# PROCESS SELECTION IN A HIERARCHICAL OPERATING SYSTEM

R. C. Varney
The Pennsylvania State University
University Park, Pennsylvania

This paper presents a new model for use in scheduling processes for the sharing of a processor. The model may be used in various modes of operation, including multiprogramming, real time and time sharing. Because the model is applicable for various modes of operation and because it is significantly useful only within a well-defined system hierarchy, the context for discussion is Hansen's system for the RC 4000.[3,4,5]

Only the preliminary work has been reported in this paper, but the author believes that the model is worth investigating further, and, therefore, research into its effects and properties will continue.

## Time-slice queue

In Hansen's system the process heirarchy can be represented by a tree structure as in Figure 1, where each node represents a process. The allocation of space (storage) is restricted so that a child process may be allocated only a subset of the spatial resources allocated to its parent process. Processes may communicate among one another with message buffers without regard to the hierarchical structure. Allocation of processor time among processes is by equal time slices, also without regard to the hierarchical structure.

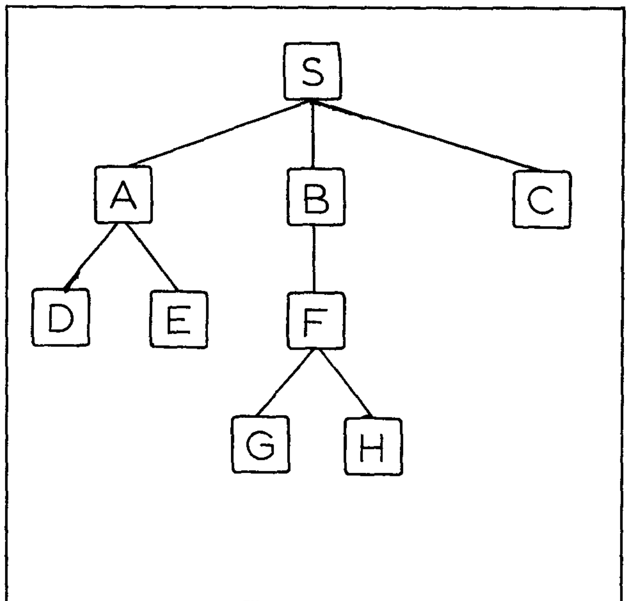

Figure 1

An active process is allocated a time slice and linked to a time-slice queue. A modified form of round-robin scheduling is used for the execution of the processes in this time-slice queue so that in general a process, even if temporarily interrupted, is allowed to use its full time slice. The details of this svstem have appeared in the literature.

## Process-selector tree

Since the processes in the system are logically related according to a tree structure, and the allocation of space is restricted according to the hierarchy and jurisdiction defined by that tree structure, it is natural to explore the consequences of allocating processor time according to the same tree structure. Let us extend the time-slice queue to a tree structure, where each node in the tree structure represents a process which has been started by its parent process but not yet stopped. (Although the RC 4000 is a single processor, the process-selector tree discussed herein is not affected by a multi-processor system.) A number of different algorithms, to be called selection algorithms, may be suggested to traverse this tree.[6] One desirable characteristic of a selection algorithm is that it be simple.

It should be noted at this point that the algorithm chosen may impose an inherent priority. The selection algorithm illustrated in Figure 3 imposes the priority that, in general, child processes will be executed before their respective parent process. One consequence of this algorithm is discussed in the section titled Global Scheduling.

Continuing, let us add a process at some level; this process (not inherently different from other processes) we will call a priority process (PP) and it will be given the task of dynamically altering the position of the nodes representing its siblings. The priority process working under the algorithm being considered, can alter the node positions in such a way as to simulate one of many scheduling algorithms for that set of siblings, including first-in-first-out, round-robin, etc. Looking further we can see that while one priority process can make a process selection algorithm appear, for example, as a round-robin scheduler, another priority process (in place of the first) can make the same selection algorithm appear as a strict priority scheduler.

In addition if processes can set the interval timer, then a priority process can take on the responsibility of allocating time-slices through the use of that interval timer. Moreover, a priority process may dynamically alter the length of the next time slice, depending on the process to be executed. If separate priority processes are used for each set of child processes, then each set of child processes at each level of the tree structure can be scheduled with a different scheduling algorithm; in fact this may be accomplished without any increase in complexity of the existing selection algorithm. And, since all processes may be created and removed, one priority process may replace another, thus dynamically altering the individual scheduling algorithms. Figure 2 depicts a tree structure where three processes perform as priority processes for their respective siblings. Interaction among all processes (including priority processes) is according to the jurisdiction specified in the overall structural design of the system.

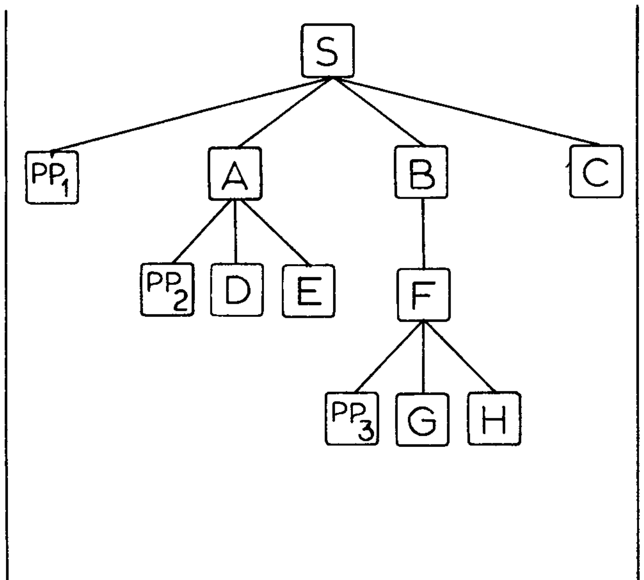

Figure 2

Because the tree structure is well-defined, a given traversal algorithm will cause the processes to be executed in some given order. If that order is satisfactory for a given set of siblings, then a priority process need not be defined for those siblings. Figure 6 contains such a case.

Using the process selector tree as defined, we see that the concern with time-slicing has been shifted to the priority processes, therefore simplifying the selection algorithm. In fact if the interrupt environment is appropriate, the entire process selection can be accomplished in the absence of all time-slicing; i.e., the system is interrupt driven and the desired scheduling can be achieved through the alteration of node order by a priority process acting with high priority after an interrupt.

Figure 3 is an algorithm which can be used for process selection using the process selector tree. Figure 4 is the binary tree representation of Figure 2 and Figure 5 is the node structure.

```
T0.  on interrupt go to interrupt process;
T1.  STACK ⇐ λ; PTR ← LLINK(TREPTR)
T2.  if PTR = λ then go to T6;
T3.  if INTUPT(PTR) then go to T5;
T4.  STACK ⇐ PTR; PTR ← LLINK(PTR); go to T2;
T5.  PTR ← RLINK(PTR); go to T2;
T6.  if STACK = λ then go to T1;
T7.  PTR ⇐ STACK;
T8.  if process(PTR) in wait state then go to
     T5;
T9.  execute process(PTR); go to T5;
```

[Note: a) TREPTR points to the top of the tree
b) λ is the empty symbol
c) ← and ⇐ are Knuth's notations for simple replacement and stack operations respectively[6]
d) STACK is a push-down stack]

Figure 3

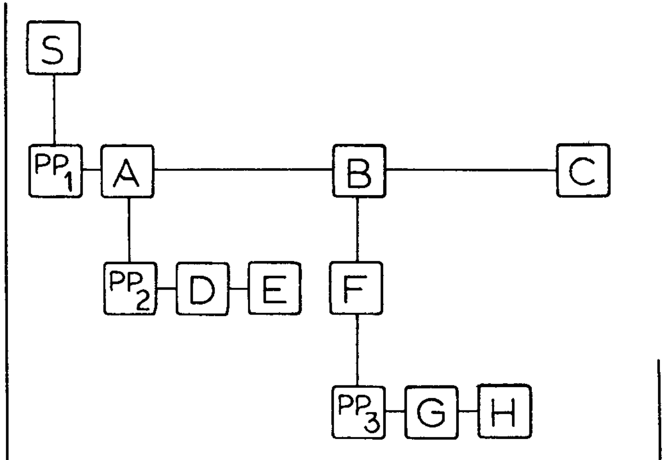

Figure 4

| node | PROCNM | INTUPT | LLINK | RLINK |
|---|---|---|---|---|

PROCNM - process name
INTUPT - a binary flag: off - the process is immediately executable
on - the process is presently blocked
LLINK - the left link (down)
RLINK - the right link (across)

Figure 5

The interrupt process is a separate process and can, of course, be implemented in a number of ways. However, one assumption included in this selection algorithm is that the interrupt process will set INTUPT(PTR) ← "on" when a process has been interrupted and is then not ready for execution: it should set INTUPT(PTR) ← "off" when a previously interrupted process becomes ready for processing. (INTUPT(PTR) is not used to indicate that a process is waiting for one or more of its child processes; in that case the parent process is put into a wait state.) This implies that the interrupt process (or some higher priority message coordinator) must examin a message queue of messages sent and received. According to reference (3) this information is already known by the monitor. Note that an interrupted process is not removed from the tree; this allows a priority process to continue to manipulate node positions even for interrupted processes.

A significant consequence of the process selector tree and selection algorithm is that a different scheduling scheme can be implemented for each set of sibling processes - and this is independent of the tree and algorithms. Therefore, in addition to the power of using multiple scheduling schemes simultaneously, this environment provides the flexibility of dynamically establishing new scheduling schemes, and so, scheduling need not be fixed in the system design.

## Global scheduling

Using the selection algorithm, process A (Figure 2) can be selected for execution if both processes D and E are interrupted. If only process D is interrupted, process E will be executed in preference to process A. If, however, process A is interrupted, neither process D nor process E can be executed since a tree search will never reach them. This means that if process A, for example, issued an input request, neither process D nor process E could be executed until that input operation were completed. Then if process D and process E were ready they would be executed before process A would be continued. Using a time-slice queue it might have been possible for processes D and E to execute while process A was interrupted for I/O, and this would cause a local increase in throughput. The global effect of this phenomenon has not yet been determined. The opposite condition, i.e. processes D and E are executed in preference to process A, seems to have a lesser possible effect on throughput. Thus, global scheduling may be a problem using the suggested algorithm, or through simulation the problem may be found to be only a problem in local scheduling. The suggested algorithm, however, need not be the only algorithm applicable to the process selector tree. In fact, since the above local scheduling problem is caused by a blocked parent process inhibiting its children from execution, a slight change in the suggested algorithm could be made. That alteration would allow tree traversal down through a blocked parent if one of its child processes is ready for execution. This, of course, would add some complexity to the algorithm. All these avenues will be examined with the goal of designing a flexible, yet straight forward, algorithm for working with the process selector tree.

## Conclusion

In Hansen's system (and in most others) the scheduling strategy is built in at a low level in the design. The process selector tree clearly allows the scheduling strategy to be determined at the same level as the processes which are to be scheduled. Moreover, the scheduling strategy may be altered dynamically and, in fact, may be different for each set of sibling processes. One obvious use of such flexibility is where the monitor is coordinating more than one operating system (Figure 6). The operating systems themselves may be executed according to one scheduling strategy and processes within each operating system may be executed according to some other scheduling strategies, including the inherent priority of the selection algorithm (i.e. no PP) as shown for SYS3 and the grandchildren of SYS1. Another obvious use is for the ease of testing various scheduling schemes at different levels of a hierarchical operating system.

To this author the process selector tree represents a structure which can be used to explore greater flexibility in scheduling strategies, especially where dynamic alteration of the scheduling algorithm is desired. Possibly a strategy of demand scheduling could be implemented, so that the processes to be executed actually cause an alteration in the scheduling algorithm. In addition this author believes that operating systems will become more well-defined and hence more well-structured, taking their basic concepts of structure from Dijkstra.[1,2] And, therefore, the process selector tree will provide a selection structure which maintains some of the structural properties of the system.

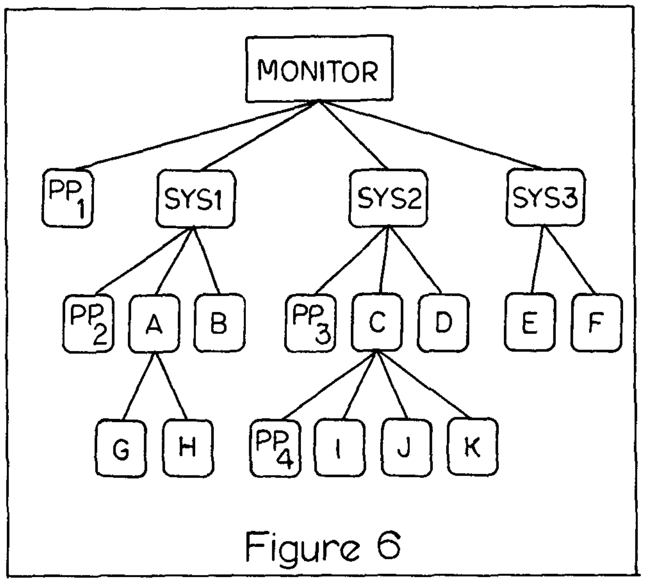

Figure 6

## Acknowledgements

The author wishes to express his appreciation to Drs. M. H. Gotterer and G. M. Campbell for their suggestions and discussions. He would especially like to thank Dr. A. N. Habermann for his kind and helpful cooperation. In addition, the author thanks Peter L. Anderson of Regnecentralen, Copenhagen for providing details of the RC 4000 which were not otherwise available.

## References

1. Dijkstra, E. W., "Co-operating Sequential Processes", in Programming Languages: NATO Advanced Study Institute (held in Villard-de-Lans 1966), F. Genuys (Ed.), Academic Press, London (1968).

2. ____________________, "The Structure of THE-Multiprogramming System", Comm. ACM, vol. 11, 5 (May 1968), pgs. 341-346.

3. Hansen, P. Brinch, "The Nucleus of a Multiprogramming System," Comm. ACM, vol. 13, 4 (April 1970), pgs. 238-241,250.

4. ____________________, (Ed.), RC4000 Software: Multiprogramming System. A/S Regnecentralen, Copenhagen, (1969).

5. ____________________, (Ed.) RC4000 Computer: Reference Manual. A/S Regnecentralen, Copenhagen, (1969).

6. Knuth, D. E., The Art of Computer Programming, vol. 1, Addison-Wesley: Reading, Massachusetts. (1968) Chapter 2.