Sharing Data and Services in a Virtual Machine System

J. D. Bagley, E. R. Floto, S. C. Hsieh and V. Watson†
Computer Science Department
IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Experimental additions have been made to a conventional virtual machine system (VM/370) in order to support a centralized program library management service for a group of interdependent users. These additions enable users to share read/write access to a data base as well as processing services. Although the primary motivation was the enhancement of performance, considerable attention was given to retaining the inherent advantages of the virtual machine system. Extended applications of the basic technique are described and the implications of such extensions on operating system design are considered.

Key Words and Phrases: Operating systems, file sharing, virtual machines, multiprogramming, virtual machine monitor, data security, intermachine communication, multiprocessors

CR Categories: 4.32, 4.34, 4.35

Introduction

Virtual machine systems have found application in two rather distinct areas; system program development and general purpose conversational time sharing. Here we are concerned mainly with the latter application area.Experimental additions have been made to a conventional virtual machine system (VM/370) in order to support a centralized program library management service for a group of interdependent users. The emphasis is on the utility of the enhancements while retaining the advantages of virtual machine systems such as simplicity, understandability and efficiency.

One way of approaching modern computing systems is to take a user's view of the interface that the system presents to the world. Buzen and Gagliardi [2] have described this interface as an Extended Machine which is the "bare (hardware) machine" plus the "privileged software nucleus". Figure 1 is an abstract diagram of a computer which shows the symbolic notation used in this paper. That notation is used in figure 2 to illustrate the extended machine concept. Most programmers (those who are not specifically systems programmers) work with the extended machine interface. This concept of a hypothetical extended machine is closely related to the notion of an abstract machine [6,7], although the emphasis is somewhat different.

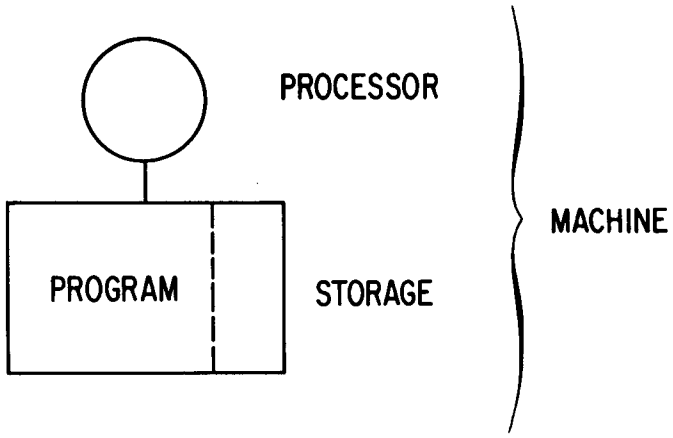

Figure 1. Symbolic Computer Notation.

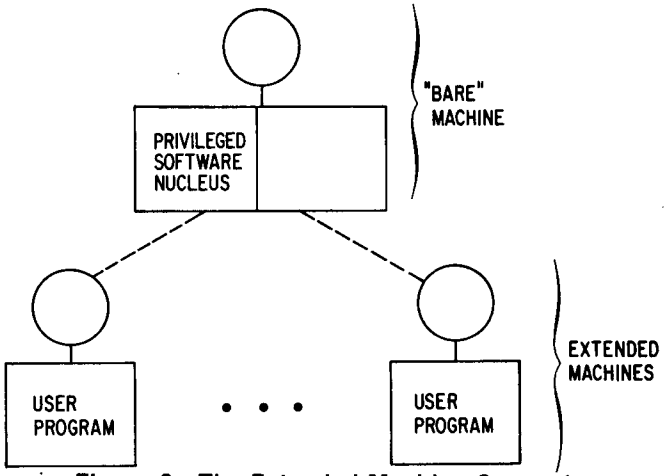

Figure 2. The Extended Machine Concept.

†Present address:
IBM Research Laboratory
San Jose, California 95193

In the special case in which the extended machine is identical, or nearly identical, to some real machine it has been described in the style of a Principles of Operation or a Reference Manual of the type used to describe real hardware products. In this case, the extended machine is termed a ***virtual machine***.

In this paper, a ***Virtual Machine System*** (***VMS***) is a combination of hardware, the ***Host Machine*** (***HM***) and software, the ***Virtual Machine Monitor***, (***VMM***) which provides an interface to the user which can conveniently be described by a Principles of Operation document. The term ***Virtual Machine Operating System*** (***VMOS***) is used to refer to operating systems that run in virtual machines. Figure 3 is an illustration of a standard VMS. Although the VMM occupies the same position as a nucleus or kernel [12,23] in the system structure, its function has usually been defined differently. Note that if the virtual machine in question happens to be identical to some real machine, then the virtual machine operating system can be any operating system that can be run in the real machine.

Applications of Virtual Machine Systems

The first application for virtual machines is in the area of the development, testing, and measurement of software (generally operating systems) which would otherwise require the exclusive use of a real machine. In this application, it is essential that the virtual machine be identical to the real machine that is the intended host for the software that is being developed. With certain exceptions [20], current VMS's meet this requirement and provide a facility that is the equal of the real machine and may even be superior to it because of the added functional convenience that the VMM can provide in its implementation of a simulated system control panel. Goldberg [9] lists many virtual machine applications in the software development area and gives a comprehensive survey of research in the field of Virtual Machine Systems.

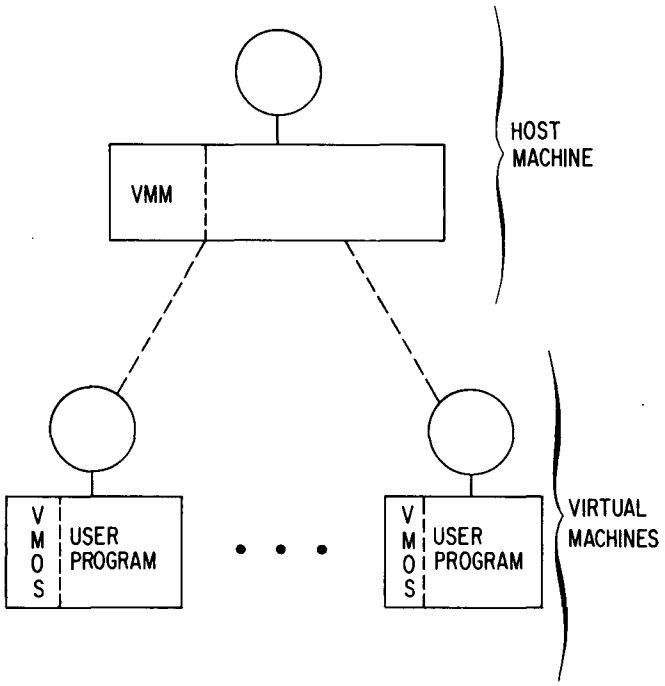

Figure 3. A Conventional Virtual Machine System.

The second application for virtual machine systems is the area of general purpose, conversational, time sharing. Here, it is of little consequence that the extended machine is identical to some real machine, because the user's interface is likely to be a secondary extended machine provided by a VMOS. The VMOS (for example, CMS in a VM/370 system) is designed to supply convenient, efficient and high quality services to a single user. It is limited in the quantity of user services that it can provide by the amount of computer resources which are allocated to its virtual machine by the VMM. These are limited by the efficiency of the VMM and thus questions of efficiency and performance are important to the general user and it is appropriate to consider them here.

VMS Performance

A prerequisite for the efficiency of a VMM is that the virtual machine interface which is supported be sufficiently similar to the real host machine on which it is implemented that the vast majority of the instructions of the virtual machine are executed by the real machine rather than being simulated by programs resident in it. In fact, in some instances, the underlying machine has been altered in order to make it more like the desired virtual machine [13,22]. Virtual machine systems derive a great deal of their efficiency from two other facets of their basic design: specialization and simplicity.

The specialization arises from the combination of the VMM, which controls the multiprogramming and the distribution of the real resources, with the VMOS, which supplies services and the man-machine interface to the end user. This separation of function makes it possible for each of these aspects to be optimized independently. The functional independence does not imply operational independence. That is, the performance of the VMOS depends directly on the allocation decisions made by the VMM, and if the VMM makes the wrong decisions performance may be severely crippled. The VMM does its resource management on the basis of the information it receives across the interface from its virtual machines. If this information is insufficient or erroneous, the efficiency of the system will be impaired.

Simplicity is provided by the stable, well designed interface which is supported by the VMM. The existence of this interface eases the job of the VMM system designer considerably. For not only is the interface relatively uncomplicated, but, more importantly, it is defined in advance by the Principles of Operation of the virtual machine it supports. This means that the environment that the VMM must provide is well defined, tested, and stable. Note that this may be considerably different from the situation faced by many software designers who are forced to implement toward an extended machine interface whose definition is very complex, possibly incomplete and inconsistent, and inevitably destined to change before the scheduled delivery date.

There is bound to be some tension between specialization and simplicity of virtual machine systems, for in order to increase the amount of information available to the VMM so that efficient resource allocation decisions can be made, it may be necessary to complicate the interface. This must be done with great care and is part of the art

and science of operating systems design. The extensions described herein have been carefully designed to add minimal complication to the virtual machine interface while preserving the facets of VMS's which lead to good performance for the general user.

Sharing In a Virtual Machine System

All general purpose time sharing systems offer their users software resources as well as a share of the hardware resource. In a VMS, the software resources are supplied by the VMOS component while the hardware resources are supplied by the VMM component. In general purpose time sharing systems of the "computer utility" type [1,17,21], the management of software and hardware resources is integrated. One of the consequences of this integration is that sharing of software resources (programs and data) among independent users is facilitated. In contrast, the isolation of resource control from user services that characterizes a VMS makes the task of sharing these resources more difficult.

The user interface to a general purpose time sharing system normally provides a set of functions for symbolic file management which enable the user to create, manipulate and share his programs and data. In a VMS like CP-67/CMS or VM/370, those functions are provided by the file mechanism of the VMOS (CMS). In time, the user becomes accustomed to manipulating programs and data in the form of VMOS files and it is these files that he wishes to share with other users of the same time sharing system. Unfortunately, the VMM component of the VMS "knows" about the other users of the system, but it is ignorant of the user's file structure; while the VMOS component of the VMS "knows" about the user's file structure, but it is ignorant of the other users of the system. The division of labor of the VMS, which proved to be an advantage as far as implementation is concerned, proves to be a disadvantage when it comes to the important service of sharing files.

The VMM, which manages the virtual resources of all of the users, can be used to implement modes of sharing among virtual machines. These modes of sharing may be thought of as the sharing of the virtual hardware and have been implemented in VM/370 both for the main storage (shared segments) and for the auxiliary storage (shared mini-disks) of a user's virtual machine [15]. A recent experimental system [10] has introduced read-write shared virtual storage segments. Although the sharing of virtual hardware can suffice for many purposes, additional function must be added to provide true logical sharing.

Mini-disks are virtual disks which differ from real disks only in that they may have fewer cylinders than a physical disk. They be shared among several users and the sharing may be initiated dynamically on the initiative of the users themselves. Mini-disks are owned by users and the owner may specify passwords to give others various degrees of access (read-only, read-write, etc.). In VM/370 the process of attaching a mini-disk to a virtual machine is called "linking".

But the ability to read from or write to a mini-disk is of limited value if the contents and location of the files it contains are not known to the VMOS. One standard method of resolving this difficulty is to store a directory to the disk contents at a fixed location on the disk itself. Then the VMOS must initially read the directory into its virtual memory before it can access the files on the disk. In CMS, this process is called "accessing" the disk.

Application To SBS

As part of an effort to provide software tools for the use of the developers of an experimental operating system, an extensive software system for automating the management of programming development called SBS [5] was created. It is concerned with the automation of the information collection and dissemination that accompanies a large program development project. It contains a data base which encompasses source, object, and load files as well as information relating versions, ownership, the composition of systems and subsystems. SBS was constructed to support a user community which shares and contributes to a common and growing data base. In addition, SBS contains facilities to aid in maintaining administrative control over the developing project.

SBS was initially implemented on a CP-67/CMS [18,19] base and used the basic features of that system to share programs and data among its users. Experience with the system led to the conclusion that a substantial performance improvement could be obtained by altering the manner in which this sharing was accomplished. To perform an SBS function in the initial implementation, the user entered an SBS command through CMS, the VMOS. This loaded an SBS object program from a shared file into the user's virtual memory and executed it. The SBS program in turn issued commands to CP, the VMM, which linked the SBS data disks to the user's virtual machine as shown in figure 4. If the SBS data disks were unavailable (usually because they were linked to some other user), the program waited for them to become free because uncontrolled multiple write links could not be permitted to the SBS data disks. Once the SBS data disks were linked to the user's virtual machine, the CMS files on them were accessed by the user's version of CMS.

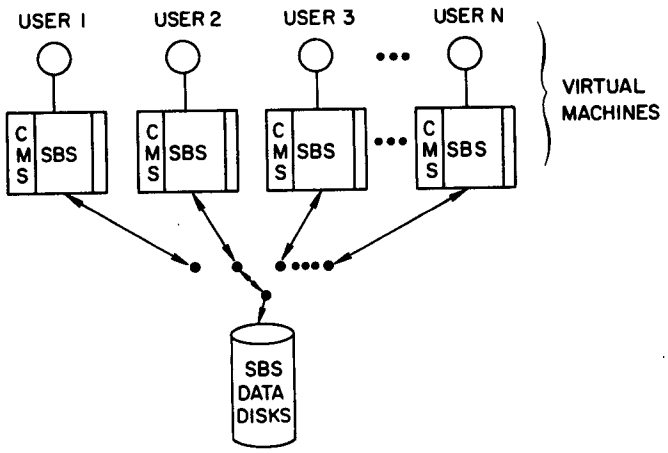

Figure 4. Sharing Data By Sequential Disk Linking.

After this initialization, SBS processing was begun and files were created and transferred to or from the user's minidisk and the SBS data disks using standard CMS disk operations. When the processing was completed, the SBS data disks were detached from the user's virtual machine

and control was returned to the CMS command interpreter.

Several opportunities for performance improvement were identified and are noted here. The first involves the practice of loading the SBS program modules into the user's virtual machine for each command. The SBS modules tend to be quite large (megabytes) and the time and resources required to load them into the user's address space can be considerable even though relocation is unnecessary. The second area where the promise for improvement exists lies in the way in which CMS gains access to the disks. Since the disks containing SBS data are shared by all users and and may be changed by other users between the SBS commands issued by any given user, each SBS user must access all of the SBS data disks each time he executes an SBS command. The third consideration is security. Since the user is allowed to put files on the SBS data disks, they must be linked to his machine with write privilege. In the event of an error or an asynchronous interrupt, control of an SBS disk may be given directly to the user (without the intermediary SBS routines). Clearly, this intolerable situation could not be allowed to exist. The fourth problem is lockout. Since the unit of sharing is an entire SBS data disk, only a single user can be given access to the disk at any one time. All other users must wait until that user has finished even though none of them may reference the same files.

A Proposed Configuration

The outline of a proposed configuration is illustrated in figure 5. It shows a VMS which contains a dedicated virtual machine (the SBS machine) which alone has access to all of the SBS files. The SBS machine contains all of the SBS modules and the SBS file directories already loaded within its address space. When a user wants to perform an SBS function, he sends his request to the SBS machine and accompanies that request by the data necessary for its fulfillment. The SBS machine then carries out the requested function and returns the results to the user.

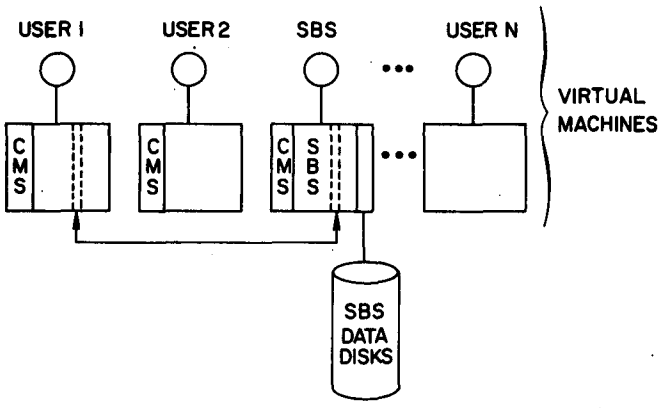

Figure 5. Data And Services Provided By A Dedicated SBS Machine.

In order to implement this configuration, two additional functions must be provided by the VMS. The first allows independent machines access to each other's address space in a way which is controlled, but less restricted than through preplanned shared segments. This function may be implemented by changes within the VMM and provides for the exchange of data between address spaces in an efficient manner by changing page table entries and not by actually moving the data. Second, a resource transfer function must be provided to enable the user to transfer some of his resources (i.e. his time slice) to the SBS machine. This is a special function which is executed by a users virtual machine in order to influence the VMM in its allocation and attribution of the resources of the real machine. The resource transfer function serves three purposes: first, it enables the SBS machine to obtain exactly the class of service to which its current user is entitled; second, it allows the resources consumed by the SBS machine to be charged to its current user; and third, it allows the resources of the system to be distributed in an equitable manner, since if 25% of the VMS's users are using the SBS machine, it should be given 25% of the systems resources and not merely the resources which would normally be allotted to another virtual machine.

This configuration offers the potential for improvement in all of the areas enumerated above. The SBS modules stay loaded in the SBS machine since that machine is used for the sole purpose of executing SBS routines. In addition since each disk is referred to by one and only one machine, there is no need continually to regain access. The user's files remain in his machine and the SBS files in the SBS machine, and as a result the SBS routines have the means to enforce their security restrictions. Finally, since the files are referred to by only one machine, many files may be opened simultaneously without confusion. The implication is that the SBS machine can process requests on behalf of many users concurrently.

The Multi-User Problem

There is, however, still a lockout problem since when the SBS machine is occupied with the request of one user, it must make other users wait for service. This occurs because SBS was built on CMS, a single user operating system that has no provision for multiprogramming. This means that the SBS machine can be doing work for only one user at a time and that it can not begin work for a second user while it is waiting for the completion of an I/O operation for the first one.

One way to attack the multiuser problem is to install a multiprogrammed VMOS in the SBS machine. This may be done by altering CMS to handle multiprogramming or by altering SBS so that it works with an existing multiprogramming operating system. The first alternative involves a good deal of labor, since CMS was designed specifically to be a single user operating system. The second alternative is also unattractive because SBS is closely tied to CMS. In addition, this course has certain inherent disadvantages, since multiprogramming operating systems may to cause performance problems for VMMs [24,8]. This occurs because, among other reasons, the multiprogramming of the VMOS may work at cross purposes to the multiprogramming of the VMM.

In order to attack the multiuser situation while avoiding the problems the configuration illustrated in figure 6 was proposed. The intent of this method is to take advantage of the multiprogramming inherently provided by the VMM subcomponent of the VMS rather than opposing it. In order to implement this alternative, it is necessary to create another class of independently dispatchable virtual

machines. These machines, called "slaves", are created by the SBS machine and assigned to users for the duration of the SBS command or the for the length of an SBS session. The slave shares SBS data and modules with the master SBS machine and it shares the user's files with the user's machine. The SBS command is given to a master SBS machine which obtains a slave and assigns it to the user.

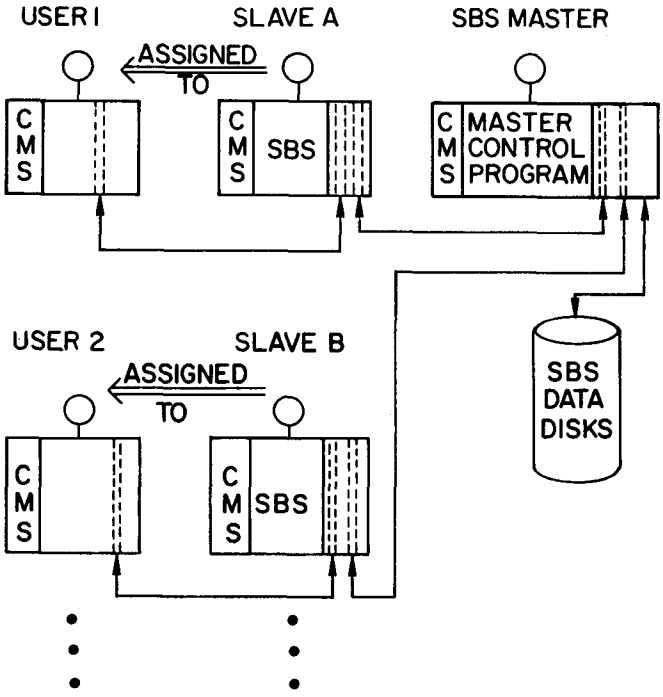

Figure 6. Centralized Data And Services Provided By Slave Machines.

The command is executed in the slave machine but its resources are charged to the user. The sharing implies that there is only one copy of the modules and one copy of the various file directories. More than one slave may be active concurrently and more than one may access the SBS disk concurrently. However, two slaves must be prevented from changing the same file at the same time. For this reason, a system of file locks must be provided in the VMOS.

It was decided to implement the configuration shown in figure 6 to provide increased performance for SBS. Doing so meant that a number of additional functional capabilities had to be added to the virtual machine system. The following sections contain descriptions of the building blocks which have been implemented both in the VMM and in the VMOS. Of course, there is a certain amount of practical trade-off involved in the decisions taken here. It has been our intention to make as few changes to the VMM and the to the VMOS as possible and still accomplish our objectives.

Additional VMM Functions

Intermachine Communication Functions

These functions facilitate the transfer of data between independent virtual machines. They are described fully in a paper by S. Hsieh [14]. The data are transferred directly between address spaces and not through the paths provided by the standard architecture as through a shared virtual I/O device or virtual channel to channel adaptor. Both the sender and the receiver must cooperate in the transfer of data and the interface functions are carefully designed to insure protection and maintain synchronism. A machine which wishes to send data to another machine must first get its attention by executing an instruction which causes a special external interrupt to be generated for the other machine. That machine responds by causing an external interrupt in reply. Messages are passed with each interrupt which convey information such as the size and location of the data block to be transferred. The actual transfer is accomplished only when both parties have given their approval.

Slave Functions

Users who have been granted a special "***creator***" privilege by the system administrator have the power to create other independent virtual machines and exert a limited amount of control over them. One of these users can specify the parameters of the slave machine he wishes to create by specifying a slave prototype which is the name of a user already admitted to the system. The slave that is created then is identical to the prototype except that it has a different name and it is owned by its creator. The creator can assign the slave machine to a new owner (to whom its resources will be charged) or it can maintain ownership of the slave itself. The owner of a slave can stop and start the slave machine and can read and write the contents of its internal storage (PSW, registers, etc.). The creator of the slave retains the power to reassign it to a different owner at any time. The power to create a slave machine and start it executing a program is precisely the power to initiate a process [6,12].

Miscellaneous Functions

The most important additional function is the AUTHORIZE function. If this function is executed by a virtual machine, it permits other machines in the system to cause an extended external interrupt for the authorizing machine. Conversely, if a virtual machine (a user) has not authorized the special extended external interrupts, it cannot be called by another machine and communication cannot occur between them. It is this function which controls the selective breaching of the walls of protection [16] provided by the VMM between independent virtual machines.

Additional VMOS Functions

Remote Files

Remote files provide a way for a VMOS to manipulate files that reside on a device belonging to a second virtual machine and accessed by its VMOS. Close cooperation is required between the VMOS's in the two machines so that file requests of the using machine are translated into messages which are transmitted to the owning machine. The owning machine then performs the appropriate file action and transmits the results back to the using machine. The user programs in the using machine obtain precisely the same results as if the file had actually been owned and accessed by the using machine.

Nothing happens in the owning machine unless that machine has issued the AUTHORIZE function. After that, it is liable to receive an interrupt requesting a record from one of its files. The interrupt identifies the requester

and the owner can check whether the requester has been authorized to receive the requested data and whether it can be safely sent (it may have been opened for writing by another requester). If all is well, the requested data are sent by means of the transfer function. All of this processing is accomplished in the owner's virtual machine by an additional interrupt routine which services the new class of external interrupts and does not require the conscious attention of the owner after he has authorized the use of his file by another user's virtual machine.

Master Machine Control Program

Besides being the formal owner of the SBS data base, the SBS master machine assists users by acting as the manager of the SBS resources. Programs residing in the master machine assign slaves to SBS users and create, activate, delete, and deactivate slaves as needed. In addition, they prepare the slaves for assignment by initializing them with the appropriate programs and obtaining for them the authorizations necessary for them to carry out their functions.

Conclusions and Extentions

This project may be viewed as an instance of adding a function (file sharing) to an operating system (VM/370) in an evolutionary manner. This runs counter to experience with much of operating system development where all of the needed functions must be designed into the system initially and the system is bound forever by the limitations of the vision of the initial design. From a broader viewpoint, the VMM has been extended to implement the creation and control of processes and communication and synchronism among processes. The result is to give the VMM the capabilities of a nucleus or kernel [12,23] of a general purpose multiprogrammed operating system.

A second purpose of this project was to capitalize on the clean interfaces of the VMS but to add shared services characteristic of a general purpose time-sharing system. This really brings to the fore a basic philosophical problem in operating system design which is: "How is the division to be made between those services that are provided centrally for all users of the system and those services that are provided locally for individual users?". On the one hand, if services are shared, the avoidance of duplication introduces certain economies. On the other hand, the more services that are shared, the more system overhead is required to manage them. One way in which the system overhead is made apparent is in supporting a complex interface. The wider the range of centralized services that are made available to users, the smaller the probability that any of them will be in use by more than one user concurrently, but the design must allow for the possibility of concurrency. In more general terms, this decision concerns the assignment of user functions to the appropriate level of abstract machine.

The net result is that the character of the operating system has been changed in the direction of providing more centralized services (ie. services have been moved to a lower level abstract machine). Instead of providing raw virtual machines (CP) and user services through an extensive local operating system (CMS), the altered system supplies many services by providing access to specialized virtual machines (the SBS machine, the communications machine, etc.) which may contain specialized operating systems. Extensions consist of providing additional common functions and perhaps replacing some of the functions currently provided by CMS (eg. language processors) by means of specialized virtual machines. Adding a function involves supplying a master machine to provide the basic function and synchronization and supplying additional slave machines to provide the multiprogramming features needed.

A third point of view sees this work as a simulation of a system which includes a hardware file processor [3]. The SBS machine and its slaves may be considered to be independent machines. This is similar to systems which include peripheral processors [4] or microprogrammable processors [11] to perform some portion of the file management function. The difference is that here the architecture of the file processor is identical to the architecture of the main processor, and the functions which it performs are "out in the open" in code rather than being buried in another processor or in hardware or microcode. The advent of LSI and the consequent reduction in cost of logic and storage has placed a greater premium on factors other than maximum utilization of hardware resources. This provides ample justification for the exploration of systems which are cleaner in design, more flexible and extendible, more reliable, and easier to understand.

This suggests a structure for systems which allow a rather fine gradation of cost and performance while retaining a fixed user interface. While there are only a relatively few users of a service, it can be implemented in each user's VMOS as was the initial implementation of SBS. As the usage increases, an enhanced system which includes dedicated single function virtual machines and slaves can be employed. Then as the usage increases even further, there may come a time when it is feasible to increase performance by adding a dedicated real machine which is specifically designed to perform the service. Extensions along these lines involve adding additional virtual or real processors for communications, display support, and terminal support. In order to provide for the possibility of such extensions, the interprocess communication functions must be carefully designed so that they support remote or external processes as well as local or internal processes. The CMS EXEC mechanism, which provides command language procedures, has proved to be sufficiently powerful to provide compatibility at the command level.

Acknowledgments. D. P. Rozenberg provided the initial impetus and many helpful suggestions in the preparation of this paper. P. G. Capek, W. E. Daniels Jr., S. P. DeJong, and C. J. Stephenson contributed a number of ideas. A. N. Chandra assumed the management of the project and saw it through to its termination. We are indebted to the referees for their constructive suggestions.

References

1. Bobrow, D.G., Burchfiel, J.D., Murphy, D.L., and Tomlinson, R.S. "TENEX, a Paged Time Sharing System for the PDP-10", *Comm. of the ACM, 15, 3,* (Mar. 1972), 135-143.

2. Buzen, J.P., and Gagliardi, U.O. "The Evolution of Virtual Machine Architecture", *AFIPS Conf. Proc. 42* (1973), 291-300.

3. Canaday, R.H., Harrison, R.D., Ivie, E.L., Ryder, J.L., and Wehr, L.A. "A Back-end Computer for Data Base Management", *Comm. of the ACM, 17, 10,* (Oct. 1974), 575-582.

4. Control Data Corporation. *Control Data 6400/6500/6600 Computer Systems SCOPE Reference Manual.* Pub. No. 60189400.

5. DeJong, S.P. "The System Building System (SBS)", RC 4486, IBM Research, Yorktown Heights, NY, (Aug. 1973).

6. Dijkstra, E.W., "The Structure of the "THE"-Multiprogramming System", *Comm. of the ACM, 11, 5,* (May 1968), 341-346.

7. Dijkstra, E.W., "Hierarchical Ordering of Sequential Processes", *Acta Informatica, 1,* (1971), 115-138.

8. Goldberg, R.P., and Hassinger, R., "The Double Paging Anomaly", *AFIPS 1974 NCC Conference Proceedings, 43,* (May 1974), AFIPS Press, Montvale, N. J., 195-199.

9. Goldberg, R.P. "Survey of Virtual Machine Research", *Computer, 7, (6),* (June 1974), 34-45.

10. Gray, J.N., and Watson, V. "A Shared Segment and Interprocess Communication Facility for VM/370", RJ 1596, IBM Research, San Jose, California, (May 6, 1975).

11. Hancock, R.J. "Microprogrammed Disk Peripheral Control Units", *1971 IEEE International Computer Society Conference Proceedings,* (Sept. 1971), 113-114.

12. Hansen, P.B. "The Nucleus of a Multiprogramming System", *Comm. of the ACM, 13, 4,* (Apr. 1970), 238-241, 250.

13. Horton, F.R., Wagler, D.W., and Tallman, P.H., "Virtual Machine Assist: Performance and Architecture", Technical Report TR 75.0006, IBM New England Programming Center, Burlington, Mass. (April 22, 1974).

14. Hsieh, S.C. "Inter-Virtual Machine Communication Under VM/370", RC 5147, IBM Research, Yorktown Heights, NY, (Nov. 1974).

15. IBM Corporation. *IBM Virtual Machine Facility/370: Introduction.* System Reference Library, Form GC20-1800.

16. Lampson, B.W. "Protection", *Proc. Fifth Princeton Symposium on Information Sciences and Systems, Princeton University,* (Mar. 1971), 437-443. reprinted in: *Operating Systems Review, 8, 1,* (Jan. 1974), 18-34.

17. Lett, A.S. and Konigsford, W.L. "TSS/360: A Time-Shared Operating System", *AFIPS FJCC Conference Proceedings, 33, 1,* (Dec. 1968), The Thompson Book Company, Washington, D.C., 15-28.

18. Meyer, R.A., and Seawright, L.H. "A Virtual Machine Time Sharing System", *IBM Systems Journal, 9, 3,* (1970), 199-218.

19. Parmelee, R.P., Peterson, T.I., Tillman, C.C., and Hatfield, D.J. "Virtual Storage and Virtual Machine Concepts", *IBM Systems Journal, 11, 2,* (1972), 99-130.

20. Popek, G.P., and Goldberg, R.P. "Formal Requirements for Virtualizable Third Generation Architectures", *Comm. of the ACM, 17, 7,* (July 1974), 412-421.

21. Saltzer, J.H. "Protection and the Control of Information Sharing in Multics", *Comm. of the ACM, 17, 7,* (July 1974), 388-402.

22. Tallman, P.H., Denson, R.A., Gilbert, T.A., Nichols, J.M., and Stucki, D.E., "Virtual Machine Assist Feature Architecture Description", Technical Report TR 00.2506, IBM Poughkeepsie Laboratory, (Jan. 9, 1974).

23. Wulf, W., Cohen, E., Corwin, W., Jones, A., Levin, R., Pierson, C., and Pollack, F., "HYDRA: The Kernel of a Multiprocessor Operating System", *Comm. of the ACM, 17, 6,* (June 1974), 337-345.

24. Young, C.J. "Extended Architecture and Hypervisor Performance", *Proc. ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems, Cambridge, MA,* (Mar. 1973), 177-183.