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 Philosophy

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.
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

<table>
<thead>
<tr>
<th>Chapter</th>
<th>INTRODUCTION</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>1.1 Deadlock Elimination Techniques</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1.1.1 Deadlock Prevention</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1.1.2 Deadlock Detection</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>1.1.3 Deadlock Avoidance</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>1.1.4 Mixed Solution</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>1.2 Database and Deadlock Problem</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>1.2.1 Centralized Database</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>1.2.2 Distributed Database</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>1.3 Deadlock in Packet Switch Networks</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>1.4 Deadlock--A Game between Operating System and Processes</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>1.5 Definitions</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>1.6 Statement of the Thesis</td>
<td>15</td>
</tr>
<tr>
<td>II.</td>
<td>GOLDMAN'S DISTRIBUTED ALGORITHM</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>2.1 Example</td>
<td>18</td>
</tr>
<tr>
<td>III.</td>
<td>THE HORIZONTAL AND VERTICAL ALGORITHM</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3.1 Basic Assumptions</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3.2 Cycles in Process-Resource Graph</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3.3 Formal Model of the Horizontal and Vertical Deadlock Detection Scheme</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>3.4 Semantics of the Horizontal and Vertical Routines</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>3.4.1 Horizontal</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>3.4.2 Vertical</td>
<td>34</td>
</tr>
<tr>
<td>IV.</td>
<td>CENTRALIZED APPROACH TO ON-LINE DEADLOCK DETECTION USING THE HORIZONTAL AND VERTICAL ALGORITHM</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>4.1 Verification of the Centralized Horizontal and Vertical Algorithm</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>4.2 Example</td>
<td>41</td>
</tr>
<tr>
<td>V.</td>
<td>DECENTRALIZED APPROACH TO ON-LINE DEADLOCK DETECTION USING THE HORIZONTAL AND VERTICAL ALGORITHM</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>5.1 Verification of the Decentralized Horizontal and Vertical Algorithm</td>
<td>51</td>
</tr>
</tbody>
</table>
VI. SIMULATION STUDY OF THE HORIZONTAL AND VERTICAL
ALGORITHMS AND GOLDMAN'S ALGORITHM ON A
RING NETWORK

6.1 Experiment Definition
6.2 Results of the Simulation Study

6.2.1 Comparison of the Algorithms' Performance for Varying Numbers of
Processes on a Three-site Network
6.2.2 Comparison of the Algorithms' Performance for Varying Numbers of
Sites
6.2.3 Frequency of Deadlock for Varying
Loading Factor

VII. QUEUEING ANALYSIS OF THE DISTRIBUTED HORIZONTAL
AND VERTICAL ALGORITHM

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

VIII. SUMMARY AND CONCLUSION

IX. BIBLIOGRAPHY

APPENDICES

A : Description of the Simulation Programs
A.1 Distributed Control
A.1.1 Distributed Horizontal and Vertical Algorithm
A.1.2 Distributed Goldman's Algorithm
A.1.3 Preemption
A.2 Centralized Control

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

C : Program Listing for the Implementation of Goldman's Distributed Algorithm
on a 3-Site Network
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. . . . 211

E : Program Listing for Distributed Implementation of Prevention Technique using Preemption, on a 3-Site Network. . . . . . . . 237

F : "PROCIO" Decoding. . . . . . . . . . . . . . . . . 254
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>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}</td>
<td>19</td>
</tr>
<tr>
<td>2</td>
<td>Process-Resource Graph</td>
<td>29</td>
</tr>
<tr>
<td>3</td>
<td>H&amp;V Transformed Graph</td>
<td>38</td>
</tr>
<tr>
<td>4</td>
<td>Process-Resource Graph for Centralized H&amp;V Example</td>
<td>42</td>
</tr>
<tr>
<td>5</td>
<td>Deadlock cycle involving Processes P#, P1,P2,...,Pn</td>
<td>54</td>
</tr>
<tr>
<td>6</td>
<td>Global deadlock cycle involving Processes P1,P2,...,Pn</td>
<td>55</td>
</tr>
<tr>
<td>7</td>
<td>Process Average Response Time vs. Number of Processes for all 4 Algorithms on a 3-site Network</td>
<td>89</td>
</tr>
<tr>
<td>8</td>
<td>Process Average Throughput vs. Number of Processes for all 4 Algorithms on a 3-Site Network</td>
<td>90</td>
</tr>
<tr>
<td>9</td>
<td>Request Average Response Time vs. Number of Processes for all 4 Algorithms on a 3-site Network</td>
<td>91</td>
</tr>
<tr>
<td>10</td>
<td>Request Average Throughput vs. Number of Processes for all 4 Algorithms on a 3-site Network</td>
<td>92</td>
</tr>
<tr>
<td>11</td>
<td>Average Message Units per Request vs. Number of Processes for all 4 Algorithms on a 3-site Network</td>
<td>93</td>
</tr>
<tr>
<td>12</td>
<td>Frequency of Rollback vs. Number of Processes for all 4 Algorithms on a 3-site Network</td>
<td>94</td>
</tr>
<tr>
<td>13</td>
<td>Frequency of Detection Initiation vs. Number of Processes for Detection Algorithms only on a 3-site Network</td>
<td>95</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>14</td>
<td>Process Average Response Time vs. Number of Sites for all 4 Algorithms assuming 1 Process and 1 Resource per Site</td>
<td>99</td>
</tr>
<tr>
<td>15</td>
<td>Process Average Throughput vs. Number of Sites for all 4 Algorithms assuming 1 Process and 1 Resource per Site</td>
<td>100</td>
</tr>
<tr>
<td>16</td>
<td>Request Average Response Time vs. Number of Sites for all 4 Algorithms assuming 1 Process and 1 Resource per Site</td>
<td>101</td>
</tr>
<tr>
<td>17</td>
<td>Request Average Throughput vs. Number of Sites for all 4 Algorithms assuming 1 Process and 1 Resource per Site</td>
<td>102</td>
</tr>
<tr>
<td>18</td>
<td>Average Message Units per Request vs. Number of Sites for all 4 Algorithms assuming 1 Process and 1 Resource per Site</td>
<td>103</td>
</tr>
<tr>
<td>19</td>
<td>Frequency of Rollback vs. Number of Sites for all 4 Algorithms assuming 1 Process and 1 Resource per Site</td>
<td>104</td>
</tr>
<tr>
<td>20</td>
<td>Frequency of Detection Initiation vs. Number of Sites for Detection Algs. only assuming 1 Process and 1 Resource per Site</td>
<td>105</td>
</tr>
<tr>
<td>21</td>
<td>Frequency of Deadlock vs. Loading Factor for Centralized H&amp;V on a 3-Site Network Running 6 Processes competing for 3 and 4 Resources</td>
<td>106</td>
</tr>
<tr>
<td>22</td>
<td>The Conceptual Global Queue</td>
<td>114</td>
</tr>
<tr>
<td>23</td>
<td>N-Site Tandem Network</td>
<td>116</td>
</tr>
<tr>
<td>24</td>
<td>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&amp;V Algorithm</td>
<td>123</td>
</tr>
<tr>
<td>25</td>
<td>Comparison of Mathematical Results with Simulation Results for Varying Numbers of Sites for the Request Response Time of Distributed H&amp;V Algorithm</td>
<td>124</td>
</tr>
<tr>
<td>26</td>
<td>3-site Network Topology for Distributed Control Model</td>
<td>141</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>27a</td>
<td>&quot;PROCIO&quot; Object for Distributed Control Model</td>
<td>142</td>
</tr>
<tr>
<td>27b</td>
<td>&quot;LINE&quot; Object for Distributed Control Model</td>
<td>142</td>
</tr>
<tr>
<td>27c</td>
<td>&quot;MACHINE&quot; Object for Distributed Control Model</td>
<td>143</td>
</tr>
<tr>
<td>28</td>
<td>Centralized Control Network Topology</td>
<td>149</td>
</tr>
<tr>
<td>29a</td>
<td>&quot;PROCIO&quot; Object for Centralized Control Model</td>
<td>150</td>
</tr>
<tr>
<td>29b</td>
<td>&quot;LINE&quot; Object for Centralized Control Model</td>
<td>150</td>
</tr>
<tr>
<td>29c</td>
<td>&quot;CONTROLLER&quot; Object for Centralized Control Model</td>
<td>151</td>
</tr>
<tr>
<td>29d</td>
<td>Process &quot;MACHINE&quot; Object for Centralized Control Model</td>
<td>152</td>
</tr>
</tbody>
</table>
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Process and Resource Tables for Goldman's Distributed Algorithm Example</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>Process-Resource Matrix</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>Process-Resource Table</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>Process-Resource Table Showing Search Paths</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>Process-Resource Table for Centralized H&amp;V Example</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>Process-Resource Tables for Distributed H&amp;V Example</td>
<td>62</td>
</tr>
<tr>
<td>7</td>
<td>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</td>
<td>73</td>
</tr>
<tr>
<td>8</td>
<td>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</td>
<td>74</td>
</tr>
<tr>
<td>9</td>
<td>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</td>
<td>75</td>
</tr>
<tr>
<td>10</td>
<td>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</td>
<td>76</td>
</tr>
<tr>
<td>11</td>
<td>Average Message Units per Request for All Algorithms with Varying Numbers of Processes Competing for 6 Resources on a 3-site Network</td>
<td>77</td>
</tr>
<tr>
<td>12</td>
<td>Frequency of Rollback for All Algorithms with Varying Numbers of Processes Competing for 6 Resources on a 3-site Network</td>
<td>78</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>13</td>
<td>Frequency of Detection Initiation for the Detection Algorithms with Varying Numbers of Processes Competing for 6 Resources on a 3-site Network</td>
<td>71</td>
</tr>
<tr>
<td>14</td>
<td>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</td>
<td>80</td>
</tr>
<tr>
<td>15</td>
<td>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</td>
<td>81</td>
</tr>
<tr>
<td>16</td>
<td>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</td>
<td>82</td>
</tr>
<tr>
<td>17</td>
<td>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</td>
<td>83</td>
</tr>
<tr>
<td>18</td>
<td>Average Message Units per Request for All Algorithms with Varying Numbers of Sites, Each Running One Process and Having One Unique Resource</td>
<td>84</td>
</tr>
<tr>
<td>19</td>
<td>Frequency of Rollback for All Algorithms with Varying Numbers of Sites</td>
<td>85</td>
</tr>
<tr>
<td>20</td>
<td>Frequency of Detection Initiation for the Detection Algorithms with Varying Numbers of Sites, Each Running One Process and Having One Unique Resource</td>
<td>86</td>
</tr>
<tr>
<td>21</td>
<td>Frequency of Deadlock for Varying Loading Factor for Centralized H&amp;V on a 3-site Network Running 6 Processes Competing for 3 and 4 Resources</td>
<td>87</td>
</tr>
<tr>
<td>22</td>
<td>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</td>
<td>121</td>
</tr>
<tr>
<td>23</td>
<td>Comparison of Mathematical Results and Simulation Results for Varying Numbers of Sites for the Request Response Time of the Horizontal and Vertical Algorithm</td>
<td>122</td>
</tr>
</tbody>
</table>
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 outstanding 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 in such a way 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, pre-empt 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 Ohlmercher [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 bi-partite 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 on-line 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 pre-emption 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 1

a) Process Table

<table>
<thead>
<tr>
<th>PROCESS NAME</th>
<th>RESOURCES HELD</th>
<th>NEW REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>R2 @ S1</td>
<td>--</td>
</tr>
<tr>
<td>P2</td>
<td>--</td>
<td>R1 @ S1</td>
</tr>
</tbody>
</table>

b) Resource Table

<table>
<thead>
<tr>
<th>RESOURCE NAME</th>
<th>PROCESSES ACCESSING</th>
<th>PROCESSES WAITING</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>P3 @ S2</td>
<td>--</td>
</tr>
<tr>
<td>R2</td>
<td>P1 @ S1</td>
<td>P2 @ S1</td>
</tr>
</tbody>
</table>

SITE 2

a) Process Table

<table>
<thead>
<tr>
<th>PROCESS NAME</th>
<th>RESOURCES HELD</th>
<th>NEW REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>R1 @ S1</td>
<td>R6 @ S3</td>
</tr>
<tr>
<td>P4</td>
<td>R6 @ S3</td>
<td>R3 @ S2</td>
</tr>
<tr>
<td>P5</td>
<td>R7 @ S3</td>
<td>R4 @ S2</td>
</tr>
<tr>
<td>P6</td>
<td>R8 @ S3</td>
<td>R5 @ S2</td>
</tr>
<tr>
<td>P7</td>
<td>--</td>
<td>R9 @ S3</td>
</tr>
</tbody>
</table>

b) Resource Table

<table>
<thead>
<tr>
<th>RESOURCE NAME</th>
<th>PROCESSES ACCESSING</th>
<th>PROCESSES WAITING</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3</td>
<td>P8 @ S3</td>
<td>P4 @ S2</td>
</tr>
<tr>
<td>R4</td>
<td>P9 @ S3</td>
<td>P5 @ S2</td>
</tr>
<tr>
<td>R5</td>
<td>P10 @ S3</td>
<td>P6 @ S2</td>
</tr>
</tbody>
</table>

SITE 3

a) Process Table

<table>
<thead>
<tr>
<th>PROCESS NAME</th>
<th>RESOURCES HELD</th>
<th>NEW REQUEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8</td>
<td>R3 @ S2</td>
<td>R7 @ S3</td>
</tr>
<tr>
<td>P9</td>
<td>R4 @ S2</td>
<td>R8 @ S3</td>
</tr>
<tr>
<td>P10</td>
<td>R5 @ S2</td>
<td>--</td>
</tr>
<tr>
<td>P11</td>
<td>R9 @ S3</td>
<td>--</td>
</tr>
</tbody>
</table>

b) Resource Table

<table>
<thead>
<tr>
<th>RESOURCE NAME</th>
<th>PROCESSES ACCESSING</th>
<th>PROCESSES WAITING</th>
</tr>
</thead>
<tbody>
<tr>
<td>R6</td>
<td>P4 @ S2</td>
<td>P3 @ S2</td>
</tr>
<tr>
<td>R7</td>
<td>P5 @ S2</td>
<td>P8 @ S3</td>
</tr>
<tr>
<td>R8</td>
<td>P6 @ S2</td>
<td>P9 @ S3</td>
</tr>
<tr>
<td>R9</td>
<td>P11 @ S3</td>
<td>P7 @ S2</td>
</tr>
</tbody>
</table>

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.

\[
\begin{array}{ccc}
R1 & P10 & P3 \\
\end{array}
\]

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.

\[
\begin{array}{ccc}
R6 & P10 & P3 \\
\end{array}
\]

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.

\[
\begin{array}{cccc}
R6 & P10 & P3 & P4 \\
\end{array}
\]
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.

<table>
<thead>
<tr>
<th>R3</th>
<th>P10</th>
<th>P3</th>
<th>P4</th>
<th>P8</th>
</tr>
</thead>
</table>

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.

<table>
<thead>
<tr>
<th>R7</th>
<th>P10</th>
<th>P3</th>
<th>P4</th>
<th>P8</th>
<th>P5</th>
</tr>
</thead>
</table>

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.
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.

| R4 | P10 | P3 | P4 | P8 | P5 | P9 |

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, \( P_i, 1 \leq i \leq n \), and each resource \( R_j, 1 \leq j \leq m \), and with edges representing process interactions in the system. Formally, the process-resource graph is a bipartite directed graph, \( G = (V,E) \), where 

\[
V = \{P_1,P_2,\ldots,P_n\} \cup \{R_1,R_2,\ldots,R_m\}
\]

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, \( S_1, S_2, \) and \( S_3 \). \( \{P_1,\ldots,P_{11}\} \) and \( \{R_1,\ldots,R_9\} \) form the process nodes and resource nodes, respectively. \( \{P_1,P_2\}, \{P_3,\ldots,P_7\}, \{P_8,\ldots,P_{11}\} \) run at sites \( S_1, S_2 \) and \( S_3 \), respectively, while \( \{R_1,R_2\}, \{R_3,R_4,R_5\} \) and \( \{R_6,R_7,R_8,R_9\} \)
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 R5, 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 \( V = \{P1,...,Pn\} \cup \{R1,...,Rm\} \). 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

<table>
<thead>
<tr>
<th>Process Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resource Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
</tr>
<tr>
<td>R2</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>Rm</td>
</tr>
</tbody>
</table>

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 $\rho$ is used to represent a rank. Thus $\rho [Pi,Rj] = (0,e(\text{exclusive}))$ means Pi has gained exclusive access to resource Rj and $\rho [Pi,Rj] = (0,s(\text{shared}))$ means Pi has gained shared access to Rj. $[Pi,Rj] = (j,e)$ and $[Pi,Rj] = (j,s)$ means Pi is jth 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

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>(0,e)</td>
<td></td>
<td>(1,e)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>(1,e)</td>
<td>(2,e)</td>
<td></td>
<td>(0,e)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td>(0,e)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td></td>
<td></td>
<td>(0,e)</td>
<td>(1,e)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>(0,e)</td>
<td></td>
<td>(1,e)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td>(0,e)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

\[
\text{for } i := 1 \text{ to } n \text{ do}

\text{if PRTABLE}[Rj, Pi] \cdot \text{Rank} > 0 \text{ then}

\quad \text{PRTABLE}[Rj, Pi] \cdot \text{Rank} := \text{PRTABLE}[Rj, Pi] \cdot \text{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

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>(0,e)</td>
<td></td>
<td>(1,e)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>(1,e)</td>
<td></td>
<td></td>
<td></td>
<td>(0,e)</td>
<td>(1,e)</td>
</tr>
<tr>
<td>R3</td>
<td>(0,e)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>(0,e)</td>
<td></td>
<td>(1,e)</td>
<td></td>
<td>(0,e)</td>
<td>(1,e)</td>
</tr>
<tr>
<td>R5</td>
<td>(0,e)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[---] 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 o [Pi,R] = 0. The routine is given below.

Procedure Horizontal (R,h,P);

```plaintext
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 begin
    if rank = 0 then begin
      h := h + 1;
      P [h] := PT [i]
    end;
  end:
end;
```
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 $\rho [P,R] > 0$ 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 ($P,R,v$);

\[
\begin{align*}
& P = \text{Process returned by Horizontal} \\
& R = \text{Resource which } P \text{ is waiting for, if waiting} \\
& v = \text{flag, } v \text{ is true if } P \text{ is waiting, false otherwise} \\
& \text{PRT = Process-Resource table} \\
& \text{RT = Resource table} \\
& m = \text{current number of resources} \\
& \text{Mark} = n \text{ dimensional boolean array,} \\
& n \text{ is the number of processes, each entry corresponding to a process} \\
\end{align*}
\]

Begin

\[
\begin{align*}
v := \text{false;} \\
\text{for } i := 1 \text{ to } m \text{ do} \\
\text{if (PRT } [i,P] > 0 \text{) and (not Mark } [P] \text{) then} \\
\text{begin} \\
\text{v := true;} \\
R := \text{RT}[i]; \\
\text{Mark } [P] := \text{true;} \\
\text{return} \\
\text{end;} \\
\text{end;} \\
\end{align*}
\]
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 EP, 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);
end;
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 $V = \{P_1, P_2, \ldots, P_n\} \cup \{R_1, R_2, \ldots, R_m\}$. Consider a graph $G' = (V', E')$ constructed from $G$ as follows. Let $P_i$ and $R_j$ be vertices in $G$ such that there is an edge from $P_i$ to $R_j$. 

```plaintext
if (stackptr = 1) and (not V) then done := true end;
end;
end; /* H&V */
```
Combine the pair $\pi \rightarrow r_j$ 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 $P_k$ be an active process vertex in $G$, such that there is no out-degree from vertex $P_k$, 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,

$$V' = \{P'i | \pi \in V \text{ is blocked waiting for } r_j\} \cup \{P'k | P_k \in V \text{ is active}\}$$

Denote all blocked nodes in $G'$ with the symbol $\square$ and all active nodes with the symbol $\{\}$. Let $V'i$ and $V'k$ be blocked vertices in $G'$, where $V'i = \pi \rightarrow r_j$ and $V'k = P_k \rightarrow r_1$. Add an edge from $V'i$ to $V'k$ if $r_j$ is accessed by $P_k$. Add an edge from a blocked vertex $\pi \rightarrow r_j$ to an active vertex $\{P_k\}$ if $r_j$ is being accessed by $P_k$. The graph $G'$, so constructed, will be called H&V transformed graph.

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

![Diagram](image)

**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 \( L(v',w') \leq 1 \) get visited. Now assume that all vertices w' with \( L(v',w') \leq d \) 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 \( d \geq 1 \). 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

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Os</td>
<td></td>
<td></td>
<td>Os</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td>Os</td>
<td>2s</td>
<td>le</td>
<td></td>
<td>Os</td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td>le</td>
<td></td>
<td></td>
<td>Ce</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>1e</td>
<td></td>
<td>Oe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>2e</td>
<td></td>
<td></td>
<td></td>
<td>Oe</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>e</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>exclusive access</td>
<td>shared access</td>
<td></td>
</tr>
</tbody>
</table>

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, \( \{ P_i, P_k, 1 \leq k < n, i \neq k \} \). 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);

1     Begin
2         done := false;  stackptr := 1;  Rk := Rj;
3     While not done do
4         begin
5             Horizontal (Rk,h,P);
6             if Pk ∈ P, 1 <= k <= h, such that Pk = P# then
7                 begin
8                     deadlock := true;
9                     done := true
10                end else
11                 begin
12                     while h >= 1 do
13                         begin
14                             stack [stackptr] := P[h];
15                             stackptr := stackptr + 1;
16                             h := h - 1
17                         end;
18                 end
19                 v := false;
20             while (stackptr > 1) and (not v) do
21                 begin
22                     stackptr := stackptr - 1 ;
23                     Pk := stack [stackptr];
24                     Vertical (Pk,Rk,v);
25                     if not v then add Pk to PP
26                 end;
27                 if (stackptr = 1) and (not v) then done := true
28             end;
29         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 \( \{R_k, P_k\} \) be an unmarked entry greater than zero in the process-resource table. Call procedure Detect with \( P_k \) and \( R_k \) as arguments and do not check for deadlock in Detect. The procedure returns the path \( P_1, P_2, \ldots, P_j \), \( j \geq 1 \), in PP. Append the set \( \{P_k, P_1, P_2, \ldots, P_j\} \) as entry in the Disjoint Path. \( P_k \) is the identification of this set. Mark the \( \{R_k, P_k\} \) 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 P1 in {P1, P2, ..., Pj} such that P1 = 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 \((V_1,V_2), (V_2,V_3), \ldots, (V_{n-1},V_n)\). It may be represented by the sequence \( V_1, V_2, \ldots, V_n \) 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 \( G_1 \cup G_2 \) of two directed graphs \( G_1 \) and \( G_2 \) is that directed subgraph with vertex set \( V(G_1) \cup V(G_2) \) and edge set \( E(G_1) \cup E(G_2) \).

Let \( G_1 = (V_1,E_1) \) and \( G_2 = (V_2,E_2) \) be two directed acyclic graphs. The following are stated from the definitions.

1. \( G = G_1 \cup G_2 \) is a forest if \( V(G_1) \cap V(G_2) = \emptyset \)
2. \( G = G_1 \cup G_2 \) combine into one directed graph if \( G_1 \) and \( G_2 \) contain at least one vertex in common, assuming \( G_1 \) and \( G_2 \) are connected.
Lemma 5.1 Let $G_1 = (V_1,E_1)$ and $G_2 = (V_2,E_2)$ be two directed acyclic graphs with directed paths $V_1,V_2,\ldots,V_i$, $i \geq 3$, and $U_1,U_2,\ldots,U_j$, $j \geq 3$, respectively. If the two paths contain one vertex, $W$, in common, then $G = G_1 \cup G_2$ has a path containing $W$.

Proof: Let $V_p = V_1,V_2,\ldots,V_i$ and $U_p = U_1,U_2,\ldots,U_j$. Then $G_1 \cup G_2$ will contain the paths $V_1,V_2,\ldots,V_i \cup U_1,U_2,\ldots,U_j$.

Case 1: $W = V_i = U_1$. Then the path of $G$ containing $W$ will be $V_1,V_2,\ldots,W,U_2,\ldots,U_j$.

Case 2: $W = V_1 = U_j$. Then the path containing $W$ will be $U_1,U_2,\ldots,W,V_2,\ldots,V_i$.

Case 3: $W = V_k = U_1$, $1 < k < i$ and $1 < j$. $G$ will contain the paths $V_1,V_2,\ldots,W,\ldots,V_i$; $V_1,V_2,\ldots,W,\ldots,U_j$; $U_1,U_2,\ldots,W,\ldots,U_j$; $U_1,U_2,\ldots,W,\ldots,V_i$.

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

Lemma 5.2 Let $G_1 = (V_1,E_1)$ and $G_2 = (V_2,E_2)$ be two directed acyclic graphs with directed paths $V_p = V_1,V_2,\ldots,V_i$, $i \geq 3$ and $U_p = U_1,U_2,\ldots,U_j$, $j \geq 3$, respectively. $V_1,V_2,\ldots,V_i$ and $U_1,U_2,\ldots,U_j$ are distinct vertices of $G_1$ and $G_2$, respectively. If there are two common vertices $X$ and $Y$ to both graphs such that there is a path from $X$ to $Y$ in $G_1$, and a path from $Y$ to $X$ in $G_2$, each of length at least 2, contained in $V_p$ and $U_p$, respectively, then $G = G_1 \cup G_2$ contains a cycle.
Proof: Let $G = G_1 U G_2$. Then $G$ contains the paths $Vp U Up = V_1, V_2, ..., V_i U U_1, U_2, ..., U_j$.
Let $\lambda$ be the smallest integer such that $V_\lambda = U_k$.
Then $V_1, ..., V_\lambda, U_k+1, ..., U_j$ 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 $P_1, P_2, ..., P_k$ wait for resource $R#$. Assume the system is deadlock free. If another process $P#$ later request for $R#$, then $P_1, P_2, ..., P_k$ 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 $P#, P_1, P_2, ..., P_n$, as shown in Figure 5.
From the figure it is apparent that none of the processes $P_1, P_2, \ldots, P_k$ can be in the deadlock path, since $P_1, P_2, \ldots, P_k$ and $P#$ all have directed edges into $R#$. Therefore, $P_1, P_2, \ldots, P_k$ 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 $P_1$ request for resource $R#$. Consider a global deadlock cycle, as shown in Figure 6, consisting of processes $P_1, P_2, \ldots, P_n$. Assume that none of the processes involved in the deadlock is aborted or rolled back while the search induced by $P_1$'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 P1,P12,...,Pik. Note that processes P1,P12,...,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 $P_1 \neq P_l$.

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 \( \{P_j, P_g\} \) be a set in Disjoint Path. This implies a path beginning at \( P_j \) and ending at \( P_g \) exists in the global process-resource graph. Also, a path exists beginning from \( P_1 \) and ending at \( P_1 \), as shown above. Now, step 8 performs a union of \( P_1, \ldots, P_l \) and \( P_j, \ldots, P_g \), for all sets in which \( P_1 = P_j \). Therefore, by Lemma 5.1, step 8 will produce the Path \( P_1, \ldots, P_l \cup P_j, \ldots, P_g = P_1, \ldots, P_m, \ldots, P_g \), if \( P_1 = P_j \), where \( P_l = P_j = P_m \). After this operation, \( P_g \) will be in Path. Step 8 also checks if \( P_g = P_1 \). 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 \( P_l \)'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 decentralized algorithm, we shall prove Corollary 5.1 and then make some observations.
Corollary 5.1  Let process \( P_i \) request for resource \( R_i \) at site \( S_i \), and at the same point in time process \( P_j \) requests for resource \( R_j \) at site \( S_j \). The distributed H&V algorithm initiated simultaneously at \( S_i \) and \( S_j \) 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 \( P_i \) and \( P_j \).

Case 2: Different search paths for process \( P_i \) and process \( P_j \).

Case 1:  We shall assume that neither \( P_i \) nor \( P_j \) is rolled back or aborted, while the searches initiated by \( S_i \) and \( S_j \), respectively, are in progress.

Assume that at the instance \( S_i \) and \( S_j \) initiate their H&V algorithm, a deadlock cycle exists in the network containing both processes \( P_i \) and \( P_j \). Since a deadlock exists, processes \( P_i \) and \( P_j \) 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 \( S_i \) and \( S_j \) 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 \( S_i \) and \( S_j \).
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.

**Case 2:** 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.

**Observation 5.3**
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.

**Observation 5.4**
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 algorithm. □
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 OBFL 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

<table>
<thead>
<tr>
<th>SITE S1:</th>
<th>R1</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td>P10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SITE S2: R3

<table>
<thead>
<tr>
<th>SITE S2:</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P8</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SITE S3:

<table>
<thead>
<tr>
<th>SITE S3:</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
<th>R9</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| P10      |    |    |

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.

<table>
<thead>
<tr>
<th>Detection</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td></td>
</tr>
<tr>
<td>Destination</td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>False</td>
</tr>
<tr>
<td>Path</td>
<td></td>
</tr>
<tr>
<td>Disjoint Path</td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Detection</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>False</td>
</tr>
<tr>
<td>P3</td>
<td>{p2,p1}</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
</tr>
<tr>
<td>S3</td>
</tr>
<tr>
<td>P10</td>
</tr>
<tr>
<td>R1</td>
</tr>
<tr>
<td>False</td>
</tr>
<tr>
<td>P3</td>
</tr>
</tbody>
</table>

\{P2,P1\}, \{P4,P8\}, \{P5,P9\}, \{P6,P10\}

Site S3:

P3 \rightarrow 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 look as follows.

\{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.

\[ P_{10} = P_{\#}, \text{ therefore } S_3 \text{ will detect the deadlock, and forward the message as shown below to } S_1. \]

\begin{table}[h]
\centering
\begin{tabular}{l}
Detection \\
\hline
S1 \\
\hline
S1 \\
\hline
P10 \\
\hline
R1 \\
\hline
True
\end{tabular}
\end{table}

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

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>PREEMPTION</th>
<th>DISTRIBUTED HLV</th>
<th>DISTRIBUTED GOLDMAN</th>
<th>CENTRALIZED HLV</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>29,494</td>
<td>6,744.374</td>
<td>16,755.306</td>
<td>17,410.667</td>
</tr>
<tr>
<td>5</td>
<td>72,955.2</td>
<td>6,186.310</td>
<td>7,680.006</td>
<td>26,182.4</td>
</tr>
<tr>
<td>6</td>
<td>106,740</td>
<td>10,971.068</td>
<td>11,515.303</td>
<td>31,175.333</td>
</tr>
<tr>
<td>7</td>
<td>129,118.31</td>
<td>18,757.953</td>
<td>20,281.519</td>
<td>40,040.057</td>
</tr>
<tr>
<td>10</td>
<td>159,249</td>
<td>32,235.504</td>
<td>23,058.346</td>
<td>64,369.6</td>
</tr>
</tbody>
</table>

* 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>
<thead>
<tr>
<th>Number of Processes</th>
<th>PREEMPTION</th>
<th>DISTRIBUTED H&amp;V</th>
<th>DISTRIBUTED GOLDMAN</th>
<th>CENTRALIZED H&amp;V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average Throughput</td>
<td>Standard Deviation</td>
<td>Standard Error</td>
<td>Average Throughput</td>
</tr>
<tr>
<td>3</td>
<td>.00003391</td>
<td>.00000822</td>
<td>1.00002042</td>
<td>.00005744</td>
</tr>
<tr>
<td>5</td>
<td>.00001371</td>
<td>.00000474</td>
<td>1.00000588</td>
<td>.00003819</td>
</tr>
<tr>
<td>6</td>
<td>.00000937</td>
<td>.00000115</td>
<td>1.00000121</td>
<td>.00000656</td>
</tr>
<tr>
<td>7</td>
<td>.00000774</td>
<td>.00000369</td>
<td>1.00000034</td>
<td>.00002497</td>
</tr>
<tr>
<td>10</td>
<td>.00000628</td>
<td>.00000294</td>
<td>1.00000021</td>
<td>.00001554</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>PREEMPTION</th>
<th>DISTRIBUTED HAY</th>
<th>DISTRIBUTED GOLDFMAN</th>
<th>CENTRALIZED HAY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average Response Time*</td>
<td>Standard Deviation</td>
<td>Standard Error*</td>
<td>Average Response Time*</td>
</tr>
<tr>
<td>3</td>
<td>58.821</td>
<td>30.709</td>
<td>10.022</td>
<td>141.917</td>
</tr>
<tr>
<td>5</td>
<td>59.070</td>
<td>32.043</td>
<td>18.769</td>
<td>189.091</td>
</tr>
<tr>
<td>6</td>
<td>60.660</td>
<td>30.462</td>
<td>23.620</td>
<td>287.085</td>
</tr>
<tr>
<td>7</td>
<td>64.144</td>
<td>31.070</td>
<td>14.144</td>
<td>311.044</td>
</tr>
</tbody>
</table>

* 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 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

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>PREEMPTION</th>
<th>DISTRIBUTED H&amp;V</th>
<th>DISTRIBUTED GOLDMAN</th>
<th>CENTRALIZED H&amp;V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average Throughput</td>
<td>Standard Deviation</td>
<td>Standard Error*</td>
<td>Average Throughput</td>
</tr>
<tr>
<td>3</td>
<td>0.0170</td>
<td>0.0014</td>
<td>1.000342</td>
<td>0.007046</td>
</tr>
<tr>
<td></td>
<td>0.0167</td>
<td>0.0027</td>
<td>1.000739</td>
<td>0.005266</td>
</tr>
<tr>
<td>4</td>
<td>0.01648</td>
<td>0.0023</td>
<td>1.000273</td>
<td>0.003483</td>
</tr>
<tr>
<td>5</td>
<td>0.01559</td>
<td>0.0016</td>
<td>1.0002124</td>
<td>0.003215</td>
</tr>
<tr>
<td>6</td>
<td>0.01483</td>
<td>0.0021</td>
<td>1.0002614</td>
<td>0.001899</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>Preemption</th>
<th>Distributed H&amp;V</th>
<th>Distributed Goldman</th>
<th>Centralized H&amp;V</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2.896</td>
<td>4.055</td>
<td>4.417</td>
<td>6.469</td>
</tr>
<tr>
<td>5</td>
<td>3.315</td>
<td>4.436</td>
<td>5.514</td>
<td>6.519</td>
</tr>
<tr>
<td>6</td>
<td>3.428</td>
<td>4.592</td>
<td>5.778</td>
<td>6.792</td>
</tr>
<tr>
<td>7</td>
<td>3.613</td>
<td>4.888</td>
<td>6.028</td>
<td>6.865</td>
</tr>
<tr>
<td>10</td>
<td>3.884</td>
<td>5.180</td>
<td>6.653</td>
<td>7.130</td>
</tr>
</tbody>
</table>
TABLE 12. Frequency of Rollback for all Algorithms with Varying Numbers of Processes Competing for 6 Resources on a 3-Site Network

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>Preemption</th>
<th>Distributed H&amp;V</th>
<th>Distributed Goldman</th>
<th>Centralized H&amp;V</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.1940</td>
<td>0.0268</td>
<td>0.02778</td>
<td>0.04082</td>
</tr>
<tr>
<td>5</td>
<td>0.48</td>
<td>0.03636</td>
<td>0.05405</td>
<td>0.04938</td>
</tr>
<tr>
<td>6</td>
<td>0.5333</td>
<td>0.0423</td>
<td>0.06098</td>
<td>0.05195</td>
</tr>
<tr>
<td>7</td>
<td>0.5429</td>
<td>0.05</td>
<td>0.07534</td>
<td>0.05208</td>
</tr>
<tr>
<td>10</td>
<td>0.63</td>
<td>0.0683</td>
<td>0.08163</td>
<td>0.05594</td>
</tr>
</tbody>
</table>
TABLE 13. Frequency of Detection Initiation for the Detection Algorithms with Varying Numbers of Processes Competing for 6 Resources on a 3-Site Network

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>Distributed H&amp;V</th>
<th>Distributed Goldman</th>
<th>Centralized H&amp;V</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>.25</td>
<td>.2778</td>
<td>.3061</td>
</tr>
<tr>
<td>5</td>
<td>.4264</td>
<td>.4459</td>
<td>.4321</td>
</tr>
<tr>
<td>6</td>
<td>.4566</td>
<td>.4912</td>
<td>.5125</td>
</tr>
<tr>
<td>7</td>
<td>.50</td>
<td>.5274</td>
<td>.5208</td>
</tr>
<tr>
<td>10</td>
<td>.5714</td>
<td>.5918</td>
<td>.5804</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Number of Sites</th>
<th>PREEEMPTION</th>
<th>DISTRIBUTED HAV</th>
<th>DISTRIBUTED GOLDMAN</th>
<th>CENTRALIZED HAV</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>51,636.667</td>
<td>7,379.975</td>
<td>10,321.571</td>
<td>16,741.667</td>
</tr>
<tr>
<td>5</td>
<td>75,680.5</td>
<td>5,796.429</td>
<td>7,206.412</td>
<td>28,076.60</td>
</tr>
<tr>
<td>8</td>
<td>117,772.75</td>
<td>14,726.464</td>
<td>12,287.555</td>
<td>44,311.625</td>
</tr>
<tr>
<td>10</td>
<td>149,849.2</td>
<td>17,879.093</td>
<td>12,770.163</td>
<td>59,414.7</td>
</tr>
<tr>
<td>12</td>
<td>182,824.5</td>
<td>16,268.765</td>
<td>10,332.053</td>
<td>88,688.5</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th>Number of Sites</th>
<th>PREEMPTIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average Throughput</td>
</tr>
<tr>
<td>3</td>
<td>.00001937</td>
</tr>
<tr>
<td>5</td>
<td>.00001321</td>
</tr>
<tr>
<td>8</td>
<td>.00000849</td>
</tr>
<tr>
<td>10</td>
<td>.00000667</td>
</tr>
<tr>
<td>12</td>
<td>.00000547</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISTRIBUTED HAV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Throughput</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DISTRIBUTED GOLDMAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Throughput</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CENTRALIZED HAV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Throughput</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th>Number of Sites</th>
<th>PREEMPTION</th>
<th>DISTRIBUTED H&amp;V</th>
<th>DISTRIBUTED GOLDMAN</th>
<th>CENTRALIZED H&amp;V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average Response</td>
<td>Average</td>
<td>Average</td>
<td>Average</td>
</tr>
<tr>
<td></td>
<td>Time*</td>
<td>Response</td>
<td>Response</td>
<td>Response</td>
</tr>
<tr>
<td></td>
<td>Standard Deviation</td>
<td>Time*</td>
<td>Standard</td>
<td>Standard</td>
</tr>
<tr>
<td></td>
<td>Standard Error*</td>
<td>Standard</td>
<td>Error*</td>
<td>Error*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deviation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>58.883</td>
<td>1,6147</td>
<td>10.8896</td>
<td>257.6009</td>
</tr>
<tr>
<td></td>
<td>138.4</td>
<td>95.778</td>
<td>12.19</td>
<td>107.6333</td>
</tr>
<tr>
<td></td>
<td>1.6147</td>
<td>10.8896</td>
<td>±32.719</td>
<td>±140.2454</td>
</tr>
<tr>
<td>5</td>
<td>108.61</td>
<td>9.69</td>
<td>14.5285</td>
<td>463.9586</td>
</tr>
<tr>
<td></td>
<td>264.0</td>
<td>236.272</td>
<td>±60.503</td>
<td>191.8837</td>
</tr>
<tr>
<td></td>
<td>4.9285</td>
<td>16.1558</td>
<td>±101.544</td>
<td>±179.2096</td>
</tr>
<tr>
<td>8</td>
<td>209.01</td>
<td>44.8</td>
<td>16.1558</td>
<td>1,971.1405</td>
</tr>
<tr>
<td></td>
<td>826.408</td>
<td>494.571</td>
<td>±101.544</td>
<td>±658.5033</td>
</tr>
<tr>
<td></td>
<td>826.408</td>
<td>494.571</td>
<td>±101.544</td>
<td>±121.058</td>
</tr>
<tr>
<td>10</td>
<td>310.76</td>
<td>83.78</td>
<td>126.7583</td>
<td>3,612.616</td>
</tr>
<tr>
<td></td>
<td>1,492.7</td>
<td>627.673</td>
<td>±102.520</td>
<td>±2,117.614</td>
</tr>
<tr>
<td></td>
<td>311.76</td>
<td>627.673</td>
<td>±102.520</td>
<td>±1402.6404</td>
</tr>
<tr>
<td>12</td>
<td>727.97</td>
<td>345.86</td>
<td>1117.016</td>
<td>5,603.231</td>
</tr>
<tr>
<td></td>
<td>2,411.22</td>
<td>2,489.296</td>
<td>±1406.565</td>
<td>±2,316.131</td>
</tr>
<tr>
<td></td>
<td>727.97</td>
<td>345.86</td>
<td>±1117.016</td>
<td>±1794.0103</td>
</tr>
<tr>
<td></td>
<td>1,195.99</td>
<td>±1,195.99</td>
<td>±1794.0103</td>
<td>±1404.6433</td>
</tr>
</tbody>
</table>

* 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.

<table>
<thead>
<tr>
<th>Number of Sites</th>
<th>PREEMPTION</th>
<th>DISTRIBUTED H&amp;V</th>
<th>DISTRIBUTED GOLDMAN</th>
<th>CENTRALIZED H&amp;V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average Throughput</td>
<td>Standard Deviation</td>
<td>Standard Error*</td>
<td>Average Throughput</td>
</tr>
<tr>
<td>3</td>
<td>.01698</td>
<td>.00053</td>
<td>±.00034</td>
<td>.007225</td>
</tr>
<tr>
<td>5</td>
<td>.009207</td>
<td>.00082</td>
<td>±.00043</td>
<td>.003788</td>
</tr>
<tr>
<td>8</td>
<td>.004784</td>
<td>.00091</td>
<td>±.00032</td>
<td>.001377</td>
</tr>
<tr>
<td>10</td>
<td>.003137</td>
<td>.00087</td>
<td>±.00020</td>
<td>.000718</td>
</tr>
<tr>
<td>12</td>
<td>.001374</td>
<td>.00064</td>
<td>±.00023</td>
<td>.0004147</td>
</tr>
</tbody>
</table>

* 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

<table>
<thead>
<tr>
<th>Number of Sites</th>
<th>Preemption</th>
<th>Distributed H&amp;V</th>
<th>Distributed Goldman</th>
<th>Centralized H&amp;V</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2.951</td>
<td>4.579</td>
<td>6.5046</td>
<td>6.099</td>
</tr>
<tr>
<td>5</td>
<td>2.995</td>
<td>7.557</td>
<td>10.224</td>
<td>9.864</td>
</tr>
<tr>
<td>8</td>
<td>3.041</td>
<td>13.688</td>
<td>19.5776</td>
<td>15.103</td>
</tr>
<tr>
<td>10</td>
<td>3.213</td>
<td>16.321</td>
<td>21.6783</td>
<td>17.782</td>
</tr>
</tbody>
</table>
TABLE 19. Frequency of Rollback for All Algorithms with Varying Numbers of Sites

<table>
<thead>
<tr>
<th>Number of Sites</th>
<th>Preemption</th>
<th>Distributed D&amp;V</th>
<th>Distributed Goldman</th>
<th>Centralized H&amp;V</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>.3415</td>
<td>.0286</td>
<td>.0486</td>
<td>.0340</td>
</tr>
<tr>
<td>5</td>
<td>.3603</td>
<td>.0328</td>
<td>.0627</td>
<td>.0563</td>
</tr>
<tr>
<td>8</td>
<td>.3962</td>
<td>.0538</td>
<td>.0680</td>
<td>.0581</td>
</tr>
<tr>
<td>10</td>
<td>.4127</td>
<td>.0631</td>
<td>.0779</td>
<td>.0651</td>
</tr>
<tr>
<td>12</td>
<td>.4541</td>
<td>.0712</td>
<td>.0853</td>
<td>.0741</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of Sites</th>
<th>Distributed H&amp;V</th>
<th>Distributed Goldman</th>
<th>Centralized H&amp;V</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>.351</td>
<td>.4107</td>
<td>.3604</td>
</tr>
<tr>
<td>5</td>
<td>.4115</td>
<td>.4318</td>
<td>.4225</td>
</tr>
<tr>
<td>8</td>
<td>.4378</td>
<td>.4828</td>
<td>.4494</td>
</tr>
<tr>
<td>10</td>
<td>.4521</td>
<td>.5065</td>
<td>.4749</td>
</tr>
<tr>
<td>12</td>
<td>.5108</td>
<td>.5504</td>
<td>.5270</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Loading Factor</th>
<th>3 Resources</th>
<th>4 Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>.2</td>
<td>.0385</td>
<td>.0418</td>
</tr>
<tr>
<td>.4</td>
<td>.061</td>
<td>.0661</td>
</tr>
<tr>
<td>.6</td>
<td>.0828</td>
<td>.0895</td>
</tr>
<tr>
<td>.8</td>
<td>.0833</td>
<td>.1039</td>
</tr>
<tr>
<td>.9</td>
<td>.1075</td>
<td>.1193</td>
</tr>
</tbody>
</table>
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, based on 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)

NUMBER OF PROCESSES

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

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)

NUMBER OF PROCESSES

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

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

FIGURE 11
FREQUENCY OF ROLLBACK VS NUMBER OF PROCESSES FOR ALL 4 ALGORITHMS ON A 3-SITE NETWORK

FIGURE 12

- PREEMPTION (1)
- DISTRIBUTED H&V (2)
- DISTRIBUTED GOLDMAN (3)
- CENTRALIZED H&V (4)
FREQUENCY OF DETECTION INITIATION
VS NUMBER OF PROCESSES
FOR DETECTION ALGORITHMS ONLY ON A 3-SITE NETWORK

FIGURE 13

FREQUENCY OF DETECTION INITIATION * 10^(-2)

NUMBER OF PROCESSES

DISTRIBUTED H&V (1)
DISTRIBUTED GOLDMAN (2)
CENTRALIZED H&V (3)
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

FIGURE 14
PROCESS AVERAGE THROUGHPUT VS NUMBER OF SITES
FOR ALL 4 ALGORITHMS
ASSUMING 1 PROCESS & 1 RESOURCE PER SITE

FIGURE 15

PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)
REQUEST AVERAGE RESPONSE TIME VS NUMBER OF SITES FOR ALL 4 ALGORITHMS ASSUMING 1 PROCESS & 1 RESOURCE PER SITE

I. 1
2
4
6
8
10
12
NUMBER OF SITES

FIGURE 16

AVERAGE RESPONSE TIME (UNITS OF 100)

(3)
(4)
(2)
(1)

PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)
REQUEST AVERAGE THROUGHPUT VS NUMBER OF SITES
FOR ALL 4 ALGORITHMS
ASSUMING 1 PROCESS & 1 RESOURCE PER SITE

FIGURE 17

PREEMPTION (1)
DISTRIBUTED H&V (2)
DISTRIBUTED GOLDMAN (3)
CENTRALIZED H&V (4)
AVERAGE MESSAGE UNITS PER REQUEST
VS NUMBER OF SITES FOR ALL 4 ALGORITHMS
ASSUMING 1 PROCESS & 1 RESOURCE PER SITE

FIGURE 18
FREQUENCY OF ROLLBACK VS NUMBER OF SITES
FOR ALL 4 ALGORITHMS
ASSUMING 1 PROCESS & 1 RESOURCE PER SITE

FIGURE 19

- - - - - - PREEMPTION (1)
- - - - - DISTRIBUTED H&V (2)
- - - - - DISTRIBUTED GOLDMAN (3)
- - - - - CENTRALIZED H&V (4)
FREQUENCY OF DETECTION INITIATION
VS NUMBER OF SITES FOR DETECTION ALGS ONLY
ASSUMING 1 PROCESS & 1 RESOURCE PER SITE

FIGURE 20

DISTRIBUTED H&V (1)
DISTRIBUTED GOLDMAN (2)
CENTRALIZED H&V (3)
FREQUENCY OF DEADLOCK VS LOADING FACTOR
FOR CENTRALIZED H&V ON A 3-SITE NETWORK RUNNING
6 PROCESSES COMPETING FOR 3 & 4 RESOURCES

FIGURE 21
6 PROCESSES and 3 RESOURCES(1)
6 PROCESSES and 4 RESOURCES(2)
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 earlier, 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]:

\[ T = \bar{x} + w \]  

(1)
where $T =$ response time
$x =$ average service time
$W =$ average waiting time (queueing time)

The average waiting time is given by

$$W = \frac{P_m}{mu(1 - \rho)}$$  \hspace{1cm} (2)

where $P_m =$ probability that all $m$ servers are busy
$m =$ number of servers
$\mu = \frac{1}{\bar{x}} =$ service rate
$\rho =$ utilization of the system

The utilization of the system is given by

$$\rho = \frac{\lambda}{m\mu} = \frac{\lambda\bar{x}}{m}$$  \hspace{1cm} (3)

$\lambda =$ 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

$$P_k = \begin{cases} 
  P_0 \frac{(m\rho)^k}{k!} & \text{for } k \leq m \\
  P_0 \frac{(\rho)^k m^m}{m!} & \text{for } k > m 
\end{cases}$$  \hspace{1cm} (4)

where

$$P_0 = \left[ \frac{\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m! (1-\rho)}}{-1} \right]$$  \hspace{1cm} (5)
From these, the expression for $P_m$ is given by

$$P_m = p_0 \frac{(m \rho)^m}{m! (1 - \rho)}$$

(6)

and

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

(7)

Therefore, equation (1) becomes

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

(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

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

(9)

where

$z =$ average number of servers including non-integer values
\[ \rho = \frac{\lambda}{z \mu} \]

\[ P_z = \frac{(z\rho)^z}{z!} \frac{1}{1 - \rho} + \sum_{k=0}^{z-1} \frac{(z\rho)^k}{k!} \frac{1}{z!} \left( \frac{1}{1 - \rho} \right) \]

(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]:

\[ P_z \approx \frac{z\rho^z}{1 + (z-1)\rho} \]

(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 \( \rho \), 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:

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

(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.

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

![Figure 22. The Conceptual Global Queue](image-url)
\[ F(t) = \Pr \{ \text{arrival occurs before time } t \} \]
\[ = 1 - \Pr \{ \text{all arrivals occur after time } t \} \]
\[ = 1 - \prod_{k=1}^{n} \Pr \{ \text{arrival occurs after time } t \} \]
\[ = 1 - \prod_{k=1}^{n} \left( 1 - \Pr \{ t_k > t \} \right) \]
\[ = 1 - \prod_{k=1}^{n} (1 - (1 - e^{-\lambda_k t})) \]
\[ = 1 - \prod_{k=1}^{n} e^{-\lambda_k t} \]

Therefore,
\[ F(t) = 1 - e^{-\sum_{k=1}^{n} \lambda_k t} \quad (13) \]

Assume \[ \lambda_1 = \lambda_2 = \cdots = \lambda_n = \lambda \]

Therefore, the probability density distribution of \( t \) is
\[ f(t) = n \lambda e^{-n \lambda t} \quad (14) \]

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 \( \mu \). 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

\[
B(t) = \Pr \{\text{arrival to site } i \text{ at time } t \leq t\}, \quad \text{and}
\]

\[
b(t) = \frac{dB}{dt} = \Pr \{\bar{t} = t\}.
\]

Then

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

\[
\approx \rho e^{-\mu t} + (1-\rho) \int_0^t \lambda e^{-\lambda(t-x)} \mu e^{-\mu t}dx
\]

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

\[
\begin{array}{c}
\lambda \\
\mu_1 \\
\mu_2 \\
\vdots \\
\mu_n
\end{array}
\]
\[ d(t) = \lambda e^{-\mu t} + (1-\rho)\lambda \mu \int_0^t e^{-\lambda t} e^{-x(\mu-\lambda)} \, dx \]

\[ = \lambda e^{-\mu t} + \frac{(1-\rho)\lambda \mu e^{-\lambda t}}{\mu-\lambda} \left[ e^{-x(\mu-\lambda)} \right]_0^t \]

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

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

This expression simplifies to give

\[ d(t) = \lambda e^{-\lambda t} \quad (15) \]

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

\[ \bar{x}_a = N\bar{x}_s \quad (16) \]

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, $\bar{x}$ 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

$$\bar{x} = \bar{x}_a + \bar{x}_c$$  \hspace{1cm} (17)

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

$$T_{H&V} = \bar{x} + \frac{pz}{z(1-\rho)}$$  \hspace{1cm} (18)

However, the arrival rate, $\lambda$, 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 $\lambda_n$ is per site arrival rate. $\rho$ in equation (18) is the global network utilization and is therefore defined by

$$\rho = \frac{n\lambda_n}{z\mu} = \frac{n\lambda}{z}$$  \hspace{1cm} (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:

$$T_{H&V} = \bar{x} + \frac{z0}{[1+(z-1)\rho](1-\rho)}$$

(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

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

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

<table>
<thead>
<tr>
<th>Number of Sites (N)</th>
<th>Degree of Concurrency (z)</th>
<th>Total Request Service Time (X)</th>
<th>System Utilization (ρ)</th>
<th>Math. Average Request Response Time</th>
<th>Simulation Average Request Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1.06</td>
<td>117.3124</td>
<td>0.2188</td>
<td>146.9180</td>
<td>138.400 ± 32.72</td>
</tr>
<tr>
<td>5</td>
<td>1.17</td>
<td>189.2927</td>
<td>0.3172</td>
<td>257.9444</td>
<td>264.000 ± 60.50</td>
</tr>
<tr>
<td>8</td>
<td>1.47</td>
<td>326.5279</td>
<td>0.7302</td>
<td>894.0713</td>
<td>826.408 ± 101.50</td>
</tr>
<tr>
<td>10</td>
<td>1.63</td>
<td>541.6230</td>
<td>0.7646</td>
<td>1544.2256</td>
<td>1492.700 ± 102.52</td>
</tr>
<tr>
<td>12</td>
<td>1.68</td>
<td>712.2500</td>
<td>0.8168</td>
<td>2491.4000</td>
<td>2411.222 ± 406.59</td>
</tr>
</tbody>
</table>

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

SIMULATION MODEL

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

![Graph showing comparison between Simulation Model and Mathematical Model for varying numbers of sites]
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:

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

where

\[
\bar{x} = \bar{x}_a + \bar{x}_c \quad \text{and} \quad \lambda = \frac{n \lambda x}{\bar{n}}
\]

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


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 mechanism 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. Local : 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

PATH 1: (1: MESS1, 1: MESS2, ..., 1: MESS15) END

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

PATH 1: (TOLINE; FRLINE) END
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 preemption model were Request, Response, Completion, Rollback, Local, and Atermi-nate. 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
1. OBJECTS

**MSGQUEUE** (Incoming Message Buffer)

- **OPERATIONS**
  - QUEPUT
  - QUEGET

- **DATA**
  - QUEBUFFER

PATH QMAX: (1: (QUEPUT); 1: (QUEGET)) END

**OUTQUEUE** (Outgoing Message Buffer)

- **OPERATIONS**
  - OUTPUTTOUTLET

- **DATA**
  - OUTBUFFER

PATH QMAX: (1: (OUTPUTTOUTLET); 1: (OUTGET)) END

2. PROCESSES

DETECTION ROUTINES

3. PROCESSES

WRITER

STARTUP
(Resource Manager)

READER

4. OPERATION

START CONTROLLER

PATH START CONTROLLER END

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

BUFFER (Outgoing Message Buffer)

OPERATIONS

DATA

BUFPUT
BUFGET

IOBUFFER

PATH BMX: (1: (BUFPUT); 1: (BUFGET)) END

PRBUF (Local Process Buffers)

OPERATIONS

DATA

PBUFFPUT
PBUFGET

PRBUFFER

PATH 1: (PBUFFPUT; PBUFGET) END

2. PROCEDURES

RANDOM NUMBER GENERATORS

3. PROCESSES

READLINE

WRITER

PROCESS 1

PROCESS 2

... 

PROCESSn

4. OPERATION

STARTMACH

PATH START MACH END

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 ALGORITHM *)

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=(REQUEST,RESPONSE,COMPLETION,ROLLBACK,LOCAL,NOTFREE,DETACH,ABORT);
  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;
    ACETYPE:STATUS;
    QUESIZE:INTEGER;
    DEADLOCK:BOOLEAN;
    DPATHS:ARRAY[0..NMAX] OF DSET;
    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 T1,T2,T3 :INTEGER;
BEGIN
  T:=I*10000+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*100000+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,OS :INTEGER);
VAR T,T1,T3:INTEGER;
BEGIN
  DU:=DU DIV 100;
  T:=I*100+J*10;
  T1:=DU*10000+T+7;
  T3:=OS*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
  T100:=10000;
  T100:=I*100;
  T1:=NF*T1000+T100+30;
IF DE<>99 THEN BEGIN
    T2:=DE*T1000+700+31;
    WRITE(T2)
END;

T3:=RES*T1000+700+32;
T4:=ROL*T1000+700+33;
T5:=COM*T1000+700+34;
T6:=ARE*T1000+700+35;
IF K1<>99 THEN BEGIN
    T7:=K1*T1000+700+36;
    T8:=K2*T1000+700+24;
    T9:=K3*T1000+700+25;
    WRITE(T7,T8,T9)
END;

END; (* MESSIS *)

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 QUEUEBUFFERS:ARRAY[1..BMAX] OF MESSAGE:
    INQ,OUTQ:1..BMAX;

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

ENTRY PROCEDURE QUEGET(VAR M:MESSAGE;
VAR Q:INTEGER;
BEGIN
M:=QUEBUFFER[OUTQ];
IF OUTQ>INQ THEN Q:=(BMAX-OUTQ)+INQ ELSE Q:=INQ-OUTQ;
OUTQ:=(OUTQ MOD BMAX) + 1;
END; (* .... *)

INIT; BEGIN
INQ:=1;
OUTQ:=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; (* PRBUFPUT *)

ENTRY PROCEDURE PRBUFGET(VAR M:MESSAGE);
BEGIN
M:=PRBUFFER
END; (* PRBUFGET *)
PRTBLE=RECORD
  RNK:INTEGER;
  TACC:STATUS
END;
MAT=ARRAY[0..MMAX,0..NMAX] OF PRTBLE;
STATE=(BLOCKED,RUNNING);
RESHLD=RECORD
  RNAM:INTEGER;
  RACC:STATUS
END;
PROCS=RECORD
  PNAME:INTEGER;
  PSITE:INTEGER;
  PSTATE:STATE;
  RHELD:ARRAY[0..MMAX] OF RESHELD
END;
RESRC=RECORD
  RNAM:INTEGER;
  RSTATUS:STATUS
END;
VAR
  MOUEUE:MSGOUEUE;
  OGUEUE:OUTOUEUE;
  PBUF:ARRAY[1..23] OF PRBUF;
  IO:PROCIO;
  PROCTAB:ARRAY[0..NMAX] OF PROCS;
  LRESTAB:ARRAY[0..NMAX] 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;
  REOACCESS:STATUS;
  MSGTEMP:MESSAGE;
  TENTRY:(REQ,REL,DETEC);
  TQREQ,IFP,JFP,TOTDEAD,NINITD:INTEGER;
  MYSITE:SITES;
  STK:ARRAY[0..20] OF INTEGER;

(* ********************** PRBUF ********************** *)
(* 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 IIHAX DO
    BEGIN
      PROCTAB[I].RHELDC[J].RNAM:=-1;
      PROCTAB[I].RHELDC[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(P:INTEGER):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
  BEGIN
    I:=I+1;
    IF I>N THEN NEWP:=TRUE ELSE NEWP:=FALSE;
  END;  (* NEWP *)

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

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 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, INTEGER;
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:=ARESPOST;
            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].RNAME<>RR DO I:=I+1;
      LRESTAB[I].RSTATUS:=PROCTAB[J].RHELD[I].RACC;
      MSGTEMP.ACESTYPE:=LRESTAB[I].RSTATUS;
      IF MSGTEMP.MSGDEST=MYSITE THEN
        PBUFMSGTEMP.PROCNAMEJ.PRBUFPUT(MSGTEMP) ELSE
        QUEUE.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[1].RHELD[1].RNAM<>RR) AND (J<=M) DO J:=J+1;
IF J>M THEN WRITELN('**ERRES**');
   BEGIN
      RNAM:=-1; RACC:=SHARED
   END;
   (* REMOVE ENTRY FROM PRTABLE *)
   WITH PRTABLECIFR,1FP1 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[1].RHELD[1].RNAM<>-1 THEN SW:=TRUE;
   IF NOT SW THEN (* REMOVE PP *)
   WITH PROCTAB[1] DO
      BEGIN
         PNAME:=-1; PSTATE:=BLOCKED
      END;
   (* ANY MORE PROCESS USING RR *)
   SW:=FALSE;
   FOR J:=0 TO M DO
      IF PRTABLECIFR,J.RNK=0 THEN SW:=TRUE;
   IF NOT SW THEN (* NO PROCESS *)
      BEGIN
         (* ANY PROCESS WAITING FOR RR *)
         SW:=FALSE;
         FOR J:=0 TO M DO
            IF PRTABLECIFR,J.RNK>0 THEN SW:=TRUE;
         IF NOT SW1 THEN
            LRESTAB[1].RSTATUS:=FREE
         ELSE ALLOCATE[1]
      END;
END; (* RESREL *)

PROCEDURE ROLLBREL;
(* ROLLS BACK A PROCESS *)
VAR J, TRAK : INTEGER;
   SW,SW1 : BOOLEAN;
BEGIN
J:=0;
AND (J<=M) DO J:=J+1;
IF J>M THEN WRITELN('**ERROR**');
BEGIN
RNAM:=-1; RACC:=SHARED
END;
TRANK:=PRTABLE[I.FR].J.FP].RNK;
WITH PRTABLE[I.FR].J.FP] DO
BEGIN
RNK:=-1; TACCES:=FREE
END;
IF TRANK=0 THEN
BEGIN
SW:=FALSE;
FOR J:=0 TO N DO
IF PRTABLE[I.FR].J.FR].RNK=0 THEN
SW:=TRUE;
IF NOT SW THEN
BEGIN
SW1:=FALSE;
FOR J:=0 TO N DO
IF PRTABLE[I.FR].J.FR].RNK>0 THEN
SW1:=TRUE;
IF NOT SW1 THEN
LRESTAB[I.FR].RSTATUS:=FREE
ELSE ALLOCATE
END;
ELSE
BEGIN
SW:=FALSE;
FOR J:=0 TO N DO
IF PRTABLE[I.FR].J.FR].RNK=TRANK
THEN SW:=TRUE;
IF NOT SW THEN
FOR J:=0 TO N DO
IF PRTABLE[I.FR].J.FR].RNK>TRANK THEN
PRTABLE[I.FR].J.FR].RNK:=
PRTABLE[I.FR].J.FR].RNK-1;
END;
END; (* ROLLLBREL *)

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
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 RIH: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[J]:=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:=RESTART[I].RNAME;
    MARKED[I]:=TRUE
  END;
END;  (* VERTICVAL *)
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:=O; K:=0;
FOR I:=O TO NMAX DO PPATHS[I]:=NMAX;
WHILE NOT DONE DO
BEGIN
   HORIZONTAL(RJ,H); SW:=FALSE;
   IF DLCHECK THEN
      FOR I:=O TO H DO
         IF P2E13=PP THEN SW:=TRUE;
   IF SW THEN
      BEGIN
         MSGTEMP.DEADLOCK:=TRUE;
         DONE:=TRUE
      END
   ELSE
      BEGIN
         WHILE H.:=O DO
            BEGIN
               STKPTR:=P2EH7;
               STKPIR:=STKPIR+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=O) AND (NOT V) THEN
            DONE:=TRUE;
      END;
   END;
END; (* HVDETECT *)

PROCEDURE DISSEARCH;
(* SEARCH DISJOINT PATHS *)
LABEL 1;
VAR I,J,K,L,I1,J1,L1,PS,R5:INTEGER;
   DLCHECK,SW:BOOLEAN;
BEGIN
FOR I:=0 TO N DO
    BEGIN
        FOR J:=0 TO N DO
            IF (PRTABLE[I],I.RNK>0) AND (NOT MARKED[I])
                THEN
                    BEGIN
                        PS:=PROCTABLE[I].PNAME;
                        RS:=LRESTAB[CJ].RNAME;
                        DLCHECK:=FALSE; J1:=PS;
                        MARKED[I]:=TRUE;
                        HVDETECT(PS,RS,K,DLCHECK);
                        (* SET DISJOINT PATH *)
                        L:=1;
                        WHILE MSGTEMP.DISJOINTE[L].DIDENT<>-1 DO
                            L:=L+1;
                        MSGTEMP.DISJOINTE[L].DIDENT:=J1;
                        J1:=1;
                        MSGTEMP.DISJOINTE[L].DPROC[S][1]:=PPATHS[1];
                        FOR L1:=2 TO K DO
                            BEGIN
                                PS:=PPATHS[L1]; SU:=FALSE;
                                FOR I1:=1 TO J1 DO
                                    IF PS=MSGTEMP.DISJOINTE[L].DPROC[S][11] THEN
                                        SU:=TRUE;
                                    IF NOT SU THEN
                                        BEGIN
                                            J1:=J1+1;
                                            MSGTEMP.DISJOINTE[L].DPROC[S][J1]:=PS
                                        END;
                                    END;
                            END;
                        MSGTEMP.DISJOINTE[L].DPROC[S][J]:=J1;
                        GOTO 1
                    END;
                END; (* DISSEARCH *)

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.DISJOINTE[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[J]:=J;
L:=FINDR(RR);
FOR I:=0 TO N DO
IF (PRTABLE[I].ILRNK<0) AND (NOT MARKED[I])
THEN MARKED[I]:=TRUE;
DISEARCH;
1:
END; (* HVINIT *)

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

PROCEDURE INSERTPATH(P: INTEGER);
VAR DUPLICATE: BOOLEAN;
DL: 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 LRESTABIII.RSTATUS<>FREE THEN
DLCHECK:=TRUE;
ELSE IF NOT DLCHECK THEN GOTO 1;
PP:=MSGTEMP.PROCNAME;
FOR I:=0 TO N DO
MARKED[II]:=FALSE;
K:=0; DL:=0; LAST:=MSGTEMP.DPATHS[0];
FOR I:=1 TO LAST DO
BEGIN
K:=K+1;
PTEMP[K]:=MSGTEMP.DPATHS[II]
END;
FOR I:=1 TO K DO
BEGIN
PS:=PTEMP[I];
IF NEUP(PS) THEN
BEGIN
V:=FALSE;
VERTICAL(PS,RS,V);
END;
IF NOT V THEN INSERTPATH(PS) ELSE
BEGIN
DL:=TRUE;
HVDETECT(PS,RS,L,DLCHECK);
END;
END;
END;
MSGTEMP.DPATHS[0]:=DL;
DISEARCH;
(* UNION OF SUPPATHS *)
K:=DL; DL:=0;
FOR I:=1 TO K DO
PTEMP[II]:=MSGTEMP.DPATHS[II];
 FOR I:=1 TO K DO
BEGIN
PS:=PTEMP[II];
FOR J:=1 TO N DO
IF PS=MSGTEMP.DISJOINTEJLDIDENT THEN
BEGIN
L:=MSGTEMP.DISJOINTEJLDPROCS[0];
FOR LAST:=1 TO L DO
BEGIN
PS:=MSGTEMP.DISJOINTEJLDPROCS[LAST];
IF PS=PP THEN
BEGIN
MSGTEMP.DEADLOCK:=TRUE;
GOTO 1
END;
INSERTPATH(PS)
END;
MSGTEMP.DISJOINT[CJ].DIDENT:=−1;
MSGTEMP.DISJOINT[CJ].DPROCSC0]:=-1
END ELSE
INSERTPATH(PS);
END;
MSGTEMP.DPATHS[C0]:=DL;
1:
IF MSGTEMP.DEADLOCK THEN
MSGTEMP.MSGDEST:=MSGTEMP.MSGORIGIN ELSE
MSGTEMP.MSGDEST:=(MYSITE MOD NITES) +1;
OQUEUE.OUTPUTT(MSGTEMP);
END; ( * DETECTCONT *)

************ END OF HAY ALGORITHM ************

PROCEDURE SENRESPONSE;
( * SENDS OUT RESPONSE TO REQUESTING PROCESSES *)
BEGIN
IFR:=FINDR(RR); JFP:=FINDP(PP);
LRESTAB(IFR1.RSTATUS:=REQACCESS;
PRTABLE(IFR,JFP1.RNK:=0;
PRTABLE(IFR,JFP1.TACCESS:=REQACCESS;
PRCTAB[JFP1.PSTATE:=RUNNING;
WITH MSGTEMP DO
BEGIN
MSGTYPE:=ARESPONSE;
MSGDEST:=MSGORIGIN;
MSGORIGIN:=MYSITE;
DPATHSC0]:=-1
END;
IF MSGTEMP.MSGDEST=MYSITE THEN
PBUF(MSGTEMP.PROCNAME).PRBUFPUT(MSGTEMP) ELSE
OQUEUE.OUTPUTT(MSGTEMP);
(* IO.MESS9(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[C0]:=-1
END;
IF MSGTEMP.MSGDEST=MYSITE THEN
PBUF(MSGTEMP.PROCNAME).PRBUFPUT(MSGTEMP) ELSE
OQUEUE.OUTPUTT(MSGTEMP);
PROCEDURE RANK;
 (* ASSIGN 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[I,F,RNKB,K] THEN K:=PRTABLE[I,F,RNKB,K];
   WITH PRTABLE[I,F,J] DO
   BEGIN
    RNKB:=K+1;
    TACCES:=REQACCESS
   END;
 END;  (* RANK *)

PROCEDURE RERANK(THELDO:STATUS);
 (* RESOURCE RR IS BEING HELD THELDO *)
 (* 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 J<>F THEN
      WITH PRTABLE[I,F,RNKB,J] DO
      BEGIN
       IF (RNKB>0) AND (TACCES=EXCLUSIVE) THEN
        WAITEXCL:=TRUE;
       IF (RNKB>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[I,F,J] DO
       IF (RNKB>0) AND (TACCES=SHARED) AND (I<>J) THEN
        PRTABLE[I,F,J,RNKB]:=RNKB;
     END;
   END;
 END;  (* RERANK *)

PROCEDURE RESFREE(VAR RFREE:BOOLEAN; VAR THELDO: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],ILRNK:0) AND (PRTABLE[IFR],11.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;
MGTYPE:=DETOK;
MSGORIGIN:=MYSITE;
MSDEST:=(MYSITE MOD NSITES)+1
END;
QUEUE.OUTPUTT(MSGTEMP)
END ELSE SENDROLLS
END;
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
          QUEUE.OUTPUTT(MSGTEMP);
          (* IO.MESS11(MSGTEMP.MSGDEST,MSGTEMP.PROCNAME,RR); *)
        END;
      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
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*399) MOD 32767;
  S:=ISEED;
  RANDOM:=S/32767.0
END; (* RANDOM *)

PROCESS WRITER(OUTLINE:LINE; SITE,TOTMAXP: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
      QUEUE.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[1].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
      PBUFIMESG.PROCNAMELPRBUFPUT(MESG) ELSE
      OQUEUE.OUTPUTT(MESG)
    END;
   COMPLETION,DETEK:
    IF MESG.MSGDEST=SITE THEN
     MQUEUE.QUEPUT(MESG) ELSE
      OQUEUE.OUTPUTT(MESG);
  END;
END;
ATERMINATE:
BEGIN
RTOTL:=RTOTL+1;
IF MESG.MSGORIGIN<>SITE THEN QUEUE.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,OSIZE,TOTLOC : INTEGER;
KERNELLING,SW :BOOLEAN;
BEGIN
KERNELLING:=TRUE;
KTOTL:=0; TOTLOC:=0;
WHILE KERNELLING DO
BEGIN
QUEUE.QUEGET(MSGTEMP,OSIZE);
CASE MSGTEMP.MSGTYPE OF
ATERMINATE:
BEGIN
KTOTL:=KTOTL+1;
QUEUE.OUTPUTT(MSGTEMP)
END;
LOCALL:
BEGIN
MSGTEMP.QUESIZE:=OSIZE;
MSGTEMP.MSGTYPE:=AREQUEST;
SW:=FALSE;
FOR I:=0 TO M DO
IF MSGTEMP.RESNAME=LRESTABL1.RNAME THEN
SW:=TRUE;
IF NOT SW THEN
QUEUE.OUTPUTT(MSGTEMP) ELSE
BEGIN
WITH MSGTEMP DO
BEGIN
PP:=MSGORIGIN*1000+PROCNAME;
RR:=RESNAME;
REDAccess:=ACETYPE
ENTRY:=REQ;
TOTLOC:=TOTLOC+1;
MANAGER
END
END;
END;
DETEK:
BEGIN
TENTRY := DETEC;
MANAGER
END;

AREQUEST, COMPLETION:
BEGIN
WITH MSGTEMP DO
BEGIN
OUESIZE := OUESIZE + OSIZE;
PP := MSGORIGIN + 100 + PROCNAME;
IF RESNAME = -1 THEN RR := -1
ELSE RR := RESNAME;
REQACCESS := ACESS;
IF MSGTYPE = AREQUEST THEN
TENTRY := REQ ELSE TENTRY := REL
END;
MANAGER
END;

PROCESS PPROC(SITE, LPROCID, TOTMAXR: INTEGER; LAMDA, MU: REAL;
MAXREQ, WACCS, 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;
UMRES, RR, MP, I, J, TOTSENT, TOTDELAY, RELPTR, REQPTR : INTEGER;
OUTREQ, THRUFORE, THRAFTER : INTEGER;
TESTCASE, TS, TD, MPUP : INTEGER;
MINSW, SW, SW1, GREATE, 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 (REOPTR=0) OR (OUTREQ=0) THEN
SW:=TRUE ELSE
BEGIN
SW1:=FALSE;
J:=(RELPR MOD 10)+1;
FOR I:=1 TO OUTREQ DO
BEGIN
IF RESRCES[JI].LRNAME=RR THEN SW1:=TRUE;
J:=(J MOD 10)+1
END;
IF NOT SW1 THEN SW:=TRUE
END END;
(* TYPE OF ACCESS *)
IF WACCESS=1 THEN ACCTYPE:=EXCLUSIVE ELSE
BEGIN
TEMP:=RAND(SEEDR);
IF TEMP>=0.5 THEN ACCTYPE:=EXCLUSIVE ELSE
ACCTYPE:=SHARED
END;
REOPTR:=(REOPTR MOD 10)+1; OUTREQ:=OUTREQ+1;
RESRCES[REOPTR].LRNAME:=RR;
RESRCES[REOPTR].TACCESS:=ACCTYPE;
(* SEND REQUEST *)
WITH MYMSG DO
BEGIN
MSGORIGIN:=SITE; PROCNAME:=LPROCID;
QUESIZE:=0; DPATHS[0]:=-1;
MSGTYPE:=LOCAL; RESNAME:=RR;
ACETYPE:=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[REOPTR].LOCATION:=MYMSG.MSGORIGIN;
REOPTR:=REOPTR-1;
IF (REOPTR=0) OR (REOPTR=-1) THEN REOPTR:=10;
OUTREQ:=OUTREQ-1;
WP:=MYMSG.MSGORIGIN; AGAIN:=TRUE
END ELSE
BEGIN
IF MYMSG.MSGTYPE=NOTFREE THEN
PROCEDURE ASSREL;
BEGIN
RELPTR:=(RELPTR MOD 10)+1; OUTREQ:=OUTREQ-1;
WITH MYMSG DO
BEGIN
PROCNAME:=LPROCID; MSGTYPE:=COMPLETION;
MSGORIGIN:=SITE;
MSDEST:=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;
THRUFORE:=TIME;
(* RELPTR POINTS TO THE LAST RESOURCE RELEASED
RELPTR POINTS TO THE LAST RESOURCE REQUESTED FOR *)
1: RELPTR:=0; REQPTR:=0; GREATR:=FALSE;
OUTREQ:=0; AGAIN:=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;
END;
IF (TOTSENT>=MAXREQ) THEN
  BEGIN
    AGAIN:=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
        QUEUE.QUEPUT(MYMSG) ELSE
        QUEUE.OUTPUT(MYMSG);
      (* IO.MESS6(SITE,LPROCID,RESRCE[RELPTH].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:=TREQUEST;
    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
          QUEUE.QUEPUT(MYMSG) ELSE
          QUEUE.OUTPUT(MYMSG);
        (* IO.MESS6(SITE,LPROCID,RESRCE[RELPTH].LRNAME) *)
      END;
      MAINSW:=FALSE;
    IF (TOTSENT>=MAXREQ) THEN
      BEGIN
        AGAIN:=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 QUEUE.QUEUEPUT(MYMSG) ELSE 
QUEUE.QUEUEPUT(MYMSG); 
(* ID.MESSG(SITE,LPROCID,RESRCRELPTRL.RNAME); *) 
LAMBDA:=TREQUEST-TRELEASE; 
(* GENERATE TIME OF NEXT RELEASE *) 
MUUBAR:=(-1.0/MU)*LN(RANDOM(SEEDR)); 
TRELEASE:=CLOCK+MUUBAR; 
IF (TOTSENT:=MAXREQ) THEN 
BEGIN 
AGAIN:=FALSE; MP:=1; PROCESING:=FALSE; 
GOTO 2 
END 
END; (* TRELEASE < TREQUEST *) 
END; (* CASE *) 
END; (* MAINSW *) 
END; (* PROCESING *) 
2: IF NOT AGAIN THEN 
THRUFTER:=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.RENAME:=1; 
IF MYMSG.MSGDEST=SITE THEN 
QUEUE.QUEUEPUT(MYMSG) ELSE 
QUEUE.QUEUEPUT(MYMSG) 
END; 
END; 
END; 
IF AGAIN 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;
MESSAGE.QUEUEPUT(MYMSG);
IO.MESS7(SITE,LPROCID,TOTSEN,THRUAFTER);
END; (* PROCESS PROCESS *)

ENTRY PROCEDURE STARTMACH(SITE:SITES;INLINE,OUTLINE:LINE;
MAXR,STARTR,TOMAXR,MAXP,TOTMAXP:INTEGER;
LAMDA,MU: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 LRESTABCIJ DO
BEGIN
RNAME:=J; RSTATUS:=FREE
END;
J:=J+1
END;
(* START PROCESSES AT THIS SITE *)
FOR I:=1 TO MAXP DO
PPROCESS(SITE,I,TOMAXR,LAMDA,MU,
MAXREQ,WACC,THRUP);
KERNEL(SITE,MAXR,TOTMAXP);
READER(INLINE,SITE,TOTAAXP);
WRITER(OUTLINE,SITE,TOTMAXP);
END; (* STARTMACH *)

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

(********************************************************************************************)
(* (*)
(* SYSTEM ACTIVATION *)
(* (*)
(********************************************************************************************)

VAR
NET:ARRAY[SITES] OF MACHINE;
LINK:ARRAY[INLINES] OF LINE;
TOTMAXP,TOTMAXR,I,K,MAXREQ,MAXR,MAXP:INTEGER;
(* 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('DISTRIBUTED 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[1],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
PRCGRAm SOLOMINPUT,OUTPUT:

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

CONST:

SITES=3; (* 3 SITE NETWORK *)
MAX=12; (* BUFFER SIZE *)
NMAX=4; (* MAXIMUM NUMBER OF PROCESSES TO RUN ON EACH NODE *)
MAXRES=4; (* MAXIMUM NUMBER OF RESOURCES AT EACH SITE *)

N=7; (* MAX PROCESSES IN NETWORK *)
M=6; (* MAX RESOURCES IN NETWORK *)

TYPE:

MESTYPE=(REQUEST, RESPONSE, COMPLETION, ROLLBACK, LOCAL, INITIAD, LOCK, NFREE, NOTFREE, DETEK, TERMINATE);

(* LOCK MSG IS SENT TO THE SITE THAT INITIATED DETECTION IF DEADLOCK IS DETECTED ALONG THE WAY BY ANOTHER SITE *)

* FREE MSG IS TO BE SENT IF NO DEADLOCK IS DETECTED (*)

* INITIAD MSG IS SENT TO THE SITE THE REQUEST RESIDES TO INITIATE DETECTION ALG *)

SITES=1, SITES; 

STATU.S=(FREE, EXCLUSIVE, SHARED);

NLINES=1, NLINES;

OBPNAME=RECORO;

OBPNAM adjusts INTEGER (* RES NAME *)

OBPNAME=INTEGER (* LOCATION OF REQUESTING PROCESS *)

OBPNAME=INTEGER (* LOCATION OF REQUESTING PROCESS *)

OBPNAME=ARRAY(1..N) OF INTEGER

END;

MESSAGE=RECORO;

MSGTYPE=MESTYPE;

MSGORIGIN=INTEGER;

MSGDEST=INTEGER;

PROCNAME=INTEGER;

RESNAME=INTEGER;

ADSTYPE=STATUS;

ADSTYPE=STATUS;

OBPNAME=INTEGER;

OBPNAME=INTEGER;

EN0;

PROCID=INTEGER;

PATH: 111(MESS1), 1111(MESS2), 1111(MESS3), 1111(MESS4),

1111(MESS5), 1111(MESS6), 1111(MESS7), 1111(MESS8), 1111(MESS9),

1111(MESS10), 11111(MESS11), 111111(MESS12), 1111111(MESS13), 11111111(MESS14), 111111111(MESS15) EN0;

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; L0 STATUS):
VAR T INTEGER;
BEGIN
T := I * 10000 + J * 1000 + K * 100 + 1;
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 * 10000 + J * 1000 + K * 100 + 1;
WRITELN(T);
END;
(* MESS4 *)

ENTRY PROCEDURE MESS5(I, J, K INTEGER):
VAR T INTEGER;
BEGIN
T := I * 10000 + J * 1000 + K * 100 + 1;
WRITELN(T);
END;
(* MESS5 *)

ENTRY PROCEDURE MESS6(I, J, K, L INTEGER):
VAR T, T1, T2, T3 INTEGER;
BEGIN
T1 := I * 10000 + J * 100 + L * 100 + 1;
T3 := L DIV 100;
T2 := T3 * 10000 + I * 100 + J * 10 + 4;
WRITELN(T1 + T2 + T3);
END;
(* MESS6 *)

ENTRY PROCEDURE MESS7(I, J, K, L INTEGER):
VAR T, T1, T2, T3 INTEGER;
BEGIN
T1 := I * 10000 + J * 100 + 1;
T3 := L DIV 100;
T2 := T3 * 10000 + I * 100 + J * 10 + 4;
WRITELN(T1 + T2 + T3);
END;
(* MESS7 *)

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

VAR T, T1 : INTEGER;
BEGIN
  T := J * 1000 + 10 + 13;
  T1 := K * 1000 + 130 + 14;
  WRITELN(T, T1);
END; (* MESS *)

ENTRY PROCEDURE MESS9(I, J, K : INTEGER):
VAR T : INTEGER;
BEGIN
  T := I * 10000 + J * 100 + 13;
  WRITELN(T);
END; (* MESS *)

ENTRY PROCEDURE MESS10(I, J, K : INTEGER):
VAR T : INTEGER;
BEGIN
  T := I * 10000 + J * 1000 + K * 100 + 13;
  WRITELN(T);
END; (* MESS *)

ENTRY PROCEDURE MESS11(I, J, K : INTEGER):
VAR T : INTEGER;
BEGIN
  T := I * 10000 + J * 1000 + K * 100 + 23;
  WRITELN(T);
END; (* MESS *)

ENTRY PROCEDURE MESS12(I, J, K : INTEGER):
VAR T : INTEGER;
BEGIN
  T := I * 10000 + J * 1000 + K * 100 + 23;
  WRITELN(T);
END; (* MESS *)

ENTRY PROCEDURE MESS13(I, J, K : INTEGER):
VAR T : INTEGER;
BEGIN
  T := I * 10000 + J * 1000 + K * 100 + 23;
  WRITELN(T);
END; (* MESS *)

ENTRY PROCEDURE MESS14(I, J, K : INTEGER):
VAR T : INTEGER;
BEGIN
  T := J * 10000 + I * 100 + 23;
  T1 := K * 1000 + I * 100 + 23;
  WRITELN(T, T1);
END; (* MESS *)

VAR: T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11; INTEGER
BEGIN
  T1 = T1 + 1;
  T2 = T2 + 1;
  T3 = T3 + 1;
  IF D1 < 99 THEN BEGIN
    T1 = D1 * T10 + T10 + 31;
    WRITE(T1);
  END;
  T4 = ROL(T100 + T100 + 31);
  T5 = COM(T100 + T100 + 31);
  T6 = ARE(T100 + T100 + 31);
  IF KL < 99 THEN BEGIN
    T7 = KL * T10 + T10 + 36;
    WRITE(T7);
  END;
  WRITE(T1, T3, T4, T5, T6);
END; (* MESSAGE *)

(* ******* PROCIO ******* *)

LINE=OBJECT
ENTRY PROCEDURE TOLINE (VAR MESSAGE);
BEGIN
  MESSAGE = M;
END; (* TOLINE *)
ENTRY PROCEDURE FRLINE (VAR MESSAGE);
BEGIN
  MESSAGE = M;
END; (* FRLINE *)
ENTRY PROCEDURE LINE; (* ******* LINE ******* *)
MACHINE=OBJECT
ENTRY PROCEDURE MACH; ENDE;
ENTRY PROCEDURE QUEUE; (* INPUT MSGES TO BE PROCESSED *)
BEGIN
  MSGQUEUE = OBJECT (* INPUT MESSAGES TO BE PROCESSED *)
  PATH SMA: QUEUE PU: QUEUE;
  VAR QUEUE:ARRAY[1..MAX] OF MESSAGE;
  ENQ, DEQ, MAX;
ENTRY PROCEDURE QUEUE;
QUE3LIF=E0( I hQ01t =m:
IN0(1)=( INQQ m00 3`14X) +1
ENO;

ENTRY PROCEDURE QUEGET(VAR M1ESSAGE;VAR Q;INTEGER);
BEGIN
M1=QUEBUFFER(OOUTQ11)
IF OOUTQ11>INQQ THEN
1S1=(B-AX-OUTQ2)+INQQ ELSE
331=INQC-OUTQ11
OUTQ11=(OUTQ11 MOD 3MAX) +1
END: (* QUEPUT *)

ENTRY PROCEDURE QUEGET(VAR M1ESSAGE;VAR Q;INTEGER);
BEGIN
M1=QUEBUFFER(OOUTQ11)
IF OOUTQ11>INQQ THEN
1S1=(B-AX-OUTQ2)+INQQ ELSE
331=INQC-OUTQ11
OUTQ11=(OUTQ11 MOD 3MAX) +1
END: (* QUEPUT *)

ENTRY PROCEDURE QUEPUT(VAR M1ESSAGE;VAR Q;INTEGER);
BEGIN
M1=QUEBUFFER(OOUTQ11)
OUTQ11=(OUTQ11 MOD 3MAX) +1
END: (* QUEPUT *)

ENTRY PROCEDURE OUTGET(VAR M1ESSAGE;
BEGIN
M1=OUTBUFFER(OOUTQ11)
OUTQ11=(OUTQ11 MOD 3MAX) +1
END: (* OUTGET *)

ENTRY PROCEDURE OUTPUTT(VAR M1ESSAGE;
BEGIN
M1=OUTBUFFER(OOUTQ11)
OUTQ11=(OUTQ11 MOD 3MAX) +1
END: (* OUTPUTT *)

ENTRY PROCEDURE OUTPUTT(VAR M1ESSAGE;
BEGIN
M1=OUTBUFFER(OOUTQ11)
OUTQ11=(OUTQ11 MOD 3MAX) +1
END: (* OUTPUTT *)

ENTRY PROCEDURE OUTPUTT(VAR M1ESSAGE;
BEGIN
M1=OUTBUFFER(OOUTQ11)
OUTQ11=(OUTQ11 MOD 3MAX) +1
END: (* OUTPUTT *)

ENTRY PROCEDURE OUTGET(VAR M1ESSAGE;
BEGIN
M1=OUTBUFFER(OOUTQ11)
OUTQ11=(OUTQ11 MOD 3MAX) +1
END: (* OUTGET *)

ENTRY PROCEDURE PRBUFFPUT(VAR M1ESSAGE;
BEGIN
PRBUFF1=1
END: (* PRBUFFPUT *)

ENTRY PROCEDURE PRBUFFPUT(VAR M1ESSAGE;
BEGIN
PRBUFF1=1
END: (* PRBUFFPUT *)
ENTRY PROCEDURE PRBUFGIT(VAR M1:MESSAGE);
BEGIN
  M1:=PRBUFGIT;
END; (* PRBUFGIT *)

STATE:=BLOKED, RUNNING;
RCURACCESS:=RECORD:
NAME:INTEGER; (* PROC NAME *)
LOCATION:INTEGER; (* LOCATION OF PROCESS *)
CURACCTYPE:STATUS; (* TYPE OF ACCESS *)
RINDEX:=SCOLZIAN
END;

LRESOURCE:=RECORD
NAME:INTEGER; (* RES NAME *)
STATUS:STATUS; (* STATUS OF RES *)
RPROCARRAY[0..N] OF RCURACCESS; (* PROCS CURRENTLY ACCESSING NAME *)
RPROCWAITARRAY[0..N] OF RCURACCESS; (* PROCS WAITING FOR NAME *)
END;

PCURACCESS:=RECORD
PROCNAME:INTEGER; (* RESOURCE NAME *)
LOCATION:INTEGER; (* LOCATION OF RESOURCE *)
CURACCTYPE:STATUS; (* TYPE OF ACCESS *)
END;

PROCESS:=RECORD
NAME:INTEGER; (* PROCESS NAME *)
STATE:STATUS; (* STATE OF PROCESS *)
PRESRCARRAY[0..N] OF PCURACCESS; (* RES CURRENTLY ACCESS BY PNAME *)
PCURACCESS; (* NEW RES IF WAITING *)
END;

VAR
MQUEUES:MSQUEUE;
OQUEUES:OUTQUEUE;
PRBUFARRAY[0..5] OF PBUF; (* MAX 5 PROCESSES PER SITE *)
N1PROC:
PROCTABARRAY[0..MAXOFLPROCESS]
RESTARITABLEARRAY[0..MAXOFLRESOURCE];
PRCTRL:INTEGER;
ALGENTRY:INTEGER;
ACSTATUS:
MGETEM dissertation:
MYTODAYSITES;

(*DETECTION ROUTINES*)
PROCEDURE INITIALIZE
(* INITIALISE LOCAL PROCESS & RESOURCE TABLES *)
VAR
I, J : INTEGER;
BEGIN
FOR I := 0 TO NMAX DO
BEGIN
PROCAB(I).PNAME := -1;
PROCAB(I).PSTATE := BLOCKED;
FOR J := 0 TO N DO
BEGIN
PROCAB(I).PRESCE(J).PCURNAME := -1;
PROCAB(I).PRESCE(J).PCURACCTYPE := FREE;
PROCAB(I).PRESCE(J).POWNER := -1;
END;
PROCAB(I).PNEWREQ.PCURNAME := -1;
PROCAB(I).PNEWREQ.PCURACCTYPE := FREE;
PROCAB(I).PNEWREQ.POWNER := -1;
END;
FOR I := 0 TO NMAX DO
BEGIN
RECIAB(I).RNAME := -1;
RECIAB(I).RSTATUS := FREE;
FOR J := 0 TO N DO
BEGIN
RECIAB(I).RPROCJ(J).RCURNAME := -1;
RECIAB(I).RPROCJ(J).RCURACCTYPE := FREE;
RECIAB(I).RPROCJ(J).RINDEX := FALSE;
END;
RECIAB(I).RPROCWAIT(J).RCURNAME := -1;
RECIAB(I).RPROCWAIT(J).RCURACCTYPE := FREE;
RECIAB(I).RPROCWAIT(J).RLOCATION := -1;
RECIAB(I).RPROCWAIT(J).RINDEX := FALSE;
END;
END:
TOTREQ1 := 0;
TOTGEO1 := 1;
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 PROCAB(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
LOCALRI=FALSE:
FOR I:=0 TO MMAX DO
  IF RESTA3[I].RNAME=R THEN LOCALRI=TRUE;
END;

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

FUNCTION FINDR(R:INTEGER):INTEGER;
(* RETURNS INDEX TO A RESOURCE TABLE *)
VAR I:INTEGER;
BEGIN
  I:=0;
  WHILE (RESTA3[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;
BEGIN
  I:=FINDR(R); RESFREE:=FALSE;
  IF RESTA3[I].RSTATUS=FREE THEN RESFREE:=TRUE ELSE
    IF RESTA3[I].RSTATUS=SHARED AND (RESTA3[I].REQACCESS=SHARED)
      THEN BEGIN
        SW:=FALSE;
        FOR J:=0 TO N DO
          IF RESTA3[I].RPROCWAIT[J].RCURPNAME<>R THEN SW:=TRUE;
        IF NOT SW THEN RESFREE:=TRUE;
      END;
END;

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

PROCEDURE REMOVEP(P:INTEGER);
BEGIN
  IF P=FINDP(P);
END.
WITH PROCAB[I], PNEWREQ DO
BEGIN
  PCURNAME = -1; POWNER = -1; PCURACCTYPE = FREE
END;

BEGIN
  PROCEDURE SENDRESPONSE;
  (* SENDS RESPONSE TO REQUESTING PROCESS *)
  VAR J: INTEGER;
  BEGIN
    IFI = FINDIR(IF1): RESTART(IF1).STATUS = REGACCESS; J = 0:
    WHILE (RESTART(IF1).PROC(WAIT).CURPNAME <> -1) DO J = J + 1;
    RESTART(IF1).PROC(WAIT).CURPNAME = PP:
    RESTART(IF1).PROC(WAIT).LOCATION = MSGTEMP.MSGORIGIN;
    RESTART(IF1).PROC(WAIT).CURACCTYPE = REGACCESS:
    IF MSGTEMP.MSGORIGIN = MYSITE THEN
      BEGIN
        IFP = FINDIR(IF1).PROCAB[I], PSTATE = RUNNING;
        PROCAB[I], PNEWREQ.PCURNAME = -1;
        J = 0:
        WHILE PROCAB[I], PNEWREQ.PCURNAME <> -1 DO J = J + 1;
      END;
      RESTART(IF1).PROC(WAIT).CURPNAME = AR; POWNER = MYSITE;
      PCURACCTYPE = REGACCESS:
      END;
    BEGIN
      MSGTEMP DO
      BEGIN
        MSGTYPE = ARRESPONSE; MSGDEST = MSGORIGIN;
        MSGORIGIN = MYSITE:
      END:
      IF MSGTEMP.MSGDEST = MYSITE THEN
        BOXED MSGTEMP.PROCNAM.I.PBUFFPUT(MSGTEMP) ELSE
        QUEUE_OUTPUT(MSGTEMP):
      END;
    BEGIN
      PROCEDURE PACK[I][J]: INTEGER:
      VAR X, I: INTEGER;
      BEGIN
        L1 = J:
        FOR K = J + 1 TO N DO
          BEGIN
            WITH RESTART[I] DO
            BEGIN
              RPROCWAIT[I], RCURPNAME = RPROCWAIT[K], RCURPNAME;
              RPROCWAIT[I], RLOCATION = RPROCWAIT[K], RLOCATION;
              RPROCWAIT[I], RCURACCTYPE = RPROCWAIT[K], RCURACCTYPE;
              END;
            L1 = L1 + 1;
          END;
      END;
WITH RESTAB1.I,PROCWAIT(I) DO
BEGIN
  RCPNAME:=I; RLOCATION:=I; RCURACCTYPE:=FREE;
  RINDEX:=FALSE
END;

PROCEDURE UPDATE;
VAR I: INTEGER;
BEGIN
  PPI=HYSITE+1000+MSGTEMP.PROCNAME;
  IF PPI=NOT FINDPP(PPI) THEN
    PROCSTATE:=RUNNING;
    PROCNAME:=MSGTEMP.PROCNAME;
    RCURACCTYPE:=FREE;
    RLOCATION:=I;
    I:=I+1;
    WHILE RESTAB(IFR).PROCSTATE<>1 DO PP=I+1;
    WITH PROCSTATE IF PSTATE=1 THEN
      PROCNAME:=MSGTEMP.PROCNAME;
      PROCSTATE:=FREE;
    END;
END;

PROCEDURE RESREL;
(* RES RR IS RELEASED : ALLOCATE IT TO PROCESS IF ANY WAITING *)
VAR I,J,K,L: INTEGER;
BEGIN
  SW,SW1: BOOLEAN
  RSTATUS:=I;
  IF PP=NOT FINDPP(J,I,R) THEN
    IF RNAME=NOT ON THEN RESTAB(IFR).PROCSTATE:=FREE;
    IF NOT RNAME AND RESTAB(IFR).PROCSTATE<>1 THEN
      SW1:=TRUE;
      J=0;
      WHILE SM DO
        WITH MSGTEMP DO
          BEGIN
            MSGTYPE:=RESPOND;
            MSGORIGIN=HYSITE;
          END;
      END;
    END;
  END;
END;
FUNCTION STILLW1:BOOLEAN;
BEGIN
VAR I : INTEGER;
RESULT : BOOLEAN;
BEGIN
IF IFR=FINOR(IFRI) THEN SW1=FALSE
FOR I=I TO N DO IF (RESTAB(IFRI).PROCWAIT(I).RCURPNAME<>PP) THEN BEGIN
RESULT:=TRUE; END;
IF SW THEN STILLW1:=TRUE ELSE STILLW1:=FALSE
END;
PROCEDURE SENONFREE;
BEGIN
IF STILLW1 THEN BEGIN
END;
WITH MSGTEMP DO
BEGIN
  MSGTYPE := NOTFREE;
  MSGDEST := OBPL.PROCNAME;
  MSGORIGIN := MYSITE;
END;
IF MSGTEMP.MSGDEST = MYSITE THEN
  BUFU(MsgTEMP.PROCNAME).PRBUFPUT(MsgTEMP) ELSE
  QUEUE.OUTPUT(MsgTEMP);
(* ID.MESS11(MsgTEMP.MSGDEST,MSGTEMP,PROCNAME,MSGTEMP,RESNAME) *)
END;

PROCEDURE SERVENF;
(* SEND NFREE MSG *)
BEGIN
  IF MSGTEMP.MSGORIGIN = MYSITE THEN
    BEGIN
      MSGTEMP.MSGTYPE := NFREE;
      MSGDEST := OBPL.PROCNAME;
      QUEUE.OUTPUT(MsgTEMP);
    END;
  END ELSE SENDNFREE;
END;

PROCEDURE SENDROLLB;
(* SENDS ROLL BACK MSG *)
VAR SW:BOOLEAN;
BEGIN
  IF STILLW THEN
    BEGIN
      WITH MSGTEMP DO
      BEGIN
        MSGTYPE := ROLLLBACK;
        MSGDEST := OBPL.PROCNAME;
        MSGORIGIN := MYSITE;
      END;
      IF MSGTEMP.MSGDEST = MYSITE THEN
        BUFU(MsgTEMP.PROCNAME).PRBUFPUT(MsgTEMP) ELSE
        QUEUE.OUTPUT(MsgTEMP);
      (* ID.MESS11(MsgTEMP.MSGDEST,MSGTEMP,PROCNAME,MSGTEMP,RESNAME) *)
      BACKUP(IF, JJ);
      IF LOCALP(PP).THEN REMOVEW(PP)
    END;
  END;

PROCEDURE SERVEOL;
(* DEADLOCK HAS OCCURED AND ACTION HAS TO BE TAKEN *)
BEGIN
  TOTDEAD := TOTDEAD + 1;
  IF MSGTEMP.MSGORIGIN = MYSITE THEN
    BEGIN
      (* SEND BLOCK MSG *)
PROCEDURE GOLCALG:
(* GOLDMAN'S DETECTION ALGORITHM *)

(* SITE INITIATING DETECTION STARTS IN STEP 10, WHILE OTHER
SITE INITIATING DETECTION STARTS IN STEP 1 *)

PROCEDURE STEP10:
(* PX MUST BE LOCAL TO SITE EXPANDING *)
BEGIN
I:=0;
WHILE MSGTEMP.O3PL.OBPLPROCS[I] <> -1 DO I:=I+1;
PX:=MSGTEMP.O3PL.OBPLPROCS[I-1];
I:=FINDP(PX);
P1:=PROCTA3[I].PNEMREQ.PCURNAME;
IF NOT LOCALA(RX).THEN
BEGIN
(* STEP 11 *)
MSGTEMP.O3PL.OBPLNAME:=RX;
QUEUE.OUTPUTT(MSGTEMP);
HALTT:=TRUE
END ELSE HALTT:=FALSE
END (* STEP 10 *)

BEGIN
STKPTR:=C;
IF ALGENTRY THEN
BEGIN (* SITE INITIATING STARTS IN STEP 13 *)
STEP13
IF NOT HALTT THEN GOTO 3 ELSE GOTO 2
END ELSE
BEGIN
(* SITE RECEIVING STARTS AT STEP 1 *)
(* STEP 1 *)
RX:=MSGTEMP.OBPL.OBPLNAME;
IF LOCALA(RX) THEN
BEGIN (* STEP 2 *)
J1:=0;
WHILE MSGTEMP.OBPL.OBPLPROCS[J1] <> -1 DO J1:=J1+1;
P2:=MSGTEMP.OBPL.OBPLPROCS[J1-1];
IF3:=FINDR(RX); SWI:=FALSE;
FOR J1:=3 TO 4 DO
IF RESTAB(IFR).RPROCWAIT[I].RCURPNAME=P2 THEN SWI:=TRUE;

END (Proceeds to Step 10)
IF NOT SW THEN GOTO 2 ELSE GOTO 3
END ELSE
BEGIN (* STEP 4 *)
J1=0:
WHILE MSGTEMP.OBPL.OBPLPROGS[J1]<=-1 DO J1=J1+1:
P2=MSGTEMP.OBPL.OBPLPROGS[J1-1]:
IF P2=FINOP(P2) THEN SW=FALSE:
FOR J1=0 TO M DO IF PROCTABIFP1.PRESRCI1.PCOURNAME=RX THEN SW=TRUE;
IF NOT SW THEN GOTO 2 ELSE (* STEP 9 *)
BEGIN
SERENV1.GOTO 2 END ELSE
BEGIN
STEP15:
IF NOT HALT THEN GOTO 3 ELSE GOTO 2
END
END (* STEP 3 *)
J1: (* FIND PROCESSES CONTROLLING RX *)
KI=1 IFRI=FINOR(RX):
FOR II=0 TO N DO IF RESTABIFR1.RPROC(I).RCURNAME<>-1 THEN
BEGIN
KI=KI+1 P1(KI)=RESTABIFR1.RPROC(I).RCURNAME:
END
WHILE KI=-1 DO
BEGIN
STACK(STKPTR1)=MSGTEMP:
STACK(STKPTR1)=P1(KI):
STKPTR2=STKPTR1+1:
KI=KI+1:
END
21
WHILE STKPTR>0 DO
BEGIN
STKPTR1=STKPTR-11: MSGTEMP=STACK(STKPTR1):
P1=STACK(STKPTR1-1):
(* CHECK IF P1 IS IN OBPL *)
DEADLOCK=FALSE:
FOR LI=0 TO N DO IF P1=MSGTEMP.OBPL.OBPLPROGS[LI].THEN DEADLOCK=TRUE:
IF DEADLOCK THEN SERVSDL (* SEND ODL MSG IF SITE WAS NOT THE INITIATOR OF DETECTION ELSE SENDROLL *) ELSE (* STEP 4 *)
IF NOT LOCALP(RX) THEN BEGIN (* STEP 7 *)
BEGIN
J1=0;
WHILE MSGTEMP.O3PL.O3PLPROCS(J)<=1 DO J1=J1+1;
MSGTEMP.O3PL.O3PLPROCS(J1)=PX1;
MSGTEMP.O3PL.O3PLNAME=RX1;
OQUEUE.OUTPUTMSGTEMP)
END ELSE
BEGIN
(* STEP 5 *)
IP1=FINDP(PX)
END IF
PCMEM;J1=J1+1;
WHILE MSGTEMP.O3PL.O3PLPROCS(J)<=1 DO J1=J1+1;
MSGTEMP.O3PL.O3PLPROCS(J1)=PX1;
STEP112;
IF ADJ|HALT THEN GOTO 3
END ELSE SERVEND
END
GOLALG
PROCEDURE O3PLINIT;
(* CREATES O3PL AND SENDS TO SITE OWNING PP TO START EXPANDING *)
VAR I 1 INTEGER
BEGIN
NINIT1=NINIT1+1;
WITH MSGTEMP DO
BEGIN
O3PL.PROCNO1=MSGORIGIN; MSGORIGIN=MYSITE;
NGTYPE1=NITDAD1; MSGDEST1=O3PL.PROCNO1;
O3PL.O3PNAME1=RR; OBPL.OBPLOWNER=MYSITE1;
END
FOR I1=1 TO I DO
MSGTEMP.O3PL.O3PLPROCS(I1)=I1;
MSGTEMP.O3PL.O3PLPROCS(I1)=PP1;
IF MSGTEMP.MSGDEST=MYSITE THEN
BEGIN
MSGTEMP.MSGTYPE1=DETEK1; ALGENTRY1=11; GOLALG
END ELSE
OQUEUE.OUTPUTMSGTEMP)
END
PROCEDURE RESREQ;
(* PROCESS PP REQUESTS FOR RESOURCE RR LOCATED AT THIS SITE *)
VAR I 1 INTEGER
RFREE BOOLEAN
BEGIN
TOTREQ=TOTREQ+1;
IFR=FINDP(AR1);
(* CHECK IF RESOURCE IS FREE *)
RFREE=RESFREE(RR1);
IF RFREE THEN
(* RESOURCE IS FREE ALLOCATE IT & SEND RESPONSE *)
SENORESPOUSE ELSE
BEGIN
(* UPDATE PROCESSES WAITING TO ACCESS RR *)
J:=0;
WHILE RESTAB(IF1).RPROCWAIT(J).RCURPNAME<>1 DO J:=J+1;
 WITH RESTAB(IF1).RPROCWAIT(J) DO
BEGIN
RCURPNAME=PP1 RCURACCTYPE=REQACCESS;
LOCATION=MSGTEMP.MSGORIGIN;
INDEX=FALSE
END;
IF MSGTEMP.MSGORIGIN=MYSITE THEN
BEGIN
 IFPI=FUNC(IFPI);
 RCURACCTYPE=REQACCESS;
 WITH RCURACCTYPE DO
 BEGIN
 PCURNAME=PR1 PCURACCTYPE=REQACCESS;
 POWNER=MYSITE
 END;
 END;
 OBINIT (* CREATE OBPL *)
END;
END (* RESEQ *).

FUNCTION RANO(VAR SEE:TREAL; MODP:INTEGER):INTEGER:
CONST
P=21474836471
A=168371
VAR ISeed:INTEGER;
BEGIN;
ISEED:=TRUNC(SEE);
ISEED=(A*ISEED) MOD P;
ISEED:=TRUNC(SEE) MOD MODP;
RAN:=ISEED;
END (* RAN *)

FUNCTION RANDOM(VAR SI:REAL):REAL:
VAR ISEE:INTEGER
BEGIN
ISEED:=TRUNC(SI);
ISEED:=(ISEED*899) MOD 327671
PROCESS WRITER(OUTLINE:LINE:SITE,MAXP:INTEGER):
(* WRITE MSG TO OUTPUT LINE *)
VAR MSG:MESSAGE;
WRITING:BOOLEAN;
TOTL, TOTMSGSENT, TOTMAXP:INTEGER;
BEGIN
WRITING:=TRUE;
TOTMSGSENT:=0; TOTMAXP:=MAXP;
TOTL:=0;
WHILE WRITING DO
BEGIN
QUEUE.OUTPUT(M);
OUTLINE.OUTPUT(M);
TOTMSGSENT:=TOTMSGSENT+1;
IF MSG.MSGTYPE=EOM THEN TOTL:=TOTL+1;
IF TOTL=TOTMAXP THEN WRITING:=FALSE;
IO.MESS(S/TZ,TOTMSGSENT);
END;
END; (* WRITER *)

PROCESS READER(INLINE:LINE:MAXP:INTEGER):
(* MONITORS THE LINE FOR ANY INCOMING MESSAGES *)
VAR MSG:MESSAGE;
TOTFREE,AP,RTOTL,TOTMSGRECVD:INTEGER;
TOTDECT,TOTRESP,TOTROLL,TOTCOMPL,TOTAREQ,TOTINIT:INTEGER;
TOTLOCK, TOTINF:INTEGER;
READING:BOOLEAN;
BEGIN
READING:=TRUE;
RTOTL:=0; TOTMSGRECVD:=0;
TOTFREE:=0; TOTDECT:=0; TOTRESP:=0; TOTROLL:=0; TOTCOMPL:=0;
TOTAREQ:=0; TOTINIT:=0; TOTLOCK:=0; TOTINF:=0;
WHILE READING DO
BEGIN
INLINE.OUTPUT(MSG);
TOTMSGRECVD:=TOTMSGRECVD+1;
CASE MSG.MSGTYPE OF
REQUEST:
BEGIN
TOTAREQ:=TOTAREQ+1;
SW:=FALSE;
FOR I:=0 TO MAX DO

IF MSG.RESTYPE=RESTAB[1].RNAME THEN SW:=TRUE;
IF SW THEN QUEUE.QUEUE(  ) ELSE QUEUE.OUTPUT(  )
END

ARSPONSE,INITDEAO,ROLLBACK,LOCK,NFREE,COMPLETION; BEGIN
CASE MSG.RESTYPE OF
  ARESPONSE: TOTRESP:=TOTRESP+1;
  INITDEAO: TOTINIT:=TOTINIT+1;
  ROLLBACK: TOTROLL:=TOTROLL+1;
  LOCK: TOTLOCK:=TOTLOCK+1;
  NFREE: TONTNF:=TONTNF+1;
  COMPLETION: TOTCOMPL:=TOTCOMPL+1
END;

IF MSG.MSDEST=MYSIDET THEN QUEUE.QUEUE(  ) ELSE QUEUE.OUTPUT(  )
END;

DETKEI: BEGIN
  TOTDETKEI:=TOTDETKEI+1;
  P:=MEMS.ORBPL.ORBPLNAME[1]=I=0;
  WHILE MSG.ORBPL.ORBPLPRCS[I]=1 DO I:=I+1;
  P:=MEMS.ORBPL.ORBPLPRCS[I]=1;
  IF LOCALP(P) OR LOCALR(R) THEN QUEUE.QUEUE(  ) ELSE QUEUE.OUTPUT(  )
END;

NONTFREE: BEGIN
  IF MSG.MSDEST=MYSITE THEN
    QUEUE.QUEUE(  ) ELSE QUEUE.OUTPUT(  )
  TONTNF:=TONTNF+1
END;

ATTEMNATEI: BEGIN
  TOTATTEMNATEI:=TOTATTEMNATEI+1;
  IF MSG.RESTYPE=ROLLBACK THEN ID.MESS15(MYSITE,TONTNFREE,
    TOTDETKEI,TOTRESP,TOTROLLB,TOTCOMPL,TOTAREQ,TOTINIT,
    TOTLOCK,TONTNF)
  END;

PROCCESS.KERNEL(SITE  ,SITESIMAX,MAXP:INTEGER):
  (* KERNEL HANDLES THE RESOURCE ALLOCATION AT EACH SITE; 
IT RUNS THE DETECTION ALGORITHM *)
VAR KTOTL,I,TOTMAXP,JOBSIZE,TOTLOG; INTEGER;
KERNELLING, SHIBOOLENI
(* MAXR IS MAXIMUM RESOURCE AT THIS SITE *)
BEGIN
    TJI.MAXR=MAXR; TOPLOC=I; KERNELLING=TRUE; KTOTL=I;
    WHILE KERNELLING DO
        BEGIN
            QUEUE.QUEST(MSGTEM, OSIZE);
            CASE MSGTEM. MSGTYPE OF
                BEGIN
                    KOTL = KOTL + I; QUEUE.OUTPUT(MSGTEM);
                    END;
                LOCAL II
                BEGIN
                    MSGTEM.QUEST = OSIZE;
                    MSGTEM. MSGTYPE = AREQUEST;
                    PPI = MSGTEM. PRONAME;
                    IF NOT LOCALP(PPI) THEN INSERTP(PPI, IFP) ELSE IFP = FINDP(PPI);
                    SWI = FALSE;
                    FOR I = O TO MAX DO
                        IF MSGTEM. RESNAME=RESTART[I].NAME THEN SWI=TRUE;
                    IF SW THEN BEGIN
                        PIP = MSGTEM. RESNAME;
                        MSGORIGIN = MSGTEM. PRONAME;
                        IF NOT SW THEN BEGIN
                            MSGMR.QUEST = OSIZE;
                            MSGMR. MSGTYPE = AREQUEST;
                            TOTLOC = TOTLOC + I;
                        END;
                        RESREQ;
                    END;
                    AREQUEST;
                    BEGIN
                        WITH MSGTEM DO
                            BEGIN
                                OSIZE = MSGORIGIN + OSIZE;
                                PPI = MSGMR. PRONAME;
                                RRI = MSGMR. RESNAME;
                                RAGACCESS = MSGMR. MSGTYPE;
                            END;
                        END;
                        RESREQ;
                    END;
                    DLOCK1. SENDROLL3;
                    NFREF1. SENDNFR3;
                    ARESPONSE1. UPTABLE;
                    DETEK1. BEGIN
                        ALGENTRY1 = 21;
                    END;
                    INITDEI;
                    BEGIN
                        MSGTEM. MSGTYPE = DETEK1;
                    END;
END.
HsGT:4P.HSGOEST:=ASGTEmP.MSGORIGIN:=

4LGENTRY:=1:

GOLOALG-ENO:
BEGIN
P;:mYSITE111J00+NSGTEMP.PROGNANZ;
l',F2:=F/SCP(PP):
PROCTABE:Fol.PNEWREQ.PCURNAmr4=-1;
P3uFEISGTE4P.PROCNAMEl.PP3UFPUT(mSGTE4P)
E404
COMPLET/ONI
BZGI4-
PP:=MSGTE4P.POISGORIGIN4,11100+MSSTEHP.PROCNAmE;
RR=mSGTEMP.RESNAME;
F LOCALP(DP) THEN
IFP:=FINCP(PP):
/1=3;
PROGTABEIFPI.PRESRGECII.PCURNAME(RR 03
I;=I+1.;
NITWPROCTABCIFP).PRESRGEEIl
G:GIN
PC;JANtolE3-7,1;.-- PR OWNE
= 1;-
PC URA CC T YP
FRE:
ENO:
ENO ;
IF MSGTEHP.MSG0EST=MYSITE THEN ;ESREL ELSE
031.1EjE.OUTPUTT(ISGTEMP)

204

IF 4TOTL=TCTIAAP-THEN.KERNELLINGJ=FALSE:
ENO:
WHILE
I0.mESS8tmf-SITE.TOTOEAO,TOTRE01:---
I0.HESS14(HYSITE,TOTLOC,NINITO)

LABEL 1,2;
LRES=RE03R0
LNAME=INTEGER
ACCESS;STATUS;
LOCATIONINTEGER

VAR
RESRCESIARRAY[1..5] OF LRES;
CLOCK,RELEASE,FREQUENT,LANDABAR,MUUBAR,SEEDR,SEEDREAL;
TEMPT2.IBEFORE.TOTSECS.*;XX1REAL;
NUMRES,RR,MR,NN,TOTESENT,TOTDELAY,RELPTR,REQPTR INTEGER;
LPROCID,OUTREQ,THU;BEFORE,THRAFTER 1 INTEGER;
TESTCASE,TS,ID,MP,ISS,100,MP,1 INTEGER;
MINSW,SW,SH1,GREATR,PROCESING,AGAIN !BOOLEAN:
MYMSGMESSAGE:
ACCTYPE1=STATUS:

procedure GENERATE:
label 3:
begin (* generate new resource *)
sw1=false;
while not sw do
begin
ri=rando(SEQE+1)MOD+1;
if (resptr=0) or (outreq=0) then
sw1=true else
begin
sw1=false;
j1=(resptr+1)MOD+1;
for i=1 to outreq do
begin
if resources[j1].lrname=rr then sw=true;
j1=(j+1)MOD+1
end
if not sw then sw1=true
end
end
(* type of access *)
if access=1 then acctype1=exclusive else
begin
temp=rando(SEQE);
if temp<=Threshold then acctype1=exclusive else
acctype1=shared
end
resptr=rando(resptr+1)MOD; outreq=outreq+1;
resources[resptr].lrname=rr;
resources[resptr].access1=acctype1
(* send request *)
with mymsg do
begin
msgorigin=site; procname1=procid1;
quesize=3;
msgtype=local; resname=rr;
acctype1=acctype
end
io.mess31=site+procid+rr,acctype1;
queue.queueput(mymsg1);
j1=time; ttotal=ttotal+1;
(* tbefore=sini(x+1) *)
queue proclamation queueget(mymsg1);
(* process blocked waiting for response *)
(* temp1=sini(x+1)+before *)
(* tsi=trunc(temp1) *)
t1=time1; mpp1=mymsg1.quesize1;
tsi=tsi;
tdo=tdo; mpp=mpp1;
IF 'MSGS.MSGTYPE'=NOTFREE THEN
BEGIN
(* TSS=TS *)
T00=T00; MPP1=MPP1;
PBUF(LPROCID).PRBUFGET(MMSG); (* TEMP1=SN(XXX)-TIME*)
T01=TIME-1;
(* TS=TRUNC(TEMP))
MPP1=MMSG.QUEUE1;
GOTO 3
END;
IF 'MSGS.MSGTYPE'=ROLLBACK THEN
BEGIN
ID 'MSGS SITE .LPROCID ,TO MPP1;
(* 0 'MSGS SITE LPROCID ,RNAME *)
RESRCS(REOPTR).LOCATION=MSGS.MSORIGIN;
REOPTR=REOPTR-1;
IF (REOPTR=0) OR (REOPTR=1) THEN REOPTR=1;
OUTREQ=OUTREQ-1;
MPP1=MSGS.MSORIGIN; AGAIN=TRUE
END;
END (* SENREQ *)
PROCEDURE ASSREL;
BEGIN
RELPRN=(RELPTR MOD 5)+1; OUTREQ=OUTREQ-1;
WITH MMSG DO
BEGIN
PROCNAME= LPROCID; MSGTYPE=COMPLETION;
MSORIGIN=RNAME;
MSDEST=RRESRCS(RELPR1).LOCATION;
RNAME=RESRCS(RELPR1).LRNAME; AGETYPE=FREE
END
END (* RELPRN *)

BEGIN
LPROCID=PROCNO;
TOSDN=0; TOSDEL=0; XXX=0.01; PROCESSING=True;
TOTSEC=0.01; C0L0K1=0.01; TR00ST=0.01;
SEED1=314159; SITE1; THUBEFORE1=TIME1;
(* RELPRN POINTS TO THE LAST RESOURCE RELEASED
REOPTR POINTS TO THE LAST RESOURCE REQUESTED FOR *)
IF RELPRN=1 THEN GREATER=FALSE;
OUTREQ=0; AGAIN=FALSE;
BEGIN
```plaintext
MP1=-1;
BEGIN
IF MP<>-1 THEN GOTO 21
(* GENERATE TIME OF NEXT REQUEST *)
MUBARI=-(1.0/4UU)*LN(RANDOM(SEED));
RELEASE=CLOCK+MUBARI;
(* GENERATE TIME OF NEXT REQUEST *)
LAMABARI=-(1.3/LAMDA)*LN(RANDOM(SEED));
REQUEST=CLOCK+LAMABARI;
MAINSW=TRUE
WHILE MAINSW DO
BEGIN
IF RELEASE>TREQUEST THEN TESTCASE=11
IF RELEASE=TREQUEST THEN TESTCASE=21
IF RELEASE<TREQUEST THEN TESTCASE=31
CASE TESTCASE OF
11 (* RELEASE REQUEST *)
BEGIN
TEMP=LAMABAR*150.0;
I=TRUNC(TEMP); T2=I+0.49;
IF TEMP>T2 THEN I=I+1;
DELAY[1111.CLOCK]=TREQUEST;
IF (TOTJENT=MAXREQ) THEN
BEGIN
AGAIN=FALSE; MP=-1; PROCESSING=FALSE; GOTO 2
END:
MUBARI=RELEASE-TREQUEST;
(* GENERATE REQUEST *)
IF OUTREQ=TOTAL THEN
BEGIN
(* REQUEST BUT RES HELD EQUALS MAX RES *)
REQUEST=TREQUEST; ASSREL;
QUEUE.QUEPUT(MYMSG);
(* ID, MESS & SITE, PROCID, RESRCES [RELPTR], NAME[111] )
MAINSW=FALSE
END ELSE
BEGIN
MP=-1; GENREQ1;
IF MP<>-1 THEN GOTO 21
(* GENERATE TIME OF NEXT REQUEST *)
LAMABARI=-(1.3/LAMDA)*LN(RANDOM(SEED));
REQUEST=CLOCK+LAMABARI;
MAINSW=TRUE
END:
END (* TESTCASE=11 *)
21 (* TESTCASE=11 *)
BEGIN
CLOCK=REQUEST;
TEMP=LAMABAR*150.0;
I=TRUNC(TEMP); T2=I+0.49;
IF TEMP>T2 THEN I=I+1; DELAY[I];
(* RELEASE RESOURCE IF ANY *)
IF OUTREQ THEN
BEGIN
```

**Note:** The text appears to be a fragment of a larger program or algorithm, possibly related to simulations or computational tasks. The code contains variables, mathematical operations, and conditional statements. The syntax is consistent with a procedural programming language, likely C or a similar language, given the use of specific keywords like `BEGIN`, `END`, `IF`, and `ELSE`. The numbers and values suggest it might be part of a simulation model, possibly for resource management or scheduling. Each section of the code seems to handle different cases or test scenarios, indicating a complex decision-making process within the program.
ASSREL;
*QUEUE,QUEPUT(MYMSG)
(* IO,MES6(SITE,PROCID,RESRCE,RELPT1,LRNAME) *)
END:

MINSW=FALSE:
IF (TOTSENT=MAXREQ) THEN
BEGIN
AGAIN=FALSE; MP1=-1; PROCESSING=FALSE; GOTO 2
END:

EN1:

31 (* TREQUEST<TRElease *)
BEGIN
TEMP1=MUUNBAR*100.0;
I=TRUNC(TEMP); T2I=I+0.49;
IF TEMP>T2 THEN I=I+1; DELAY(I);
BEGIN (* NO RES TO RELEASE *)
CLOCK1=TREQUEST:
TEMP1=(TREQUEST-TREQUEST)*100.01;
I=TRUNC(TEMP): T2I=I+0.49:
IF TEMP>T2 THEN I=I+1; DELAY(I); MAINS=FALSE
END ELSE
BEGIN (* RELEASE RESOURCE *)
CLOCK1=TREQUEST; ASSREL1
*QUEUE,QUEPUT(MYMSG): (* IO,MES6(SITE,PROCID,RESRCE,RELPT1,LRNAME) *)
MUUNBAR1=TREQUEST-TREQUEST:
(* GENERATE TIME NEXT RELEASE *)
MUUNBAR1=(-1.0/MUUNLN(RANDOM(SEDGR))):
BEGIN (* TREQUEST-RELEASE *)
IF (TOTSSENT=MAXREQ) THEN
BEGIN
AGAIN=FALSE; MP1=-1; PROCESSING=FALSE; GOTO 2
END:

ENDIF (* TREQUEST<TREQUEST *)
ENDIF (* CASE *)
ENDIF (* MAINSW *)
ENDIF (* PROCESSING *)

21 IF OUTREN>0 THEN
BEGIN
ASSREL1
QUEUE,QUEPUT(MYMSG)
(* IO,MES6(SITE,PROCID,MYMSG,RESNAME) *)
END:

IF AGAIN THEN
BEGIN
I=100:
J=RAND(100)+rand:
JELLY(i):
IF THROUGH = 1 THEN TOTSSENT=0;
ENTRY PROCEDURE STARTMACH(SITE,SITES;INLINE,OUTLINE,LINE;MAX,SRES, STARTR,MAXP,PROC; INTEGER;LAMDA,MU,MAXREQ,WACC,THRUP 1 INTEGER); VAR I,J 1 INTEGER;
BEGIN
  TOTREQI=0; TOTDEADI=0; NINITO=0; MYSITE=SITE;
  (* INITIALISE RESOURCE TABLE *)
  FOR I:=1 TO SRES DO
    BEGIN
      NAMEI=J; RSTATUS=FREE
    END;
    J:=J+1;
  END;
  (* START PROCESSES AT THIS SITE *)
  FOR I:=1 TO PROC DO
    BEGIN
      PPROCSS(SITE,MAXI,LAMDA,MU,MAXREQ,WACC,THRUP);
      KERNEL(SITE,SRES,MAXP);
      READER(INLINE,MAXP);
      WRITER(OUTLINE,SITE,MAXP);
    END;
  END; (* STARTMACH *)

END: (* ----------- MACHINE ----------- *)

(* ******************************************************* *)
(* | SYSTEM ACTIVATION | *)
(* ******************************************************* *)

VAR
  NETARRAY[SITES] OF MACHINE;
  LINKARRAY[LINES] OF LINE;
  MAXREQ,MAXP,MAX,MACC,THRUP 1 INTEGER;
  (* MAC=0 IF SHARED & EXCL AND 1 IF EXCL ONLY *)
  (* THRUP=0 PERFORMANCE MEASURE ... =1 THRUPUT MEASURE *)
  LAMDA,MU,MU,MU;
  APERSITEARRAY[SITES] OF INTEGER;
  PPERSITEARRAY[SITES] OF INTEGER;
  K,L,J,Y 1 INTEGER;
BEGIN
READ(MAXP, MAXR, LAMDA, MUU, MAXREQ, WACC, THRP); (* DISTRIBUTE RESOURCES AMONG SITES *)

FOR I=1 TO NSITES DO
BEGIN
RPERSITE[I]=L; K=K+1;
PERSITE[I]=P; J=J+1;
END;
I=1;

WHILE K< MAXR DO
BEGIN
I=I+1;
PERSITE[I]=PERSITE[I]+1; K=K+1
END;
I=0;

WHILE J< MAXP DO
BEGIN
I=I+1;
PERSITE[I]=PERSITE[I]+1;
J=J+1;
END;

WRITE(L(1); LDCHAN 1; *ALGORITHM); 1.
WRITE(L(1); N= N++; MAXP; MAXR; MAXR)
WRITE(L(1); MUU = MUU; LAMDA = LAMDA)
WRITE(L(1; MAXIMUM; *REQUEST = MAXREQ)
NEXT[I, START MACH(I), LINK(I), MAXR, PERSITE(I), MAXP, PERSITE(I), LAMDA, MUU, MAXREQ, WACC, THRP]
I=I+1

NEXT[I, START MACH(2), LINK(2), MAXR, PERSITE(2), MAXP, PERSITE(2), LAMDA, MUU, MAXREQ, WACC, THRP]
I=I+1

NEXT[I, START MACH(3), LINK(3), MAXR, PERSITE(3), MAXP, PERSITE(3), LAMDA, MUU, MAXREQ, WACC, THRP]
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 CONSIM(Input, Output): (* Simulation Program for horizontal & vertical deadlocks *) (* 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
SITES=4: (* 4 Sites *)
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
MESS=TYPE(A REQUEST, RESPONSE, COMPLETION, ROLLBACK, TERMINATE, NOT FREE)
SITES=1..NSITES:
RSITES=1..RSITES:
STATUS=FREE, EXCLUSIVE, SHARED:
MESSAGE=RECORD

END:
(* Controller Data Types *)

PROC= RECORD
MARK: INTEGER;
ACCESS STATUS
END:

NPROC=1..NMAX:
MRES=1..MMAX:
MRESARRAY[MRES, NPROC] OF PROCESSES
STATE= (BLOCKED, RUNNING):
RESHEL= RECORD
NAME: INTEGER;
RAC.Status
END:

PROCS= RECUR:\nNAME: INTEGER;
PSITE= INTEGER;
PSSTATE= INTEGER;
RHOLDARRAY[MRES] OF RESHEL
END:

RESRC=RECORD

PROGRAM CONSIM(Input, Output):

(* Simulation Program for horizontal & vertical deadlocks *)
(* 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 *)
(* *)
ENTRY PROCEDURE MESS1(I,J:INTEGER):
VAR K:INTEGER;
BEGIN
  K1=(J*100)+I*10;
  WRITELN(K);
END;
(* MESS1 *)

ENTRY PROCEDURE MESS2(I,J:INTEGER):
VAR K:INTEGER;
BEGIN
  K1=(J*100)+I*100;
  WRITELN(K);
END;
(* MESS2 *)

ENTRY PROCEDURE MESS3(I,J,K:INTEGER; L:STATUS):
VAR T:INTEGER;
BEGIN
  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
  T1:=I*100000*J*10000*K*1000;
  WRITELN(T1);
END;
(* MESS4 *)

ENTRY PROCEDURE MESS5(I,J,K:INTEGER):
VAR T:INTEGER;
BEGIN
  T1:=I*100000*J*10000*K*1000;
  WRITELN(T1);
END;
(* MESS5 *)

ENTRY PROCEDURE MESS6(I,J,K:INTEGER):
VAR T:INTEGER;
BEGIN
  T1:=I*100000*J*10000*K*1000;
  WRITELN(T1);
END;
(* MESS6 *)
ENTRY PROCEDURE MESS7(I,J,K: INTEGER):
VAR T,T1,T2,T3: INTEGER;
BEGIN
T:=I*1000+J*100+11;
T1:=I*1000+J*100+12;
T2:=T*1000+I*100+J*10+8;
END;

ENTRY PROCEDURE MESS3(I,J,K: INTEGER):
VAR T1: INTEGER;
BEGIN
T1:=I*1000+J*100+13;
END;

ENTRY PROCEDURE MESS9(I,J,K: INTEGER):
VAR T1: INTEGER;
BEGIN
T:=I*1000+J*100+14;
END;

ENTRY PROCEDURE MESS11(I,J,K: INTEGER):
VAR T1: INTEGER;
BEGIN
T:=I*1000+J*100+15;
END;

ENTRY PROCEDURE MESS11(I,J,K: INTEGER):
VAR T1: INTEGER;
BEGIN
T:=I*1000+J*100+16;
END;

ENTRY PROCEDURE MESS13(I,J,QS: INTEGER):
VAR T1,T2,T3: INTEGER;
BEGIN
T1:=I*100+J*11;
T1:=I*100+J*12;
T1:=I*100+J*13;
END;

ENTRY PROCEDURE MESS13(I,J,DU,QU: INTEGER):
VAR T1,T2:T3: INTEGER;
BEGIN
DU:=DU DIV 100;
T1:=I*100+J*11;
T1:=I*100+J*12;
T1:=I*100+J*13;
END;

ENTRY PROCEDURE MESS13(I,J,DU,QU: INTEGER):
VAR T1,T2:T3: INTEGER;
BEGIN
DU:=DU DIV 100;
T1:=I*100+J*11;
T1:=I*100+J*12;
T1:=I*100+J*13;
END;
ENTRY PROCEDURE MESS15(I, NF, RES, ROL, COM, ARE, K1, K2, K3: INTEGER);
    VAR T1, T2, T3, T4, T5, T6, T7, T8, T9, T100: INTEGER;
BEGIN
    T1 := NF * T100 + T10 + 30;
    IF J <> 9 THEN BEGIN
        T2 := NF * T100 + T10 + 31;
        WRITE (T2);
    END;
    T3 := RES * T100 + T10 + 32;
    T4 := ROL * T100 + T10 + 33;
    T5 := COM * T100 + T10 + 34;
    T6 := ARE * T100 + T10 + 35;
    IF K1 <> 9 THEN BEGIN
        T7 := K1 * T100 + T10 + 36;
        T8 := K2 * T100 + T10 + 24;
        T9 := K3 * T100 + T10 + 25;
        WRITE (T7, T8, T9);
    END;
    WRITE (T1 + T2 + T3 + T4 + T5 + T6);
END;

ENTRY PROCEDURE FRLINE (VAR MESSAGE: MESSAGE);
    VAR MSGBUF: MESSAGE;
BEGIN
    MSGBUF = MESSAGE;
END;

ENTRY PROCEDURE FRLINE (VAR MESSAGE: MESSAGE);
BEGIN
    MESSAGE = MSGBUF;
END;
CONTROLEROBJECT
(* THE CONTROLLER SIMULATES THE MACHINE RUNNING THE *)
(* DETECTION ALGORITHM. IT HAS 2 BUFFERS: ONE BUFFER IS *)
(* USED TO STORE INCOMING MESSAGES AND THE SECOND TO *)
(* STORE MESSAGES TO BE SENT OUT. THE AGREGATION PROCESS *)
(* MONITORS TRAFFIC ON THE INPUT LINE, READS OFF MESSAGES *)
(* AND STORES IN #QUEBUFFER#. THE #STARTUP# PROCESS RUNS *)
(* THE DETECTION ALGORITHM AND DOES THE RESOURCE ALLOCATION *)
(* IT PUTS THE RESPONSE IN THE #OUTBUFFER# TO BE SENT OUT *)
(* BY THE #WRITER# PROCESS *)
PATH: SITECONTRA.END

TYPE
MSGQUEUE=OBJECT (* INPUT MESSAGES *)
PATH QMAX: (INOUT)(BUFFER)(OUTGET) END;
VAR QUEBUFFER: ARRAY [1..QMAX] OF MESSAGE;
ENO;
VAR INQ, OUTQ: 1..QMAX;
ENTRY PROCEDURE QUEPUT(MESSAGE)
BEGIN
QUEBUFFER[INQ] := M;
INQ := (INQ +1) MOD QMAX +1
END;
ENTRY PROCEDURE QUEGET(VAR M: MESSAGE; VAR Q: INTEGER)
BEGIN
Q := QUEBUFFER[OUTQ];
IF OUTQ = INQ THEN Q := (QMAX + OUTQ + INQ)
ELSE Q := INQ - OUTQ;
OUTQ := (OUTQ + QMAX) + 1
END; (* QUEPUT *)

INIT
BEGIN
INQ := 1;
OUTQ := 1
END; (* INIT *)

END; (* *** MSGQUE *** *)

OUTQUEUE=OBJECT (* MESSAGES TO BE SENT OUT *)
PATH QMAX: (INOUT)(OUTPUT)(OUTGET) END;
VAR OUTBUFFER: ARRAY [1..QMAX] OF MESSAGE;
END;
ENTRY PROCEDURE OUTPUTT(MESSAGE)
BEGIN
OUTBUFFER[INQ] := M;
INQ := (INQ + QMAX) + 1
END; (* OUTPUT *)
ENTRY PROCEDURE OUTGET(VAR MESSAG): 
BEGIN 
MI=OUTBUFFER[OUT]: 
OUT=OUT+1: 
END (* OUTGET *) 

INIT: BEGIN 
INJ=1: 
OUT=1: 
END (* INIT *) 

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

VAR: 
QUEUE:MSGQ:OUTQ: 
ID:PROC: 
PROCESSTRAY[NPRO]:OF PROC: 
RESRCARRAY[NRES]:OF RESRC: 
PRINT:INITI: 
MARKED:ARRAY[NPROC]:OF BOOLEAN: 
DEADLOCK:BOOLEAN: 
NPP,MK:INTEGER: 
P2: ARRAY[NMAX]:OF INTEGER: 
REACCESS:STATUS: 
QستانRESQ:MESSAGE: 
ENTRY:REQUEST: 
TOTREQ,TOQAUD,TSRQ:RECEIVED,TOTMSGSENT,NINIT:INTEGER: 
IFINDR,JFINOP,MAXMSG:INTEGER: 

[*] *********************** 
[*] DETECTION ROUTINES  
[*] *********************** 

PROCEDURE INITIALIZE: 
(* INITIALIZE THE PROCRESS RESOURCE TABLE, THE PROCESS *) 
(* AND RESOURCE TABLES *) 
VAR: INTEGER: 
BEGIN 
FOR II=0 TO NMAX DO 
BEGIN 
PROCESI.PNAME=-l: 
PROCESI.PSTATE=-l: 
PROCESI.PSTATE=BLOCKED: 
FOR J1=0 TO NMAX DO 
BEGIN 
PROCESI.RNAME=https: 
PROCESI.RSTATE=SHARED: 
END:*
EN0;
FOR I = 1 TO MMAX
  RESOURCES(I).NAME := -1
  SOURCES(I).STATUS := FREE

EN0;
FOR I = 1 TO MMAX
  FOR J = 0 TO MAX
    PR_TABLE(I, J).INDEX := -1.
    PR_TABLE(I, J).ACCESS_FLAG := 0;
    TOT_REQ(I, J) := 0;
    TOTAL := 0;
  END;
END;

FUNCTION INITIALIZE(): BOOLEAN
  * INITIALIZE THE REQUESTING PROCESS
  IF NEWP THEN RETURN TRUE
  ELSE BEGIN
    I := 1
    IF NEWP = TRUE THEN RETURN TRUE
    ELSE BEGIN
      WHILE PR_TABLE(I).INDEX = -1 DO
        I := I + 1
      IF NEWP THEN RETURN TRUE
      ELSE RETURN FALSE;
    END;
  END;
END;

FUNCTION FINDYPES(): INTEGER
  * FIND THE PROCESS IN THE TABLES
  IF NEWP THEN RETURN TRUE
  ELSE BEGIN
    I := 1
    IF NEWP = TRUE THEN RETURN TRUE
    ELSE BEGIN
      WHILE PR_TABLE(I).INDEX = -1 DO
        I := I + 1
      IF NEWP THEN RETURN TRUE
      ELSE RETURN FALSE;
    END;
  END;
END;

FUNCTION MAIN(): BOOLEAN
  * MAIN FUNCTION
  IF NEWP THEN RETURN TRUE
  ELSE BEGIN
    I := 1
    IF NEWP = TRUE THEN RETURN TRUE
    ELSE BEGIN
      WHILE PR_TABLE(I).INDEX = -1 DO
        I := I + 1
      IF NEWP THEN RETURN TRUE
      ELSE RETURN FALSE;
    END;
  END;
END;
BEGIN
  I:=0;
  WHILE PROCESI.I.PNAME<>"DO
    I:=I+1;
    FINOP.I:=I;
  END; (* FINOP.OS *)

FUNCTION FINDR(II INTEGER) RESULT I
  (* RETURNS INDEX TO A RESOURCE IN THE RESOURCE TABLE *)
BEGIN
  I:=0;
  WHILE RESOURCESI.RNAME<>R DO
    I:=I+1;
    FINOR.I:=I;
  END; (* FINOR.OS *)

PROCEDURE REMOVE(INC.PI 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 *)
BEGIN
  IF FINO=0 THEN (* PROCESS *)
     BEGIN
       I:=FINOR.BE:
       L:=I+1;
       FOR J=1 TO NMAX DO
       BEGIN
         WITH PROCESI.I DO
         BEGIN
           PNAME=PROCESI.J.PNAME;
           PSITE1=PROCESI.J.PSITE;
           PSTATE1=PROCESI.J.PSTATE
         END;
         FOR K=1 TO NMAX DO
         BEGIN
           PROCESI.J.HELD(K).NAME=PROCESI.J.HELD(K).NAME;
           PROCESI.J.HELD(K).RACC=PROCESI.J.HELD(K).RACC
         END;
         I:=I+1
       END;
       WITH PROCESI.NMAXI 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 MAX DO
BEGIN
RESOURCES[I].RNAME:=RESOURCES(J).RNAME;
RESOURCES[I].RSTATUS:=RESOURCES(J).RSTATUS;
I:=I+1
END;
RESOURCES[MAX].RNAME:=L;
RESOURCES[MAX].RSTATUS:=FREE;
MAX:=1
END; (* REMOVE *)

PROCEDURE REMOVECOL(P, IN 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:=1 TO 4 DO
BEGIN
J:=40;
L:=I+1;
FOR K:=L TO MAX DO
BEGIN
PRTABLE[I,J].RNK:=PRTABLE[I,K].RNK;
PRTABLE[I+1,J].TACES:=PRTABLE[I,K].TACES;
J:=J+1
END;
PRTABLE[I,NMAX].RNK:=1;
PRTABLE[I,NMAX].TACES:=FREE
END;
K:=1;
REMOVE(K,P);
END; (* REMOVECOL *)

PROCEDURE REMOVEROW(R, IN INTEGER); (* REMOVES ROW CORRESPONDING TO RESOURCE R FROM PR TABLE *)
VAR I,J,K,L INTEGER;
BEGIN
FOR I:=1 TO 4 DO
BEGIN
J:=20;
L:=I+1;
FOR K:=1 TO MAX DO
BEGIN
PRTABLE[I,J].RNK:=PRTABLE[K,J].RNK;
PRTABLE[I,K].TACES:=PRTABLE[K,I].TACES;
J:=J+1
END;
PRZ.ALLT[I].TICC=SJ=FREE
END;
PROCEDURE ALLOCATER;
(* ALLOCATES RESOURCES TO WAITING PROCESSES *)
VAR ROW, J: INTEGER;
BEGIN
  ROW := FIND (RR);
  FOR J = 0 TO N DO
    IF PRTABL (ROW, J).RNK < I THEN
      PRTABL (ROW, J).RNK = PRTABL (ROW, J).RNK - 1;
    END;
  END;
  FOR J = 0 TO N DO
    IF PRTABL (ROW, J).RNK = 1 THEN
      BEGIN
        (* ALLOCATE RESOURCES TO PROCESS WITH INDEX J *)
        PROCSESC (J).PSTATE = RUNNING;
        (* SEND RESPONSE MSG *)
        MSGTEMP CO
        BEGIN
          MSGTYPE = RESPONSE; MSGIJ = PROCSESC (J).PSITE;
          PROCJI = PROCSESC (J).PNNAME = PROCSESC (J).PSITE*100
          I := 0;
          WHILE PROCSESC (J).RHEDGLI.RNAM <= RR DO
            I := I + 1;
            RESOURCES (ROW).RSTATE = PROCSESC (J).RHEDGLI.RACC;
            MSGTEMP, ACCESS = RESOURCES (ROW).RSTATE;
            PRTABL (ROW, J).RNK = RESOURCES (ROW).RSTATE;
            (* SEND RESPONSE TO WAKE UP REQUESTING PROCESS *)
            QUEUE_OUTPUT (MSGTEMP);
            (* WRITE MESSAGE *)
            (* TO PROCESSES MSGTEMPS MSGIJ, MSGTEMP, PROCID, RR *)
          END;
        END;
      END;
      (* ALLOCATER *)
    END;
END;
(* ALLOCATER *)
PROCEDURE RESREL;
(* HANDLES RESOURCE RELEASE *)
VAR I, J, I, I: INTEGER
BEGIN
  I := FIND (PP); J := 0;
  WHILE PROCSESC (J).RHEDGJ.RNAM <= RR DO
    I := J + 1;
  END;
END.
FOR K1=L TO NMAX DO
  BEGIN
    PROCSII,RHELJ1.RNAME=PROCSSII,RHELJ1.RNAME;
    PROCSSII,RHELJ1.RNAME=PROCSSII,RHELJ1.RNAME;
    J1=J+1
  END;
  PROCSSII,RHELJ1.RNAME=II;
  PROCSSII,RHELJ1.RNAME=II;
  II=FINR(RR);
  IF PROCSSII.RHELJ1.RNAME=-1 THEN
    REMOVE(P,J);
  ELSE BEGIN
    PR_TABLE[I1,I1,RNK]=II;
    PR_TABLE[I1,I1,TAGGES]=FREE
  END;
  SW1=False;
  FOR J1=0 TO NMAX DO
    IF TRANK>0 THEN
      IF PR_TABLE[I1,J1,RNK]=TRANK THEN SW1=TRUE;
    IF NOT SW1 THEN
      FOR J1=0 TO NMAX DO
        IF (TRANK>0) AND (PR_TABLE[I1,J1,RNK]=TRANK) THEN
          PR_TABLE[I1,J1,RNK]=PR_TABLE[I1,J1,RNK];
          (* ANY MORE PROCES USING RR *)
        SW1=False;
      FOR J1=0 TO NMAX DO
        IF PR_TABLE[I1,J1,RNK]>0 THEN SW1=TRUE;
      IF NOT SW1 THEN
        BEGIN
          SW1=False;
          FOR J1=0 TO NMAX DO
            IF PR_TABLE[I1,J1,RNK]>0 THEN SW1=TRUE;
          IF SW1 THEN ALLOCAIER
            ELSE REMOVE(RR,II);
        END;
      END; (* RESREL *)
  END;

PROCEDURE ROLL1;
(* ABORTS A PROCESS AND ALLOCATES ALL RESOURCES TO OTHER WAITING PROCESSES *)
VAR I,J,K:INTEGER;
BEGIN
  K=1;
  J=0;
  WHILE PROCSSII,FINDOP.RHELJ1.RNAME<>-1 DO
    BEGIN
      K=K+1;
      P2[K]=PROCSSII,FINDOP.RHELJ1,RNAME;
      J=J+1
    END;
  FOR I1=1 TO K DO
    BEGIN
      END;
END;
PROCEDURE HORIZONTAL(VAR R, H: INTEGER);
(* THE HORIZONTAL ALGORITHM *)
(* IT RETURNS IN P2 ALL PROCESSES WITH RANK(j) ZERO ON *)
VAR I, J: INTEGER;
BEGIN
  H := 1;
  I := FINDOR(R);
  FOR J := 0 TO N DO
    IF PTABLE(J)[J].RANK = 0 THEN
      BEGIN
        H := H + 1;
        P2[H][I].NAME := PNAME;
      END;
  END;
END; (* HORIZONTAL *)

PROCEDURE VERTICAL(VAR VP, V: INTEGER; VAR V: BOOLEAN);
(* THE VERTICAL ALGORITHM *)
(* V IS TRUE IF VR EXISTS SUCH THAT VR = RANK(J) *)
VAR J: INTEGER;
BEGIN
  FOR J := 0 TO M DO
    IF PTABLE(J)[J].RANK >= 0 THEN
      BEGIN
        V := TRUE;
        VR := RESOURCES[J].NAME;
        MARKED[J] := TRUE;
      END;
  END;
END; (* VERTICAL *)

PROCEDURE DETECT(VAR PT, A: INTEGER);
(* PERFORMS THE HORIZONTAL AND VERTICAL DEADLOCK DETECTION *)
(* IT USES A STACK TO PERFORM THE ALGORITHM *)
VAR S: ARRAY[0..N] OF INTEGER;
SM, DONE, V: BOOLEAN;
STKPTR, IH, PL: INTEGER;
BEGIN
  (* DETECT *)
  DONE := FALSE;
  STKPTR := 1;
  WHILE NOT DONE DO
    BEGIN
      HORIZONTAL(R, H);
      SM := FALSE;
      FOR J := 1 TO M DO
        IF P2[J] = PP THEN SM := TRUE;
      IF SM = TRUE THEN
        BEGIN
          S[STKPTR] := IH;
          IH := IH + 1;
          STKPTR := STKPTR + 1;
        END;
    END;
  END;
END;
IF SW THEN
BEGIN
DEADLOCK1=TRUE;
DONE1=TRUE
END
ELSE
BEGIN
WHILE H>0 DO
BEGIN
SELSKPTR1=P2;HI;
STKPTR=STKPTR+1;
MTH=1;
END;
V1=FALSE;
WHILE (STKPTR<>S) AND (NOT V) DO
BEGIN
STKPTR=STKPTR-1;
PI=STK (STKPTR);
VERTICAL (PI, AJ, V)
END;
IF (STKPTR=1) AND (NOT V) THEN DONE1=TRUE;
END;
END;(* DETECT *)

PROCEDURE MV1:
(* INITIATES THE DETECTION ALGORITHM *)
VAR PS, PR: INTEGER;
BEGIN (* MV *)
DEADLOCK1=FALSE;
FOR PS=0 TO N DO
MARKED[PS]=FALSE;
PS=PRI;
PS=PR;
DETECT[PS, PR];
END;(* MV *)

PROCEDURE RANK:
(* ASSIGN A RANK TO A REQUESTING PROCESS *)
(* RE-RANK WILL REASSIGN THE RANK IF NECESSARY *)
VAR K, L: INTEGER;
BEGIN
K1=1;
FOR L=0 TO N DO
IF PRABLE(IFIND, L, RANK) THEN
K1=PRABLE(IFIND, L, RANK);
PRABLE(IFIND, J, FINDP1, RANK) =K1+1;
PRABLE(IFIND, J, FINDP1, ACCESS) = REGACCESS;
END; (* RANK *)

PROCEDURE RERANK(HELD, STATUS):
(* RESOURCE RR IS BEING HELD * )
(* REASSIGNS A RANK TO THE NEW REQUEST *)
(* IF THE REQUEST IS FOR SHARED ACCESS *)
VAR WAITEXCL: BOOLEAN; WAITSHARED: BOOLEAN; SW: BOOLEAN;
I, J INTEGER
BEGIN
WAITEXCL = FALSE;
WAITSHARED = FALSE;
IF REDACCESS = SHARED THEN
BEGIN
FOR I = 0 TO N - DO
IF I <> JFINOP THEN
WITH PRTABLE[I] FINDR, II DO
BEGIN
IF (RNK > 0) AND (TAGCES = EXCLUSIVE) THEN
WAITEXCL = TRUE;
ENDIF
SW = FALSE;
IF (THED = EXCLUSIVE) AND (WAITSHARED) THEN
SW = TRUE
ELSE
IF (THED = SHARED) AND (WAITEXCL) THEN SW = TRUE;
ENDIF
IF SW THEN
FOR I = 0 TO N - DO
WITH PRTABLE[I] FINDR, II DO
IF (RNK > 0) AND (TAGCES = SHARED) AND (I <> JFINOP)
THEN PRTABLE[I] FINDR, JFINOP, RNK = RNK;
ENDDO
END; (* RERANK *)

PROCEDURE RESFREE(VAR RFREE: BOOLEAN; VAR THED: STATUS); VAR I: INTEGER; SW: BOOLEAN;
BEGIN
RFREE = FALSE;
THED = RESOURCES[I] FINDR, STATUS;
IF THED = FREE THEN RFREE = TRUE
ELSE IF (THED = SHARED) AND (REDACCESS = 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[I] FINDR, I) AND REDACCESS = EXCLUSIVE THEN SW = TRUE;
ENDIF
IF NOT SW THEN RFREE = TRUE
END;
END; (* RESFREE *)
PROCEDURE RESFEE1;
(* PROCESSES PP REQUEST FOR RR AND DOES THE RESOURCE ALLOCATION *)
VAR i,j:INTEGER;
FREE:BOOLEAN;
THD01:STATUS;
BEGIN
TOTAL1:=TOTAL1+1;
IF 4EPP(P) THEN
BEGIN
  N:=N+1;
  PROCESSE[i].PNAME:=PP;
  PROCESSE[i].PSIZE:=4STEMP.MSGID
END;
IF NOT(FRE) THEN
BEGIN
  M:=M+1;
  RESOURCES[M].RNAME:=RR;
  RESOURCES[M].RSTATUS:=FREE
END;
IF INDEX=INDEX(0) THEN
BEGIN
  N:=N+1;
  PROCESSE[N].RNAME=RR;
  PROCESSE[N].RSTATUS:=FREE
END;
ELSE
BEGIN
  PROCESSE[N].RNAME=RR;
  PROCESSE[N].RSTATUS:=FREE
END;
END
WHERE PROCESSE[JFINDP].RHELO11.RNAME=PP DO
BEGIN
  J:=J+1;
  PROCESSE[JFINDP].RHELO11.RNAME=PP;
  PROCESSE[JFINDP].RHELO11.RACC:=REQACCESS
  RESFREE(FREE,FIELD)
END;
ELSE
BEGIN
  PROCESSE[JFINDP].PSTATE:=LOCKED;
  NINIT0:=NINIT0+1
  RANK:=
  IF NOT DEALOCK THEN
BEGIN
  IF REGACCESS=SHARED THEN RERANK(THD0);
  MSGTEMP.MSGTYPE=NOTFREE;
  QUEUE.OUTPUT(MSGTEMP)
(* ID.MESS11(MSGTEMP,MSGID,MSGTEMP,PROCID,RR) *)
END
ELSE
BEGIN

END
END;
PROCEDURE RESALLOC:
VAR I,J : INTEGER;
BEGIN
IF ENTRY=RE0 THEN RESREQ
ELSE BEGIN
IF ENTRY=RE0 THEN RESREQ
END
(* RESREQ *)
ENDE

PROCESS CRTER(OUTLINE LINE MAX LINE INTEGER);
(* WRITES RESPONSE TO OUTPUT LINE *)
VAR MESSAGE:
BEGIN:
CWRITING = TRUE; TOT=0;
WHILE CWRITING DO:
BEGIN
QUEUE, OUTLINE;
OUTLINE, TOLINE(M);
TOTMSGSENT=1; TOTMSGSENT+1;
IF M=MSGTYPE=TERMINATE THEN TOT=TOT+1;
IF TOT=MAX THEN CWRITING = FALSE
I=MAX; TOTMSGSENT;
(* WRITER *)
END:

PROCESS STARTUP(TOTP INTEGER);
(* RUNS THE RESOURCE ALLOCA TION TO IDENTIFY EACH PROCESS *)
(* THE SITE OF THE PROCESS IS ATTACHED TO THE PROCESSID *)
(* PROCESSES ALL REQUESTS IN QUEUE BUFFER AND WRITES RESPONSE *)
VAR MQ,MTAIL INTEGER; STARTING,SW BOOLEAN;
BEGIN:
INITIALIZE:
STARTING=TRUE; TOT=0;
(* STARTUP *)
SWI = TRUE
WHILE STARTING DO
BEGIN
  QUEUE:QUEGET(MSGTEMP, QSIZE)
  IF MSGTEMP.MSGTYPE = ATERMINATE THEN
    BEGIN
      TOT1 = TOT + 1;
      QUEUE.OUTPUT(MSGTEMP)
      END
    ELSE
      BEGIN
        PPI = MSGTEMP.MSGID + MSGTEMP.PROCID;
        PII = MSGTEMP.REGION;
        REGACCESS = MSGTEMP.ACCESSID;
        IF MSGTEMP.MSGTYPE = AREQUEST THEN ENTRY(REQ);
        ELSE ENTRY(REQ);
        RESULT
        END
      IF TOT = TOTP THEN STARTING = FALSE
      END
    IC. MESSAGE(TOTREAD, TOTREQ);
  IC. MESSAGE(TOTRECV, TOTREQ);
END (* STARTUP *)

PROCESS CREADER(INLINE, LINET; TOTP: INTEGER):
(* MONITORS THE INPUT LINE FOR A MESSAGE. WAITS *)
(* MESSAGE TO THE QUEUE BUFFER TO BE PROCESSED BY STARTUP *)
VAR MESSAGE:
  CREADING: BOOLEAN;
  TOT: INTEGER;
BEGIN
  CREADING = TRUE; TOT = 0;
  WHILE CREADING DO
    BEGIN
      INLINE. FR(Line(M));
      QUEUE. QUEUE(M); TOTMSGRECV = TOTMSCRECV + 1;
      IF MSGTYPE = ATERMINATE THEN TOT1 = TOT + 1;
      IF TOT = TOTP THEN CREADING = FALSE
    END
  IC. MESSAGE(TOTMSGRECV)
END (* CREADER *)

ENTRY PROCEDURE SITCONTR(INLINE, OUTLINE, LINEMAXP: INTEGER):
(* STARTS ALL THE PROCESSES IN CONTROLLER *)
BEGIN
  TOTMSGRECV = 0;
  TOTMSGSENT = 0;
  CREADER(INLINE, MAXP);
  STARTUP(MAXP);
  CWRITER(OUTLINE, MAXP);
END (* SITCONTR *)
ENO: (* ***** CONTROLLER ***** *)

PMACHINE=OBJECT

(* PMACHINE SIMULATES THE MACHINES ON WHICH THE USER PROCESSES *)
(* ARE RUNNING. EACH MACHINE HAS 2 BUFFERS, INCOMING MESSAGES *)
(* ARE READ BY READLINE4. 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 *)
(* PMBUF, ON WHICH IT WAITS FOR A RESPONSE TO ITS REQUEST *)

PATH: SIMMACH ENDI

TYPE

BUFFER=OBJECT

PATH BMAX+1(3BUFPUT):1(3BUFGET) ENDI
VAR I0BUFF+1 ARRAY[1..BMAX] OF MESSAGE;

ENTRY PROCEDURE BUFPUT(MIMESSAGE);
BEGIN
I0BUFFER[INPP]:=1;
INPP=(INPP MOD BMAX) + 1
END; (* BUFPUT *)

ENTRY PROCEDURE BUFGET(VAR MIMESSAGE);
BEGIN
M:=I0BUFFER[OUTP];
OUTP=(OUTP MOD BMAX)+1
END; (* BUFGET *)

INIT BEGIN
INPP=1;
OUTP=1
END; (* INIT *)

END: (* ***** BUFFER ***** *)

PRBUF = OBJECT

PATH 11(PRSUFPUT|PRBUF|PRBUF) ENDI
VAR PRBUFFER: MESSAGE
ENTRY PROCEDURE PRBUFPUT(MIMESSAGE):
BEGIN
PRBUFFER=M
END; (* PRBUFPUT *)

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

END: (* ***** PRBUFFER ***** *)

VAR BUFFERBUFFER:

BUFARRAY[1..7] OF PRBUF (* ASSUME MAX OF 7 PROCESS PER SITE*)
FUNCTION RANDO(VAR SEED:REAL;MOD:INTEGER):INTEGER;

VAR ISSEED:INTEGER;
BEGIN
  ISSEED = TRUNC(SEED);
  SEED = (ISSEED * MOD + P) % MOD;
  RANDO = ISSEED
END (* RANDO *)

FUNCTION RANDOM(VAR ISSEED:REAL;MOD:INTEGER):INTEGER;

VAR ISSEED:INTEGER;
BEGIN
  ISSEED = TRUNC(ISSEED);
  SEED = (ISSEED * MOD) MOD 32767;
  ISSEED = SEED / 32767;
END (* RANDOM *)

PROCESS READLINE(WHO:SITE;INLINE:LINE;MAX:INTEGER);
(* MONITORE INPUT LINE FOR ALL INCOMING MESSAGES IF MESSAGE *)
(* IS FOR PROCESS I RUNNING ON THE LOCAL SITE IT JUST BLOCKS *)
(* PROCESS I TO ACCEPT THE RESPONSE *)
VAR M:MESSAGE;
I,TOT :INTEGER;
TOTRESP,TOTROLLS,TOTCOMPL,TOTAREQ,TOTNFREE,TOTMREC:INTEGER;
READING:BOOLEAN;
BEGIN
  READING=TRUE; TOT=0;
  TOTRESP:=0; TOTROLLS:=0; TOTCOMPL:=0; TOTAREQ:=0;
  TOTNFREE:=0; TOTMREC:=0;
  WHILE READING DO
    BEGIN
      INLINE,FLINE(M);
      IF M.MSGTYPE=TERMINATE THEN
        BEGIN
          TOT=10;
          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;
            REQUEST: TOTAREQ:=TOTAREQ+1;
            NOTFREE: TOTNFREE:=TOTNFREE+1;
            ROLLBACK;
          BEGIN
            END;
        END;
    END;
END;
TOTROLL3! = TOTROLLB + 1;
END;
I04. MESS16 (WHO, TOTREC);
END; (* READLINE *)

PROCESS WRITER (OUTLINE: LINE: TOTP: INTEGER);
(* WRITES MESSAGE TO OUTPUT LINE *)
VAR MESSAGE: RECORD;
TOT: INTEGER;
BEGIN
WRITING = TRUE;
TOT = 0;
WHILE WRITING DO
BEGIN
BUFF, BUFGET();
IF TOT > TOTP THEN WRITING := FALSE
END;
END; (* WRITER *)

PROCESS PROCESSOR (PROCID: INTEGER);
(* SIMULATE A LOCAL PROCESS ACTIVITIES *)
LABEL 1,2:
TYPE
RES = RECORD
LNAME: INTEGER;
TACCESS: INTEGER;
END;
VAR
RES = ARRAY [1..201] OF RES;
CLOCK, RELEASE, REQUEST, LARGABAR, NUUBAR, SEEDR, SEED1: REAL;
TIME1, TBEFORE, TOTBEGIN, XXX: REAL;
NUMRES, RR, MP, I, J, TOSSENT, TOTDELAY, RELPTR, RQPTR: INTEGER;
OUTREQ, TRUBefore, THRUAFTER: 1 INTEGER;
TESTCASE, ITS, TP, MPP, TOTLOC: INTEGER;
MAINSM, S, M, SLGREAT, PROCESSING, AGAN: 1 BOOLEAN;
MYMSG, MESS1: MESSAGE;
AGC, TTYPE: INTEGER;
PROCEDURE GENREQ:
BEGIN (* GENERATE NEW RESOURCE *)
  SWI=FALSE
  WHILE NOT SWI DO
    BEGIN
      R1=RAND(SEED,TOTMAX)+1;
      IF (REGPTRC) OR (OUTREG=1) THEN
        SWI=TRUE ELSE
        BEGIN
          SWI=FALSE;
          J1=(REGPTR 400 20)+1;
          FOR I=1 TO OUTREG DO
            BEGIN
              IF RESCRES[I].LRNAME=RR THEN SWI=TRUE;
              J1=(J 400 20)+1;
            END;
          IF NOT SWI THEN SWI=TRUE
        END;
      (* TYPE OF ACCESS *)
      IF ACCE=1 THEN ACCTYPE=EXCLUSIVE ELSE
        BEGIN
          TEMP=RANO(SEED);*
          IF TEMP>=1.5 THEN ACCTYPE=EXCLUSIVE ELSE
            ACCTYPE=SHARE;
        END;
      RESCRES[REGPTR].LRNAME=RR;
      RESCRES[REGPTR].LACCESS=ACCTYPE;
      IF RR=SITE THEN TOTLOC=TOTLOC+1;
      (* SEND REQUEST *)
      WITH MYMSG DO
        BEGIN
          MSGID=SITE I PROCID=LPORCID;
          QUESIZE=0;
          MSGTYPE=REQUEST; RESID=ARI;
          ACCE=ACCTYPE;
          ELSE
            BEGIN
              IMI.=MR.S(SITE,LPORCID,RR,ACCTYPE);
              J1=TIME; TOTSENT=TOTSENT+1;
              (* TBEFORE=)=SIN(XXX)*)
              PBUF(PCRID).PBUFGET(MYMSG);
            (* PROCESS BLOCKED WAITING FOR RESPONSE *)
            (* TEMP=SIN(XXX) TBEFORE *)
            (* TBEFORE=)=SIN(XXX)*)
            BEGIN
              MPI=MYMSG.MSGID
              IF MYMSG.MSGTYPE=ROLLBACK THEN
            END
          END
        END ELSE
      ELSE
        END
BEGIN
IF MSG_TYPE=NOTFREE THEN PUFF(ProcID, MSG_TYPE FAILURE)
(* DOW:<<S(SITE, ProcID, MSG, RESID) = *)
END (* GENREQ *)
PROCEDURE ASSREQ:
BEGIN
RELTR=(RELTR-MOD-2)+1 OUTREQ=OUTREQ-1
WITH MSG DO
BEGIN
    ProcID=LPORC; MSGTYPE=COMPLETION;
    MSGID=SITE;
    RESID=RESOURCES[RELTR].LRNAME; ACCESSID=FREE
END
END (* RELTR *)
BEGIN
TOTSENT=1; TOTDELAY=0; XXX=5.0; PROCESSING=TRUE;
TOTSECS=0.0;
TOTLOC=0; SEED=1/4/5.1/SITE; SEED=SITE;
RELEASE=0.01; REQUEST=0.01; CLOCK=0.01
THRUFORE=TIME;
(* RELTR POINTS TO THE LAST RESOURCE RELEASED
RELPTR POINTS TO THE LAST RESOURCE REQUESTED FOR *)
11 RELTR=0; OUTREQ=0; RELPTR=0; GREATER=FALSE;
AGAIN1=FALSE; AGAI12=FALSE;
WHILE PROCESSING DO
BEGIN
MP=1;
IF (TOTSENT=MAXREQ) THEN GOTO 2;
GENREQ;
IF MP<>1 THEN
BEGIN
I=1000; J=RAND(SEED,1)*100;
DELAY(J);
IF THRUFORE THEN TOTSENT=0;
GOTO 1
END;
(* GENERATE TIME OF NEXT RELEASE */
MUUBAR=-(1.0/MUU)*LN(RANDOM(SEED));
RELEASE=CLOCK+MUUBARI
(* GENERATE TIME OF NEXT REQUEST *)
LAMDABARI=-(1.0/100)*LN(RANDOM(SEED));
REQUEST=CLOCK+LAMDABARI;
MAIN=TRUE;
WHILE MAIN DO
BEGIN
IF RELEASE=REQUEST THEN TESTCASE=1;
IF RELEASE=REQUEST THEN TESTCASE=2;
IF RELEASE=REQUEST THEN TESTCASE=3;
CASE TESTCASE OF
1 I (* RELEASE=REQUEST *)
BEGIN
  TEMP=LAONOAR*100.0;
  IT=TRUNC(TEMP);  T2I=I+0.49;
  IF TEMP>T2 THEN IT=I+1;
  DELAY(I);  CLOCK=TREQUEST;
  IF (TOSSENT>=MAXREQ) THEN
    GOTO 21;
  MUUBAR=RELEASE-TREQUEST;
  (** GENERATE REQUEST **)
  IF OUTREQUEST=MAX THEN
    BEGIN
      (** REQUEST BUT RES HELD EQUALS MAX RES **)
      RELEASE=TREQUEST;  ASSREL;
      BUFF=BUFFPUT(MYMSG);
      (* IOW,MESSAGE(SITE,PROCID,RESRCE(RELPTR),LNAME)*);
      MAINSW=FALSE;
    END ELSE BEGIN
      #T=I+1;
      IF #T<=1 THEN BEGIN
        I=1001;  #T=101*SEED+1+100;
        DELAY(J);
        IF THRUPUT=1 THEN TOSSENT:=0;
        GOTO 1
      END;
      (** GENERATE TIME OF NEXT REQUEST **)
      LAONOAR=1-LOG(LAONOAR*(1-RAND(SEED)));  TREQUEST=CLOCK+LAONOAR
      MAINSW=TRUE
    END;
  ELSE BEGIN
    (* TESTCASE=1 *)
    BEGIN
      (** RELEASE=REQUEST **)  
      CLOCK=RELEASE;  
      TEMP=LAONOAR*100.0;
      IT=TRUNC(TEMP);  T2I=I+0.49;
      IF TEMP>T2 THEN IT=I+1;  DELAY(I);
      (** RELEASE RESOURCE IF ANY **)  
      IF CUREX=0 THEN BEGIN
        ASSREL;
        BUFF=BUFFPUT(MYMSG);
        /* IOW,MESSAGE(SITE,PROCID,RESRCE(RELPTR),LNAME) */
      END;
      MAINSW=FALSE;
      IF (TOSSENT=MAXREQ) THEN
        GOTO 21;
    END;
  IF TOSSENT=MAX THEN
  BEGIN
    TEMP=MUUBAR*100.0;
    IT=TRUNC(TEMP);  T2I=I+0.49;
    IF TEMP>T2 THEN IT=I+1;  DELAY(I):
IF OJREQ<0 THEN
BEGIN (* NO RES TO RELEASE *)
  CLOCK := TREQUEST;
  TREQUEST := TREQUEST - OJREQ;
END

IF OJREQ<0 THEN
BEGIN (* NO RES TO RELEASE *)
  CLOCK := TREQUEST;
  TREQUEST := TREQUEST - OJREQ;
END

ELSE
BEGIN (* RELEASE RESOURCE *)
  CLOCK := TREQUEST;
  TREQUEST := TREQUEST - OJREQ;
  IF TREQUEST = 0 THEN
  BEGIN
    OR := 1;
    OJREQ := OJREQ + 1;
    GOTO 2;
  END
END

END (* RELEASE < REQUEST *)

END (*) CASE (*)

END (*) MAINSW (*)

ENTRY PROCEDURE STTACH(WHO, SITE, IN, OUTLINE, LINE, MAXP, TOTMAXP, INTEGER, REAL, MAXREQ, WACC, THRP ; INTEGER;

VAR
  I, K, P - INTEGER;
BEGIN
  FOR I = 1 TO MAXP DO
    PROCSS(WHO, I, MAXP, REAL, LAMDA, MUU, MAXREQ, WACC, THRP);
  WRITEP(OUTLINE, IN, TOTMAXP);
END (* STTACH *)

(* *** SYSTEM ACTIVATION *** *)
VAR
   NETIARRAY(RSITES) OF PMACHINE;
   CONTROLLER;
   LINESARRAY(SITE) OF LINE;
   MAXR, MAXP,TOTMAXP, MACC : INTEGER;
   J, Y, THRU P : INTEGER;
   DISTPRIARRAY(RSITES) OF INTEGER;
   LAMDA, MUU REAL;
   MAXREQ, I : INTEGER;
BEGIN
   READ(TOTMAXP, MAXR, LAMDA, MUU, MAXREQ, MACC, THRUP);
   JI = 0;
   YI = TOTMAXP DIV RSITES;
   FOR I = 1 TO RSITES DO
   BEGIN
      DISTPRI[I] = YI;
      JI = JY
   END;
   I = 1;
   WHILE J < TOTMAXP DO
   BEGIN
      = I + 1;
      DISTPRI[I] = DISTPRI[I] + 1;
      JI = J + 1
   END;
   WRITELN("CENTER I is +zed Max!");
   WRITELN("No. of *RESOURCES* = +,MAXP");
   WRITELN("No. of *PROCESSES* = +,TOTMAXP");
   WRITELN("MUU = +,MUU");
   WRITELN("LAMDA = +,LAMDA");
   WRITELN("MAXIMUM *REQUEST = +, MAXREQ");
   NETI1: STIMACH1, LINES1, LINES11, MAXR, DISTPRI1, TOTMAXP, LAMDA, MUU,
       MAXREQ, MACC, THRUP);
   NETI2: STIMACH2, LINES2, LINES21, MAXR, DISTPRI2, TOTMAXP, LAMDA, MUU,
       MAXREQ, MACC, THRUP);
   NETI3: STIMACH3, LINES3, LINES31, MAXR, DISTPRI3, TOTMAXP, LAMDA, MUU,
       MAXREQ, MACC, THRUP);
   CONT, STIMACH, LINES4, LINES41, 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:
SITES=3 (* 3 SITE NETWORK *)
BMAX=10 (* BUFFER SIZE *)
NMAX=10 (* MAXIMUM NUMBER OF PROCESSES *)
MMAX=1 (* MAXIMUM NUMBER OF RESOURCES AT EACH SITE *)
LINES=31

TYPE:
MESSTYPE=(REQUEST,RESPONSE,COMPLETION,ROLLBACK,LOCAL,TERMINATE):
SITES=3...SITES:
STATUS=(FREE,EXCLUSIVE,SHARED):
LINES=3...LINES:
MESSAGE=MESS:
MESS=(REQUEST,REQUEST):
REQUEST=INTEGER:
PROGRAM=INTEGER:
RESPNAME=INTEGER:
AGESTYPE=STATUS:
MESSSIZE=INTEGER:

ENTRY
PROCEDURE MESS1(I,J,INTEGER):
VAR K,INTEGER:
BEGIN
K=(J+10)*11+1;
WRITELN(K);
END (* MESS1 *)

ENTRY PROCEDURE MESS2(I,J,INTEGER):
VAR K,INTEGER:
BEGIN
K=(J+10)*11+1+
WRITELN(K)
END (* MESS2 *)

ENTRY PROCEDURE MESS3(I,J,INTEGER;I,STATUS):
VAR K,INTEGER:
BEGIN
/* The rest of the code */
END

BEGIN
  T:=I*10000+J*1000+K*100+1;
  IF L=EXCLUSIVE THEN T:=T+2 ELSE T:=T+3;
  WRITELN(T);
END;

(* MESS1 *)

ENTRY PROCEDURE MESS4(I,J,K:INTEGER):
BEGIN
  T:=I*10000+J*1000+K*100+4;
  WRITELN(T);
END;

(* MESS4 *)

ENTRY PROCEDURE MESS5(I,J,K:INTEGER):
BEGIN
  T:=I*10000+J*1000+K*100+5;
  WRITELN(T);
END;

(* MESS5 *)

ENTRY PROCEDURE MESS6(I,J,K:INTEGER):
BEGIN
  T:=I*10000+J*1000+K*100+6;
  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*10000+I*1000+J*100+12;
  T3:=L DIV 100;
  T2:=T3*10000+I*100+K*100+13;
  WRITELN(T,T1,T2);
END;

(* MESS7 *)

ENTRY PROCEDURE MESS8(I,J,K:INTEGER):
VAR T,T1:INTEGER;
BEGIN
  T:=J*1000+I*100+14;
  T1:=K*1000+I*100+15;
  WRITELN(T,T1);
END;

(* MESS8 *)

ENTRY PROCEDURE MESS9(I,J,K:INTEGER):
VAR T:INTEGER;
BEGIN
  T:=J*1000+I*100+16;
  WRITELN(T);
END;

(* MESS9 *)

ENTRY PROCEDURE MESS10(I,J,K:INTEGER):
VAR T:INTEGER;
BEGIN
T:=I*IJC4-JJO+J*1J3.:0+K*1.00+21;
WRITE(T)
END;

ENTRY PROCEDURE MESS1(I,J,K:INTEGER):
VAR T:INTEGER;
BEGIN
T:=I*IJC4-JJO+J*1J3.:0+K*1.00+21;
WRITE(T)
END;

ENTRY PROCEDURE MESS12(I,J,K:INTEGER):
VAR T:INTEGER;
BEGIN
T:=I*IJC4-JJO+J*1J3.:0+K*1.00+21;
WRITE(T)
END;

ENTRY PROCEDURE MESS13(I,J,DU,AS INTEGER):
VAR T1,T3:INTEGER;
BEGIN
DU:=DU DIV 133;
T1:=I*IJC4-JJO+J*1J3.:0+K*1.00+21;
WRITE(T)
END;

ENTRY PROCEDURE MESS14(I,J:INTEGER):
VAR T1:INTEGER;
BEGIN
T:=I*IJC4-JJO+J*1J3.:0+K*1.00+21;
WRITE(T)
END;

ENTRY PROCEDURE MESS15(I,RES,ROL,COM,ARE,K1,K2,K3:INTEGER):
VAR T2,T3,T4,T5,T6,T7,T8,T9:T100,T1000 INTEGER;
BEGIN
T2:=T1000; T3:=I*1.001;
T4:=T1000*T1000+T1000+32;
T5:=T1000*T1000+T1000+33;
T6:=T1000*T1000+T1000+34;
IF K1<>99 THEN BEGIN
T7:=K2*T1000+T1000+36;
T8:=K2*T1000+T1000+24;
T9:=K3*T1000+T1000+25;
WRITE(T7,T8,T9)
END;
WRITE(T3,T4,T5,T6)
ENTRY PROCEDURE TOLINE(MESSAGE);
BEGIN
MESSAGE=MESSAGE;
END;
(* TOLINE *)
ENTRY PROCEDURE FRLINE(MESSAGE);
BEGIN
MESSAGE=MESSAGE;
END;
(* FRLINE *)
ENTRY PROCEDURE QUEGET(MESSAGE1:MESSAGE;
BEGIN
MESSAGE=MESSAGE1;
END;
(* QUEGET *)
ENTRY PROCEDURE QUEPUT(MESSAGE);
BEGIN
MESSAGE=MESSAGE;
END;
(* QUEPUT *)
ENTRY PROCEDURE TOLINE(MESSAGE);
BEGIN
MESSAGE=MESSAGE;
END;
(* TOLINE *)
ENTRY PROCEDURE FRLINE(MESSAGE);
BEGIN
MESSAGE=MESSAGE;
END;
(* FRLINE *)
ENTRY PROCEDURE QUEGET(MESSAGE:MESSAGE1:MESSAGE;
BEGIN
MESSAGE=MESSAGE1;MESSAGE;
END;
(* QUEGET *)
ENTRY PROCEDURE QUEPUT(MESSAGE);
BEGIN
MESSAGE=MESSAGE;
END;
(* QUEPUT *)
ENTRY PROCEDURE TOLINE(MESSAGE);
BEGIN
MESSAGE=MESSAGE;
END;
(* TOLINE *)
ENTRY PROCEDURE FRLINE(MESSAGE);
BEGIN
MESSAGE=MESSAGE;
END;
(* FRLINE *)
ENTRY PROCEDURE QUEGET(MESSAGE:MESSAGE1:MESSAGE;
BEGIN
MESSAGE=MESSAGE1;MESSAGE;
END;
(* QUEGET *)
ENTRY PROCEDURE QUEPUT(MESSAGE);
BEGIN
MESSAGE=MESSAGE;
END;
(* QUEPUT *
OUTQUEUE=OBJECT (* MSGS TO BE SENT OUT *)
PATH BMAX=(OUTPUT)11; (OUTGET) END;
VAR OUTBUFFER:ARRAY(1..BMAX) OF MESSAGE;
OUTO,OUTG:1..BMAX

ENTRY PROCEDURE OUTPUT MMESSAGE:
BEGIN
OUTBUFFER[OUTP]=M;
OUTP=(OUTP MOD BMAX) +1
END: (* OUTPUT *)
ENTRY PROCEDURE OUTGET VAR MMESSAGE:
BEGIN
M=OUTBUFFER[OUTG];
OUTG=(OUTG MOD BMAX) +1
END: (* OUTGET *)
INIT: BEGIN
OUTP=1;
OUTG=1
END: (* INIT *)
END: (* ----------- OUTQUEUE ----------- *)
PRBUFS=OBJECT (* PRIVATE BUFFER FOR EACH PROCESS *)
PATH 111(PRBUFS)(PRBUFS) END;
VAR PRBUFFER:MESSAGE;
ENTRY PROCEDURE PRBUFFPUT MMESSAGE:
BEGIN
PRBUFFER=M
END: (* PRBUFFER *)
ENTRY PROCEDURE PRBUFFGET VAR MMESSAGE:
BEGIN
M=PRBUFFER
END: (* PRBUFFGET *)
END: (* ----------- PRBUF ----------- *)
RESRC=RECORD
NAME:INTEGER;
STATUS:STATUS;
ACCESS:INTEGER;
END;
VAR
HQUEUE=MSGQUEUE;
QUEUE=OUTQUEUE;
PRBUFSARRAY[1..5] OF PRBUFS;
ID:PROC;
Larestabarray[1..BMAX] OF RESRC;
N,M,PP,RR1 INTEGER;
REQACCESS:STATUS;
FUNCTION FINOR(R:INTEGER):INTEGER;

(* RETURNS AN INDEX TO A RESOURCE IN RESOURCETABLE *)

VAR II:INTEGER;

BEGIN
II:=0;
WHILE LRESTA[I].RNAME<>R DO II:=II+1;
FINOR:=II;
END;  (* FINOR *)

PROCEDURE RESREL;

VAR I:INTEGER;

BEGIN
II:=FINOR(RR);
LRESTAB[II].NACCESS:=LRESTAB[II].NACCESS+1;
IF LRESTAB[II].NACCESS=0 THEN
LRESTAB[II].STATUS:=FREE
END;

PROCEDURE SENRESPONSE;

(* SEND OUT RESPONSE TO REQUESTING PROCESSES *)

BEGIN
FINOR(RR);
LRESTAB[II].STATUS:=REQACCESS;
LRESTAB[II].NACCESS:=LRESTAB[II].NACCESS+1;
WITH MSGTEMP DO
BEGIN
MSGTYPE:=RESPONSE;
MSGDEST:=MSGORIGIN;
MSGORIGIN:=MYSITE;
END;
END;

PROCEDURE SENDROLL;

(* SENDS ROLLBACK MESSAGE *)

BEGIN
TIDDEAD:=TIDDEAD+1;
WITH MSGTEMP DO
BEGIN
MSGTYPE:=ROLLBACK;
MSGDEST:=MSGORIGIN;
MSGORIGIN:=MYSITE
END;
END;

IF MSGTEMP.MSGDEST=MYSITE THEN
PROCEDURE RESREQ;
(* PROCESS PP REQUEST FOR RESOURCE RR *)
VAR I:INTEGER;
RFREE:BOOLEAN;
THEO:STATUS;
BEGIN
TOTREQ:=TOTREQ+1;
IF T=RETURN(RR); RFREE:=FALSE; THEO:=RESTAB(IFR).RSTATUS;
IF THEO=FREE THEN RFREE:=TRUE ELSE
IF THEO=SHAREO AND R ACCESS=SHAREO THEN
RFREE:=TRUE;
IF RFREE THEN SENRESPONSE ELSE SENDROLL;
END; (* RESREQ *)
PROCEDURE MANAGER;
BEGIN...
IF TENTRY=REQ THEN RESREQ ELSE RESREL
END; (* MANAGER *)

FUNCTION RAND(VAR SEED:REAL; MOD:INTEGER):INTEGER
CONST
P=2147483647;
A=16645;
VAR ISEED:INTEGER;
BEGIN
ISEED:=TRUNC(SEED);1
SEE=(ISEED*P) MOD P;
ISEED:=TRUNC(SEED) MOD MOD;
END; (* RAND *)

FUNCTION RANNYVAR SEED:REAL; MOD:INTEGER):REAL
VAR ISEED:INTEGER;
BEGIN
ISEED:=TRUNC(SEED);
S=(ISEED*3) MOD 32767;
RANOMI=S/32767.0
END; (* RANOM *)
PROCESS WRITER(INLINES, LINESSITE, MAXP: INTEGER):
(* WRITE MSG TO OUTPUT LINE *)
VAR ITEMSGSENT: INTEGER;
BEGIN
WRITING := TRUE;
ITEMSGSENT := 0;
WHILE WRITING DO
BEGIN
QUEUE.OUTGET(ITEMSGSENT);
WRITING := WRITING + 1;
IF MSGTYPE = ABORTED THEN WRITING := 0;
END;
ITEMSGSENT := ITEMSGSENT + 1;
IF WRITING = 0 THEN WRITING := 1;
END;
(* WRITER *)

PROCESS READER(INLINES, LINESSITE, MAXP: INTEGER):
(* MONITOR INPUT LINE FOR ALL INCOMING MESSAGES; IF MSG IS FOR A LOCAL
PROCESS IT AWAKES 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 CUTFBUFFER TO
BE PASSED ON; IF THE MSG IS A DETECTION MSG OR RESOURCE RELEASE
FOR A LOCAL RESOURCE IT PUTS IT IN MSGQUEUE *)
VAR MSGMESSAGE:
BEGIN
READING := TRUE;
RETRY := 0;
TOTAL := 0;
WHILE READING DO
BEGIN
INLINE.FRLINE(MSGMESSAGE);
CASE MSGMESSAGE.
END;
(* READER *)
AREQUEST:
BEGIN
  TOTAREQ:=TOTAREQ+1;
  S:=FALSE;
  FOR I:=J TO M DO
    IF RLISTAB[I].RNAME=MESS.RESNAME THEN SW:=TRUE;
    IF SW THEN QUEUE.QUEUE(MESS) ELSE QUEUE.QUEUE(MESS)
END:

ARESPONSE,ROLLBACK:
BEGIN
  CASE MESS.MSGTYPE OF
    ARESPONSE: TOTRESP:=TOTRESP+1;
    ROLLBACK: TOTROLLB:=TOTROLLB+1
  END;

  IF MESS.MSGDEST=SITE THEN
    PROCESS(MESS,PROCMODE,PRSPUT(MESS))
  ELSE QUEUE.QUEUE(MESS)
END:

COMPLETION:
BEGIN
  TOTCOMPL:=TOTCOMPL+1;
  IF MESS.MSGDEST=SITE THEN QUEUE.QUEUE(MESS)
  ELSE QUEUE.QUEUE(MESS)
END:

ATERMINATE:
BEGIN
  RTOTL:=RTOTL+1;
  IF MESS.MSGORIGIN=SITE THEN QUEUE.QUEUE(MESS)
  IF RTOTL=MAXP THEN READING:=FALSE
END:

PROCESS KERNEL(SITE=SITE,MAXP=MAXP, INTEGER):
(* THE KERNEL HANDLES THE RESOURCE ALLOCATION AT EACH SITE *)
  IT RUNS THE DETECTION ALGORITHM *)
VAR KTOFL,  TOTMAX, TOTLOC, QSIZE: INTEGER;
KERNELLING, SW: BOOLEAN;
BEGIN
  KERNELLING:=TRUE;
  ID:=MAXP-1;  KIDTL:=0;  TOTLOC:=0;
(* THE KERNEL DOES NOT PROCESS ANY OTHER RESOURCE REQUEST UNTIL *)
IT HAS RESOLVED ANY OUTSTANDING REQUEST *)
WHILE KERNELING DO
BEGIN
  QUEUE.QUEUE=(MSGTEMP.OSIZE);
  CASE MSGTEMP.MSGTYPE OF
  TERMINATE:
    BEGIN
      KTOTL=KTOTL+1;
      QUEUE.OUTPUT((MSGTEMP)
    END;
  LOCAL:
    BEGIN
      MSGTEMP.QUEUESIZE=O SIZE;
      MSGTEMP.MSGTYPE=AREQUEST;
      SW:=FALSE;
      FOR I=0 TO M DO
        IF MSGTEMP.RESTYPE=LESTABLE.I.RESTYPE THEN
          SW:=TRUE;
      IF NOT SW THEN QUEUE.OUTPUT((MSGTEMP) ELSE
    BEGIN
      WITH MSGTEMP DO
        BEGIN
          PP=MSGORIGIN*1000+PROCNAME;
          RR=RESNAME;
          REOACCESSSI=ACESTYPE
          TENTRYI=RE0;
          TOTLOC=TOTLOC+1;
        END;
        MANAGER
        END
      END;
    AREQUEST. COMPLETION:
    BEGIN
      WITH MSGTEMP.DO
        BEGIN
          QUEUESIZE=QUEUESIZE+OSIZE;
          PP=MSGORIGIN*1000+PROCNAME;
          RR=RESNAME;
          REOACCESSSI=ACESTYPE
          IF MSGTYPE=AREQUEST THEN TENTRYI=REL.
          END;
        MANAGER
        END;
        END;
        END;
      IF KTOTL=TOTM.AP THEN KERNELING:=FALSE;
    END;
    END;
  KERNELLING := TRUE;
  IO.MESSI(SITE,TODEAD,TOTREQ);
  IO.MESSI=(OSIZE,TOTLOC);
END;
(* KERNEL *)
PROCESS PPROCSS(SITE,TOTMAXR,PROCHO:INTEGER;LAMAX,MMAX:REAL; MAXREQ,MAXACC,THRUPUT:INTEGER);

(* SIMULATE A LOCAL PROCESS ACTIVITIES *)

LABEL 1.21

TYPE
LRES=RECORD
LRNAME:INTEGER;
TACCESS:STATUS;
LOCATION:INTEGER
END;

VAR
RESPCES:ARRAY[1..19] OF LRES;
REOPTR,OUTREQ,REQUEST,LANDBAR,MMUBAR,SEEDR,SEED1:REAL;
TEMP,T2,T3,BEFORE,TOTSO,XXX:REAL;
NUMRES,NUMR,J,LOCTR,LIN_CI,OUTREQ1,REOPTR,REO1R,REPOTR:INTEGER;
LPRAOC,TOUTRE,TTHRU,BEFORE,THRUAFER:INTEGER;
TESTCASE,TS,T0,PPP:INTEGER;
*INSW,SW1,SW1,SW1,GREATR,PROCESSING,AGAIN TRUE,1AN;
ACCTYPE,STATUS;

PROCEDURE GENFREQ;
BEGIN (* GENERATE NEW RESOURCE *)
SW1:=FALSE;
WHILE NOT SW1 DO
BEGIN
UNI:=RANDOM(TOTMAXR)+1;
REOPTR:=REO1R+1;
FOR J:=1 TO OUTREQ DO
BEGIN
IF RESRCES[J].LRNAME=RR THEN SW1:=TRUE;
J:=(J MOD 19)+1
END;
IF NOT SW1 THEN SW1:=TRUE
END;

(* TYPE OF ACCESS *)
IF #ACCESS=1 THEN ACCTYPE:=EXCLUSIVE ELSE
BEGIN
TEMP:=RANDOM(SEEDR);
IF TEMP=0.5 THEN ACCTYPE:=EXCLUSIVE ELSE ACCTYPE:=SHARED
END;
REOPTR:=(REO1R MOD 19)+1;
OUTREQ:=OUTREQ+1;
RESRCES[REO1R].LRNAME:=RR;
RESRCES[REO1R].TACCESS:=ACCTYPE
(* SEND REQUEST *)
WITH MYMSG DO

BEGIN
MSGORIGIN=SITE; PROCNAME=LPROCID;
MSGTYPE=LOCAL; RESNAME=RR; QUESIZE=:
AGETYPE=ACCTYPE;
END
BEGIN
MSGORIGIN=SITE; PROCNAME=LPROCID;
MSGTYPE=LOCAL; RESNAME=RR; QUESIZE=:
AGETYPE=ACCTYPE;
END
BEGIN
IO.MESS(SITE,LPROCID,RR,ACCTYPE);
QUEUE.JUEPUT(MYMSG);
J=J+1; TOTSENT=TOTSENT+1;
(* TBEFORE=SIN(XXX): *)
BEGIN
IO.MESS(SITE,LPROCID,TC,INP)
BEGIN
IO.MESS4(SITE,LPROCID,MYMSG,RESNAME);
RESR; RSCRECPTRI.LOCATION1=MYMSG.ORIGIN;
IF at Ns....ISO:2r RE-agC1.4.,-SACK._ THEN__
BEGIN
(* IO.MESS4(SITE,LPROCID,MYMSG,RESNAME); *)
RESR; RSCRECPTRI.LOCATION1=MYMSG.ORIGIN;
ELSE
BEGIN
(* GENREQ *)
PROCEDURE ASSREL:
BEGIN
RELPTR=(RELPTR-MOD-15)+1; OUTREQ=OUTREQ-1;
WITH MYMSG DO
BEGIN
PROCNAME=LPROCID; MSGTYPE=COMPLETION;
MSGORIGIN=SITE;
MSGRST=RESR; RELPTR1.LOCATION;
RESNAME=RESR; RELPTR1.LNAME; ACCTYPE=FREE
END
END
case (* RELPTR *)
BEGIN
LPROCID=LPROCID;
TOTSENT=01; TOTDELY=01; XXX=5.01; PROCESING1=TRUE;
TOTSEC=2.0; CLXXH=0.1;
SEOR=31; ISITE; SEED=SITE; TRELEASE=0.01; FREQUENCY1=0.01
THREFORE=TIME;
(* RELPTR POINTS TO THE LAST RESOURCE RELEASED)
RELPTR.POINTS.TO THE LAST RESOURCE REQUESTED FOR *)
IF RELPTR=01; RELPTR1=01; GREATER1=FALSE;
OUTREQ=1; AGAIN1=FALSE;
MAINSW1=FALSE;
WHILE PROCESSING DO
    MP1=-1;
    GENERQ:
        IF MP1<>0 THEN GOTO 21;
        (* GENERATE TIME OF NEXT RELEASE *)
        MUUBAR1=(-1.0/MUUBAR)*LN(RANDOM(SEEJR));
        RELEASE=CLOCK+MUUBAR1;
        LAMABAR1=-(1.0/LAMABAR)*LN(RANDOM(SEEJR));
        TREQUEST=CLOCK+LAMABAR1;
        MAINSW=TRUE;
        WHILE MAINS DO
            IF RELEASE>TREQUEST THEN TESTCASE1=1;
            IF RELEASE<TREQUEST THEN TESTCASE1=2;
            IF RELEASE=TREQUEST THEN TESTCASE1=3;
            CASE TESTCASE OF
                ( RELEASE>TREQUEST )
                    BEGIN
                        TEMP=TMUBAR1*10.0;
                        T2=I+3.49;
                        IF TEMP>T2 THEN I=I+1;
                        IF (TOT<>lT&&T<>T) THEN
                            BEGIN
                                AGAIN=FALSE; MP1=-1; PROCESSING=FALSE; GOTO 2
                            END
                    END
                    MUUBAR1=RELEASE-TREQUEST;
                    ( * GENERATE REQUEST * )
                    IF TREQUEST=TMAXR THEN
                        BEGIN
                            (* REQUEST BUT RES HELD EQUALS MAX RES *)
                            TREQUEST=TREQUEST-ASSREL;
                            IF MYMSG.MSGEST=SITEM THEN
                                QUEUE.QUEUE=MYMSG;
                                ELSE
                                QUEUE.QUEUE=MYMSG;
                            END
                            MAINSW=FALSE
                        END ELSE
                        BEGIN
                            MP1=-1; GENERQ;
                            IF MP1<>0 THEN GOTO 21;
                            (* GENERATE TIME OF NEXT REQUEST *)
                            LAMABAR1=-(1.0/LAMABAR)*LN(RANDOM(SEEJR));
                            TREQUEST=CLOCK+LAMABAR1; MAINSW=TRUE
                        END
                ( TREQUEST=TREQUEST )
                    BEGIN
                        CLOCK=RELEASE;
                        TEMP=LAMABAR1*10.0;
                        I=TRUNC(TMP); T2=I+3.49;
                ( * TESTCASE=2 *)
                21 (* RELEASE=TREQUEST *)
                    BEGIN
                        CLOCK=RELEASE;
                        TEMP=LAMABAR1*10.0;
                        I=TRUNC(TMP); T2=I+3.49;
IF TEMP>T2 THEN I:=I+1; DELAY(I);
(* RELEASE RESOURCE IF ANY *)
IF OUTREQ>0 THEN
BEGIN
ASSREL;
IF MYMSG.MSGDEST=SITE THEN QUEUE.QUEPUT(MYMSG) ELSE QUEUE.OUTPUT(MYMSG)
(* IO.MSGDEL(SITE,LPROCID,RESRCES(RELPTR),LNAME) *)
END;
NAMISM:=FALSE;
IF (TOTSsent>TMAXREQ) THEN
BEGIN
AGAIN:=FALSE; MPI=-1; PROCESSING:=FALSE; GOTO 2
END;
END;

(* TRLEASURE*REQUEST *)
BEGIN
TEMP=MUUBARI*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:=REQUEST;
TEMP=(REQUEST-RELEASE)*100.1;
I=TRUNC(TEMP); T2:=I+0.49;
IF TEMP>T2 THEN I:=I+1; DELAY(I); NAMISM:=FALSE
END ELSE
BEGIN (* RELEASE RESOURCE *)
CLOCK:=RELEASE; ASSREL;
IF MYMSG.MSGDEST=SITE THEN QUEUE.QUEPUT(MYMSG) ELSE QUEUE.OUTPUT(MYMSG);
(* IO.MSGDEL(SITE,LPROCID,RESRCES(RELPTR),LNAME) *)
MUUBARI=(1.0/MUUBARI)*LN(RANDOM(SEEOR));
RELEASE:=CLOCK+MUUBARI;
IF (TOTSsent>TMAXREQ) THEN
BEGIN
AGAIN:=FALSE; MPI=-1; PROCESSING:=FALSE;
GOTO 2
END
END
(* TRLEASURE < REQUEST *)
END;
(* MAINSM *)
END;
(* PROCESSING *)
21 IF OUTREQ>0 THEN
BEGIN
ASSREL;
IF MYMSG.MSGDEST<>MP THEN
BEGIN
IF MYMSG.MSGDEST=SITE THEN QUEUE.QUEPUT(MYMSG)
ELSE QUEUE.OUTPUT(MYMSG)
(* IO.MESS6(SITE,LPROCID,MYMSG,RESNAME) *)
END;

IF AGAN THEN
BEGIN
I:=100; J:=RAND(SEED,1)+100;
DELAY(J);
IF THAUPUT=1 THEN TOTSENT:=T;
GOTO 1
END;

IF THAUPUT=1 THEN THRAFTER:=TIME-THUBEORE
ELSE THRAFTER:=39900;
MYMSG.MSGTYPE:=TERMINATE;
MYMSG.MSGQEST:=SITE1;
QUEUE.QUEUEPUT(MYMSG);
IO.MESS7(SITE,LPROCID,TOTSENT,THRAFTER);
END (* PROCESS PPROCESS *)

ENTRY PROCEDURE STARTMACH(SITE); SITE1; INLINE; OUTLINE; LINE; MAXR,SRES, SIARX; MAXP; PDEP; INTEGER; LAMDA; MUUX; REAL; MAXRE; MAXAC; THRUP; INTEGER;
VAR I, J; INTEGER;
BEGIN
I:=1; M:=SRES-1; TOTREQ:=J; TOTDEAD:=0; MYSITE:=SITE1
(* INITIALISE RESOURCES FOR THIS SITE *)
J:=STARTR;
FOR I:=1 TO SRES+1 DO
BEGIN
WITH LRESTABII DO
BEGIN
RNAME:=J; RSTATUS:=FREE; MAXAC:=0
END
J:=J+1
END; (* START PROCESSES AT THIS SITE *)
FOR I:=1 TO PROCS DO
PPROCESS(SITE,MAXR,I,LAMDA,MUUX,MAXRE,MACTH,THRUP):
KERNEL(SITE1,SRES,CUXP):
READER(INLINE,LAMDA):
WRITER(OUTLINE,SITE,MAXR):
END (* STARTMACH *)
END; (* SYSTEM ACTIVATION *)

VAR
NETARRAY(SITES) OF MACHINE1
LINK, ARRAY [NLINES] OF LINE;
MAXREQ, MAXP, MAXR, WACC, THRUP 1 INTEGER;
RPER SITE, ARRAY [SITES] OF INTEGER;
PERSITE, ARRAY [SITES] OF INTEGER;
I, J, Y, K 1 INTEGER;
(*) WACC=0 IF SHARED & EXCL & 1 IF EXCL ONLY *)
(*) THRUP=0 PERFORMANCE MEASURE, =1 THRUP MEASURE *)
LAMOA, MUU REAL;
BEGIN
READ (MAXP, MAXR, LAMOA, MUU, MAXREQ, WACC, THRUP);
(* DISTRIBUTE RESOURCES AMONG SITES *)
K=0; I=MAXR DIV NSITES;
J=1; Y=MAXP DIV NSITES;
FOR I=1 TO NSITES DO
BEGIN
RPERSITE(I) = I; K=K+1;
PERSITE(I) = Y; J=J+1
END;
I=1;
WHILE K<MAXR DO
BEGIN
I=I+1
RPERSITE(I) = RPERSITE(I)+1
K=K+1
END;
I=1;
WHILE J<MAXP DO
BEGIN
J=J+1
RPERSITE(I) = RPERSITE(I)+1
END;
WRITE (*, READLOCK, +, PREVENTION, +, METHOD+);
WRITE (* M = *, MAXP, # M = *, MAXR);
WRITE (* MUU = *, MUU, # LAMOA = *, LAMOA);
WRITE (* MAXIMUM #, REQUEST #, MAXREQ#);
NET(I), STARTMACH(I), LINK(I), MAXRX, RPERSITE(I),
MAXP, PERSITE(I), LAMOA, MUU, MAXREQ, WACC, THRUP;
I=APRERSITE(I)+1;
NET(2), STARTMACH(2), LINK(2), MAXR, RPERSITE(2),
MAXP, PERSITE(2), LAMOA, MUU, MAXREQ, WACC, THRUP;
I=1+RPERSITE(2);
NET(3), STARTMACH(3), LINK(3), MAXR, RPERSITE(3),
MAXP, PERSITE(3), LAMOA, 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 qesize 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