MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Project MAC

Machine Structures Group
Memo No. 7

Memorandum MAC-M-188
October 1964

AUTOMATIC SCHEDULING OF PRIORITY PROCESSES

by

J. B. Dennis

An increase of several orders of magnitude in the frequency of allocation and scheduling events will be required in future MAC systems. In this light it is appropriate to consider implementing some of the more critical and heavily burdened executive functions with special system hardware. One candidate is the assignment of processors to processing units, and is the subject of this note.

Our objective is to make the switching of attention of a processing unit from one task to another sufficiently efficient that strong interaction between user programs and external events (e.g., light pen tracking), and the inclusion of supervisory tasks in the same operating scheme as user tasks is reasonable. The proposeal made in this note could be implemented on present day multi-processor hardware so as to realize switching time on the order of ten to one hundred microseconds depending on the degree of elaboration.

In designing machine features to perform such an executive function, one must be careful not to limit the flexibility of process scheduling that is possible through software implementation. It is felt by the writer,

that the scheme proposed can perform rational scheduling on a time scale that could not be approached through a programmed procedure. However, it is only intended that the proposed mechanism be used to implement scheduling of processes at the highly dynamic end of the spectrum, (i.e., at the highest levels of a Corbato' multi-level queue)[1]. Further, the mechanism would only apply to processes which have vestiges of their data in main memory. The dumping and retrieval of information, and the handling of the lower levels of the process queue are presumed to be accomplished by supervisory processes, themselves scheduled by the mechanism to be described.

We assume the reader is acquainted with the concepts of memory segmentation outlined in MAC TR-11[2]. As discussed by Dennis and Glaser[3], we suppose that a definite number of i/o functions are defined for the multiprocessor computer system under consideration. These i/o functions are invoked by procedure steps we shall call i/o instructions. Here we will adopt a more common termonology and say that a process is dormant during execution of an i/o function. Parallel performance of i/o functions and computations are presumed to be accomplished through the use of fork meta-instructions to initiate independent processes and quit meta-instructions to terminate processes. (See Witzenhausen [4]).

We assume that a process can have only one associated i/o function in progress at a time (i.e., parallel processing is achieved through forking), and an i/o function is not called by any other process while a previous call is in progress. Under these assumptions all queuing in the system takes the form of active processes awaiting execution, and dormant processes awaiting completion of i/o functions.

The State Word of a Process (Processor)

The state word of a process is the information that must be placed in the registers and indicators of a processor to place a process in execution. As shown in Figure 1, the state word includes the contents of processor arithmetic logical and index registers (general registers), the states of modal and conditional indicators, and the address word (attachment tag and word address) of the next procedure step to be executed. The state word must also include data specifying the sphere of protection within which the process operates and the set of system names (segment name, i/o functions names, and sphere names) currently "attached" to the process. It is evident that switching a processor from one process to another involves solely a substitution of state words.

Process Priority

Let there be n levels of priority numbered 0,1, ...,n-1 where one process has precedence over another if the first has the higher priority number. A process has an initial priority determined by the event that created the process (i/o function completion, fork, or return from a sphere entry as discussed below). The priority of a process decreases as execution time accumulates according to the following rules. Let a quantum be the unit by which execution time is measured. Then a process assumes priority K-1 after execution for 2^{n-k} quanta with priority k. (This can be implemented quite easily by a mechanism described later in this note).

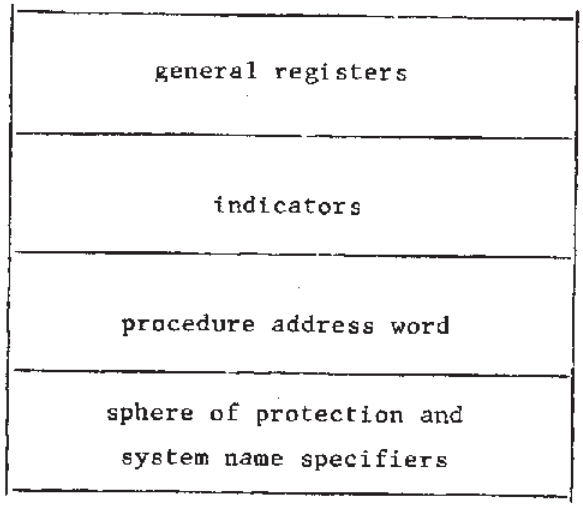


Figure 1 The process state word.

The priority of a processor is the priority of the process it is executing.

The Process List

The process list is made up of process entries belonging to five subsets called the free list, the execution set, the dormant set, the dead set, and the queue list. Each process entry contains space for a process state word, two pairs of pointers that link related entries, and two indicators - lock out and deactivate. One pair of pointers links all queue list entries of the same priority to the corresponding entry of a queue priority table. The other pair of pointers links all process list entries belonging to a particular sphere of protection to the descriptor of the sphere. The use of the indicators will be discussed when we introduce a more specific implementation. The structure of the process list is shown in Fig. 2. The free list is a set of vacant process entries linked to a word at a fixed location, called the free list tie. The execution set contains process entries for those processes currently being executed by processors of the system. The state word of each processor came from a process entry of the execution set, and will be returned there any time the processor is preemted for a distinct process, through the mechanism to be described. The dormant set consists of process entries that retain the states of processes that have encountered i/o pauses. The dead set contains process entries that represent the states of processes that have called for action by protected service

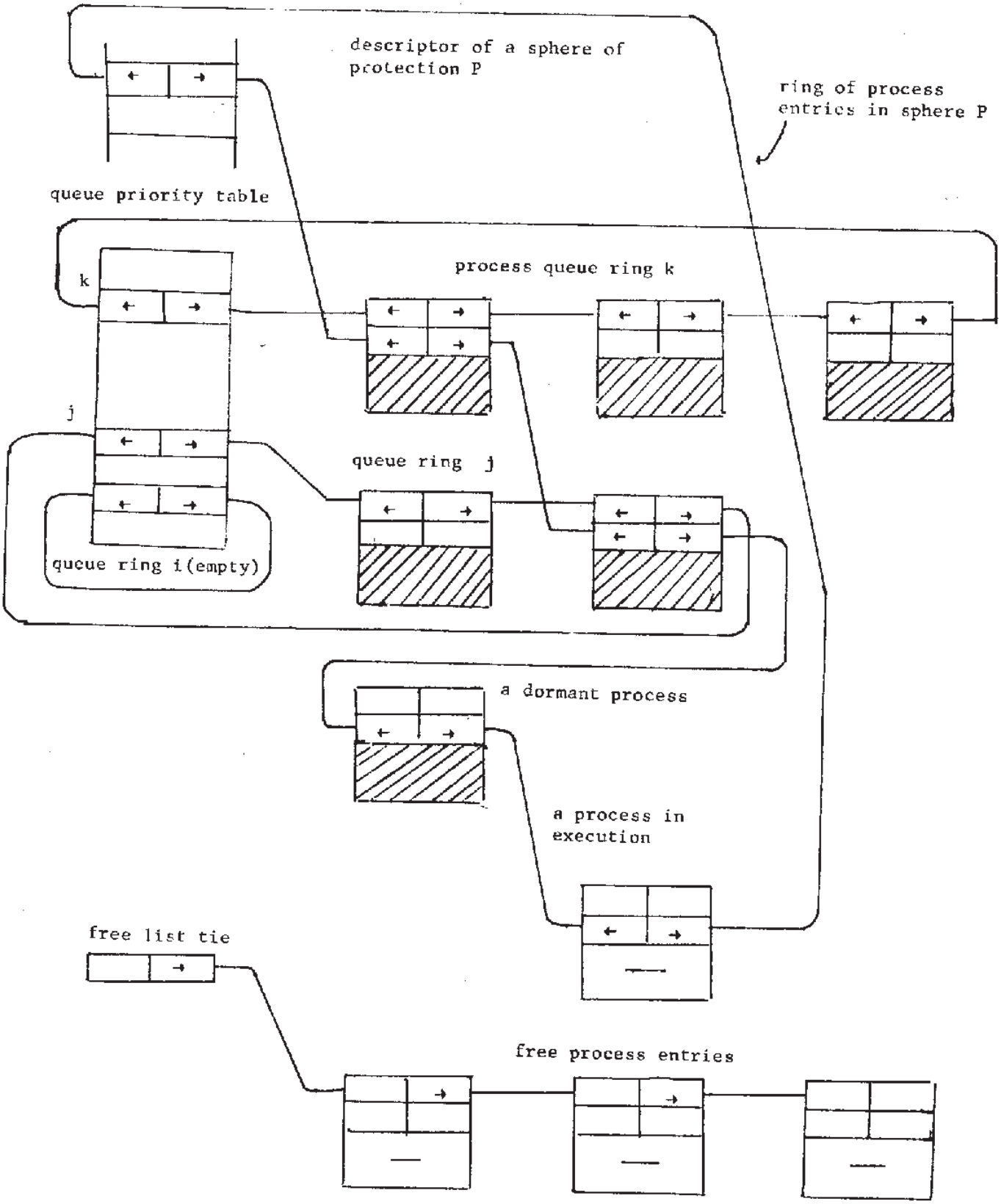


Fig. 2 Structure of a process list.

routines, or have been terminated by exceptional conditions. The queue list consists of a linked ring of process entries for each distinct priority number. Each ring may be empty, or contain an ordered set of state words of processes available for execution. In the latter instance, one of the pair of entries in the ring that are linked to the queue priority table is designated as the tail of the ring, and the other, the head. Any state of the queue list has an associated queue priority equal to the priority of the highest occupied ring.

i/o Function Table

The i/o function table contains an entry for each distinct i/o function as shown in Fig. 3. The table is indexed by i/o function number, the hardware designator by which the i/o function is called into operation. Each entry of the table includes:

1) a busy indicator

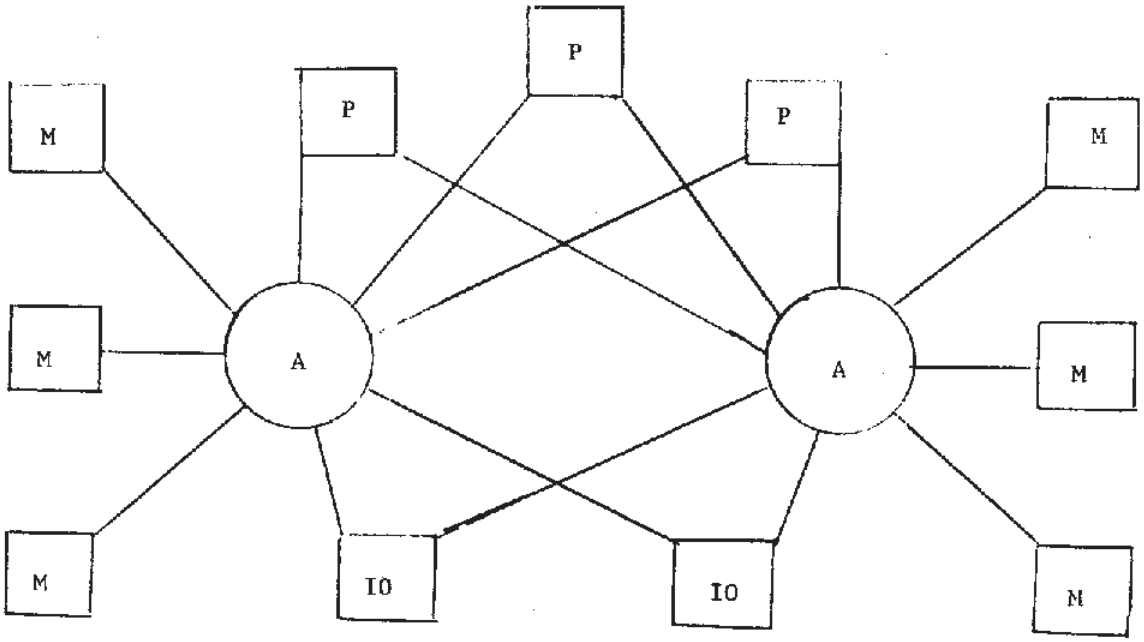
2) a pointer to an entry of the process list

3) a priority number.

The i/o function number forms part of the descriptor of an i/o function name when it is "attached" to a process.

Execution of i/o Functions

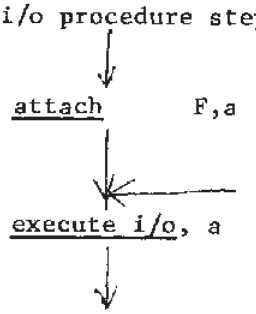
To perform an i/o function, a process must first "attach the corresponding i/o function name by the use of an attach meta-instruction.



A	arbiter
P	processor
M	memory
IO	I/O controller

Figure 3 Modular computer system

Once this has been done the i/o function has been deemed valid in the effective sphere of protection and the attachment tag has an associated descriptor that contains the appropriate i/o function number. If F is an i/o function name and a is an attachment tag, programming and i/o procedure step would be as follows:



The effect of the execute i/o instruction is;

1) The i/o function number of the descriptor in attachment register a is used to index the i/o function table.

2) The state word of process is entered into its associated process entry in the execution set, and the entry is made dormant.

3) A pointer to the newly created dormant process entry is placed in the device table entry and the busy indicator is set. (If the busy indicator was already set an i/o synchronizing exceptional condition has occurred.)

4) The i/o function is initiated.

5) The processor is free to pick up the top process in the queue list.

i/o Compeletion - Activation of Dormant Processes

Upon completion of an i/o function, the corresponding dormant process entry must be placed in the process queue at the appropriate priority level. This involves only a few steps.

1) The pointer to the dormant process entry and the priority number n are read from the i/o function table.
2) The process list pointers are modified to place the dormant process entry at the tail of level n of the process queue.
3) If the queue priority is less than n it is set to n.
4) The busy indicator for the i/o function is reset.

Process Switching by Processors

Each processor continually compares its own priority number with the process queue priority that indicates the priority number of the highest priority process in the process queue. Unless the latter number is higher, execution of the current process proceeds normally.

If it is higher, the processor state word is dumped into the corresponding process entry, of the execution set and the entry is placed in the queue list ring for the present priority number of the process. The processor is now free to take up the top process in the queue list by loading its state word and adjusting the queue list pointers to place the process entry in the execution set. The queue priority must be updated if necessary. If a high priority process enters the queue, an arrangement must be made to insure that the processor of lowest priority is the one interrupted. Otherwise needless process swapping would occur.

Forks and Quits

A fork meta instruction causes a process entry to be made in the

process queue from an entry in the free list with priority equal to that of the process executing the fork. A quit meta instruction leaves the processor free to pick up the top member of the process queue as above returning the execution set process entry to the free list. In these cases, it is necessary to adjust the pointers so that process entries pertaining to a particular sphere of protection remain correctly linked. These connections are necessary so that the system can be "aware" of all processes operating in a particular sphere of protection and can delete them if requested to do so.

Comments

Several essential features have been omitted from the scheme described in this note for the sake of simplicity:

1) Since different system entries will need to reference the process list occassionally, some form of lockout is necessary to resolve race conditions among those entities. The lockout indicators are included for this purpose.

2) In some applications it will be necessary to permit a processor to execute a single process without interruption for an extended period.

3) For reliability it should be possible to implement several concurrently operating queues in a large system.

The latter considerations are satisfied through system partitioning as described at the end of this note.

Hardware Implementation

We suppose the computer system contains modules of four classes.

1) processors

2) memories

3) arbiters (system controllers)

4) i/o controllers.

We suppose there would be many units of each class in an actual system and demand an implementation with general application. Communication among these modules is assumed to be only as indicated in Fig. 3. Note that the only physical links to a processor are those it has to the arbiters. One of the functions of the arbiter is to resolve conflicts that arise when two or more processors address the same memory module. The arbiter also performs the synchronizing and decision functions required to implement the scheduling system outlined above.

The i/o controllers also communicate with memory solely through the arbiter modules. All signalling of initiation and completion of i/o functions must be done through the arbiters. Therefore it is natural to make the arbiter the focal point for implementation of a process switching scheme. We suppose there may be a process list associated with each arbiter of a system. Since modification of the process list can be accomplished by processors and i/o controllers through the normal memory access mechanism, the special features required in the arbiter are:

1) A means of recognizing when a processor is to be switched from one process to another and signalling the processor to take action.

2) A means of signalling the start of i/o functions to i/o controllers and transmitting to the i/o controllers the codes of the functions.

The occurrence of process switching is dependent on the priorities of processes currently being executed by processing units, and the priority of the top entry in each process queue. Therefore logic levels representing these data must be continually available to the arbiter.

To provide these levels, each processor has a priority counter (PC) of $2^k - 1$ bits, the contents of which determines a k-bit processor priority number (PP) equal to the number of adjacent ones preceeding the first zero in scanning from left. The PC is decremented by one for every N memory reference by the process being executed, N defining the quantum of process execution time. Decrementing stops when the PC reaches zero. Each arbiter has a k-bit queue priority register (QP) that contains the priority number of the top process entry in the associated process queue.

Description of Operation

Normally the decision logic of an arbiter module is in the quiescent state with its queue priority register QP containing the priority number of the top entry in the associated process queue. The decision logic of an arbiter compares the QP with the processor priority PP from each processing unit. Whenever any PP is less than QP, and the decision logic is quiescent, the process swap signal (PS) is raised to each processor for which the

difference QP - PP is maximal. When a processor is receiving a process swap signal from any arbiter and is also interruptable, it attempts to execute a process swap accept memory access to the arbiter module transmitting the PS signal. If several arbiters have raised PS signals to the same processor, the processor selects one by a simple priority rule. The arbiter's priority logic for controlling memory access will determine which of possibly several competing processors is successful in accepting the sway. At the completion of the successful process sway accept access, the decision logic is in the busy state. An unsuccessful process swap accept cycle occurs if the decision logic is found to be busy, and the processor continues execution of its current process. The processor accepting the swap enters master mode and performs the following actions by executing a normal instruction sequence.

1) stores the processor state word in the corresponding process entry in the execution set.
2) locks (by setting a lock bit indicator) the levels (PP and QP) of the process queue that must be rearranged in recording the new process list status.
3) places appropriate new priority numbers in PP and QP by means of restricted instructions:

$$QP \longrightarrow PP$$

$$K \longrightarrow QP$$

where $K \leq QP$ is the new process queue priority number.

4) returns the arbiter decision logic to the quiescent state.

5) alters the process queue pointers to record its new status.

6) unlocks the process queue levels that were locked in (2).

7) loads the processor state word from the execution set process entry created by (5).

8) leaves master mode to establish execution of the new process.

Initiation of i/o Functions

The initiation of i/o functions by execute i/o procedure steps must be arranged such that there is no possibility of interference among several processors calling for distinct i/o functions during simultaneous execution by distinct processors.

The execution of an execute i/o instruction must accomplish the following.

1) Enter the process state word as a dormant entry of a process list.

2) Obtain the i/o controller tag and the i/o function index from the descriptor associated with the attachment tag a. These data specify the i/o controller involved, and select a particular i/o function where the i/p controller handles multiple i/p functions.

3) Inform the designated i/o controller of the location of the newly created dormant process entry, and transmit to it the i/o function index.

To be consistent with the implementation of process swapping described above, step 1 above (creation of the dormant process entry).

is performed by a processor operating in master mode - with suitable locks on the process list. Step 3 requires communication between the processor and the designated i/c controller and must be mediated by an arbiter for the system structure shown in Fig. 3. To accomplish step 3 the processor initiates a start i/o memory access that presents the i/o controller tag, the i/o function index and the address of the formant entry (the dormant process pointer) to all arbiters accessible to the processor.

An arbiter with any start i/o access requests pending selects one by a simple priority scheme and raises the i/o request line to the i/o controller designated by the corresponding tag.

As soon as an i/o controller sensing any raised i/o request lines to itself becomes free, it selects one request and executes an accept i/o memory access to the corresponding arbiter. In consequence, the i/o controller becomes busy and receives from processor via the arbiter the i/o function index and the dormant process pointer. At this point the start i/o access of the processor has been completed and it is free to pick up the top entry on the process queue (via master mode programmings). The i/o controller remains busy until it is prepared to accept a subsequent i/o command. An i/o controller performing a single i/o function may remain busy until the function is completed. A multiple function i/o controller should remain busy for as short as interval as possible - presumably only long enough to permit acting upon or queueing the command.

Performing i/o Functions

An i/o controller has access to the process state word of any process executing an i/o function performed by it. Thus, it may reference information in the state word for transmission to output devices and enter information into the state word from input devices. Further, the state word, through the descriptors of attached segments, permits the i/o controller to monitor block transfers or random addressing of a segment by an i/o device.

Completion of i/o Function

Upon completion of any i/o function, and i/o controller is required to place the process entry indicated by the dormant process pointer in the process queue at the priority level specified for this i/o function, and change the queue priority QP as necessary.

System Partitioning

For flexibility and reliability it is desirable to be able to partition a large computer system into fragments each of which is capable of performing all basic system functions. Partitioning can be done physically by decisions as to which modules are interconnected, logically by programmable registers that determine which physical interconnections are allowed to be used, and functionally by only programming for subsets of permitted logical interconnections. Physical

partitioning drastically decreases the flexibility and time scale on which system resources may be shifted among different system goals. Functional partitioning is completely flexible but particularly prone to catastrophic failure in the event of programming errors or hardware failure.

Here, we propose a form of logical partitioning which is a compromise between utmost flexibility of an entirely programmed system and the complete lack of interaction that would result from physical partitioning. We establish two levels of partitioning

partitioning of memory

partitioning of control.

Memory Partitioning

Each arbiter of the system has sole control over access to a certain protion of total system memory. Partitioning is established by enabling or disabling access to an arbiter for memory references by an indicator for each processor or i/o controller connected to the arbiter.

Control Partitioning

Control partitioning is established at each arbiter by indicators that determine which processors and i/o controller participate in the manipulation of the process queue associated with the arbiter.

The partitioning indicators are set by master mode instructions (perhaps with some safeguard against accidental use).

References

1) F. J. Corbato, et.al., An Experimental Time-Sharing System, AFIPS Proceeding, Vol. 21, pp. 335-344 (National Press, Palo Alto, California 1962).

2) J. B. Dennis, Program Structure in a Multi-Access Computer, Project MAC Technical Report MAC-TR-11, (M.I.T., Cambridge, Mass., 1964).

3) J. B. Dennis and E. L. Glaser, The Structure of On-line Information Processing Systems, a paper prepared for the second Congress on the Information System Sciences (MITRE Corp., Lexington, Mass., November 1964).

4) J. Witsenhausen, A Note on Asynchronous Parallel Processing, Project MAC Memorandum AC-M-186 (M.I.T., Cambridge, Mass., 1964).