP.MSGORIGIN=MYSITE THEN
                    DTECTEND ELSE DTECTCONT
```

```
      END;  (*  CASE  *)
    END;  (*  MANAGER  *)


(*****************************************************)
(*                                                   *)
(*       RANDOM NUMBER GENERATORS                    *)
(*                                                   *)
(*****************************************************)

    FUNCTION RAND(VAR SEED:REAL; MODPNTEGER):INTEGER;
      CONST
        P=2147483647;
        A=16807;
      VAR ISEED:INTEGER;
      BEGIN
        ISEED:=TRUNC(SEED);
        SEED:=(A*ISEED) MOD P;
        ISEED:=TRUNC(SEED) MOD MODP;
        RAND:=ISEED
      END;  (*  RAND  *)

    FUNCTION RANDOM(VAR S:REAL):REAL;
      VAR ISEED:INTEGER;
      BEGIN
       ISEED:=TRUNC(S);
       ISEED:=(ISEED*899) MOD 32767;
       S:=ISEED;
       RANDOM:=S/32767.0
     END;   (*   RANDOM  *)


(*************************************************)
(*                                               *)
(*   E N D    O F    R O U T I N E S             *)
(*                                               *)
(*************************************************)


  PROCESS WRITER(OUTLINE:LINE; SITE,TOTMAXF:INTEGER);
    (*  WRITE MSG TO OUTPUT LINE  *)
   VAR M:MESSAGE;
       WRITING:BOOLEAN;
       TOTL,TOTMSGSENT :INTEGER;
   BEGIN
    WRITING:=TRUE;
    TOTMSGSENT:=0;
    TOTL:=0;
    WHILE WRITING DO
      BEGIN
        OQUEUE.OUTGET(M);
        OUTLINE.TOLINE(M);
        TOTMSGSENT:=TOTMSGSENT+1;
```

```
      IF M.MSGTYPE=ATERMINATE THEN TOTL:=TOTL+1;
      IF TOTL=TOTMAXP THEN WRITING:=FALSE
    END;
 IO.MESS1(SITE,TOTMSGSENT)
END;   (*  WRITER  *)


 PROCESS READER(INLINE:LINE;SITE,MAXP:INTEGER);
  (* MONITOR INPUT LINE FOR ALL INCOMING MESSAGES;
     IF MSG IS FOR A LOCAL PROCESS IT WAKES UP THE
     PROCESS TO ACCEPT THE RESPONSE; NOTE THAT THE
     KERNEL CAN ALSO WAKE UP A LOCAL PROCESS IF THE
     REQUEST MADE IS FOR A LOCAL RESOURCE; IF THE
     MSG IS FOR A RESOURCE REQUEST, CHECKS IF THE
     REQUESTED RESOURCE IS LOCAL; IF LOCAL PUTS THE
     MSG IN MSGQUEUE FOR THE KERNEL TO PROCESS; IF
     NOT IT PUTS IT IN OUTBUFFER TO BE PASSED ON;
     IF THE MSG IS A DETECTION MSG OR RESOURCE
     RELEASE FOR A LOCAL RESOURCE IT PUTS IT IN
     MSGQUEUE                                      *)

  VAR MESG:MESSAGE;
      I,RTOTL,TOTMSGRECVD:INTEGER;
      SW,READING:BOOLEAN;
   BEGIN
     READING:=TRUE;
     RTOTL:=0;
     TOTMSGRECVD:=0;
     WHILE READING DO
       BEGIN
         INLINE.FRLINE(MESG);
         TOTMSGRECVD:=TOTMSGRECVD+1;
         CASE MESG.MSGTYPE OF
           AREQUEST:
             BEGIN
               SW:=FALSE;
               FOR I:=0 TO M DO
                IF LRESTAB[I].RNAME=MESG.RESNAME THEN SW:=TRUE;
               IF SW THEN MQUEUE.QUEPUT(MESG) ELSE
                          OQUEUE.OUTPUTT(MESG)
             END;

            ARESPONSE,ROLLBACK,NOTFREE:
               BEGIN
                  IF MESG.MSGDEST=SITE THEN
                    PBUF[MESG.PROCNAME].PRBUFPUT(MESG) ELSE
                    OQUEUE.OUTPUTT(MESG)
               END;

             COMPLETION,DETEK:
              IF MESG.MSGDEST=SITE THEN
                 MQUEUE.QUEPUT(MESG) ELSE
                 OQUEUE.OUTPUTT(MESG);
```

```
          ATERMINATE:
             BEGIN
               RTOTL:=RTOTL+1;
               IF MESG.MSGORIGIN<>SITE THEN MQUEUE.QUEPUT(MESG);
               IF RTOTL=MAXP THEN
                  READING:=FALSE
             END;
        END; (*  CASE *)
    END;  (*  WHILE READING *)
    IO.MESS2(SITE,TOTMSGRECVD);
END;  (*  READER  *)

PROCESS KERNEL(SITE:SITES;MAXR,TOTMAXP:INTEGER);
 (* HANDLES RESOURCE ALLOCATION AT EACH SITE
        IT RUNS THE DETECTION ALGORITHM  *)
   VAR KTOTL,I,QSIZE,TOTLOC : INTEGER;
       KERNELLING,SW :BOOLEAN;
   BEGIN
     KERNELLING:=TRUE;
     KTOTL:=0;  TOTLOC:=0;
     WHILE KERNELLING DO
       BEGIN
           MQUEUE.QUEGET(MSGTEMP,QSIZE);
          CASE MSGTEMP.MSGTYPE OF
           ATERMINATE:
              BEGIN
               KTOTL:=KTOTL+1;
               OQUEUE.OUTPUTT(MSGTEMP)
              END;

           LOCALL:
              BEGIN
                  MSGTEMP.QUESIZE:=QSIZE;
                  MSGTEMP.MSGTYPE:=AREQUEST;
                  SW:=FALSE;
                  FOR I:=0 TO M DO
                     IF MSGTEMP.RESNAME=LRESTAB[I].RNAME THEN
                                    SW:=TRUE;
                  IF NOT SW THEN
                     OQUEUE.OUTPUTT(MSGTEMP) ELSE
                       BEGIN
                        WITH MSGTEMP DO
                          BEGIN
                           PP:=MSGORIGIN*1000+PROCNAME;
                           RR:=RESNAME;
                           REQACCESS:=ACESTYPE
                          END;
                        TENTRY:=REQ;
                        TOTLOC:=TOTLOC+1;
                        MANAGER
                       END
```

```
              END;

          DETEK:
           BEGIN
             TENTRY:=DETEC;
             MANAGER
           END;

          AREQUEST,COMPLETION:
            BEGIN
              WITH MSGTEMP DO
               BEGIN
                QUESIZE:=QUESIZE+QSIZE;
                PP:=MSGORIGIN*1000+PROCNAME;
                IF RESNAME=-1 THEN RR:=-1
                     ELSE RR:=RESNAME;
                REQACCESS:=ACESTYPE;
                IF MSGTYPE=AREQUEST THEN
                   TENTRY:=REQ ELSE
                   TENTRY:=REL
              END;
              MANAGER
             END;
        END;  (*  CASE  *)
        IF KTOTL=TOTMAXP THEN KERNELLING:=FALSE;
       END;  (*  WHILE  *)
       IO.MESS8(SITE,TOTDEAD,TOTREQ);
       IO.MESS14(SITE,TOTLOC,NINITD)
   END;  (*  KERNEL  *)

PROCESS PPROCSS(SITE,LPROCID,TOTMAXR:INTEGER;LAMDA,MUU:REAL;
                MAXREQ,WACCES,THRUPUT : INTEGER);
    (* SIMULATE A LOCAL PROCESS ACTIVITIES  *)
  LABEL 1,2;
  TYPE
    LRES=RECORD
     LRNAME:INTEGER;
     TACCESS:STATUS;
     LOCATION:INTEGER
    END;
 VAR
   RESRCES:ARRAY[1..10] OF LRES;
   CLOCK,TRELEASE,TREQUEST,LAMDABAR,MUUBAR,SEEDR,SEED:REAL;
   TEMP,T2:REAL;
   NUMRES,RR,MP,I,J,TOTSENT,TOTDELAY,RELPTR,REQPTR :INTEGER;
   OUTREQ,THRUBEFORE,THRUAFTER : INTEGER;
   TESTCASE,TS,TD,MPPP : INTEGER;
   MAINSW,SW,SW1,GREATR,PROCESING,AGAN :BOOLEAN;
   MYMSG:MESSAGE;
   ACCTYPE:STATUS;

 PROCEDURE GENREQ;
     BEGIN (* GENERATE NEW RESOURCE  *)
```

```
  SW:=FALSE;
  WHILE NOT SW DO
    BEGIN
      RR:=RAND(SEED,TOTMAXR)+1;
      IF (REQPTR=0) OR (OUTREQ=0) THEN
        SW:=TRUE ELSE
        BEGIN
         SW1:=FALSE;
         J:=(RELPTR MOD 10)+1;
         FOR I:=1 TO OUTREQ DO
          BEGIN
            IF RESRCES[J].LRNAME=RR THEN SW1:=TRUE;
            J:=(J MOD 10)+1
          END;
         IF NOT SW1 THEN SW:=TRUE
        END
       END;
     (* TYPE OF ACCESS  *)
     IF WACCES=1 THEN ACCTYPE:=EXCLUSIVE ELSE
       BEGIN
         TEMP:=RANDOM(SEEDR);
         IF TEMP>=0.5 THEN ACCTYPE:=EXCLUSIVE ELSE
                           ACCTYPE:=SHARED
       END;
     REQPTR:=(REQPTR MOD 10)+1;  OUTREQ:=OUTREQ+1;
     RESRCES[REQPTR].LRNAME:=RR;
     RESRCES[REQPTR].TACCESS:=ACCTYPE;
(* SEND REQUEST  *)
 WITH MYMSG DO
  BEGIN
   MSGORIGIN:=SITE; PROCNAME:=LPROCID;
   QUESIZE:=0;   DPATHS[0]:=-1;
   MSGTYPE:=LOCALL; RESNAME:=RR;
   ACESTYPE:=ACCTYPE
  END;
 IO.MESS3(SITE,LPROCID,RR,ACCTYPE);
 MQUEUE.QUEPUT(MYMSG);
 J:=TIME; TOTSENT:=TOTSENT+1;
 PBUF[LPROCID].PRBUFGET(MYMSG);
 (* PROCESS BLOCKED WAITING FOR RESPONSE  *)
  TD:=TIME-J;            MPPP:=MYMSG.QUESIZE;
 IO.MESS13(SITE,LPROCID,TD,MPPP);
 IF MYMSG.MSGTYPE=ROLLBACK THEN
   BEGIN
    (* IO.MESS4(SITE,LPROCID,MYMSG.RESNAME); *)
    RESRCES[REQPTR].LOCATION:=MYMSG.MSGORIGIN;
    REQPTR:=REQPTR-1;
    IF (REQPTR=0) OR (REQPTR=-1) THEN REQPTR:=10;
    OUTREQ:=OUTREQ-1;
    MP:=MYMSG.MSGORIGIN; AGAN:=TRUE
   END  ELSE
BEGIN
     IF MYMSG.MSGTYPE=NOTFREE THEN
```

```
            PBUF[LPROCID].PRBUFGET(MYMSG);
         (* IO.MESS5(SITE,LPROCID,MYMSG.RESNAME);  *)
         RESRCES[REQPTR].LOCATION:=MYMSG.MSGORIGIN;
      END;
   END;  (* GENREQ  *)

 PROCEDURE ASSREL;
   BEGIN
    RELPTR:=(RELPTR MOD 10)+1;  OUTREQ:=OUTREQ-1;
     WITH MYMSG DO
       BEGIN
        PROCNAME:=LPROCID;  MSGTYPE:=COMPLETION;
        MSGORIGIN:=SITE;
        MSGDEST:=RESRCES[RELPTR].LOCATION;
        RESNAME:=RESRCES[RELPTR].LRNAME; ACESTYPE:=FREE
        END
 END;  (* RELPTR  *)


 BEGIN
  TOTSENT:=0; TOTDELAY:=0;    PROCESING:=TRUE;
   TRELEASE:=0.0;  CLOCK:=0.0;
  SEEDR:=31415.0/SITE; SEED:=SITE; TREQUEST:=0.0;
  THRUBEFORE:=TIME;
  (*  RELPTR POINTS TO THE LAST RESOURCE RELEASED
      REQPTR POINTS TO THE LAST RESOURCE REQUESTED FOR  *)
1: RELPTR:=0;  REQPTR:=0;    GREATR:=FALSE;
   OUTREQ:=0;  AGAN:=FALSE;
    MAINSW:=FALSE;
  WHILE PROCESING DO
     BEGIN
      MP:=-1;
      GENREQ;
      IF MP<>-1 THEN GOTO 2;
      (* GENERATE TIME OF NEXT RELEASE  *)
      MUUBAR:=(-1.0/MUU)*LN(RANDOM(SEEDR));
      TRELEASE:=CLOCK+MUUBAR;
      (* GENERATE TIME OF NEXT REQUEST *)
      LAMDABAR:=(-1.0/LAMDA)*LN(RANDOM(SEEDR));
      TREQUEST:=CLOCK+LAMDABAR;
      MAINSW:=TRUE;
      WHILE MAINSW DO
       BEGIN
        IF TRELEASE>TREQUEST THEN TESTCASE:=1;
        IF TRELEASE=TREQUEST THEN TESTCASE:=2;
        IF TRELEASE<TREQUEST THEN TESTCASE:=3;
        CASE TESTCASE OF
          1:  (* TRELEASE>TREQUEST  *)
            BEGIN
               TEMP:=LAMDABAR*100.0;
               I:=TRUNC(TEMP);  T2:=I+0.49;
               IF TEMP>T2 THEN I:=I+1;
               DELAY(I);  CLOCK:=TREQUEST;
```

```
IF  (TOTSENT>=MAXREQ) THEN
  BEGIN
   AGAN:=FALSE; MP:=-1; PROCESING:=FALSE; GOTO 2
  END;
MUUBAR:=TRELEASE-TREQUEST;
  (* GENERATE REQUEST  *)
IF OUTREQ>=TOTMAXR THEN
 BEGIN
  (* REQUEST BUT RES HELD EQUALS MAX RES *)
   TRELEASE:=TREQUEST; ASSREL;
   IF MYMSG.MSGDEST=SITE THEN
   MQUEUE.QUEPUT(MYMSG)   ELSE
   OQUEUE.OUTPUTT(MYMSG);
   (* IO.MESS6(SITE,LPROCID,RESRCES[RELPTR].LRNAME);  *)
   MAINSW:=FALSE
END ELSE
BEGIN
 MP:=-1;  GENREQ;
 IF MP<>-1 THEN GOTO 2;
 (* GENERATE TIME OF NEXT REQUEST  *)
 LAMDABAR:=(-1.0/LAMDA)*LN(RANDOM(SEEDR));
 TREQUEST:=CLOCK+LAMDABAR;  MAINSW:=TRUE
END
END;  (* TESTCASE=1  *)
2: (* TRELEASE=TREQUEST  *)
  BEGIN
   CLOCK:=TRELEASE;
   TEMP:=LAMDABAR*100.0;
   I:=TRUNC(TEMP); T2:=I+0.49;
   IF TEMP>T2 THEN I:=I+1; DELAY(I);
   (* RELEASE RESOURCE IF ANY  *)
   IF OUTREQ>0 THEN
     BEGIN
      ASSREL;
       IF MYMSG.MSGDEST=SITE THEN MQUEUE.QUEPUT(MYMSG) ELSE
           OQUEUE.OUTPUTT(MYMSG)
  (* IO.MESS6(SITE,LPROCID,RESRCES[RELPTR].LRNAME)  *)
  END;
  MAINSW:=FALSE;
  IF  (TOTSENT>=MAXREQ) THEN
    BEGIN
     AGAN:=FALSE; MP:=-1; PROCESING:=FALSE; GOTO 2
    END;
END;
3:  (* TRELEASE<TREQUEST  *)
 BEGIN
  TEMP:=MUUBAR*100.0;
  I:=TRUNC(TEMP); T2:=I+0.49;
  IF TEMP>T2 THEN I:=I+1;   DELAY(I);
  IF OUTREQ<=0 THEN
   BEGIN (* NO RES TO RELEASE  *)
    CLOCK:=TREQUEST;
    TEMP:=(TREQUEST-TRELEASE)*100.0;
```

```
        I:=TRUNC(TEMP);  T2:=I+0.49;
        IF TEMP>T2 THEN I:=I+1;   DELAY(I);  MAINSW:=FALSE
        END  ELSE
        BEGIN  (* RELEASE  RESOURCE  *)
         CLOCK:=TRELEASE;  ASSREL;
         IF MYMSG.MSGDEST=SITE THEN MQUEUE.QUEPUT(MYMSG) ELSE
             OQUEUE.OUTPUTT(MYMSG);
         (* IO.MESS6(SITE,LPROCID,RESRCES[RELPTR].LRNAME);  *)
         LAMDABAR:=TREQUEST-TRELEASE;
         (* GENERATE TIMOF NEXT RELEASE *)
          MUUBAR:=(-1.0/MUU)*LN(RANDOM(SEEDR));
          TRELEASE:=CLOCK+MUUBAR;
          IF  (TOTSENT>=MAXREQ) THEN
            BEGIN
              AGAN:=FALSE;  MP:=-1;  PROCESING:=FALSE;
              GOTO 2
         END
    END
   END;  (* TRELEASE < TREQUEST  *)
  END;  (*  CASE  *)
  END; (*  MAINSW  *)
END;  (* PROCESING  *)
2: IF NOT AGAN THEN
     THRUAFTER:=TIME-THRUBEFORE;
   IF OUTREQ>0 THEN
     BEGIN
      TS:=-1;  TD:=-1;
      WHILE OUTREQ>0 DO
       BEGIN
        ASSREL;
        IF (MYMSG.MSGDEST<>MP) AND
           (MYMSG.MSGDEST<>TS) AND
           (MYMSG.MSGDEST<>TD) THEN
         BEGIN
          IF MP=-1 THEN MP:=MYMSG.MSGDEST
           ELSE IF TS=-1 THEN TS:=MYMSG.MSGDEST
           ELSE IF TD=-1 THEN TD:=MYMSG.MSGDEST;
          MYMSG.RESNAME:=-1;
          IF MYMSG.MSGDEST=SITE THEN
            MQUEUE.QUEPUT(MYMSG) ELSE
            OQUEUE.OUTPUTT(MYMSG)
         END;
       END;
     END;
      IF AGAN THEN
          BEGIN
            I:=1000;
            J:=RAND(SEED,I)+100;
            DELAY(J);
            IF THRUPUT=1 THEN TOTSENT:=0;
            GOTO 1
          END;
      MYMSG.MSGTYPE:=ATERMINATE;
```

```
          MYMSG.MSGDEST:=SITE;
          MQUEUE.QUEPUT(MYMSG);
        IO.MESS7(SITE,LPROCID,TOTSENT,THRUAFTER);
    END;   (*  PROCESS PPROCSS  *)


    ENTRY PROCEDURE STARTMACH(SITE:SITES;INLINE,OUTLINE:LINE;
         MAXR,STARTR,TOTMAXR,MAXP,TOTMAXP:INTEGER;
         LAMDA,MUU:REAL;MAXREQ,WACC,THRUP:INTEGER);
      VAR I,J,K,P:INTEGER;
      BEGIN
       N:=TOTMAXP-1;  M:=MAXR-1; TOTREQ:=0;
       TOTDEAD:=0;  NINITD:=0;  MYSITE:=SITE;
       INITIALIZE;
       (* INITIALISE RESOURCE FOR THIS SITE  *)
       J:=STARTR;
       FOR I:=0 TO MAXR-1 DO
         BEGIN
           WITH LRESTAB[I] DO
             BEGIN
               RNAME:=J;  RSTATUS:=FREE
             END;
           J:=J+1
         END;
         (*  START PROCESSES AT THIS SITE  *)
          FOR I:=1 TO MAXP DO
          PPROCSS(SITE,I,TOTMAXR,LAMDA,MUU,
                   MAXREQ,WACC,THRUP);
          KERNEL(SITE,MAXR,TOTMAXP);
          READER(INLINE,SITE,TOTMAXP);
          WRITER(OUTLINE,SITE,TOTMAXP);
      END;   (*   STARTMACH   *)

  END;    (*  ******   MACHINE   ******  *)

(*******************************************************************)
(*                                                                 *)
(*   S Y S T E M   A C T I V A T I O N                             *)
(*                                                                 *)
(*******************************************************************)

VAR
  NET:ARRAY[SITES] OF MACHINE;
  LINK:ARRAY[NLINES] OF LINE;
  TOTMAXP,TOTMAXR,I,K,MAXREQ,MAXR,MAXP NTEGER;
         (* TOTMAXP = TOTAL # PROCESSES IN NETWORK
            TOTMAXR = TOTAL # RESOURCES IN NETWORK
            MAXREQ = MAX # REQUESTS FOR EACH PROCESS
            THRUP=0 : STOP AFTER MAXREQ
                 =1 : RUN UNTIL ALL PROCESSES
             ACQUIRE MAXREQ
             WACC=0 IF BOTH EXCL & SHARED RESOURCE ARE ALLOWED
                 AND 1 IF ONLY EXCL  *)
```

```
  LAMDA,MUU,TEMP:REAL;
  J,L,Y,WACC,THRUP :INTEGER;
  RESDISTR:ARRAY[SITES] OF INTEGER;
  DISTPR:ARRAY[SITES] OF INTEGER;
  BEGIN
 READ(TOTMAXP,TOTMAXR,LAMDA,MUU,MAXREQ,WACC,THRUP);
    (*  DISTRIBUTE RESOURCES AMONG SITES  *)
    K:=0;  J:=0;
    L:=TOTMAXR DIV NSITES;
    Y:=TOTMAXP DIV NSITES;
    FOR I:=1 TO NSITES DO
     BEGIN
       RESDISTR[I]:=L;
       DISTPR[I]:=Y;
       K:=K+L;
       J:=J+Y
     END;
    I:=0;
    WHILE K<TOTMAXR DO
      BEGIN
        I:=I+1;
        RESDISTR[I]:=RESDISTR[I]+1;
        K:=K+1
      END;
    I:=0;
    WHILE J<TOTMAXP DO
      BEGIN
        I:=I+1;
        DISTPR[I]:=DISTPR[I]+1;
        J:=J+1
      END;
    WRITELN(' DISTRIBU','TED H&V');
    WRITELN(' NO OF ','RESOURCES',' = ',TOTMAXR);
    WRITELN(' NO OF ','PROCESSES',' = ',TOTMAXP);
    WRITELN(' MUU = ',MUU);
    WRITELN(' LAMDA = ',LAMDA);
    WRITELN(' MAXIMUM ','REQUEST =',MAXREQ);
    NET[1].STARTMACH(1,LINK[3],LINK[1],RESDISTR[1],1,
    TOTMAXR,DISTPR[1],TOTMAXP,LAMDA,MUU,MAXREQ,WACC,THRUP);
    K:=RESDISTR[1]+1;
    NET[2].STARTMACH(2,LINK[1],LINK[2],RESDISTR[2],K,
    TOTMAXR,DISTPR[2],TOTMAXP,LAMDA,MUU,MAXREQ,WACC,THRUP);
    K:=K+RESDISTR[2];
    NET[3].STARTMACH(3,LINK[2],LINK[3],RESDISTR[3].K,
    TOTMAXR,DISTPR[3],TOTMAXP,LAMDA,MUU,MAXREQ,WACC,THRUP);
END.
```

APPENDIX C

Program Listing for the Implementation of Goldman's Distributed Algorithm on a 3-Site Network

```
PROGRAM GOLDM(INPUT,OUTPUT);

    (*****************************************************)
    (*                                                   *)
    (*    GOLDMAN'S DEADLOCK DETECTION ALGORITHM         *)
    (*                                                   *)
    (*****************************************************)

    CONST
      NSITES=3;         (* 3 SITE NETWORK  *)
      BMAX=12;          (* BUFFER SIZE     *)
      NMAX=4;          (* MAXIMUM NUMBER OF PROCESSES  TO RUN ON EACH NODE *)
      MMAX=2;           (* MAXIMUM NUMBER OF RESOURCES A EACH SITE *)
      LINES=3;
      N=7;      (* MAX PROCESSES IN NETWORK  *)
      M=6;      (* MAX RESOURCES IN NETWORK  *)
    TYPE
      MESSTYPE=(AREQUEST,ARESPONSE,COMPLETION,ROLLBACK,LOCALL,
               INITDEAD,DLOCK,NFREE,NOTFREE,DETEK,ATERMINATE);
        (* DLOCK MSG IS SENT TO THE SITE THAT INITIATED DETECTION
           IF DEADLOCK IS DETECTED ALONG THE WAY BY ANOTHER SITE
           NFREE MSG IS TO BE SENT IF NO DEADLOCK IS DETECTED    *)
           (* INITDEAD MSG IS SENT TO THE SITE THE REQ.
           PROCESS RESIDES TO INITIATE DETECTION ALG *)
      SITES=1..NSITES;
      STATUS=(FREE,EXCLUSIVE,SHARED);
      NLINES=1..LINES;
      OBPLREC=RECORD
          OBPLRNAME:INTEGER;   (* RES NAME  *)
          OBPLOWNER:INTEGER;   (* LOCATION OF RESOURCE *)
          PROCNODE : INTEGER;  (* LOCATION OF REQUESTING PROCESS *)
          OBPLPROCS:ARRAY[0..N] OF INTEGER
      END;

      MESSAGE=RECORD
        MSGTYPE:MESSTYPE;
        MSGORIGIN:INTEGER;
        MSGDEST:INTEGER;
        PROCNAME:INTEGER;
        RESNAME:INTEGER;
        ACESTYPE:STATUS;
        QUESIZE : INTEGER;
        OBPL : OBPLREC
      END;

      PROCIO=OBJECT
        PATH 1:( 1:(MESS1),1:(MESS2), 1:(MESS3),1:(MESS4),
               1:(MESS5),1:(MESS6),1:(MESS7),1:(MESS8),1:(MESS9),
          1:(MESS10),1:(MESS11),1:(MESS12),1:(MESS13),1:(MESS14),1:(MESS15) ) END;

        ENTRY PROCEDURE MESS1(I,J:INTEGER);
```

```
  VAR K:INTEGER;
  BEGIN
    K:=(J*100+I)*100;
    WRITELN(K)
  END;                                        (*  MESS1  *)

ENTRY PROCEDURE MESS2(I,J:INTEGER);
  VAR K:INTEGER;
  BEGIN
    K:=(J*100+I)*100+1;
    WRITELN(K)
  END;                                          (*  MESS2  *)

ENTRY PROCEDURE MESS3(I,J,K:INTEGER;L:STATUS);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100;
    IF L=EXCLUSIVE THEN T:=T+2 ELSE T:=T+3;
    WRITELN(T)
  END;                                           (*  MESS3  *)

ENTRY PROCEDURE MESS4(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
   T:=I*100000+J*10000+K*100+4;
   WRITELN(T)
  END;                                              (*  MESS4  *)

ENTRY PROCEDURE MESS5(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+5;
    WRITELN(T)
  END;                                             (*  MESS5  *)

ENTRY PROCEDURE MESS6(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+10;
    WRITELN(T)
  END;                                                 (*  MESS6  *)

ENTRY PROCEDURE MESS7(I,J,K,L :INTEGER);
  VAR T,T1,T2,T3 :INTEGER;
  BEGIN
    T:=I*1000+J*100+11;
    T1:=K*100000+I*1000+J*100+12;
    T3:=L DIV 100;
    T2:=T3*10000+I*100+J*10+8;
    WRITELN(T,T1,T2)
  END;                                                   (*  MESS7  *)

ENTRY PROCEDURE MESS8(I,J,K:INTEGER);
```

```
VAR T,T1 : INTEGER;
BEGIN
  T:=J*10000+I*100+13;
  T1:=K*10000+I*100+14;
  WRITELN(T,T1)
END;                                    (*  MESS8  *)

ENTRY PROCEDURE MESS9(I,J,K:INTEGER);
 VAR T:INTEGER;
 BEGIN
  T:=I*100000+J*10000+K*100+15;
  WRITELN(T)
 END;                                   (*  MESS9  *)

ENTRY PROCEDURE MESS10(I,J,K:INTEGER);
 VAR T:INTEGER;
 BEGIN
  T:=I*100000+J*10000+K*100+20;
  WRITELN(T)
 END;                                (*  MESS10  *)

ENTRY PROCEDURE MESS11(I,J,K:INTEGER);
 VAR T:INTEGER;
 BEGIN
  T:=I*100000+J*10000+K*100+21;
  WRITELN(T)
 END;                                    (*  MESS11  *)

ENTRY PROCEDURE MESS12(I,J,K:INTEGER);
 VAR T:INTEGER;
 BEGIN
  T:=I*100000+J*10000+K*100+22;
  WRITELN(T)
 END;                                    (* MESS12  *)
 ENTRY PROCEDURE MESS13(I,J,OU,QS :INTEGER);
     VAR T,T1,T3:INTEGER;
     BEGIN
       OU:=OU DIV 100;
       T:=I*100+J*10;
       T1:=OU*10000+I+7;
       T3:=QS*10000+T+9;
       WRITELN(T1,T3)
  END;                                (* MESS13  *)

ENTRY PROCEDURE MESS14(I,J,K :INTEGER);
     VAR T,T1 :INTEGER;
     BEGIN
       T:=J*10000+I*100+23;
       T1:=K*10000+I*100+25;
       WRITELN(T,T1)
     END;                           (*  MESS14  *)

ENTRY PROCEDURE MESS15(I,NF,DE,RES,ROL,COM,ARE,K1,K2,K3:INTEGER);
```

```
VAR T1,T2,T3,T4,T5,T6,T7,T8,T9,T1000,T100 : INTEGER;
BEGIN
 T1000:=10000;
 T100:=I*100;
 T1:=NF*T1000+T100+30;
 IF OE<>99 THEN BEGIN
                  T2:=OE*T1000+T100+31;
                  WRITE(T2)
                END;
 T3:=RES*T1000+T100+32;
 T4:=ROL*T1000+T100+33;
 T5:=COM*T1000+T100+34;
 T6:=ARE*T1000+T100+35;
 IF K1<>99 THEN BEGIN
                 T7:=K1*T1000+T100+36;
                 T8:=K2*T1000+T100+24;
                 T9:=K3*T1000+T100+25;
                 WRITE(T7,T8,T9)
                END;
  WRITELN(T1,T3,T4,T5,T6)
END;                                          (*  MESS15  *)

END;      (*  ******  PROCIO  ******  *)

LINE=OBJECT

 PATH 1:(TOLINE;FRLINE) END;
 VAR MESGBUF:MESSAGE;

 ENTRY PROCEDURE TOLINE(M:MESSAGE);
  BEGIN
   MESGBUF:=M
  END;      (*  TOLINE  *)

 ENTRY PROCEDURE FRLINE(VAR M:MESSAGE);
  BEGIN
   M:=MESGBUF
  END;      (*  FRLINE  *)

END;      (*  ******  LINE  ******  *)

MACHINE=OBJECT
 PATH STARTMACH END;

 TYPE
  MSGQUEUE=OBJECT   (* INPUT MSGES TO BE PROCESSED  *)
   PATH BMAX :(1:(QUEPUT);1:(QUEGET)) END;
   VAR QUEBUFFER:ARRAY[1..BMAX] OF MESSAGE;
       INQQ,OUTQQ:1..BMAX;

   ENTRY PROCEDURE QUEPUT(M:MESSAGE);
    BEGIN
```

```
      QUEBUFFER[INQQ]:=M;
      INQQ:=(INQQ MOD BMAX)+1
    END;   (*   QUEFUT   *)

  ENTRY PROCEDURE QUEGET(VAR M:MESSAGE;VAR QS:INTEGER);
    BEGIN
     M:=QUEBUFFER[OUTQQ];
     IF OUTQQ>INQQ THEN
        QS:=(BMAX-OUTQQ)+INQQ ELSE
        QS:=INQQ-OUTQQ;
     OUTQQ:=(OUTQQ MOD BMAX) + 1;
    END;   (*   QUEPUT   *)

 INIT; BEGIN
        INQQ:=1;
        OUTQQ:=1
       END;        (*  INIT  *)

END;    (*  ******  MSGQUEUE  ******  *)

OUTQUEUE=OBJECT    (*  MSGES TO BE SENT OUT  *)
  PATH BMAX:(1:(OUTPUTT);1:(OUTGET)) END;
  VAR OUTBUFFER:ARRAY[1..BMAX] OF MESSAGE;
      OUTP,OUTG:1..BMAX;

  ENTRY PROCEDURE OUTPUTT(M:MESSAGE);
     BEGIN
       OUTBUFFER[OUTP]:=M;
       OUTP:=(OUTP MOD BMAX) + 1
     END;   (*  OUTPUTT  *)

  ENTRY PROCEDURE OUTGET(VAR M:MESSAGE);
     BEGIN
       M:=OUTBUFFER[OUTG];
       OUTG:=(OUTG MOD BMAX )+1
     END;     (*  OUTGET   *)

  INIT; BEGIN
          OUTP:=1;
          OUTG:=1
        END;  (*  INIT  *)

END;   (*  ******  OUTQUEUE  ******  *)

PRBUF=OBJECT    (* PRIVATE BUFFER FOR EACH PROCESS*)
  PATH 1:(PRBUFPUT;PRBUFGET) END;
  VAR  PRBUFFER:MESSAGE;

  ENTRY PROCEDURE PRBUFPUT(M:MESSAGE);
    BEGIN
     PRBUFFER:=M
    END;   (* PRBUFFER  *)
```

```
    ENTRY PROCEDURE PRBUFGET(VAR M:MESSAGE);
      BEGIN
        M:=PRBUFFER
      END;  (*  PRBUFGET  *)

  END;  (*  ******  PRBUF  ******  *)

STATE=(BLOCKED,RUNNING);
RCURACCESS=RECORD
 RCURPNAME:INTEGER;    (* PROC NAME  *)
 RLOCATION:INTEGER;    (* LOCATION OF PROCESS  *)
 RCURACCTYPE:STATUS;  (* TYPE OF ACCESS        *)
 RINDEX : BOOLEAN
END;

LRESOURCE=RECORD
 RNAME:INTEGER;    (*RES NAME  *)
 RSTATUS:STATUS; (* STATUS OF RES  *)
 RPROC:ARRAY[0..N] OF RCURACCESS; (* PROCS CURRENTLY ACCESSING RNAME *)
 RPROCWAIT:ARRAY[0..N] OF RCURACCESS (* PROCS WAITING FOR RNAME *)
END;

PCURACCESS=RECORD
 PCURNAME:INTEGER; (* RESOURCE NAME  *)
 PROWNER:INTEGER;  (* LOCATION OF RESOURCE  *)
 PCURACCTYPE:STATUS  (* TYPE OF ACCESS  *)
END;

LPROCESS=RECORD
 PNAME:INTEGER;   (* PROCESS  NAME  *)
 PSTATE: STATE;   (* STATE OF PROCESS  *)
 PRESRCE:ARRAY[0..M] OF PCURACCESS; (* RES CURRENTLY ACCESSED BY PNAME *)
 PNEWREQ:PCURACCESS (* NEW REQ IF WAITING  *)
END;

VAR
  MQUEUE:MSGQUEUE;
  OQUEUE:OUTQUEUE;
  PBUF:ARRAY[1..5] OF PRBUF; (* MAX 5 PROCESSES PER SITE *)
  IO:PROCIO;
  PROCTAB:ARRAY[0..NMAX] OF LPROCESS;
  RESTAB:ARRAY[0..MMAX] OF LRESOURCE;
  DEADLOCK:BOOLEAN;
  PP,RR,IFR,IFP,ALGENTRY : INTEGER;
  REQACCESS:STATUS;
   MSGTEMP:MESSAGE;
   JJ,TOTDEAD,TOTREQ,NINITD : INTEGER;
   MYSITE:SITES;

(*************************************************************)
(*                                                           *)
(*                   DETECTION   ROUTINES                    *)
```

```
(*                                                                    *)
(********************************************************************)

PROCEDURE INITIALIZE;
  (* INITIALISE LOCAL PROCESS ^ RESOURCE TABLES  *)
  VAR
    I,J : INTEGER;
  BEGIN
    FOR I:=0 TO NMAX DO
      BEGIN
        PROCTAB[I].PNAME:=-1;
        PROCTAB[I].PSTATE:=BLOCKED;
        FOR J:=0 TO M DO
          BEGIN
            PROCTAB[I].PRESRCE[J].PCURNAME:=-1;
            PROCTAB[I].PRESRCE[J].PCURACCTYPE:=FREE;
            PROCTAB[I].PRESRCE[J].PROWNER:=-1
          END;
        PROCTAB[I].PNEWREQ.PCURNAME:=-1;
        PROCTAB[I].PNEWREQ.PCURACCTYPE:=FREE;
        PROCTAB[I].PNEWREQ.PROWNER:=-1
      END;
    FOR I:=0 TO MMAX DO
      BEGIN
        RESTAB[I].RNAME:=-1;
        RESTAB[I].RSTATUS:=FREE;
        FOR J:=0 TO N DO
          BEGIN
            RESTAB[I].RPROC[J].RCURPNAME:=-1;
            RESTAB[I].RPROC[J].RCURACCTYPE:=FREE;
            RESTAB[I].RPROC[J].RINDEX:=FALSE;
            RESTAB[I].RPROCWAIT[J].RCURPNAME:=-1;
            RESTAB[I].RPROCWAIT[J].RCURACCTYPE:=FREE;
            RESTAB[I].RPROCWAIT[J].RLOCATION:=-1;
            RESTAB[I].RPROCWAIT[J].RINDEX:=FALSE
          END
      END;
    TOTREQ:=0; TOTDEAD:=0;
  END; (*  INITIALIZE  *)

  FUNCTION LOCALP(P:INTEGER):BOOLEAN;
   (*  RETURNS TRUE IF P IS IN LOCAL SITE  *)
   VAR I : INTEGER;
   BEGIN
    LOCALP:=FALSE;
    FOR I:=0 TO NMAX DO
      IF PROCTAB[I].PNAME=P THEN LOCALP:=TRUE;
   END; (*  LOCALP  *)

   FUNCTION LOCALR(R : INTEGER):BOOLEAN;
    (* RETURNS TRUE IF R IS IN LOCAL SITE  *)
    VAR I : INTEGER;
    BEGIN
```

```
LOCALR:=FALSE;
FOR I:=0 TO MMAX DO
 IF RESTAB[I].RNAME=R THEN LOCALR:=TRUE;
END;

FUNCTION FINDP(P:INTEGER):INTEGER;
 (* RETURNS INDEX TO PROCESS IN PROCESS TABLE  *)
 VAR I:INTEGER;
 BEGIN
  I:=0;
  WHILE (PROCTAB[I].PNAME<>P) AND (I<=NMAX) DO I:=I+1;
  IF I>NMAX THEN BEGIN WRITELN(' ***ERR*** ',P); FINDP:=999
     END  ELSE FINDP:=I;
 END;  (* FINDP  *)

FUNCTION FINDR(R:INTEGER):INTEGER;
 (* RETURNS INDEX TO A RESOURCE TABLE  *)
 VAR I : INTEGER;
 BEGIN
   I:=0;
   WHILE (RESTAB[I].RNAME<>R) AND (I<=MMAX) DO I:=I+1;
   IF I>MMAX THEN BEGIN WRITELN(' ***ERR***',R); FINDR:=999
     END ELSE FINDR:=I;
END;

FUNCTION RESFREE(R:INTEGER):BOOLEAN;
 (* RETURNS TRUE IF R IS FREE  *)
VAR
   I,J : INTEGER;
   SW:BOOLEAN;
BEGIN
  I:=FINDR(R);  RESFREE:=FALSE;
  IF RESTAB[I].RSTATUS=FREE THEN RESFREE:=TRUE ELSE
     IF (RESTAB[I].RSTATUS=SHARED) AND (REQACCESS=SHARED)
          THEN BEGIN
            SW:=FALSE;
            FOR J:=0 TO N DO
             IF RESTAB[I].RPROCWAIT[J].RCURPNAME<>-1 THEN SW:=TRUE;
            IF NOT SW THEN  RESFREE:=TRUE
          END;
   END;

PROCEDURE INSERTP(P:INTEGER;VAR I:INTEGER);
  (* INSERTS LOCAL PROCESS IN PROCESS  *)
  BEGIN
   I:=0;
   WHILE (PROCTAB[I].PNAME<>-1) AND (I<=NMAX) DO I:=I+1;
   PROCTAB[I].PNAME:=P;
  END;

 PROCEDURE REMOVEPW(P:INTEGER);
   BEGIN
     IFP:=FINDP(P);
```

```
      WITH PROCTAB[IFP].PNEWREQ DO
        BEGIN
          PCURNAME:=-1; PROWNER:=-1 ;  PCURACCTYPE:=FREE
        END;
    END;

 PROCEDURE SENDRESPONSE;
  (* SENDS RESPONSE TO REQUESTING PROCESS  *)
  VAR J : INTEGER;
  BEGIN
   IFR:=FINDR(RR);  RESTAB[IFR].RSTATUS:=REQACCESS; J:=0;
   WHILE (RESTAB[IFR].RPROC[J].RCURPNAME<>-1) DO J:=J+1;
   RESTAB[IFR].RPROC[J].RCURPNAME:=PP;
   RESTAB[IFR].RPROC[J].RLOCATION:=MSGTEMP.MSGORIGIN;
   RESTAB[IFR].RPROC[J].RCURACCTYPE:=REQACCESS;
   IF MSGTEMP.MSGORIGIN=MYSITE THEN
      BEGIN
        IFP:=FINDP(PP); PROCTAB[IFP].PSTATE:=RUNNING;
        PROCTAB[IFP].PNEWREQ.PCURNAME:=-1;
        J:=0;
        WHILE PROCTAB[IFP].PRESRCE[J].PCURNAME<>-1 DO J:=J+1;
        WITH PROCTAB[IFP].PRESRCE[J] DO
           BEGIN
             PCURNAME:=RR;  PROWNER:=MYSITE; PCURACCTYPE:=REQACCESS
           END;
    END;
        WITH MSGTEMP DO
           BEGIN
             MSGTYPE:=ARESPONSE; MSGDEST:=MSGORIGIN;
             MSGORIGIN:=MYSITE
           END;
        IF MSGTEMP.MSGDEST=MYSITE THEN
          PBUF[MSGTEMP.PROCNAME].PRBUFPUT(MSGTEMP) ELSE
           OQUEUE.OUTPUTT(MSGTEMP);
        (* IO.MESS9(MSGTEMP.MSGDEST,MSGTEMP.PROCNAME,RR);  *)
   END;

 PROCEDURE PACKRW(I,J:INTEGER);
  VAR K,L:INTEGER;
    BEGIN
     L:=J;
     FOR K:=J+1 TO N DO
      BEGIN
       WITH RESTAB[I] DO
         BEGIN
          RPROCWAIT[L].RCURPNAME:=RPROCWAIT[K].RCURPNAME;
          RPROCWAIT[L].RLOCATION:=RPROCWAIT[K].RLOCATION;
          RPROCWAIT[L].RCURACCTYPE:=RPROCWAIT[K].RCURACCTYPE;
          RPROCWAIT[L].RINDEX:=RPROCWAIT[K].RINDEX
         END;
       L:=L+1
      END;
```

```
    WITH RESTAB[I].RPROCWAIT[N] DO
      BEGIN
       RCURPNAME:=-1;  RLOCATION:=-1; RCURACCTYPE:=FREE;
       RINDEX:=FALSE
      END;
   END;

  PROCEDURE UPTABLE;
   VAR I : INTEGER;
   BEGIN
    PP:=MYSITE*1000+MSGTEMP.PROCNAME;
    IFP:=FINDP(PP);
    PROCTAB[IFP].PSTATE:=RUNNING;
    PROCTAB[IFP].PNEWREQ.PCURNAME:=-1;
    I:=0;
    WHILE PROCTAB[IFP].PRESRCE[I].PCURNAME<>-1 DO  I:=I+1;
    WITH PROCTAB[IFP].PRESRCE[I] DO
      BEGIN
        PCURNAME:=MSGTEMP.RESNAME;
        PROWNER:=MSGTEMP.MSGORIGIN;
        PCURACCTYPE:=MSGTEMP.ACESTYPE
      END;
    PBUF[MSGTEMP.PROCNAME].PRBUFPUT(MSGTEMP)
  END;

  PROCEDURE RESREL;
   (* RES RR IS RELEASED ; ALLOCATE IT TO PROCESS IF ANY WAITING *)
   VAR I,J,K,L : INTEGER;
     SW,SW1:BOOLEAN;
     ACC:STATUS;
   BEGIN
    (* I:=PP MOD 1000;  J:=PP DIV 1000;  IO.MESS12(J,I,RR); *)
    IFR:=FINDR(RR);  J:=0;
    WHILE RESTAB[IFR].RPROC[J].RCURPNAME<>PP DO J:=J+1;
     WITH RESTAB[IFR].RPROC[J] DO
       BEGIN
        RCURPNAME:=-1; RLOCATION:=-1: RCURACCTYPE:=FREE;
        RINDEX:=FALSE
       END;
    (* CHECK IF ANY MORE PROCESS IS USING RR  *)
    SW:=FALSE;
    FOR J:=0 TO N DO
      IF RESTAB[IFR].RPROC[J].RCURPNAME<>-1 THEN SW:=TRUE;
    IF NOT SW THEN RESTAB[IFR].RSTATUS:=FREE;
    IF (NOT SW) AND (RESTAB[IFR].RPROCWAIT[0].RCURPNAME<>-1) THEN
      BEGIN
        SW1:=TRUE;  J:=0;
        WHILE SW1 DO
         BEGIN
          WITH MSGTEMP DO
           BEGIN
            MSGTYPE:=ARESPONSE;
            MSGORIGIN:=MYSITE;
```

```
          MSGDEST:=RESTAB[IFR].RPROCWAIT[J].RLOCATION;
          ACESTYPE:=RESTAB[IFR].RPROCWAIT[J].RCURACCTYPE;
          L:=RESTAB[IFR].RPROCWAIT[J].RCURPNAME;
          PROCNAME:=L MOD 1000;
          RESNAME:=RR
        END;
        PACKRW(IFR,J);
        I:=0;
        WHILE RESTAB[IFR].RPROC[I].RCURPNAME<>-1 DO I:=I+1;
        WITH RESTAB[IFR].RPROC[I] DO
          BEGIN
            RCURPNAME:=L;
            RLOCATION:=MSGTEMP.MSGDEST;
            RCURACCTYPE:=MSGTEMP.ACESTYPE;
            RINDEX:=FALSE
          END;
         RESTAB[IFR].RSTATUS:=MSGTEMP.ACESTYPE;
         IF MSGTEMP.MSGDEST=MYSITE THEN UPTABLE ELSE
            OQUEUE.OUTPUTT(MSGTEMP);
         (* IC.MESS9(MSGTEMP.MSGDEST,MSGTEMP.PROCNAME,
                  MSGTEMP.RESNAME); *)
         IF RESTAB[IFR].RSTATUS=SHARED THEN
            BEGIN
              J:=-1;
              FOR I:=0 TO N DO
               IF (RESTAB[IFR].RPROCWAIT[I].RCURPNAME<>-1)
                AND (RESTAB[IFR].RPROCWAIT[I].RCURACCTYPE=SHARED)
                THEN J:=I;
               IF J=-1 THEN SW1:=FALSE
            END ELSE SW1:=FALSE
        END
     END;
END;   (*  RESREL  *)

FUNCTION STILLW:BOOLEAN;
   (* PROCESS STILL WAITING FOR RESOURCE.  *)
   VAR I : INTEGER;
       SW : BOOLEAN;
   BEGIN
     RR:=MSGTEMP.RESNAME;  PP:=MSGTEMP.OBPL.OBPLPROCS[0];
     IFR:=FINDR(RR);  SW:=FALSE;
     FOR I:=0 TO N DO
      IF RESTAB[IFR].RPROCWAIT[I].RCURPNAME=PP THEN
      BEGIN JJ:=I;  SW:=TRUE  END;
     IF SW THEN STILLW:=TRUE
            ELSE STILLW:=FALSE
   END;

PROCEDURE SENDNFREE;
  (* SEND NOTFREE MSG TO REQUESTING PROCESS  *)
 BEGIN
   IF STILLW THEN
   BEGIN
```

```
      WITH MSGTEMP DO
        BEGIN
          MSGTYPE:=NOTFREE;  MSGDEST:=OBPL.PROCNODE;
          MSGORIGIN:=MYSITE
        END;
      IF MSGTEMP.MSGDEST=MYSITE THEN
        PBUF[MSGTEMP.PROCNAME].PRBUFPUT(MSGTEMP) ELSE
        OQUEUE.OUTPUTT(MSGTEMP)
      (* IO.MESS11(MSGTEMP.MSGDEST,MSGTEMP.PROCNAME,MSGTEMP.RESNAME)   *)
    END
  END;

  PROCEDURE SERVENF;
    (* SEND NFREE MSG  *)
   BEGIN
    IF MSGTEMP.MSGORIGIN<>MYSITE THEN
     BEGIN
      MSGTEMP.MSGTYPE:=NFREE;
      OQUEUE.OUTPUTT(MSGTEMP)
     END ELSE SENDNFREE
  END;

  PROCEDURE SENDROLLB;
    (* SENDS ROLL BACK MSG  *)
    VAR SW:BOOLEAN;
        K,I,J,L : INTEGER;
        BEGIN
          IF STILLW THEN
            BEGIN
              WITH MSGTEMP DO
                BEGIN
                  MSGTYPE:=ROLLBACK;
                  MSGDEST:=OBPL.PROCNODE;
                  MSGORIGIN:=MYSITE
                END;
              IF MSGTEMP.MSGDEST=MYSITE THEN
              PBUF[MSGTEMP.PROCNAME].PRBUFPUT(MSGTEMP) ELSE
                  OQUEUE.OUTPUTT(MSGTEMP);
              IO.MESS10(MSGTEMP.MSGDEST,MSGTEMP.PROCNAME,MSGTEMP.RESNAME);
              IO.MESS8(MYSITE,TOTDEAD,TOTREQ);
              (* REMOVE PP FROM WAITING LIST OF RR  *)
              PACKRW(IFR,JJ);
              IF LOCALP(PP) THEN REMOVEPW(PP)
        END;
  END;

  PROCEDURE SERVEDL;
     (* DEADLOCK HAD OCCURED AND ACTION HAS TO BE TAKEN *)
     BEGIN
       TOTDEAD:=TOTDEAD+1;
       IF MSGTEMP.MSGORIGIN<>MYSITE THEN
          BEGIN (* SEND  DLOCK  MSG  *)
```

```
            MSGTEMP.MSGTYPE:=DLOCK;
            OQUEUE.OUTPUTT(MSGTEMP)
          END  ELSE  SENDROLLB
    END;

 PROCEDURE GOLDALG;
  (*  GOLDMAN'S DETECTION ALGORITHM  *)

  (* SITE INITIATING DETECTION STARTS IN STEP 10, WHILE OTHER
     SITES START IN STEP 1  *)
  LABEL 2,3;
  VAR PX,RX,I,J,K,L,P2,STKPTR : INTEGER;
      P1:ARRAY[0..N] OF INTEGER;
      STACK:ARRAY[0..N] OF MESSAGE;
      STACP:ARRAY[0..N] OF INTEGER;
      HALTI,SW : BOOLEAN;

    PROCEDURE STEP10;
     (* PX MUST BE LOCAL TO SITE EXPANDING *)
      BEGIN
       I:=0;
       WHILE MSGTEMP.OBPL.OBPLPROCS[I]<>-1 DO I:=I+1;
       PX:=MSGTEMP.OBPL.OBPLPROCS[I-1];
       I:=FINDP(PX);
       RX:=PROCTAB[I].PNEWREQ.PCURNAME;
       IF NOT LOCALR(RX) THEN
          BEGIN
           (* STEP 11 *)
           MSGTEMP.OBPL.OBPLRNAME:=RX;
           OQUEUE.OUTPUTT(MSGTEMP);
           HALTI:=TRUE
          END ELSE  HALTI:=FALSE
      END; (* STEP 10  *)

  BEGIN
   STKPTR:=0;
   IF ALGENTRY=1 THEN
      BEGIN (* SITE INITIATING STARTS IN STEP 10 *)
        STEP10;
        IF NOT HALTI THEN GOTO 3 ELSE GOTO 2
      END  ELSE
      BEGIN
       (* SITE RECEIVING STARTS AT STEP 1 *)
       (* STEP 1 *)
        RX:=MSGTEMP.OBPL.OBPLRNAME;
        IF LOCALR(RX) THEN
          BEGIN  (* STEP 2  *)
           J:=0;
            WHILE MSGTEMP.OBPL.OBPLPROCS[J] <> -1 DO J:=J+1;
            P2:=MSGTEMP.OBPL.OBPLPROCS[J-1];
             IFR:=FINDR(RX);  SW:=FALSE;
             FOR J:=0 TO N DO
              IF RESTAB[IFR].RPROCWAIT[J].RCURPNAME=P2 THEN SW:=TRUE;
```

```
     IF NOT SW THEN GOTO 2 ELSE GOTO 3
    END ELSE
  BEGIN (* STEP 3 *)
   J:=0;
   WHILE MSGTEMP.OBPL.OBPLPROCS[J]<>-1 DO J:=J+1;
   P2:=MSGTEMP.OBPL.OBPLPROCS[J-1];
   IFP:=FINDP(P2);
   SW:=FALSE;
   FOR J:=0 TO M DO
    IF PROCTAB[IFP].PRESRCE[J].PCURNAME=RX THEN SW:=TRUE;
   IF NOT SW THEN GOTO 2 ELSE
      (* STEP 9 *)
     IF PROCTAB[IFP].PSTATE=RUNNING THEN
            BEGIN
             SERVENF; GOTO 2 END ELSE
        BEGIN
         STEP10;
          IF NOT HALTT THEN GOTO 3 ELSE GOTO 2
        END
   END; (* STEP 8 *)
 END; (* STEP 1 *)
3: (* FIND PROCESSES CONTROLLING RX *)
   K:=-1; IFR:=FINDR(RX);
   FOR I:=0 TO N DO
    IF RESTAB[IFR].RPROC[I].RCURPNAME<>-1 THEN
     BEGIN
      K:=K+1;
      P1[K]:=RESTAB[IFR].RPROC[I].RCURPNAME
     END;
    WHILE K>=0 DO
      BEGIN
       STACK[STKPTR]:=MSGTEMP;
       STACP[STKPTR]:=P1[K];
       STKPTR:=STKPTR+1;
       K:=K-1
      END;
 2:
      WHILE STKPTR>0 DO
       BEGIN
        STKPTR:=STKPTR-1; MSGTEMP:=STACK[STKPTR];
        PX:=STACP[STKPTR];
        (* CHECK IF PX IS IN OBPL *)
        DEADLOCK:=FALSE;
        FOR L:=0 TO N DO
         IF PX=MSGTEMP.OBPL.OBPLPROCS[L] THEN DEADLOCK:=TRUE;
        IF DEADLOCK THEN
         SERVEDL (* SEND DLOCK MSG IF SITE WAS NOT THE
                  INITIATOR OF DETECTION ELSE SENDROLLB *)
                ELSE
           (* STEP 4 *)
          IF NOT LOCALP(PX) THEN
           BEGIN (* STEP 7 *)
```

```
                    J:=0;
                    WHILE MSGTEMP.OBPL.OBPLPROCS[J]<>-1 DO  J:=J+1;
                    MSGTEMP.OBPL.OBPLPROCS[J]:=PX;
                    MSGTEMP.OBPL.OBPLRNAME:=RX;
                    OQUEUE.OUTPUTT(MSGTEMP)
                   END ELSE
                   BEGIN
                     (*  STEP 5  *)
                    IFP:=FINDP(PX)
                    ; IF PROCTAB[IFP].PSTATE=BLOCKED THEN
                    BEGIN  (* STEP 6 *)
                      J:=0;
                      WHILE MSGTEMP.OBPL.OBPLPROCS[J]<>-1 DO J:=J+1;
                      MSGTEMP.OBPL.OBPLPROCS[J]:=PX;
                      STEP10;
                      IF NOT HALTT THEN GOTO 3
                    END ELSE SERVENF
                   END
              END;  (*  WHILE  *)
 END;  (*  GOLDALG  *)

 PROCEDURE OBPLINIT;
  (* CREATES OBPL AND SENDS TO SITE OWNING PP TO START EXPANDING *)
  VAR I : INTEGER;
  BEGIN
    NINITO:=NINITO+1;
    WITH MSGTEMP DO
     BEGIN
       OBPL.PROCNODE:=MSGORIGIN;  MSGORIGIN:=MYSITE;
       MSGTYPE:=INITDEAD;  MSGDEST:=OBPL.PROCNODE;
       OBPL.OBPLRNAME:=RR; OBPL.OBPLOWNER:=MYSITE;
     END;
    FOR I:=0 TO N DO
      MSGTEMP.OBPL.OBPLPROCS[I]:=-1;
    MSGTEMP.OBPL.OBPLPROCS[0]:=PP;
    IF MSGTEMP.MSGDEST=MYSITE THEN
      BEGIN
        MSGTEMP.MSGTYPE:=DETEK; ALGENTRY:=1; GOLDALG
      END ELSE
      OQUEUE.OUTPUTT(MSGTEMP)
  END;

 PROCEDURE RESREQ;
   (* PROCESS PP REQUESTS FOR RESOURCE RR LOCATED AT THIS SITE *)
   VAR I,J : INTEGER;
      RFREE:BOOLEAN;
   BEGIN
    TOTREQ:=TOTREQ+1;   IFR:=FINDR(RR);
    (*   CHECK IF RESOURCE IS FREE   *)
    RFREE:=RESFREE(RR);
    IF RFREE THEN
         (* RESOURCE IS FREE,ALLOCATE IT  ^ SEND RESPONSE *)
```

```
SENDRESPONSE  ELSE
BEGIN
 (* UPDATE PROCESSES WAITING TO ACCESS RR *)
 J:=0;
 WHILE RESTAB[IFR].RPROCWAIT[J].RCURPNAME<>-1 DO J:=J+1;
 WITH RESTAB[IFR].RPROCWAIT[J] DO
    BEGIN
      RCURPNAME:=PP;  RCURACCTYPE:=REQACCESS;
      RLOCATION:=MSGTEMP.MSGORIGIN;
      RINDEX:=FALSE
    END;
 IF MSGTEMP.MSGORIGIN=MYSITE THEN
    BEGIN
      IFP:=FINDP(PP);
      PROCTAB[IFP].PSTATE:=BLOCKED;
      WITH PROCTAB[IFP].PNEWREQ DO
        BEGIN
          PCURNAME:=RR;  PCURACCTYPE:=REQACCESS;
          PROWNER:=MYSITE
        END;
    END;
 DBPLINIT  (* CREAT DBPL *)
END;
END;  (*  RESREQ  *)

(****************************************************************)


(**********************************************)
(*                                            *)
(*       RANDOM NUMBER GENERATORS             *)
(*                                            *)
(**********************************************)

FUNCTION RAND(VAR SEED:REAL; MODP:INTEGER):INTEGER;
  CONST
    P=2147483647;
    A=16807;
  VAR ISEED:INTEGER;
  BEGIN
    ISEED:=TRUNC(SEED);
    SEED:=(A*ISEED) MOD P;
    ISEED:=TRUNC(SEED) MOD MODP;
    RAND:=ISEED
  END;  (*  RAND  *)

FUNCTION RANDOM(VAR S:REAL):REAL;
  VAR ISEED:INTEGER;
  BEGIN
    ISEED:=TRUNC(S);
    ISEED:=(ISEED*899) MOD 32767;
```

```
        S:=ISEED;
        RANDOM:=S/32767.0
      END;   (*   RANDOM   *)
(**************************************************************)
(*                                                           *)
(*   E N D   OF   R O U T I N E S                            *)
(*                                                           *)
(**************************************************************)

   PROCESS WRITER(OUTLINE:LINE;SITE,MAXP:INTEGER);
     (*  WRITE MSG TO OUTPUT LINE  *)
    VAR M:MESSAGE;
        WRITING:BOOLEAN;
        TOTL,TOTMSGSENT,TOTMAXP :INTEGER;
    BEGIN
     WRITING:=TRUE;
     TOTMSGSENT:=0;   TOTMAXP:=MAXP;
     TOTL:=0;
     WHILE WRITING DO
      BEGIN
        OQUEUE.OUTGET(M);
        OUTLINE.TOLINE(M);
        TOTMSGSENT:=TOTMSGSENT+1;
        IF M.MSGTYPE=ATERMINATE THEN TOTL:=TOTL+1;
        IF TOTL=TOTMAXP THEN WRITING:=FALSE
      END;
     IO.MESS1(SITE,TOTMSGSENT);
    END;  (*  WRITER  *)

    PROCESS READER(INLINE:LINE;MAXP:INTEGER);
     (*  MONITORS THE LINE FOR ANY INCOMING MESSAGES  *)
     VAR MESG:MESSAGE;
        TOTNFREE,R,P,I,RTOTL,TOTMSGRECVD : INTEGER;
        TOTDETEK,TOTRESP,TOTROLLB,TOTCOMPL,TOTAREQ,TOTINIT: INTEGER;
        TOTDLOCK,TOTNF : INTEGER;
        SW,READING : BOOLEAN;
     BEGIN
       READING:=TRUE;   RTOTL:=0;   TOTMSGRECVD:=0;
       TOTNFREE:=0; TOTDETEK:=0; TOTRESP:=0; TOTROLLB:=0; TOTCOMPL:=0;
       TOTAREQ:=0; TOTINIT:=0; TOTDLOCK:=0; TOTNF:=0;
       WHILE READING DO
         BEGIN
          INLINE.FRLINE(MESG);
          TOTMSGRECVD:=TOTMSGRECVD+1;
          CASE MESG.MSGTYPE OF
            AREQUEST:
              BEGIN
               TOTAREQ:=TOTAREQ+1;
               SW:=FALSE;
               FOR I:=0 TO MMAX DO
```

```
          IF MESG.RESNAME=RESTAB[I].RNAME THEN SW:=TRUE;
         IF SW THEN MQUEUE.QUEPUT(MESG) ELSE
                    OQUEUE.OUTPUTT(MESG)
        END;
      ARESPONSE,INITDEAD,ROLLBACK,DLOCK,NFREE,COMPLETION:
      BEGIN
           CASE MESG.MSGTYPE OF
             ARESPONSE: TOTRESP:=TOTRESP+1;
             INITDEAD : TOTINIT:=TOTINIT+1;
             ROLLBACK : TOTROLLB:=TOTROLLB+1;
             DLOCK    : TOTDLOCK:=TOTDLOCK+1;
             NFREE    : TOTNF:=TOTNF+1;
             COMPLETION: TOTCOMPL:=TOTCOMPL+1
           END;
           IF MESG.MSGDEST=MYSITE THEN MQUEUE.QUEPUT(MESG)
                                   ELSE OQUEUE.OUTPUTT(MESG)
      END;
      DETEK:
        BEGIN
          TOTDETEK:=TOTDETEK+1;
          R:=MESG.OBPL.OBPLRNAME; I:=0;
          WHILE MESG.OBPL.OBPLPROCS[I]<>-1 DO I:=I+1;
          P:=MESG.OBPL.OBPLPROCS[I-1];
          IF LOCALP(P) OR LOCALR(R) THEN MQUEUE.QUEPUT(MESG)
                                     ELSE OQUEUE.OUTPUTT(MESG)
        END;
       NOTFREE:
          BEGIN
           IF MESG.MSGDEST=MYSITE THEN
             PBUF[MESG.PROCNAME].PRBUFPUT(MESG) ELSE
             OQUEUE.OUTPUTT(MESG);
          TOTNFREE:=TOTNFREE+1
          END;
       ATERMINATE:
          BEGIN
            RTOTL:=RTOTL+1;
            IF MESG.MSGORIGIN<>MYSITE THEN MQUEUE.QUEPUT(MESG)
            ; IF RTOTL=MAXP THEN READING:=FALSE
          END;
      END; (* CASE *)
      IF MESG.MSGTYPE=ROLLBACK THEN IO.MESS15(MYSITE,TOTNFREE,
       TOTDETEK,TOTRESP,TOTROLLB,TOTCOMPL,TOTAREQ,TOTINIT,
       TOTDLOCK,TOTNF)
    END; (* WHILE READING *)
    IO.MESS2(MYSITE,TOTMSGRECVD);
    IO.MESS15(MYSITE,TOTNFREE,TOTDETEK,TOTRESP,TOTROLLB,
       TOTCOMPL,TOTAREQ,TOTINIT,TOTDLOCK,TOTNF);
  END; (* READER *)

  PROCESS KERNEL(SITE:SITES;MAXR,MAXP : INTEGER);
    (* KERNEL HANDLES THE RESOURCE ALLOCATION  AT EACK SITE;
       IT RUNS THE DETECTION ALGORITHM  *)
    VAR KTOTL,I,TOTMAXP,GSIZE,TOTLOC:INTEGER;
```

```
KERNELLING,SW:BOOLEAN;
(*  MAXR IS MAXIMUM RESOURCE AT THIS SITE  *)
BEGIN
 TOTMAXP:=MAXP;  TOTLOC:=0;   KERNELLING:=TRUE;   KTOTL:=0;
 WHILE KERNELLING DO
   BEGIN
    MQUEUE.QUEGIT(MSGTEMP,QSIZE);
    CASE MSGTEMP.MSGTYPE OF
      ATERMINATE:
        BEGIN
          KTOTL:=KTOTL+1; OQUEUE.OUTPUTT(MSGTEMP)
        END;
      LOCAL 1:
        BEGIN
          MSGTEMP.QUESIZE:=QSIZE; MSGTEMP.MSGTYPE:=AREQUEST;
          PP:=MYSITE*1000+MSGTEMP.PROCNAME;
          IF NOT LOCALP(PP) THEN INSERTP(PP,IFP) ELSE
                                 IFP:=FINDP(PP);
          SW:=FALSE;
          FOR I:=0 TO MMAX DO
           IF MSGTEMP.RESNAME=RESTAB[I].RNAME THEN SW:=TRUE;
           (* BLOCK PROCESS  *)
           PROCTAB[IFP].PSTATE:=BLOCKED;
           PROCTAB[IFP].PNEWREQ.PCURNAME:=MSGTEMP.RESNAME;
           IF NOT SW THEN OQUEUE.OUTPUTT(MSGTEMP)  ELSE
             BEGIN
               RR:=MSGTEMP.RESNAME;
               REQACCESS:=MSGTEMP.ACESTYPE;
               TOTLOC:=TOTLOC+1;
               RESREQ
             END
         END;
      AREQUEST:
        BEGIN
          WITH MSGTEMP DO
            BEGIN
             QUESIZE:=QUESIZE+QSIZE;
             PP:=MSGORIGIN*1000+PROCNAME;
             RR:=RESNAME;
             REQACCESS:=ACESTYPE
            END;
          RESREQ
        END;
      DLOCK: SENDROLLB;
      NFREE: SENDNFREE;
      ARESPONSE: UPTABLE;
      DETEK: BEGIN
               ALGENTRY:=2;
               GOLDALG
             END;
      INITDEAD:
         BEGIN
           MSGTEMP.MSGTYPE:=DETEK; ,
```

```
                MSGTEMP.MSGDEST:=MSGTEMP.MSGORIGIN;
                ALGENTRY:=1;  GOLDALG
              END;
          ROLLBACK:
           BEGIN
            PP:=MYSITE*1000+MSGTEMP.PROCNAME;
            IFP:=FINDP(PP);
            PROCTAB[IFP].PNEWREQ.PCURNAME:=-1;
            PBUF[MSGTEMP.PROCNAME].PRBUFPUT(MSGTEMP)
           END;
          COMPLETION:
           BEGIN
            PP:=MSGTEMP.MSGORIGIN*1000+MSGTEMP.PROCNAME;
            RR:=MSGTEMP.RESNAME;
            IF LOCALP(PP) THEN
              BEGIN
                IFP:=FINDP(PP);  I:=0;
                WHILE PROCTAB[IFP].PRESRCE[I].PCURNAME<>RR DO
                    I:=I+1;
                WITH PROCTAB[IFP].PRESRCE[I] DO
                   BEGIN
                     PCURNAME:=-1;  PROWNER:=-1;
                     PCURACCTYPE:=FREE
                   END;
                END;
             IF MSGTEMP.MSGDEST=MYSITE THEN RESREL ELSE
                OQUEUE.OUTPUTT(MSGTEMP)
           END;
       END;  (*  CASE  *)
      IF KTOTL=TOTMAXP THEN KERNELLING:=FALSE;
     END;  (*  WHILE  *)
    IO.MESS8(MYSITE,TOTDEAD,TOTREQ);
    IO.MESS14(MYSITE,TOTLOC,NINITD)
END;  (*  KERNEL  *)

PROCESS PPROCSS(SITE,TOTMAXR,PROCNO:INTEGER;LAMDA,MUU:REAL;
                MAXREQ,WACCES,THRUPUT : INTEGER);
   (* SIMULATE A LOCAL PROCESS ACTIVITIES  *)
  LABEL 1,2;
  TYPE
    LRES=RECORD
      LRNAME:INTEGER;
      TACCESS:STATUS;
      LOCATION:INTEGER
    END;
  VAR
    RESRCES:ARRAY[1..5] OF LRES;
    CLOCK,TRELEASE,TREQUEST,LAMDABAR,MUUBAR,SEEDR,SEED:REAL;
    TEMP,T2,TBEFORE,TOTSECS,XXX:REAL;
    NUMRES,RR,MP,I,J,TOTSENT,TOTDELAY,RELPTR,REQPTR :INTEGER;
    LPROCID,OUTREQ,THRUBEFORE,THRUAFTER : INTEGER;
    TESTCASE,TS,ID,MPPP,TSS,IDD,MPP : INTEGER;
    MAINSW,SW,SW1,GREATR,PROCESING,AGAN :BOOLEAN;
```

```
MYMSG:MESSAGE;
ACCTYPE:STATUS;

PROCEDURE GENREQ;
 LABEL 3;
    BEGIN (* GENERATE NEW RESOURCE  *)
     SW:=FALSE;
     WHILE NOT SW DO
       BEGIN
         RR:=RAND(SEED,TOTMAXR)+1;
         IF (REQPTR=0) OR (OUTREQ=0) THEN
          SW:=TRUE ELSE
          BEGIN
           SW1:=FALSE;
           J:=(RELPTR MOD 5)+1;
           FOR I:=1 TO OUTREQ DO
            BEGIN
              IF RESRCES[J].LRNAME=RR THEN SW1:=TRUE;
              J:=(J MOD 5)+1
            END;
           IF NOT SW1 THEN SW:=TRUE
          END
         END;
       (* TYPE OF ACCESS  *)
       IF NACCES=1 THEN ACCTYPE:=EXCLUSIVE ELSE
          BEGIN
           TEMP:=RANDOM(SEEDR);
           IF TEMP>=0.5 THEN ACCTYPE:=EXCLUSIVE ELSE
                              ACCTYPE:=SHARED
          END;
       REQPTR:=(REQPTR MOD 5)+1;  OUTREQ:=OUTREQ+1;
       RESRCES[REQPTR].LRNAME:=RR;
       RESRCES[REQPTR].TACCESS:=ACCTYPE;
     (* SEND REQUEST  *)
    WITH MYMSG DO
     BEGIN
      MSGORIGIN:=SITE; PROCNAME:=LPROCID;
      QUESIZE:=0;
      MSGTYPE:=LOCALL; RESNAME:=RR;
      ACESTYPE:=ACCTYPE
     END;
    IO.MESS3(SITE,LPROCID,RR,ACCTYPE);
    MQUEUE.QUEPUT(MYMSG);
    J:=TIME;  TOTSENT:=TOTSENT+1;
    (* TBEFORE:=SIN(XXX);  *)
    PBUF[LPROCID].PRBUFGET(MYMSG);
    (* PROCESS BLOCKED WAITING FOR RESPONSE  *)
    (* TEMP:=SIN(XXX)-TBEFORE;  *)
    (* TS:=TRUNC(TEMP);  *)
    TD:=TIME-J;
      MPPP:=MYMSG.QUESIZE;
    TSS:=TS;
    TDD:=TD;        MPP:=MPPP;
```

```
      3:
      IF MYMSG.MSGTYPE=NOTFREE THEN
         BEGIN
           (* TSS:=TS;  *)
           TDO:=TD;      MPP:=MPPP;
           PBUF[LPROCID].PRBUFGET(MYMSG);
           (* TEMP:=SIN(XXX)-TBEFORE;  *)
            TD:=TIME-J;
            (* TS:=TRUNC(TEMP);   *)
            MPPP:=MYMSG.QUESIZE;
            GOTO 3
         END;
      IF MYMSG.MSGTYPE=ROLLBACK THEN
        BEGIN
         IO.MESS13(SITE,LPROCID,TD,MPPP);
         (* IO.MESS4(SITE,LPROCID,MYMSG.RESNAME);  *)
         RESRCES[REQPTR].LOCATION:=MYMSG.MSGORIGIN;
         REQPTR:=REQPTR-1;
         IF (REQPTR=0) OR (REQPTR=-1) THEN REQPTR:=5;
         OUTREQ:=OUTREQ-1;
         MP:=MYMSG.MSGORIGIN; AGAN:=TRUE
        END  ELSE
     BEGIN
          IO.MESS13(SITE,LPROCID,TDO,MPP);
          (* O.MESS5(SITE,LPROCID,MYMSG.RESNAME);  *)
          RESRCES[REQPTR].LOCATION:=MYMSG.MSGORIGIN;
       END;
    END;  (* GENREQ  *)
  PROCEDURE ASSREL;
    BEGIN
     RELPTR:=(RELPTR MOD 5)+1;  OUTREQ:=OUTREQ-1;
      WITH MYMSG DO
        BEGIN
         PROCNAME:=LPROCID;  MSGTYPE:=COMPLETION;
         MSGORIGIN:=SITE;
         MSGDEST:=RESRCES[RELPTR].LOCATION;
         RESNAME:=RESRCES[RELPTR].LRNAME; ACESTYPE:=FREE
         END
  END;  (* RELPTR  *)

  BEGIN
    LPROCID:=PROCNO;
   TOTSENT:=0; TOTDELAY:=0; XXX:=5.0;   PROCESING:=TRUE;
   TOTSECS:=0.0; CLOCK:=0.0; TRELEASE:=0.0; TREQUEST:=0.0;
   SEEDR:=31415.0/SITE;  SEED:=SITE;  THRUBEFORE:=TIME;
   (*  RELPTR POINTS TO THE LAST RESOURCE RELEASED
       REQPTR POINTS TO THE LAST RESOURCE REQUESTED FOR  *)
 1: RELPTR:=0;  REQPTR:=0;    GREATR:=FALSE;
   OUTREQ:=0;  AGAN:=FALSE;
    MAINSW:=FALSE;
   WHILE PROCESING DO
      BEGIN
```

```
MP:=-1;
GENREQ;
IF MP<>-1 THEN GOTO 2;
(* GENERATE TIME OF NEXT RELEASE  *)
MUUBAR:=(-1.0/MUU)*LN(RANDOM(SEEDR));
TRELEASE:=CLOCK+MUUBAR;
(* GENERATE TIME OF NEXT REQUEST *)
LAMDABAR:=(-1.0/LAMDA)*LN(RANDOM(SEEDR));
TREQUEST:=CLOCK+LAMDABAR;
MAINSW:=TRUE;
WHILE MAINSW DO
 BEGIN
  IF TRELEASE>TREQUEST THEN TESTCASE:=1;
  IF TRELEASE=TREQUEST THEN TESTCASE:=2;
  IF TRELEASE<TREQUEST THEN TESTCASE:=3;
  CASE TESTCASE OF
    1:  (* TRELEASE>TREQUEST  *)
      BEGIN
         TEMP:=LAMDABAR*100.0;
         I:=TRUNC(TEMP);  T2:=I+0.49;
         IF TEMP>T2 THEN I:=I+1;
         DELAY(I);  CLOCK:=TREQUEST;
         IF  (TOTSENT>=MAXREQ) THEN
           BEGIN
            AGAN:=FALSE; MP:=-1; PROCESING:=FALSE; GOTO 2
           END;
         MUUBAR:=TRELEASE-TREQUEST;
           (* GENERATE REQUEST  *)
         IF OUTREQ>=TOTMAXR THEN
          BEGIN
           (* REQUEST BUT RES HELD EQUALS MAX RES *)
            TRELEASE:=TREQUEST; ASSREL;
            MQUEUE.QUEPUT(MYMSG);
            (* IO.MESS6(SITE,LPROCID,RESRCES[RELPTR].LRNAME);*)
            MAINSW:=FALSE
         END ELSE
         BEGIN
          MP:=-1;  GENREQ;
          IF MP<>-1 THEN GOTO 2;
          (* GENERATE TIME OF NEXT REQUEST  *)
          LAMDABAR:=(-1.0/LAMDA)*LN(RANDOM(SEEDR));
          TREQUEST:=CLOCK+LAMDABAR;  MAINSW:=TRUE
      END
   END;  (* TESTCASE=1  *)
   2: (* TRELEASE=TREQUEST  *)
     BEGIN
       CLOCK:=TRELEASE;
       TEMP:=LAMDABAR*100.0;
       I:=TRUNC(TEMP); T2:=I+0.49;
       IF TEMP>T2 THEN I:=I+1; DELAY(I);
       (* RELEASE RESOURCE IF ANY  *)
       IF OUTREQ>0 THEN
         BEGIN
```

```
                ASSREL;
                 MQUEUE.QUEPUT(MYMSG)
           (* IO.MESS6(SITE,LPROCID,RESRCES[RELPTR].LRNAME) *)
          END;
          MAINSW:=FALSE;
          IF  (TOTSENT>=MAXREQ) THEN
            BEGIN
             AGAN:=FALSE; MP:=-1; PROCESING:=FALSE; GOTO 2
            END;
        END;
     3:  (* TRELEASE<TREQUEST  *)
        BEGIN
         TEMP:=MUUBAR*100.0;
         I:=TRUNC(TEMP); T2:=I+0.49;
         IF TEMP>T2 THEN I:=I+1;   DELAY(I);
         IF OUTREQ<=0 THEN
          BEGIN (* NO RES TO RELEASE  *)
           CLOCK:=TREQUEST;
           TEMP:=(TREQUEST-TRELEASE)*100.0;
           I:=TRUNC(TEMP);  T2:=I+0.49;
           IF TEMP>T2 THEN I:=I+1;   DELAY(I);   MAINSW:=FALSE
           END  ELSE
           BEGIN  (* RELEASE  RESOURCE  *)
            CLOCK:=TRELEASE;  ASSREL;
            MQUEUE.QUEPUT(MYMSG);
            (* IO.MESS6(SITE,LPROCID,RESRCES[RELPTR].LRNAME); *)
            LAMDABAR:=TREQUEST-TRELEASE;
            (* GENERATE TIMOE NEXT RELEASE *)
             MUUBAR:=(-1.0/MUU)*LN(RANDOM(SEEDR));
             TRELEASE:=CLOCK+MUUBAR;
             IF  (TOTSENT>=MAXREQ) THEN
               BEGIN
                 AGAN:=FALSE;  MP:=-1;  PROCESING:=FALSE;
                 GOTO 2
            END
      END
     END;  (* TRELEASE < TREQUEST  *)
   END;  (*  CASE  *)
   END; (*  MAINSW  *)
  END;  (* PROCESING  *)
     2:  IF OUTREQ>0 THEN
          WHILE OUTREQ>0 DO
           BEGIN
            ASSREL;
            MQUEUE.QUEPUT(MYMSG)
            (* IO.MESS6(SITE,LPROCID,MYMSG.RESNAME) *)
          END;
       IF AGAN THEN
           BEGIN
            I:=1000;
            J:=RAND(SEED,I)+100;
            DELAY(J);
            IF THRUPUT = 1 THEN TOTSENT:=0;
```

```
            GOTO 1
           END;
         IF THRUPUT=1 THEN THRUAFTER:=TIME-THRUBEFORE
                          ELSE THRUAFTER:=99900;
         MYMSG.MSGTYPE:=ATERMINATE;
         MYMSG.MSGDEST:=SITE;
         MQUEUE.QUEPUT(MYMSG);
       IC.MESS7(SITE,LPROCID,TOTSENT,THRUAFTER);
     END;   (*  PROCESS PPROCSS  *)

     ENTRY PROCEDURE STARTMACH(SITE:SITES;INLINE,OUTLINE:LINE;MAXR,SRES,
       STARTR,MAXP,PROCS: INTEGER;LAMDA,MUU:REAL;MAXREQ,WACC,THRUP : INTEGER);
       VAR I,J : INTEGER;
       BEGIN
        TOTREQ:=0; TOTDEAD:=0; NINITO:=0; MYSITE:=SITE;
        INITIALIZE;
       (* INITIALISE RESOURCE TABLE *)
        J:=STARTR;
        FOR I:=0 TO SRES-1 DO
         BEGIN
           WITH RESTAB[I] DO
            BEGIN
             RNAME:=J; RSTATUS:=FREE
            END;
           J:=J+1
         END;
          (* START PROCESSES AT THIS SITE  *)
         FOR I:=1 TO PROCS DO
            PPROCSS(SITE,MAXR,I,LAMDA,MUU,MAXREQ,WACC,THRUP);
         KERNEL(SITE,SRES,MAXP);
         READER(INLINE,MAXP);
         WRITER(OUTLINE,SITE,MAXP);
      END;  (* STARTMACH  *)

   END;    (*  ******   MACHINE   ******  *)

 (**************************************************************)
 (*                                                            *)
 (*   S Y S T E M   A C T I V A T I O N                        *)
 (*                                                            *)
 (**************************************************************)

 VAR
   NET:ARRAY[SITES] OF MACHINE;
   LINK:ARRAY[NLINES] OF LINE;
   MAXREQ,MAXP,MAXR,WACC,THRUP : INTEGER;
     (* WACC=0 IF SHARED ^ EXCL AND 1 IF EXCL ONLY  *)
     (* THRUP=0 PERFORMANCE MEASURE, =1 THRUPUT MEASURE  *)
    LAMDA,MUU:REAL;
    RPERSITE:ARRAY[SITES] OF INTEGER;
    PPERSITE:ARRAY[SITES] OF INTEGER;
     I,K,L,J,Y : INTEGER;
    BEGIN
```

```
READ(MAXP,MAXR,LAMDA,MUU,MAXREQ,WACC,THRUP);
(* DISTRIBUTE RESOURCES AMONG SITES  *)
K:=0;  L:=MAXR DIV NSITES; J:=0; Y:=MAXP DIV NSITES;
FOR I:=1 TO NSITES DO
 BEGIN
   RPERSITE[I]:=L; K:=K+L;
   PPERSITE[I]:=Y; J:=J+Y
 END;
I:=0;
WHILE K<MAXR DO
  BEGIN
    I:=I+1;
    RPERSITE[I]:=RPERSITE[I]+1;   K:=K+1
  END;
  I:=0;
  WHILE J<MAXP DO
    BEGIN
      I:=I+1;
      PPERSITE[I]:=PPERSITE[I]+1;
      J:=J+1
    END;
 WRITELN(' GOLDMAN ','ALGORITHM');
 WRITELN(' N= ',MAXP,' M = ',MAXR);
 WRITELN(' MUU = ',MUU,' LAMDA = ',LAMDA);
 WRITELN(' MAXIMUM ','REQUEST =',MAXREQ);
 NET[1].STARTMACH(1,LINK[3],LINK[1],MAXR,RPERSITE[1],1,MAXP,
   PPERSITE[1],LAMDA,MUU,MAXREQ,WACC,THRUP);
 I:=RPERSITE[1]+1;
 NET[2].STARTMACH(2,LINK[1],LINK[2],MAXR,RPERSITE[2],I,MAXP,
   PPERSITE[2],LAMDA,MUU,MAXREQ,WACC,THRUP);
 I:=I+RPERSITE[2];
 NET[3].STARTMACH(3,LINK[2],LINK[3],MAXR,RPERSITE[3],I,MAXP,
   PPERSITE[3],LAMDA,MUU,MAXREQ,WACC,THRUP);
END.
```

APPENDIX D

Program Listing for Centralized Implementation of the Horizontal and Vertical Algorithm on a 4-Site Network, Where the Fourth Site is the Controller Site

```
PROGRAM CENDEAD(INPUT,OUTPUT);
    (* ***************************************************** *)
    (*  SIMULATION PROGRAM FOR HORIZONTAL ^ VERTICAL DEADLOCK *)
    (*  DETECTION ALGORITHM -- CENTRALIZED CONTROL            *)
    (*   SIMULATION LANGUAGE : PATH PASCAL                    *)
    (*   A FOUR-SITE NETWORK: SITE 4 IS THE RESOURCE MANAGER  *)
    (*   ALL RESOURCES ARE CONTROLLED AND ALLOCATED BY THE    *)
    (*   RESOURCE MANAGER. PROCESSES RUN ON SITES 1 TO 3      *)
    (*                                                        *)
    (*                                                        *)
    (* ***************************************************** *)

    CONST
      NSITES=4;       (*  4  SITES  *)
      BMAX=10;      (*  BUFFER SIZE ON PROCESS MACHINES    *)
      RSITES=3;       (*  3 SITES RUNNING THE PROCESSES     *)
      NMAX=10;  (*  MAXIMUM NUMBER OF PROCESSES RUNNING ON ALL SITES *)
      MMAX=10;  (*  MAXIMUM NUMBER OF RESOURCES         *)
      QMAX=20;      (*  QUEUE SIZE ON THE CONTROLLER MACHINE   *)
    TYPE
      MESSTYPE=(AREQUEST,RESPONSE,COMPLETION,ROLLBACK,
                                  ATERMINATE,NOTFREE);
      SITE=1..NSITES;
      RSITE=1..RSITES;
      STATUS=(FREE,EXCLUSIVE,SHARED);
      MESSAGE=RECORD
        MSGTYPE:MESSTYPE;
        MSGID:RSITE;
        RESID:INTEGER;
        PROCID:INTEGER;
        QUESIZE : INTEGER;
        ACCESSID:STATUS
      END;
       (*    CONTROLLER DATA TYPES         *)
     PRTBLE=RECORD
        RNK : INTEGER;
        TACCES:STATUS
     END;
      NPROC=-1..NMAX;
      MRES=-1..MMAX;
      MAT=ARRAY[MRES,NPROC] OF PRTBLE;
      STATE=(BLOCKED,RUNNING);
      RESHELD=RECORD
           RNAM:INTEGER;
           RACC:STATUS
         END;
      PROCS=RECORD
          PNAME:INTEGER;
          PSITE:INTEGER;
          PSTATE:STATE;
          RHELD:ARRAY[MRES] OF RESHELD
       END;
      RESRC=RECORD
```

```
    RNAME:INTEGER;
    PSTATUS:STATUS
   END;

 PROCIO=OBJECT
   PATH 1:( 1:(MESS1),1:(MESS2), 1:(MESS3),1:(MESS4),
          1:(MESS5),1:(MESS6),1:(MESS7),1:(MESS8),1:(MESS9),
     1:(MESS10),1:(MESS11),1:(MESS12),1:(MESS13),1:(MESS14),
     1:(MESS15) ) END;

   ENTRY PROCEDURE MESS1(I,J:INTEGER);
     VAR K:INTEGER;
     BEGIN
       K:=(J*100+I)*100;
       WRITELN(K)
     END;                                   (*  MESS1   *)

   ENTRY PROCEDURE MESS2(I,J:INTEGER);
     VAR K:INTEGER;
     BEGIN
       K:=(J*100+I)*100+1;
       WRITELN(K)
     END;                                     (*  MESS2   *)

   ENTRY PROCEDURE MESS3(I,J,K:INTEGER;L:STATUS);
     VAR T:INTEGER;
     BEGIN
       T:=I*100000+J*10000+K*100;
       IF L=EXCLUSIVE THEN T:=T+2 ELSE T:=T+3;
       WRITELN(T)
     END;                                      (*  MESS3  *)

   ENTRY PROCEDURE MESS4(I,J,K:INTEGER);
     VAR T:INTEGER;
     BEGIN
      T:=I*100000+J*10000+K*100+4;
      WRITELN(T)
     END;                                       (*  MESS4   *)

   ENTRY PROCEDURE MESS5(I,J,K:INTEGER);
     VAR T:INTEGER;
     BEGIN
       T:=I*100000+J*10000+K*100+5;
       WRITELN(T)
     END;                                       (*  MESS5   *)

   ENTRY PROCEDURE MESS6(I,J,K:INTEGER);
     VAR T:INTEGER;
     BEGIN
       T:=I*100000+J*10000+K*100+10;
       WRITELN(T)
     END;                                          (*  MESS6   *)
```

```
ENTRY PROCEDURE MESS7(I,J,K,L :INTEGER);
  VAR T,T1,T2,T3 :INTEGER;
  BEGIN
    T:=I*1000+J*100+11;
    T1:=K*100000+I*1000+J*100+12;
    T3:=L DIV 100;
    T2:=T3*10000+I*100+J*10+8;
    WRITELN(T,T1,T2)
  END;                                          (*  MESS7  *)

ENTRY PROCEDURE MESS8(I,J,K:INTEGER);
  VAR T,T1 : INTEGER;
  BEGIN
    T:=J*10000+I*100+13;
    T1:=K*10000+I*100+14;
    WRITELN(T,T1)
  END;                                          (*  MESS8  *)

ENTRY PROCEDURE MESS9(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+15;
    WRITELN(T)
  END;                                          (*  MESS9  *)

ENTRY PROCEDURE MESS10(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+20;
    WRITELN(T)
  END;                                       (*  MESS10  *)

ENTRY PROCEDURE MESS11(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+21;
    WRITELN(T)
  END;                                            (*  MESS11  *)

ENTRY PROCEDURE MESS12(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+22;
    WRITELN(T)
  END;                                          (* MESS12  *)
  ENTRY PROCEDURE MESS13(I,J,OU,QS :INTEGER);
      VAR T,T1,T3:INTEGER;
      BEGIN
        OU:=OU DIV 100;
        T:=I*100+J*10;
        T1:=OU*10000+T+7;
        T3:=QS*10000+T+9;
```

```
            WRITELN(T1,T3)
         END;                                          (* MESS13  *)

      ENTRY PROCEDURE MESS14(I,J,K :INTEGER);
           VAR T,T1 :INTEGER;
           BEGIN
             T:=J*10000+I*100+23;
             T1:=K*10000+I*100+26;
             WRITELN(T,T1)
           END;                                          (*  MESS14  *)

      ENTRY PROCEDURE MESS15(I,NF,DE,RES,ROL,COM,ARE,K1,K2,K3:INTEGER);
           VAR T1,T2,T3,T4,T5,T6,T7,T8,T9,T1000,T100 : INTEGER;
           BEGIN
            T1000:=10000;
            T100:=I*100;
            T1:=NF*T1000+T100+30;
            IF DE<>99 THEN BEGIN
                                T2:=DE*T1000+T100+31;
                                WRITE(T2)
                              END;
            T3:=RES*T1000+T100+32;
            T4:=ROL*T1000+T100+33;
            T5:=COM*T1000+T100+34;
            T6:=ARE*T1000+T100+35;
            IF K1<>99 THEN BEGIN
                               T7:=K1*T1000+T100+36;
                               T8:=K2*T1000+T100+24;
                               T9:=K3*T1000+T100+25;
                               WRITE(T7,T8,T9)
                             END;
            WRITELN(T1,T3,T4,T5,T6)
      END;                                            (*  MESS15  *)

    END;        (*  ******  PROCIO  ******  *)

    LINE=OBJECT
     (* THE *LINE* SIMULATES THE PHYSICAL LINE BETWEEN MACHINES *)
     (*  EACH MACHINE REFERENCES TWO DFFERENT LINES :           *)
     (*  ONE FOR INPUT AND ONE FOR OUTPUT . MESSAGES ARE PASSED *)
     (*  CLOCKWISE                                              *)
      PATH 1:(TOLINE;FRLINE) END;
      VAR MESGBUF : MESSAGE;
      ENTRY PROCEDURE TOLINE(M:MESSAGE);
        BEGIN
         MESGBUF:=M
        END;  (*  TOLINE  *)

      ENTRY PROCEDURE FRLINE(VAR M:MESSAGE);
        BEGIN
         M:=MESGBUF
```

```
      END;  (*   FRLINE  *)

   END;  (*    ***** LINE *****    *)

   CONTROLLER=OBJECT
    (* THE ≠CONTROLLER≠ SIMULATES THE MACHINE RUNNING THE  *)
    (* DETECTION AGORITHM. IT HAS 2 BUFFERS: ONE BUFFER IS *)
    (* USED TO STORE INCOMING MESSAGES AND THE SECOND TO   *)
    (* STORE MESSAGES TO BE SENT OUT. THE ≠CREADER≠ PROCESS *)
    (* MONITORS TRAFFIC ON THE INPUT LINE,READS OFF MESSAGES *)
    (* AND STORES IN ≠QUEBUFFER≠. THE ≠STARTUP≠ PROCESS RUNS *)
    (* THE DETECTION AGORITHM AND DOES THE RESOURCE ALLOCATION *)
    (* IT PUTS THE RESPONSE IN THE ≠OUTBUFFER≠ TO BE SENT OUT  *)
    (* BY THE ≠CWRITER≠ PROCESS                              *)
     PATH STTC?NTR END;
    TYPE
      MSGQUEUE=OBJECT    (*  INPUT MESSAGES  *)
        PATH QMAX:(1:(QUEPUT);1:(QUEGET)) END;
        VAR QUEBUFFER:ARRAY[1..QMAX] OF MESSAGE;
            INQQ,OUTQQ:1..QMAX;
        ENTRY PROCEDURE QUEPUT(M:MESSAGE);
          BEGIN
            QUEBUFFER[INQQ]:=M;
            INQQ:=(INQQ MOD QMAX)+1
          END;

        ENTRY PROCEDURE QUEGET(VAR M:MESSAGE; VAR QS : INTEGER);
          BEGIN
            M:=QUEBUFFER[OUTQQ];
            IF OUTQQ>INQQ THEN QS:=(QMAX-OUTQQ)+INQQ
                          ELSE QS:=INQQ-OUTQQ;
            OUTQQ:=(OUTQQ MOD QMAX)+1
          END;  (*  QUEPUT  *)

        INIT; BEGIN
                INQQ:=1;
                OUTQQ:=1
              END;  (*  INIT  *)

    END;   (*  ***  MSGQUE  ***  *)

    OUTQUEUE=OBJECT    (*  MSGES TO BE SENT OUT *)
      PATH QMAX:(1:(OUTPUTT);1:(OUTGET)) END;
      VAR OUTBUFFER:ARRAY[1..QMAX] OF MESSAGE;
          INO,OUTO:1..QMAX;

     ENTRY PROCEDURE OUTPUTT(M:MESSAGE);
      BEGIN
        OUTBUFFER[INO]:=M;
        INO:=(INO MOD QMAX)+1
      END;  (*  OUTPUTT  *)
```

```
ENTRY PROCEDURE OUTGET(VAR M:MESSAGE);
  BEGIN
    M:=OUTBUFFER[OUTO];
    OUTO:=(OUTO MOD QMAX)+1
  END;  (*  OUTGET  *)

INIT; BEGIN
        INO:=1;
        OUTO:=1
      END;  (*  INIT  *)

END;  (*  ***  OUTQUEUE  ***  *)


VAR
    CQUEUE:MSGQUEUE;
    OQUEUE:OUTQUEUE;
    IO:PROCIO;
    PROCSES:ARRAY[NPROC] OF PROCS;
    RESOURCES:ARRAY[MRES] OF RESRC;
    PRTABLE:MAT;
    MARKED:ARRAY[NPROC] OF BOOLEAN;
    DEADLOCK:BOOLEAN;
    N,PP,M,RR : INTEGER;
    P2 : ARRAY[0..NMAX] OF INTEGER;
    REQACCESS:STATUS;
    MSGTEMP:MESSAGE;
    TENTRY:(REQ,REL);
    TOTREQ,TOTDEAD,I,TMSGRECEIVED,TOTMSGSENT,NINITO: INTEGER;
    IFINDR,JFINDP,MAXMSG : INTEGER;


  (*    ********************************     *)
  (*                                         *)
  (*      DETECTION    ROUTINES              *)
  (*                                         *)
  (*    ********************************     *)

   PROCEDURE INITIALIZE;
   (* INITIALIZE THE PROCESS RESOURCE TABLE,THE PROCESS *)
   (* AND RESOURCE TABLES *)
    VAR I,J:INTEGER;
    BEGIN
      FOR I:=0 TO NMAX DO
        BEGIN
          PROCSES[I].PNAME:=-1;
          PROCSES[I].PSITE:=-1;
          PROCSES[I].PSTATE:=BLOCKED;
          FOR J:=0 TO MMAX DO
            BEGIN
              PROCSES[I].RHELD[J].RNAM:=-1;
              PROCSES[I].RHELD[J].RACC:=SHARED
```

```
              END;
          END;
          FOR I:=0 TO MMAX DO
            BEGIN
              RESOURCES[I].RNAME:=-1;
              RESOURCES[I].RSTATUS:=FREE
            END;
          FOR I:=0 TO MMAX DO
            FOR J:=0 TO NMAX DO
            BEGIN
              PRTABLE[I,J].RNK:=-1;
              PRTABLE[I,J].TACCES:=FREE
            END;
          TOTREQ:=0;  NINITO:=0;
          TOTDEAD:=0;
           N:=-1;
            M:=-1;
      END;  (*  INITIALIZE  *)

    FUNCTION NEWP(P:INTEGER):BOOLEAN;
     (* RETURNS TRUE IF THE REQUESTING PROCESS IS NOT IN *)
     (*  ANY OF THE TABLES        *)
     VAR I:INTEGER;
     BEGIN
       IF N<0 THEN NEWP:=TRUE
         ELSE BEGIN
           I:=0;
           WHILE(PROCSES[I].PNAME<>P) AND (I<=N) DO
             I:=I+1;
           IF I>N THEN NEWP:=TRUE
                  ELSE NEWP:=FALSE;
         END;
      END;  (*  NEWP  *)

    FUNCTION NEWR(R:INTEGER):BOOLEAN;
     (* RETURNS TRUE IF THE RESOURCE REQUESTED FOR IS  *)
     (*  NOT IN ANY OF THE TABLES        *)
      VAR I:INTEGER;
      BEGIN
       IF M<0 THEN NEWR:=TRUE
          ELSE
          BEGIN
            I:=0;
            WHILE (RESOURCES[I].RNAME<>R) AND (I<=M) DO
                I:=I+1;
            IF I>M THEN NEWR:=TRUE
                   ELSE NEWR:=FALSE;
          END;
     END;  (*  NEWR  *)

    FUNCTION FINDP(P:INTEGER):INTEGER;
     (* RETURNS AN INDEX TO A PROCESS IN THE PROCESS TABLE *)
       VAR I : INTEGER;
```

```
    BEGIN
     I:=0;
     WHILE PROCSES[I].PNAME<>P DO
       I:=I+1;
     FINOP:=I;
    END;  (*  FINOP  *)

  FUNCTION FINDR(R:INTEGER):INTEGER;
   (* RETURNS AN INDEX TO A RESOURCE IN THE RESOURCE TABLE *)
    VAR I:INTEGER;
    BEGIN
     I:=0;
     WHILE RESOURCES[I].RNAME<>R DO
       I:=I+1;
     FINDR:=I;
    END;  (*  FINDR  *)

  PROCEDURE REMOVE(IND,PR:INTEGER);
   (* DELETES A PROCESS AND A RESOURCE FROM THEIR RESPECTIVE *)
   (* TABLES. A PROCESS IS DELETED IF IT HAS NO OUTSTANDING  *)
   (* REQUEST AND DOES NOT HOLD ANY RESOURCE. A RESOURCE IS  *)
   (* DELETED IF THERE IS NO REQUEST FOR IT    *)
    VAR I,J,K,L:INTEGER;
    BEGIN
     IF IND=0 THEN   (*  PROCESS  *)
      BEGIN
       I:=FINDP(PR);
       L:=I+1;
       FOR J:=L TO NMAX DO
        BEGIN
         WITH PROCSES[I] DO
              BEGIN
               PNAME:=PROCSES[J].PNAME;
               PSITE:=PROCSES[J].PSITE;
               PSTATE:=PROCSES[J].PSTATE
              END;
         FOR K:=0 TO MMAX DO
          BEGIN
           PROCSES[I].RHELD[K].RNAM:=PROCSES[J].RHELD[K].RNAM;
           PROCSES[I].RHELD[K].RACC:=PROCSES[J].RHELD[K].RACC
          END;
         I:=I+1
        END;
        WITH PROCSES[NMAX] DO
          BEGIN
            PNAME:=-1;
            PSITE:=-1;
            PSTATE:=BLOCKED
          END;
        N:=N-1
      END
     ELSE  (*  RESOURCE  *)
```

```
        BEGIN
         I:=FINDR(PR);
         L:=I+1;
         FOR J:=L TO MMAX DO
          BEGIN
           RESOURCES[I].RNAME:=RESOURCES[J].RNAME;
           RESOURCES[I].RSTATUS:=RESOURCES[J].RSTATUS;
           I:=I+1
          END;
         RESOURCES[MMAX].RNAME:=-1;
         RESOURCES[MMAX].RSTATUS:=FREE;
         M:=M-1
        END;
    END;  (*   REMOVE   *)

    PROCEDURE REMOVECOL(P,IND:INTEGER);
      (* REMOVE COLUMN CORRESPONDING TO PROCESS P FROM THE    *)
      (* PR TABLE. THE PR TABLE IS ONLY MAINTAINED FOR PROCESSES *)
      (* THAT HAVE OUTSTANDING REQUESTS AND/OR HAVE ACCESS TO  *)
      (* A RESOURCE      *)
       VAR I,J,K,L : INTEGER;
       BEGIN
        FOR I:=0 TO M DO
         BEGIN
          J:=IND;
          L:=IND+1;
          FOR K:=L TO NMAX DO
           BEGIN
            PRTABLE[I,J].RNK:=PRTABLE[I,K].RNK;
            PRTABLE[I,J].TACCES:=PRTABLE[I,K].TACCES;
            J:=J+1
           END;
          PRTABLE[I,NMAX].RNK:=-1;
          PRTABLE[I,NMAX].TACCES:=FREE
         END;
         K:=J;
         REMOVE(K,P);
      END;  (*  REMOVECOL   *)

    PROCEDURE REMOVEROW(R,IND:INTEGER);
      (* REMOVES ROW CORRESPONDING TO RESOURCE R FROM PR TABLE *)
      VAR I,J,K,L:INTEGER;
      BEGIN
       FOR I:=0 TO N DO
        BEGIN
         J:=IND;
         L:=IND+1;
         FOR K:=L TO MMAX DO
          BEGIN
           PRTABLE[J,I].RNK:=PRTABLE[K,I].RNK;
           PRTABLE[J,I].TACCES:=PRTABLE[K,I].TACCES;
           J:=J+1
          END;
```

```
      END;
     FOR I:=0 TO N DO
       PRTABLE[MMAX,I].RNK:=-1;
       PRTABLE[MMAX,I].TACCES:=FREE;
     K:=1;
     REMOVE(K,R);
    END;   (*   REMOVEROW   *)

    PROCEDURE ALLOCATER;
      (* ALLOCATES RESOURCES TO WAITING PROCESSES *)
      VAR ROW,I,J : INTEGER;
      BEGIN
       ROW:=FINDR(RR);
       FOR J:=0 TO N DO
        IF PRTABLE[ROW,J].RNK<>-1 THEN
           PRTABLE[ROW,J].RNK:=PRTABLE[ROW,J].RNK-1;
        FOR J:=0 TO N DO
          IF PRTABLE[ROW,J].RNK=0 THEN
            BEGIN
             (* ALLOCATE RESOURCES TO PROCESS WITH INDEX J *)
             PROCSES[J].PSTATE:=RUNNING;
              (* SEND RESPONSE MSG  *)
              WITH MSGTEMP DO
                 BEGIN
                   MSGTYPE:=RESPONSE;
                   MSGID:=PROCSES[J].PSITE;
                   RESID:=RR;
                   PROCID:=PROCSES[J].PNAME-PROCSES[J].PSITE*1000
                 END;
             I:=0;
              WHILE PROCSES[J].RHELD[I].RNAM<>RR DO
                I:=I+1;
              RESOURCES[ROW].RSTATUS:=PROCSES[J].RHELD[I].RACC;
             MSGTEMP.ACCESSID:=RESOURCES[ROW].RSTATUS;
             PRTABLE[ROW,J].TACCES:=RESOURCES[ROW].RSTATUS;
             (* SEND RESPONSE TO WAKE UP REQUESTING PROCESS *)
             OQUEUE.OUTPUT(MSGTEMP)
             (* WRITE MESSAGE  *)
             (* IO.MESS9(MSGTEMP.MSGID,MSGTEMP.PROCID,RR) *)
            END;
      END;  (* ALLOCATER  *)

    PROCEDURE RESREL;
    (* HANDLES RESOURCE RELEASE  *)
      VAR I,J,K,L,I1,TRANK:INTEGER;
         SW,SW1:BOOLEAN;
    BEGIN
       I:=FINDP(PP);
       J:=0;
       WHILE  PROCSES[I].RHELD[J].RNAM<>RR DO
         J:=J+1;
       L:=J+1;
```

```
FOR K:=L TO MMAX DO
   BEGIN
    PROCSES[I].RHELD[J].RNAM:=PROCSES[I].RHELD[K].RNAM;
    PROCSES[I].RHELD[J].RACC:=PROCSES[I].RHELD[K].RACC;
    J:=J+1
   END;
PROCSES[I].RHELD[MMAX].RNAM:=-1;
PROCSES[I].RHELD[MMAX].RACC:=SHARED;
I1:=FINDR(RR);
TRANK:=PRTABLE[I1,I].RNK;
IF PROCSES[I].RHELD[0].RNAM=-1 THEN
    REMOVECOL(PP,I)
    ELSE BEGIN
             PRTABLE[I1,I].RNK:=-1;
             PRTABLE[I1,I].TACCES:=FREE
           END;
SW:=FALSE;
FOR J:=0 TO NMAX DO
 IF TRANK>0 THEN
   IF PRTABLE[I1,J].RNK=TRANK THEN SW:=TRUE;
IF NOT SW THEN
   FOR J:=0 TO NMAX DO
    IF (TRANK>0) AND (PRTABLE[I1,J].RNK>TRANK) THEN
     PRTABLE[I1,J].RNK:=PRTABLE[I1,J].RNK-1;
(*  ANY MORE PROCSES USING RR  *)
SW:=FALSE;
FOR J:=0 TO N DO
   IF PRTABLE[I1,J].RNK=0 THEN SW:=TRUE;
IF NOT SW THEN
   BEGIN
    SW1:=FALSE;
    FOR J:=0 TO N DO
     IF PRTABLE[I1,J].RNK>0 THEN SW1:=TRUE;
    IF SW1 THEN ALLOCATER
           ELSE REMOVEROW(RR,I1);
   END;
END;  (*  RESREL  *)

PROCEDURE ROLLB;
 (*  ABORTS A PROCESS AND ALLOCATES ALL RESOURCES IT  *)
 (*  TO OTHER WAITING PROCESSES                       *)
 VAR I,J,K:INTEGER;
 BEGIN
  K:=-1;
  J:=0;
  WHILE PROCSES[JFINDP].RHELD[J].RNAM<>-1 DO
   BEGIN
    K:=K+1;
    P2[K]:=PROCSES[JFINDP].RHELD[J].RNAM;
    J:=J+1
   END;
  FOR I:=0 TO K DO
   BEGIN
```

```
       RR:=P2[I];
       RESREL
      END;
  END;  (* ROOLB *)


      PROCEDURE HORIZONTAL(VAR R,H : INTEGER);
      (* THE HORIZONTAL ALGORITHM  *)
       (* IT RETURNS IN P2 ALL PROCESSES WITH RANKOF ZERO ON *)
       (* R: H INDCATES THE NUMBER OF PROCESSES WITH THE RANK *)
        VAR I,J:INTEGER;
        BEGIN
          H:=-1;
          I:=FINDR(R);
          FOR J:=0 TO N DO
            IF PRTABLE[I,J].RNK=0 THEN
              BEGIN
                H:=H+1;
                P2[H]:=PROCSES[J].PNAME
              END;
        END;  (*   HORIZONTAL   *)

      PROCEDURE VERTICAL(VAR VP,VR : INTEGER;VAR V : BOOLEAN);
    (* THE VERTICAL ALGORITHM   *)
    (* V IS TRUE IF VR EXISTS SUCH THAT VP'S RANK>0   *)
        VAR I,J:INTEGER;
        BEGIN
         I:=FINDP(VP);
         FOR J:=0 TO M DO
          IF (PRTABLE[J,I].RNK>0) AND (NOT MARKED[I]) THEN
           BEGIN
            V:=TRUE;
             VR:=RESOURCES[J].RNAME;
             MARKED[I]:=TRUE
           END;
        END;  (*  VERTICAL  *)

   PROCEDURE DTECT(VAR PI,RJ : INTEGER);
  (*  PERFORMS THE HORIZONTAL AND VERTICAL DEADLOCK DETECTION *)
  (* RETURNS 'DEADLOCK' TRUE IF DEADLOCK EXISTS, FALSE OTHERWISE *)
  (*  IT USES A STACK TO PERFORM THE ALGORITHM     *)
     VAR STK:ARRAY[1..35] OF INTEGER;
          SW,DONE,V:BOOLEAN;
          STKPTR,I,H,P1 : INTEGER;
      BEGIN   (*  DTECT     *)
        DONE:=FALSE;
        STKPTR:=0;
        WHILE NOT DONE DO
         BEGIN
          HORIZONTAL(RJ,H);
          SW:=FALSE;
          FOR I:=0 TO H DO
           IF P2[I]=PP THEN SW:=TRUE;
```

```
            IF SW THEN
              BEGIN
               DEADLOCK:=TRUE;
               DONE:=TRUE
              END
              ELSE
              BEGIN
               WHILE H>=0 DO
                 BEGIN
                   STK[STKPTR]:=P2[H];
                   STKPTR:=STKPTR+1;
                   H:=H-1
                  END;
                 V:=FALSE;
                 WHILE (STKPTR>0) AND (NOT V) DO
                  BEGIN
                   STKPTR:=STKPTR-1;
                   P1:=STK[STKPTR];
                   VERTICAL(P1,RJ,V)
                  END;
                 IF (STKPTR=0) AND (NOT V) THEN DONE:=TRUE;
              END;
            END;
      END; (*  DTECT  *)

   PROCEDURE HV;
   (*  INITIATES THE DETECTION ALGORITHM *)
      VAR PS,RS : INTEGER;

   BEGIN   (*  HV  *)
     DEADLOCK:=FALSE;
     FOR PS:=0 TO N DO
      MARKED[PS]:=FALSE;
     PS:=PP;
     RS:=RR;
     DTECT(PS,RS);
   END;  (*   HV   *)

   PROCEDURE RANK;
   (* ASSIGNS A RANK TO A REQUESTING PROCESS  *)
   (* RERANK WILL REASSIGN THE RANK IF NECESSARY  *)
     VAR K,L : INTEGER;
     BEGIN
      K:=-1;
      FOR L:=0 TO N DO
       IF PRTABLE[IFINDR,L].RNK>K THEN
         K:=PRTABLE[IFINDR,L].RNK;
     PRTABLE[IFINDR,JFINDP].RNK:=K+1;
     PRTABLE[IFINDR,JFINDP].TACCES:=REQACCESS;
     END;  (*  RANK  *)

   PROCEDURE RERANK(THELD:STATUS);
   (* RESOURCE RR IS BEING HELD THELD    *)
```

```
(* REASSIGNS A RANK TO THE NEW REQUEST *)
(* IF THE REQUEST IS FOR SHARED ACCESS *)
VAR WAITEXCL,WAITSHARED,SW : BOOLEAN;
    I,K : INTEGER;
BEGIN
  WAITEXCL:=FALSE;
  WAITSHARED:=FALSE;
  IF REQACCESS=SHARED THEN
    BEGIN
     FOR I:=0 TO N DO
       IF I<>JFINOP THEN
          WITH PRTABLE[IFINOR,I] DO
           BEGIN
          IF (RNK>0) AND
             (TACCES=EXCLUSIVE) THEN
               WAITEXCL:=TRUE;
          IF (RNK>0) AND
             (TACCES=SHARED) THEN
               WAITSHARED:=TRUE
          END;
     SW:=FALSE;
     IF (THELD=EXCLUSIVE) AND (WAITSHARED)
        THEN SW:=TRUE ELSE
        IF (THELD=SHARED) AND (WAITEXCL) THEN SW:=TRUE;
     IF SW THEN
      FOR I:=0 TO N DO
       WITH PRTABLE[IFINOR,I] DO
        IF (RNK>0) AND
          (TACCES=SHARED) AND (I<>JFINOP)
          THEN PRTABLE[IFINOR,JFINOP].RNK:=RNK;
 END;
END; (* RERANK *)

PROCEDURE RESFREE(VAR RFREE:BOOLEAN;VAR THELD:STATUS);
    VAR I : INTEGER;
        SW:BOOLEAN;
    BEGIN
     RFREE:=FALSE;
     THELD:=RESOURCES[IFINOR].RSTATUS;
     IF THELD=FREE THEN RFREE:=TRUE ELSE
     IF (THELD=SHARED) AND (REQACCESS=SHARED) THEN
      BEGIN
        (* CHECK IF THERE IS ANY PROCESS WAITING ON RR FOR *)
        (* EXCLUSVE ACCESS       *)
        SW:=FALSE;
        FOR I:=0 TO N DO
          IF (PRTABLE[IFINOR,I].RNK>0) AND
             (PRTABLE[IFINOR,I].TACCES=EXCLUSIVE) THEN SW:=TRUE;
        IF NOT SW THEN RFREE:=TRUE
      END;
    END; (* RESFREE *)
```

```
PROCEDURE RESREQ;
(* PROCESSES PP REQUEST FOR RR  AND DOES THE RESOURCE ALLOCATION *)
  VAR I,J:INTEGER;
      RFREE:BOOLEAN;
      THELD:STATUS;
   BEGIN
    TOTREQ:=TOTREQ+1;
    IF NEWP(PP) THEN
     BEGIN
      N:=N+1;
      PROCSES[N].PNAME:=PP;
      PROCSES[N].PSITE:=MSGTEMP.MSGID
     END;
     IF NEWR(RR) THEN
      BEGIN
       M:=M+1;
       RESOURCES[M].RNAME:=RR;
       RESOURCES[M].RSTATUS:=FREE
      END;
     IFINOR:=FINOR(RR);
     JFINOP:=FINOP(PP);
     J:=1;
     WHILE PROCSES[JFINOP].RHELD[J].RNAM<>-1 DO
       J:=J+1;
     PROCSES[JFINOP].RHELD[J].RNAM:=RR;
     PROCSES[JFINOP].RHELD[J].RACC:=REQACCESS;
     RESFREE(RFREE,THELD);
     IF RFREE THEN
      BEGIN
        RESOURCES[IFINOR].RSTATUS:=REQACCESS;
        PRTABLE[IFINOR,JFINOP].RNK:=0;
         PRTABLE[IFINOR,JFINOP].TACCES:=REQACCESS;
        PROCSES[JFINOP].PSTATE:=RUNNING;
        MSGTEMP.MSGTYPE:=RESPONSE;
        OQUEUE.OUTPUTT(MSGTEMP)
        (* IO.MESS9(MSGTEMP.MSGID,MSGTEMP.PROCID,RR)   *)
       END
       ELSE
        BEGIN
         PROCSES[JFINOP].PSTATE:=BLOCKED;
         NINITO:=NINITO+1;
          RANK;
          HV;
         IF NOT DEADLOCK THEN
           BEGIN
             IF REQACCESS=SHARED THEN RERANK(THELD);
             MSGTEMP.MSGTYPE:=NOTFREE;
             OQUEUE.OUTPUTT(MSGTEMP)
             (* IO.MESS11(MSGTEMP.MSGID,MSGTEMP.PROCID,RR) *)
           END
           ELSE
           BEGIN
```

```
                TOTDEAD:=TOTDEAD+1;
                MSGTEMP.MSGTYPE:=ROLLBACK;
                OQUEUE.OUTPUTT(MSGTEMP);
                IO.MESS10(MSGTEMP.MSGID,MSGTEMP.PROCID,RR);
                IO.MESS8(4,TOTDEAD,TOTREQ);
                ROLLB ;
              END;
          END;
END;   (*  RESREQ   *)
PROCEDURE RESALLOC;
  VAR I,J : INTEGER;
  (*  INITIATES RESOURCE ALLOCATION *)
  BEGIN
    IF TENTRY=REQ THEN RESREQ
    ELSE BEGIN
              (* I:=PP MOD 1000;
              J:=PP DIV 1000;
              IO.MESS12(J,I,RR);   *)
              RESREL
            END;
  END;   (*  RESALLOC  *)

PROCESS CWRITER(OUTLL:LINE;TMAXP:INTEGER);
  (* WRITES RESPONSE TO OUTPUT LINE  *)
  VAR M:MESSAGE;
      CWRITING : BOOLEAN;
      TOT:INTEGER;
BEGIN
  CWRITING:=TRUE;    TOT:=0;
  WHILE CWRITING DO
   BEGIN
    OQUEUE.OUTGET(M);
    OUTLL.TOLINE(M);
    TOTMSGSENT:=TOTMSGSENT+1;
    IF M.MSGTYPE=ATERMINATE THEN TOT:=TOT+1;
    IF TOT=TMAXP THEN CWRITING:=FALSE
   END;
   IO.MESS1(4,TOTMSGSENT)
END;  (*  CWRITER   *)

PROCESS STARTUP(TOTP:INTEGER);
(* RUNS THE RESOURCE ALLOCATION . TO IDENTIFY EACH PROCESS *)
(* THE SITE OF THE PROCESS IS ATTACHED TO THE PROCESSID. *)
(* PROCESSES ALL REQUESTS IN *QUEBUFFER*.AND WRITES RESPONSE *)
(* TO THE *OUTBUFFER*         *)
  VAR MTOT,MTOTL : INTEGER;
      STARTING,SW:BOOLEAN;
      QS,TOT : INTEGER;
  BEGIN
    INITIALIZE;
    STARTING:=TRUE;   TOT:=0;
```

```
      SW:=TRUE;
      WHILE STARTING DO
    BEGIN
      CQUEUE.QUEGET(MSGTEMP,QS);
       IF MSGTEMP.MSGTYPE=ATERMINATE THEN
           BEGIN
             TOT:=TOT+1;    OQUEUE.OUTPUTT(MSGTEMP)
            END  ELSE
          BEGIN
       MSGTEMP.QUESIZE:=QS;
       PP:=MSGTEMP.MSGID*1000+MSGTEMP.PROCID;
       RR:=MSGTEMP.RESID;
       REQACCESS:=MSGTEMP.ACCESSID;
       IF MSGTEMP.MSGTYPE=AREQUEST
          THEN TENTRY:=REQ
          ELSE TENTRY:=REL;
       RESALLOC
      END;
     IF TOT=TOTP THEN STARTING:=FALSE
    END;
      IC.MESS8(4,TOTDEAD,TOTREQ);
      IC.MESS14(4,0,NINITD)
    END:    (* STARTUP  *)

  PROCESS CREADER(INLINE:LINE; TOTP:INTEGER);
    (* MONITORS THE INPUT LINE FOR A MESSAGE. WRITES  *)
    (* MESSAGE TO THE #QUEBUFFER# TO BE PROCESSED BY STARTUP *)
    VAR  M:MESSAGE;
         CREADING : BOOLEAN;
         TOT:INTEGER;
   BEGIN
     CREADING:=TRUE;   TOT:=0;
     WHILE CREADING DO
      BEGIN
        INLINE.FRLINE(M);
        CQUEUE.QUEPUT(M);
        TOTMSGRECEIVED:=TOTMSGRECEIVED+1;
        IF M.MSGTYPE=ATERMINATE THEN TOT:=TOT+1;
        IF TOT=TOTP THEN CREADING:=FALSE
      END;
      IC.MESS2(4,TOTMSGRECEIVED)
   END;  (*  CREADER   *)

  ENTRY PROCEDURE SITCONTR(INLINE,OUTLINE:LINE;MAXP:INTEGER);
   (* STARTS ALL THE PROCESSES IN CONTROLLER  *)
   BEGIN
     TOTMSGRECEIVED:=0;
     TOTMSGSENT:=0;
     CREADER(INLINE,MAXP);
     STARTUP(MAXP);
     CWRITER(OUTLINE,MAXP);
  END;   (*  SITCONTR   *)
```

```
END; (*  *****  CONTROLLER  *****  *)

PMACHINE=OBJECT
  (* PMACHINE SIMULATES THE MACHINES ON WHICH THE USER PROCESSES *)
  (* ARE RUNNING. EACH MACHINE HAS 2 BUFFERS. INCOMING MESSAGES  *)
  (* ARE READ BY ≠READLINE≠; IF MESSAGE IS TO BE PASSED ON IT IS *)
  (* PUT IN THE ≠BUFFER≠. EACH USER PROCESS RUNNING ON THE       *)
  (* MACHINE IS ASSIGNED A BUFFER LOCATION IN THE SECOND BUFFER  *)
  (* ≠PRBUF≠,ON WHICH IT WAITS FOR A RESPONSE TO ITS REQUEST    *)
  PATH SITMACH END;

  TYPE
    BUFFER=OBJECT
     PATH BMAX:(1:(BUFPUT);1:(BUFGET)) END;
     VAR IOBUFFER:ARRAY[1..BMAX] OF MESSAGE;
         INPP,OUTP:1..BMAX;
     ENTRY PROCEDURE BUFPUT(M:MESSAGE);
        BEGIN
          IOBUFFER[INPP]:=M;
          INPP:=(INPP MOD BMAX) + 1
         END;  (*   BUFPUT  *)

  ENTRY PROCEDURE BUFGET(VAR M:MESSAGE);
      BEGIN
        M:=IOBUFFER[OUTP];
        OUTP:= (OUTP MOD BMAX)+1
      END;   (*     BUFGET    *)

  INIT; BEGIN
          INPP:=1;
          OUTP:=1
        END;   (*  INIT  *)

 END;  (*  *****  BUFFER  *****  *)

 PRBUF = OBJECT
   PATH 1:(PBUFPUT;PBUFGET) END;
   VAR PRBUFFER: MESSAGE;
     ENTRY PROCEDURE PBUFPUT(M:MESSAGE);
        BEGIN
          PRBUFFER:=M
        END;   (*   PBUFPUT  *)

   ENTRY PROCEDURE PBUFGET(VAR M:MESSAGE);
       BEGIN
         M:=PRBUFFER
     END;    (*  PBUFGET    *)

 END;  (*   *****  PRBUFFER  *****   *)

   VAR BUFF:BUFFER;
     PBUFF:ARRAY[1..7] OF PRBUF;  (* ASSUME MAX OF 7 PROCESS PER SITE*)
```

```
IOW:PROCIO:

FUNCTION RAND(VAR SEED:REAL;MOOP:INTEGER):INTEGER;
    CONST
      P=2147483647;
      A=16807;
    VAR ISEED:INTEGER;
    BEGIN
      ISEED:=TRUNC(SEED);
      SEED:=(A*ISEED) MOD P;
      ISEED:=TRUNC(SEED) MOD MOOP;
      RAND:=ISEED
    END;  (*  RAND  *)

FUNCTION RANDOM(VAR S:REAL):REAL;
      VAR ISEED:INTEGER;
      BEGIN
        ISEED:=TRUNC(S);
        ISEED:=(ISEED*899) MOD 32767;
        S:=ISEED;
        RANDOM:=S/32767.0
      END;   (*   RANDOM   *)

PROCESS READLINE(WHO:RSITE:INLINE:LINE:MAXP : INTEGER);
(* MONITORS INPUT LINE FOR ALL INCOMING MESSAGES: IF MESSAGE *)
(* IS FOR PROCESS I RUNNING ON THE LOCAL SITE, IT UNBLOCKS  *)
(* PROCESS I TO ACCEPT THE RESPONSE      *)
VAR M:MESSAGE;
    I,TOT : INTEGER;
    TOTRESP,TOTROLLB,TOTCOMPL,TOTAREQ,TOTNFREE,TOTMREC: INTEGER;
    READING:BOOLEAN;
BEGIN
  READING:=TRUE;    TOT:=0;
  TOTRESP:=0; TOTROLLB:=0; TOTCOMPL:=0; TOTAREQ:=0;
  TOTNFREE:=0;  TOTMREC:=0;
  WHILE READING DO
   BEGIN
    INLINE.FRLINE(M);
    IF M.MSGTYPE=ATERMINATE THEN
        BEGIN
          TOT:=TOT+1;
          IF M.MSGID<>WHO THEN BUFF.BUFPUT(M);
          IF TOT=MAXP THEN READING:=FALSE
      END ELSE
      BEGIN
        CASE M.MSGTYPE OF
         RESPONSE: TOTRESP:=TOTRESP+1;
         COMPLETION: TOTCOMPL:=TOTCOMPL+1;
         AREQUEST  : TOTAREQ:=TOTAREQ+1;
         NOTFREE  : TOTNFREE:=TOTNFREE+1;
         ROLLBACK:
           BEGIN
```

```
                TOTROLLB:=TOTROLLB+1;
                IOW.MESS15(WHO,TOTNFREE,99,TOTRESP,TOTROLLB,
                 TOTCOMPL,TOTAREQ,99,99,99)
              END
            END;
            TOTMREC:=TOTMREC+1;
          IF M.MSGID=WHO THEN (*  WAKE UP OWNER PROCESS *)
              PBUFF[M.PROCID].PBUFPUT(M)
            ELSE (*  PASS MESSAGE ON  *)
              BUFF.BUFPUT(M)
          END;
        END;
        IOW.MESS2(WHO,TOTMREC);
        IOW.MESS15(WHO,TOTNFREE,99,TOTRESP,TOTROLLB,TOTCOMPL,
                TOTAREQ,99,99,99)
      END; (* READLINE *)

    PROCESS WRITER(OUTLINE:LINE; TOTP : INTEGER);
     (*  WRITES MESSAGE TO OUTPUT LINE *)
       VAR M:MESSAGE;
           WRITING:BOOLEAN;
           TOT : INTEGER;
        BEGIN
          WRITING:=TRUE;    TOT:=0;
          WHILE WRITING DO
            BEGIN
              BUFF.BUFGET(M);
              OUTLINE.TOLINE(M);
              IF M.MSGTYPE=ATERMINATE THEN TOT:=TOT+1;
              IF TOT=TOTP THEN WRITING:=FALSE
            END;
        END;  (* WRITER *)

    PROCESS PPROCSS(SITE,LPROCID,TOTMAXR:INTEGER;LAMDA,MUU:REAL;
              MAXREQ,WACCES,THRUPUT : INTEGER);
        (* SIMULATE A LOCAL PROCESS ACTIVITIES  *)
      LABEL 1,2;
      TYPE
        LRES=RECORD
         LRNAME:INTEGER;
         TACCESS:STATUS
        END;
     VAR
       RESRCES:ARRAY[1..20] OF LRES;
       CLOCK,TRELEASE,TREQUEST,LAMDABAR,MUUBAR,SEEDR,SEED:REAL;
       TEMP,T2,TBEFORE,TOTSECS,XXX:REAL;
       NUMRES,RR,MP,I,J,TOTSENT,TOTDELAY,RELPTR,REQPTR :INTEGER;
       OUTREQ,THRUBEFORE,THRUAFTER : INTEGER;
       TESTCASE,TS,TD,MPPP,TOTLOC : INTEGER;
       MAINSW,SW,SW1,GREATR,PROCESING,AGAN :BOOLEAN;
       MYMSG:MESSAGE;
       ACCTYPE:STATUS;
```

```
PROCEDURE GENREQ;
      BEGIN (* GENERATE NEW RESOURCE  *)
       SW:=FALSE;
       WHILE NOT SW DO
         BEGIN
           RR:=RAND(SEED,TOTMAXR)+1;
           IF (REQPTR=0) OR (OUTREQ=0) THEN
             SW:=TRUE ELSE
             BEGIN
              SW1:=FALSE;
              J:=(RELPTR MOD 20)+1;
              FOR I:=1 TO OUTREQ DO
               BEGIN
                 IF RESRCES[J].LRNAME=RR THEN SW1:=TRUE;
                 J:=(J MOD 20)+1
               END;
              IF NOT SW1 THEN SW:=TRUE
             END
           END;
         (* TYPE OF ACCESS  *)
         IF WACCES=1 THEN ACCTYPE:=EXCLUSIVE ELSE
           BEGIN
              TEMP:=RANDOM(SEEDR);
              IF TEMP>=0.5 THEN ACCTYPE:=EXCLUSIVE ELSE
                                ACCTYPE:=SHARED
           END;
         REQPTR:=(REQPTR MOD 20)+1;  OUTREQ:=OUTREQ+1;
         RESRCES[REQPTR].LRNAME:=RR;
         RESRCES[REQPTR].TACCESS:=ACCTYPE;
         IF RR=SITE THEN TOTLOC:=TOTLOC+1;
    (* SEND REQUEST  *)
     WITH MYMSG DO
      BEGIN
       MSGID:=SITE; PROCID:=LPROCID;
       QUESIZE:=0;
       MSGTYPE:=AREQUEST; RESID:=RR;
       ACCESSID:=ACCTYPE
      END;
     IOW.MESS3(SITE,LPROCID,RR,ACCTYPE);
     BUFF.BUFPUT(MYMSG);
     J:=TIME; TOTSENT:=TOTSENT+1;
     (* TBEFORE:=SIN(XXX);  *)
     PBUFF[LPROCID].PBUFGET(MYMSG);
     (* PROCESS BLOCKED WAITING FOR RESPONSE  *)
     (* TEMP:=SIN(XXX)-TBEFORE; *)
     (* TS:=TRUNC(TEMP); *)
      TD:=TIME-J;          MPPP:=MYMSG.QUESIZE;
     IOW.MESS13(SITE,LPROCID,TD,MPPP);
     IF MYMSG.MSGTYPE=ROLLBACK THEN
        BEGIN  MP:=MYMSG.MSGID
         (* IOW.MESS4(SITE,LPROCID,MYMSG.RESID)  *)
       END
         ELSE
```

```
    BEGIN
        IF MYMSG.MSGTYPE=NOTFREE THEN PBUFF[LPROCID].PBUFGET(MYMSG);
        (* IOW.MESS5(SITE,LPROCID,MYMSG.RESID); *)
      END;
   END;  (* GENREQ   *)
  PROCEDURE ASSREL;
   BEGIN
    RELPTR:=(RELPTR MOD 20)+1;   OUTREQ:=OUTREQ-1;
     WITH MYMSG DO
      BEGIN
       PROCID:=LPROCID;   MSGTYPE:=COMPLETION;
       MSGID:=SITE;
       RESID:=RESRCES[RELPTR].LRNAME;  ACCESSID:=FREE
       END
  END;   (* RELPTR   *)

  BEGIN
   TOTSENT:=0; TOTDELAY:=0; XXX:=5.0;    PROCESING:=TRUE;
   TOTSECS:=0.0;
   TOTLOC:=0; SEEDR:=31415.0/SITE; SEED:=SITE;
   TRELEASE:=0.0; TREQUEST:=0.0; CLOCK:=0.0;
   THRUBEFORE:=TIME;
   (*  RELPTR POINTS TO THE LAST RESOURCE RELEASED
       REQPTR POINTS TO THE LAST RESOURCE REQUESTED FOR   *)
 1: RELPTR:=0; OUTREQ:=0;  REQPTR:=0;    GREATR:=FALSE;
     MAINSW:=FALSE; AGAN:=FALSE;
   WHILE PROCESING DO
      BEGIN
       MP:=-1;
       IF (TOTSENT>=MAXREQ)   THEN GOTO 2;
       GENREQ;
       IF MP<>-1 THEN
          BEGIN
            I:=1000; J:=RAND(SEED,I)+100;
            DELAY(J);
            IF THRUPUT=1 THEN TOTSENT:=0;
            GOTO 1
          END;
       (* GENERATE TIME OF NEXT RELEASE   *)
       MUUBAR:=(-1.0/MUU)*LN(RANDOM(SEEDR));
       TRELEASE:=CLOCK+MUUBAR;
       (* GENERATE TIME OF NEXT REQUEST *)
       LAMDABAR:=(-1.0/LAMDA)*LN(RANDOM(SEEDR));
       TREQUEST:=CLOCK+LAMDABAR;
       MAINSW:=TRUE;
       WHILE MAINSW DO
        BEGIN
         IF TRELEASE>TREQUEST THEN TESTCASE:=1;
         IF TRELEASE=TREQUEST THEN TESTCASE:=2;
         IF TRELEASE<TREQUEST THEN TESTCASE:=3;
         CASE TESTCASE OF
           1:  (* TRELEASE>TREQUEST   *)
```

```
        BEGIN
           TEMP:=LAMDABAR*100.0;
           I:=TRUNC(TEMP);  T2:=I+0.49;
           IF TEMP>T2 THEN I:=I+1;
           DELAY(I);  CLOCK:=TREQUEST;
           IF  (TOTSENT>=MAXREQ) THEN
                      GOTO 2;
           MUUBAR:=TRELEASE-TREQUEST;
             (* GENERATE REQUEST  *)
           IF OUTREQ>=ICIMAXR THEN
            BEGIN
             (* REQUEST BUT RES HELD EQUALS MAX RES *)
              TRELEASE:=TREQUEST; ASSREL;
               BUFF.BUFPUT(MYMSG);
              (* IOW.MESS6(SITE,LPROCID,RESRCES[RELPTR].LRNAME); *)
              MAINSW:=FALSE
           END ELSE
           BEGIN
            MP:=-1;  GENREQ;
             IF MP<>-1 THEN
                           BEGIN
              I:=1000;  J:=RAND(SEED,I)+100;
              DELAY(J);
              IF THRUPUT=1 THEN TOTSENT:=0;
              GOTO 1
           END;
            (* GENERATE TIME OF NEXT REQUEST  *)
            LAMDABAR:=(-1.0/LAMDA)*LN(RANDOM(SEEDR));
            TREQUEST:=CLOCK+LAMDABAR;  MAINSW:=TRUE
       END
    END;  (* TESTCASE=1  *)
    2: (* TRELEASE=TREQUEST  *)
      BEGIN
        CLOCK:=TRELEASE;
        TEMP:= LAMDABAR*100.0;
        I:=TRUNC(TEMP); T2:=I+0.49;
        IF TEMP>T2 THEN I:=I+1; DELAY(I);
        (* RELEASE RESOURCE IF ANY  *)
        IF OUTREQ>0 THEN
          BEGIN
          ASSREL;
          BUFF.BUFPUT(MYMSG)
      (* IOW.MESS6(SITE,LPROCID,RESRCES[RELPTR].LRNAME) *)
       END;
       MAINSW:=FALSE;
       IF  (TOTSENT>=MAXREQ) THEN
                  GOTO 2;
     END;
 3:  (* TRELEASE<TREQUEST  *)
     BEGIN
      TEMP:=MUUBAR*100.0;
      I:=TRUNC(TEMP);  T2:=I+0.49;
      IF TEMP>T2 THEN I:=I+1;   DELAY(I);
```

```
IF OUTREQ<=0 THEN
 BEGIN (* NO RES TO RELEASE  *)
  CLOCK:=TREQUEST;
  TEMP:=(TREQUEST-TRELEASE)*100.0;
  I:=TRUNC(TEMP);  T2:=I+0.49;
  IF TEMP>T2 THEN I:=I+1;   DELAY(I);   MAINSW:=FALSE
  END  ELSE
  BEGIN  (* RELEASE  RESOURCE  *)
   CLOCK:=TRELEASE;  ASSREL;
    BUFF.BUFPUT(MYMSG);
   (* IOW.MESS6(SITE,LPROCID,RESRCES[RELPTR].LRNAME); *)
   LAMDABAR:=TREQUEST-TRELEASE;
   (* GENERATE TIMOF NEXT RELEASE *)
    MUUBAR:=(-1.0/MUU)*LN(RANDOM(SEEDR));
    TRELEASE:=CLOCK+MUUBAR;
    IF  (TOTSENT>=MAXREQ) THEN
        GOTO 2
 END
 END;  (* TRELEASE < TREQUEST  *)
END;  (*  CASE  *)
END; (*  MAINSW  *)
END;  (* PROCESING  *)
2: IF OUTREQ>0 THEN
     WHILE OUTREQ>0 DO
       BEGIN
         ASSREL;  BUFF.BUFPUT(MYMSG)
         (* IOW.MESS6(SITE,LPROCID,RESRCES[RELPTR].LRNAME) *)
       END;
   IF THRUPUT=1 THEN THRUAFTER:=TIME-THRUBEFORE
                 ELSE THRUAFTER:=99900;
   MYMSG.MSGTYPE:=ATERMINATE;
   MYMSG.MSGID:=SITE;
   BUFF.BUFPUT(MYMSG);
   IOW.MESS7(SITE,LPROCID,TOTSENT,THRUAFTER);
   IOW.MESS14(SITE,TOTLOC,0)
END;  (*  PROCESS PPROCSS  *)

ENTRY PROCEDURE STTMACH(WHO:RSITE:INLINE,OUTLINE:LINE;MAXR,MAXP,
      TOTMAXP:INTEGER;LAMDA,MUU:REAL;MAXREQ,WACC,THRUP :INTEGER);
   VAR
       I,K,P  : INTEGER;
   BEGIN
    FOR I:=1 TO MAXP DO
       PPROCSS(WHO,I,MAXR,LAMDA,MUU,MAXREQ,WACC,THRUP);
    READLINE(WHO,INLINE,TOTMAXP);
    WRITER(OUTLINE,TOTMAXP);
   END;   (*    STTMACH   *)

END;  (*     *******  PMACHINE   *******     *)

(*  ***    SYSTEM   ACTIVATION    ***     *)
```

```
VAR
  NET:ARRAY[RSITE] OF PMACHINE;
  CONTR:CONTROLLER;
  LINES:ARRAY[SITE] OF LINE;
  MAXR,MAXP,TOTMAXP,WACC   : INTEGER;
  J,Y,THRUP : INTEGER;
  DISTPR:ARRAY[RSITE] OF INTEGER;
  LAMDA,MUU:REAL;
  MAXREQ,I : INTEGER;
BEGIN
  READ(TOTMAXP,MAXR,LAMDA,MUU,MAXREQ,WACC,THRUP);
  J:=0;
  Y:=TOTMAXP DIV RSITES;
  FOR I:=1 TO RSITES DO
    BEGIN
      DISTPR[I]:=Y;
      J:=J+Y
    END;
  I:=0;
  WHILE J<TOTMAXP DO
   BEGIN
    I:=I+1;
    DISTPR[I]:=DISTPR[I]+1;
    J:=J+1
   END;
  WRITELN(' CENTRALI','ZED HAV');
  WRITELN(' NO OF ','RESOURCES',' = ',MAXR);
  WRITELN(' NO OF ','PROCESSES',' = ',TOTMAXP);
  WRITELN(' MUU = ',MUU);
  WRITELN(' LAMDA = ',LAMDA);
  WRITELN(' MAXIMUM ','REQUEST =',MAXREQ);
  NET[1].STTMACH(1,LINES[4],LINES[1],MAXR,DISTPR[1],TOTMAXP,LAMDA,MUU,
                 MAXREQ,WACC,THRUP);
  NET[2].STTMACH(2,LINES[1],LINES[2],MAXR,DISTPR[2],TOTMAXP,LAMDA,MUU,
          MAXREQ,WACC,THRUP);
  NET[3].STTMACH(3,LINES[2],LINES[3],MAXR,DISTPR[3],TOTMAXP,LAMDA,MUU,
           MAXREQ,WACC,THRUP);
  CONTR.STTCONTR(LINES[3],LINES[4],TOTMAXP);
END.
```

APPENDIX E

Program Listing for Distributed Implementation of Prevention Technique Using Preemption on a 3-Site Network

```
PROGRAM PREV(INPUT,OUTPUT);

  (*************************************************)
  (*                                               *)
  (*    DEADLOCK PREVENTION TECHNIQUE USING PREEMPTION *)
  (*                                               *)
  (*************************************************)

  CONST
    NSITES=3;          (* 3 SITE NETWORK  *)
    BMAX=10;           (* BUFFER SIZE      *)
    NMAX=10;           (* MAXIMUM # PROCESSES *)
    MMAX=2;            (* MAXIMUM # RESOURCES A EACH SITE *)
    LINES=3;
  TYPE
    MESSTYPE=(AREQUEST,ARESPONSE,COMPLETION,ROLLBACK,LOCALL,
              ATERMINATE);
    SITES=1..NSITES;
    STATUS=(FREE,EXCLUSIVE,SHARED);
    NLINES=1..LINES;
    MESSAGE=RECORD
      MSGTYPE:MESSTYPE;
      MSGORIGIN:INTEGER;
      MSGDEST:INTEGER;
      PROCNAME:INTEGER;
      RESNAME:INTEGER;
      ACESTYPE:STATUS;
      QUESIZE : INTEGER;
    END;

    PROCIO=OBJECT
      PATH 1:( 1:(MESS1),1:(MESS2), 1:(MESS3),1:(MESS4),
             1:(MESS5),1:(MESS6),1:(MESS7),1:(MESS8),1:(MESS9),
        1:(MESS10),1:(MESS11),1:(MESS12),1:(MESS13),1:(MESS14),1:(MESS15) ) END;

      ENTRY PROCEDURE MESS1(I,J:INTEGER);
        VAR K:INTEGER;
        BEGIN
          K:=(J*100+I)*100;
          WRITELN(K)
        END;                         (*  MESS1  *)

      ENTRY PROCEDURE MESS2(I,J:INTEGER);
        VAR K:INTEGER;
        BEGIN
          K:=(J*100+I)*100+1;
          WRITELN(K)
        END;                         (*  MESS2  *)

      ENTRY PROCEDURE MESS3(I,J,K:INTEGER;L:STATUS);
        VAR T:INTEGER;
```

```
  BEGIN
    T:=I*1000000+J*10000+K*100;
    IF L=EXCLUSIVE THEN T:=T+2 ELSE T:=T+3;
    WRITELN(T)
  END;                                    (*  MESS3  *)

ENTRY PROCEDURE MESS4(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*1000000+J*10000+K*100+4;
    WRITELN(T)
  END;                                    (*  MESS4  *)

ENTRY PROCEDURE MESS5(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*1000000+J*10000+K*100+5;
    WRITELN(T)
  END;                                    (*  MESS5  *)

ENTRY PROCEDURE MESS6(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*1000000+J*10000+K*100+10;
    WRITELN(T)
  END;                                    (*  MESS6  *)

ENTRY PROCEDURE MESS7(I,J,K,L :INTEGER);
  VAR T,T1,T2,T3:INTEGER;
  BEGIN
    T:=I*1000+J*100+11;
    T1:=K*100000+I*1000+J*100+12;
    T3:=L DIV 100;
    T2:=T3*10000+I*100+J*10+8;
    WRITELN(T,T1,T2)
  END;                                    (*  MESS7  *)

ENTRY PROCEDURE MESS8(I,J,K:INTEGER);
  VAR T,T1 : INTEGER;
  BEGIN
    T:=J*10000+I*100+13;
    T1:=K*10000+I*100+14;
    WRITELN(T,T1)
  END;                                    (*  MESS8  *)

ENTRY PROCEDURE MESS9(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*1000000+J*10000+K*100+15;
    WRITELN(T)
  END;                                    (*  MESS9  *)

ENTRY PROCEDURE MESS10(I,J,K:INTEGER);
```

```
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+20;
    WRITELN(T)
  END;                                   (*  MESS10  *)

ENTRY PROCEDURE MESS11(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+21;
    WRITELN(T)
  END;                                              (*  MESS11  *)

ENTRY PROCEDURE MESS12(I,J,K:INTEGER);
  VAR T:INTEGER;
  BEGIN
    T:=I*100000+J*10000+K*100+22;
    WRITELN(T)
  END;                                              (* MESS12  *)

  ENTRY PROCEDURE MESS13(I,J,OU,QS :INTEGER);
      VAR T,T1,T3 :INTEGER;
      BEGIN
        OU:=OU DIV 100;
        T:=I*100+J*10;
        T1:=OU*10000+T+7;
        T3:=QS*10000+T+9;
        WRITELN(T1,T3)
   END;                                               (* MESS13  *)

ENTRY PROCEDURE MESS14(I,J :INTEGER);
      VAR T: INTEGER;
      BEGIN
        T:=J*10000+I*100+23;
        WRITELN(T)
      END;                                          (*  MESS14  *)

ENTRY PROCEDURE MESS15(I,RES,ROL,COM,ARE,K1,K2,K3:INTEGER);
      VAR T2,T3,T4,T5,T6,T7,T8,T9,T1000,T100 : INTEGER;
      BEGIN
       T1000:=10000;
       T100:=I*100;
       T3:=RES*T1000+T100+32;
       T4:=ROL*T1000+T100+33;
       T5:=COM*T1000+T100+34;
       T6:=ARE*T1000+T100+35;
       IF K1<>99 THEN BEGIN
                        T7:=K1*T1000+T100+36;
                        T8:=K2*T1000+T100+24;
                        T9:=K3*T1000+T100+25;
                        WRITE(T7,T8,T9)
                      END;
       WRITELN(T3,T4,T5,T6)
```

```
   END;                                              (*  MESS15  *)

  END;         (*  ******  PROCIO  ******  *)

  LINE=OBJECT

    PATH 1:(TOLINE:FRLINE) END;
    VAR MESGBUF:MESSAGE;

    ENTRY PROCEDURE TOLINE(M:MESSAGE);
      BEGIN
        MESGBUF:=M
      END;       (*  TOLINE  *)

    ENTRY PROCEDURE FRLINE(VAR M:MESSAGE);
      BEGIN
        M:=MESGBUF
      END;       (*  FRLINE  *)

  END;      (*  ******  LINE  ******  *)

  MACHINE=OBJECT
    PATH STARTMACH END;

    TYPE
     MSGQUEUE=OBJECT   (* INPUT MSGES TO BE PROCESSED  *)
       PATH BMAX :(1:(QUEPUT);1:(QUEGET)) END;
       VAR QUEBUFFER:ARRAY[1..BMAX] OF MESSAGE;
           INQQ,OUTQQ:1..BMAX;

      ENTRY PROCEDURE QUEPUT(M:MESSAGE);
        BEGIN
          QUEBUFFER[INQQ]:=M;
          INQQ:=(INQQ MOD BMAX)+1
        END;   (*   QUEPUT   *)

      ENTRY PROCEDURE QUEGET(VAR M:MESSAGE;VAR QS : INTEGER);
        BEGIN
         M:=QUEBUFFER[OUTQQ];
         IF OUTQQ>INQQ THEN QS:=(BMAX-OUTQQ)+INQQ
                       ELSE QS:=INQQ-OUTQQ;
         OUTQQ:=(OUTQQ MOD BMAX) + 1;
        END;   (*   QUEPUT   *)

      INIT: BEGIN
             INQQ:=1;
             OUTQQ:=1
            END;        (*  INIT  *)

     END;    (*  ******  MSGQUEUE  ******  *)
```

```
OUTQUEUE=OBJECT      (*  MSGES TO BE SENT OUT   *)
  PATH BMAX:(1:(OUTPUTT):1:(OUTGET)) END;
  VAR OUTBUFFER:ARRAY[1..BMAX] OF MESSAGE;
      OUTP,OUTG:1..BMAX;

  ENTRY PROCEDURE OUTPUTT(M:MESSAGE);
     BEGIN
       OUTBUFFER[OUTP]:=M;
       OUTP:=(OUTP MOD BMAX) + 1
     END;     (*  OUTPUTT   *)

  ENTRY PROCEDURE OUTGET(VAR M:MESSAGE);
     BEGIN
       M:=OUTBUFFER[OUTG];
       OUTG:=(OUTG MOD BMAX )+1
     END;      (*  OUTGET    *)

  INIT: BEGIN
          OUTP:=1;
          OUTG:=1
        END;  (*  INIT   *)

END;    (*  ******  OUTQUEUE  ******  *)

PRBUF=OBJECT    (* PRIVATE BUFFER FOR EACH PROCESS*)
  PATH 1:(PRBUFPUT;PRBUFGET) END;
  VAR  PRBUFFER:MESSAGE;

  ENTRY PROCEDURE PRBUFPUT(M:MESSAGE);
    BEGIN
      PRBUFFER:=M
    END;   (* PRBUFFER   *)

  ENTRY PROCEDURE PRBUFGET(VAR M:MESSAGE);
    BEGIN
      M:=PRBUFFER
    END;  (*  PRBUFGET   *)

END;  (*  ******  PRBUF  ******   *)

RESRC=RECORD
  RNAME:INTEGER;
  RSTATUS:STATUS;
  NACCES : INTEGER
END;
VAR
   MQUEUE:MSGQUEUE;
   OQUEUE:OUTQUEUE;
   PBUF:ARRAY[1..5] OF PRBUF;
   IO:PROCIO;
   LRESTAB:ARRAY[0..MMAX] OF RESRC;
   N,M,PP,RR: INTEGER;
   REQACCESS:STATUS;
```

```
MSGTEMP:MESSAGE;
TENTRY:(REQ,REL);
TOTREQ,IFR,JFP,TOTDEAD :INTEGER;
MYSITE:SITES;
STK:ARRAY[1..10] OF INTEGER;

FUNCTION FINDR(R:INTEGER):INTEGER;
 (* RETURNS AN INDEX TO A RESOURCE IN RESOURCETABLE *)
VAR I:INTEGER;
BEGIN
 I:=0;
 WHILE LRESTAB[I].RNAME<>R DO I:=I+1;
 FINDR:=I;
END;  (*  FINDR  *)

PROCEDURE RESREL;
   VAR I : INTEGER;
   BEGIN
     I:=FINDR(RR);
     LRESTAB[I].NACCES:=LRESTAB[I].NACCES-1;
     IF LRESTAB[I].NACCES=0 THEN
       LRESTAB[I].RSTATUS:=FREE
  END;

 PROCEDURE SENRESPONSE;
   (* SENDS OUT RESPONSE TO REQUESTING PROCESSES   *)
  BEGIN
    IFR:=FINDR(RR);
    LRESTAB[IFR].RSTATUS:=REQACCESS;
    LRESTAB[IFR].NACCES:=LRESTAB[IFR].NACCES+1;
    WITH MSGTEMP DO
        BEGIN
          MSGTYPE:=ARESPONSE;
          MSGDEST:=MSGORIGIN;
          MSGORIGIN:=MYSITE
        END;
    IF MSGTEMP.MSGDEST=MYSITE THEN
    PBUF[MSGTEMP.PROCNAME].PRBUFPUT(MSGTEMP)  ELSE
    OQUEUE.OUTPUTT(MSGTEMP)
    (* IO.MESS3(MSGTEMP.MSGDEST,MSGTEMP.PROCNAME,RR) *)
  END;

 PROCEDURE SENDROLLB;
     (*  SENDS ROLLBACK MESSAGE  *)
   BEGIN
     TOTDEAD:=TOTDEAD+1;
     WITH MSGTEMP DO
        BEGIN
          MSGTYPE:=ROLLBACK;
          MSGDEST:=MSGORIGIN;
          MSGORIGIN:=MYSITE
        END;
     IF MSGTEMP.MSGDEST=MYSITE THEN
```

```
            PBUF[MSGTEMP.PROCNAME].PRBUFPUT(MSGTEMP) ELSE
          OQUEUE.OUTPUTT(MSGTEMP)
       END;   (*   SEND ROLLBACK   *)

     PROCEDURE RESREQ;
          (* PROCESS PP REQUEST FOR RESOURCE RR   *)
       VAR I,J:INTEGER;
           RFREE:BOOLEAN;
           THELD:STATUS;
        BEGIN
         TOTREQ:=TOTREQ+1;
         IFR:=FINDR(RR);
         RFREE:=FALSE; THELD:=LRESTAB[IFR].RSTATUS;
         IF THELD=FREE THEN RFREE:=TRUE ELSE
         IF (THELD=SHARED) AND (REQACCESS=SHARED) THEN
           RFREE:=TRUE;
         IF RFREE THEN SENRESPONSE ELSE SENDROLLB;
     END;   (*  RESREQ  *)

     PROCEDURE MANAGER;
        BEGIN
       IF TENTRY=REQ THEN RESREQ ELSE RESREL
     END;  (*  MANAGER  *)

(**********************************************)
(*                                            *)
(*        RANDOM NUMBER GENERATORS            *)
(*                                            *)
(**********************************************)

     FUNCTION RAND(VAR SEED:REAL; MOOP:INTEGER):INTEGER;
        CONST
          P=2147483647;
          A=16807;
        VAR ISEED:INTEGER;
        BEGIN
          ISEED:=TRUNC(SEED);
          SEED:=(A*ISEED) MOD P;
          ISEED:=TRUNC(SEED) MOD MOOP;
          RAND:=ISEED
        END;  (*  RAND  *)

     FUNCTION RANDOM(VAR S:REAL):REAL;
        VAR ISEED:INTEGER;
        BEGIN
         ISEED:=TRUNC(S);
         ISEED:=(ISEED*899) MOD 32767;
         S:=ISEED;
         RANDOM:=S/32767.0
        END;  (*  RANDOM  *)
```

```
(**************************************************************)
(*                                                            *)
(*   E N D    OF    R O U T I N E S                           *)
(*                                                            *)
(************************************************************)

PROCESS WRITER(OUTLINE:LINE;SITE,MAXP:INTEGER);
  (*  WRITE MSG TO OUTPUT LINE   *)
  VAR M:MESSAGE;
      WRITING:BOOLEAN;
      TOTL,TOTMSGSENT,TOTMAXP :INTEGER;
  BEGIN
   WRITING:=TRUE;
   TOTMSGSENT:=0;
   TOTL:=0;  TOTMAXP:=MAXP;
   WHILE WRITING DO
     BEGIN
       OQUEUE.OUTGET(M);
       OUTLINE.TOLINE(M);
       TOTMSGSENT:=TOTMSGSENT+1;
       IF M.MSGTYPE=ATERMINATE THEN TOTL:=TOTL+1;
       IF TOTL=TOTMAXP THEN WRITING:=FALSE
     END;
  IO.MESS1(SITE,TOTMSGSENT);
 END;   (*  WRITER  *)

 PROCESS READER(INLINE:LINE;MAXP:INTEGER);
  (* MONITOR INPUT LINE FOR ALL INCOMING MESSAGES; IF MSG IS FOR A LOCAL
     PROCESS IT WAKES UP THE PROCESS TO ACCEPT THE RESPONSE; NOTE THAT
    THE KERNEL CAN ALSO WAKE UP A LOCAL PROCESS IF THE REQUEST MADE
    WAS FOR A LOCAL RESOURCE; IF THE MSG IS FOR A RESOURCE REQUEST
    CHECKS IF THE REQUESTED RESOURCE IS LOCAL; IF LOCAL PUTS THE MSG
    IN MSGQUEUE FOR THE KERNEL TO PROCESS; IF NOT IT PUTS IT IN OUTBUFFER TO
    BE PASSED ON;  IF THE MSG IS A DETECTION MSG OR RESOURCE RELEASE
    FOR A LOCAL RESOURCE IT PUTS IT IN MSGQUEUE   *)

  VAR MESG:MESSAGE;
      I,RTOTL,TOTMSGRECVD:INTEGER;
      SITE,TOTRESP,TOTROLLB,TOTCOMPL,TOTAREQ: INTEGER;
      SW,READING:BOOLEAN;
   BEGIN
     READING:=TRUE;
     RTOTL:=0;
      TOTRESP:=0;  TOTROLLB:=0;  TOTCOMPL:=0;
     TOTAREQ:=0;   SITE:=MYSITE;
     TOTMSGRECVD:=0;
     WHILE READING DO
       BEGIN
         INLINE.FRLINE(MESG);
         TOTMSGRECVD:=TOTMSGRECVD+1;
         CASE MESG.MSGTYPE OF
```

```
        AREQUEST:
          BEGIN
            TOTAREQ:=TOTAREQ+1;
            SW:=FALSE;
            FOR I:=0 TO M DO
             IF LRESTAB[I].RNAME=MESG.RESNAME THEN SW:=TRUE;
            IF SW THEN MQUEUE.QUEPUT(MESG) ELSE
                          OQUEUE.OUTPUTT(MESG)
          END;

        ARESPONSE,ROLLBACK:
          BEGIN
            CASE MESG.MSGTYPE OF
              ARESPONSE: TOTRESP:=TOTRESP+1;
              ROLLBACK : TOTROLLB:=TOTROLLB+1
            END;

          IF MESG.MSGDEST=SITE THEN
                PBUF[MESG.PROCNAME].PRBUFPUT(MESG)
                                ELSE OQUEUE.OUTPUTT(MESG)
        END;

        COMPLETION:
          BEGIN
           TOTCOMPL:=TOTCOMPL+1;
           IF MESG.MSGDEST=SITE THEN MQUEUE.QUEPUT(MESG)
                                ELSE OQUEUE.OUTPUTT(MESG)
          END;

          ATERMINATE:
            BEGIN
              RTOTL:=RTOTL+1;
              IF MESG.MSGORIGIN<>SITE THEN MQUEUE.QUEPUT(MESG);
              IF RTOTL=MAXP THEN
                 READING:=FALSE
            END;
      END; (* CASE *)
    END;  (*  WHILE READING *)
    IC.MESS2(SITE,TOTMSGRECVD);
    IC.MESS15(SITE,TOTRESP,TOTROLLB,TOTCOMPL,
              TOTAREQ,99,99,99);
END;  (*  READER  *)

PROCESS KERNEL(SITE:SITES;MAXR,MAXP : INTEGER);
   (* KERNEL HANDLES THE RESOURCE ALLOCATION AT EACH SITE;
      IT RUNS THE DETECTION ALGORITHM  *)
  VAR KTOTL,I,TOTMAXP,TOTLOC,QSIZE : INTEGER;
     KERNELLING,SW :BOOLEAN;
  BEGIN
    KERNELLING:=TRUE;
    TOTMAXP:=MAXP; KTOTL:=0;  TOTLOC:=0;
    (* THE KERNEL DOES NOT PROCESS ANY OTHER RESOURCE REQUEST UNTIL
```

```
   IT HAS RESOLVED ANY OUTSTANDING REQUEST *)
  WHILE KERNELLING DO
    BEGIN
      MQUEUE.QUEGET(MSGTEMP,QSIZE);
      CASE MSGTEMP.MSGTYPE OF
       ATERMINATE:
         BEGIN
          KTOTL:=KTOTL+1;
          OQUEUE.OUTPUTT(MSGTEMP)
         END;

       LOCALL:
         BEGIN
            MSGTEMP.QUESIZE:=QSIZE;
            MSGTEMP.MSGTYPE:=AREQUEST;
            SW:=FALSE;
            FOR I:=0 TO M DO
               IF MSGTEMP.RESNAME=LRESTAB[I].RNAME THEN
                           SW:=TRUE;
            IF NOT SW THEN
               OQUEUE.OUTPUTT(MSGTEMP) ELSE
                 BEGIN
                  WITH MSGTEMP DO
                    BEGIN
                     PP:=MSGORIGIN*1000+PROCNAME;
                     RR:=RESNAME;
                     REQACCESS:=ACESTYPE
                    END;
                  TENTRY:=REQ;
                  TOTLOC:=TOTLOC+1;
                  MANAGER
                 END
          END;

       AREQUEST,COMPLETION:
         BEGIN
          WITH MSGTEMP DO
           BEGIN
            QUESIZE:=QUESIZE+QSIZE;
            PP:=MSGORIGIN*1000+PROCNAME;
            RR:=RESNAME;
            REQACCESS:=ACESTYPE
            ;IF MSGTYPE=AREQUEST THEN TENTRY:=REQ ELSE TENTRY:=REL
          END;
          MANAGER
         END;
      END; (* CASE *)
      IF KTOTL=TOTMAXP THEN KERNELLING:=FALSE;
     END; (* WHILE *)
     IO.MESS8(SITE,TOTDEAD,TOTREQ);
     IO.MESS14(SITE,TOTLOC)
  END; (* KERNEL *)
```

```
PROCESS PPROCSS(SITE,TOTMAXR,PROCNO:INTEGER;LAMDA,MUU:REAL;
                MAXREQ,WACCES,THRUPUT : INTEGER);
  (* SIMULATE A LOCAL PROCESS ACTIVITIES *)
  LABEL 1,2;
  TYPE
    LRES=RECORD
     LRNAME:INTEGER;
     TACCESS:STATUS;
     LOCATION:INTEGER
    END;
  VAR
   RESRCES:ARRAY[1..15] OF LRES;
   CLOCK,TRELEASE,TREQUEST,LAMDABAR,MUUBAR,SEEDR,SEED:REAL;
   TEMP,T2,TBEFORE,TOTSECS,XXX:REAL;
   NUMRES,RR,MP,I,J,TOTSENT,TOTDELAY,RELPTR,REQPTR :INTEGER;
   LPROCID,OUTREQ,THRUBEFORE,THRUAFTER : INTEGER;
   TESTCASE,TS,TD,MPPP : INTEGER;
   MAINSW,SW,SW1,GREATR,PROCESING,AGAN :BOOLEAN;
   MYMSG:MESSAGE;
   ACCTYPE:STATUS;

  PROCEDURE GENREQ;
      BEGIN (* GENERATE NEW RESOURCE  *)
       SW:=FALSE;
       WHILE NOT SW DO
         BEGIN
          RR:=RAND(SEED,TOTMAXR)+1;
          IF (REQPTR=0) OR (OUTREQ=0) THEN
            SW:=TRUE ELSE
            BEGIN
             SW1:=FALSE;
             J:=(RELPTR MOD 15)+1;
             FOR I:=1 TO OUTREQ DO
              BEGIN
               IF RESRCES[J].LRNAME=RR THEN SW1:=TRUE;
               J:=(J MOD 15)+1
              END;
             IF NOT SW1 THEN SW:=TRUE
            END
          END;
         (* TYPE OF ACCESS  *)
         IF WACCES=1 THEN ACCTYPE:=EXCLUSIVE ELSE
            BEGIN
             TEMP:=RANDOM(SEEDR);
             IF TEMP>=0.5 THEN ACCTYPE:=EXCLUSIVE ELSE
                                ACCTYPE:=SHARED
            END;
         REQPTR:=(REQPTR MOD 15)+1;  OUTREQ:=OUTREQ+1;
         RESRCES[REQPTR].LRNAME:=RR;
         RESRCES[REQPTR].TACCESS:=ACCTYPE;
      (* SEND REQUEST  *)
      WITH MYMSG DO
```

```
BEGIN
 MSGORIGIN:=SITE; PROCNAME:=LPROCID;
 MSGTYPE:=LOCALL; RESNAME:=RR;  QUESIZE:=0;
 ACESTYPE:=ACCTYPE
END;
IO.MESS3(SITE,LPROCID,RR,ACCTYPE);
MQUEUE.QUEPUT(MYMSG);
J:=TIME; TOTSENT:=TOTSENT+1;
(* TBEFORE:=SIN(XXX);  *)
PBUF[LPROCID].PRBUFGET(MYMSG);
(* PROCESS BLOCKED WAITING FOR RESPONSE  *)
(*  TEMP:=SIN(XXX)-TBEFORE;  *)
 (* TS:=TRUNC(TEMP);  *)
 TD:=TIME-J;  MPPP:=MYMSG.QUESIZE;
IO.MESS13(SITE,LPROCID,TD,MPPP);
IF MYMSG.MSGTYPE=ROLLBACK THEN
  BEGIN
   (* IO.MESS4(SITE,LPROCID,MYMSG.RESNAME);  *)
   RESRCES[REQPTR].LOCATION:=MYMSG.MSGORIGIN;
   REQPTR:=REQPTR-1;
   IF (REQPTR=0) OR (REQPTR=-1) THEN REQPTR:=15;
   OUTREQ:=OUTREQ-1;
   MP:=MYMSG.MSGORIGIN; AGAN:=TRUE
  END ELSE
BEGIN
   (* IO.MESS5(SITE,LPROCID,MYMSG.RESNAME);  *)
   RESRCES[REQPTR].LOCATION:=MYMSG.MSGORIGIN;
 END;
END;  (* GENREQ  *)
PROCEDURE ASSREL;
 BEGIN
  RELPTR:=(RELPTR MOD 15)+1;  OUTREQ:=OUTREQ-1;
   WITH MYMSG DO
    BEGIN
     PROCNAME:=LPROCID;  MSGTYPE:=COMPLETION;
     MSGORIGIN:=SITE;
     MSGDEST:=RESRCES[RELPTR].LOCATION;
     RESNAME:=RESRCES[RELPTR].LRNAME; ACESTYPE:=FREE
     END
END;  (* RELPTR  *)


BEGIN
 LPROCID:=PROCNO;
 TOTSENT:=0; TOTDELAY:=0; XXX:=5.0;   PROCESING:=TRUE;
 TOTSECS:=0.0;  CLOCK:=0.0;
 SEEDR:=31415.0/SITE;  SEED:=SITE;  TRELEASE:=0.0; TREQUEST:=0.0;
 THRUBEFORE:=TIME;
 (*  RELPTR POINTS TO THE LAST RESOURCE RELEASED
     REQPTR POINTS TO THE LAST RESOURCE REQUESTED FOR  *)
1: RELPTR:=0;  REQPTR:=0;  GREATR:=FALSE;
 OUTREQ:=0;  AGAN:=FALSE;
  MAINSW:=FALSE;
```

```
WHILE PROCESING DO
   BEGIN
    MP:=-1;
    GENREQ;
    IF MP<>-1 THEN GOTO 2;
    (* GENERATE TIME OF NEXT RELEASE  *)
    MUUBAR:=(-1.0/MUU)*LN(RANDOM(SEEDR));
    TRELEASE:=CLOCK+MUUBAR;
    (* GENERATE TIME OF NEXT REQUEST *)
    LAMDABAR:=(-1.0/LAMDA)*LN(RANDOM(SEEDR));
    TREQUEST:=CLOCK+LAMDABAR;
    MAINSW:=TRUE;
    WHILE MAINSW DO
     BEGIN
      IF TRELEASE>TREQUEST THEN TESTCASE:=1;
      IF TRELEASE=TREQUEST THEN TESTCASE:=2;
      IF TRELEASE<TREQUEST THEN TESTCASE:=3;
      CASE TESTCASE OF
        1:  (* TRELEASE>TREQUEST  *)
         BEGIN
            TEMP:=LAMDABAR*100.0;
            I:=TRUNC(TEMP);  T2:=I+0.49;
            IF TEMP>T2 THEN I:=I+1;
            DELAY(I);  CLOCK:=TREQUEST;
            IF (TOTSENT>=MAXREQ) THEN
              BEGIN
               AGAN:=FALSE; MP:=-1; PROCESING:=FALSE; GOTO 2
              END;
            MUUBAR:=TRELEASE-TREQUEST;
              (* GENERATE REQUEST  *)
            IF OUTREQ>=TOTMAXR THEN
             BEGIN
              (* REQUEST BUT RES HELD EQUALS MAX RES *)
               TRELEASE:=TREQUEST; ASSREL;
               IF MYMSG.MSGDEST=SITE THEN
               MQUEUE.QUEPUT(MYMSG)   ELSE
               OQUEUE.OUTPUTT(MYMSG);
               (* IO.MESS6(SITE,LPROCID,RESRCES[RELPTR].LRNAME); *)
               MAINSW:=FALSE
            END ELSE
            BEGIN
             MP:=-1;  GENREQ;
             IF MP<>-1 THEN GOTO 2;
             (* GENERATE TIME OF NEXT REQUEST  *)
             LAMDABAR:=(-1.0/LAMDA)*LN(RANDOM(SEEDR));
             TREQUEST:=CLOCK+LAMDABAR;  MAINSW:=TRUE
         END
      END;  (* TESTCASE=1  *)
      2: (* TRELEASE=TREQUEST  *)
        BEGIN
         CLOCK:=TRELEASE;
         TEMP:=LAMDABAR*100.0;
         I:=TRUNC(TEMP); T2:=I+0.49;
```

```
            IF TEMP>T2 THEN I:=I+1; DELAY(I);
            (* RELEASE RESOURCE IF ANY  *)
            IF OUTREQ>0 THEN
              BEGIN
               ASSREL;
                IF MYMSG.MSGDEST=SITE THEN MQUEUE.QUEPUT(MYMSG) ELSE
                     OQUEUE.OUTPUTT(MYMSG)
           (* IO.MESS6(SITE,LPROCID,RESRCES(RELPTR).LRNAME)   *)
          END;
          MAINSW:=FALSE;
          IF (TOTSENT>=MAXREQ) THEN
            BEGIN
             AGAN:=FALSE; MP:=-1; PROCESING:=FALSE; GOTO 2
            END;
        END;
    3:  (* TRELEASE<TREQUEST  *)
        BEGIN
         TEMP:=MUUBAR*100.0;
         I:=TRUNC(TEMP); T2:=I+0.49;
         IF TEMP>T2 THEN I:=I+1;    DELAY(I);
         IF OUTREQ<=0 THEN
           BEGIN (* NO RES TO RELEASE  *)
            CLOCK:=TREQUEST;
            TEMP:=(TREQUEST-TRELEASE)*100.0;
            I:=TRUNC(TEMP);  T2:=I+0.49;
            IF TEMP>T2 THEN I:=I+1;   DELAY(I);  MAINSW:=FALSE
            END  ELSE
            BEGIN  (* RELEASE  RESOURCE  *)
             CLOCK:=TRELEASE:  ASSREL;
             IF MYMSG.MSGDEST=SITE THEN MQUEUE.QUEPUT(MYMSG) ELSE
                OQUEUE.OUTPUTT(MYMSG);
             (* IO.MESS6(SITE,LPROCID,RESRCES(RELPTR).LRNAME); *)
             LAMDABAR:=TREQUEST-TRELEASE;
             (* GENERATE TIMOF NEXT RELEASE *)
              MUUBAR:=(-1.0/MUU)*LN(RANDOM(SEEDR));
              TRELEASE:=CLOCK+MUUBAR;
              IF (TOTSENT>=MAXREQ) THEN
                BEGIN
                  AGAN:=FALSE;  MP:=-1;  PROCESING:=FALSE;
                  GOTO 2
             END
      END
     END;  (* TRELEASE < TREQUEST  *)
    END;  (*  CASE  *)
    END: (*  MAINSW  *)
  END;  (* PROCESING  *)
    2:  IF OUTREQ>0 THEN
         WHILE OUTREQ>0 DO
           BEGIN
            ASSREL;
            IF MYMSG.MSGDEST<>MP THEN
              BEGIN
               IF MYMSG.MSGDEST=SITE THEN MQUEUE.QUEPUT(MYMSG)
```

```
                    ELSE OQUEUE.OUTPUTT(MYMSG)
                    (* IO.MESS6(SITE,LPROCID,MYMSG.RESNAME) *)
                    END;
              END;
         IF AGAN THEN
            BEGIN
              I:=1000;
              J:=RAND(SEED,I)+100;
              DELAY(J);
              IF THRUPUT=1 THEN TOTSENT:=0;
              GOTO 1
            END;
         IF THRUPUT=1 THEN THRUAFTER:=TIME-THRUBEFORE
                       ELSE THRUAFTER:=99900;
         MYMSG.MSGTYPE:=ATERMINATE;
         MYMSG.MSGDEST:=SITE;
         MQUEUE.QUEPUT(MYMSG);
       IO.MESS7(SITE,LPROCID,TOTSENT,THRUAFTER);
    END;  (* PROCESS PPROCSS *)

    ENTRY PROCEDURE STARTMACH(SITE:SITES;INLINE,OUTLINE:LINE;MAXR,SRES,
      STARTR,MAXP,PROCS:INTEGER;LAMDA,MUU:REAL;MAXREQ,WACC,THRUP:INTEGER);
     VAR I,J:INTEGER;
     BEGIN
      N:=-1;  M:=SRES-1;  TOTREQ:=0; TOTDEAD:=0; MYSITE:=SITE;
      (* INITIALISE RESOURCES FOR THIS SITE *)
      J:=STARTR;
      FOR I:=0 TO SRES-1 DO
       BEGIN
        WITH LRESTAB[I] DO
          BEGIN
           RNAME:=J; RSTATUS:=FREE; NACCES:=0
          END;
        J:=J+1
       END;
      (* START PROCESSES AT THIS SITE  *)
       FOR I:=1 TO PROCS DO
       PPROCSS(SITE,MAXR,I,LAMDA,MUU,MAXREQ,WACC,THRUP);
       KERNEL(SITE,SRES,MAXP);
       READER(INLINE,MAXP);
       WRITER(OUTLINE,SITE,MAXP);
     END;  (* STARTMACH  *)

  END;   (* ******  MACHINE  ****** *)

(******************************************************************)
(*                                                                *)
(*  S Y S T E M   A C T I V A T I O N                             *)
(*                                                                *)
(******************************************************************)

VAR
  NET:ARRAY[SITES] OF MACHINE;
```

```
LINK:ARRAY[NLINES] OF LINE;
MAXREQ,MAXP,MAXR,WACC,THRUP : INTEGER;
RPERSITE:ARRAY[SITES] OF INTEGER;
PPERSITE:ARRAY[SITES] OF INTEGER;
I,K,L,J,Y : INTEGER;
  (* WACC=0 IF SHARED ^ EXCL AND 1 IF EXCL ONLY  *)
  (* THRUP=0 PERFORMANCE MEASURE. =1 THRUPUT MEASURE. *)
 LAMDA,MUU:REAL;
 BEGIN
  READ(MAXP,MAXR,LAMDA,MUU,MAXREQ,WACC,THRUP);
   (* DISTRIBUTE RESOURCES AMONG SITES  *)
   K:=0;  L:=MAXR DIV NSITES;
   J:=0;  Y:=MAXP DIV NSITES;
   FOR I:=1 TO NSITES DO
    BEGIN
     RPERSITE[I]:=L;  K:=K+L;
     PPERSITE[I]:=Y;  J:=J+Y
    END;
   I:=0;
   WHILE K<MAXR DO
    BEGIN
     I:=I+1; RPERSITE[I]:=RPERSITE[I]+1;
     K:=K+1
    END;
    I:=0;
    WHILE J<MAXP DO
      BEGIN
      I:=I+1; PPERSITE[I]:=PPERSITE[I]+1; J:=J+1
      END;
    WRITELN(' DEADLOCK ','PREVENTION',' METHOD');
    WRITELN(' N = ',MAXP,' M = ',MAXR);
    WRITELN(' MUU = ',MUU,' LAMDA = ',LAMDA);
    WRITELN(' MAXIMUM ','REQUEST =',MAXREQ);
    NET[1].STARTMACH(1,LINK[3],LINK[1],MAXR,RPERSITE[1],1,
             MAXP,PPERSITE[1],LAMDA,MUU,MAXREQ,WACC,THRUP);
    I:=RPERSITE[1]+1;
    NET[2].STARTMACH(2,LINK[1],LINK[2],MAXR,RPERSITE[2],I,
             MAXP,PPERSITE[2],LAMDA,MUU,MAXREQ,WACC,THRUP);
    I:=I+RPERSITE[2];
    NET[3].STARTMACH(3,LINK[2],LINK[3],MAXR,RPERSITE[3],I,
             MAXP,PPERSITE[3],LAMDA,MUU,MAXREQ,WACC,THRUP);
END.
```

APPENDIX F

"PROCIO" Decoding

1. MESS1(i,j : integer)

   Total Message Units sent by Site i = j

2. MESS2(i,j : integer)

   Total Message Units received by Site i = j

3. MESS3(i,j,k : integer ; l = status)

   Process j at Site i Requests l access to Resource k

4. MESS4(i,j,k : integer)

   Process j at Site i on Resource k Rollsback

5. MESS5(i,j,k : integer)

   Process j at Site i Receives Resource k

6. MESS6(i,j,k : integer)

   Process j at Site i Releases Resource k

7. MESS7(i,j,k,l : integer)

   Process j at Site i terminates :
   Total Requests made by Process j at Site i = k
   Total Delay in Units of 100 of Process j at Site i = l

8. MESS8(i,j,k : integer)

   Total Deadlock detected by Site i = j
   Total Resource Requests Received by Site i = k

9. MESS9(i,j,k : integer)

   Process j at Site i granted access to Resource k

10. MESS10(i,j,k : integer)

    Process j at Site i Request for Resource k causes deadlock

11. MESS11(i,j,k : integer)

    Process j at Site i must wait for Resource k
    Resource is not free for immediate allocation. Deadlock detection algorithm had been initiated and there is no deadlock.

12. MESS12(i,j,k : integer)

    Release of Resource k by Process j at Site i acknowledged
    by owner of resource

13. MESS13(i,j,k,l : integer)

    Process j at Site i request delay in units of 100 = k
    Process j at Site i total quesize for request = l

14. MESS14(i,j : integer)

    Total number of Local Resource Requests by local Processes
    at Site i = j

15. MESS15(i,nf,de,res,rol,com,are,int,dl,nfre : integer)

    Total NOTFREE Message Units received by Site i = nf
    Total Detection Message Units received by Site i = de
    Total Request Granted Message Units received by Site i = res
    Total Rollback Message Units received by Site i = rol
    Total Release Message Units received by Site i = com
    Total Resource Request Message Units received by Site i = are
    Total INITDEAD Message Units received by Site i = int
    Total DLOCK Message Units received by Site i = dl
    Total NFREE Message Units received by Site i = nfre