FTCS-11

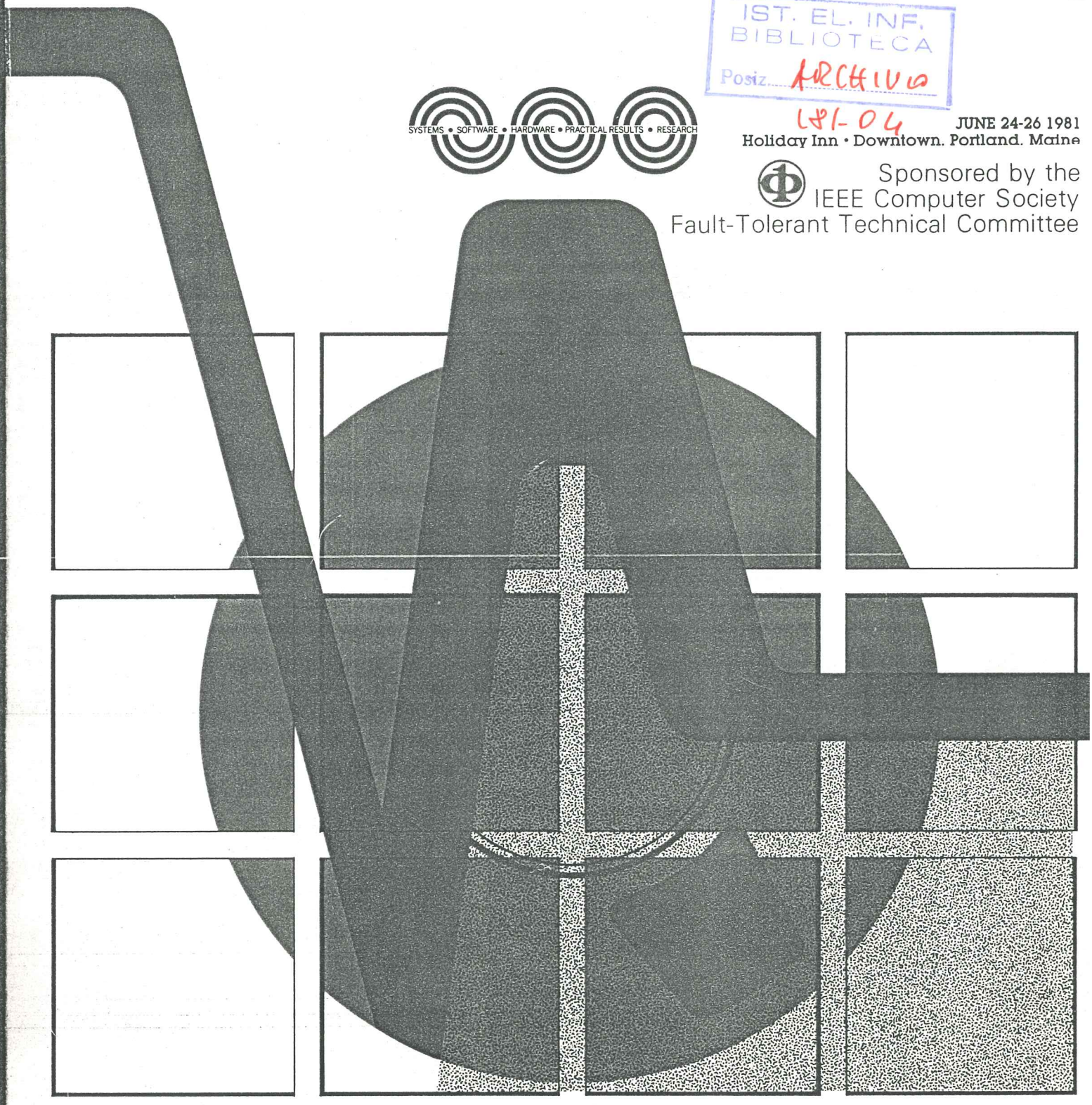
DIGEST of PAPERS

The Eleventh Annual International Symposium on Fault-Tolerant Computing

SYSTEMS • SOFTWARE • HARDWARE • PRACTICAL RESULTS • RESEARCH

JUNE 24-26 1981
Holiday Inn • Downtown, Portland, Maine

Sponsored by the
IEEE Computer Society
Fault-Tolerant Technical Committee



IEEE Catalog No. 81CH1600-6
Library of Congress No. 79-613257
Computer Society Order No. 350

IEEE COMPUTER SOCIETY PRESS

DISTRIBUTED DIAGNOSIS IN MULTIPROCESSOR SYSTEMS: THE MuTEAM APPROACH

P. Ciompi, F. Grandoni, L. Simoncini

Istituto di Elaborazione dell'Informazione
Consiglio Nazionale delle Ricerche, Pisa, Italy.

ABSTRACT

Abstract models of self-diagnostic techniques are well known, and established theoretical results are present in the literature.
Multiprocessor systems are naturally suited for the application of these techniques but no design is currently reported which embeds these techniques. This paper deals with the problems arising in the implementation of self-diagnostic techniques in a multimicroprocessor environment. A diagnostic procedure is proposed and evaluated.

INTRODUCTION

The main requirements of the MuTEAM project, namely modularity, functional distribution, asynchronism among processes and exploitation of parallelism are reflected in the self-diagnosis policy which will be implemented in the experimental prototype.

From a general point of view of high dependability, self-diagnostic techniques constitute a valid alternative to other methods, particularly in systems composed of off-the-shelf components, since they can be implemented at a sufficiently high level of abstraction and therefore be considered independent of the technologies used. They are used for a periodic diagnosis as an easy maintenance aid, to pinpoint errors caused by latent faults to avoid a possible catastrophic failure of the system caused by the superimposition of many undetected faults. Moreover they can be considered an aid to the on-line fault detection and diagnosis by organizing them in a user transparent way (concept of concurrent diagnosis[1]) without the need of interrupting the normal computation of the system for starting the diagnostic phase. Finally, they can be fitted to degraded configurations of the system without a complete loss of diagnostic power.

PROBLEMS IN THE IMPLEMENTATION OF SELF DIAGNOSTIC PROCEDURES

Several abstract models of self-diagnosis have been proposed, and established theoretical results are present in the literature[2,3]. All the proposed models rely on the outline of the diagnostic characteristics of a system through a diagnostic graph. In such a graph, in the models which are more investigated, the nodes represent the units u_i of the system and the arcs the testing relations among them. An arc from u_i to u_j means that u_i is able to apply a diagnostic test to u_j and judge about its status (faulty or fault-free). Moreover the meaning of an arc from u_i to u_j is that the test result and judgment can be only influenced by malfunctions in u_i and u_j and not by the malfunction of another unit. By decoding the ordered set of test results (syndrome), the actual faulty units in the system can be identified.

These abstract models, by their nature, rely on simple hypotheses which determine several problems on the applicability in real systems.

Elimination of hard-core. An actual self diagnostic procedure is organized in a temporal succession of phases and has to run in a potentially faulty environment. The solution of dedicating a centralized unit, like a support processor[4], in our opinion has to be avoided since it introduces a singular point which has to constitute the hard-core of the system and which has been shown[5] to be highly dangerous in a distributed system.

Therefore we consider that the general organization of a diagnostic procedure is based on a set of asynchronous cooperating processes, all at the same level, without any master-slave relation among them, and without any centralized hard-core.

On this basis each unit of the multiprocessor system will run a diagnostic process; on its completion all the fault-free units will reach the same conclusions about the faulty ones and about the overall status of the system.

Phases of a diagnostic process. Several phases can be identified in a diagnostic process: they can be described as follows:
a) Initialization: the diagnostic process is ini-

CH1600-6/81/0000/0025$00.75

tialized either on error detection signalling or on an external signalling for periodic maintenance;

b) Test execution: the unit judges the status (faulty or fault-free) of the units which it tests;

c) Syndrome decoding: the unit acquires the test results of all the other units, decodes the syndrome and identifies the faulty units in the system.

At this point, all the fault free units can agree on a common action for the reconfiguration and the recovery of the system.

Since this is the organization of the proposed diagnostic process, the problems, which arise for the actual implementation in a multiprocessor environment, derive both from the need of organizing independent and cooperating processes in a possibly faulty environment and from the requirements of the self-diagnostic procedure.

Autonomy of tested units. During the test execution it is required that no unit u_i can act as a master for the tested unit u_j. This is for avoiding that a malfunction of u_i makes definitely slave the fault-free u_j. Should this situation happen, other units testing u_j could judge it as faulty, bringing to an incorrect diagnosis or the overall procedure could not terminate since the fault free u_j could not be able to make its judgements available to the other units.

Therefore it is required that each unit maintains its autonomy even during the test execution by basing any interaction among units on explicit requests of actions and explicit acknowledgments of execution of the requested actions.

Deadlocks. Another problem derives from the fact that each unit, during the procedure, has to take both the statuses of "testing" and "under test" and by the fact that the diagnostic graph contains at least one closed loop. Since these statuses are defined by the dialog among the units, which dynamically and consistently identify what units are "testing" and what units are "under test", deadlock situations must be avoided in which the units in the diagnostic loop are either in the same status or oscillate between the two possible statuses for an indefinitely long time.

Status contamination. The diagnostic process requires that the judgement of u_i about the status of u_j is not invalidated by any other unit. This determines the need of protected communication channels between units, to avoid the situation in which a faulty unit u_k makes the fault-free unit u_j to appear as faulty to the testing fault-free unit u_i, or the same faulty unit u_k makes the faulty unit u_j to appear as fault-free to the testing fault-free unit u_i.

Syndrome decoding. For what concerns the decoding of the syndrome, in the first step, the unit u_i has to acquire the test results from all the other units. It is possible that either a faulty unit u_j does not make available its test results to u_i or that, for the asynchronism among the diagnostic processes, a fault-free unit u_k is not yet ready to make its results available to u_i. This introduces the problem of the correct decoding of an incomplete syndrome.

Moreover for the decoding of the syndrome, as previously mentioned, we want to avoid a validation policy which is not distributed (like a NMR with centralized voting), and we prefer to rely on a "consensus" policy[6] in which the agreement and the validation is implicitly reached on the a priori knowledge that all the fault free units will behave in the same consistent way. The use of this "consensus" policy introduces the problem of the synchronization among the fault-free units to be able to proceed consistently to the operative actions successive to the diagnosis.

In the next section the structuring of the diagnostic processes which embed the proposed solutions to these problems is presented and the influence on the architecture of a multiprocessor system is pointed out.

IMPLEMENTATION OF DIAGNOSTIC PROCESSES AND INFLUENCE ON THE ARCHITECTURE

General algorithm. Let us consider the flow diagram of a diagnostic process, as outlined in Fig. 1.

Fig. 1 shows the outline of the diagnostic process which runs on the unit u_i which has to test units $u_{k1} \ldots u_{kj} \ldots u_{ks}$ and has to be tested by $u_{m1} \ldots u_{ml} \ldots u_{mr}$.

j(1) is the current index of the tested (testing) units. In the block a), the presence of legal requests from some of the units $u_{m1} \ldots u_{mr}$ is examined. If some request is pending, one of them (let us say from u_{ml}) is accepted. u_i sends an acknowledge to u_{ml} and a busy to all the others (block b). The test from u_{ml} on u_i is executed

(block t) and after that a not busy is sent from u_i to $u_{ml} \ldots u_{mr}$ (block v). If no request is pending, u_i starts the scan of the unit u_{kj} which have not yet been tested (block c). If u_{kj} is not busy (block d), u_i sends a request to it and a busy to $u_{ml} \ldots u_{mr}$ (block e). u_{kj} last answer to the request from u_i with an acknowledge or a busy in a finite time which can be determined. This basic conversation is used to avoid the master-slave relation among the units.
A time-out (block f) will evidence a malfunction of the units during this basic conversation. If a time out is detected, this is recorded by u_i (block w) for use in the termination rule and a not busy is sent to $u_{ml} \ldots u_{mr}$ (block y). If u_{kj} answers with an acknowledge (block g), u_i executes the test of u_{kj} (block h). On completion of the test, u_i sends a not busy to $u_{ml} \ldots u_{mr}$ (block n) to reposition itself at the starting point of the process and controls if the overall procedure is terminated (block o). This termination rule will be examined in detail later. If u_{kj} answers with a busy (block g), a potential deadlock is present since as pointed out in the previous section, all the units in the loop of the diagnostic graph may have found out that the successors are available to be tested, they may have sent their request and all have got a busy answer. The solutions to this problem, which are present in the literature[7,8], are based on either a unique mutual exclusion semaphore or a unique control procedure which avoids the closure of the loop of requests and answers. In any case the presence of such singular point is not suitable in a potentially faulty environment and therefore another solution is proposed. This solution has been studied with relation to the avoidance of conflicts in computer networks[9]: u_i clears its request and send a not busy to all the units $u_{ml} \ldots u_{mr}$ (block p) and after it introduces a random delay which has not to be correlated among the units, before continuing the process loop (block q).

This solution does not insure the deterministic avoidance of deadlock situations but statistics can be derived which relate the deadlock duration with the characteristics of the random delay. After the insertion of the random delay, the termination rule is controlled and if not matched (block o) the process loop is reinitiated. If the termination rule is matched, unit u_i executes its diagnosis, for identifying the faulty units (block z).

Termination. Several termination rules can be conceived; they have large influence on the complexity and on the performance of the overall procedure.

The logically simplest termination rule is based on the use of a time-out. If the maximal number of operations and the maximal waiting time due to conflicts can be determined, the maximal time for a fault free unit to terminate its test and to make its test results available can be determined. Therefore a time-out could be inserted for each unit to signal the termination of the procedure. This is very difficult in the proposed diagnostic process due to the insertion of random delays.

Another termination rule is based on a modification of the basic diagnostic model. Redundant tests are executed in such a way that the diagnostic graph is t-fault diagnosable even if any t units out of n units do not answer. The termination rule becomes the following: each unit waits until the results from n-t units are available and starts the decoding of the syndrome. The termination is guaranteed by the fact that at most t units, the faulty ones, may not make available their results. This solution may be inefficient due to the high number of tests which have to be executed to insure the correct degree of diagnosability.

A more efficient termination rule is based on the procedure outlined in Fig. 2, which does not require redundant tests.
It substitutes the block o) and z) of Fig. 1. The unit u_i starts the syndrome decoding as soon as n-t other units have made their results available. If there is at least a fault free unit among the units which have not made their results available or more than t units are diagnosed as faulty, the procedure goes back to the main loop of the diagnostic process, since u_i has to wait for more test results and it may be that u_i has to be tested by this unit which has not yet made its results available. Otherwise the diagnostic process ends. In other words, u_i tries to decode a partial syndrome to verify if fault-free units are present among those ones which have not made available their results. If no such unit exists, the syndrome is considered as complete and its decoding correct.
The hypotheses on which this procedure relies are:
a) the fault-free units will make their results available to the others;
b) the syndrome decoding is executed supposing

that the units, which have not yet made available their test results, judge the tested units as fault free, that is fixing the not available test results to a predetermined value;

c) t faulty units are at most present in the system and they can all be identified on completion of the diagnostic process.

In[12] it is formally shown that, under the previous hypotheses it is always possible to recognize the existence (if present) of fault-free units among the set of units which have not made available their test results, and that the diagnostic process correctly terminates.

Communication channels. Communication channels must be provided for the diagnostic procedure.

In particular: a) a communication channel is required between u_i and u_j, able to support the test request, and all the successive communications during the test; b) a communication channel is required between u_j and u_i to allow the busy and acknowledge signalling; c) a communication channel is required among u_i and all the other units, through which u_i makes available its test results to them.

As pointed out in the previous Section, these channels among the units must be protected to avoid the invalidation by another unit on the judgment that a unit takes. This requirement determines a strong influence on the architecture of a multiprocessor system. A physical point to point connection among the units can provide the required communication channels as well as the necessary protection among the communication channels. However this solution is expensive and not sufficiently flexible. A more suitable approach may be based on the use of a common bus with shared memory. This organization provides the necessary communication channels, as specified in the previous a), b), c) points but does not provide the required protection. A possible solution is based on the use of a logically segmented shared memory. The accesses to the memory segments are explicitly enabled by a protection unit which controls the access rights which are assigned to the several units u_i. Possible solutions are shown in Fig. 3.

In Fig. 3a) both the protection units and the shared memory are centralized. Both the protection unit and the shared memory must be considered hard-core and therefore this organization is not consistent with the overall approach we have tried to implement. A different solution is presented in Fig. 3b) in which the protection unit is partitioned among the u'_is. Even in this case a fault in one of the protection units may allow the arbitrary access from a unit u_i to the shared memory which even in this case has to be considered hard-core. A more suitable solution is presented in Fig. 3c) where even the shared memory is partitioned in banks protected by their own protection units. Each protection unit is controlled by the associated unit u_i. A fault in a protection unit can influence only the memory bank which is associated to it and cannot influence all the shared memory.

The solution outlined in Fig. 3c) is suitable even during the normal functioning of the system if the communication channels have to be dynamically assigned and protected. In this case the problem of the management of the assignments arises. In Fig. 3c) the dynamic management is in charge of the unit u_i, which, if malfunctioning, can influence only the protection unit which is controlled by it. This organization is referred in the literature as access control list[10], and is based on the concept that, in a controlled flow of information, all the information after a faulty controller becomes unreliable; in other words, with the organization in Fig. 3c), the risk of having such a situation is partitioned, in the system. This protection, compared with that outlined in Fig. 3b), is more complex. In fact, in Fig. 3b), each protection unit knows the name of the unit u_i which tries to access it, while in Fig. 3c) this name has to be transmitted explicitily through the bus.

CONCLUSION

The diagnostic procedure will be implemented and experimented in the experimental MuTEAM prototype.

The diagnostic procedure will be described using a CSP based language, and by a suitable specification of the operating system kernel primitives which are necessary for the correct and efficient support of the diagnostic procedure.
The experimentation on MuTEAM will deal basically with the efficiency of such procedure implemented at a rather high level of abstraction and will not be focused on the aspects of completeness and complexity of the test of the units. These aspects which are related to the design of easily testable units, may be less relevant in the near future by the introduction of VLSI custom chips which will embed testing facilities and features.

REFERENCES

/1/ L. Simoncini, F. Saheban, A.D. Friedman, "Design of Self-Diagnosable Multiprocessor Systems with Concurrent Computation and Diagnosis", IEEE Trans. on Computer, Vol.

C-29, n° 6, June 1980, pp. 540-546.

/2/ F. Preparata, G. Metze, R.T. Chien, "On the Connection Assignment Problem of Diagnosable Systems", IEEE Trans. on Computers, Vol. EC-16, n° 6, pp. 848-854, Dec. 1967.

/3/ A.D. Friedman, L. Simoncini, "System Level Fault Diagnosis", Computer, Vol. 13, n° 3, March 1980, pp. 47-53.

/4/ D.J. Kushier, D.R. Mueller, "Support Processor Based System Fault Recovery", Proc. FTCS-10, Kyoto, Japan, Oct. 1-3, 1980, pp. 297-301.

/5/ A.K. Jones, P. Schwarz, "Experiences Using Multiprocessor Systems. A Status Report", ACM Comp. Surveys, 12, 2, June 1980, pp. 121-165.

/6/ D. Katsuki et at., "Pluribus-An Operational Fault-Tolerant Multiprocessor", Proc. IEEE, Vol. 66, n° 10, October 1978, pp. 1146-1159.

/7/ E.W. Dijkstra, "Hierarchical Ordering of Sequential Processes", in Operating System Techniques, Academic Press, London 1972.

/8/ C.A.R. Hoare, "Communicating Sequential Processes", CACM, Vol. 21, n° 8, Aug. 1978, pp. 668-679.

/9/ R. Binder, "ALOHA Packet Broad-Casting - a Retrospect", AFIPS Conf. Proc., Vol. 44, 1975, NCC, p. 203.

/10/ J.H. Saltzer, M.D. Schroeder, "The Protection of Information in Computer Systems", Proc. IEEE 63, n° 9, Sept. 1975, pp. 1278-1308.

/11/ S.L. Hakimi, A.T. Amin, "Characterization of Connection Assignment of Diagnosable Systems", IEEE Trans. on Computers, Vol. C-23, n° 1, Jan. 1974, pp. 86-88.

/12/ P. Ciompi, F. Grandoni, L. Simoncini, "The MuTEAM Fault Treatment: a Proposal for Self-Diagnosis in Multimicroprocessor Systems", Techn. Rep., MUMICRO Series, June 1981.

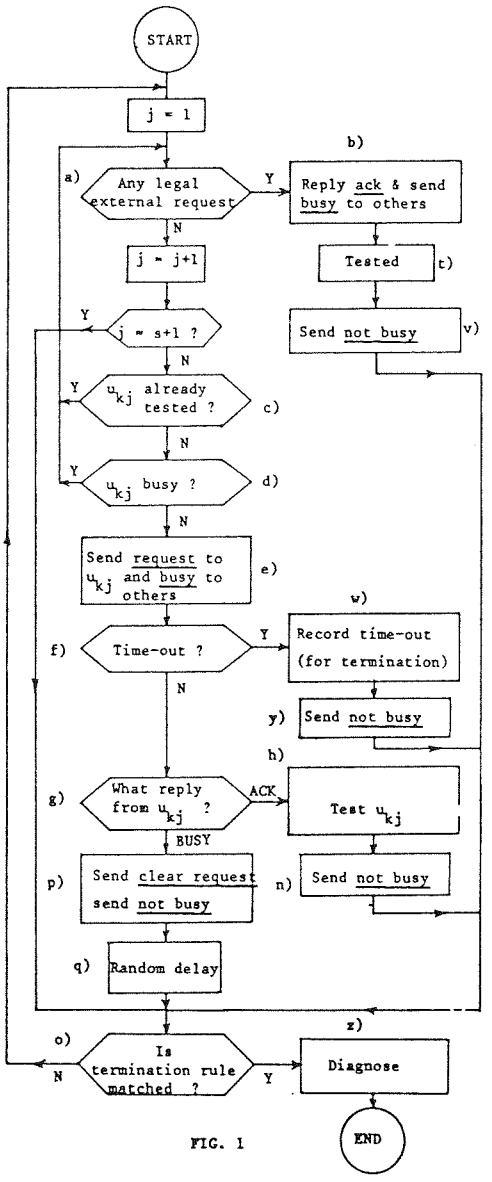


FIG. 1

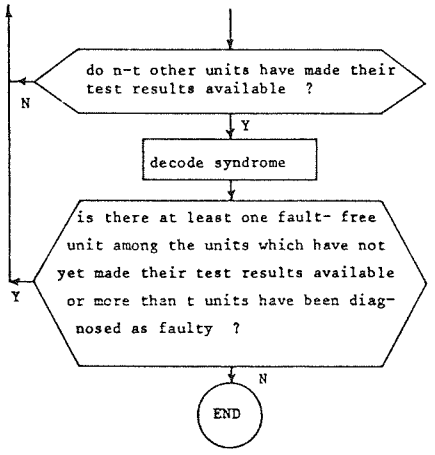


FIG. 2

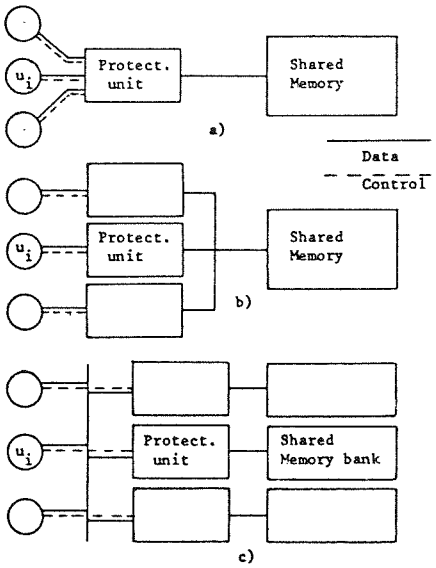


FIG. 3