APPLICATION AND ANALYSIS OF THE VIRTUAL MACHINE APPROACH TO INFORMATION SYSTEM SECURITY AND ISOLATION

Stuart E. Madnick(*) and John J. Donovan(**)
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

ABSTRACT

Security is an important factor if the programs of independent and possibly malicious users are to coexist on the same computer system. In this paper we show that a combined virtual machine monitor/operating system (VMM/OS) approach to information system isolation provides substantially better software security than a conventional multiprogramming operating system approach. This added protection is derived from redundant security using independent mechanisms that are inherent in the design of most VMM/OS systems.

I. INTRODUCTION

During the past decade the technique of multiprogramming (i.e., the concurrent execution of several independent programs on the same computer system) has been developed to take full advantage of medium- and large-scale computer systems (e.g., cost economics, flexibility, ease of operation, hardware reliability and redundancy, etc.). Unfortunately, in transferring physically isolated information systems (see Figure 1(a)) to physically shared information systems (see Figure 1(b)), we must cope with the problems of: operating system compatibility, reliability, and security. In this paper we show that the Virtual Machine approach provides effective solutions to these problems.

* Assistant Professor, Project MAC and Sloan School of Management.
** Associate Professor, Project MAC and Department of Electrical Engineering.

Work reported herein was supported in part by Project MAC, an M.I.T. research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01).

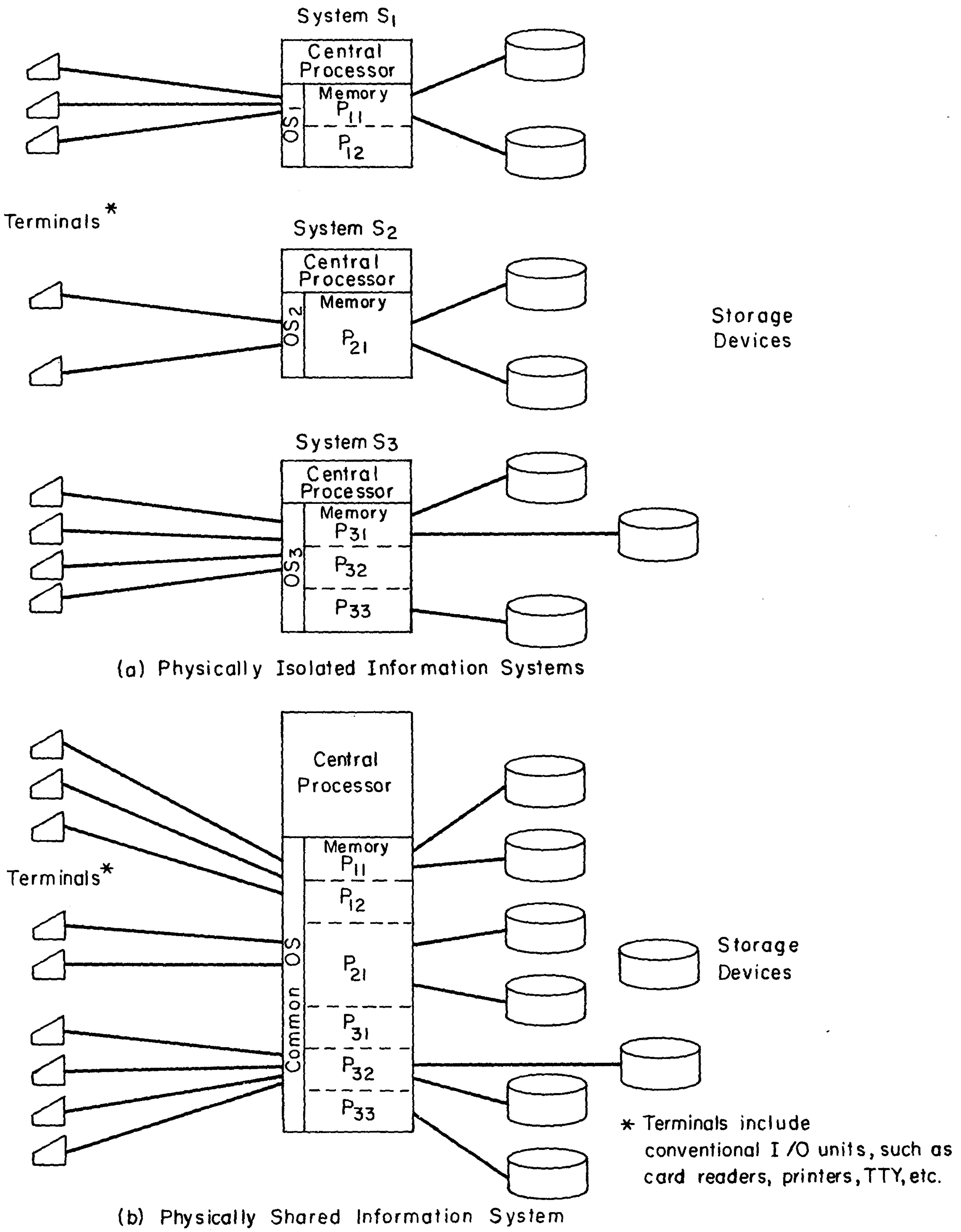
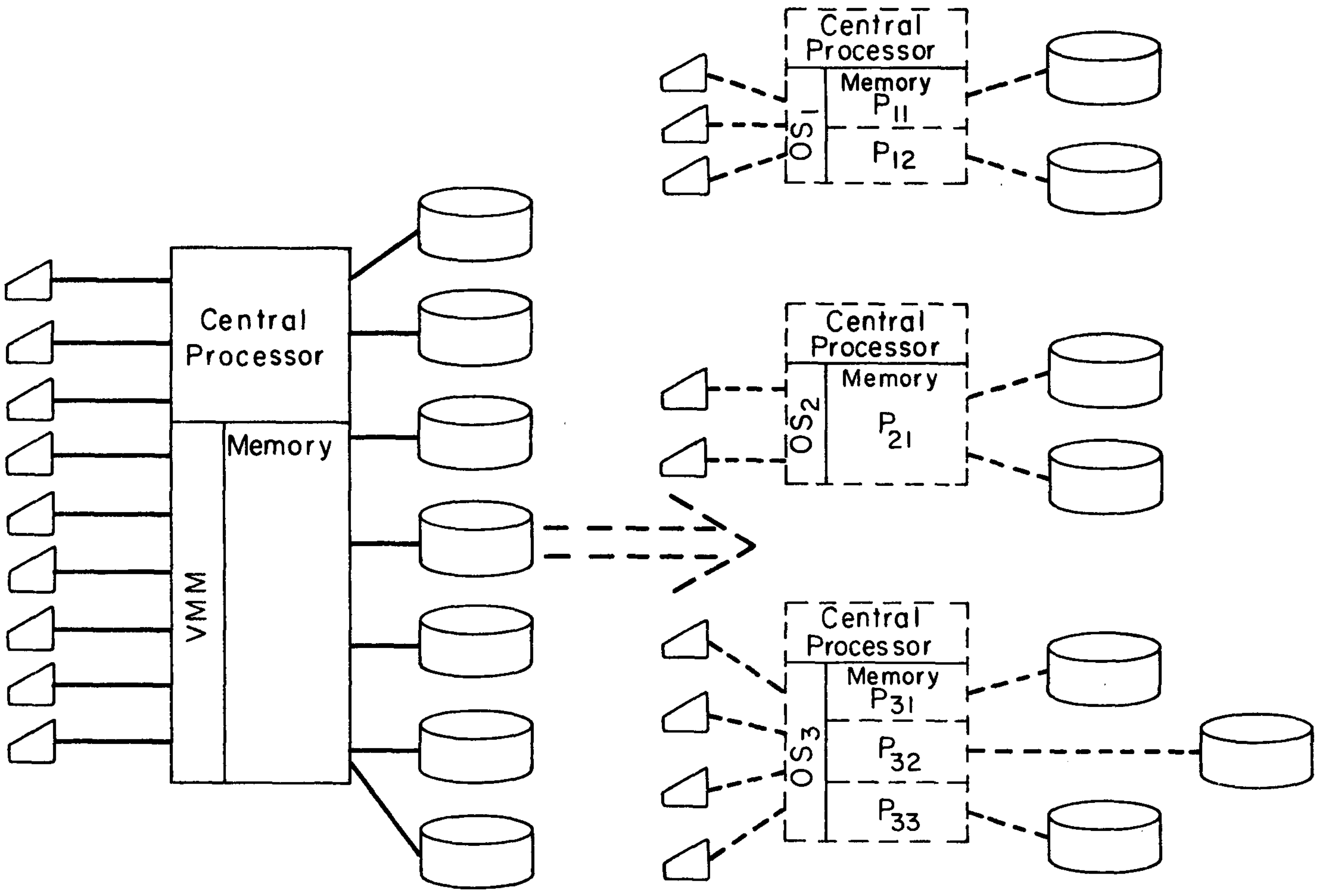


Figure 1. Isolated and Shared Information Systems

II. VIRTUAL MACHINE APPROACH TO ISOLATION AND COMPATIBILITY

Since virtual machines and their applications have been described extensively in the literature (Madnick(5), Parmelee(6)), we will only briefly review the key points. A virtual machine may be defined as a replica of a real computer system simulated by a combination of a Virtual Machine Monitor (VMM) software program and appropriate hardware support. (See Goldberg (3,4) for a more precise definition). For example, the VM/370 system enables a single IBM System/370 to appear functionally as if it were multiple independent System/370's (i.e., multiple "virtual machines"). Thus, a VMM can make one computer system function as if it were multiple physically isolated systems as depicted in Figure 2. A VMM accomplishes this feat by controlling the multiplexing of the physical hardware resources in a manner analogous to the way that the telephone company multiplexes communications enabling separate and, hopefully, isolated conversations over the same wires.

A VMM is totally unlike a conventional operating system. A VMM restricts itself to the task of multiplexing and allocating the physical hardware, it presents an interface that appears identical to a "bare machine". In fact, it is necessary to load a conventional operating system into each virtual machine in order to accomplish useful work. This latter fact provides the basis for the solution to the operating system compatibility problem. Each virtual machine is controlled by a separate, and



(a) Real Information System Hardware

(b) Virtual Information System Hardware

Figure 2. Real and Virtual Information Systems

if necessary different, operating system. The feasibility of this solution has been demonstrated on the VM/370 system and the earlier CP-67 system. The extra VMM software and hardware do introduce additional overhead in the information system operation, but this overhead can be kept rather low (e.g., 10-15%). Depending upon the precise economics and benefits of a large-scale system, the VMM approach is often preferable to the operation of the multiple physically isolated real systems.

III. SECURITY AND RELIABILITY IN A VIRTUAL MACHINE ENVIRONMENT

In the preceeding section it was shown that the virtual machine approach solves the OS compatibility problems by allowing different operating systems to run and coexist on the same computer at the same time. In this section we will analyze security and reliability in a virtual machine environment. We will show that the virtual machine approach results in a system that is much less susceptible to such failures than a conventional multiprogramming operating system. The problems of software reliability and security are quite similar. A reliability failure is any action of a user's program that causes the system to cease correct operation (e.g., "stops" or "crashes"), a security failure is a form of reliability failure that allows one user's program to access or destroy the data or programs of another isolated user or gain control of the entire computer system. The reliability problem has been studied by

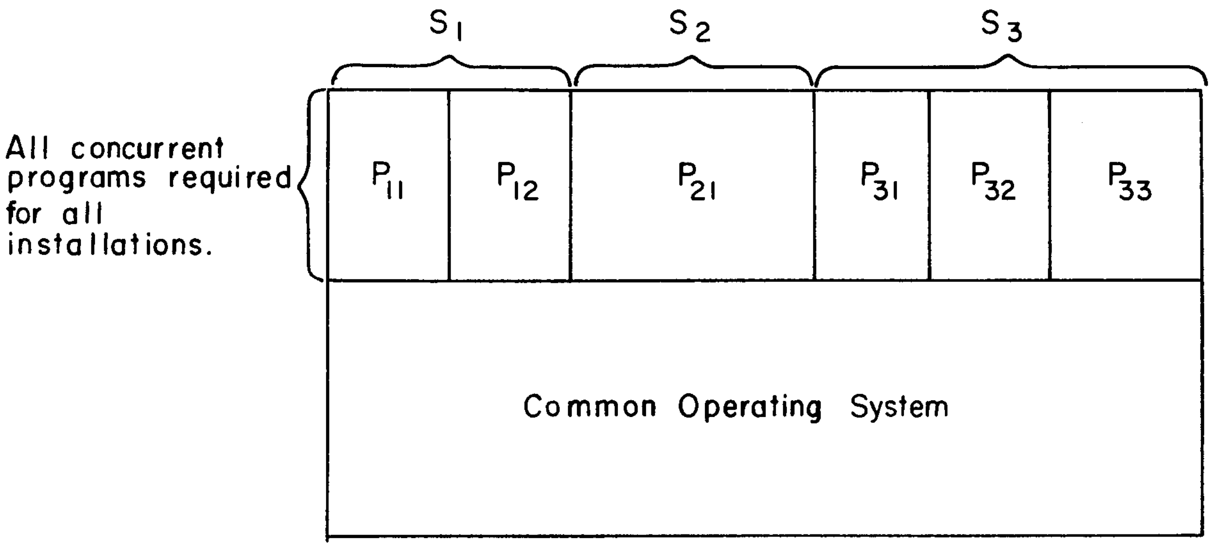
Buzen, Chen, and Goldberg (1).

1. Contemporary Operating System Environment

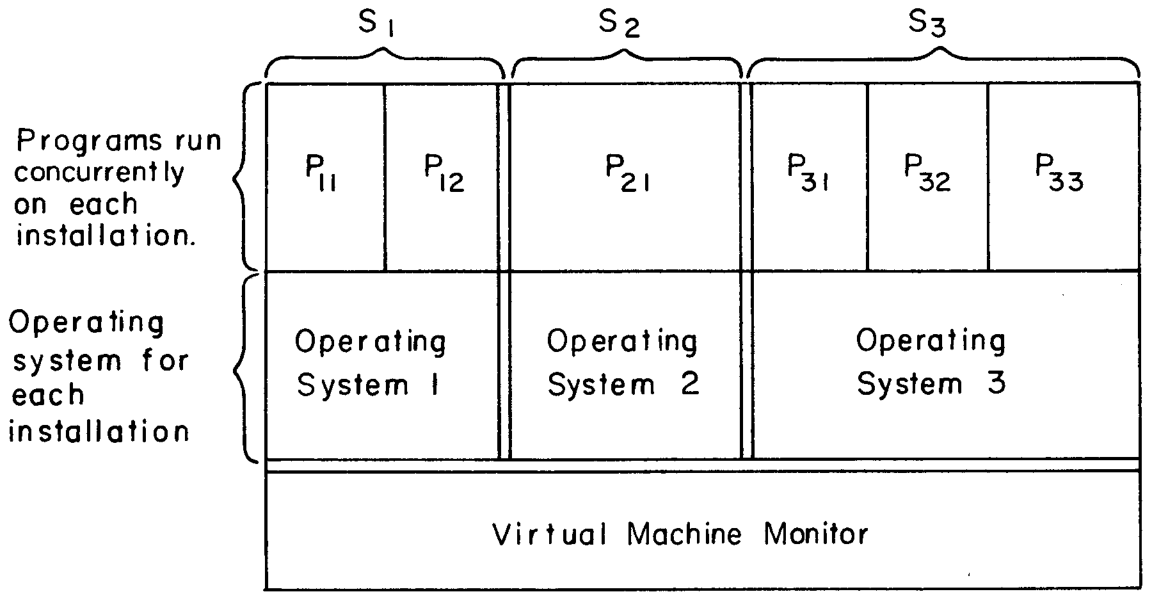
Most contemporary operating systems, in conjunction with appropriate hardware support, provide mechanisms to prevent reliability and security failures (e.g., supervisor/problem state modes of operation, etc.). In this paper we are only concerned about complete isolation security (i.e., no user is allowed access to any other user's information). The problem of generalized controlled access (i.e., a user is allowed restrictive access to another user's information) is much more difficult but, fortunately, such a facility is not needed for the environment illustrated in Figure 1.

Under "ideal" circumstances, most current operating systems can provide isolation security. OS/360, for example, uses the System/360's lock and key protection to insulate users from each other and from the operating system. The supervisor/problem state modes further prevent users from "gaining control" of the system. Thus, it should be possible to isolate users.

Figure 3(a) illustrates the coexistence of multiple programs on the same information system. Such a system is susceptible to a security violation if a single hardware or software failure were to occur. Typical modern operating systems consist of thousands, possibly millions, of instructions. The user programs interface with the operating system through hundreds of



(a) Conventional Operating System Approach



(b) Virtual Machine Approach

Figure 3. Comparison of OS and VMM/OS Approaches

parameterized entries (e.g., supervisor calls, program interrupts, I/O requests and interrupts, etc.). At the present time there is no known way to systematically validate the correct functioning of the operating system for all possible parameters for all entries. In fact, most systems tend to be highly vulnerable to invalid parameters. For example, a popular form of sabotage is to issue certain data-returning supervisor calls (e.g., "what time is it?" request) providing an invalid address as a parameter. The operating system, running with protection disabled and assuming that the address parameter corresponds to a user's data area, transfers the return data to that location. If the address provided actually corresponds to locations within the operating system, the system can be made to destroy or disable itself. Most "secure" systems, of course, attempt to detect this kind of error but there are many other sabotage techniques and complete security is unlikely.

Referring back to Figure 3(a) we can see some of the factors contributing to the problem. In order to provide sufficient functionality to be effective for a large and heterogeneous collection of user programs, the operating system must be quite comprehensive and, thus, more vulnerable to error. In general, a single logical error in the operating system software can invalidate the entire security mechanism. Furthermore, as depicted in Figure 3(a), there is no more protection between the programs of differing user groups or the operating system than there is between the application programs of a single user group.

The security of such conventional operating systems is sufficiently weak that the military has strict regulations that appear to forbid the use of the same information system for both SECRET and TOP SECRET use - even though using separate systems is more costly. Even industrial competitors or different functions in the same company (e.g., payroll and engineering) are often reluctant to share the same computer.

2. Virtual Machine Environment

Figure 3(b) illustrates the virtual machine approach to a physically shared system. This arrangement has numerous security advantages. If we define Ps(P) to be the probability that a given run of program P will cause a security violation to occur, the following conditions would be expected to hold:

A. $Ps(P|OS(n)) < Ps(P|OS(m))$ for $n<m$

OS(i) refers to a conventional operating system multiprogramming at level i (i.e., supporting i concurrent programs). The probability of system failure tends to increase with the load on the operating system (i.e., the number of different requests issued, the variety of functions provided, the frequency of requests, etc.). In particular, a monoprogramming system, OS(1), tends to be much simpler and reliable than a comprehensive multiprogramming system. Furthermore, the m-degree multiprogramming system often requires intricate alterations to support the special needs of the m users, especially if m is large. These problems have been experienced in most large-scale multiprogramming systems. These problems are diminished in a VM environment since each virtual machine may run a separate operating system. Each operating system may be simpler and less error-prone than a single comprehensive all-encompassing operating system.

B. $Ps(OS|VMM(k)) < Ps(P|OS(m))$ for $k<m$

VMM(i) means a virtual machine monitor, VMM, supporting i virtual machines. The operating system, OS, on a particular virtual machine has the same relationship to the VMM(k) as a user's program, P, has to a conventional multiprogramming operating system, OS(m). Using the same rationale as in A above, the smaller the degree of multiprogramming (i.e., $k<m$), the smaller the probability of a security violation. Furthermore, since virtual machine monitors tend to be shorter, simpler, and easier to debug than conventional multiprogramming operating systems, even when $k=m$, the VMM is less error-prone. For example, since the VMM is defined by the hardware specifications of the real machine, the field engineer's hardware diagnostic software can be used to checkout the correctness of the VMM.

We can define the probability of a program P on one virtual machine violating the security of another concurrent program on another virtual machine as:

C. $Ps(P|OS(n)|VMM(k)) = Ps(P|OS(n)) \times Ps(OS|VMM(k))$

Based on the inequalities of A and B above and the multiplicative dependency in C, we arrive at the conclusion:

D. $Ps(P|OS(n)|VMM(k)) << Ps(P|OS(m))$ for $n,k<m$

$Ps(P|OS(n)|VMM(k))$ is the probability of the simultaneous security failure of P's operating system and the virtual machine monitor. If a single operating system's security fails, the VMM isolates this failure from the other virtual machines. If the VMM's security fails, it exposes information of other virtual machines to the operating system of one virtual machine. But, if functioning correctly, P's operating system will not take advantage of the security breach. This assumes that the designers of the individual operating systems are not in collusion with malicious users, this seems to be a reasonable hypothesis; otherwise, using the same collusion, $Ps(P|OS(m))=1$ could be attained by subverting the common operating system.

We are particularly concerned about the overall system security, that is, the probability that a security violation occurs due to any program in the system. This situation can be computed by:

E. $$\begin{aligned} Ps(P11,P12,\ldots,P33) &= Ps(P11)x(1-Ps(P12))x\ldots x(1-Ps(P33)) \\ &+ (1-Ps(P11))xPs(P12)x\ldots x(1-Ps(P33)) \\ &+ \ldots \\ &+ Ps(P11)xPs(P12)x\ldots xPs(P33) \end{aligned}$$

Alternately, it can be represented as:

$$Ps(P11,P12,\ldots,P33) = 1 - (1-Ps(P11))x(1-Ps(P12))x\ldots x(1-Ps(P33))$$

We note that $Ps(P11,P12,\ldots,P33)$ is minimized when the individual Ps's are minimized. The effect is accentuated due to the multiplicative nature of Equation E. Thus, from the inequality of D, we conclude:

F. $Ps(P11,P12,\ldots,P33|OS(n)|VMM(k)) \lll Ps(P11,P12,\ldots,P33|OS(m))$ for $n,k<m$.

That is, the security in a virtual machine environment is _very much better_ than in a conventional multiprogramming operating system environment. This conclusion depends upon the probabilistic independence of the security failures. In the following section we show that the independence condition applies.

3. Redundant Security Mechanisms

If the individual operating systems, OS, and the virtual machine monitor, VMM, used identical security mechanisms and algorithms, then any user action that resulted in penetration of one could also penetrate the other. That is, first take control of the OS and then, using the same technique, take control of the VMM. This is logically analogous to placing one safe inside another safe - but having the same combination on both safes. To combat this danger, the OS and VMM must have _redundant security_ based upon _independent mechanisms_. A similar approach has been taken in the PRIME modular computer system being constructed at the University of California, Berkeley. They use the term _dynamic verification_ to mean "that every time a decision is made there is a consistency check performed on the decision using independent hardware and software" (Fabry(2)).

Table 1 illustrates redundant security mechanisms possible in a VMM/OS environment using VM/370 and OS/360 as example systems. Let us consider main memory security first. OS/360 uses the System/360-370 lock and key hardware to isolate one user's memory area from invalid access by another user's program. VM/370, on the other hand, uses the System/370 Dynamic Address Translation (DAT) hardware to provide a separate virtual memory (i.e., address space) for each virtual machine - independent of the locks and keys. Thus, a malicious user would have to overwhelm both the lock and key and the DAT mechanisms to violate the isolation security of another coexisting program on another virtual machine. The software algorithms, of course, used by OS/360 and VM/370 for memory security are quite different since the mechanisms that are used are so different. Thus, it is highly unlikely that they would both be susceptible to the same penetration techniques.

We find the same kind of redundant security in the area of secondary storage devices. OS/360, especially with the Resource Security System (RSS) option, provides an elaborate set of mechanisms to restrict access to data sets (files). Each storage volume has a recorded label that is read by OS/360 to verify that it is the correct volume to be used (i.e., Automatic Volume Recognition, AVR). Furthermore, under RSS, the specific data sets on the volume may be individually protected by means of password codes or user authorization restrictions. VM/370, on the other hand, may have the volumes assigned to the virtual machines by the computer operator or a directory on the basis of

FUNCTION	VMM Mechanism (e.g., VM/370)	OS Mechanism (e.g., OS/360)
Main Memory Security	Dynamic Address Translation (DAT)	Locks and Keys
Storage Device Security	Device Address Mapping	Volume Label Verification and Data Set Passwords
Process Allocation Security	Clock Comparator and Time-Slicing	Priority Interrupt (and, optionally, Interval Timer)

Table 1.
Examples of Redundant Security Mechanisms in a VMM/OS Environment

the physical storage device address being used. Once again, the logical mapping of OS/360 is independent of the physical mapping of VM/370. These redundant security mechanisms can be found in many other areas.

Although most existing VMM's were not designed specifically to provide such comprehensive isolation, they frequently include substantial redundant security mechanisms. In order to provide the needed isolation, future VMM's may be designed with increased redundant security.

IV. CONCLUSIONS

In this paper we have shown that the VMM/OS approach to information system isolation provides substantially better software reliability and security than a conventional multiprogramming OS approach. This added protection is obtained through the use of redundant security mechanisms that are inherent in the design of most VMM/OS systems.

REFERENCES

1. Buzen, J. P., Peter P. Chen, and Robert P. Goldberg, "Virtual Machine Techniques for Improving System Reliability", Proceedings of the ACM Workshop on Virtual Computer Systems, (March 26-27, 1973).

2. Fabry, R. S., "Dynamic Verification of Operating System Decisions", submitted for publication in the Communications of the ACM, (February 23, 1972).

3. Goldberg, R. P., "Virtual Machines: Semantics and Examples", Proceedings of IEEE Computer Society Conference, (September 1971), 141-142.

4. Goldberg, R. S., Architectural Principles for Virtual Computer Systems, PhD dissertation, Harvard University, (November 1972).

5. Madnick, S. E., "Time-Sharing Systems: Virtual Machine Concept vs. Conventional Approach", Modern Data 2, 3 (March 1969), 34-36.

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