Diverse workloads and a wide range of hardware configurations compound the complexity of an operating system's memory management policies.

Virtual Memory Management in the VAX/VMS Operating System

Henry M. Levy and Peter H. Lipman, Digital Equipment Corporation

The VAX-11/780, introduced in 1978, and the smaller VAX-11/750, introduced in 1980, are Digital Equipment Corporation's first implementations of the 32-bit VAX-11 minicomputer architecture.[1] Eventually, there will be a family of VAX-11 minicomputers. The VAX-11 and its VAX/VMS operating system were designed to provide extended address space and enhanced performance for Digital's PDP-11 customers. Built on Digital's experience with a multitude of PDP-11 operating systems, VAX/VMS was intended to provide a single environment for all VAX-based applications, whether real-time, timeshared (including program development), or batch. In addition, VAX/VMS had to operate on a family of processors having different performance characteristics and physical memory capacities ranging from 250K bytes to more than 8M bytes. To meet the requirements posed by these applications environments, the memory management system had to be capable of adjusting to the changing demands characteristic of timesharing while allowing the predictable performance required by real-time and batch processes.

Memory management policies and decisions made in the design and implementation of the first release of the VAX/VMS operating system in 1978 reflected the concerns of the developers. This article examines those concerns and the mechanisms selected to deal with them.

VAX-11 hardware

The basic entity in the VAX-11 system is the process. Each process has a byte-addressable, 32-bit virtual address space, which is divided into 512-byte pages. A 32-bit virtual address contains a 21-bit virtual page number and a 9-bit byte offset within the page. The page is the basic unit of mapping and protection.

The upper two bits of the virtual address divide the process address space into a number of functional regions or spaces. Figure 1 shows the division of the address space into system-wide and per-process segments. The high-

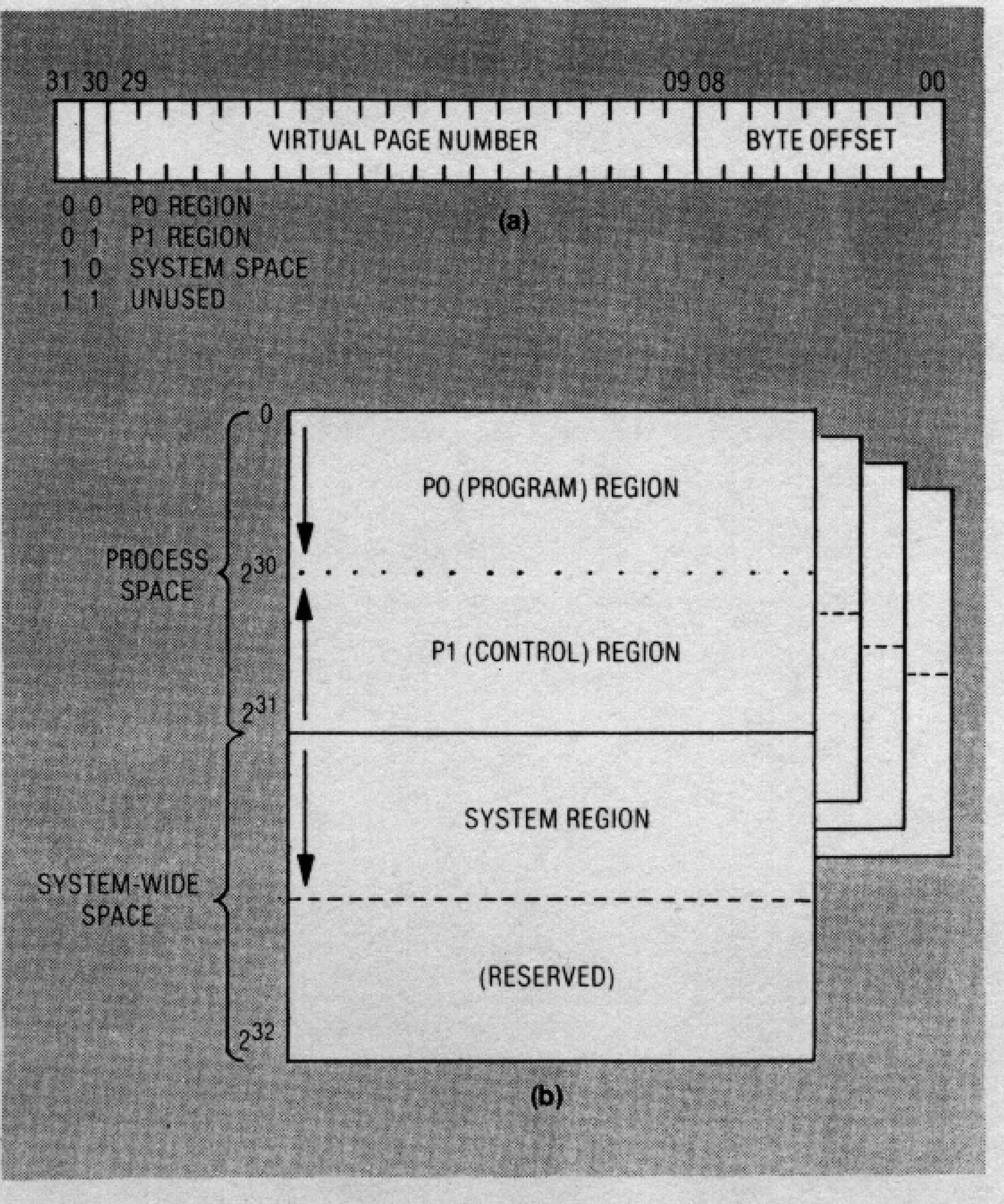


Figure 1. VAX-11 process virtual address (a) and virtual address space (b).

address half of the address space (bit 31 = 1) is known as system space and is shared by all processes in the system. That is, a system-space virtual address generated by any process will access the same physcial memory location. Only half of the system space is currently used by the architecture.

The low-address half of the address space (bit 31 = 0) is known as process space and is unique to each process. Process space is itself divided into two regions by bit 30 of the virtual address. The low-address half, known as the P0, or program region, grows toward higher addresses, as is shown by the arrows in Figure 1. The high-address half, known as the P1, or control region, grows toward lower addresses. Each process, therefore, has two local segments offering two directions of expansion.

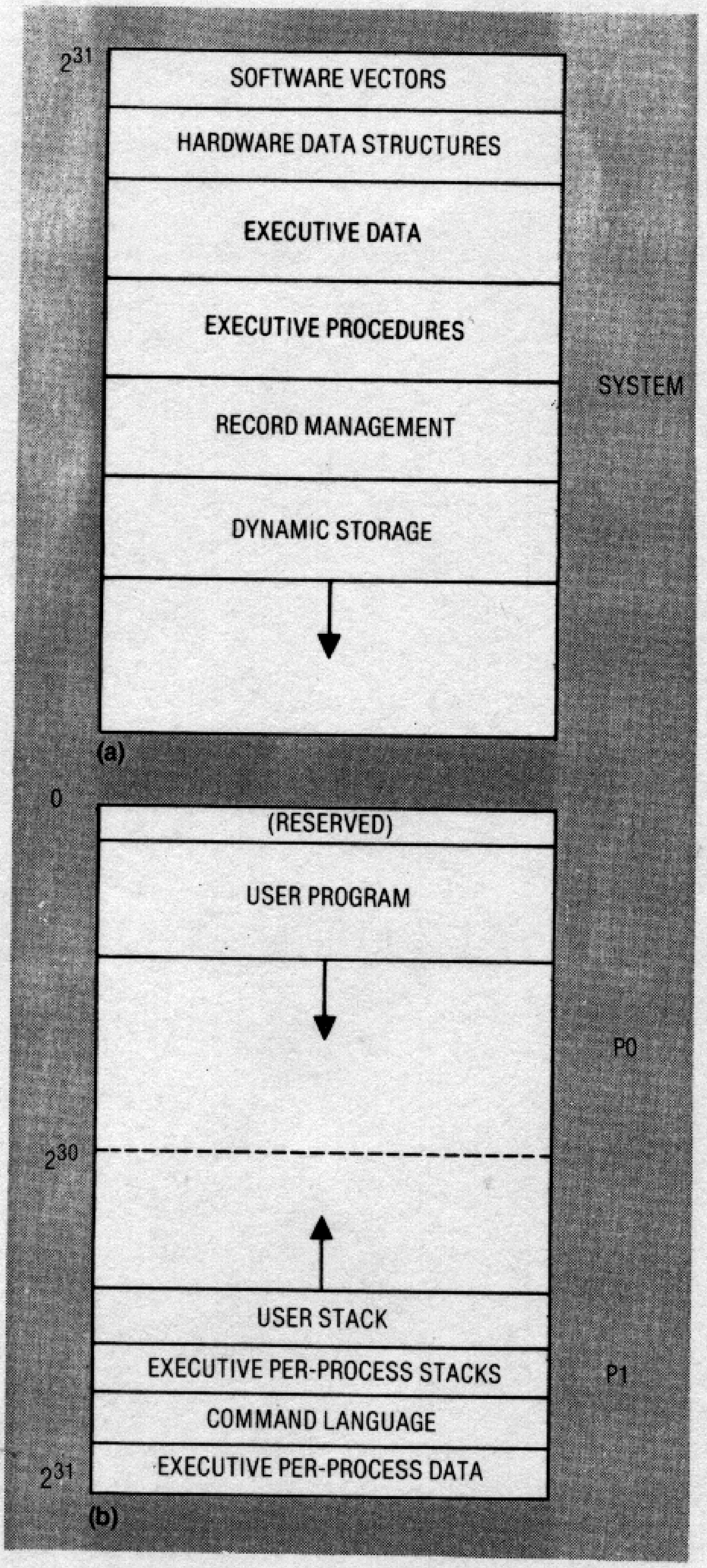


Figure 2. VAX/VMS use of VAX-11 address space: (a) system space layout; (b) per-process space layout.

Each region (system, P0, and P1) is defined by a page table. A VAX-11 page table is a contiguous array of 32-bit ("longword") page-table entries. Each page-table entry, or PTE, contains the following fields:

(1) a valid bit (PTE<31>) that indicates whether the page-table entry contains mapping information;
(2) a protection field (PTE<30:27>) that indicates what privilege is required to read or write the page;
(3) a modify bit (PTE<26>) that indicates whether a write access has occurred to the page;
(4) a field used by the operating system (PTE<25:21>); and
(5) the physical page frame number (PTE<20:0>) that locates the page in physical memory.

If the valid bit is zero, all other bits, with the exception of the protection field, can be used by software. Because the hardware checks protection for a page whether or not the valid bit is set, an illegal access to a page cannot cause a page to be loaded.

Each page table is defined by two hardware registers: a base address register and a length register. The page table for system space, called the system page table, is located by reference to the system page table base register, which contains its physical address. The P0 and P1 page tables for each process are located in the system-space section of the address space; therefore, the P0 and P1 page table base registers contain virtual addresses. The P0 and P1 page tables can themselves be paged because they are in virtual memory. The translation of a process-space virtual address involves two accesses: one to the system page table, to calculate the physical address of the process page table, and one to the process page table, to calculate the physical address of the specified element. Since the P0 and P1 page tables are process specific, the corresponding base and length registers are changed by a process context switch. The system page table, because it is shared by all processes, is not affected by a context switch.

The VAX-11 hardware provides a translation buffer for caching virtual-to-physical translations, so multiple references to page tables in physical memory are usually avoided. The translation buffer is divided into two sections, one for system translations and one for per-process translations. When a context switch occurs, only the per-process section needs to be flushed.

Use of address space by VAX/VMS

As their names imply, the three address regions, or spaces, are used by VAX/VMS for specific purposes. All executive code and data, including some process-specific data structures and process page tables, are stored in the system region. Figure 2(a) shows the layout of system-space virtual memory. The first few pages of system space, called the vector region, contain pointers to executive service routines in system space. Users call service routines through these fixed-location vectors. This calling scheme allows routines to be relocated in later operating-system releases without affecting user programs.

Thus, the VMS operating system is shared implicitly by all processes in the system. Users call operating-system service routines as they would call any user-written procedure. The operating system is protected through the memory management system, which uses a hardware access mode mechanism; executive code and data are not made accessible to programs executing in the least privileged (user) mode.

VAX/VMS, then, is a collection of procedures that exist in the address space of each process. These procedures can be called explicitly or—for example, when an exception or page fault occurs—implicitly to perform services on behalf of a process. Because the operating system does not have a separate address context, a context switch can occur even while a process is executing code within the executive. Several processes can be executing simultaneously within the operating system.

The P0 (program) region contains the user's executable program. The user's program can dynamically grow into higher-addressed sections of P0 space. In constructing an executable program, the linker usually allocates the program beginning at the second page of the virtual address space. The first page, address 0 through 511 decimal, is marked as no access so that any reference of address 0 will cause an exception. This policy has proved to be valuable in catching program errors involving uninitialized pointers (including several "working" programs ported to VAX/VMS after several years of operation). VAX/VMS uses the P1 (control) region to store process-specific data, as well as the program image for the command interpreter, as shown in Figure 2(b). The command interpreter executes within this region so as not to disturb user program images. The user's stack is located in the low-address part of the P1 region and grows toward lower addresses. The P1 region also contains fixed-sized stacks for use by executive code that executes on behalf of the process.

Memory-management implementation

Most mainframe computer systems include special hardware (such as fixed-head paging disks) to enhance the paging performance of virtual memory systems. Minicomputer systems, in contrast, often provide inadequate hardware support for virtual memory operation. Moreover, they must support a wide range of hardware implementations. Therefore, the VAX/VMS memory management system had to be able to operate on hardware configurations having small, inexpensive CPUs, slow moving-head disks, and small memories.

In light of the requirements posed by these low-end systems, the designers of VAX/VMS were concerned with a number of problems typically encountered in paging systems. Their major concerns included:

(1) the effect of one or more heavily paging programs on other programs in the system;
(2) the high cost of program startup and restart time in virtual memory environments that require a program to fault its way into main memory;
(3) the increased disk workload caused by paging; and
(4) the processor time consumed by searching page lists in the operating system.

These problems are only amplified by the constraints of minicomputer hardware.

To manage these problems, the VAX/VMS memory-management system is divided into two basic components: the pager and the swapper. The pager is an operating system procedure that executes as the result of a page fault. It executes within the context of the faulting process and is responsible for loading and removing process pages into and out of memory. The swapper is a separate process responsible for loading and removing entire processes into and out of memory.

The pager. The first design concern listed above is the effect of heavily faulting programs on other programs in the system. Some paging systems use global replacement policies whereby a fault can cause removal of any program's pages. In this scheme, the pages usually selected for removal are those least recently used on a global basis;

Trade-offs were made in favor of reducing processor usage at the possible cost of increased memory requirements.

that is, the page not referenced for the longest time (the LRU page) is removed. When restarted, the affected program must fault these pages back into memory one at a time. In an attempt to limit the effect of a heavily faulting process on other processes in the system, VAX/VMS uses a process-local page replacement policy. That is, when a page must be removed from memory, the page to be removed is selected from the process requesting a new page.

A limit is placed on the amount of physical memory a process may occupy. The set of pages currently in memory for a process is called the process's resident set, and the resident-set limit is the maximum size of the resident set. For each process, the pager maintains a data structure, called the resident-set list, that describes the pages contained in that process's resident set.

When a new program is started, its resident set is initially empty. As the program executes, pages are loaded into its resident set by the pager whenever a nonresident page is referenced. When the resident-set limit is reached, the faulting process must release a page for each newly faulted page added to its resident set. The pager uses simple first-in, first-out replacement within the resident-set list to select the page to be removed.

In most virtual memory systems, a reference bit in each page table entry is set whenever the page is referenced. By periodically clearing all reference bits and remembering when each was last set, the operating system can keep track of the relative "age" of each page. (Other reference-bit schemes, such as a clock, do not require a complete scan of all reference bits.)*

*Although VAX does not have reference bits, work at the University of California, Berkeley,[2] on VAX/Unix, which uses a reference-bit algorithm, suggests that software simulation of the reference bit consumes only 0.05 percent of the processor.

The VAX/VMS designers, however, were concerned with the cost of reference-bit scanning for extremely large programs. On slower processors with large physical memories, the computation time required to search process pages in support of more sophisticated algorithms might be prohibitive. And, since it is easier to add more memory than it is to increase processor cycles, trade-offs were made in favor of reducing processor usage at the possible cost of increased memory requirements. Therefore, the amount of computation required of the pager is limited; a quick decision is made, and a software caching mechanism is used to reduce the penalty for removing a page that is still in use. In addition, no overhead is required to maintain well-behaved programs executing in sufficient physical memory.

When a page is removed from a process's resident set, it is placed on one of two page lists: the free page list or the modified page list. If the modify bit is zero in the page-table entry, a copy of that page still exists on disk, and the page is added to the tail of the free list. The free list is the source of available page frames for the system. For example, when the pager requires a page of memory into which to read a newly faulted page, it takes the frame from the head of the free list. Therefore, a page removed from a resident set will remain on the free list for some time before it makes its way to the head of the list and is consumed; how long it remains depends on the size of the list and the fault rates of currently running programs.

If the modify bit in the page-table entry of the removed page is one, the page has been changed and must be written to the paging file. Such a page is queued on the tail of the modified page list, where, again, it will remain for some time before being written back to disk.

If a process faults a page that is on either list, the page is returned to the process's resident set. The time required for a fault from one of these lists is approximatley 200 microseconds on a VAX-11/780. These lists, then, act as physical memory caches for recently removed pages. As long as the list sizes remain above some threshold, the cost of removing a page about to be referenced from a resident set is minimal.

In addition to providing a high-speed cache for pages, the VAX/VMS free-list mechanism acts to reduce program fault rates. This reduction occurs because the faulting of a page from the free list causes the page to be placed back at the top of the FIFO resident-set list. Results from simulation studies,[3] as seen in Figure 3, show the effects of reducing the size of a program's resident set by 10, 20, and 30 percent and using the removed memory as a private free list. The data shown were obtained from a Fortran compilation consisting of approximately four million references to 359 pages. Although this differs slightly from VAX/VMS, in which the secondary list is global, the graph indicates that the addition of a free list can reduce the fault rate of strict FIFO replacement to a level arbitrarily close to LRU. This effect has been demonstrated by other analytic and simulation studies.[4]

Clustering of pages. Since most minicomputers do not have a separate paging device, the additional I/O caused by paging can have a dramatic effect on normal file system and swapping response time. In the VAX, this prob-

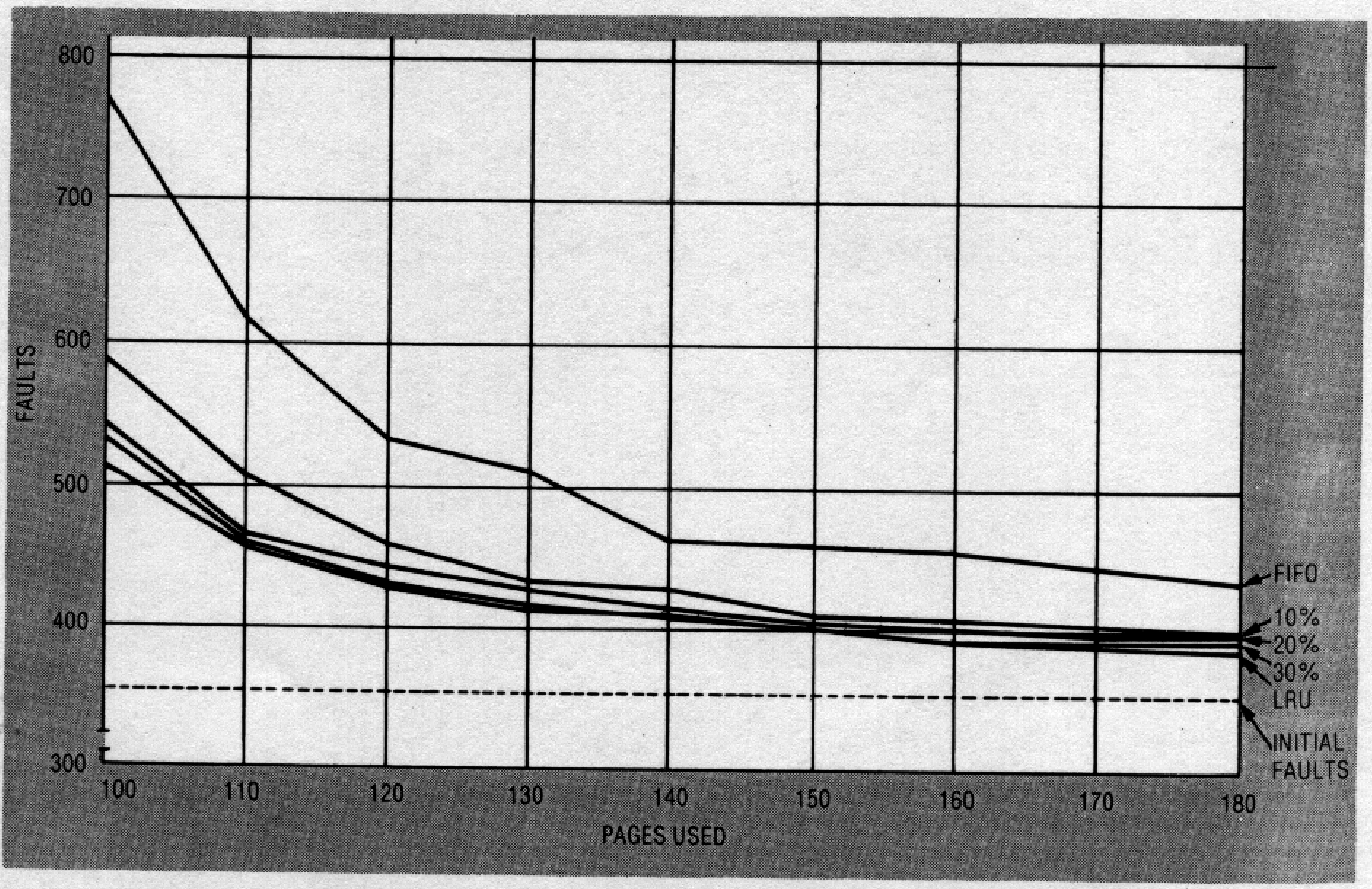


Figure 3. Faults vs. memory usage in Fortran compilation.

lem is magnified by the relatively small page size of 512 bytes. (The page size was chosen for file-system compatibility with the PDP—11 and because of the promise of low-latency semiconductor disk technologies.) To combat the additional paging I/O load, VAX/VMS attempts to reduce the number of physical disk operations by reading and writing several pages at a time, a process known as clustering.

When a page fault occurs, the pager may load several pages into the process's resident set, the number to be loaded depending on several parameters. The page to be read can be found on either the modified or the free list, in the paging file, in the original executable file for the program, or in some other file that has been "mapped" into the address space. (A program can request that a file be mapped into its virtual memory; any access of a mapped page may then cause a corresponding file block to be loaded into memory.) The program loader simply maps the executable program file and begins program execution. The largest gain in read clustering of pages comes from initial program execution when fault rates are typically high.

An executable program is composed of several segments. A segment is a set of virtually contiguous pages with similar attributes (for example, all are read-only). A user can specify a default cluster size for each segment when a program is linked. This cluster size is the maximum number of pages the pager will attempt to load at once. Normally, no cluster size is supplied by the user, and a systemwide default is used.

When the pager locates a page to be read from the executable file, it determines the cluster size for the associated segment. The pager then checks to see whether the adjacent blocks following the requested page on disk contain pages that are adjacent to it in virtual memory. Since the linker produces physically contiguous executable files on disk whenever possible, it is likely that several pages can be read with a single disk operation. If fewer than two contiguous pages are found, the pager will check preceding pages within the segment. Finally, the pager allocates the required number of page frames from the free page list and initiates one disk read request to add the chosen pages to the resident set.

A cluster size is also associated with a paging file, and the pager performs a similar service there. However, since the paging file contains pages from many processes, there is less chance of finding virtually contiguous pages that are physically contiguous on disk.

Page writes occur from the modified page list, which receives all modified pages that are removed from resident sets. The modified page list serves two purposes: (1) it acts as a cache of recently removed resident-set pages, and (2) it allows for clustering writes to the page file. Two system parameters, a low limit and a high limit, are associated with the modified list size. VAX/VMS tries to maintain at least the low limit, so that caching can take place. However, when the list size reaches the high limit, the pager writes a number of pages equal to the high limit minus the low limit back to the paging file. In general, this is done with one physical I/O operation to contiguous blocks of the paging file. On a typical one-megabyte VAX-11 system, the modified list's high limit is set at more than 100 pages, while the low limit is set at 20 or 30 pages.

When preparing the operation, the pager searches the modified list to find any virtually contiguous pages from any of the processes. The write is performed so that virtually contiguous pages from any process will be written to contiguous paging-file disk blocks. When one of these pages is faulted again, the pager may be able to cluster on the read request.

Most paging systems attempt to write modified pages to the paging file as quickly as possible in order to recover the space. However, by delaying modified-page write requests through the modified page list mechanism, VAX/VMS realizes four optimizations:

(1) The modified page list acts as a cache of recently removed pages. If a page on the list is referenced, it is returned to the process at minimum cost.
(2) When a write request must be performed, many pages can be written at once. Our development system typically writes about 100 pages with a single write to the paging file.
(3) Pages can be arranged on the paging file so that clustering on read requests is possible.
(4) Because the writes are delayed, many page writes are entirely avoided because either the pages are faulted again or the program terminates.

Demand-zero and copy-on-reference pages. Pages that were initially zero in a compiled program, such as uninitialized arrays, are not stored by the linker in the executable file. Instead, the linker creates a null-sized segment of demand-zero pages. A demand-zero page is a page that is created for a program and initialized to zero on demand. When the program is run, the corresponding page-table entries are marked invalid, with the remaining software bits indicating the presence of a demand-zero page. When the page is faulted, the pager allocates a physical memory page, fills it with zeros, and adds the page to the resident set of the process. The pager also sets the modified bit of the page-table entry so that the page will be written back to the paging file when it is removed from the resident set.

In a similar way, the pager sets the modify bit of the page-table entry for a writable page when it is first loaded from the executable file. From that point on, the paging file will be used as backup storage for the page. Such pages are known as copy-on-reference pages. When several processes are sharing an executable file, each process faulting a copy-on-reference page will receive its own copy of that page.

Paging in system space. The VAX/VMS operating system is composed of both paged and nonpaged code and data. For paged code and data, the operating system has a system resident set and a system resident-set list analogous to those of the process structures. The size of the system resident set is a parameter that can be adjusted for different environments.

As page faults occur for the system, pages are loaded into the system resident set. Once it is full, system page faults cause pages to be removed from the system resident set and placed on the free or modified page list.

One exception is the treatment of process page tables, which are pageable and located in system space. When a process faults one of its page tables, the page table is added to the process's private resident set. The page table will not be eligible for removal from the resident set as long as it contains any valid page-table entries. As a member of the resident set, however, it will be swapped along with the process. When a page table no longer contains any valid mapping information, it becomes eligible for removal along with other pages of the resident set.

Paging system operation. Some simple measurements demonstrate the effectiveness of some of the paging mechanisms described above. The data in Table 1 were collected during an experiment with a simulated 20-user educational workload. The columns show the total counts and average rate per second of several system events during that interval. The rows marked "pages read" and "pages written" give the number of 512-byte pages moved from or to disk, while the "read and write I/O operation" counters give the number of disk transfers used to read or write those pages. From these measurements we see the effect of clustering; an average of 5.6 pages were transferred with each read operation from disk, while an average of 98.2 pages were transferred with each write from the modified list. We also see that over half of the page faults were satisfied from the free list, from the modified list, or from creation of demand-zero pages.

The swapper. In addition to paging, VAX/VMS swaps entire resident sets between memory and backing store. The objectives of swapping are (1) to keep the highest-priority processes resident and (2) to avoid the typically high paging rates generated by resuming a process in most paged systems. Swapping is handled by a process called the swapper, which is also responsible for writing the modified page list. The swapper executes whenever it is awakened by the operating system.

Whenever a process is removed from memory, its entire resident set is written to the swap file, along with some process-specific operating system data base. The swap operation may be performed in several pieces. For example, if an I/O operation is in progress to some pages of the resident set, those pages and their page tables are locked in memory until the opeation completes. The swapper writes all resident-set pages to a contiguous section of the swap file, including the pages with I/O in progress. If the I/O was a read operation, the pages will be written to their slots in the swap file when the read completes.

When a process not in memory becomes able to execute (as an I/O operation is completed or a necessary resource becomes available), it is read in by the swapper. The swapper allocates pages for the resident set, page tables, and operating system data structures. If needed, space is found by writing the modified page list or swapping out other processes. Once free pages are allocated and the read is initiated, the swapper updates the process's page-table entries to reflect the new virtual-to-physical mapping.

VAX/VMS will not load a process that has been swapped out unless there is sufficient physical memory for its entire resident set. When a process resumes execution after it has been swapped in, its resident set has exactly the same membership that it had before its removal from memory.

The swapper is also involved in process creation. When a new process is created, the swapper swaps in a process shell, which provides the initial environment in which a program can be executed.

Program control of memory

For real-time processes that need a closely controlled environment, or for any program that wishes to control its use of memory, VAX/VMS provides a number of executive services. For example, a process can call routines to:

- Expand (or contract) its P0 or P1 region.
- Increase (or decrease) the resident set size (each process has lower and upper limits set by the system manager).
- Lock (or unlock) pages in the resident set so that they are always resident with the process.
- Lock (or unlock) the process in memory (the balance set) so that it is never swapped.
- Create and/or map a global or private section into the process address space.
- Produce a record of the page-fault activity of the process.

These routines allow a program to monitor its behavior and to act on the information the routines obtain. Programs that have knowledge of their use of virtual memory can tell the operating system which pages are needed or not needed at any time.

In VAX/VMS, Digital has taken a slightly unorthodox approach to virtual memory management. This approach was adopted for two major reasons. First, because of the variety of intended hardware and application environments, the system needs to tightly control the use of memory by processes, to limit the effect any one process's use of memory might have on another process's operation, and to limit overhead processing in the paging system. Second, because of the range of hardware configurations (and disks in particular) to be supported, specific approaches had to be taken to reduce the I/O workload.

Table 1.
Sample VMS Paging System Measurements
(Interval time is 824.0 seconds)

	COUNT	RATE/SECOND
TOTAL PAGE FAULTS	79847	96.902
PAGES READ	39957	48.492
PAGE READ I/O OPERATIONS	7012	8.510
PAGES WRITTEN	11501	13.958
PAGE WRITE I/O OPERATIONS	117	0.142
FAULTS FROM FREE PAGE LIST	21483	26.072
FAULTS FROM MODIFIED PAGE LIST	17916	21.743
DEMAND-ZERO FAULTS	9740	11.820

Three major mechanisms have been used to enhance the performance of the operating system. The first is caching of pages, which reduces the number of disk I/O transactions on pages that have been recently removed from a process's resident set. A fault from either the free or the modified page list is handled rapidly within the pager. The free-list mechanism has also been shown to help reduce the fault rate.*

The second mechanism, clustering, provides the transfer efficiency of large pages along with the fragmentation characteristics of small pages (at the cost of some additional space in the page data base). The modified-page-list mechanism has a significant effect on system performance. By delaying the writing of modified pages, this mechanism allows the operating system to completely avoid some writes—for example, in the case of refault or program termination. When pages must be written, they can be transferred very efficiently in large units—say, 100 pages at a time. The size of the modified page list is possibly the most important performance parameter in the system.

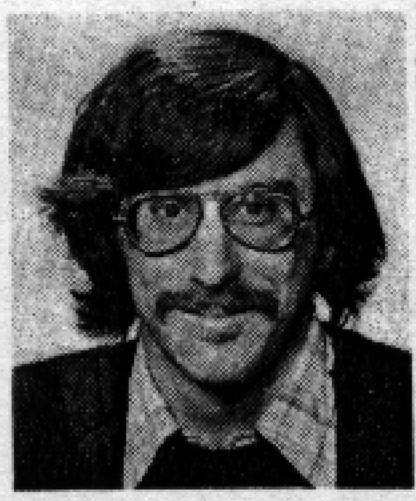
The third mechanism is to use process-local replacement, along with swapping, to reduce the effect of one process's paging activities on another's. When a process is restarted following a context switch, it always resumes with its resident set as the set existed before the interruption. A process is moved from and to memory with only a few I/O transfer operations.

VAX/VMS is currently being used in such areas as airplane simulation, real-time data collection, design automation, timesharing development, commercial data processing, and computational tasks. Since the first release of VAX/VMS, the memory management system has been greatly enhanced; however, the basic mechanisms have continued to prove useful, even in more general and dynamic environments. ■

References

1. W. D. Strecker, "The VAX-11/780—A Virtual Address Extension to the DEC PDP-11 Family," *AFIPS Conf. Proc.*, Vol. 47, 1978 NCC, AFIPS Press, Montvale, N.J., 1978.
2. O. Babaoglu, W. Joy, and J. Porcar, "Design and Implementation of the Berkeley Virtual Memory Extensions to the UNIX Operating System," Department of Electrical Engineering and Computer Science, University of California, Berkeley, Dec. 1979.
3. R. Turner and H. Levy, "Segmented FIFO Page Replacement," *Proc. ACM/Sigmetrics Conf. on Measurement and Modeling of Computer Systems*, Sept. 1981, Las Vegas, Nev., pp. 48-51.
4. O. Babaoglu, "Virtual Storage Management in the Absence of Reference Bits," PhD thesis, Univeristy of California, Berkeley, Nov. 1981.

*These lists do suffer from an inherent unfairness, however, because a heavily faulting process can cause a turnover of the lists. This competes with the goal of isolating programs from each other.



Henry M. Levy is a member of the VAX Systems Architecture Group at Digital Equipment Corporation. His interests are in operating systems, computer architecture, and distributed processing. He was a member of the design and implementation team for the VAX/VMS operating system. Levy has a BS from Carnegie-Mellon University and an MS from University of Washington. He is the co-author of the book *Computer Programming and Architecture: The VAX-11.*

Peter Lipman is a consulting engineer with Digital Equipment Corporation in Maynard, Massachussetts, working on operating systems advanced development. He was one of the principal designers and implementors of the VAX/VMS operating system, responsible for memory management on VMS release 1. Previously, Lipman designed and implemented file systems for the RSX-11M and RSX-11D operating systems. Before coming to Digital, he worked at Stanford Research Institute in Menlo Park, California. Lipman received an ScB from Brown University in 1965, and an MS from Stanford University in 1969.

TERMINALS FROM TRANSNET

PURCHASE PLAN • 12-24 MONTH FULL OWNERSHIP PLAN • 36 MONTH LEASE PLAN

	DESCRIPTION	PURCHASE PRICE	PER MONTH 12 MOS.	24 MOS.	36 MOS.
DEC	LA36 DECwriter II	$1,095	$105	$ 58	$ 40
	LA34 DECwriter IV	995	95	53	36
	LA34 DECwriter IV Forms Ctrl.	1,095	105	58	40
	LA120 DECwriter III KSR	2,295	220	122	83
	LA120 DECwriter III RO	2,095	200	112	75
	VT100 CRT DECscope	1,695	162	90	61
	VT101 CRT DECscope	1,195	115	67	43
	VT125 CRT Graphics	3,295	315	185	119
	VT131 CRT DECscope	1,745	167	98	63
	VT132 CRT DECscope	1,995	190	106	72
	VT18XAC Personal Computer Option	2,395	230	128	86
TEXAS INSTRUMENTS	TI745 Portable Terminal	1,595	153	85	58
	TI765 Bubble Memory Terminal	2,595	249	138	93
	TI Insight 10 Terminal	695	67	37	25
	TI785 Portable KSR, 120 CPS.	2,395	230	128	86
	TI787 Portable KSR, 120 CPS	2,845	273	152	102
	TI810 RO Printer	1,695	162	90	61
	TI820 KSR Printer	2,195	211	117	80
LEAR SIEGLER	ADM3A CRT Terminal	595	57	34	22
	ADM5 CRT Terminal	645	62	36	24
	ADM32 CRT Terminal	1,165	112	65	42
	ADM42 CRT Terminal	1,995	190	106	72
DATAMEDIA	EXCEL 12 CRT Terminal	1,695	162	90	61
	EXCEL 42 Smart Buffered CRT	995	96	54	36
	COLORSCAN 10 Color CRT	3,195	307	171	116
TELEVIDEO	925 CRT Terminal	850	82	46	31
	950 CRT Terminal	1,075	103	57	39
NEC SPINWRITER	Letter Quality, 7715 RO	2,895	278	154	104
	Letter Quality, 7725 KSR	3,295	316	175	119
GENERAL ELECTRIC	2030 KSR Printer 30 CPS	1,195	115	67	43
	2120 KSR Printer 120 CPS	2,195	211	117	80
HAZELTINE	Executive 80/20	1,345	127	75	49
	Executive 80/30	1,695	162	90	61
EPSON	MX-80 F/T Printer	745	71	42	27
	MX-100 Printer	895	86	48	32
TIMEPLEX	E0400 4 Channel Stat Mux	1,525	147	82	55
	E0800 8 Channel Stat Mux	2,050	197	110	74

FULL OWNERSHIP AFTER 12 OR 24 MONTHS • 10% PURCHASE OPTION AFTER 36 MONTHS

MICROCOMPUTERS

APPLE • COMMODORE • HP85 • DEC LSI 11

ACCESSORIES AND PERIPHERAL EQUIPMENT

ACOUSTIC COUPLERS • MODEMS • THERMAL PAPER • RIBBONS • INTERFACE MODULES • FLOPPY DISK UNITS

TRANSNET CORPORATION

1945 ROUTE 22 • UNION, N.J. 07083 • (201) 688-7800

TWX 710-985-5485 800-526-4965 OUTSIDE N.J.