PROJECT MAC

5/5/75

Computer Systems Research Division Request for Comments No. 76

A Multi-process Implementation of a Paging System

by Andrew R. Huber

This RFC reproduces my Master's thesis proposal.

Abstract:

One method of managing memory in a computer system is by paging. The paging system consists of those parts of the system which allocate physical pages in the various levels of the memory system. The operation of the paging system offers many instances for exploiting parallelism. This thesis will demonstrate how a paging system can be designed as several concurrently-executing cooperating processes. The Multics paging system will be used as a specific case of a paging system which can be modelled as multiple processes.

Introduction

The Process Concept

Computer operating systems are increasingly being organized around and modelled by the process concept. (1)(3)(4) A process is an abstraction of a job or task or a program in execution; it consists of a pair: an execution point in a program and an address space, which is simply the set of all memory addresses the process may reference. The process concept is useful for at least two reasons: First, an operating system can be viewed as a task manager. Managing each resource such as processors, memory, etc. is a separate task; from this viewpoint it is natural to use a process to allocate each resource. Each user's work is a process or perhaps several cooperating processes. Then the operating system, and in fact the entire system, can be modelled as a collection of cooperating processes. Second, with the proliferation of multi-processor computer systems, the process concept offers a convenient and natural means of harnessing the multiple processors. A processor may be used to execute any user or system process, hence multi-processing of both users and the operating system itself is possible.

Paging Systems

One of the most important resources the operating system must manage is the memory or storage, which must be allocated among the various competing processes. Normally the memory system of a computer utility consists of many different types of physical devices or components, organized logically into two or more levels L0, L1, ... Ln where both the capacity and the access time increase with n.

A typical configuration of such a memory system might consist of high speed core memory as primary memory (level 0) with several disk drives as secondary or backup storage (level 1). Since in general the smaller the access time the higher the cost and therefore the smaller the capacity, the goal of combining such components with orders of magnitude differences in size and speed is to achieve an overall memory system whose capacity is equal to that of the largest component yet with an effective or average speed approaching that of the fastest device.

Paging is one common strategy for managing a multi-level memory. Examples of paged systems abound and include Multics (1), TENEX (2) and IBM's VS systems (6) (7). In a paged system, each level in the memory is physically divided into contiguous areas of a fixed size called page frames. Similarly each file or segment of a user is partitioned into contiguous pieces of the same size called pages. Then when allocating memory to a process, any page may be placed in any available page frame.

Usually a page may be referenced (i.e. read from or written into) only if it resides in the fastest level (level 0) of the multi-level memory. When a referenced page has not been allocated a page frame in level 0, it must be given one and placed in it. Hence the (scarce) level 0 page frames must be rationed among many competing processes. The program modules and data bases of the operating system which perform these allocations comprise the paging system, or "page control". Thus page control is a resource manager, the resources it manages being page frames.

The paging system may be invoked either explicitly by a request to move a desired page into level 0, or more often implicitly by merely referencing a page that is not currently residing in level 0 (this latter event is known as a page fault). Page control then executes the necessary algorithms to find an available level 0 page frame (which may involve transferring some other page to another level of the multi-level memory) and bring in the referenced page.

Note there are two distinct operations involved after the page fault occurs: first, the page removal operation, or "page replacement algorithm", which decides which page should be thrown out of level 0 to make room for the referenced page; second, fetching the desired page from wherever it resides in the multi-level memory into level 0. Clearly the first operation can be done at any time prior to the second. In fact, as long as a pool of free page frames existed the page removal operation need not even take place when a page fault occurs.

Since it is a repetitive operation that can be done when convenient, a process could be dedicated to this task of replenishing the pool of free level 0 page frames. While this process was freeing page frames for later use there is no reason why some other process could not be bringing a page into the level 0 memory. Thus the two operations, removing a page from level 0 and reading a page into level 0, could be done in parallel.

This is only one example of the parallelism inherent in page control. In a system such as Multics which has a multi-level memory system of more than two levels, the page replacement operation must be carried out at each level. For example, when a page is removed from core (level 0) in Multics, an attempt is

made to move the page to a so called "paging device" (level 1, in reality a large bulk store memory) used as a high speed backing store. In doing so, some other page will normally have to be moved to the disks (level 2, the last level). Thus a page replacement operation must be done on the level 1 page frames. In general, at each level in a multi-level memory except the last (which is assumed to be large enough to hold all pages) a page removal algorithm must be in operation. Extending the page replacement idea introduced above, for each level a process can be assigned the job of maintaining a pool of free page frames for later use. These processes may operate essentially independently.

Hence it seems natural in modelling the operating system as cooperating processes to consider the paging system as several such processes. The replacement algorithm for each level of the multi-level memory is a prime candidate for implementation as a process since each is really a resource manager whose resource is the page frames of that level. Each repeats the same sequential task over and over again, namely freeing page frames for one level. Moreover, these operations can be done simultaneously at each level, and asynchronously with respect to page faults.

Purpose of Thesis

The thesis proposed here will develop a design of a paging system that will be implemented as several cooperating processes. Conceptually all of these may be executing simultaneously; in an actual system the number will of course be limited by the number of processors. The design will be based on the current Multics page control, and the test implementation will be on the Multics system.

The Multics paging system is currently implemented as a single module all of which attempts to run as part of the user's process at the time a page fault happens. If a part of page control is unable to continue, for example when waiting for an input operation to complete, the user's process is suspended. When the operation causing the suspension finishes, whatever process was currently running is interrupted and the appropriate part of page control continues from the point where it was suspended.

Note the operations of finding a free page frame at both level 0 and level 1 are done sequentially, and only immediately after a page fault has occurred. In contrast, the proposed design will place these functions in separate processes distinct from the operation of resolving a page fault. More specifically, the Multics page control module will be redesigned into three parts: 1. A level 0 memory manager process (or core manager),

which will search level 0 of the memory system, freeing page frames so that they will be available when needed. (This constitutes the level 0 replacement algorithm). 2. A similar level 1 memory manager process (or paging device manager), to free page frames in level 1 of the memory. (This implements the level 1 replacement algorithm). 3. A page fault handling module to perform the necessary input operations to bring a missing page into core. This third module will, as before, be invoked implicitly by the user process and run as part of the user process in the same fashion as all of page control does currently.

Aside from its availability, the Multics system is a good choice for such a project because it is fairly typical of large paged systems. Its three level memory system is sufficiently general to illustrate the applicability of the method to n-level memory systems without undue complexity caused only by having a large number of levels. More importantly, Multics provides the process notion in its implementation.

Motivation

In this section, the benefits which will accrue to a page control redesigned as multiple processes and therefore which motivate this research are discussed. This work was originally suggested during consideration of alternate strategies and structures for a paging system that would simplify the current Multics page control while correcting several observed deficiencies.

The primary motivation for this work is to demonstrate the validity of such a design. While there appear to be no insurmountable difficulties in a multi-process page control such as described here, no such implementation is known to the author. Hoare (5) has proposed a high level model of a paging system as several cooperating processes and suggests it as a model for system designers. In any event, his model is not general enough to fit many systems including Multics because it is restricted to a two level memory system (versus Multics' three) and it assumes a page will reside in only one of the two levels at any given instant. In Multics, a copy of a page may reside in one, two or all three of the levels at the same time.

Various proposals have been made with regard to implementing variations of this proposal on Multics. One such proposal would have separated the management of the paging device from the rest of page control and implemented it as a separate process (i.e. only the second of the three parts proposed here.) A recent revision of page control includes the ability to run the core page replacement algorithm when the system would otherwise be idle. This is similar to the first of the three parts proposed

here, but does not go as far since the replacement algorithm does not become a separate process.

Thus the implementation proposed here would be the first actually to incorporate all of these proposals into a unified, comprehensive design. Moreover, the paging system designed as multiple processes would include the following desirable properties which represent substantial improvements over the current Multics page control:

1. Simplicity

The current page control is extremely complex and difficult to understand for several reasons. One of the most important sources of this complexity is its strictly sequential operation. Performing essentially independent but related operations in a fixed sequence forces in many cases an awkward, unnatural ordering to prevent deadlocks. The situation becomes worse when attempts are made to optimize the resulting algorithms, because often the optimizations are necessary only because the operations are done synchronously or in a specified sequence.

By designing page control as independent but cooperating processes, it will be possible to remove the strict sequential ordering of operations, simplifying the code and rendering some optimizations unnecessary. Each process will be understandable by itself, without reference to the other processes except at explicit synchronizing points or via explicit interfaces. Improved understandability means easier modification and clearer examination of alternative algorithms and their implications for system performance. Simplification is also of direct benefit to any effort to certify or prove correct the algorithms of page control.

2. Certifiability

A chief goal of current research on the Multics system is to provide a system with a certifiably correct security kernel. The proposed design aids in this certification effort in three ways: 1. The design is simpler and more modular, hence the certification task is simplified. 2. The modularity of the design makes possible separation of the components behind protective "firewalls". 3. That portion of the resulting system that must be certified is minimized and confined to easily identified modules.

For example, the only parts of the design that need be certified to insure against unauthorized release of information are the actual input and output routines. Since none of the rest of page control has any reason to look at the contents of a page and hence will not do so, it need not be certified. No attempt will be made to certify any part of the proposed system however.

3. Expandability

The existing Multics page control, while adequate for the present configuration of two processors, would not expand well to larger numbers of processors. This is largely because page control can execute only in a single process at any time since it uses a global lock which a process must set before it enters page control. This is done in order to protect the consistency of page control's data bases. Thus it is impossible in the present implementation for one process to be freeing core page frames while another is freeing paging device page frames. Splitting the paging system into distinct processes each operating on its own data bases will alleviate much of the need for this lock (but not all as it is used for other purposes also). The core manager and the paging device manager processes will no longer have to set this lock before they begin executing.

In fact by careful design it will be possible to have several identical core manager processes (or paging device manager processes). Hence as the system grows in size to the point where one core manager process can no longer keep a plentiful supply of free core page frames, additional processes can be added as necessary. Page control will change from a system that does not scale up well to an easily expandable system.

A second limitation imposed by this global lock can possibly also be removed. The lock prevents two processes from handling page faults at the same time. Simultaneous handling of multiple page faults may be possible through careful design, although some restrictions will undoubtedly be necessary. (An obvious one is that two processes cannot attempt to bring in the same page since multiple copies might result.)

To summarize, this thesis proposes a design and trial implementation of a paging systems for the Multics system structured as three processes, with the goal of demonstrating that such a design will result in a more natural, simpler page control that will be easier to certify and expand readily and efficiently as the system grows.

Schedule and Resources

The design of an initial page control following the general plan described here but allowing only one process to be executing the page fault handling code is essentially complete. A model implementation of this design can begin immediately. The model implementation is considered important to validate the design and insure complete understanding of the issues and difficulties involved in such a design. In particular, the efficiency of the resulting code is not considered relevant as long as any major

inefficiencies are attributable to other sources such as the process scheduler. Permanent installation of this model implementation into the Multics system is not contemplated at this time.

The problems involved in allowing multiple page fault handling are currently being investigated. The results of this research will be incorporated into a refined design. At this writing it is not clear what restrictions will be necessary on processing simultaneous page faults; thus it may not be possible to completely implement this refined design. The thesis will however contain a discussion of the factors which make simultaneous page faults difficult to handle.

The only resources necessary for this research are the Multics system itself and an undetermined amount of computer time. Both of these resources are readily available. It should be possible to complete the proposed research and write the thesis by the end of June.

References

1. Daley, Robert C., and Dennis, Jack, "Virtual Memory, Processes, and Sharing in MULTICS", Communications of the ACM, vol. 11, no. 5, (May 1968), pp. 308-312.

2. Murphy, D. L., "Storage Organization and Management in TENEX", AFIPS Conf. Proc. 41, vol. 1, (FJCC 1972), pp. 23-32.

3. Dijkstra, E. W., "The Structure of the THE Multiprogramming System", Communications of the ACM, vol. 11, no. 5, (May, 1968), pp. 341-346.

4. Hansen, P. B., "The Nucleus of a Multiprogramming System", Communications of the ACM, vol. 14, no. 4, (April 1970), pp. 238-241.

5. Hoare, C. A. R., "A Structured Paging System", The Computer Journal, vol. 16, no. 3, (August 1973), pp. 209-215.

6. Wheeler, Jr., T. F., "OS/VS1 Concepts and Philosophies", IBM Systems Journal, vol. 13, no. 3, 1974, pp. 213-229.

7. Scherr, A. L., "Functional Structure of IBM Virtual Storage Operating Systems Part II: OS/VS2 Concepts and Philosophies", IBM Systems Journal, vol. 12, no. 4, 1973, pp. 382-400.