An addressing space larger than main storage—virtual storage—in System/370 challenges the operating system architect to design a disk operating (DOS/VS) for superior program execution performance.

Compared are the bases for program execution by DOS/VS with those of the earlier Disk Operating System. Increased system versatility is presented in terms of storage management, channel management, and program management.

Functional structure of IBM virtual storage operating systems Part III: Architecture and design of DOS/VS

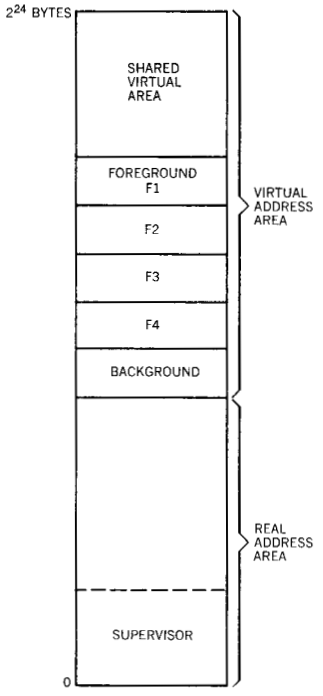
By J. P. Birch

The disk operating system for virtual storage in System/370—DOS/VS—may be considered as an extrapolation of the disk operating system for System/360—DOS—into the realm of IBM computers that are equipped for virtual storage. DOS/VS is discussed in this paper from the points of view of its design and structure and its application to the System/370 models equipped for virtual storage. Virtual storage, which is an address space that is larger than main storage, is discussed as it applies to DOS/VS.

Inasmuch as a new line of hardware has evolved, one may ask whether this might not be the appropriate time to introduce a more uniform line of virtual storage operating systems. The answer is one of the themes of this paper: the compatibility of DOS/VS and DOS. With such compatibility, the user can extend his application capability through the use of virtual storage, and at the same time build upon and execute more efficiently his present base of DOS application programs.

Another theme is that of the versatile capabilities of the operating system, which permit the programmer to concentrate more freely on applications. Many environmental considerations that impede DOS programmers—such as the planning of overlays, sizes of partitions, and knowledge of the partitions in which application programs are to run—have been greatly ameliorated in DOS/VS. Also discussed are ways in which decisions formerly

Figure 1 DOS/VOS virtual storage map



left to the system operator at job execution time have been subsumed by the design of DOS/VS and by the programmer.

With the introduction of System/370, the relative costs of real storage, central processing unit power, direct-access storage device speed and capacity have altered with respect to each other as compared with System/360. Thus another design criterion of DOS/VS is that of optimizing the effects of the new system components so as to minimize job execution cost.

Because of the architectural orientation of this paper, system performance is discussed from a relative point of view, i.e., as a basis for choosing design options that are expected to maximize performance in particular cases. In a System/360 DOS batch processing environment, the number of jobs run per shift or per day has often been a more important performance measure than the speed of running a given individual job. In a mixed batch-teleprocessing environment, improved response time has been the strategy. Of course, in any environment, the higher priority job should acquire system resources ahead of lower priority jobs. A system objective of DOS/VS is to use system resources with greater efficiency so as to achieve better balance between higher and lower priority jobs.

Operating modes

real mode

The power of virtual storage and facilites for using it are embodied in System/370 Models 115, 125, 135, 145, 155II, and 158. A storage map for virtual storage in these models is shown in Figure 1. The size of the real address area (real storage) is specified for each of these models with a complementing virtual address area to constitute a total storage size of up to 16 million bytes. The System/370 Models previously mentioned can operate in either the real mode or the virtual mode. The real mode of DOS/VS is similar to the DOS environment in that paging is not involved. A type of application for which the real mode is prefered is one in which there are time dependencies, such as magnetic character reading and processing. Here the time available for processing is relatively fixed, and there is no time available to bring in pages from peripheral storage. Therefore, all pages in such fixed-processing-time applications are held in real storage.

virtual mode

In the virtual mode, the system may execute in as many as five variable-size multiprogramming partitions, consisting of four foreground programs (F1-F4) and one background partition, all ranked by priority. Virtual addresses within a program are converted to real addresses within main storage by the Dynamic Address Translation, DAT, facility which is similar in principle

for all models of System/370. Channel programs are translated from virtual to real addresses by virtual storage management, which is discussed in the next section in conjunction with the channel indirect addressing feature of the channels and integrated device adapters and controllers.

The shared virtual storage area shown in Figure 1 is used to hold access methods and a list of pointers (Directory) to a Core Image Library (programs) instead of each application program having to hold these access methods separately. Shared virtual storage is discussed in more detail later in this paper.

Programs executed in the virtual mode are paged in and out of main storage in 2K-byte pages, which is to say that only those parts of the active programs that are required for immediate execution need be in real storage. As other parts of the programs are required, DOS/VS is designed to page in the necessary parts.

partition design

Studies of DOS revealed several problems experienced by users that could be eliminated in the DOS/VS system design. The constraint of having three fixed-size partitions had been a limitation on users who were running larger programs. Correspondingly, the fixed-size partitions had used storage inefficiently when smaller programs were being executed.

One solution to the large-program limitation had been the structuring of overlay programs. The difficult technique of overlay programming had only to do with adapting a program to its environment and had nothing to do with the application. To avoid the difficulties of overlaying, some users found that it was preferable to divide a given application into several steps and run each step separately.

In teleprocessing applications, as another case-study example, it was found that partitions tended to have large real-storage requirement variations. Real storage was found to be wasted when traffic was light because it continued to hold unused code and data. When traffic was heavy, teleprocessing applications often suffered response-time degradation because it was not possible to retain all the required code and data within real storage.

paging algorithm

Experiences such as these motivated the decision to design virtual storage support around five variable-size partitions within the 16 million byte (maximum) virtual storage configuration. The designers also concluded from their studies that in many cases it would be possible to increase the level of multiprogramming with little or no overcommitment of real storage by increasing the available storage via storage paging. The specific storage paging algorithm is based on the following criteria.

1. *Minimum number of pageins* is of first importance because a task or partition cannot execute until a pagein is complete.
2. *Minimal number of instructions in the paging algorithm* is vital because the CPUs of the models in which DOS/VS runs are relatively slower than those in which OS/VS1, OS/VS2, and VM/370 execute. (Thus the overall DOS/VS supervisor size and number of instructions executed in the supervisor are crucial to maintaning optimum operation rates.)
3. *Minimum number of pageouts* is also very important because paging I/O competes with other I/O operations for channels and the control unit.

Also, DOS/VS does not include pageout ahead. Therefore, in some cases, it may be necessary to do a pageout operation before a pagein operation.

A working-set algorithm is used in DOS/VS because storage is expected to be used cyclically. A *working set* is the set of pages that have been used during a fixed period of time. Pages occupy specified spaces, called *page frames*, in main storage.

The working-set algorithm operates as follows. A list of pages used during the given time period is maintained, i.e., the working set. Frame numbers of pages that are not in the working set are kept in another list. These latter pages are eligible for replacement and are placed in two queues in first-in-first-out (FIFO) order as follows:

1. Pages that are unchanged since the last time they were brought into main storage, and that also have not been referenced during that time period.
2. Pages that have been modified since the last time they were brought into main storage, but not referenced during the time period.

The algorithm first replaces pages in the first queue in FIFO order and updates a page-frame table. When the first queue has been exhausted, the procedure is to pageout from the second queue and update the table.

The working-set algorithm is one of the Class 2 algorithms described by Belady.[1] This simply means that information is kept about pages in main storage as opposed to not keeping such information (Class 1), on the one hand, or extending the record keeping to peripheral storage, (Class 3), on the other hand.

The pure first-in-first-out algorithms that were studied were found to adjust poorly to the cyclic nature of the application programs being considered and, therefore, page selection of FIFO algorithms was poor. Algorithms based purely on Least Recently

Used (LRU) criteria tend toward excessive pageouts, because they do not give sufficient weight to unchanged pages. More complex algorithms such as those that use anticipatory pageout techniques and a multialgorithm approach were found to excessively increase the resident supervisor size, relative to the benefit gained.

Thus the chosen algorithm has been found to meet the algorithm-selection criteria very well, and has also been especially efficient in operations involving overcommitment. It has been estimated that some large programs without overlays and overcommitted by three hundred percent may run better than programs with overlays and overcommitted by twenty percent. Programming for good performance in a virtual storage environment is discussed in greater detail in the DOS/VS *System Management Guide*.[2]

The page replacement algorithm selected for DOS/VS has also been found to meet the minimum-pageout criterion, and the control program size remains satisfactorily small. Interference among paging I/O operations, other I/O operations, and processing meets design specifications. Paging I/O is discussed in greater detail in the section on channel management.

storage management

Regardless of the efficacy of the paging algorithm, there may come a point at which there is no longer sufficient real storage available to execute all active programs concurrently. Allowing the system to attempt to continue executing under these conditions is likely to cause an overall system performance degradation. DOS/VS, however, has been designed to detect excessive overcommitment and to take corrective action by suspending the execution of the lowest-priority virtual partition. This action frees real main storage for executing the remaining programs. If the suspension of one partition does not improve performance to within system tolerances, suspensions continue until only one program is being executed. Suspending other partitions in a batch-teleprocessing environment when message traffic is at a peak not only gives added real main storage to the teleprocessing partition, but also gives that program more CPU time, faster access to control program services, and faster access to devices. The partition suspension action is taken on a priority basis, with the lowest-priority partition being suspended first, because, prior to the low-priority partition suspension, it is competing (albeit on a low-priority basis) with the teleprocessing partition for the CPU, I/O devices, and control program services.

paging I/O

DOS/VS attempts to overlap paging I/O with other processing. Since, in other multiprogramming environments, the system can be successful in overlapping I/O with processing, little effort has been given to trying to optimize the paging I/O itself. Although

paging I/O is overlapped, no sophisticated I/O techniques—such as dynamically modifying channel programs while they are running—are used. Keeping the Control Program size small is believed to be more important. The accomplishment of this objective tends to further reduce the need for paging because more real storage is available to the application.

When paging I/O is occurring in one partition, the system switches to another partition. A similar situation exists when there is multitasking within a partition, where the system switches tasks if it encounters a page fault within the task that is currently executing. As part of our system design approach, we took account of the fact that many DOS users had written their own multitasking support. To avoid forcing users to convert their own multitasking support, which would create a compatibility problem, a user exit has been provided to allow user-written multitasking mechanisms to be aware of page exceptions and pagein completions. In this exit, the user-written multitasking mechanism should be able to do its own task switching so as to overlap paging I/O operations.

The overlapping of paging I/O with other processing becomes more complicated when a page fault occurs while a Supervisor service is being processed. The worst case is to lock the entire Supervisor. There are two ways to prevent this. The first is to make the function re-entrant. (In Supervisor routines that are not re-entrant, when one task wants to use the Supervisor another task cannot do so.) In re-entrant Supervisor routines, if a page exception occurs while a Supervisor function is being processed for one partition, that partition waits until the page is brought into real storage. While the first partition is waiting for the page to be brought in, another partition can execute the same Supervisor function. Many frequently used Supervisor functions are re-entrant. Where re-entrance is not practical, a gating-locking structure is used to lock a specific portion of the Supervisor while paging is in progress. Should another partition attempt to use a locked function, it is queued up on the lock, and the system attempts to execute another partition.

Channel management

Mentioned earlier in this paper is the fact that computer systems that use DOS/VS are equipped with dynamic address translation (DAT) capability in the CPU, and that data addresses for instructions are translated by the CPU. Before channel programs can be executed by the channel, the Supervisor must translate virtual channel program addresses into real channel program addresses because the channel does not have address translation capability. An alternative approach, considered but rejected for lack of

Table 1 Value of increased blocking and double buffering in three cases of file maintenance

	CPU time	*Value of increased blocking*	*Value of double buffering*
Case 1	Low	High	Medium
	Medium	Medium	Medium
	High	Medium	Negligible
Case 2	Low	Negligible	High
	Medium	Medium	Medium
	High	Medium	Negligible
Case 3	Low	Medium	Medium
	Medium	Medium	Medium
	High	Medium	Negligible

DOS-DOS/VS compatibility, is that of requiring the user to recompile (and in some cases recode) his programs so that they can be executed under DOS/VS.

channel programs

Within the technique chosen, data management access methods have been designed into DOS/VS to minimize the effect on throughput of channel program translation. The basic concept is to translate channel programs only at the time the I/O request is issued (EXCP time). In deciding on this approach, a method was considered and rejected that would have required a second translation at the time the physical I/O command is issued. Such an approach would have caused a delay in reinstructing the channel, which would have been especially noticeable in the DOS/VS environment where relatively high-speed I/O devices (e.g., the IBM 3340 disk storage unit) are sometimes matched to relatively low-speed CPUs (e.g., System/370 Model 115).

The tradeoff is the flexibility of altering the channel program after the EXCP has been issued. This is regarded as a fair exchange, since channel program alteration is typically not done in DOS/VS access methods. In those rare cases for which it is necessary, the access method locates and alters the translated copy of the channel program.

blocking and double buffering

The effect and value of blocking and double buffering differ in the DOS environment from DOS/VS, where I/O is relatively fast compared to the CPU. User experimentation is required to optimize the effect of increasing the blocking factor and using double buffering under these conditions. Our experiences with three cases of I/O operations are given in Table 1. These cases are summarized as follows:

Case 1. One file is used more than the others, as exemplified by a program that merges files.

Case 2. Two files, both on the same channel, have greater usage than others, as in a file maintenance program with sequential input and output.

Case 3. Two files, on different channels, have greater usage than others.

In all three cases, of course, the CPU time required to process a record is an important consideration. In Case 3, the double buffering should first be tried on the slower device if the device types are unlike. Also, an increase in blocking factor or the introduction of double buffering may cause an increase in paging activity.

Despite the efficiency of channel program translation, there remain situations where serious degradation can occur. Programs with very high I/O rates—such as sorts, for example—tend to have high system channel translation overhead. Degradation is especially noticeable if the sort is running in a multiprogramming environment where the total CPU usage is high. DOS/VS has the capability of allowing user programs to do their own channel program translation, as exemplified by the IBM program SORT/VS. User programs rarely have I/O rates as high as that of a sort. Therefore, the system design favors the blocking and double buffering techniques previously discussed before attempting to use a more specialized channel program translation.

In addition to the classic benefit of multiprogramming—the ability to execute one program while another is waiting for I/O—another factor has been found to be very important in DOS/VS. The use of Rotational Position Sensing devices allows for the overlapping of multiple I/O requests.

When, in a multiprogramming system, an I/O interruption occurs during the processing of an EXCP, it is possible to reinstruct the channel quickly without the saving, restoring, and dispatching overhead that was required by DOS. DOS/VS had introduced a standardized saving and restoring of all registers throughout the Supervisor, whereas, in the DOS Supervisor, each routine selectively does the saving and restoring. Although the saving and restoring of all registers by DOS/VS are slower than selective saving, there is the overall benefit in a multiprogramming system of making it possible to branch from routine to routine in the Supervisor. The DOS/VS method is based on studies showing that, with this method, the total number of Supervisor instructions required to execute a job stream in a multiprogramming environment is lower than the number required for executing the same job stream in a single batch environment.

Program management

One of the functions of DOS that has been a performance and operational bottleneck has been that of the fetching of programs. At link editing time, it has been necessary to specify the starting address in main storage into which a program is to be loaded. This has meant that multiple copies of the same program would be required if a program were to be executed in more than one partition. Thus, if a subroutine were to be used in common by a number of programs, it would typically be link edited with each program. This has made the maintenance of such subroutines difficult. A cumbersome alternative in the DOS environment would have been for the user to write a self-relocating program.

relocating loader

In DOS/VS, operations scheduling is done by the operator, and a *relocating loader* loads programs into any of the five partitions. The relocating loader resolves the program addresses when it loads the program. The elimination of multiple copies of a program eases program maintenance.

program library and directory

The fact that a great deal of time had previously been spent in simply searching for programs is the reason that the ordering of programs in the library has had a dramatic effect on system performance. In DOS, both the Program Directory (pointers) and the programs themselves (Core Image Library) are maintained in first-in-first-out order.

The fetching mechanism and library structure have been changed in DOS/VS to improve system performance. The Core Image Library Directory (pointers) is now maintained in alphabetic order and the programs (Core Image Library) are held in first-in-first-out order. Thus a gain has been achieved in searching for programs in the library. Although this means that the earlier format of DOS private libraries is no longer usable, the gains are considered worth that degree of incompatibility.

Although the program search mechanism has been improved in DOS/VS particularly for the environment in which there is a large real storage and a large amount of fetching, a program search mechanism cannot be eliminated entirely. The System Directory List, which contains pointers directly to the program modules, resides in virtual storage. Fetching performance is thus improved, especially for transients and program phases that are frequently called by an active program.

shared virtual area

The storing of copies of programs used in common by a number of application programs in each application program partition, as is the case with DOS, has been found to be wasteful of real storage. Therefore, the shared virtual area is provided in DOS/VS to allow such commonly used programs to be shared and concur-

rently executed by other programs in multiple partitions. As was mentioned earlier in this paper, the shared virtual area can hold access methods and pointers to the Core Image Library.

Job management

Improved job management techniques in DOS/VS have contributed greatly to the increase in jobs processed per day over our previous experience with DOS. A large contributing factor toward improved job throughput efficiency is increased operator efficiency. In the DOS environment, the operator is required to handle cards and make a great many object-time decisions. Incorporated into DOS/VS is a library of cataloged procedures that smooth job-to-job transitions especially for production jobs. *Job control statements*, which are sets of control statements that cause a job to be executed, can be placed in the procedure library to greatly reduce operator card handling and typing. The user need not be aware of the system configuration or understand the job control statements to begin executing a job by using a cataloged procedure.

DOS requires the explicit specification of physical device addresses with the requirement that, for example, the operator assign specific devices at object time or that a program can run only in one specific partition. DOS/VS incorporates the capability of *generic device allocation*, by which the operating system assigns the devices. Generic device allocation allows the job control statements (including catalog procedures) to specify a device type—such as a tape unit—and volume serial number, if necessary. DOS/VS is designed to make the logical decision of which physical device to assign. Besides increasing operator efficiency, generic device assignment also improves multiprogramming efficiency.

Concluding remarks

DOS/VS represents a significant set of enhancements to the Disk Operating System. The addition of virtual storage support, additional partitions, and the relocating loader, among other items, extends the ability of the user to perform multiprogramming operations on System/370. Demands on the system operator are reduced, as are those on the scheduling function at the user installation. This type of operation, however, may require some changes in thinking by many DOS users, especially in considering factors that affect the total productivity of an entire computer installation.

CITED REFERENCES

1. L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal* **5**, 2, 78–101 (1966).
2. *IBM DOS/VS System Management Guide, IBM Systems Reference Library*, Form GC33-5371, IBM Corporation, Data Processing Division, White Plains, New York 10604.