AD-A135 458 PRESERVING ASYMMETRY BY SYMMETRIC PROCESSES AND DISTRIBUTED FAIR CONFLICT..(U) TEXAS UNIV AT AUSTIN DEPT OF COMPUTER SCIENCES K M CHANDY ET AL. APR 83 1/1

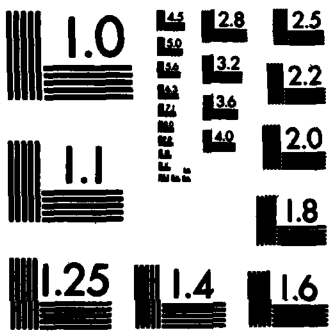
UNCLASSIFIED AFOSR-TR-83-0990 AFOSR-81-0205 F/G 9/2 NL

END

FILMED

1 84

DTIC


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

3

A135458

Preserving Asymmetry by Symmetric Processes and Distributed Fair Conflict Resolution

K. M. Chandy
J. Misra
Department of Computer Sciences
University of Texas At Austin
Austin, Texas 78712

April 1983

This work was supported by the Air Force Office of Scientific Research under Grant AFOSR-81-0205.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 83-0990	2. GOVT ACCESSION NO. A135458	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Preserving Asymmetry by Symmetric Processes and Distributed Fair Conflict Resolution		5. TYPE OF REPORT & PERIOD COVERED interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) K. M. Chandy and J. Misra		8. CONTRACT OR GRANT NUMBER(s) AFOSR-81-0205
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Sciences Department The University of Texas at Austin Austin, Texas 78712		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS AFOSR/NM Bolling AFB, DC 20332		12. REPORT DATE April 83
		13. NUMBER OF PAGES 11 pages
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Conflicts arising in distributed systems, as in contentions for shared resources, are resolved either (1) by a central process or (2) by resorting to probabilistic decision making by individual processes or (3) by assigning a static global priority to each process. All known non-probabilistic solutions to the conflict resolution problem are asymmetric in the sense that they distinguish between processes by ordering process ids or by having some processes carry out special functions. We propose an efficient, fair

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AFOSR-TR- 83-0990

symmetric solution for this problem: asymmetry is present initially by judicious placement of shared resources and asymmetry is preserved in a fair manner by our solution. To providea concrete framework for our discussion of conflict resolution we couch our discussion in terms of a generalization of the classical dining philosophers' problem.

A-1

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Abstract

Conflicts arising in distributed systems, as in contentions for shared resources, are resolved either (1) by a central process or (2) by resorting to probabilistic decision making by individual processes or (3) by assigning a static global priority to each process. All known non-probabilistic solutions to the conflict resolution problem are asymmetric in the sense that they distinguish between processes by ordering process ids or by having some processes carry out special functions. We propose an efficient, fair, symmetric solution for this problem: asymmetry is present initially by judicious placement of shared resources and asymmetry is preserved in a fair manner by our solution. To provide a concrete framework for our discussion of conflict resolution we couch our discussion in terms of a generalization of the classical dining philosophers' problem.

Asymmetry in Message Passing Systems

Conflicts arise in distributed systems because two or more processes cannot take arbitrary actions simultaneously. For instance, some resources, such as "write locks," cannot be exercised simultaneously by 2 or more processes. Exclusive access to a shared resource introduces "asymmetry" among processes who wish to share the resource in the sense that at any given point in the computation, the process holding the resource may behave differently from those which don't. Usually, it is desired that though some processes are treated preferentially in the short term, all processes are treated fairly in the long term: this desideratum may be thought of as "short-term asymmetry and long-term symmetry." Lehmann and Rabin [1] have proved that it is impossible for an ensemble of perfectly symmetric processes in a symmetric global state to create asymmetry without resorting to probabilistic decision making. We argue that processes in a *message-passing* network *can never be in identical states, with respect to resources* because the locations of the *indivisible* resources (either at a process, or traveling towards a process) introduce asymmetry. We exploit this resource location asymmetry to resolve conflicts for resources and ensure long-term symmetry despite inherent short-term asymmetry. To provide a concrete framework, we couch our discussion in terms of the *distributed drinking philosophers'* problem, described next; we propose an efficient fair solution to this problem. Our solution makes use of a novel solution to the distributed dining philosophers' problem, which was first defined in [2].

The Distributed Drinking Philosophers' Problem

The following problem is a variant of one due to E. W. Dijkstra [2]. Dijkstra's original problem [4] has achieved the status of legend since it captures the essence of many synchronization problems. A far-flung network of philosophers is represented by a finite undirected graph G with one philosopher at each vertex. A philosopher is in one of 3 states: (1) tranquil (2) thirsty (i.e., waiting to drink) or (3) drinking. Associated with each edge $\{v_i,v_j\}$ in G is a bottle.[1] A philosopher can drink only from bottles associated with his incident edges. Each philosopher chooses a subset of bottles that he wishes to drink from, when he becomes thirsty. *He may choose different subsets of bottles for different drinking sessions.* A philosopher is thirsty if he desires to proceed to drinking state but is unable to do so because he does not have all the bottles he needs. On receiving all needed bottles a thirsty philosopher starts drinking; a thirsty philosopher remains thirsty until he gets all bottles he needs to drink. On entering the drinking state, a philosopher remains in that state for a finite period, after which he becomes tranquil. A philosopher may be in the tranquil state for an arbitrary period of time.

Philosophers v_i and v_j are *neighbors* if and only if edge $\{v_i,v_j\}$ exists in G. Neighbors may mail packages to one another. Philosopher v_i has a mailbox which can hold an arbitrary number of packages; we show later that only a bounded number of packages will ever be in any mailbox. The postal service guarantees that it will deliver all mailed packages in finite time without mutilating the contents. A package is delivered to (the address of) a philosopher v_i if and only if the package was mailed to v_i by another philosopher; the postal service does not duplicate or create packages. The bottle associated with an edge $\{v_i,v_j\}$ is either at v_i or at v_j or in the mail from v_i to v_j or from

[1]The solution given in this paper also applies to multiple bottles on every edge. The assumption of one bottle per edge is made for simplicity in exposition.

v_j to v_i.

The problem is to devise a non-probabilistic solution which satisfies the following constraints.

fairness:	No philosopher remains thirsty forever.
symmetry:	All philosophers obey precisely the same rules for acquiring and releasing bottles. There is no priority or any other form of externally specified static partial ordering among philosophers or bottles.
economy:	A philosopher sends and receives a finite number of packages in every state (tranquil, thirsty, drinking). In particular, permanently tranquil philosophers do not send or receive an infinite number of packages.
concurrency:	The solution does not deny the possibility of simultaneous drinking from different bottles by any two philosophers.

Importance of the Distributed Drinking Philosophers' Problem

The distributed drinking philosophers' problem is a general paradigm for modelling conflicts between processes. This paradigm has the following features: (1) two neighboring philosophers may be prevented from simultaneously drinking in some cases, i.e. drinking from the same bottle, which corresponds for instance, to writing into shared a file, (2) two neighboring philosophers may drink simultaneously in some cases, i.e. drinking from different bottles, which corresponds for instance, to writing into different files or reading from the same file.

Therefore the drinking philosophers' problem models *dynamic conflicts*, i.e., conflicts that may be determined by the data on which a process operates. A conservative solution to this problem may always avoid conflicts among *all* neighbors; this however leads to loss in concurrency.

The paradigm can also model N-way conflicts arising, as in distributed resource allocation problem, where there are m_i indivisible units of resource r_i to be shared among N processes. A process wishing to enter its critical section specifies the set of resources it needs. Thus there is an N-way conflict for a resource. N-way conflicts may be modelled as a set of N^2 2-way conflicts: two processes are neighbors if it is possible that they may conflict.

Previous Work

If a philosopher requires *all* bottles from its incident edges for *all* drinking sessions, then our problem reduces to Dijkstra's distributed dining philosophers' problem [2]; in this problem, there is a single "fork" on each edge and a philosopher "eats" (corresponding to drinking) only if he holds forks for *all* incident edges. Therefore the drinking philosophers' problem is a generalization of the distributed dining philosophers' problem. The distributed dining philosophers' problem is a generalization of Dijkstra's classical dining philosophers' problem [4]; in the latter problem the philosophers sit around a table (i.e. they are arranged in the form of a ring). The wealth of literature on the classical dining philosophers' problem shows that it is a fundamental paradigm of concurrent processing. Dijkstra's solutions to the distributed dining philosophers' problem [2] assume instantaneous transmission of packages or a static ordering of forks (which is asymmetric). Lynch [3] has carried out an extensive analysis of static resource ordering algorithms.

The problem of mutual exclusion among a group of processes in executing their critical sections, is a special case of the distributed dining philosophers' problem : every process is a neighbor of every other process and execution of a critical section corresponds to eating. Distributed solutions to mutual exclusion using process id's to break ties, appear in Lamport [5], Ricart and Agrawala [6].

A symmetric distributed solution to the dining philosophers' problem appears in Frances and Rodeh [7]. They use an extended form of CSP [8], in which both input and output commands are used in guards. Lehmann and Rabin [1] give a perfectly symmetric probabilistic algorithm and show that there is no perfectly symmetric non-probabilistic solution to the dining philosophers' problem. Therefore it follows that the extended form of CSP cannot be implemented by a symmetric protocol.

We first present a solution to the distributed dining philosophers' problem and then use this solution in solving the distributed drinking philosophers' problem.

A Hygienic Solution to the Distributed Dining Philosophers' Problem

In the distributed dining philosophers' problem, a philosopher is either thinking, hungry or eating. Associated with each edge of the graph is a fork and a hungry philosopher needs forks of *all* incident edges, to eat. We begin by presenting a solution to the distributed dining philosophers' problem with the properties of fairness, symmetry, economy and concurrency. We consider the more general, drinking philosophers' problem in the next section.

A fork is either *clean* or *dirty*. A fork that is being used to eat with, is dirty and remains dirty until it is cleaned. A philosopher can only clean forks that he holds. A clean fork remains clean until it is used for eating. A fork is said to be *free* if the philosopher holding it is either thinking or hungry , i.e. not eating. A philosopher cleans a fork prior to mailing it (because he is hygienic). The key issue in the dining philosophers' problem is: which requests should a philosopher defer till he has next eaten? In our algorithm, a philosopher defers satisfying requests for forks that are clean (because his altruism is limited) or not free (because he is eating with it). A philosopher satisifies requests for free, dirty forks immediately.

We now state the algorithm in detail, for a philosopher v_i. In this description "satisfy a request" means clean and send the requested fork and "forks v_i needs" denotes the set of forks associated with the edges incident on v_i.

Thinking State {all forks held by v_i are free and dirty.}
Satisfy all requests received.

Hungry State {all forks held by v_i are free; forks received in the mail by v_i since last entering hungry state are clean; all the remaining forks held by v_i are dirty.}
Request every fork that v_i needs and does not hold. {Therefore, v_i must request a fork that is released in this state.} Enter eating state when v_i holds all forks it needs. Satisfy requests in the hungry state if they are for dirty forks, else defer request.

Eating State {forks for all incident edges are held by v_i; none of these forks is free.} Defer all requests received in this state. Eat. Upon completion of eating {all forks are dirty and free} satisfy all deferred requests and all requests in the mailbox.

The proof of correctness of the algorithm and the specification of initial conditions require the definition of a directed graph H; this definition and the initial condition are given next. Observe that our solution satisfies the constraints of fairness, symmetry, economy and concurrency.

The Graph H

We shall derive a directed graph H from G and the states of forks, as follows: direct each edge $\{v_i,v_j\}$ in G from v_i to v_j if and only if the fork associated with the edge is: (1) at v_i and dirty or (2) in transit from v_i to v_j or (3) at v_j and clean.

Initial Condition

All forks are dirty and H is acyclic.

Proof of Correctness

We first show that H is always acyclic. The only change to H occurs when a philosopher dirties a clean fork (since cleaning a dirty fork and sending it does not change the direction of the edge). A fork is dirtied only when a philosopher, say v_i eats, in which case he must be holding all forks which must all be dirty. Therefore v_i cannot belong to a cycle on eating, because all edges incident on v_i are directed away from it (since all forks held by v_i are dirty). It is given that H is acyclic initially. Since all changes to H preserve acyclicity, H must always be acyclic.

We show that if (v_i,v_j) is an edge in H and v_j is hungry, then v_j either holds now or *will hold later, the clean fork associated with this edge. Since* (v_i,v_j) *is an edge in H,* then either (1) v_j holds a clean fork, (2) a clean fork is in the mail from v_i to v_j or (3) v_i is holding a dirty fork. The result is trivial for the first two cases. In the third case v_i will receive a request from v_j in finite time. v_i must mail v_j the fork after cleaning it, when the fork is dirty and free; the fork remains dirty until v_i cleans it and mails it to v_j. Therefore v_i must mail the fork to v_j when it becomes free. The fork is free unless v_i is eating. Since an eating session lasts for finite time, the fork must become free in finite time. Therefore v_j will hold the clean fork in finite time.

We show in theorem 3 that every hungry philosopher will (hold all forks it needs and) eat in finite time. We have observed that for each incoming edge (v_i,v_j) to v_j, if v_j is hungry, then v_j will hold the clean fork associated with this edge. Now we must show that for every outgoing edge (v_j,v_k) of v_j, either (1) v_j holds the dirty fork for this edge and receives no requests from v_k or (ii) v_k eats and then sends v_j the clean fork associated with this edge. In either case v_j will hold the fork for this edge and continue to hold it at least until it next eats. The result is proved by induction in theorem 3 (below).

Lemma 1: H is always acyclic.

Proof: See above discussion.

Lemma 2: If v_j is hungry and has an incoming edge (v_i,v_j) in H then v_j holds, or will hold the clean fork for this edge in finite time.

Proof: See above discussion.

Theorem 3: A hungry philosopher will eat in finite time.

Proof: Let H^* be the graph H at the point in computation when the given philosopher is hungry. (H^* is a specific, static graph referring to a particular point in the computation whereas H is a dynamic graph which changes with transmission of forks.) We shall show that all philosophers who are hungry at this point in the computation will eat in finite time.

We define the *height* of a vertex as follows: the height of a vertex with no outgoing edges in H^* is 0; the height of a vertex with outgoing edges is the length of the longest path from it to a vertex of height 0, where length is defined as the number of edges along the path.

We show by induction on k that a hungry philosopher at height k will eat in finite time. This is certainly true of hungry philosophers at height 0 since they have only incoming edges and lemma 2 applies for each incident edge. Now assume that the claim is true for philosophers at heights 0,1,..,k-1. Consider a hungry philosopher v_j at height k (k=1,2,..) and a neighboring philosopher v_i. If (v_i,v_j) is an edge then v_j holds or will hold the clean fork for this edge, from lemma 2. If (v_j,v_i) is an edge, either v_j holds the (dirty) fork for this edge or v_i is hungry (since all edges incident on thinkers and eaters are directed away from them). If v_i is hungry then he eats in finite time, according to the induction hypothesis. Since v_i must stop eating in finite time, the forks for all edges incident on v_i become dirty and free in finite time. Therefore v_j will receive (a clean) fork from v_i in finite time. No clean fork that v_j holds will be released until v_j eats. Hence v_j holds all forks in finite time and eats.

Efficiency of the Algorithm

A thinking philosopher who has M neighbors and F (dirty) forks, on becoming hungry, must send M-F requests and receive a fork corresponding to each request; in addition, in the worst case he may lose all F forks he held initially and therefore have to request and receive them. In the latter case, the philosopher may send the fork and the request for it in one message. Therefore no more than 2M messages are needed for entering one eating state. In the best case, a philosopher may receive no requests for forks and therefore he may live his life (think and eat) free of interaction with others.

A Solution to the Distributed Drinking Philosophers' Problem

The drinking philosophers' problem is a significant generalization of the distributed dining philosophers' problem. We seek a distributed non-probabilistic solution satisfying the same conditions - *fairness, symmetry, economy, concurrency* - as for the distributed dining philosophers' problem. Every solution to the drinking problem is a solution to the dining problem, though the converse is not true. In particular, our solution to the distributed dining philosophers' problem does not solve the drinking philosophers' problem. For example, suppose two or more philosophers are arranged in a ring each with two incident edges, *left* and *right* , and all of them are now drinking with their left bottles. If they all require *both* bottles for their next drinking session, then our dining philosophers' solution results in a deadlock. The reason for deadlock is that the deadlocked state is *symmetric*, because the philosophers are arranged in a ring and each holds his left bottle. The system can leave a symmetric state only by resorting to probabilistic decision making. Since we seek non-probabilistic algorithms, we must prevent the system from entering symmetric states. However, it is certainly feasible for all philosophers sitting in a ring to hold their left bottles. If we were to disallow this state we would be disallowing a feasible state merely to solve our problem; disallowing feasible states violates our constraint of *concurrency*. We appear to be in a quandary because the constraints of symmetric processes, non-probabilistic solutions and concurrency are incompatible. We resolve this quandary by introducing artificial indivisible resources and ensuring that *every state that the system enters is asymmetric with respect to the artificial resources* though the state may be symmetric with respect to the genuine resources (viz. bottles). We shall ensure that the sharing of artificial resources is such that long-term symmetry with respect to the artificial resources (and genuine resources as well) is achieved despite inherent short-term asymmetry.

The artificial resource we introduce are forks in the distributed dining philosophers' problem. We have a solution to the dining philosophers' problem which ensures that forks are shared in a fair manner. We shall use the locations of forks to resolve conflicts for bottles. Our philosopher can eat and drink *simultaneously* and we emphasize that *eating is an artifact of our solution*, used only to guarantee fair drinking. In our solution, the state of a philosopher is a pair (dining philosopher's state, drinking philosopher's state), where a dining philosopher's state is one of thinking, hungry and eating and a drinking philosopher's state is one of tranquil, thirsty and drinking. Our next step is to define the dining characteristics of our philosophers: the drinking characteristics are specified by the problem. We shall give rules for dining which ensure that all thirsty philosophers drink in finite time.

Consider the state transitions of a dining philosopher. The only transitions that are decided by the philosopher are thinking-to-hungry and eating-to-thinking; the only transition completely specified by the dining philosophers' problem is hungry-to-eating (which occurs when the philosopher holds all forks he needs). We will now give rules for the dining philosopher to decide the point of the first two transitions.

D1 (*Thinking-to-Hungry Transition*)

A thinking philosopher becomes hungry on becoming thirsty. A philosopher cannot stop thinking until he becomes thirsty.

D2 (*Eating-to-Thinking Transition*)

An eating non-thirsty philosopher starts thinking. A philosopher cannot stop eating as long as he is thirsty.

This solution requires a philosopher to check his mailbox while eating because a thirsty, eating philosopher will never get to drink (and thus terminate eating) unless he checks his mail and get the bottles he desires. In the distributed dining philosophers' problem, a philosopher can think for arbitrary time though he must eat for finite time. Therefore our obligation, arising out of rules (D1) and (D2), is to ensure that each eating period is finite. This can be accomplished (see lemma 4) by (D3) given below. Note that a philosopher cannot give up a bottle from which he is currently drinking; thus analogous to *free forks*, we define a bottle to be *free* if the philosopher holding it is not drinking from it.

D3 (*Rule of Bottle Transmission*)

Philosopher v_i sends a bottle he holds to v_j in response to a request from v_j, if and only if the bottle is free and (v_i does not need the bottle or v_i does not hold the fork for the edge $\{v_i,v_j\}$).

These rules lead to the following algorithm for philosopher v_i.

Algorithm for v_i

(D0) If *thirsty* :: Send requests for all needed bottles that v_i does not hold, (and for which requests have not already been sent since last becoming thirsty).

(D1) If *thirsty* and *thinking* ::
Become hungry. {Take action appropriate to a dining philosopher when he becomes hungry.}

(D2) If *not thirsty* and *eating* ::
Stop eating and transit to thinking state.

(D3) If *there is a request from a philosopher v_j for a bottle* and *the bottle is free* and *(the bottle is not needed by v_i* or *v_i does not hold the fork for the edge $\{v_i,v_j\}$)*::
Send bottle to v_j.

(D4) If *thirsty* and *holding all needed bottles*::
Drink. On completion of drinking become tranquil. {At this point all bottles held by v_i are free and from rule D3 all pending requests for bottles will now be satisfied.}

Note: v_i must also obey the algorithm for a dining philosopher. Initial locations of bottles are irrelevant.

Proof of Correctness

We will now show that every thirsty philosopher drinks in finite time. We will use theorem 3 from the last section to show that every hungry philosopher will eat in finite time; however in order to do so we must prove that every eating period is finite. This is next proved in lemma 4.

Lemma 4: Every eating period is finite.

Proof: If v_i is eating and not thirsty, he completes eating. Assume therefore that v_i is eating and thirsty. We will show that v_i will receive every bottle he needs.

A neighbor v_j of v_i will send the bottle needed by v_i (if v_j holds it) in finite time (using D3) because drinking periods are finite and v_j does not hold the fork for $\{v_i,v_j\}$.

v_i will not release any bottle that he needs, because he holds all forks. Therefore v_i will drink in finite time and become not thirsty, thus terminating eating.

Since every eating period is finite, theorem 3 applies and we have,

Corollary 5: Every hungry philosopher starts eating in finite time.

Corollary 6: A thirsty, eating philosopher must drink in finite time.

Proof: Eating periods are finite and are terminated only by drinking.

Theorem 7: Every thirsty philosopher drinks in finite time.

Proof: When a philosopher becomes thirsty he is either thinking, hungry or eating. A (thirsty,thinking) philosopher becomes hungry in finite time (from D1); a hungry philosopher starts eating in finite time (from Corollary 6). Therefore every philosopher that remains thirsty must be eating in finite time. *The theorem follows from Corollary 6.*

Efficiency of the Algorithm

For the analysis of the algorithm, we assume that the packages a philosopher sends to a neighbor are received in the order sent. We show that a bottle can travel at most twice between two neighboring philosophers v_i and v_j before one of them drinks. Consider the two cases corresponding to whether the fork for $\{v_i,v_j\}$ is dirty or clean.

Case 1 One of the philosophers, say v_i, holds the dirty fork for the edge $\{v_i,v_j\}$: if v_i mails the bottle to v_j (in response to a request), he must also mail the fork. Therefore on receipt of the bottle, v_j will have a clean fork and hence will not mail the bottle again until he drinks. Therefore the bottle travels at most once from v_i to v_j before v_j drinks.

Case 2 v_i holds a clean fork or a clean fork has been mailed to v_i: the bottle may travel at most once from v_j to v_i and will not travel from v_i to v_j, until v_i drinks.

Lemma 8: There are at most 4d message transmissions for d drinking sessions among all philosophers.

Proof: There is at most one request (for fork and/or bottle), one transmission of a fork and two transmissions of a bottle between neighbors before one of them drinks.

References

1. Lehmann, Daniel and Rabin, Michael, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem," *Proceedings of the Eigth Annual ACM Symposium on Principles of Programming Languages*, Williamsburgh, Virginia, January 26-28, 1981.

2. Dijkstra, E. W., "Two Starvation Free Solutions of a General Exclusion Problem," EWD 625, Plataanstraat 5, 5671 AL Nuenen, The Netherlands.

3. Lynch, Nancy A., "Fast Allocation of Nearby Resources in a Distributed System," *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, Los Angeles, California, April 28-30, 1980.

4. Dijkstra, E. W., "Hierarchical Ordering of Sequential Processes," Operating Systems Techniques, Academic Press, 1972.

5. Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, Vol. 21, No. 7, July 1978, pp. 558-565.

6. Ricart, Glenn, and Agrawala, Ashok, "An Optimal Algorithm for Mutual Exclusion in Computer Networks," *Communications of the ACM*, Vol. 24, No. 1, January 1981, pp. 9-17.

7. Francez, N. and Rodeh, M., "A Distributed Abstract Data Type Implemented by a Probabilistic Communication Scheme," IBM Israel Scientific Center, TR-080, April 1980 (presented at the 21st Annual Symposium on F.O.C.S., Syracuse, NY, October 1980).

8. Hoare, C. A. R., "Communicating Sequential Processes," *Communications of the ACM*, Vol. 21, No. 8, August 1978, pp. 666-676.