Correspondence

To the Editor
The Computer Journal

An argument against paging hardware

Sir,

On reviewing Dr. Laski's excellent paper on segmentation and virtual address topology (this *Journal* Vol. 11 pp. 35–40) I am struck by the similarity of his approach to that outlined by Peter Wegner in his recent book (1). Both make the essential distinction between paging and segmentation which has confused us all for so long now. Both outline perfectly acceptable addressing schemes for segmented virtual processors. Undoubtedly the concept of a segmented virtual processor has to be completely understood before a workable machine will exist on which to implement an effective multi-access system. The influence of the English computer faction at Cornell must be impressive.

My quarrel with both authors, however, is that they assume that some form of paging scheme is necessary to map the virtual address into the physical addresses of the real processor. I am becoming more and more convinced that paging hardware is purely a temporary aberration of machine designers to overcome the shortcomings of the present technology. Peter Wegner admits that paging schemes have so far been unworkable, not because of the overhead incurred in the extra indirect addressing cycle(s) but because most real processors unfortunately do not restrict their references to the physical addresses of the few pages conveniently placed in main memory by the supervisor. Real processes tend to ramble all over their currently active segments.

So why not admit this fact and build hardware which will allow all the complete segments of any one process to live in main memory at the time they are active? Multiprogramming can then be achieved by time-slicing complete processes into main memory from auxiliary memory. That is, only one process is in main memory at one time and if it tries to activate a non-resident segment during its time-slice or alter the length of any active segment it is immediately transferred to auxiliary memory and another process started. During its next time-slice it is read into main memory together with the new segments it previously tried to activate. Segments can be allowed to expand and contract in auxiliary memory where there is a non-critical memory management problem but the problem for the main memory disappears. This scheme of multiprogramming has been called mono-programming as only one process is resident in the main memory at any one time.

The hardware necessary to do this clearly requires a sufficiently large main memory so that all the active segments of any single process can exist there during the time-slice given to that process by the supervisor. It also requires an auxiliary memory sufficiently large to contain all the active and dormant segments for all the active process and sufficiently fast that the access time to a process is negligibly small with a transfer rate sufficient to saturate main memory. The basics of the hardware necessary to do this exist in the Control Data 6600 with Extended Core Storage (2). If John Laski were to suggest that the addressing techniques of that machine were inadequate for the scheme of segmented virtual space he proposed I would entirely agree with him. However the basic technology nevertheless exists. Surely this points more accurately to the next generation of computers than some elaborate paging scheme with descriptors containing page-use bits which attempt to predict which page will next be referenced by some unmanageable process with a user on the end of a telephone line? Perhaps the Burroughs B8500 will have all the facilities we require?

Yours faithfully,
DAVID P. OWEN

References

(1) WEGNER, P. (1968). *Programming Languages, Information Structures and Machine Organisation.* McGraw-Hill.
(2) Control Data Corporation—6400/6500/6600 Computer Systems Reference Manual Pub. No. 60100000.

Stanford Research Institute
Menlo Park
California 94025.
4 June 1968

Dr. Laski replies:

It is not paging, but, run-time indirection to translate virtual address to physical address that is in fact what Mr. Owen objects to. The only possible alternative is load-time translation.

Under Mr. Owen's scheme this will have to occur not just at initial load but, via the indirection of a segment table, at each scatter load for each reference pointer occurring in non-private objects and for each reference in both private and non-private objects whenever he expands his physical space. Contracting physical space is also necessary and rather difficult to provide within his proposed organisation. Finally, of course, such a scheme is impossible if there is more than one processor: firstly since intersegment references have to be concurrently construed distinctly for distinct processes; secondly, and more importantly, since 'common' read-write segments must not be duplicated and thus cannot be made available to the processors of more than two (concurrent) processes.

For these reasons, Mr. Owen's scheme (with needed extensions) would, in my opinion, be much more expensive to implement than the kind of scheme I outlined in my paper for environments in which processers make substantive use of common data-objects that cross-reference one-another–the mark of an information utility—or vary dynamically in size—the mark of non-number-crunching; and conversely if processes require a fixed amount of private space only, Mr. Owen's scheme is vastly cheaper—as witness the cost-performance of the 6600 for its designed purpose if it has ECS.

Generality is always costly and, post MULTICS, we can see much more clearly what generality we need for concurrent processes to co-operate through a common dynamic data base. With present hardware prices this may be too expensive as yet. But if we want communal information utilities we will have to pay their price. Until hardware is cheap enough—which in my opinion will be 5 years—we must wait and experiment to gain experience and insight so that when we can afford our dreams they will have solidified into designs that will stand the light of day.