The architecture of a virtual machine system has specific advantages over that of conventional operating systems because virtual machines are well separated from one another and from the control program. This structure requires that a protected, multi-user resource manager be placed in a distinct virtual machine because the protection domain and scheduling unit are one entity, the virtual machine. But cooperation between distinct virtual machines necessarily entails scheduling overhead and often delay.

This paper describes an experimental extension to VM/370 whereby a distinct execution and data domain (Virtual Control Storage) is made available to virtual machines that require access to a resource manager, without requiring a change in the scheduling unit. Thus scheduling overhead and delays are avoided when transition is made between user program and resource manager. A mechanism is described for exchanging data between execution domains by means of address-space mapping.

Virtual Control Storage—security measures in VM/370

by C. R. Attanasio

For the purposes of this paper, the security of a computing system relates to its ability to perform according to design objectives and administrative policies, regardless of the use to which it is subjected, particularly in the face of conscious attempts to subvert its protection mechanisms. Examples of design objectives and administrative policies are the ability to allow access to data only on presentation of a password by the user or through the mediation of some system (or user) program, charging of each user according to a function of his central processor and his main and auxiliary storage usage, and scheduling the usage of system resources according to a "fair share" algorithm (or perhaps some other). The system's protection mechanisms are the components that enforce the objectives and policies. If those mechanisms are subverted, the objectives and policies are not met, and the system is said to be not secure. It is generally acknowledged that no commercially significant system can be confidently termed secure by this definition.[1–6]

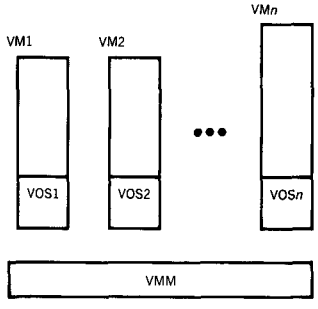
In this paper, *improving the security* (or *integrity*) of a system means taking steps to help make the system secure. The attainment of security is not claimed. There is no standard terminology in this area.[1,7]

It is widely thought that the security problems of existing systems arise from the absence of a structure designed for security[1,4,5] or from practical limitations on the ability to maintain that structure,[8] and that hopes for more secure systems lie in systems better structured to protect the resources entrusted to their control.[9–13] In particular, the architecture of virtual machine systems has been perceived to offer improved security because of the well defined separation between system and user programs[2] and, in some circumstances, because there are two levels of supervisory programs.[14]

This architectural separation, however, provides no particular support for the pairing of virtual machines when cooperation between distinct execution domains is desired. The unit of protection and the unit of independent scheduling are the same—namely, the virtual machine. This paper describes the design and implementation of a modification to IBM's Virtual Machine Facility/370 (VM/370)[15,16] wherein the user's virtual machine can obtain service from an execution domain that is part of the same scheduling unit, but is a distinct, protected domain. This protected domain, modeled on the concept of programmable control storage, is called *Virtual Control Storage (VCS)*. The main point of the work is to provide protected service to a virtual machine without entailing the expense and delay of transition between scheduling units. Other benefits result from greater convenience of software structure.

Virtual machine systems

Figure 1 Typical configuration of virtual machines controlled by one VMM.



A virtual machine system is generally understood to be one in which the interface presented to the user process is the same as that defined for the host hardware computing system. The control program, which creates multiple copies of the host hardware, is called the virtual machine monitor (VMM). Virtual machine systems are particularly useful in that they allow system programs (those designed to control bare hardware) to be written and tested in the normal VMM multiprogramming environment, rather than requiring stand-alone time for testing. They also allow applications written for different operating systems to be executed on the same hardware system, under control of the VMM. Figure 1 illustrates a typical configuration of n virtual machines (VM1 through VMn), each using a selected (virtual) operating system (not necessarily all different) on one hardware system controlled by one VMM.

Since the VMM provides the user with an unenhanced virtual machine, he can select an operating system appropriate for his use. Typically a spare, efficient interactive monitor system is available for normal usage, providing for an impressive level of multiprogramming for interactive users. VM/370 is the basis for the enhancement described in this paper. Goldberg[17, 18] has developed a formal description of virtual machine systems.

It is the author's opinion that the advantages of virtual machine systems arise from the simplicity and cleanness of the interface between the VMM and the virtual machines. This interface is defined by a *principles of operation* document. In contrast, systems such as MULTICS,[8] designed primarily for security, provide a high-level interface to the user and also maintain programming generality, thereby leading to a much larger privileged supervisor (*kernel* in MULTICS terminology). In systems such as HYDRA,[10] protection is obtained by structuring the objects and operations available to the user's programs. (These comparisons are meant to be illustrative only, not to imply relative values among the systems.)

Resource sharing in a virtual machine environment

Virtual machine systems are structured advantageously for security because of the strict separation of virtual machines at a basic level in the VMM. However, the separation makes resource sharing difficult or awkward, beyond simple read-only sharing. Initially, VM/370 provided mechanisms modeled on real systems—for example, the ability to pass virtual card images between virtual machines and to establish virtual channel-to-channel connections between virtual machines. More recently the Virtual Machine Communication Facility (VMCF),[19] which has no direct analog in real systems, has become available. Bagley et al.[20] and Gray and Watson[21] describe research efforts to extend the capability of VM/370 to support the sharing of data and services among virtual machines in a meaningful way.

As the use of VM/370 has become more widespread, the requirement for controlled sharing of resources has become greater. The way in which additional capability has been provided to virtual machines—for example, access to a shared data base—has been by establishing an additional virtual machine which owns the shared resource and mediates all access by the several users. Communication and data transfer are performed through one of the mechanisms discussed above. Bagley et al.[20] present a thorough description of one application structured in this way, as well as some proposed techniques for solving problems of scheduling and data communication within such a structure.

When a service is provided to many virtual machines by one server machine, the resulting structure has several important characteristics:

- Since the service is provided by cooperating, asynchronously executing entities (the user virtual machine and the server virtual machine), a general communication protocol is required. This protocol can be elaborate.
- Since the server responds to multiple requestors, it must have a strategy for scheduling its service for timely response to independent users. It must try to avoid inordinate delay to one user because of lengthy service to another. But scheduling the resources of a virtual machine can be different from scheduling real resources and may, in fact, be counterproductive.[22]
- The VMM schedules the server simply as a distinct virtual machine, perhaps with different priority from user machines, but it has proved quite difficult to schedule the server according to which user is being served (the most desirable way), since effective scheduling of interactive systems is based on individual virtual machine resource usage characteristics related to system-wide resource availability.
- Since there is only one scheduling unit (virtual machine) to serve multiple users, any event that blocks execution of the server machine, such as a page fault, an input/output wait, or a time-slice end, blocks service for all clients of the server.
- Transmission of data between user and server machines is nontrivial, involving interaddress-space data moves by the VMM, as well as storage management in the server.
- Since user and server are distinct virtual machines, communication between them involves additional scheduler action when the user must wait for service. In any case, there will be a delay between the time when the request is made and the time when the server begins to satisfy the request.

Each of the characteristics described above adds overhead or response time when providing the user with facilities beyond the virtual machine definition. The quantitative importance of each item depends on the service being provided, particularly the frequency of interaction between each user and the server, the number of users being served, and the ratio of system overhead to server resource utilization per communication.

Virtual Control Storage

Virtual Control Storage (VCS)[23, 24] is an experimental extension to VM/370 which addresses each of the above characteristics with a view toward minimizing VMM execution in support of user-to-server communication and minimizing response-time interference between users. It provides the virtual machine with added capa-

bility in a secure, architecturally coherent way, using the concept of programmable control storage.[25] The new capability (for example, access to a shared data base through a logical interface) is provided in the form of a new instruction executed by the user's virtual main storage (VMS) program. The instruction is emulated by a user-selected program in his VCS. Since the user can request access to the shared resource only through a VCS program in a very stylized way, the integrity of the VCS program and the resources it controls are protected by the fundamental structure of the virtual machine system.

The analogy with programmable control storage implies several features. The VMS program never has addressability to VCS, so the VCS program is protected through the fundamental virtual memory management mechanisms of CP, the virtual machine monitor of VM/370. (CP console functions[26] that display or change the virtual machine state do not operate on VCS.) However, VCS does have addressability to VMS, allowing easy communication between the two, always under control of the VCS program. The user invokes this new capability through a synchronous, instruction-like interface. A VCS program is sharable among multiple users simultaneously, as is programmable control storage, and similarly need not explicitly schedule its service to those multiple users.

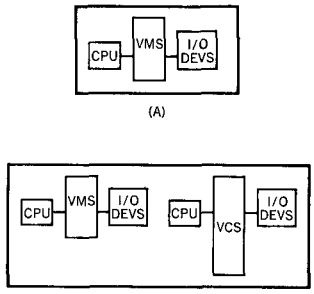
Departures from the programmable control storage analogy are dictated by the fact that the intended purpose of VCS (that is, the nature of the instructions emulated) differs significantly from the nature of the instructions emulated in real programmable control storage. Thus VCS has its own device space, in general disjoint from that of VMS, subject to the normal linking mechanisms in VM/370. VCS may have addressability only to a portion of VMS, depending on the limitations dictated by the finite size of the address space in System/370. Since VCS supports simultaneous access to a shared resource by multiple users, a VCS program must be able to serialize possibly conflicting simultaneous requests. This is the only explicit information it must have about its multiple users. In general, it is much less than the information required by one virtual machine that serves many users.

VCS provides a different perspective on each of the structural characteristics listed above, in terms of their potential capacity for inhibiting performance:

- Since the virtual machine's new capability is provided in the form of a new instruction, the service is provided in a synchronous fashion. Thus the communication protocol is (computer) instruction-like rather than communication-like, with parameters passed in registers or storage, and results returned

to the next sequential instruction in storage, registers, or condition code. This protocol should be much simpler to use by both requestor and server.

Figure 2 A standard CP virtual machine (A) compared with a CP virtual machine with VCS (B).



- A VCS program, although sharable among many users, executes as part of the user's virtual machine in each instance (see Figure 2) and need not explicitly serve multiple users so long as the program is re-entrant (in a weaker sense than normally—see below). With regard to scheduling, it is irrelevant to CP whether the virtual machine is executing the VMS or VCS program; its normal scheduling decisions are unaffected. In fact, while switching execution domains between VMS and VCS, no scheduler service is required, so there is no substantial CP involvement. Since the VCS program executes as part of the user's virtual machine while serving a particular user, any event that blocks the execution of the VCS program blocks only that user, not all users as in the case of a single virtual machine.
- Data communication between VMS and VCS is accomplished by sharing VMS storage with the VCS program as described below. Large amounts of data can be communicated, with no physical movement of data.

VCS system preparation

A program is specified to execute in VCS much as *named systems* are specified in CP.[27] As with CP *saved systems*, the program that is to execute in VCS is loaded and executed to the point at which future users will encounter it. Then the privileged command SAVEVCS is executed, creating in CP-owned storage a copy of the contents of the VCS associated with the name given as the argument of the SAVEVCS command. SAVEVCS refers to a named-system table entry and a directory entry created previously for the VCS system. The table is similar in function to (but distinct from) that for currently provided CP named systems. The directory entry is new, reflecting the differences between CP saved systems and VCS systems. Specifically, memory size, virtual machine characteristics, and virtual devices are associated with the named VCS system, rather than with the user virtual machine as is the case for CP saved systems.

Thus systems intended to execute in VCS can be defined only by a user with more-than-general privilege. There is no possibility for a general-class user to redefine or otherwise operate on the names or characteristics of svstems that execute in VCS. The security of the VCS program and of the resources under its control are protected by the fact that no operations are available to the general-class user except selection of the VCS system and invocation of it through a restricted, instruction-level interface. The mechanisms

by which the VCS system and its characteristics are defined (SAVEVCS command and directory entry) are not available to the general-class user.

VCS—the user's view

IMPL command

The user invokes a service provided through VCS by executing the CP IMPL command.[26] This command accepts one argument, which is the name of the program that provides the requested service, much as the *IPL named system* facility of VM/370[27] allows the user to load his virtual memory with a previously saved main storage image of a system. Associated with the system named in the IMPL command (through the CP directory) is the size of VCS required, the types of segments in the system (see below), the virtual machine options required by the system, and the devices owned by the system. (IMPL differs from IPL in that the virtual machine resources required are specified by and created for the system named, rather than the user executing the command.) To enhance security, the system loaded into VCS is dispatched before the IMPL command is completed. Thus the system can perform initialization functions which may include checking the user's authorization to use the system.

If IMPL is executed by a virtual machine that already contains a VCS, the existing VCS system is deleted from the virtual machine, and the new system named, if valid, is installed in a new VCS. However, to support applications such as data base systems, which may need to be informed when a user is disconnecting, the VCS system, before being deleted, is given a machine-check interruption (conceptually a power-down interruption) to allow it to perform termination processing before being deleted. If IMPL is executed with no argument, or with an invalid one, in a virtual machine with a VCS, execution is as above except that no new VCS is created. IMPL can also be executed by the VCS system, allowing a system to annihilate itself if it is invoked by an unauthorized user. If IMPL is executed from VCS, no power-down interruption is generated.

segments of VCS

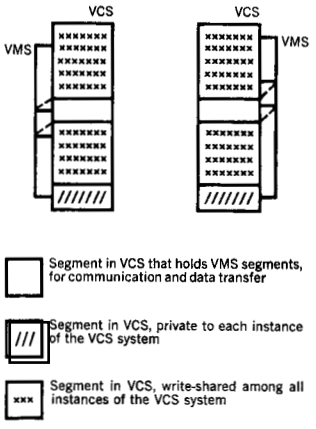
CP currently provides for the sharing of address space segments among virtual machines in read-only fashion, in connection with the named-system facility. If a user changes a shared segment, that segment becomes private to him. All other users maintain the unchanged version. This facility enables main and auxiliary storage to be used more efficiently by avoiding the requirement to keep multiple copies of the same data. Since user virtual machines execute arbitrary programs, CP must protect users of shared segments from changes by any of them.

In VCS, segments shared among users of the same VCS system are writable. This is necessary for the management of shared re-

sources among multiple users, to support shared locks which control simultaneous conflicting requests, and for shared buffers for data base systems. User programs never have addressability to VCS segments, so accidental or intentional change by user programs is not a concern. If protection within VCS of some read-only code segments is desired to protect against errors in VCS code, it can be provided by the VCS program for itself using storage key facilities available in System/370.[28]

Any number of VCS segments can be specified as private to each instance of a VCS program that serves one user. Thus working storage can be statically allocated in the VCS program and declared to be in private segments in the directory description of the program. There is no requirement for explicit storage management in support of multiple users in the VCS program—it is provided automatically by CP virtual storage management.

Figure 3 The types of VCS segments.



In real computing systems, the control storage program can address all of main storage. The implementation of VCS is constrained by the limitation of System/370's addressing capability to 16 777 215 bytes, which implies a limitation on the combined size of VMS and VCS if the latter were to be able to address all of VMS. In designing VCS, therefore, an alternate approach was taken: The directory specification of the VCS system defines some number of segments as communication segments—that is, segments in VCS whose purpose is to hold segments of VMS to allow data communication between VMS and VCS. Parameters of the instruction to be implemented by the VCS program identify segments of VMS to be mapped into VCS. This mapping can be different for different users of the VCS system and for different invocations of the VCS program by the same user. When CP is changing the execution domain of the virtual machine from VMS to VCS, it establishes or changes the communication segment mapping, if necessary. Thus each VCS segment is one of three types:

- Shared with all users of the VCS program, in read/write mode.
- Private in VCS for each instance of the VCS program.
- Defined in VCS to hold segments of VMS for communication of data.

Figure 3 illustrates in a general way the use of the types of VCS segments. The write-sharable segments (perhaps the largest number for a data base application) used for code, data base buffers, and shared data areas, are indicated by the symbol X. The private segments (probably one or a small number) are indicated by the symbol /. VCS segments that hold VMS segments for data communication are indicated by blank spaces.

virtual machine operation with VCS

VCS implements a previously undefined instruction for VMS, one which causes an illegal-operation program interruption in the ab-

Table 1 Assembler-language representation of the new instruction implemented by VCS

dc	x'ffe1'	operation code
dc	h'r'	number of ranges of segments specified
dc	h'k'	number of individual segments
dc	rh	two bytes for each range, the first byte specifying the first segment, the second specifying the last segment
dc	kx	one byte for each individual segment

sence of VCS. An assembler-language representation of the instruction in the current implementation is given in Table 1, which illustrates the format of the description of the segments of VMS to be mapped into VCS. The half-word that follows x'ffe1' must agree with the directory description of the communication segments of the VCS system—that is, the VCS system must expect the same number of ranges of communication segments and the same number of individual communication segments that are specified in the instruction, or else a specification exception is reflected to VMS. (The range in the instruction may specify fewer segments than VCS is prepared to accept, however, to allow for the possibility that VCS has access to all of VMS and that different users may have different virtual machine sizes.) It is expected that normal usage would specify no range or one, or a small number of individual segments. Observe that the instruction is of variable length.

Upon valid execution of the instruction by VMS, the specified segments of VMS are mapped into the corresponding slots in VCS, the program status word and registers of VMS are made available to the VCS program at defined locations of VCS page zero, and the virtual machine configuration is made to correspond to that of VCS. The virtual machine is then dispatched in VCS. When execution is completed in VCS, exit is effected by executing the same instruction, but no segment mapping occurs. Execution returns to the next sequential instruction in VMS unless the VCS program modified the VMS program status word, or some enabled interruption condition for VMS became pending while execution was in VCS, in which case the appropriate new program status word is activated for VMS.

To improve performance, the VMS program can execute the x'fff1' operation code, which is equivalent to x'ffe1' except that

the VMS-to-VCS mapping is not performed, but remains unchanged. In this form of the instruction, the two half-words that follow the two-byte operation code are not examined. The length of the instruction is six bytes. The purpose is to shorten the length of the path between VMS and VCS when there is no need to change the VMS-to-VCS mapping.

Scheduling decisions about this virtual machine are made independently of the activity just described. Central processor utilization, main storage utilization, etc., are accounted for without considering whether the virtual machine is in VMS or VCS. The server and the requestor execute synchronously, and information is transmitted between them through memory locations, register values, and the program status word (condition code).

Asynchronous events, such as interruptions from various components of the virtual system, must be identified as belonging to VMS or VCS and presented to the correct address space. The order in which interruptions are presented is defined by the following rules:

- If the interruption belongs to the address space in which the virtual machine is executing, then normal interruption reflection is performed.
- If the virtual machine is executing in VMS and an interruption for VCS occurs, the virtual machine is switched to VCS and the interruption is presented (if the VCS program is enabled for the interruption). Upon exiting from VCS, the virtual machine is returned to VMS execution.
- If the virtual machine is executing in VCS and an interruption for VMS occurs, the interruption is kept pending for VMS until the VCS program exits, at which time the interruption is presented if the VMS program is enabled for it.

VCS is designed to support sharing of resources by multiple users, particularly when there may be conflicts between simultaneous requests. Shared writable segments are provided to allow the lock management facilities of System/370 to control such accesses. The question arises as to what is to be done when a VCS program executing on behalf of one user requests a lock held by that same program executing for another user. One solution is to establish the convention that associated with each lock is a list of users waiting for the lock. When an instance of the VCS program encounters a held lock, it can wait for the lock by entering its name on the list for that lock, then entering a wait state. When the holder of the lock releases it, an interruption is posted to one of the waiting virtual machines by means of existing mechanisms in CP—the Virtual Machine Communication Facility (VMCF),[19] for example. (To contain errors in VCS programs, as well as for other reasons, an $n+1st$ virtual machine is required—see below.)

Features of VCS

Several interesting and some novel features of system structure follow from the design of VCS. These features are described briefly in the following paragraphs.

synchronous interface

The synchronous interface between VMS and VCS places a minimal burden on the user when invoking the VCS facility, compared with using a distinct, asynchronous virtual machine. The difference is essentially the difference between coding a new instruction and coding to a communication protocol, which requires establishing the connection, allowing for arbitrary severing of the connection, and asynchronous processing on every message.

cooperation between programs

The VCS program has available the System/370 interface, as does the VMS program. In particular, the VCS facility can be based in any operating system that executes in a VM/370 virtual machine. Apart from protection and performance features, the simplicity of coding for a synchronous interface and the storage sharing facilities available in VCS allow VCS to be used as a simple way to establish cooperation between programs based in different operating systems, or to make a facility based in one operating system available to users working in a different operating system.[29]

protection and scheduling

The VCS extension to the virtual machine structure separates two distinct functions which were previously coupled in virtual machine systems. The functions are *change of execution domain* (that is, protection domain) and *change of task* (scheduling unit). In VM/370, substantial effort is exerted to monitor virtual machine usage of real storage, which involves considerable software execution when a change occurs in the scheduling status of a virtual machine. If a virtual machine must wait for service from another virtual machine, the system structure causes a change in scheduling status for the requestor, although the nature of the computation does not. VCS avoids this change in status by making the server part of the scheduling unit of the requestor. Researchers with MULTICS have long recognized that these two functions must be separated, and in fact a hardware extension has been defined to aid the separation.[30] In the IBM IMS/VS environment, the cost of domain crossing has required special steps to improve performance.[31]

CP accounting

In addition to decreasing the length of the path between requestor and server, the VCS structure automatically allows CP to account properly for server execution on behalf of individual requestors. When one server machine is used for multiple requestors, this accounting burden must be borne by the server machine, since it is essentially impossible for CP to do it.

single-user programs

The VCS structure provides one process (instruction counter, register set, etc.) and some private storage for each user of a facility. Therefore the VCS program need not explicitly control multiple users, apart from the logical requirement that simultaneous execution sequences in the VCS code must be serialized if they involve simultaneous conflicting accesses to some shared resource. It is expected that a locking strategy would be used in VCS to manage conflicting accesses, but the structure does not preclude any other strategy by the VCS program.

re-entrant code

The ability to specify any segment of VCS as shared, private, or reserved for communication has implications for re-entrancy of code in VCS which may not be immediately obvious. Writable working storage for each user is provided automatically by CP, so no explicit storage management interface is required in the VCS program. Code segments that usually are (always ought to be) read-only can be placed in shared segments. Thus any existing single-user program can be shared among multiple users in VCS with, at most, repackaging of modules into private and shared categories. By definition, single-user programs contain no multi-user logic. A single-user system being adapted to serve multiple users through VCS must be enhanced to manage simultaneous conflicting requests, if any; the code and data used to accomplish this are placed in a shared segment. Other portions of the program are packaged into shared and private segments as necessary, and the only code changes are the necessary logical enhancements.

execute-only code

Since VCS never is directly addressable by the VMS program, it provides the capability of defining certain areas of main storage as containing program text that cannot be read or written, but only executed. This capability, not provided by System/370 architecture, can be essential, particularly when a high degree of security is required.

***n+1st* virtual machine**

When an application that involves a shared resource manager is being considered for insertion into a VCS structure, it quickly becomes apparent that some components of the manager relate to the application as a whole, not to only a particular user. For example, in a data base management system, there may be a requirement for deadlock detection, or for recovery in the event of failure in support of a particular user, or for periodic housekeeping functions. Since each virtual machine executes the shared VCS program independently of all others (except for shared locks), it is natural for a system administrator function to execute in an *n+1st* virtual machine (if there are *n* users) to carry out necessary system functions asynchronously with regard to the execution for any particular user. The shared storage can be configured to provide asynchronous functions with access to all

required code and data. The *n+1st* virtual machine enables the system administrator to perform whatever security or reliability services are required by the application.

Performance

The motivation for the design and implementation of VCS was to improve the capability of VM/370 to control access to a shared resource in a secure fashion. Design features that contribute to improved security have been discussed above. It is the author's belief, however, that performance is a crucial consideration in all aspects of operating systems research, in the sense that function is meaningful only if it can be delivered with acceptable performance, and that improved performance often reveals enhanced functional capability. Thus this discussion includes much consideration of performance characteristics.

Initially the mechanisms available for communication between virtual machines were modeled on input/output facilities of System/370. Data could be transmitted as virtual unit records, through virtual card punches, printers, and card readers, or through virtual channel-to-channel connections between virtual machines. Subsequently the Virtual Machine Communication Facility (VMCF)[19] became available. Chandra and Katcher[32] compared virtual machine communication via unit records with communication using an experimental precursor of VMCF and found a substantial decrease—more than 50 percent in all cases—in the elapsed time required to communicate with a local virtual disk manager.

The ways in which the VCS structure should provide better performance than a cooperating virtual machine structure have been listed above. Briefly they are: synchronous interface, single-user code in the server, decreasing supervisor cost of data transmission, and avoiding scheduler overhead on calls between user and server.

A very simple data base program was used to test the VCS implementation and provide some measurements of performance. With this program, meaningful counts could be taken of supervisor (CP) instructions in support of the execution domain switch between VMS and VCS, which included the CP cost of data transmission. Quantitative measures of the other items would require measurement of a significant application executing both in VCS and in a multiple virtual machine environment.

The technique used to count supervisor instructions was to employ the CP TRACE INST command,[26] creating a print line for each instruction executed in the virtual machine, which in this case

Table 2 Performance comparison of VCS and VMCF

	Number of instructions executed	*Execution time* (in microseconds)
VCS	874	290.77
VMCF	7320	2622.86

was executing the version of CP modified to provide VCS. One complete communication was traced between a VMS program and the data base program executing in VCS. The total number of CP instructions executed from the time when the request was made to the time when the response was received by the VMS program were counted, and execution times for the instructions executed were calculated, using timing figures for the System/370 Model 168. Equivalent programs were written to execute in two distinct virtual machines using VMCF for intermachine communication, and the same timing runs were performed. The results are given in Table 2.

A detailed study of the instruction trace of the VMCF sequence reveals that the great majority of the instructions executed result from two characteristics of the VMCF facility: The interface between virtual machines is asynchronous, and in general there must be CP scheduler action for the communication to be completed.

Much of the cost of the asynchronous interface is due to supervisor actions such as dynamically allocating working storage for each message (since a user can have more than one message outstanding at a time), dynamically searching the list of active users (one party to the communication may have logged off since the communication was begun), reflecting an interruption to the target of a communication (asynchronous communications require asynchronous events to notify the communicators), and various validity checks on the memory addresses specified by the communicators. No one of these items is surprisingly large; all are required by the nature of the system structure, and all are reflected in the total supervisor cost.

Other sources of supervisor cost are execution of the dispatcher and scheduler, and a resource usage monitor component which is invoked when execution changes between distinct virtual machines.

The design and implementation of VCS were carried out within the context of existing VM/370 scheduling strategy, which contains no

provision for coupling scheduling parameters of two (or more) virtual machines. Bagley et al.,[20] and others informally, have suggested the possibility of changing the scheduling strategy of VM/370 to be more hospitable to a structure that contains multiserver subsystems, in such a way as to allow rational CP scheduling of cooperating virtual machines, but no detailed design or implementation of such strategies is known to the author.

It should be mentioned that the scheduling advantages claimed for the VCS structure accrue only if a requestor in VCS encounters a held lock infrequently. Since CP is not involved in the setting of locks by the VCS program, it schedules virtual machines regardless of locks held or required. If the nature of an application is such that held locks are encountered frequently, that application is not suitable for the VCS structure, which is designed to allow multiple, independent, simultaneous executions of the server program. If an application is such that individual users must hold locks for long periods, that application does not lend itself to a (mostly) single-user code implementation and hence is not suitable for VCS.

While a communication is active, events can occur which lengthen the time required to complete the communication, but are not directly caused by it. For example, during the communication traced, there were two real interval-timer interruptions which may or may not have been caused by the communication. In an operational environment, events of this type often occur during a communication, increasing response time. (See Reference 25 for a more complete treatment of such events.) Although such events can occur in the midst of a VMS-VCS communication, the probability of increasing the path length is less because less time is taken.

The supervisor instructions that support VMS-VCS communication result mainly from two functions that need to be performed. When changing between VMS and VCS, the execution domain of the virtual machine must be redefined. This is accomplished by exchanging a fairly large number of fields in the VMBLOK, the basic control block used by CP to describe the virtual machine. Since these fields are in general not contiguous, a fairly large number of instructions (approximately 25 percent of the total) are required to accomplish the exchange. To minimize the implementation time of VCS, the fields in the VMBLOK were not rearranged to allow fewer instructions to accomplish the exchange, although in a production version this could be done, with a corresponding decrease in the path length.

The other major contributor to the path length is the analysis performed in the dispatcher to determine whether the interruption status of the machine has changed, since the execution domain

will have changed. If certain restrictions on the capability of the VCS program were accepted (in the area of interruptions possible and extended architectural features available), the total path length could be reduced by approximately 33 percent.

Summary

An experimental extension to VM/370 provides the virtual machine with a protected, fast-access execution and data domain. New capability in the form of a new instruction, whose meaning is defined by an arbitrary, protected program, is made available to the virtual machine. Protection is provided by basic, existing mechanisms in VM/370, namely virtual storage and virtual device management. Supervisor overhead required to support communication between a user and a server executing in VCS is minimized because the domain switch does not involve a virtual machine (task) switch, and because the interface provided is synchronous.

The architecture of VCS is based on the concept of microcode which implements instructions for a machine architecture. When this concept is applied as a means of providing service to multiple users in a virtual machine environment, there are many interesting implications for the structure of the code that implements the service. The intention is to provide a system structure in which service can be provided in a way that avoids unnecessary invocation of function or duplication of function in server code and in the underlying control program.

CITED REFERENCES AND NOTE

1. W. S. McPhee, "Operating system integrity in OS/VS2," *IBM Systems Journal* **13,** No. 3, 230–252 (1974).
2. C. R. Attanasio, P. W. Markstein, and R. J. Phillips, "Penetrating an operating system: a study of VM/370 integrity," *IBM Systems Journal* **15,** No. 1, 102–116 (1976).
3. P. A. Karger and R. R. Schell, *MULTICS Security Evaluation—Volume II: Vulnerability Analysis*, Report No. ESD-TR-74-193, Information Systems Technology Applications Office, Deputy for Command and Management Systems, Electronic Systems Division, Hanscomb Air Force Base, Massachusetts 01730; National Technical Information Service (NTIS) No. ADA001120, U.S. Department of Commerce, Springfield, Virginia 22151.
4. L. Flato, "Navy sinks 1108," *Computer Decisions* **8,** No. 7, 35–36 (July 1976).
5. W. M. Inglis, "Security problems in the WWMCCS GCOS system," *Joint Technical Support Activity Operating System Bulletin 730S-12*, Defense Communication Agency (August 1973).
6. T. Alexander, "Waiting for the great computer rip-off," *Fortune*, July 1974, page 143.
7. C. S. Chandersekaran and K. S. Shankar, "On virtual machine integrity," *IBM Systems Journal* **15,** No. 3, 264–278 (1976).
8. J. H. Saltzer, "Protection and the control of information sharing in MULTICS," *Communications of the ACM* **17,** No. 7, 388–402 (July 1974).

9. G. Popek and C. Kline, "Verifiable secure operating system software," *AFIPS Conference Proceedings, National Computer Conference* **43,** 145-151 (1974).
10. A. K. Jones and W. A. Wulf, "Towards the design of secure systems," *Software—Practice and Experience* **5,** No. 4, 321-336 (October-December 1975).
11. T. Linden, *Operating System Structures to Support Security and Reliable Software*, NBS Technical Note 919, U.S. Department of Commerce, National Bureau of Standards, Washington, DC 20234.
12. E. B. Fernandez, R. C. Summers, and T. Lang, *Architectural Support for System Protection and Database Security*, Report No. G320-2683, IBM Los Angeles Scientific Center, 9045 Lincoln Blvd., Los Angeles, California 90045 (1976).
13. R. Bisbey and G. Popek, *Encapsulation: An Approach to Operating System Security*, University of Southern California, Information Systems Institute, 4676 Admiralty Way, Marina Del Rey, California 90291 (1973).
14. J. Donovan and S. Madnick, "Hierarchical approach to computer system integrity," *IBM Systems Journal* **14,** No. 2, 188-202 (1975).
15. *IBM Virtual Machine Facility/370 Introduction*, IBM Systems Library, order number GC20-1800, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, New York 12602.
16. R. P. Parmelee, T. I. Peterson, C. C. Tillman, and D. J. Hatfield, "Virtual storage and virtual machine concepts," *IBM Systems Journal* **11,** No. 2, 99-130 (1972).
17. R. P. Goldberg, *Architectural Principles for Virtual Computer Systems*, Ph.D. Thesis, Harvard University, Cambridge, Massachusetts (1972).
18. R. P. Goldberg, "Architecture of virtual machines," *ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems* (Harvard University, March 1973), 74-112, ACM, 1133 Avenue of the Americas, New York, New York 10036.
19. *IBM Virtual Machine Facility/370 System Programmer's Guide*, IBM Systems Library, order number GC 20-1807, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, New York 12602.
20. J. D. Bagley, E. R. Floto, S. C. Hsieh, and V. Watson, "Sharing data and services in a virtual machine system," *ACM SIGOPS Operating Systems Review* **9,** No. 5, 82-88 (1975).
21. J. N. Gray and V. Watson, *A Shared Segment and Interprocess Communication Facility for VM/370*, Research Report RJ1579, IBM Research Division, 5600 Cottle Road, San Jose, California 95193 (May 1975).
22. C. Young, "Extended architecture and hypervisor performance," *ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems*, (Harvard University, March 1973), 177-183, ACM, 1133 Avenue of the Americas, New York, New York 10036.
23. C. R. Attanasio, *An Additional Protection Ring for Virtual Machine Systems*, Research Report RC5617, IBM T. J. Watson Research Center, Yorktown Heights, New York, 10598 (November 1974).
24. C. R. Attanasio, *An Implementation of a Protected, Fast Access Execution and Data Domain for the VM/370 Virtual Machine*, Research Report RC6327, IBM T. J. Watson Research Center, Yorktown Heights, New York, 10598 (December 1976).
25. R. F. Rosin, "Contemporary concepts of microprogramming and emulation," *Computing Surveys* **1,** No. 4, 197-212 (1969).
26. *IBM Virtual Machine Facility/370: Terminal User's Guide*, IBM Systems Library, order number GC20-1810, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, New York 12602.
27. *IBM Virtual Machine Facility/370: Planning and System Generation Guide*, IBM Systems Library, order number GC20-1801, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, New York 12602.
28. *IBM System/370 Principles of Operation*, IBM Systems Library, order number GA22-7000, IBM Corporation, Department D58, P.O. Box 390, Poughkeepsie, New York 12602.

29. This aspect of VCS usage was suggested by Burt Grad, formerly of the Computing Sciences Department, IBM Thomas J. Watson Research Center.
30. M. D. Schroeder and J. H. Saltzer, "A hardware architecture for implementing protection rings," *Communications of the ACM* **15**, No. 3, 157–170 (March 1972).
31. *IMS/VS Version 1 Fast Path Feature General Information Manual*, IBM Systems Library, order number GH20-9069, IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150.
32. A. N. Chandra and A. M. Katcher, *VM/370 Intermemory Communication—Comparative Measurements*, Research Report RC5820, IBM T. J. Watson Research Center, Yorktown Heights, New York, 10598 (January 1976).

Reprint Order No. G321-5088.