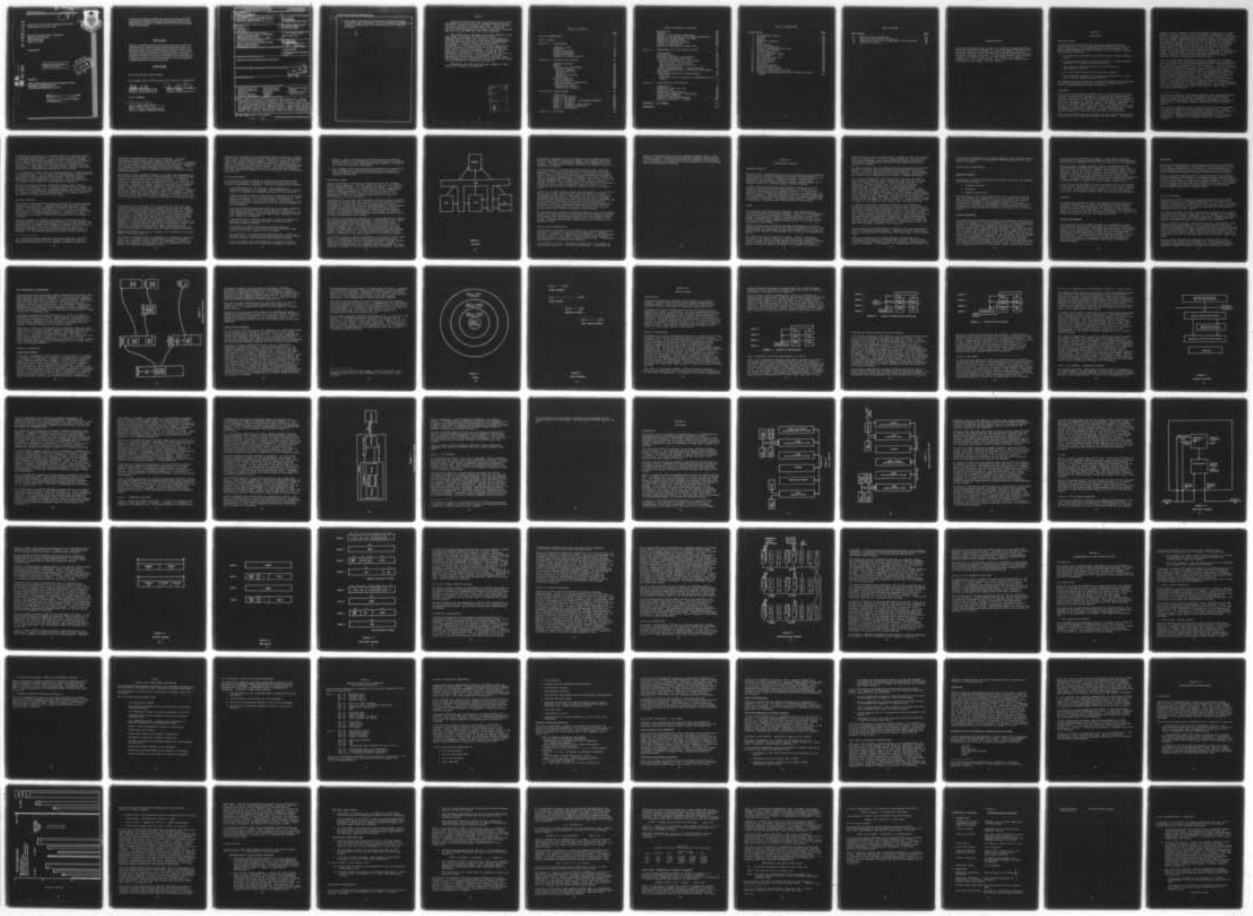
AD-A055 164 HONEYWELL INFORMATION SYSTEMS INC MCLEAN VA FEDERAL --ETC F/G 9/2
ANALYSIS OF SECURE COMMUNICATIONS PROCESSOR ARCHITECTURE. VOLUM--ETC(U)
NOV 75 J GILSON, J MEKOTA F19628-74-C-0205
UNCLASSIFIED ESD-TR-76-351-VOL-1 NL

1 OF 2
AD
A055164



FOR FURTHER TRANSLATION

ESD-TR-76-351, Vol. 1

AD A055164

ANALYSIS OF SECURE COMMUNICATIONS PROCESSOR ARCHITECTURE

Honeywell Information Systems, Incorporated
Federal Systems Operations
7900 Westpark Drive
McLean, VA 22101

November 1975

Approved for Public Release;
Distribution Unlimited.

DDC
RECEIVED
JUN 16 1978
F

AD No.
DDC FILE COPY

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
HANSCOM AIR FORCE BASE, MA 01731

THIS DOCUMENT IS BEST QUALITY PRACTICABLE.
THE COPY FURNISHED TO DDC CONTAINED A
SIGNIFICANT NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

LEGAL NOTICE

When U.S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

This technical report has been reviewed and is approved for publication.

William R. Price

WILLIAM R. PRICE, Captain, USAF
Techniques Engineering Division

Donald P. Eriksen

DONALD P. ERIKSEN
Techniques Engineering Division

FOR THE COMMANDER

Frank J. Emma

FRANK J. EMMA, Colonel, USAF
Director, Computer Systems Engineering
Deputy for Command & Management Systems

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ESD-TR-76-351, Vol I	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): ANALYSIS OF SECURE COMMUNICATIONS PROCESSOR ARCHITECTURE, Volume I	5. TYPE OF REPORT & PERIOD COVERED: Final Report.
6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s): John Gilson, John Mekota	8. CONTRACT OR GRANT NUMBER(s): F19628-74-C-0205
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Honeywell Information Systems, Incorporated, Federal Systems Operations, 7900 Westpark Drive, McLean, VA 22101	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: CDRL ITEM A005
11. CONTROLLING OFFICE NAME AND ADDRESS: Deputy for Command and Management Systems, Electronic Systems Division, Hanscom AFB, MA 01731	12. REPORT DATE: November 1975
	13. NUMBER OF PAGES: 107 (121 pp.)
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

DDC RECEIVED JUN 16 1978 F

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Reference Monitor
Complete Mediation
Isolation
Address Mapping
Virtual Environment
Ring Structure
Security Kernel
Processes
Segments
Paging
Security Protection Module
Multilevel Security Certification

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report studies the architecture requirements for a Secure Communications Processor intended to serve both as a front end and a remote communications processor for the secure Multics system being developed under Project Guardian. The security issues are addressed through the reference monitor concept based on the mathematical model of security developed at MITRE. The reference monitor must provide completeness, isolation and certifiability.

(over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

409 699

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(cont)

This report addresses the architectural features needed to support these properties and also addresses implementation issues including speed and reliability of the resultant processor.

Preface

Because of funding and other limitations, the Air Force terminated the effort which this document describes before the effort reached its logical conclusion. This report is not satisfactory but was published in the interest of capturing and disseminating the computer security technology that was available when the effort was terminated.

This report was to document the analysis phase of a one year study to determine the architecture for a communications processor that could 1) provide effective internal security controls by supporting the implementation of a software security kernel 2) perform communications processing in a variety of DoD applications and environments, 3) have performance efficiency comparable to similar non-secure processors and 4) be cost competitive with similar non-secure processors. During the analysis phase various alternative architecture features were considered. The above four criteria guided the selection of the various alternatives. Unfortunately, the report concentrates on describing the particular features rather than identifying the alternatives and the analysis that lead to the selection of a particular alternative.

Additional, more specific technical comments on this report are attached as an appendix.

ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist.
A 23 C.F.

TABLE OF CONTENTS

		Page
LIST OF ILLUSTRATIONS		5
LIST OF TABLES		6
SECTION I	INTRODUCTION	8
	PURPOSE OF STUDY	8
	BACKGROUND	8
	TECHNICAL APPROACH	12
	DESIGN REQUIREMENTS	14
	RECOMMENDED DESIGN	15
	SUMMARY AND RECOMMENDATIONS	17
SECTION II	ARCHITECTURAL OVERVIEW	19
	REFERENCE MONITOR	19
	MODELS	19
	ARCHITECTURAL ALTERNATIVES	21
	Reference Monitor	21
	Complete Mediation	21
	Isolation	22
	Provable Correctness	22
	Efficiency	23
	MAPPING ADDRESSES	23
	Map Manipulations	25
	Map Realization - Descriptors	26
	PROTECTION MECHANISMS	26
	Generalized Domains	26
	Multics Ring Structure	28
SECTION III	DESIGN CONCEPT	32
	INTRODUCTION	32
	LEVELS OF ABSTRACTION	32
	LEVELS OF THE KERNEL	35
	LEVELS 4 AND GREATER - UNCERTIFIED SOFTWARE	36
	LEVEL 3 - SECURITY	38
	LEVEL 2 - SEGMENTED VIRTUAL MEMORY	39
	LEVEL 1 - SEQUENTIAL PROCESSES	41
	LEVEL 0 - THE HARDWARE	44
SECTION IV	SPM DESIGN	46

TABLE OF CONTENTS (concluded)

	INTRODUCTION	46
	THE SPM	50
	SUPPORT OF THE PROCESS ABSTRACTION	50
	SUPPORT OF THE AUGMENTED MAIN MEMORY ABSTRACTION	53
	SUPPORT FOR ACCESS CONTROL AND RINGS	58
	PERFORMANCE CONSIDERATIONS	58
	SUPPORT OF MULTIPLE PROCESSORS	59
	SUPPORT OF INPUT/OUTPUT	60
	SUPPORT FOR THE INITIAL SECURE STATE	63
SECTION V	REQUIREMENTS FOR THE PROTECTED SYSTEM	64
	INTRODUCTION	64
	SPM INSTRUCTIONS	64
	PROCESSOR REALIZATION REQUIREMENTS	72
	Required Certified Operations	73
	REALIZATION REQUIREMENTS - MAIN MEMORY	74
	Error Detection and Correction	74
	Read-Alter-Write Memory Operation	74
	Module Identification	75
	Rapid Standardization of Memory Contents	75
	REALIZATION REQUIREMENTS: PERIPHERAL CONTROLLERS AND DEVICES	75
	CACHE MEMORY REALIZATION CONSIDERATIONS	76
	Cache Size	77
	Descriptor Bits Not Mandatorily Represented in the Cache	77
SECTION VI	IMPLEMENTATION CONSIDERATIONS	79
	INTRODUCTION	79
	ESTIMATES OF SIZE AND COST	80
	TRADE-OFF NOTES	84
	RELIABILITY CONSIDERATIONS	85
	DESIGN RECOMMENDATIONS - RELIABILITY	93
	PERFORMANCE ESTIMATES	94
	SUMMARY - EXPECTED PERFORMANCE	101
	PERFORMANCE ANALYSIS - SUMMARY	101
APPENDIX A	AF COMMENTS	102
REFERENCES		104

LIST OF ILLUSTRATIONS

Figure Number		Page
1	The Reference Monitor	11
2	The SPM	16
3	Mapping	24
4	Descriptor Structure	27
5	Rings	30
6	Ring Brackets	31
7	Levels of Abstraction	33
8	Levels of Abstraction and Rings	34
9	Outward Ring Crossing	35
10	Kernel Structure	37
11	Process Execution States	43
12	Host Computer	47
13	Host Computer With SPM	48
14	SPM Block Diagram	51
15	Virtual Address	55
16	BDR Layout	56
17	Descriptor Formats	57
18	Multiprocessor System	61
19	Size vs. Address Width	82
20	Estimated Performance Reduction	98
21	Effective Memory Speed Loss Due To Descriptor Cache Delay	100

LIST OF TABLES

Table Number		Page
I	Fault and Trap Information	69
II	Security Related Fault Information	71
III	Reliability Function for Independent Fault Detection	88
IV	Analyses Required	91

Acknowledgement

This document is an edited version of a technical report produced by Honeywell Information Systems, Inc. under contract F19628-74-C-0205. While the majority of the work was done by Honeywell, extensive revisions have been made to reflect the editorial comments of project members at ESD/MCIT and MITRE. The editorial comments were provided by Maj. R. R. Schell, Capt. W. R. Price, Capt. P. A. Karger of ESD/MCIT, and K. J. Biba of MITRE. The authors for Honeywell were John Mekota, John Gilson, and many others.

SECTION I

INTRODUCTION

PURPOSE OF STUDY

The objectives of this study were to detail the architectural requirements for a Secure Communications Processor and to study the design trade-offs in the specification of such a processor. The Secure Communications Processor is needed for many diverse military applications, including:

* As a front end processor for a secure general purpose computer system
* As an interface message processor connecting a general purpose computer to a computer network environment
* As a node in a store-and-forward or packet-switched communications network
* As a stand alone communications terminal processor
* As a switching processor for dynamic reconfiguration of large secure general purpose computer systems.

The architecture was derived from fundamental security design concepts that are demonstrably sufficient to ensure the effectiveness of internal hardware/software security controls.

BACKGROUND

The military and civilian branches of the U. S. Government are heavy users of data processing equipment. This generates much economic pressure to obtain the most efficient computing arrangements possible, including sharing of facilities by the maximum number of government users and applications. Moreover, many of the uses of computers, such as in data networks, command and control, and message switching systems, inherently require the facilities to be shared by many users in order to perform their functions. Other applications such as data concentrators, satellite computers, etc. have communications characteristics in the routing and transmittal of information.

Hence the Government has much interest in multi-user computer systems for maximizing the value obtained from each tax dollar. (Reference 1)

However, much of this work involves processing information of widely different security levels (1) and categories, (2) generated by many different projects in many different agencies and branches, which must be kept carefully segregated lest highly sensitive information reach the wrong hands. Current multiuser computer systems are fundamentally inadequate to provide protection for the information they process. While current systems do an excellent job of keeping non-malicious users from interfering with one another, the consistent success of "Tiger Teams" in penetrating these systems shows that they are vulnerable to the attack of knowledgeable, patient, and malicious users. See References 34 and 35. The least vulnerable is the Multics system, currently under security development sponsored by the USAF. However, Multics is several orders of magnitude too large and expensive for the applications contemplated here.

There is little risk (except for denial of service) in a processor that manipulates only the address or header information of messages in the clear, with the body of the message encrypted. The security of the information is safeguarded by the encryption. A hostile agent who accessed the encrypted data would have to break the code to get access to the information. The encrypted version would be more easily accessible to a hostile party through tapped communication circuits rather than computer penetration. The risk becomes large when the entire message is handled in the clear and prohibitively large when the processor is used for other tasks, concurrent with message handling. The increasing capabilities of small processors, brought about by technological advances in speed and memory capability, produce pressures to use a single processor both as a message handler and also as a processor of the messages handled. Such usage increases the risk of compromise and leads to a need for a small Secure Communications Processor.

---

(1) security levels in this context refers to the DoD classification system for controlling access to defense sensitive information. Security levels are applied to instances of information and DoD clearances are granted by level for basic permission to access such information. These levels include Confidential, Secret and Top Secret.

(2) security categories in this context refers to further refinement of control of access to defense sensitive information. Access requires the person to hold a sufficient security clearance level and also have the proper formal category designation to use the information. Categories are normally associated with the type of information. Examples of categories include NATO, Nuclear, and Crypto.

Use of a communications processor as a front-end for a large multiuser system processing data at several security levels presents an inviting target for a malicious user. The risks include both the risk of a user acquiring or modifying information to which he has no access authorization and the possibility of a malicious user denying use of the system at a critical time. In military use, the value of the information acquired or modified or the gain achieved by denying use of the system may warrant virtually any level of effort to achieve the penetration.

The Air Force has shown in earlier work (Reference 7) that a secure computer system can be constructed using the concept of the reference monitor. A reference monitor is a hardware-software mechanism that controls the access of subjects (active system elements) to objects (units of information) within the computer system. Figure 1 presents a schematic diagram of the relation among subjects, objects, reference monitor, and reference monitor authorization data base (access matrix). See Reference 22. The figure gives examples of typical subjects, objects, and data base items.

In operation, the reference monitor allows or forbids access by subjects to objects, making its decisions on the basis of subject identity, object identity, and security parameters of the subject and object. The reference monitor both mechanizes the access rules of the military security system and assures that they are enforced within the computer. An implementation of the reference monitor is termed a "security kernel".

A reference monitor must meet the following three requirements in order to provide the basis for a multilevel secure computer system:

a. Completeness - the reference monitor must be invoked on every access by a subject to an object;

b. Isolation - the reference monitor and its data base must be protected from unauthorized alterations;

c. Certifiability - the reference monitor must be small, simple and understandable so that it can be tested and verified to perform its functions properly.

Both the requirement for completeness and that for certifiability demand that the reference monitor include hardware as well as software - the former because software validation of every access by a subject to an object would add intolerable complexity and overhead to the reference monitor, the latter because certain hardware architectures preclude the construction of a simple understandable operating system. The purpose of this study is to define a suitable architecture.

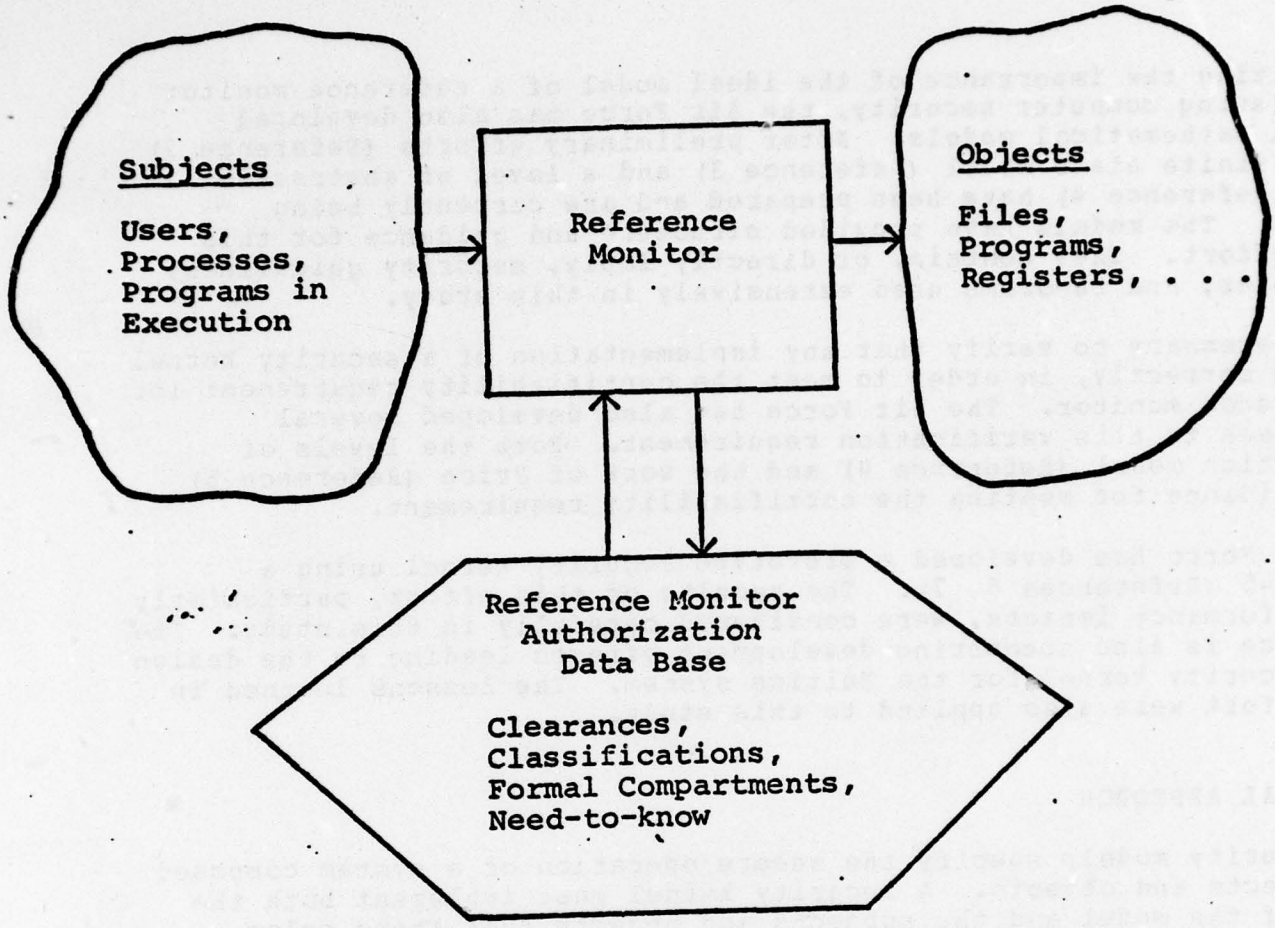


Figure 1

The Reference Monitor

Recognizing the importance of the ideal model of a reference monitor to achieving computer security, the Air Force has also developed several mathematical models. After preliminary efforts (Reference 2) both a finite state model (Reference 3) and a level of abstraction model (Reference 4) have been prepared and are currently being refined. The models have provided structure and guidance for this study effort. They contain, or directly imply, security guidelines, principles, and theorems used extensively in this study.

It is necessary to verify that any implementation of a security kernel is done correctly, in order to meet the certifiability requirement for a reference monitor. The Air Force has also developed several approaches to this verification requirement. Both the levels of abstraction model (Reference 4) and the work of Price (Reference 5) give guidance for meeting the certifiability requirement.

The Air Force has developed a prototype security kernel using a PDP-11/45 (References 6, 7). The results of this effort, particularly the performance impacts, were considered carefully in this study. The Air Force is also sponsoring development efforts leading to the design of a security kernel for the Multics system. The lessons learned in that effort were also applied to this study.

TECHNICAL APPROACH

The security models specify the secure operation of a system composed of subjects and objects. A security kernel must implement both the rules of the model and the subjects and objects that these rules control. The usual interpretation of subject is that of a process (execution of a program, see Reference 23.) The usual interpretation of an object is a virtual memory segment. Provision of a true reference monitor requires complete implementation of all aspects of the model.

The implementation of subjects and objects is constrained by the hardware on which the security kernel operates. If the hardware does not facilitate the simple implementation of subjects and objects, the certifiability requirement for a reference monitor will not be met. A minimum requirement for secure computer systems is the use of descriptor driven processor architectures that implement segmented memories. (3) With such processors, the objects of the model can

---

(3) A descriptor driven processor is one whose hardware interprets each "virtual" address issued by a program in terms of a set of descriptors that specify the real physical address and permitted

correspond to segments supported by the hardware. A properly organized segmented memory merges primary storage (core or semiconductor RAM) and secondary storage (bulk store, disk, peripheral devices) management functions, eliminating from security consideration any separate, complex, and security related "file system". Further, the subjects of the model correspond to processes (address space-processor state pairs) supported directly by descriptor driven processor hardware.

The security kernel software defined by the model and implemented on descriptor driven hardware is a simple mechanism that implements only the security rules, the subjects, and the objects. It does not provide the full facilities of an operating system; it could not do so without developing so much complexity that it would no longer be a prototype. Instead, the complex functions required of an operating system are provided by programs external to and controlled by the kernel. These functions can be arbitrarily complex, but are not security related. However, some may be sensitive in terms of assuring the smooth operation of the computer system. For example, a typical operating system (not kernel) function such as a scheduler which has no access to user information cannot compromise security, but it can slow service to users. (4) Denial of service is not a security issue.

To assure that user programs can be separated from (and kept from interfering with) such sensitive programs, the previous development efforts in multilevel security have identified the need for hardware with at least three separate domains of execution (states of program privilege). Of these, one can be allocated to the security kernel, the second to the operating system, and the third to user programs. The security kernel can then protect the operating system from user programs and, with suitable architectural organization for the hardware, the transitions from one domain to another can be rapid and efficient. Thus the technical approach for this study is based on the concept of a security kerrel, using a model of the kernel that represents the secure operation of an ideal reference monitor. The ideal reference monitor requires that the kernel implement subjects,

---

access modes (e.g., read, write, execute) to be associated with every possible "virtual" address.

(4) Malicious tampering with the scheduling of a process or operation cannot give a process access to forbidden data, only the security kernel can change data access rights. The scheduling algorithm can prevent an authorized process from access to a processor and hence prevent that process from operating at all.

objects, and the security rules governing their interactions. Simple (certifiable) implementation hinges on development of an architectural design that supports these functions. Prior work (Reference 24) has shown that a secure computer system must supply a segmented virtual memory, controlled I/O facilities, at least two processor domains, and requires use of descriptor driven hardware. The above considerations and extensive experience and study of the prior work were combined with good engineering judgement to make the numerous design trade-offs required here.

DESIGN REQUIREMENTS

The work reported here attempted to cover all the necessary studies and analysis necessary to define an architecture for a secure and effective communications processor. Specific considerations included:

* The implementaion of the reference monitor concept or a functional equivalent is required. The development of a security kernel to implement the reference monitor is one alternative.

* Isolation of the security kernel from the remainder of the system must be provided by use of at least three hierarchically ordered protection domains. Both ring mechanisms, as proposed by Schroeder (Reference 8) and used in Multics, and capability based domain mechanisms as suggested by Price and others (Reference 5) were considered.

* A virtual addressing mechanism using descriptor based addressing must provide the potential for relatively large number of independent segments. A true two dimensional addressing facility with independent access controls (read, write, execute, etc) for each segment is recommended.

* A mechanism must be provided for rapidly switching between the descriptor sets for various processes, in order to allow efficient context switching.

* The design must ensure that those mechanisms actually implementing security controls shall be well defined and well isolated from other elements of the processor.

* The isolation of the implementing mechanisms must include any component which, if subject to malicious tampering and/or random hardware failure, could invalidate the security controls.

* Input and output (I/O) operations must be subject to direct security controls since I/O constitutes exchange of data with

- objects. Both the techniques of virtualized I/O, proposed by Clark (Reference 9) and implementation of individual I/O devices as parts of the virtual memory of processes operating outside of the security kernel were considered.

* The resulting Secure Communications Processor must be capable of being implemented to meet the demands of various military specifications including adverse environmental conditions and airborne applications.

RECOMMENDED DESIGN

Various approaches to providing a hardware base for the reference monitor were considered. The most attractive one, from the viewpoint of the systems architect, is a completely new design. This allows maximum freedom to provide an optimum solution of the requirements, and gives minimum constraints. Unfortunately, it also tends to maximize development costs, lead time, and support costs. The alternative of modifying an existing minicomputer (or family of minicomputers) was also considered.

Consideration of the total cost and effort needed to provide a working communications system lent great weight to the modification approach. The effort to produce system and user software and engineering documentation for testing, training, and maintenance, so far outweighs the effort to produce hardware that the trade-offs were heavily weighted towards an approach that would allow use of most existing software and engineering documentation. In an effort to provide a minimum cost solution, several different base systems including those of other manufacturers were studied.

The result of the study was selection of a unique architectural design for the Secure Communications Processor. Consideration of the design requirements listed above and examination of both current and anticipated computer system designs lead to the use of a Security Protection Module (SPM). The Security Protection Module is a new piece of hardware that will be logically and physically interposed between the components of a standard, commercially available computer system. Figure 2 shows the use of the Security Protection Module. Note the similarity to the ideal reference monitor shown in Figure 1.

The SPM will provide a direct realization of the security kernel model since its location between the active units (central processor, memory, I/O controller, etc) of the protected computer allows meeting the first two requirements for a reference monitor. Complete mediation is ensured since all references must pass through the SPM. Isolation is ensured by use of a separate item of hardware. The final

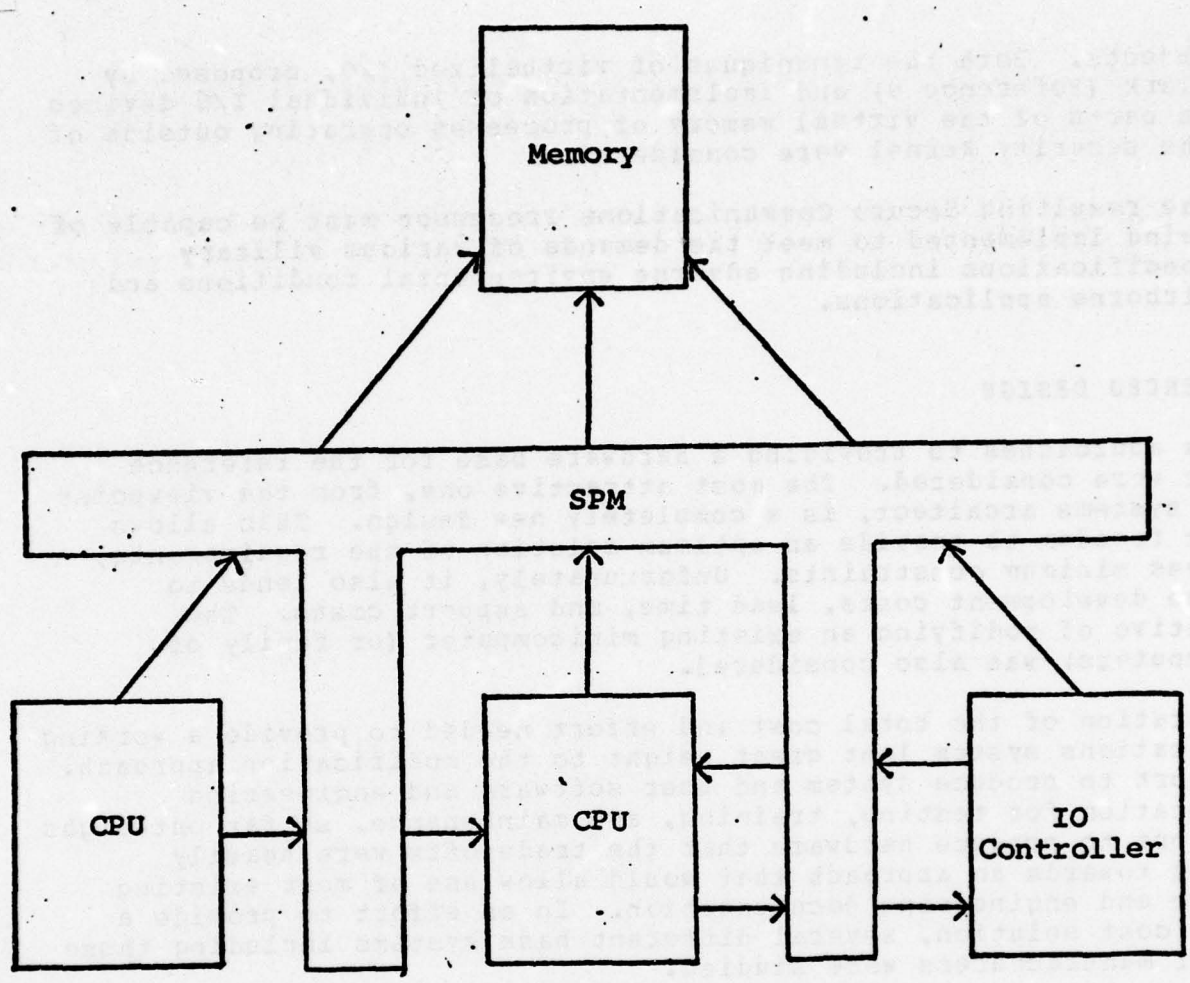


Figure 2

The SPM

requirement, simplicity, may be attained since the SPM has only to consider the security rules of the model. Only aspects requiring physical isolation or performance assistance are supported in the hardware. The protected computer provides the operating system and application programs.

An SPM connected in this way could potentially be used with many different computers at different levels of complexity, cost, and performance. The requirements placed on the computer are detailed later. Use of an SPM could allow an otherwise standard computer to serve as a Secure Communications Processor. There would be need for special documentation, training, and support only for the SPM and its related software, with the remainder of the computer receiving standard support. This is particularly attractive for the development efforts as only the SPM, the security kernel software, and the interfaces need be proved correct in accordance with the security model.

Use of an SPM could severely degrade the performance of the system by adding time to each action of the protected central processor. A considerable amount of the design effort was devoted to avoiding or minimizing such degradation. Estimates indicate that performance degradation of less than twenty five percent should be realizable even if the protected processor already takes maximum advantage of available technology. Some combinations of protected processor and SPM design could have performance degradations of under half this amount, and it is possible to design an SPM which would actually improve performance of many unprotected minicomputers.

The isolation of the Security Protection Module and the close relation between the ideal reference monitor model and the Security Protection Module should make the task of verification or certification easier. The design of the SPM has been formulated with the guidance of the models. Continuing this pattern should result in an SPM that can be certified secure.

SUMMARY AND RECOMMENDATIONS

The problem of specifying an architecture for a Secure Communications Processor was studied in light of the previous work on computer security. A particular architecture was selected and the requirements applied against it. The result is the SPM centered architecture introduced above and described in greater detail later. A functional specification for the SPM was also prepared.

Much remains to be done. The general concept of the SPM should be refined and expanded to a detailed specification. A prototype SPM

should be fabricated for use with a specific computer system. This prototype should be tested and evaluated to determine the utility of the SPM approach. The verification and certification efforts should also continue as the project moves towards production of a certifiable Secure Communications Processor.

SECTION II

ARCHITECTURAL OVERVIEW

REFERENCE MONITOR

The concept of a reference monitor is well established in the computer security field for both theoretical and practical applications. A realization of a reference monitor is called a security kernel and is expected to be specific to a certain computer. The Security Protection Module proposed here could provide hardware support for a security kernel for a general class of computer systems.

An SPM, combined with the software required to drive it, and the connections to the protected computer system, can function as a security kernel and thus as a reference monitor. It is currently estimated that 20-30% of the capability will be in hardware with the rest in software. Functions have been placed in the SPM hardware when a practical system requires hardware support for adequate performance and efficient operation. The remaining functionality, primarily that of control of the hardware has been assigned to the software.

MODELS

The government has devoted much attention to the development of mathematical models of computer security. This effort is essential to the development of a certifiable reference monitor. To be certified secure, the implemented computer and supporting software must be proved to conform to the model of security certified correct by DoD. The mathematical models produced (References 1, 2, 3, 4, etc) were considered in this study.

Modeling efforts have identified numerous requirements for building secure systems and have codified the security preservation rules. The models define the fundamental notions with precision and give rules of operation for keeping a system secure once it has been initialized as secure.

The models deal with subjects and objects. Figure 1 shows that subjects are the active entities in the system, such as processes acting for users and processes executing for system purposes. Objects are the passive entities in the system, such as data and program segments in memory, machine registers, and peripheral devices. The

models define the types of access that a subject may have to an object under all conditions. The access allowed depends on the contents of the reference monitor authorization data base which identifies the clearances, classifications, and need-to-know attributes of both subjects and objects.

The model requires that the reference monitor authorization data base contain the current access relationships between all subjects and objects. The data base must identify three elements: the identifier of the subject, the identifier of the object, and the allowed access mode (read, write, execute) for the subject to access the object.

The reference monitor data base must contain the security level of each subject and object. This is used along with the mode to grant access of subjects to objects. A security level consists of two components, a classification and a set of access categories. Classifications or clearances form a hierarchy with higher classifications having access to all lower classifications. The simple security condition requires that a subject have a clearance level equal to or greater than that of an object before read access to that object can be granted. Categories are not hierarchically ordered but are ordered by set inclusion. The simple security condition allows subject to have read access objects only if the subject has all categories of the object (it may have more). A subject may not access an object if the object has any category that the subject lacks.

In addition to the simple security condition it is necessary to enforce the "*-property" (pronounced "star property") (Reference 3). The simple security condition allows a subject read access to objects only with a security level less than or equal to the security level of the subject. The *-property allows a subject to have write access only to objects with a security level equal to or greater than that of the subject (and with the proper categories). These properties are sometimes stated as preventing a subject from "reading up" (reading material with a classification greater than that of the subject) and from "writing down" (writing material at a classification level less than that originally assigned to the material).

The combination of classification and category set gives the security level of the subject or the object. Possible relations between two security levels include greater than, less than, equal to and isolated from.

Uniform application of the simple security condition and the *-property rules results in a system that is secure but not useful. Some subjects must be able to violate the property. These are called "trusted subjects". In particular, the "System Security Officer" is a

trusted subject supported by a special process in the security kernel. This special process can arbitrarily change the security level of a subject or object.

ARCHITECTURAL ALTERNATIVES

Reference Monitor

The model of a secure system shows the reference monitor must possess three characteristics:

* Complete mediation
* Isolation
* Provable correctness

The architecture must fully support these three characteristics as well as providing the necessary facilities to perform the required functions of the applications the secure architecture must support. The two most important of these secondary characteristics are efficiency and facility for information sharing. If a system is not efficient no real work can be supported. If information cannot be easily shared, the system might as well be a single level of security system.

Complete Mediation

The requirement for complete mediation of the reference monitor can be achieved in several ways. The reference monitor could use software to achieve all mediation. This would require up to an order of magnitude increase in the instruction execution rate since each address reference would have to be checked for validity by a software routine. This is clearly too inefficient to be practical. There are two ways that hardware could mediate the access for the reference monitor. The first might be called the static case. In the static case the memory space would be allocated statically. Each allocation would be checked for access privileges at allocation time. At the time of actual access the reference would only have to insure that the hardware performed correct limit checking to see that the access did not exceed the boundaries of the allocation (Reference 25). The limitations of this system require that all information sharing be performed through

the interprocess communication channel. This requires that the information by physically copied twice for each act of communication. This imposes an intolerable burden on the efficiency of the operation.

The other major hardware technique that can be employed is the implementation of a descriptor based virtual memory system. Each logical object is defined as a separate virtual memory segment with a descriptor giving the access controls. The reference monitor set the descriptors and gives access to subjects with the descriptors. The hardware must check via the descriptors for each access to real memory. If this application of the descriptors is implemented through hardware properly, the efficiency of the whole process can be made sufficiently high to support the needed systems capabilities.

Mediation must be performed both for cpu access and for I/O as well. In the virtual memory system, the I/O devices which respond directly to commands by transferring a unit of data at a time, can be treated as objects. Those which transfer blocks of data by direct memory access must be treated as subjects with given security capabilities.

Isolation

To enable the reference monitor to be securely isolated from all of the other programs in the system, the system must provide memory protection to the reference monitor and provide it with a set of privileged instructions which allow the reference monitor alone to control the creation and assignment of descriptors. The necessity for memory protection and special privileged instructions is common to all systems of implementation for the reference monitor.

Provable Correctness

For the system to be certifiably secure it is necessary to prove mathematically that the system observes the rules of the security model under all conditions. This means that each step from the model through formal specifications to algorithms and finally to computer hardware and software must be provable. The key to proving the correctness of the specifications, algorithms and programs is rigor and simplicity. The proofs must obey a rigorous formalism to be proved mathematically. The computer hardware and the requisite software to implement the reference monitor must be simple and straightforward.

Efficiency

There are several efficiency considerations that must be addressed. The sharing of information between subject has been mentioned. This is a primary consideration. If sharing is not an objective, the secure computing can be provided by several single level machines, uni-programmed with complete cleansing between users if necessary.

Another efficiency consideration is the overhead that the reference monitor imposes on the exchange of processor between multiple active processes (subjects). If there is too much overhead, then the sharing of the system by several subjects is severely impaired. This overhead is less visible than the more common concern of the overhead involved with the access by a single process to several descriptor based virtual memory segments as mentioned above.

MAPPING ADDRESSES

One way for a security kernel to perform its function of complete mediation, that is, of checking each reference of a subject to an object, is by providing a mapping function. In this view, the kernel maps (or translates) requests made by a subject into approved requests transmitted to an object. The central position of the SPM makes it possible to enforce such mapping.

The subject (process) addresses its resources (objects such as memory and peripheral devices) by names known to the subject and the kernel, commonly called "virtual" addresses. The kernel approves each request and maps (or translates) the virtual address to a physical address. Physical addresses are known to the kernel and the objects.

Figure 3 shows this mapping function of the SPM. The subjects, which are processes in operation, deal with virtual addresses for both memory and peripheral devices. The SPM checks these references by the security preservation rules and translates or maps the virtual address into a physical address. This mapping function is similar to that performed in the Multics system. The map can carry both translation information and much of the information from the security kernel data base.

Mapping memory addresses from virtual to absolute form is well understood and involves few complications. More elaborate memory management assistance, such as paging, may be incorporated into the hardware without introducing new kinds of architectural problems.

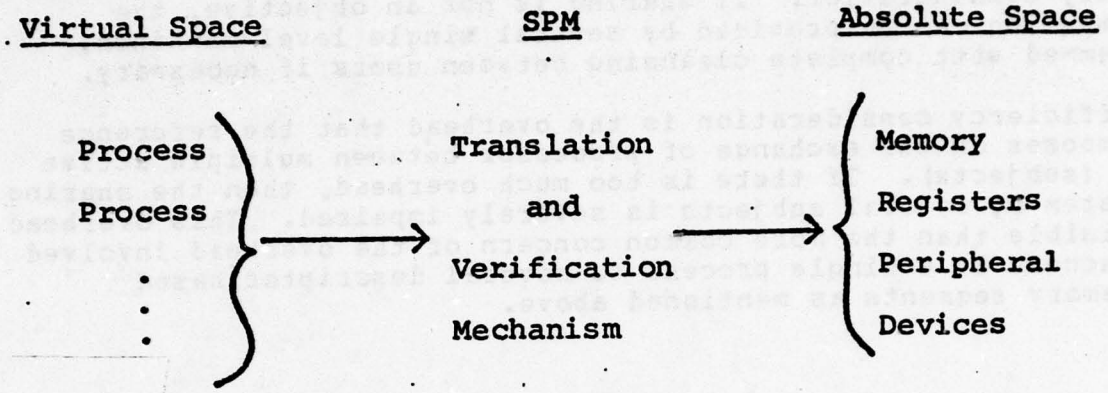


Figure 3

Mapping

Mapping peripheral addresses from virtual to absolute form is an extension of the mapping function to enforce security on peripheral operations. There are several ways to handle the representation, the peripheral devices exhibit wide variation in characteristics, and the resulting data exchange may extend over long periods of time. Two ways to handle the mapping are of particular interest here, the use of explicit names for devices and the consideration of devices as part of memory.

In the consideration of devices as memory, the subjects access the peripheral devices in exactly the same way as other objects. Each object has a virtual address and access capabilities. The SPM enforces the access limitations during the virtual address to absolute address translation process. The operation of I/O in this manner is the technique used in the PDP-11 Unibus architecture.

The problem with this approach arises with Direct Memory Access (DMA) type devices. These devices act like subjects as they access data segments to transfer their data. There are two choices for these devices, they can be given identifications as resources, i.e. set up as recognized processes, and provided with hardware connection to the SPM to allow their addresses to be translated by the SPM on the same basis as those of a CPU or they may be given absolute (pre-mapped) addresses and then must be certified to operate correctly, i.e. certified secure and treated as trusted processes, logically in the kernel domain.

Map Manipulations

The mapping functions must be maintained in a form accessible only to the security kernel to meet the requirement of isolation. The security kernel must be able to manipulate the maps in order to reflect changes in the operating state of the system. Manipulations required include:

* generating a resource description and placing it in a map
* altering a resource description in a map
* removing a resource description from a map

Examples are assigning a block of memory (core or secondary), assigning a peripheral device, extending the size of a memory block, or releasing a memory block or peripheral device.

Map Realization - Descriptors

Various schemes that provide for memory segmentation and paging have been known for many years (References 26, 27, 28, and 29). Based on past experience and current thinking, a a method has been developed which directly implements the abstract model of mapping. This is the hardware descriptor, as exemplified by the Honeywell level 68 processor used by the Multics system. The descriptor mechanism is essentially a formalized hierarchy of pointer tables. References by subjects (virtual addresses) are transformed to object references (absolute addresses) through one or more levels of descriptor. Figure 4 shows this in skeleton form.

Descriptor references perform the mapping of virtual to absolute addresses required by the model. Descriptors may be cascaded, with one pointing to another, etc., to handle complex transformations. Access control information may be included in the descriptor data so the descriptor reference may provide both the isolation and the complete mediation required of the security kernel.

Referring to Figure 4, each subject has a descriptor base root (DBR) which points to a table of memory descriptor segments (BASEM) and to a table of I/O descriptor segments (BASEI). Each descriptor can be indirect, i.e. a pointer to another descriptor (BASE D=001), or direct, i.e. describing an object (BASE D=100). The descriptors also carry access permission information. A direct memory segment descriptor may specify a memory location directly for unpaged segments or may describe the memory location through the use of a page table which allows paged segments.

PROTECTION MECHANISMS

Generalized Domains

Prior work (References 22 and 23) has led to the concept of a region or "domain" of execution for a process. A domain may be a partition of the process's resources: a subset of the memory and peripheral device space available to the process. A set of access control attributes such as Read, Write, Execute and Call may be associated with each domain. Access attributes may be granted or removed simultaneously for all elements in the domain. This provides a tool for one process to use an function while neither granting the function access to more information than it needs to accomplish the requested service nor granting the subject access to information it would not otherwise be able to reach. It is very efficient to associate a large, arbitrarily chosen, set of segments to form a common domain.

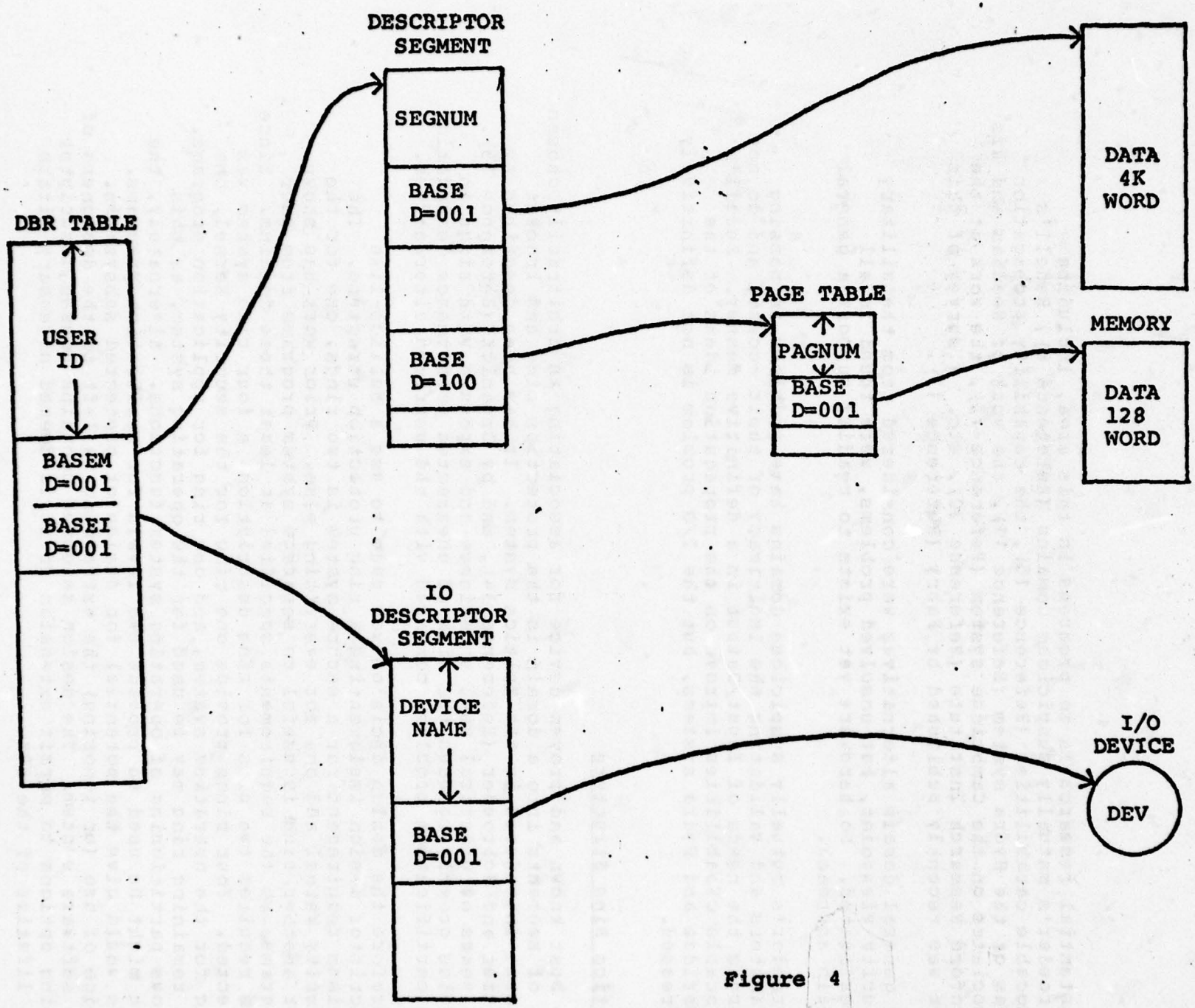


Figure 4

Descriptor Structure

Substantial research is in progress in this area, including Schroeder's mutually suspicious domains (Reference 8), Redell's revocable capabilities (Reference 10), the capability propagation ideas of the Hydra system (Reference 11), the work of Needham and his associates on the Cambridge system (Reference 12), the work at the Stanford Research Institute (Reference 30), etc. A survey of this work was recently published by Fabry (Reference 13).

The general domain alternatives were considered from the military security viewpoint, but unsolved problems, were found in all investigated. No hardware yet exists to readily support a general domain approach.

Schroeder's mutually suspicious domains have problems in passing parameters and validating the legitimacy of their access, and do not address the needs of Input/Output in a definitive manner. Redell's revocable capabilities improve on the propagation ideas of the Cambridge and Hydra systems, but the I/O problem is not definitively addressed.

Multics Ring Structure

The best known and proven device for associating an arbitrarily chosen set of segments into a domain is the protection ring and bracket concept introduced in the Multics system. It has been described by Saltzer and Schroeder (Reference 14), and by Organick (Reference 15). The years of practical use, experience and exposure with rings in Multics greatly reduces the risk of unexpected performance degradation and certification problems compared with the unproven alternatives.

Therefore the design decision was made to use a Multics-like descriptor design implementing a ring protection structure. The minimum requirement for a secure system is two rings, one for the security kernel and one for everything else. Prior work has shown that another ring is useful to separate system programs from user programs, so the requirements specified at least three domains. Since this requires two bits for ring description, a four ring system was selected. Four rings provide one ring for the security kernel, one ring for the operating system, and one ring for application programs. The remaining ring may be used for the operating system, as this allows partitioning of operating system functions. Alternately, the ring might be used to provide two levels for application programs. This would give the potential for design of protected subsystems. Choice of use (or ignoring) the extra ring is left to the designers of the software system. The design selected contains unused descriptor coding options to permit extension if it is deemed necessary within the lifetime of the SPM.

The hierarchical ring structure, as illustrated in Figure 5, considers the domains as concentric barriers, with execution in the innermost rings the most privileged. Descriptors used for controlling mapping include the ring privilege information of the subject. The usages of rings 1, 2, and 3 shown in this figure is only one possible assignment. Other allocations of these rings are possible.

Multics practice has shown the value of using three ring boundary indicators, called "ring brackets". An ordered triple of ring numbers (R1,R2,R3) specify the rings in which Read-and-Write ($0 \leq Reff \leq R1$), Read-and-Execute ($R1 \leq Reff \leq R2$), and Procedure Call ($R1 \leq Reff \leq R3$) are permissible. (5) No access is permitted with $Reff > R3$. A required relationship is $R1 \leq R2 \leq R3$. Figure 6 shows the interpretation of the ring bracket numbers.

The resource descriptor specifies the values of R1, R2, and R3 for the object. SPM hardware maintains a record of the current ring of execution. The effective ring of execution is controlled by the SPM and can only be changed by interrupt, trap, CALL, RETURN, or by the security kernel's using a privileged instruction. The validation of CALLs and their arguments and the handling of traps and interrupts by the security kernel ensures that users cannot change their ring of execution to breach security.

---

(5) Reff is the effective ring number, which is the current ring number as modified by the CALL mechanism. See Section V, "Ring Crossing".

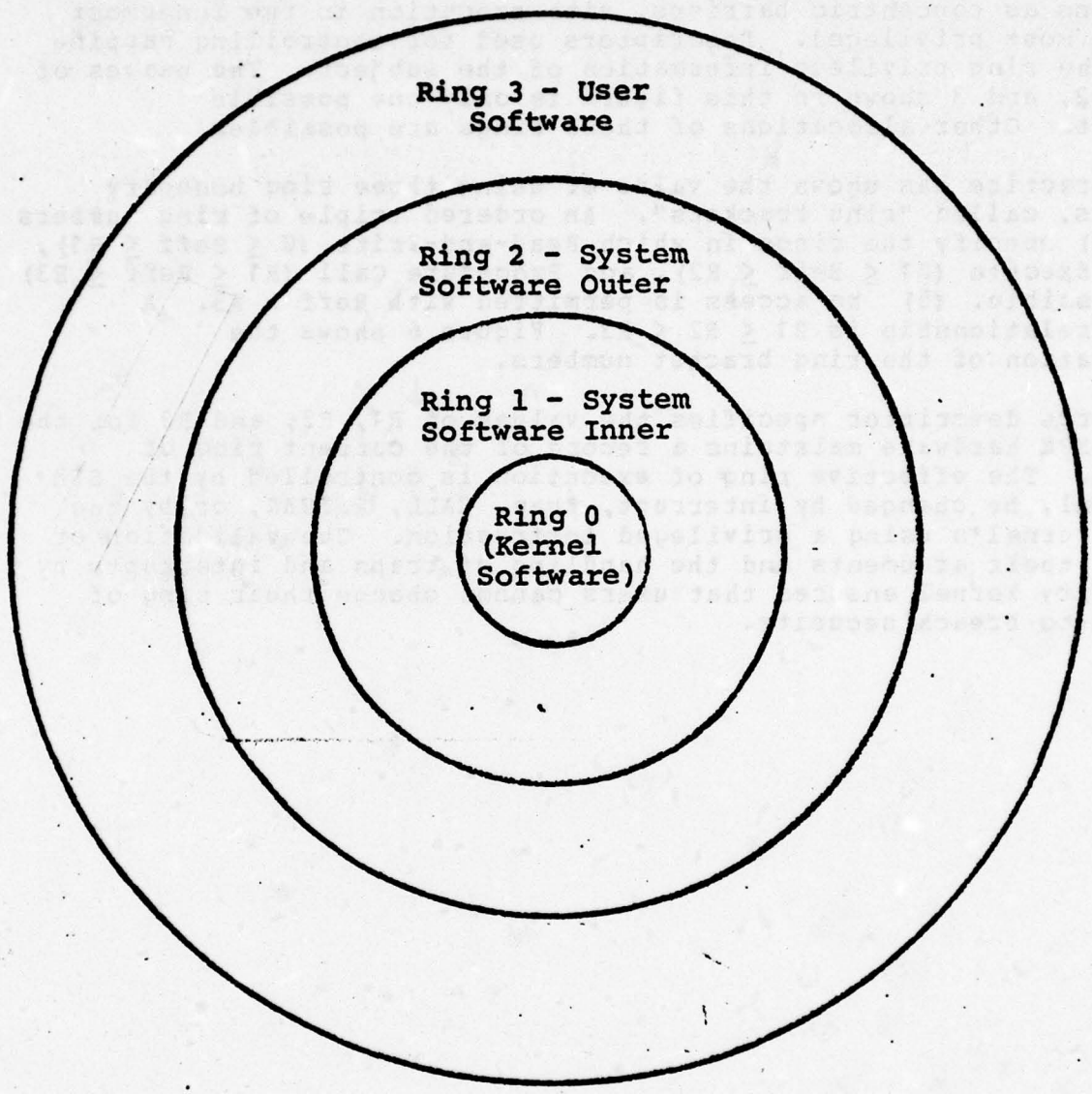


Figure 5

Rings

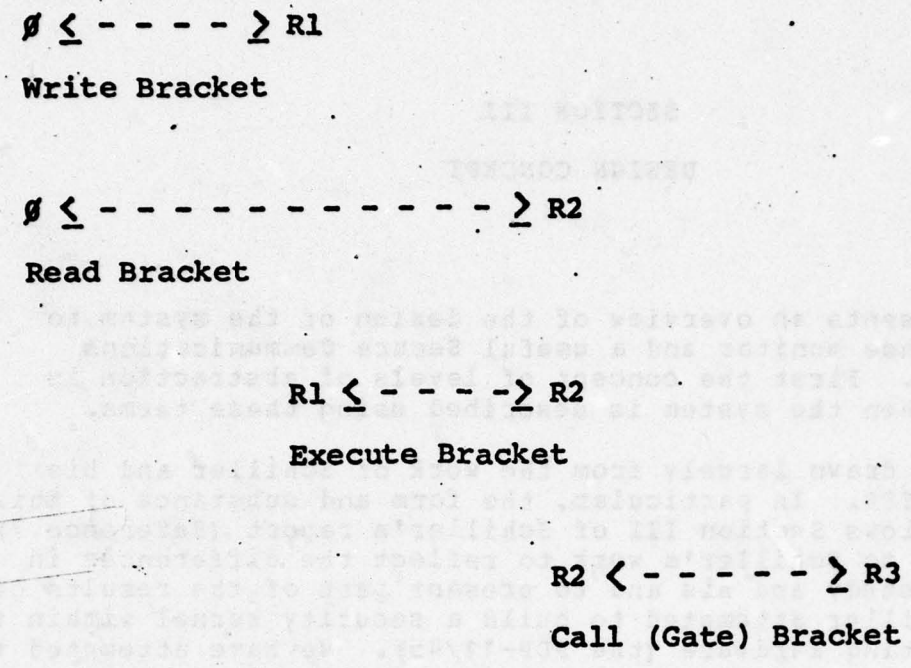


Figure 6

Ring Brackets

SECTION III

DESIGN CONCEPT

INTRODUCTION

This section presents an overview of the design of the system to provide a reference monitor and a useful Secure Communications Processor system. First the concept of levels of abstraction is introduced and then the system is described using these terms.

This material is drawn largely from the work of Schiller and his associates at MITRE. In particular, the form and substance of this presentation follows Section III of Schiller's report (Reference 7). Changes are made to Schiller's work to reflect the differences in purpose of this study and his and to present part of the results of this study. Schiller attempted to build a security kernel within the confines of existing hardware (the PDP-11/45). We have attempted to identify and specify a hardware architecture well suited to support a security kernel. Thus many of Schiller's firm constraints have become, in this study, outputs of the design.

LEVELS OF ABSTRACTION

Abstraction is a way of avoiding complexity and a mental tool by means of which a finite piece of reasoning can cover a myriad of cases (Reference 16). The purpose of abstracting is not to be vague, but to create a semantic level in which one can be absolutely precise. Dijkstra's levels of abstraction have been demonstrated to be a powerful design methodology for complex systems, most notably Dijkstra's "THE" system (Reference 17) and the Venus Operating System (Reference 18). In general, the use of levels of abstraction leads to a better design with greater clarity and fewer errors. A level is defined not only by the abstraction that it supports (for example, a segmented virtual memory) but also by the resources employed to realize that abstraction. Lower levels (closer to the machine) are not aware of the abstractions or resources of higher levels; higher levels may apply the resources of lower levels only by appealing to the functions of the lower levels. This pair of restrictions reduces the number of interactions among parts of a system and makes them more explicit.

Each level of abstraction creates a virtual machine environment. Programs above some level do not need to know how the virtual machine of that level is implemented. For example, if a level of abstraction

creates sequential processes and multiplexes one or more hardware processors among them, then at higher levels the number of physical processors in the system is not important.

By the rules of levels of abstraction, calls to a procedure at a different level must always be made in the downward direction, and the corresponding return in the upward direction. For maximum clarity, downward calls should be to the next lower level, but there will always be cases where calls that skip over one or more levels can be justified. Returns are always to the calling program, except in the event of a severe error where several of the calling procedures may be skipped over by the return. Figure 7 shows the structure of a system where most calls are to functions of the next lower level, but the

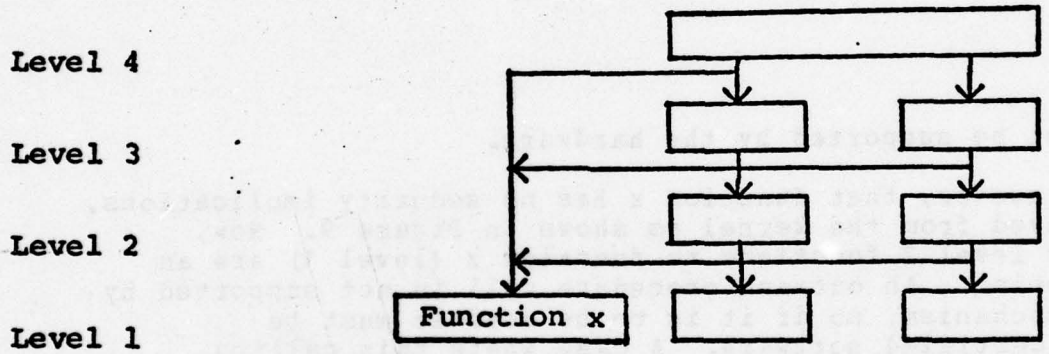


Figure 7 Levels of Abstraction

level 1 function x is called from levels 2, 3, and 4.

When a ring (hierarchical domain of execution) structure is added to the system, simplicity is enhanced by having each ring consist of contiguous levels. Thus the kernel, which must be the innermost ring (ring 0), should consist of the level of abstraction that implements the reference monitor concept and the supporting levels beneath that level. In our example system, the boundary between ring 0 and ring 1 may come between level 2 and level 3 as shown in Figure 8. Following the policy of making a ring consist of contiguous levels, all cross ring calls are automatically to an inner ring. This type of ring

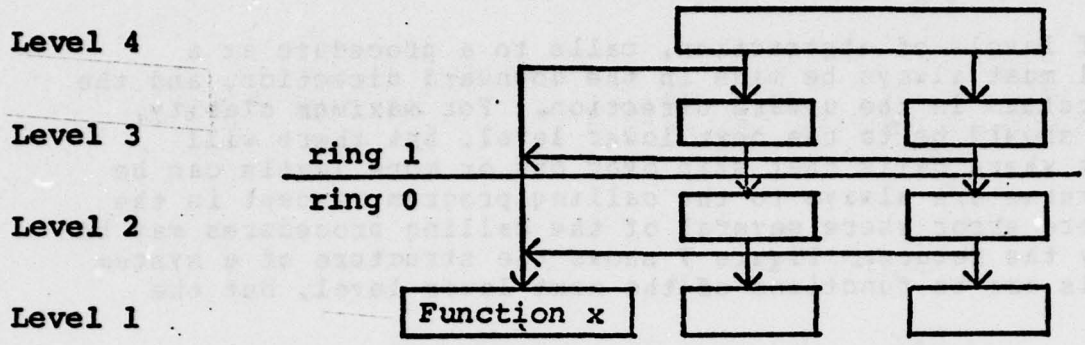


Figure 8 Levels of Abstraction and Rings

crossing call must be supported by the hardware.

It is possible, however, that function x has no security implications, so it can be removed from the kernel as shown in Figure 9. Now, however, calls by level 2 functions to function x (level 1) are an outward ring crossing. An outward procedure call is not supported by a hardware ring mechanism, so if it is to be used it must be implemented with certified software. A case where this calling structure might occur is with the scheduler of a multiprogramming system. The scheduler may appear at a low level of abstraction, but if we make a distinction between the scheduler - code that implements the policy that selects the next process to run - and the process multiplexor - code that implements the mechanism that binds a process to the hardware, - then it can probably be proved that the correctness of the scheduler is not necessary for security. Thus, we would want to remove it from the kernel, in spite of the fact that it may be called from the kernel. It would, however, be necessary to show that a malicious user could not use the scheduler to pass information in violation of security.

This example illustrates an apparent conflict between the goal of overall system clarity and the goal of a small and simple kernel. One could argue that either one of these goals, or the use of levels of abstraction with its requirement of strict hierarchical layering, or

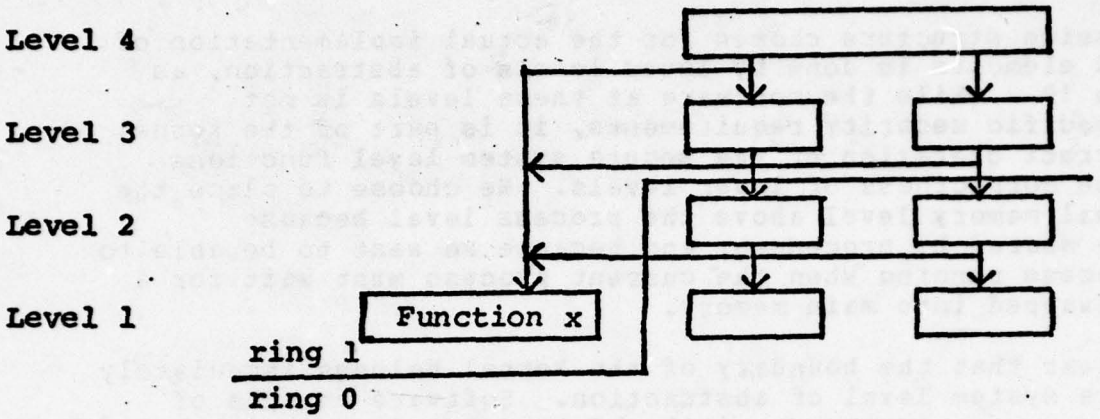


Figure 9 Outward Ring Crossing

the use of protection rings causes the conflict. A machine that provided the more general form of non-hierarchical protection domains would solve this problem by allowing an internal partitioning of the kernel. Domain machines, however, are not currently available. Since we have decided to take advantage of the proven theory of a ring machine and we believe that the levels of abstraction design methodology will facilitate certification of the kernel, we must consider compromising one or both of the design goals of overall system clarity and a small simple kernel. This issue will be discussed further as design details are presented.

LEVELS OF THE KERNEL

In designing the hardware to support a security kernel, levels of abstraction have been used in the translation of the abstract elements of the mathematical model to tangible elements of a secure computer system. The first steps taken were to make an interpretation of the model elements (i.e., objects are virtual memory segments and subjects are sequential processes) and to provide at some level of abstraction a set of functions that controls access to these elements. Thus the abstraction created by this level is that of a secure computer system. It must be emphasized that what this secure system level of abstraction does is to effect the implementation of the reference

monitor, thus insuring that the system is always in a secure state.

The specific design structure chosen for the actual implementation of the interpreted elements is done by lower levels of abstraction, as shown by Figure 10. While the software at these levels is not cognizant of specific security requirements, it is part of the kernel because the correct operation of the secure system level functions depends upon the correctness of lower levels. We choose to place the segmented virtual memory level above the process level because segments can be shared by processes, and because we want to be able to start a new process running when the current process must wait for a segment to be swapped into main memory.

It should be clear that the boundary of the kernel belongs immediately above the secure system level of abstraction. Software outside of this perimeter can execute the unprivileged hardware instructions and invoke the functions provided by the secure system level with arbitrary arguments. Since the unprivileged machine instructions cannot put the system into a non-secure state and the secure system functions are proven secure for all arguments passed to them, the security of the system is independent of what the software above the secure system level of abstraction does or does not do. Thus the implementation of the security level of abstraction and the implementation of the lower, supporting levels, gives us a complete security kernel.

Five levels of abstraction are used to describe the system including the security kernel. The highest (Level 4) includes most of the software of the system and is exterior to the security kernel. The next level (Level 3) holds the security kernel procedures responsible for checking acceptability of access to resources and for requesting the levels below to create appropriate descriptions of the resources. The system is to be implemented so that no resource may be used without this description provided by the lower levels. Level 2 creates a segmented virtual memory for the use of processes running at higher levels. Level 1 creates and manipulates the machine language descriptions of the resources handled by the higher levels. Level 0 is the hardware of the system itself. Our concern in this study was to design the hardware of Level 0 so that the functions of the higher levels could be supported completely and efficiently.

LEVELS 4 AND GREATER - UNCERTIFIED SOFTWARE

The software of Level 4 (and higher) is the reason for existence of the entire system. The applications programs, such as communications programs, message switching software, or front-end processing programs operates at the higher levels. At Level 4 is the operating system

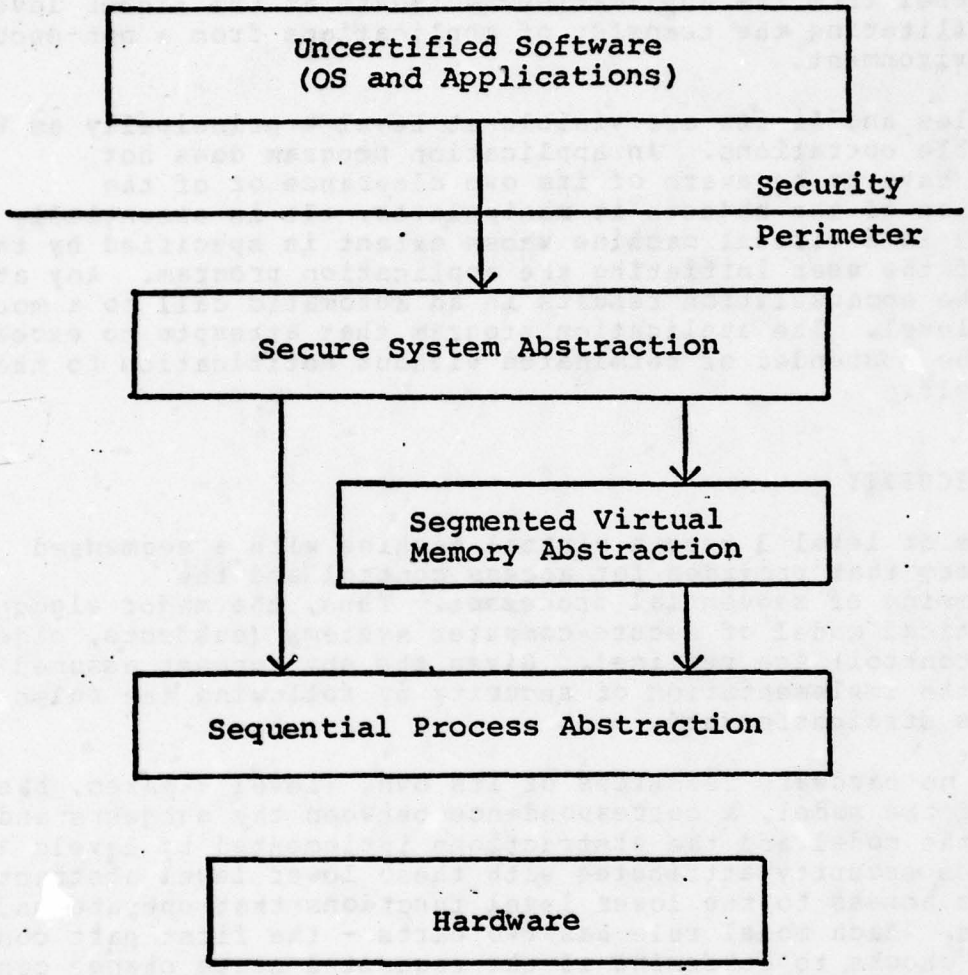


Figure 10

Kernel Structure

which, in turn, exists to serve and support the applications programs. The operating system at level 4 can serve to mask the operation of the security kernel from the applications software at the higher levels, thereby facilitating the transfer of applications from a non-secure to a secure environment.

Security rules and limits are visible at Level 4 principally as bounds on permissible operations. An application program does not necessarily have to be aware of its own clearance or of the classification of the objects it manipulates. It is essentially encapsulated in a virtual machine whose extent is specified by the clearance of the user initiating the application program. Any attempt to escape the encapsulation results in an automatic call to a more privileged level. The application program that attempts to exceed its limits may be suspended or terminated without notification to the program itself.

LEVEL 3 - SECURITY

The software of level 3 sees a virtual machine with a segmented virtual memory that provides for access control and the multiprogramming of sequential processes. Thus, the major elements of the mathematical model of secure computer systems (subjects, objects and access control) are realized. Given the environment assumed by the model, the implementation of security by following the rules of the model is straightforward.

Level 3 has no hardware resources of its own. Level 3 makes, based on the rules of the model, a correspondence between the subjects and objects of the model and the abstractions implemented by levels 1 and 2, associates security attributes with these lower level abstractions, and controls access to the lower level functions that operate on these abstractions. Each model rule has two parts - the first part consists of security checks to determine if the requested state change can be permitted; the second part of the rule indicates how the state change is to be made. In the kernel, level 3 functions perform security checking and then direct levels 1 and 2 to perform state changes if security requirements are satisfied. Level 3 maintains the access matrix which records the security level information on all subjects and objects. This is used to direct the lower level machines to invoke the proper access control.

The kernel uses processes as the basic interpretation of subjects, and segments as the basic interpretation of objects. In addition, semaphores and interprocess communication messages are also objects. The data structures used by level 1 to support processes include a specification of each process's current address space (the segments

accessible) and an identification of the user associated with the process together with the user's security attributes.

Since the model has been implemented at Level 3, all remaining software above it in the system can be uncertified - contain bugs or malicious penetration attempts - without a threat of security compromise if two conditions are satisfied. First the kernel must be protected, and second, access to its functions must be controlled. These conditions are met by preventing uncertified software from gaining write access to the kernel segments and by permitting only the kernel to execute in the inner ring.

LEVEL 2 - SEGMENTED VIRTUAL MEMORY

The second level of abstraction creates a segmented virtual memory, building on the segmented main memory provided by level 0 (the hardware). Segments are the primary storage entities of the system and will be the basic object to which access is controlled by the security level of abstraction. Since even the largest hardware segment may be small for some applications, we anticipate the creation of a virtual memory abstraction . This abstraction will allow several segments to be treated as a single object and permit subsections (segments) to be individually swapped in and out of main memory. Level 2 implements a one level virtual memory because: 1) segments are the only type of storage entity, and 2) as different segment descriptors are loaded into the segmentation registers the address space of a process can be greater than the size of main memory. For applications requiring very large collections of data the operating system can create a file structure at level 4 or higher using the virtual memory created here.

One would like (as Multics does) to implement variable sized segments consisting of fixed sized pages. The use of paging facilitates the dynamic growth of segments, permits only part of a segment to be swapped into main memory, and vastly simplifies the allocation of both primary and secondary memory. This has been left as an option for implementation.

Segments have attributes - information that describes the characteristics of a segment. From the mathematical model we know that at level 3 segments will have security attributes; at level 2 they have implementation attributes. Implementation attributes include a segment's size and disk address. The attributes of a segment are contained in an entry in a directory. A directory is an object which describes other objects. Each directory entry contains the attributes of the named object. At level 2 space is provided in directory entries for security attributes but the operation of this

level is independent of the values of security attributes. To simplify conceptual design and implementation, directories are themselves segments with attributes residing in other directories, the total structure is a directory hierarchy in the form of a tree. The attributes of the root segment (directory) of this tree are fixed by the design and implementation.

All segments in the hierarchy are either directory segments or data segments. (Segments containing executable code are considered data segments by level 2.) Although level 2 does not enforce access control to segments in general, it cannot permit software above it to write directly into directory segments, because the correct operation of level 2 requires the integrity of the (implementation) attributes of segments. Functions at this level provide users with an interpretive directory write capability. The security requirements enforced at the security level will further restrict access to directories because of the nature of some of the segment attributes.

To allow dynamic segment sharing, level 2 associates a semaphore (6) with each segment and requires write access to the segment in order to operate on the semaphore. (See the discussion of Level 1 for more detail on semaphores and the operations on them). The I/O segment semaphores have a special use - the kernel translates I/O interrupts into releases of the appropriate semaphores. Thus, when a process wishes to wait for an interrupt from an I/O device, it seizes the I/O segment semaphore, (presumably) blocking itself. When the interrupt occurs, a release is performed and the process becomes unblocked. The kernel is only concerned with controlling access to the I/O segments and semaphores, not with the correct use necessary to assure proper synchronization.

Level 2 implements a segmented virtual memory by building upon level 0's segmented main memory, using secondary storage devices for segment swapping, and employing a data base to indicate the state of the virtual memory. The data base consists of the directory segments, tables for managing the allocation of secondary storage, and the Active Segment Table (AST). The AST is a mechanism that facilitates

---

(6) A semaphore is a special variable accessible only to two special operations, seize (privé, P) and release (vrij, V). When a process needs exclusive access to a resource, it must seize (P) the associated semaphore. If the resource is in use when the seize operation is invoked, the process is suspended until some future time when the resource is released by another process. After a process is finished with exclusive access to a resource, it must invoke the release (V) operation to allow other processes access to the resource. See Reference 31.

the sharing of segments in main memory - if two different processes wish to access the same segment they both access the same physical segment and not two different copies. Any segment that is in the address space of one or more processes or is "wired down" (permanently swapped into main memory) is active - it has an entry in the AST. An active segment table entry (ASTE) contains the segment's permanent attributes - copied from the directory - as well as additional attributes associated with the fact that the segment is active. These additional attributes include a list of the processes that have the segment in their address space and the main memory address of the segment if it is currently swapped in.

Segments in the hierarchy can be uniquely identified in a variety of ways. If a segment is active, identifying its entry number in the AST (ASTE#) specifies the segment. If a segment is not active but its parent is, then the ASTE# of the parent directory and the identification of the entry within the directory that contains the segment's attributes (directory, entry#) specifies the segment. A generalization of the (directory, entry#) identification method is the complete pathname - a specification of all directory entries, beginning with the root, that identify the segment. Finally, each segment has a unique identifier - its disk address. Within the security kernel the primary segment identification techniques are the ASTE# and the (directory, entry#). See Reference 23.

If provision is made for descriptors of I/O and peripheral devices, which are also forced to be completely unalterable by any higher numbered level, Level 2 supports all system storage in a secure manner. That is, I/O and peripheral devices may be accounted for by data structures analogous to those for segments. If compatible controls are placed on I/O and peripheral devices, they can be handled like segments.

Level 2 provides functions for creating and deleting segments, adding and removing segments from a process's address space, and creating and destroying segment descriptors, for both data and directory segments. The segments created at this level are the basic interpretation of the objects of the mathematical model. Although segment descriptors permit access control to segments, the only access control policy enforced at this level is the requirement for interpretive directory writes.

LEVEL 1 - SEQUENTIAL PROCESSES

Level 1 creates the process abstraction. We use the "standard" (and somewhat vague) definition of process - a process is a procedure in execution. See Reference 23. The design supports a variable number

of processes; each runs on a virtual machine and consists of an address space and control information about the process. At level 1 it is sufficient to know that the address space is defined by the control information, part of which is the contents of the descriptor segment. Level 1 software has the responsibility for allocating the processor to one of the processes whose dynamic progress is permissible.

At a given time, a process is in one of several possible execution states (Reference 19). Figure 11 shows the relationships among the various execution states and the actions that move a process from one state to another. In the inactive state a process does not have an address space and cannot run. A process can only be moved out of (and also into) the inactive state by the supervisor, a special executive process that is never in the inactive state. At the time that it moves a process out of the inactive state, the supervisor must establish the initial address space of the process. The purpose of the inactive state is to create a mechanism for minimizing the resources required to support a process that is not currently needed.

An active process is either blocked or unblocked. In the blocked state, a process is waiting for the occurrence of some event. An unblocked process is either in the running state or the ready state. The running state simply signifies that the process has the CPU allocated to it. Above level 1 the running and ready states are logically equivalent. In the ready state a process is ready to run (its dynamic progress is permissible) but must wait for the CPU to be allocated to it. Processes enter the ready state from the blocked state when the event for which they were waiting occurs. Transition of processes between the ready and running states is controlled by a mechanism internal to level 1.

The hardware resources of this level are the CPU and a real time clock. A data base is employed to contain state information about the processes and to help manage them. This state information includes a definition of each process's address space, an indication of its execution state, and a specification of the user associated with the process and his security attributes. No interpretation of these security attributes is made at this level (the operation of level 1 is independent of their value). Rather, space is set aside in level 1's data base for security attributes as a convenience for higher levels.

Several different types of functions are provided by level 1. Two sets of functions are provided for the synchronization of processes - semaphores and Dijkstra's P and V functions primarily for intraprocess coordination along with message send and receive for interprocess communication. In the level 2 subsection we noted how P and V are used to handle I/O interrupts.

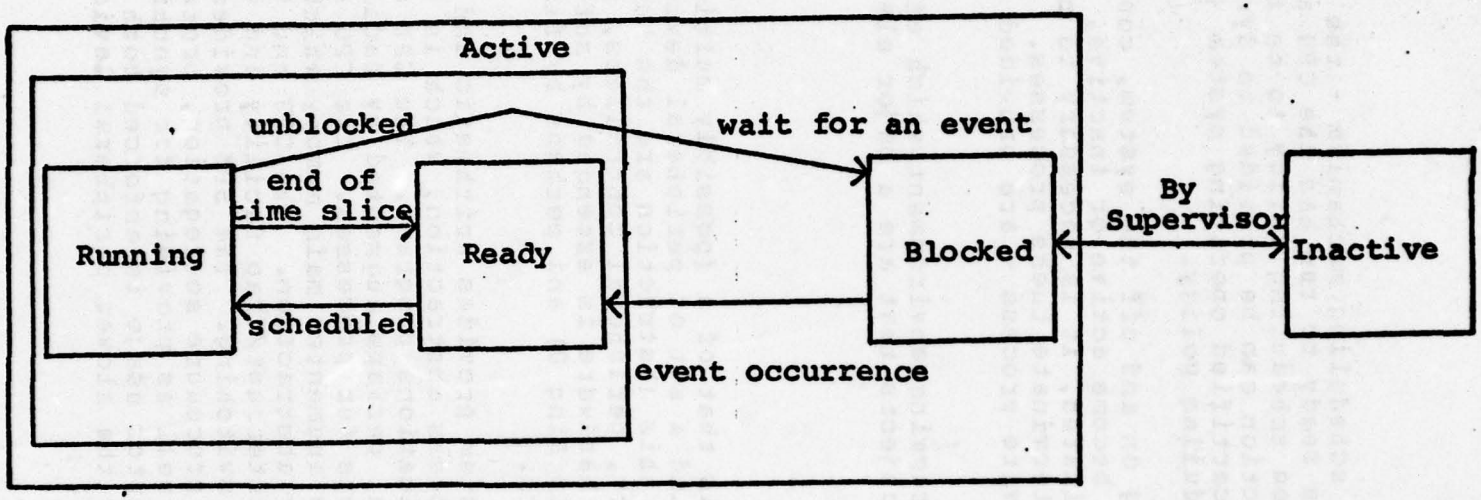


Figure 11

Process Execution Statos

Level 1 implements a simple scheduling mechanism - the highest priority (7) process that is ready to run has the CPU allocated to it. To allow a more sophisticated scheduling policy to be implemented outside of the kernel a function can be provided to dynamically change process priorities. The uncertified operating system (Level 4) normally provides this scheduling policy.

Finally, as online users log on and off the system, communications lines or control procedures become active or inactive, and as batch jobs are initiated and terminated, it is necessary to provide them with processes and then to terminate these processes. Two functions - activate process and deactivate process - are provided for this purpose.

Level 1 creates a multiprogramming environment which effectively implements the coexisting subjects that are a major element of the model.

LEVEL 0 - THE HARDWARE

The abstraction at Level 0 is that of a (possibly multi-) processor with augmented main memory and a set of peripheral devices. The basic resources used in realizing this abstraction are the hardware of the protected system (CPU, memory, peripheral controllers, peripheral devices) and the SPM. This hardware is extended by software in the security kernel (operating in Ring 0) and perhaps by firmware in some of the peripheral controllers.

The CPU of the protected system provides arithmetic and logical functions to support the process abstraction, which is used both for the kernel and for the applications programs. The CPU also supplies the real time clock function, perhaps augmented by facilities of the SPM to give unique identifiers for processes. The CPU and the SPM work together to support the augmented main memory which provides for the segmented virtual memory abstraction. The CPU and the SPM also cooperate in providing the interrupt/trap facility and the ability to perform secure fast context switching. The SPM provides hardware support for access control, procedure segregation, protection rings, and segment description, as well as providing for synchronization of multiple processors. Descriptor usage is enforced both for the rapid core memory accesses and for the slower peripheral device accesses.

---

(7) Priority as used here may consist of many variables as perceived by the overall system. At this level however, all considerations are reduced to a single integer used for ordering.

The next section of this report is devoted to an expansion of the characteristics of the hardware, with particular attention paid to the SPM.

SECTION IV

SPM DESIGN

INTRODUCTION

This section will be devoted to an expansion of the requirements and characteristics of the hardware needed to produce a Secure Communications Processor. Particular attention will be paid to the specification of functions for the new element in the system, the SPM. Software requirements will be mentioned only when needed to identify a hardware capability or when the function is deemed best carried out by software.

The reader will recall the picture of the ideal reference monitor, shown in Figure 1. The idealized reference monitor sits between all subjects and all objects and mediates every reference of a subject to an object. The SPM has been devised to follow this ideal as closely as practical. The central position of the SPM was shown in the protected computer system of Figure 2.

The block diagram of Figure 12 shows a minicomputer system which is suitable both as a communications processor and also as the basis for a protected system and thus for a Secure Communications Processor. All components of the system are connected by the common bus which transfers both control information and data between the units of the system.

Figure 13 shows the same minicomputer system with the addition of the SPM. The common bus is now divided into a virtual bus and an absolute bus. The virtual bus carries addresses in virtual (unmediated) form while the absolute bus carries addresses in absolute (mediated) form. The SPM serves to mediate and translate the addresses from virtual to absolute form, applying the rules of the security model in the process. In the ideal case, the SPM would perform all of the functions of the reference monitor and realize Figure 1 directly. Unfortunately this would require that the SPM essentially duplicate the hardware of the system to be protected, losing the savings that were the reason for introduction of the SPM in the first place. Practical considerations force us to make multiple use of the hardware of the protected system to implement the reference monitor.

In Figure 13, the device coupler, supporting the high data rate peripherals - tape and disk - is placed on the absolute bus in contravention of the model requirements. This is a practical consideration, since every use of the virtual bus is individually

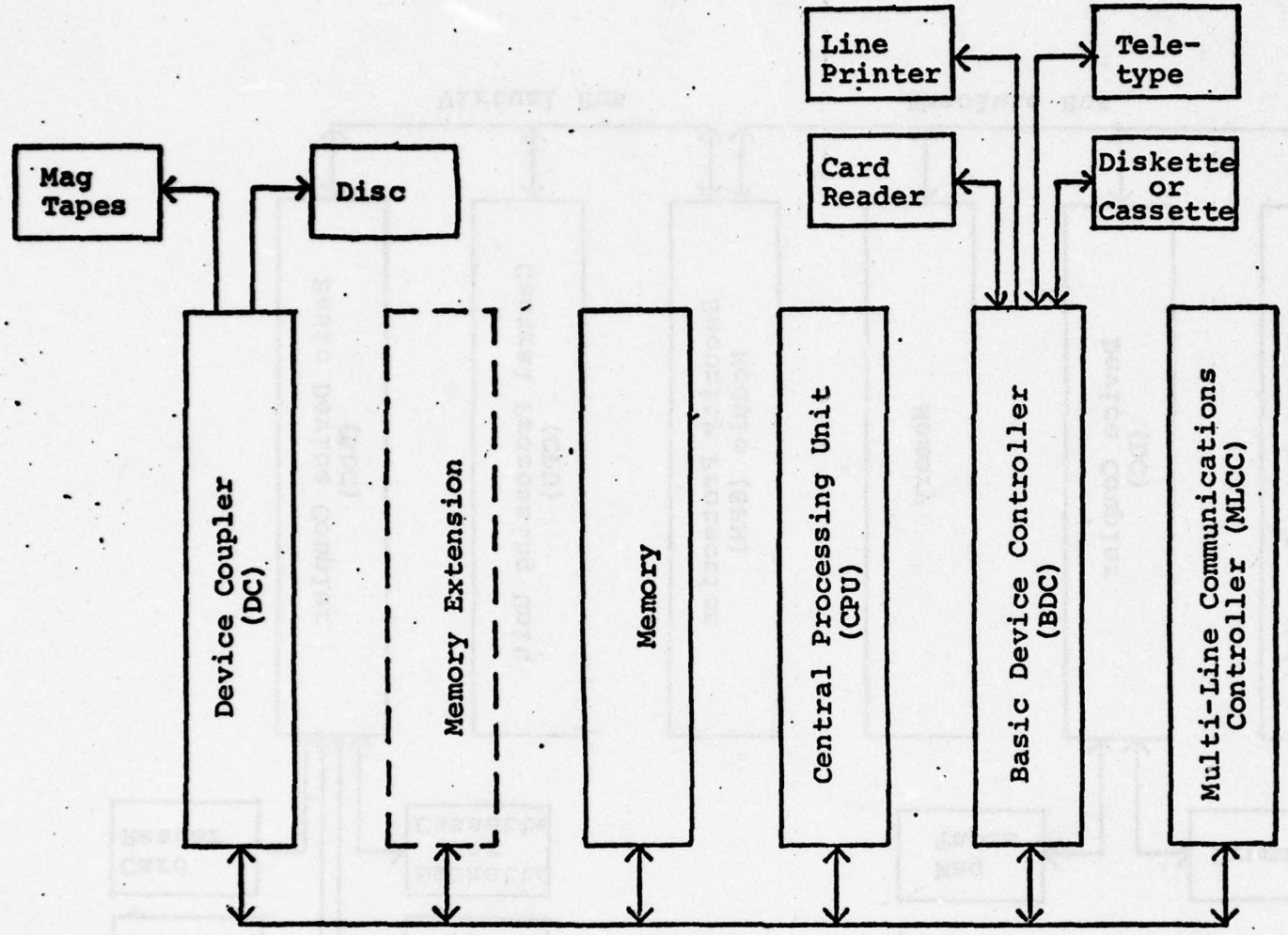


Figure 12

Host Computer

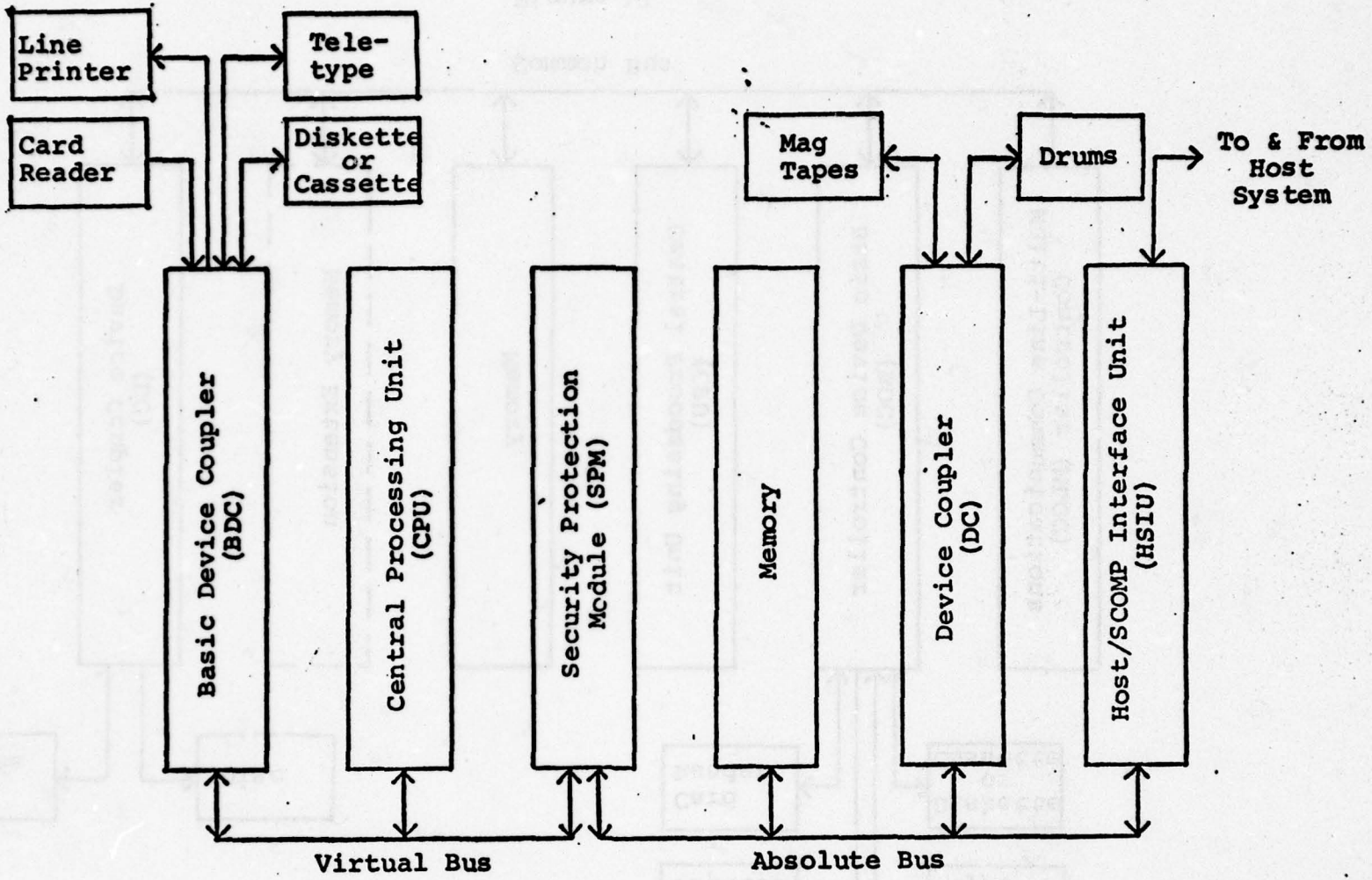


Figure 13

Host Computer With SPM

mediated by the SPM, the addition of several high speed peripherals transferring data at the same time could drive the speed requirements of the SPM beyond the current state of the art. The device coupler being on the absolute bus must assume some of the duties of the reference monitor in I/O transfers.

Hardware of the SPM, CPU, and memory components of the system is used to support the security kernel. The SPM provides required security capabilities not built into the CPU, acting as a "security unit" in much the same fashion as a "floating point" unit supplies its services. The CPU provides logic and arithmetic capabilities for the various processes, among them the security kernel, operating system, and application programs. The memory stores instructions and data for all processes from security kernel through application program.

The security kernel operates as normal software, using the CPU and memory. The security kernel operates in Ring 0, which grants it full system privileges. Among these privileges is the ability to directly address and control the SPM, the ability to address and modify the descriptors stored in the memory, and the ability to utilize restricted instructions, functions of the SPM, and other resources which are forbidden to other (non-Ring 0) software.

The operating system is also software which uses the CPU and memory. It operates in Ring 1 (and possibly in Ring 2 as well). The operating system has some of the privileges normally associated with a "master mode" (I/O functions), but none of the special privileges accorded to the security kernel in Ring 0. The operating system may call on the security kernel for sensitive activities essentially in the same fashion that an application program calls on the operating system. The action of the SPM in translating and checking addresses is invisible to the operating system. The Secure Communications Processor will require design and implementation of an operating system to properly couple with the security protection characteristics of this system.

Application programs also operate using the CPU and memory, but reside in the outer ring (or rings). They have the restricted privileges normally associated with a "slave mode", calling on the operating system for services which are beyond their realm of privilege. The actions of the SPM are invisible to the applications programs, except for security constraints. It is expected that applications or systems programs that operate on the unprotected system will continue to operate on the protected system of the Secure Communications Processor, so long as they obey the rules of security. Any attempt to communicate beyond their security privilege will result in suspension of the offending process.

The choice of four rings for the SPM leaves a major system design area open for the implementor of the secure system. There is one ring available for use to partition the operating system or to partition the application programs or to partition the kernel if it proves to aid verification. If the designer chooses to use it for the operating system, then the operating system will occupy Ring 1 and Ring 2. This choice allows the operating system to be divided into sections of greater and lesser privilege and may lead to a cleaner and simpler design than is normally encountered. Alternately, the designer may chose to use two rings for application programs. This will allow definition of protected subsystems by users of the computer. Application programs could then be partitioned to use the two rings much as a conventional computer system uses its "master" and "slave" modes. The restrictions placed on the designer are use of Ring 0 for the security kernel, use of Ring 1 for the operating system, and use of the two remaining outer rings as best suits the particular situation.

THE SPM

Figure 14 shows a block diagram of the SPM. Within the SPM there is a set of control registers and logic circuits which perform the security functions and respond to the SPM commands issued by the security kernel. At this level, the security kernel is recognized as the CPU operating in Ring 0. There is also a mechanism that performs the address mapping from virtual address to absolute address and also checks for proper access permission. If an error, or an attempt to violate the security rules is detected, then the security kernel is signaled via the Error Trap Line. This trap signal must be directed to activate the security kernel (operating in Ring 0) and only to that process.

Among the security related items of information that must be under the control of the SPM hardware are the current Process_id, the current effective ring number, and the descriptor base root and set of descriptors used by the current process. These items may be stored in the memory of the system, rather than physically within the SPM, but they must be under the exclusive control of the SPM and the security kernel.

SUPPORT OF THE PROCESS ABSTRACTION

A process was referred to earlier as an address space/processor state pair, recognizing that a process may be identified both by the resources it owns and by the activity of the processor that executes it. It is essential to be able to explicitly identify each process

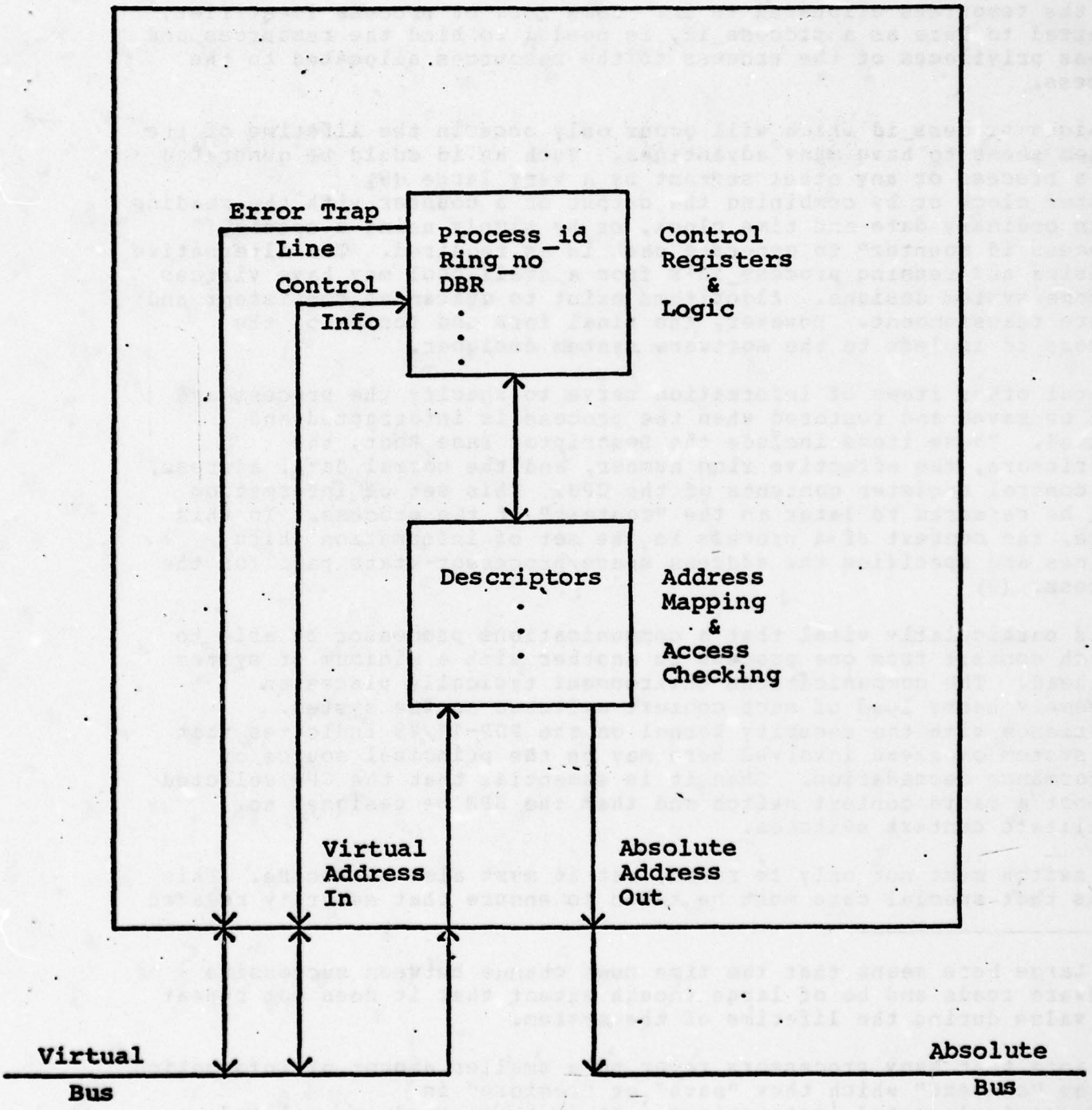


Figure 14

SPM Block Diagram

and the resources allocated to it. Some form of process identifier, referred to here as a process_id, is needed to bind the resources and access privileges of the process to the resources allocated to the process.

A unique process_id which will occur only once in the lifetime of the system seems to have many advantages. Such an id could be generated for a process or any other segment by a very large (8) system clock or by combining the output of a counter with the reading of an ordinary date and time clock, or by simply using a special "process_id counter" to generate each id as required. The alternative of using and reusing process_id's from a small pool may have virtues in some system designs. Algorithms exist to guarantee consistent and secure reassignment. However, the final form and format of the process_id is left to the software system designer.

Several other items of information serve to specify the process and must be saved and restored when the process is interrupted and resumed. These items include the Descriptor Base Root, the descriptors, the effective ring number, and the normal data, address, and control register contents of the CPU. This set of information will be referred to later as the "context" of the process. In this sense, the context of a process is the set of information which defines and specifies the address space/processor-state pair for the process. (9)

It is particularly vital that a communications processor be able to switch context from one process to another with a minimum of system overhead. The communications environment typically places an extremely heavy load of such context switches on the system. Experience with the security kernel on the PDP-11/45 indicates that the system overhead involved here may be the principal source of performance degradation. Thus it is essential that the CPU selected support a rapid context switch and that the SPM be designed to facilitate context switches.

The switch must not only be rapid, but it must also be secure. This means that special care must be taken to ensure that security related

---

(8) Large here means that the time must change between successive hardware reads and be of large enough extent that it does not repeat any value during the lifetime of the system.

(9) Note that many processors refer to a smaller amount of information as the "context" which they "save" or "restore" in hardware-implemented instructions. It is these hardware related instructions which are mentioned subsequently in Section V.

items of the context are not alterable by any software other than the security kernel itself.

In some systems, it must be noted that the protected processor itself may be wired to produce context switches upon occurrence of certain events, usually referred to as "faults" and "interrupts". "Faults" are usually taken to be events detected automatically within the process currently being executed by the processor; "interrupts" are events detected by the system hardware which involve actions not necessarily within the process currently in execution by the processor. Often the response to "interrupts" may, or even must, be postponed for many instructions, but a response to a fault, especially if it indicates a potential hardware failure, usually brooks no delay beyond the completion - as best as possible - of the current processor action. For this reason, the protected processor may contain within itself implementation of automatic partial context switching in these cases.

A more normal implementation of fault and interrupt processing is to switch instruction streams and, usually, protection rings, to process the fault or interrupt. The result of this processing which occurs within the process in operation when the external event happened, may be a change of process. However, the change of process as contrasted to the change of instruction stream will not automatically happen, but will be the result of an analysis of the event.

The system must be rendered secure without forcing a gross redesign of these functions. The presence of the SPM will inevitably introduce the need for additional security related and non-security related context switching information to be generated in the presence of both faults and interrupts. The SPM design must add this information gracefully and securely to the context, to resume activity later.

The security kernel must also be the controlling entity in the creation of a new process, in the allocation of its resources, and in the eventual release of those resources and the destruction of the process. All functions involved in these activities must have the privilege of the security kernel (in Ring 0). The functions required of the SPM will be listed later.

SUPPORT OF THE AUGMENTED MAIN MEMORY ABSTRACTION

The augmented main memory abstraction is used to provide the virtual memory of higher levels. It is supplied by combining use of virtual addresses in the CPU and translation of these addresses to absolute

form by the SPM. The concept of descriptors and of descriptor based addressing was introduced earlier. Figures 3 and 4 introduced the idea of address translation and of the descriptor technique.

Use of descriptor based addressing allows the choice of paged or unpaged memory to be left to the system designer and brings great consistency to all memory transformation actions. It also allows inclusion of access checking into the process of address transformation.

Support of virtual memory requires that, in the address transformation process, a non-resident segment fault (and if paging is used, a non-resident page fault) be provided. Since in an active paging system supporting several processes with large virtual memories, the page fault will be a very common occurrence. The efficient support by the processor and the SPM of page and segment faults is another area of concern in architecture selection. If paging is used, the page fault processing and restart must be efficiently handled.

Consideration of typical minicomputers used for communications processing led to selection of a 22 bit virtual address format for illustrative purposes. This format is shown in Figure 15, set up for use in both an a paged and an unpaged environment. The selection of 22 bits is a compromise between the desire for the largest possible virtual memory space and the hardware needed to support the larger virtual address field. The larger address field imposes a penalty in the CPU where it is necessary to manipulate the larger address, in the memory where it is necessary to store virtual addresses and in the SPM where it is necessary to mediate them. The wider hardware paths for virtual addresses cost both in money and in reliability from increased parts count. Typical field sizes for segment number, page number and offsets have been selected for illustration, but the actual field size used is at the discretion of the system designer. Within the 22 bit virtual address this virtual address format will be the one used by the CPU and transmitted to the SPM.

Figure 16 shows a typical layout for the Descriptor Base Root. A two word descriptor base is provided for memory, BASEM and BASEM Ext, and another for Input/Output, BASEI and BASEI Ext. Both provide for a descriptor type field "D" which may be used to indicate chaining or indirection in the use of the descriptors. Both also provide for a limit field, used to restrict the extent of the reference permissible by the descriptor base. The fields marked "RFU" are reserved for future use.

Figure 17 shows typical layouts for memory descriptors and for I/O descriptors. Again, the field lengths are typical values assuming a 16 bit CPU, subject to alteration by the system designer. Provision

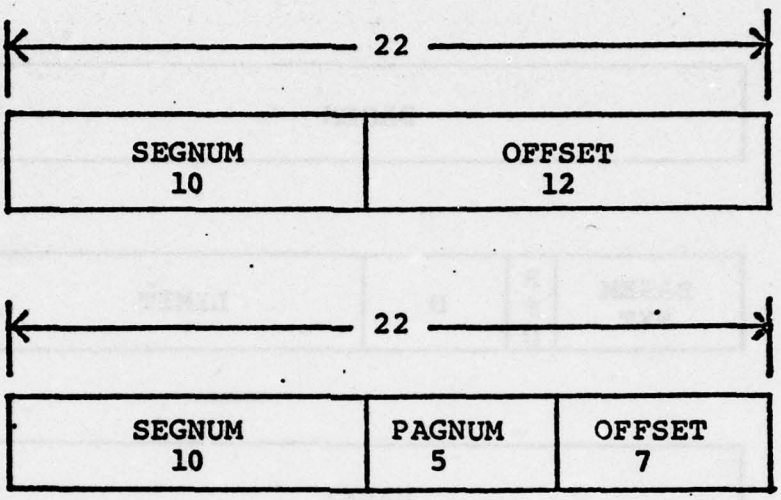


Figure 15

Virtual Address

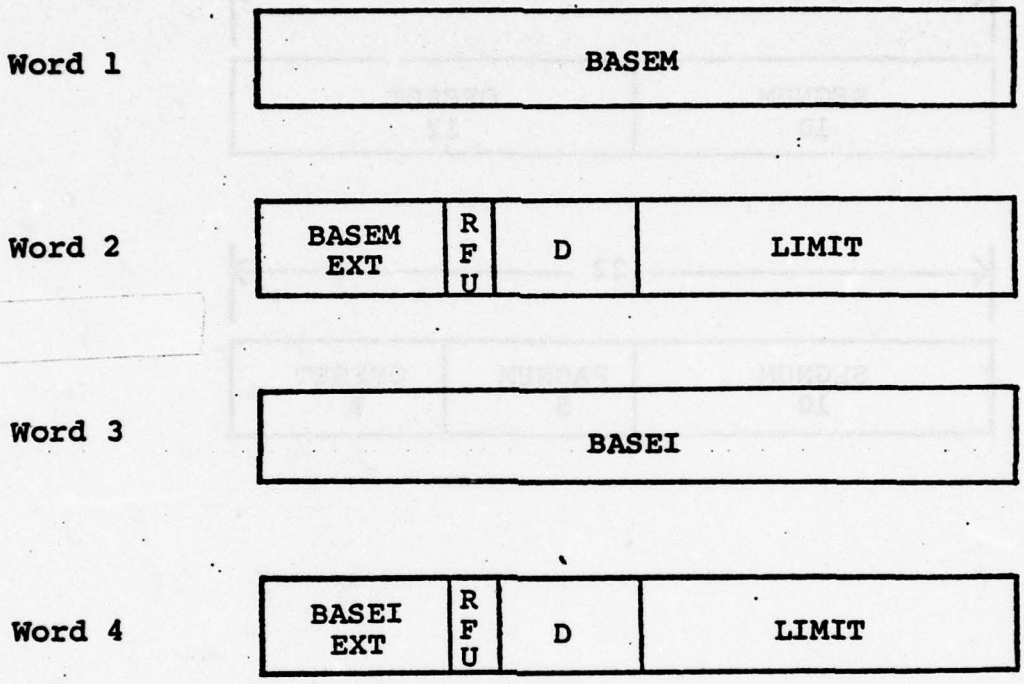
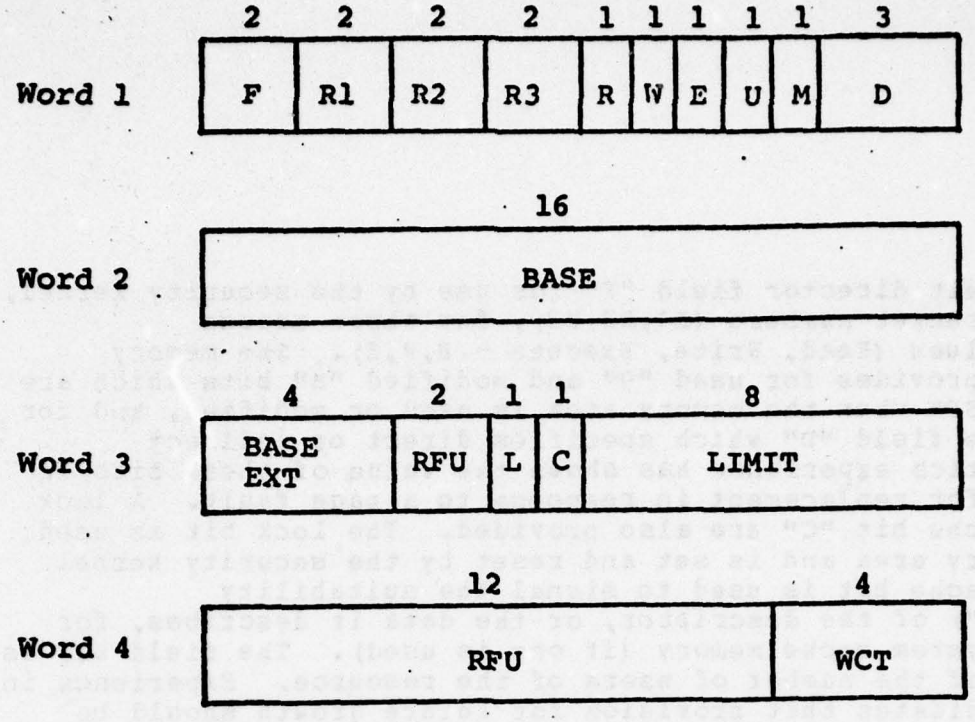
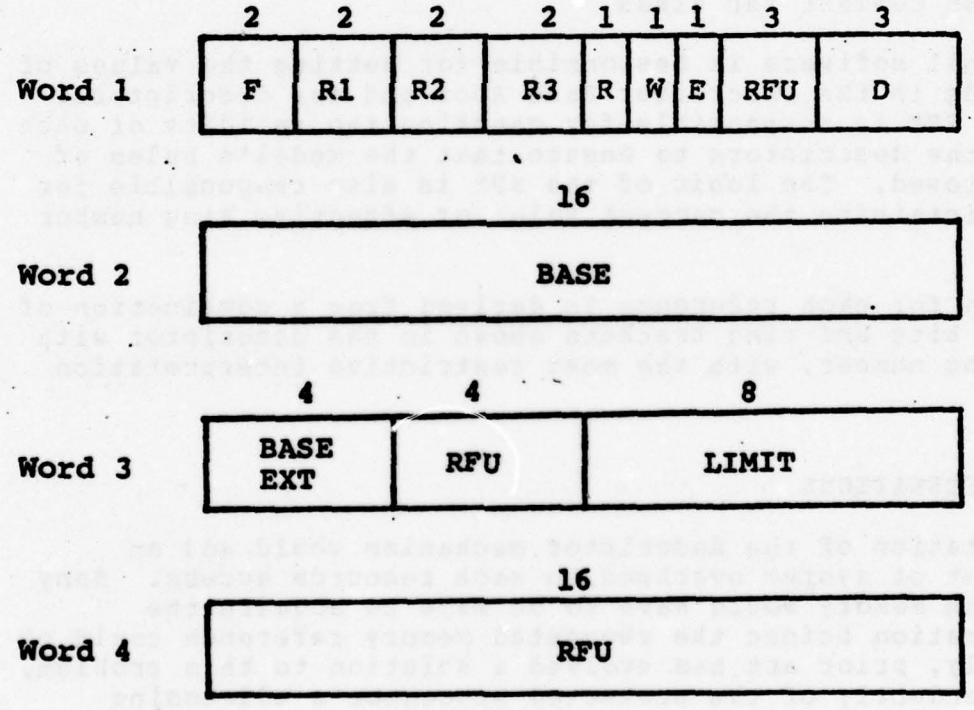


Figure 16

DBR Layout



Memory Descriptor Format



IO Descriptor Format

Figure 17

Descriptor Formats

is made for a fault director field "F" for use by the security kernel, for three ring bracket numbers (R1,R2,R3), for three access authorization values (Read, Write, Execute - R,W,E). The memory descriptor also provides for used "U" and modified "M" bits which are set "on" by the SPM when the memory area is used or modified, and for a descriptor type field "D" which specifies direct or indirect descriptor. Multics experience has shown the value of these bits in selecting pages for replacement in response to a page fault. A lock bit "L" and a cache bit "C" are also provided. The lock bit is used to lock the memory area and is set and reset by the security kernel software. The cache bit is used to signal the suitability ("encacheability") of the descriptor, or the data it describes, for inclusion in a system cache memory (if one is used). The field WCT is used as a count of the number of users of the resource. Experience in system design indicates that provision for future growth should be made. The base address (BASE) and its extension (BASE EXT) and size limit (LIMIT) fields are provided.

SUPPORT FOR ACCESS CONTROL AND RINGS

The security kernel software is responsible for setting the values of the various fields in the Descriptor Base Root and the descriptors. The logic of the SPM is responsible for checking the validity of each reference using the descriptors to ensure that the model's rules of security are followed. The logic of the SPM is also responsible for computing and maintaining the current value of effective ring number for the process.

Access permission for each reference is derived from a combination of the R,E,W access bits and ring brackets shown in the descriptor with the effective ring number, with the most restrictive interpretation controlling.

PERFORMANCE CONSIDERATIONS

A naive implementation of the descriptor mechanism would add an intolerable amount of system overhead to each resource access. Many references to main memory would have to be made to acquire the descriptor information before the requested memory reference could be made. Fortunately, prior art has evolved a solution to this problem, applicable independently of the protected processor's addressing structure. This is the use of an associative memory cache to hold recently used descriptor information. On each reference, the contents of the cache are interrogated first. If the desired information is in the cache, it is used directly. If the information is not in the cache, it is automatically acquired from memory without software

intervention, added to the cache, and used for the reference. Implementation techniques are many and well known.

Use of an associative memory adds complexity to the SPM but holds the performance impact to acceptable levels. The loss of performance, due to introduction of the SPM, may be in the range of 5% to 25% for typical communications applications and common protected processor architectures. The descriptor cache must have an operation time that is fast relative to the cycle time of the main memory in order to realize the full benefits of its use. If the protected processor has a sophisticated system organization and already uses the finest currently available technology, any protected system must accept the higher (25%) range of relative performance loss. The obvious source of comparative performance data would be between the 6080 and the 6180. Because of the totally different software and modes of applications of the two machines, no satisfactory performance comparison studies of these machines have been published. The estimates of 25% here are studied engineering judgements based on Multics and other experience with minicomputers with address translation hardware.

SUPPORT OF MULTIPLE PROCESSORS

There are at least three motivations for multiple processor configurations. First, it is to be expected that some sites in a communications network may experience frequent peak load situations in which a single processor is unable to keep up with the workload. This is particularly true if the site's processor system is satisfying more than one requirement such as serving as a node of a communications network and also serving as a front-end processor to a local host tied to the communications network, or is itself performing some of the functions of a local host. In such a case, it is advantageous to have a physical configuration in which the workload may use memory and processing facilities wherever available. Second, sites supporting large numbers of terminals, memories, and peripheral equipment may run into electrical and logical limitations of a single processor bus system's connection potential, and require a means of permitting a terminal on one bus system to work with a peripheral device which happens to be connected to another physical bus system. Third, and most important, communications and similar real time applications must exhibit graceful degradation of performance in presence of hardware failure; no single hardware failure should make any substantial portion of the system's functions unusable. This is especially important with a military system whose successful operation is most important at times when it may be under greatest physical stress due to explosive shocks, electrical shock, radiation, etc.

These requirements can be satisfied if a physical module is available which permits interconnection of similar protected miniprocessor systems, and another similar module is available to permit connection of a protected miniprocessor to a host processor. For high reliability, the interconrection modules should permit confiquring a system with multiple paths available between each pair of modules - whether processor or peripheral device or memory - of the system. An example of such a configuration, interconnecting systems of the type described in Figure 18. The similar processors are tied together in a "delta" formation by modules referred to as Inter-System Links (ISL); the host system is tied in via Host System Interconnection Units (HSIU). Communications lines, backup storage, and other peripherals providing information uniquely available from a single source, or accepting information uniquely destined for a single sink, may be connected to their appropriate peripheral control units (BDC or DC in the figure) by multiple paths available to the controllers. Devices for which only generic access to some device is required, such as one of several line printers or terminals at the same physical site, may be connected to a single controller. Well known reliability theory will guide this choice.

The ISL units will perform a static mapping of addresses between busses so that each bus occupies a unique area of the physical space (possibly having been expanded for the multi-processor configuration.) This simple mapping allows each processor to refer to the local bus physical address space independent of the configuration, yet allows each processor complete access to all facilities of the total system.

Use of multiprocessor systems makes control of the contents of the individual system cache memories vital. Provision must be made to allow intersystem signaling of the alteration or invalidation of descriptors. The system must also provide efficient means of discovering that a particular processor's SPM may have a copy of a particular descriptor. The presence of multiple physical processors requires additional consideration of interprocess communication's requirement for write only path description. (This may be considered similar to an I/O process initiation.) The SPM functions listed below will address these requirements.

SUPPORT OF INPUT/OUTPUT

Assignment of Input/Output devices to a process essentially extends the address space available to the process to include the I/O device. The control of this extension has proven to be one of the most difficult areas in the design of secure systems. In part, the problem is due to the wide variation in peripheral device characteristics and in part due to the unpredictably long times involved in peripheral

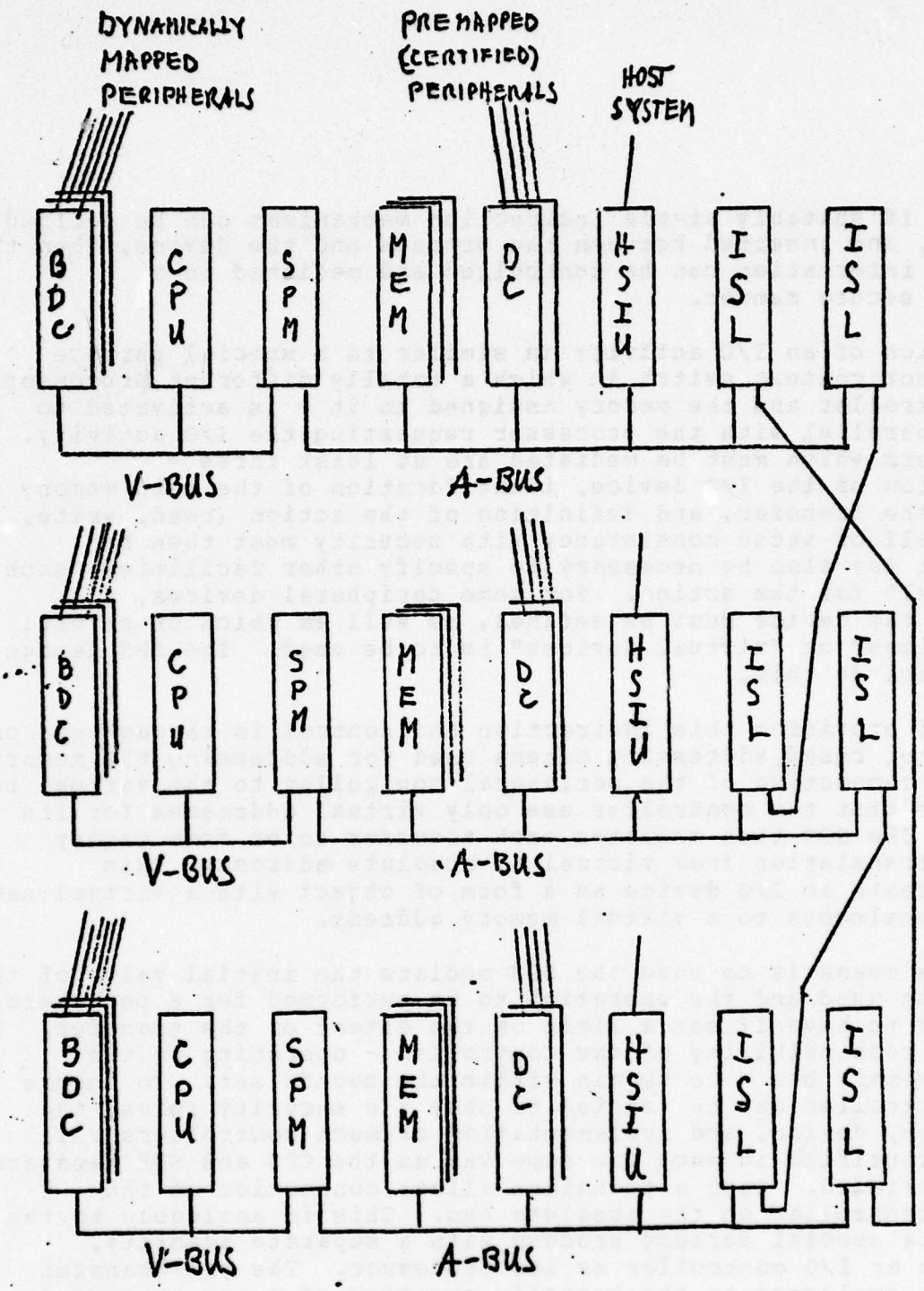


Figure 18

Multiprocessor System

transfers. If suitably simple indirection mechanisms can be defined, implemented, and inserted between the process and the device, then the transfer of information can be controlled and mediated in a certifiably secure manner.

The initiation of an I/O activity is similar to a special purpose multiprocessor context switch in which a totally different processor - the I/O controller and the memory assigned to it - is activated to operate in parallel with the processor requesting the I/O activity. The parameters which must be mediated are at least three - identification of the I/O device, identification of the main memory to be used in the transfer, and definition of the action (read, write, control) - all of whose consistency with security must then be checked. It may also be necessary to specify other facilities, such as a "channel" for the action. For some peripheral devices, a location on the device must be defined, as well as which of several "pseudo-devices" or "virtual devices" is to be used. The SPM design must not preclude this.

One means of providing this indirection and control is through use of the descriptor based addressing scheme used for addressing the memory. This allows connection of the peripheral controller to the virtual bus and provides that the controller use only virtual addresses for its transfers. The SPM then mediates each transfer to or from memory during the translation from virtual to absolute address. This technique treats an I/O device as a form of object with a virtual name completely analogous to a virtual memory address.

An alternate means is to have the SPM mediate the initial value of the address to be used and the operation to be performed for a peripheral transfer and to have it set a limit on the extent of the transfer. It is then the responsibility of the controller - operating on the "absolute" memory bus - to remain within the bounds set. To insure that the controller can be trusted to obey the security rules, the specification, design, and implementation of such controllers will have to be certified in much the same way as the CPU and SPM hardware is to be certified. Such a technique allows connection of the peripheral controller on the absolute bus. This is analogous to the creation of a special purpose process with a separate identity, operating on an I/O controller as its processor. The pre-transfer mediation is analogous to the kernel's creation of a new process in that the kernel must insure that the process is initially in a secure state and has only valid permissions. The initiation, control and termination of an I/O transfer is analogous to the initiation, scheduling and termination of a normal process.

Two methods of handling peripheral device names and physical addresses have evolved. One, the "explicit names" approach, associates a

virtual name with each physical device available to the system (and with each virtual device as well). The other, the "devices as memory" approach, associates a set of virtual memory locations with each device. Either of these techniques may be used with the descriptor based addressing scheme used by the SPM; either is consistent with security actions of any existing peripheral device.

Termination of an I/O transfer results in an interrupt to the system. This initiates an automatic context switch according to properties of the protected processor, as augmented by the SPM to ensure security. These actions will be the same as considered above in Section IV's overview of secure context switching requirements.

SUPPORT FOR THE INITIAL SECURE STATE

The fundamental security models deal with security preservation - the problem of keeping the system in a secure state assuming that it was initially in a secure state. One of the problems for the system designer is to establish the initial secure state of the system. Systems are traditionally started up by use of a bootstrap routine or initial program load procedure of some sort. It is essential to the security of the Secure Communications Processor that a means be provided to establish this initial state with great confidence.

Both the SPM and the security kernel (in ring 0) must be initialized to a secure state. It is suggested that as much of this initial state as practical be established by use of a Read-Only Memory (ROM) unit placed within the SPM. Physical protection of the initializing function may be provided by use of a key lock, combination lock, or some other such device to prevent unauthorized activation of the secure initialization procedure. The initialization will also use the secure context switch and other facilities to create a new process, as mentioned in IV above.

SECTION V

REQUIREMENTS FOR THE PROTECTED SYSTEM

INTRODUCTION

This section will consider design requirements for the components of the system used to build a Secure Communications Processor. First the control requirements of the SPM will be discussed and listed. Then the functional requirements for the CPU, Memory, and Peripheral Controllers of the protected system will be covered. Finally, some of the functional requirements of the associative or cache memory for the SPM will be considered.

SPM INSTRUCTIONS

There will be a set of instructions or functions used by the security kernel in controlling the SPM. These instructions must be executed, in part, by the CPU that is supporting the security kernel process. These instructions must be restricted to operate only when the CPU is in Ring 0 so that only the security kernel may issue them. Most activities of the security kernel will not require special instructions or supporting actions of the SPM. For example, the creation or modification of a descriptor may be done by the security kernel without special aid from the SPM. The SPM will be involved in such an action, but only to the extent that it is involved in any reference to memory by any process.

The minimum set of SPM control instructions is outlined below. The designer of the system may find that some additional instructions are desirable or he may choose to implement these instructions in different ways, depending on the nature of the system he is working with.

1. Clear Associative Memories

This instruction will effectively clear or invalidate the contents of the specified associative (cache) memory of the SPM, removing descriptor information as specified. The logic of the SPM will handle loading the contents of the cache memory as part of its address mediation activity. This command gives the security kernel control over the removal of such information from the memory.

At least two variables are needed for this command to allow specification of the extent of the command. The variables are:

a. SPM Address; The command must be able to operate on all SPM's in the system or just on the SPM associated with the processor that issued the command.

b. Descriptor Address; Clear all descriptors from the addressed SPM or remove those descriptors specified.

The ability to remove some specified subset of all descriptors must be implemented in a fail safe manner, leaving only those descriptors which cannot possibly be "SIMILAR to" any descriptor(s) specified, where "SIMILAR" is fail safe, and otherwise defined for convenience of implementation. This facility will be a necessary time saver if SPM descriptor storage is large and the system has several multiprocessors; it may be unnecessary if analysis proves that a small descriptor storage in the SPM is adequate.

2. Furnish Unique Identifier

This instruction or function will cause the SPM to generate a unique value usable to identify a process or segment. This instruction could be performed instead by software in the security kernel, if there can be provision for reliable storage of the generating parameters. The requirement is for an unique process_id with no possibility of duplication over a long time span exceeding expected system life, even if the system is stopped and powered down repeatedly.

The Multics system contains a hardware clock, accessible by the CPU, which counts microseconds since Midnight (GMT) January 1, 1900. The clock will provide unambiguous counts until late in the year 2042 AD, and provides unique identifier bit strings for use in processor and message ID's. The resolution of 1 microsecond prevents two accesses of the same count. The period of 142+ years assures non-repetition. A similar counter would suffice for the Secure Communications Processor.

3. Save Context; Restore Context

These instructions and functions will cause the SPM to store (and restore) the registers and control information needed to specify the status of a process. Among the items to be stored are the current effective ring number, Descriptor Base Root, descriptor fault identification information, instruction and address evaluation progress ("depth") information at last fault, and various other internal security related items. The information must be stored in memory, accessible only to the security kernel running in Ring 0.

These functions will automatically augment the save and restore context instructions of the CPU, handling all items that are sensitive from a security point of view. It is essential that the user software not be able to bypass or circumvent this procedure in any way. It is vital to the performance of the system that the combined actions of the CPU and SPM operate as quickly as possible for a context switch so that system overhead will be minimized in these frequently used areas.

It is important to note that fault response processing is likely the most frequent route to the kernel. The system should make it easy to save and restore the minimum amount of information in these cases. Implementation which does not require clearing all registers of the protected processor will not only save time, but also avoid forced removal of information necessary to the kernel's functions. The inclusion of a multiple save and clear registers and the complementary load multiple registers instructions will allow the kernel to decide the disposition of a trap before completely changing context, yet efficiently exchange contexts if a domain change is actually required.

4. Test

A comprehensive set of SPM testing instructions is needed so that the security kernel may dynamically verify that the system is meeting the security rules. The test instruction set should allow testing of functions of the SPM, interconnections of the system, and major components of the system. These instructions will also be of use in diagnosis of hardware problems and in monitoring the system for hardware failure. About 5% to 10% extra SPM hardware and firmware must be allowed to implement adequate fault detection and diagnostic functions.

5. Ring Crossing

The operation of changing a process from one ring of execution to another and returning, without compromising security, may be borrowed from the Multics system. A Call instruction

Call (gate entry, procedure_name, parameters)

specifies in one of its arguments that a "gate" descriptor array is needed. The current operating ring r must satisfy the inequalities

$$R2 \leq r \leq R3 \quad \text{(gate parameter descriptor)}$$

or a call fault will be signaled. The parameters R2 and R3 are the read and call bracket values, respectively, for the current process. The new effective ring of execution r' is the most secure of the

present ring of execution, r, and the Write bracket of the descriptor used to pass the parameters to the new execution ring, namely,

new ring r' = min(r,R2 of procedure_name descriptor)

In cross ring calls, argument validation is an important consideration. It is important that the called routine (at a higher privilege) access no data passed as parameters that the calling program cannot access. Likewise upon return, the calling program must not be passed data to which it has no access rights. Call by value arguments present no problem since the SPM mediates their access prior to the call in the action of putting the arguments in the calling sequence. The problem comes from indirect or call by reference arguments which are not actually mediated until used and will be used at a higher privilege ring.

Several possible solutions exist. The indirection constant may carry the caller's ring and access rights established prior to the call. Then the SPM can temporarily use these to check the actual fetch of the data by the called program. This requires a special form of indirection and a special SPM mode to mediate the pointer using the proper ring and access rights. Theoretically, the called program, given access to the caller's ring and access rights, could be depended upon to check all accesses via software. This poses a significant burden by requiring the verification of all called routines and would add an intolerable performance penalty.

If multi-level indirection is available in the hardware, the caller can pass a pointer to an indirection constant in the callers ring. At actual mediation time, the SPM will mediate each indirection pointer using the Descriptor Base Root of the ring in which the pointer exists. This requires extra mediation by the SPM for each access and requires multi-level indirection in the hardware with the attendant problems associated with unbounded levels of indirection.

A third technique requires a special instruction which causes the SPM to validate the access rights without the CPU's performing the memory access. This instruction allows the kernel to have the SPM perform the mediation. If the mediation is successful, the kernel can then copy the reference into the called domain for further use without multi-level indirection.

The choice of actual method of argument validation is dependent on the design of the CPU and SPM. The designers may use one of these methods or another equivalent method, but one must be provided.

6. Automatic Data Storage - Faults and Interrupts ("Traps")

Table I summarizes information which the protected processor and the Security Protection Module together must ensure is saved whenever a Fault, Interrupt or other "trap" event occurs. The critical items from the security point of view are marked with a double asterisk (**). Table II summarizes an acceptable content of fault data words in the SPM.

7. Passage of effective rings and process_id.

When I/O operations are begun, or any other operation is performed activating another physical process to operate independently, the SPM must be sure that the effective access ring number and process_id are passed in user unalterable form. From the analytic point of view, this is merely a special case of secure process creation, but it requires care in practice.

TABLE I

FAULT & TRAP (RARE EVENT) INFORMATION

This summarizes the information stored by the protected processor and the SPM during a trap due to fault, interrupt, or other rare events.

The information to be stored in a trap frame includes at least the following:

For a processor related rare event:

*the instruction (INST)

*the effective address of this (or the next) instruction (Program Counter)

*the effective address (EA) being processed at the time

*identification of this address within the instruction (1st,2nd, etc)

*the indirection level, including each descriptor or indirect address as a new indirection level;

*cause: data, processing, address, or description;

**execution ring, user_id

*other user visible trap defining information;

**security related trap defining information

*whether an automatic processor context save was performed by this trap;

*if context saved, pointer to its location;

*where the next trap frame may be stored, if necessary

*where the previous trap frame was stored, if required

For a peripheral related rare event (interrupt):

The instruction, operands, and other particulars relating to the instruction's effective address being executed at the time of the interrupt are irrelevant. Interruption may be delayed until the termination of the next instruction at which an interrupt is acceptable. Instead of these, there must be

* identification of the peripheral device responsible for the interrupt;
* identification of the reason for the interrupt;
* status of the peripheral device at time of the interrupt;
* status of its peripheral transfer process at time of the interrupt.

TABLE II

SECURITY RELATED FAULT INFORMATION
fault data word content

The following information is stored in the two words reserved for SPM fault data collection:

Word 1:	Bit 0:	Hardware fault
	Bit 1:	Hardware fault
	Bit 2:	Illegal user_id
	Bit 3:	Illegal user_id
	Bit 4:	Ring > 0, Ring 0 address
	Bit 5:	Ring > 0, Ring 0 privileged instruction
	Bit 6:	Reserved for Future Use
	Bit 7:	RFU
	Bit 8:	Directed fault
	Bit 9:	Directed fault
	Bit 10:	Limit exceeded (no space)
	Bit 11:	Limit exceeded (no space)
	Bit 12:	Read fault
	Bit 13:	Execute fault
	Bit 14:	Write fault
	Bit 15:	Call fault
Word 2:	Bit 16:	Permission missing
	Bit 17:	Permission missing
	Bit 18:	Bracket improper
	Bit 19:	Bracket improper
	Bit 20:	CALL gate required
	Bit 21:	RFU
	Bit 22:	RFU
	Bit 23:	Indirection depth exceeded (if not in 26-31)
	Bit 24:	Multiprocess usage limit exceeded
	Bit 25:	Multiprocess semaphore access fault
	Bits 26-31:	reserved for depth indicators

(Details of the format and content of this list are illustrative. The actual usage and definitions will depend on actual hardware and software implementation).

PROCESSOR REALIZATION REQUIREMENTS

Compatibility with communications requires a processor capable of efficiently handling bytes. Maximum efficiency combined with maximum availability of commercially modifiable peripheral and memory modules suggests that two bytes be handled in parallel. Maintenance of maximum potential compatibility with existing equipments suggests that 8 bit bytes are a must, that 6 bit bytes may be a frequently required subset, and that 9 bit bytes should not be precluded.

Parity checking is a must if early recognition of hardware failure is to be achieved, and other types of hardware checking should be used wherever available and mandatorily provided for obviously security critical circuitry. The need for creating operation codes reserved for the reference monitor's use alone implies a processor with a vocabulary of unused "generic" or "unassigned" or "user assignable" operation formats.

A processor connected with its peripheral and memory modules by a standard type of interface for a large set of existing equipments is a must. The bus design, moreover, must permit asynchronous operation, to tolerate the delays required in implementation of protection checking.

A fast context switching facility is required, if performance bottlenecks found with a previous study (Reference 7) are to be avoided. Means of identifying, in an unforgeable manner, which module, and which subunit within a module, is requesting access to the bus, must be present or easily providable. All private storage, including any cache locations, must be capable of complete erasure by external action, to prevent leakage of information between processes in context switching. A real time clock is needed to ensure periodic entry to the reference monitor; it also assists the generation of unique identifications not repeatable in the system's life.

These features may be summarized as:

* byte compatibility
* two byte data path width
* 9 bit byte potential
* parity checking

* other checking

* standard bus for intermodule communication

* asynchronous operation

* fast context switching

* potential for unforgeable module and submodule identification

* reserved operation codes

* available peripherals for communications, disks, tapes, printers, and a large variety of other standard equipments compatible with present government usages.

* real time clock

* interruption turn off

* uninterruptible increment/decrement or test and set (for semaphores)

Required Certified Operations

The kernel will require a subset of instructions verified to do exactly what they are specified to do, and a means of using these to change each component of the maps. This subset should be kept as small as possible to permit verification. The suggested set is summarized:

Load to working register(s) from memory.
Store to memory from working register(s).
(Notation: [x] ≡ contents of, "op" ≡ operation,
EA ≡ "Effective Address",
WR ≡ "Working Register", IND ≡ branch indicators.)
Arithmetic to working registers:
[WR] op [EA] --> WR, with op = +, -, *, /, Modulo (small operands);
[EA] ± 1 --> [EA] (increment or decrement semaphore).
Logic:
[WR] op [EA] --> [WR] with op = &, |, ⊕;
[WR] :: [EA] --> IND (comparison with storage).
Branch on IND conditions; all combinations of +, -, = or equivalent indicators.
I/O: Read, write, control op or status information.

It would also be convenient, although not absolutely necessary, to take advantage of frequently used composites of these which are often available on modern minicomputers. Certified instructions to swap WR and memory, and instructions to provide loading and storing with protected masking of the memory contents (to manipulate arbitrary bit fields in descriptors) would greatly shorten the time spent in the Reference Monitor software. A means of providing the equivalent of a test and set is also required; this may be accomplished by the uninterruptible execution of a combination of other verified instructions of the subset, or by a separate verified instruction.

A certified facility is also needed to establish efficient addressability to selected descriptors of an address, rather than the data ordinarily described. There must also be a quick means of establishing efficient and certified access to fast access working space for descriptor manipulation. These may well be found inherent in practical facilities of the protected processor and its interfaces to the SPM.

REALIZATION REQUIREMENTS - MAIN MEMORY

The main memory modules for use with the Secure Communications Processor should be basically a standard commercial design, with the following properties to provide maximum utility and performance.

Error Detection and Correction

The reliability requirements are high, especially since unreliability results in a substantially greater risk of breach of security. Therefore every reasonable reliability increasing practice should be used. The provision of Error Detection and Correction (EDAC) in the memory modules is one of the most useful techniques for extending the Mean Time Between Failure. An increase of the order of a factor of six between necessary repairs, for the same probability of failure per unit time, is expected on one typical design. Moreover, this technique gives warning of potential failure long before the failure is actually experienced in most cases. Hence error detection and correction is strongly recommended for a Secure Communications Processor system.

Read-Alter-Write Memory Operation

There are a number of circumstances in descriptor manipulation and secure certifiable queueing in which it is mandatory that information be removed from the main memory, and returned, in altered form,

without the possibility of another user - such as a peripheral processor or another processor of a multiprocessor system - being able to access the same resource in the meantime. Manipulation of the U, M, C, F, WCT, I fields of descriptors are cases in point. This uninterruptible manipulation is also important in incrementing or decrementing semaphores (Dijkstra's "P" and "V" operations) for properly secure queueing and dispatching (processor assignment). This facility must be built into the memory module for maximum efficiency and usability.

Module Identification

Security requires that each module connected to the absolute, or certified, bus, be able to furnish an unforgeable identification of itself at each access. The memory module is particularly important in this regard, especially when it is responding to requests for service made some time previously.

Rapid Standardization of Memory Contents

A secure system operates on the assumption that it may "go down" at any time. In order to prevent leakage of information from one user to another, it will clear each section of memory to a standard value, such as zero, immediately upon each release of facilities, without waiting for new quantities to be assigned and written over the released memory. There are memory modules in which large parts of the memory can be zeroed or otherwise returned to an uninformative standard value with great speed. Such memory modules should be carefully considered since all unassigned memory should be cleared.

REALIZATION REQUIREMENTS: PERIPHERAL CONTROLLERS AND DEVICES

The major requirement is to provide for operation of all the peripheral devices and controllers needed. These include all common types - especially communications controllers.

The controllers themselves must be certifiable to perform exactly as requested in at least the following ways:

* responding to the correct controller identification, and no other;
* selecting the proper device, and no other;
* selecting the proper location on the device (where appropriate) and no other;

* performing the proper action with the device, and no other. In particular, no confusion must be made possible between read and write, and between a request for data action and a request for control action.

If the controller is to operate on a bus shared with the memory modules - an "absolute" or "certified" bus - the following certified correct properties are also necessary:

* correct acceptance of absolute addresses furnished to be used in peripheral data transfer or other control usage;

* correct manipulation of absolute addresses furnished for use in peripheral data transfer, or control usage;

* correct termination of a data transfer at or before the point where assigned memory space has been exceeded;

* correct presentation of a request for "write" or "read" to the memory bus;

* requesting action of the bus only when previously instructed to begin an action requiring it.

Each controller must, in any case, either furnish, or be modified to furnish, an unforgeable unique identification of itself and the device in use to accompany each request for service on the bus. Controllers of devices which typically must be shared among users, such as disks or drums, should be designed or connectable to permit definition of pseudo-devices comprising arbitrarily selected combinations of locations on the device.

CACHE MEMORY REALIZATION CONSIDERATIONS

There are three principal design decisions in the implementation of the cache. The first decision is the method of operation. The cache will be made of a fast access memory, of course, but there are two principal design choices. One is to use a true associative memory, in which any quantity may be stored in any location and recovered. Such memories provide a higher "hit" ratio - that is, probability of finding a quantity desired already in the cache - per word stored, because there are fewer restrictions in selecting a location to be rewritten when a new quantity must be referenced. On the other hand, they tend to be slower and somewhat more expensive. The other alternative is to design a "pseudo-associative" cache of standard fast access memory, using locations within the cache to perform part of the association by choosing a subset of the bits of the associated

quantity to address them, and using comparison logic to perform the remainder of the association.

Cache Size

Another choice in implementation is the size of the cache. Previous work has suggested the maximum possibly useful descriptor storage to be of the order of 512 descriptor positions, and that a number of the order of 64 to 128 may be adequate in practice (Reference 20). If pure associative realization is used, the greater flexibility of the replacement choices with a genuine associator may make as few as sixteen per user a highly effective choice. (10) Since it is expected that there may be frequent switches to the Reference Monitor software, it is very advantageous to design the cache to be large enough, and with an associating algorithm so chosen, that the reference monitor and the system code's most recently used descriptors are very likely not to use the same space as the user's descriptors. This can be done, for example, by providing at least sixteen descriptor storages per ring, and, if a pseudo-associator is used, connecting the bit(s) of the associative address which discriminate(s) between the reference monitor system code and user code, to be part of the bits wired to select cache locations directly, rather than part of the bits wired to the comparison logic. For an explanation of operation of one specific pseudo-associative cache see Reference 33, pages 7-1 to 7-4.

Descriptor Bits Not Mandatorily Represented in the Cache

A third implementation consideration is which fields of the descriptor need be explicitly represented in the cache. Those which represent information of use only in main memory need not be so represented. The following bits of Figure 13 must be represented explicitly:

R1,R2,R3
R,E,W,M
Base location
Base location extension
D (type)
Limit

---

(10) Multics experiments (Reference 20) indicated a small but measurable difference between a 16-associator and an 8-associator descriptor storage.

Other bits need not necessarily be represented within the fast access descriptor cache storage, but may be represented if other design considerations dictate. The bits describing whether the descriptor has been used (U), is encacheable (C), has been locked into memory (L), or how many users have "wired" it (WCT), are all needed only in algorithms involving main memory communication between processes and processors. These bits and the unused bits may be eliminated from the cache storage if economical. However, their main memory representations are consulted and changed by SPM firmware algorithms.

It is noteworthy that explicit representation of the first two words of the descriptor in cache prepares the way for a very efficient data cache design as well. Explicit representation of the first three words of the recommended descriptor format creates a storage format also useful for I/O descriptor storage if the limit and base fields are extended to describe single words during the I/O data transfers.

In addition to the encachement of descriptors as considered above, other information may be included in a cache. A data cache increases the effective cycle speed of main memory. A data cache in the SPM, added to a processor which possesses none, may conveniently recover much performance cost of the protection.

The DBR (Descriptor Base Root) might also be encached. A small conventionally indexed storage system of perhaps 16 words would probably suffice. Since the DBR storage is expected to be consulted infrequently, it is not important that all of its contents be available in parallel. It may also be stored in a general descriptor cache location.

Details of the cache implementation must be the responsibility of the implementor; these guidelines may help. The best economical approximation to an optimum replacement algorithm is also a function of this cache implementation.

SECTION VI

IMPLEMENTATION CONSIDERATIONS

INTRODUCTION

This section is concerned with a number of implementation areas. First, there are some estimates of the physical size of a practical SPM, using different system structures. This leads to some design recommendations and tradeoff discussion. Next, the subject of system reliability is considered. The estimate of the risk per unit time of security breach due to hardware failure is discussed. An outline is presented of the analysis to estimate security breach per unit time on a practical system. The section ends with an attempt to predict the performance of a protected system used as a Secure Communications Processor.

The principal implementation conclusions can be summarized as follows:

* The SPM can be built for the approximate cost of a bare miniprocessor without memory or peripheral equipment or console. This is a small fraction - approximately 10% or less - of a typical minicomputer system's hardware cost.

* Implementation using cache memory techniques and a variable size segment descriptor rather than a page table is preferable for system growth and flexibility of application. Moreover, this technique costs no more than others, either in performance or in hardware.

* Installing an SPM in most minicomputer systems will imply a performance loss of approximately 5% to 25%. This is a small fraction of the ratio of system performance ordinarily expected between a particular miniprocessor model and the next most costly compatible model of the same manufacturer's line.

* While pre-existing software may be used on the protected system without change if the operating system and kernel software are properly designed, guidelines can be furnished to increase the efficiency of common system or application programs in a secured environment. This is particularly true in a multiprocessor environment, which is preferred for high reliability.

* Implementation using standard minicomputer boards, subsystems, components, and bus organization is practical.

* A useful model of the probability per unit time that hardware failure causes a security breach can be developed in terms of well known characteristics of any miniprocessor's subsystems - The Mean Time Between Failure (MTBF), and the logical organization.

* An upper limit on the rate of security breach due to hardware failure can be specified. It appears practical to obtain the order of a million system hours of operation between expected hardware caused security breach.

ESTIMATES OF SIZE AND COST

Estimates of relative size and cost of a special purpose processor and peripherals vs. a special purpose security unit conclude that the descriptor processing facility will be of the order of magnitude of a present day miniprocessor, requiring at least 500 integrated circuit packages ("IC's") using current technology, to manufacture both the processor and its necessary security supporting equipment. This agrees with the Multics processor design experience, where the descriptor manipulating equipment required the equivalent of another arithmetic unit - about 25% of the processor subsystem.

With the packaging techniques available today, about three minicomputer boards will be necessary to hold all of the integrated circuitry needed to implement a special purpose security unit on a single level of construction. However, because of the large amount of circuitry required, the interconnection problems encourage additional effort to provide more compact packaging.

Implementation size estimates conclude the following:

1. The SPM might possibly be built on a single minicomputer board if it used a simple page table, limited in the number of pages it supported.

2. The SPM, with very compact packaging, might be built on a single board if it used a descriptor arranged associative system, and very careful attention is given to sharing a memory array between I/O descriptor support and processor descriptor support.

3. It is safe to predict that the SPM, with associative descriptor design, could be built on two boards.

The relative cost and usefulness of different designs of protection modules have been estimated by outlining trial logical designs, and estimating the amount of equipment it would take to realize them. Highly accurate cost estimates were not expected; good relative costs of different approaches were sought to guide system architectural decisions. Three basic types of protection units were considered.

(a) a simple page table, with the simplest possible types of protection indications added to each fixed size page;

(b) augmentation of the simple page table to obtain a fairly elaborate descriptor, stored in a fast access memory. A limit field, and some space reserved for software usage, multiprocessor controls, and other unforeseeable detail, was added to create a descriptor of four words.

(c) augmentation of the descriptor to one requiring much more elaborate storage space. The descriptor was expanded to eight words.

The relative sizes are shown in Figure 19.

It was found that there was an irreducible minimum of logic and firmware which varied relatively little from the minimum to the maximum designs considered. Without any storage elements, the simple page table scheme was estimated to require about 200 integrated circuits (MSI), while the support for the 8 word descriptor requires a minimum of 322 MSI circuits. It requires about 40 extra IC's to get a practical unit with the minimum page table approach, and about 270 extra IC's to get a practically viable unit with the maximum capability 8 word descriptor. The most elaborate unit, including a general purpose cache, requires about 720 IC's.

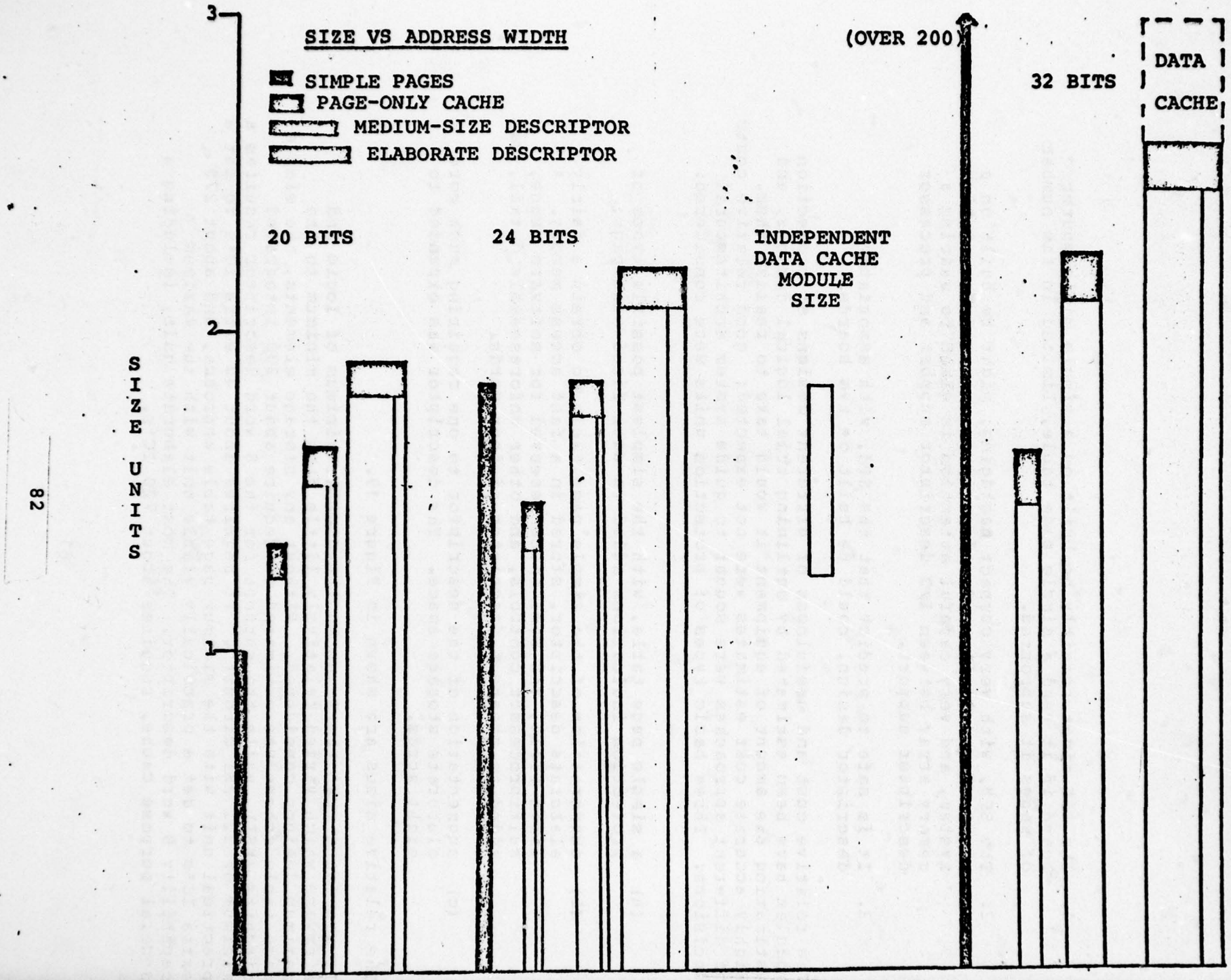

SIZE VS ADDRESS WIDTH
SIMPLE PAGES
PAGE-ONLY CACHE
MEDIUM-SIZE DESCRIPTOR
ELABORATE DESCRIPTOR
20 BITS
24 BITS
INDEPENDENT DATA CACHE MODULE SIZE
(OVER 200)
32 BITS
DATA CACHE
SIZE UNITS
3
2
1

Figure 19

Relative cost and usefulness was estimated for the following alternative design choices:

* Segment size: Fixed maximum length vs. fixed fraction of memory
* Memory used: Full descriptor table vs. cache
* Type of entry: simple page table vs. segment descriptors
* Maximum size describable per entry: single vs. multiple

The use of a single fixed size page table per process with fixed size pages (similar to IBM 370) provides a much simpler hardware implementation and rapidity of access to the page descriptions at each use. However, this simplicity of hardware implementation depends upon choosing the full page table memory, which introduces a substantial performance penalty each time the context of the processor's usage is shifted from one process (user level) to another. Designs to reduce this penalty depend either upon a novel memory technology, or upon a larger number of slower integrated circuit memory packages; either approach seems technically unwise. Therefore some hardware complexity should be accepted and the choice should be for variable size segments. The fastest page table approach also requires the page size to remain a constant fraction of the size of the virtual or physical memory managed in order to maintain the hardware simplicity, and thus undesirably constrains all software designs. Utilizing the constant descriptor size approach, the size of the descriptor table memory must increase in proportion to the size of the virtual or absolute memory managed. The extra circuit delays imply performance penalties and physical layout complications, as well as higher costs. The simple descriptor table's advantages thus appeared chimerical.

The cache memory approach risks performance loss in interrogating the cache and then fetching data into the cache which is not present and must be extracted from memory. This also implies logic and firmware to implement a replacement algorithm. However, published performance of data caches indicated that "hit ratios" (probability that desired information was present) exceeding 90% could be expected with practical designs. Multics experiments also indicate that a very small descriptor cache was more than adequate for obtaining a high "hit ratio." See Reference 20.

The fact that the system is to be used for communications makes it desirable to define independent protection for relatively small segments of memory used as communications buffers, while real time procedure characteristics suggest that large amounts of permanently resident code might usefully be described and protected by a single

descriptor. If only large maximum size segments can be described, a small entity such as a communications buffer, a parameter list, or a procedure link table, can be independently protected only by sacrificing part of the user's virtual space. If a small maximum is chosen, more memory must be given to the page descriptor tables, and the performance is reduced when page tables are changed in context switching. The theoretically optimum size for a 512K byte memory (adequate for SATIN-IV) is near 1024 bytes - small for the procedure and large for the buffers. Furthermore, if the fixed page size are chosen, physical as well as virtual memory would be wasted, requiring either more time lost in procedure swapping or else a larger memory. This favors the multiple size ability in alternative 4 above.

The above interrelated items must be considered together before any recommendation can be chosen. After considering the impact of each item upon the others, the most useful design would use a variable size segment description ability, with a descriptor cache realization, and preferably the ability to use at least two qualitatively different maximum segment sizes.

TRADE-OFF NOTES

These size and cost considerations result in a few design guiding tradeoff rules. The most useful may be summarized:

Hardware Cost vs System Growth Potential

1. Increasing virtual address support by 1-4 bits costs only 10-22 integrated circuits (IC's) in cache like descriptor storage, while storage costs double for each additional bit in non cache implementations. Therefore, increasing virtual address support from 256 large to 4096 small pages increase storage cost by about 15 times in a page table implementation.

2. For a 20-bit virtual address, a simple page table storage uses the least circuitry; for a 24-bit virtual address a cache storage uses the least circuitry. The differences involved a small fraction of a minicomputer board. Moreover, at 24-bits of virtual address support, the medium scale descriptor unit and the simple page table have the same cost: less than two minicomputer boards. This suggests that descriptor and cache techniques are not inherently too costly or complicated to use, and that the additional system growth potential of the cache comes at low hardware cost.

Size and Layout Limits

1. There seems no practical way to achieve the design goals without at least one minicomputer board's worth of circuitry.

2. It is difficult to visualize a layout of the interconnections needed which would not imply that the protection module would grow past a single board. Physical layout permitting, two adjacent boards seems more graceful.

3. The probability of constructing the protecting module on two boards or less, with current technology, seems high, assuming the protected processor and peripheral units could provide the necessary information for control.

Performance Trade-off Notes

1. Cache usage of fast access storage (100 - 200 nsec access time) is slower than providing a simple page table, but not but more than one logic level propagation time (about 10-15 nsec) per use, for virtual addresses of at least 20 bits.

2. The performance penalty for increasing the size of the virtual address is far less for cache than for page table storage.

3. At 24-bit virtual addresses, there appears no substantial difference in cache vs. page table performance.

For these reasons it was decided that

(a) a cache like unit should be recommended as the most generally useful design, and

(b) it should strive to contain a medium scale descriptor. There is only a small size penalty if a larger descriptor were required.

RELIABILITY CONSIDERATIONS

This section addresses three considerations related to the reliability with which security can be maintained in the event of hardware failure. They are:

1. What are useful models of the relationships between hardware failure and security breach?

2. What practical algorithms may be used to make realistically helpful quantitative predictions of the security of a practical system?

3. What principles should be followed in the architecture, implementation, and configuration, in order to minimize the risk of hardware caused security breach?

First, let us consider a model for estimating limits on probability per unit time of security breach due to hardware failure. Gross, conservative, order of magnitude estimates of probability of security breach may be made by assuming that any hardware failure in the system leads to a potential breach of security. In this upper limit estimating technique we require estimates of the probability per unit time of 1) hardware failure; 2) undetected error due to hardware failure, and 3) security breach due to undetected error. Probabilities per unit time are usually characterized by their reciprocals:

1. The Mean Time Between Failure (MTBF(i)) for each component (i) of the system, from which the entire system MTBF can be derived by

$$1/MTBF = 1/MTBF(1) + 1/MTBF(2) + \ldots 1/MTBF(n)$$

2. The processing activity which may take place before presence of a failure produces a detected error, the Mean Processing Before Error detection (MPBE); with continuous system operation, the Mean Time Before Error detection (MTBE) is equivalent.

3. The probability that a given fault in processing results in a security breach.

The MTBF may be estimated for most physical designs using current technology. The MTBF is a much more difficult measure, particularly in the presence of a possible malicious attempt to penetrate the system. In fact, no complete evaluation can possibly be made until the final design is complete. Adequate design guiding estimates of MTBE due to hardware failure can be made relative to the MTBF and the system configuration by estimating the expectation of undetected errors per occurrence of a failure.

It is reasonable to assume, for any particular implementation, that the presence of a hardware error converts a proper information value to an incorrect value, with probability p(d) that it is recognizably incorrect. The information presented on successive usages of the failed hardware may often be assumed sufficiently uncorrelated that successive trials may be regarded as nearly statistically independent. Then the expectation of undetected errors per path usage can be proven to be bounded by

$$(1 - p(d))/p(d).$$

The expectation of undetected error per unit processing time, 1/MTBE, is then related to the probability of failure per unit processing time, 1/MTBF, by

$$1/MTBE \leq (1 - p(d))/p(d)\,MTBF \qquad \text{(Eqn. 6-1)}$$

where in practice it is expected that $(1 - p(d)) \ll 1/2$, so that the undetected error rate is substantially less than the failure rate. This provides an upper limit estimate of the probability per unit time that hardware failure causes an undetected change to an invalid security state.

The probability of detecting an error may usefully be divided into a component f(^r) independent of the information transmitted, and a component p(r) which is a function of the information. (For example, single bit failures in a packet of information checked by a parity circuit are always detected; multiple bit errors are detected with various probabilities depending on the original information.) From

$$p(d) = f(\hat{}r) + (1 - f(\hat{}r))p(r),$$

it can be seen that p(d) is high if either f(^r) or p(r) is high. Attention to practical design details can provide both.

For example, a high f(^r) may be found in transmitting a single byte and parity bit, because single bit errors are several times more common than multi-bit errors. Moreover, power failure to all circuits transmitting a 9 bit parity checked byte transmits the invalid value 00000000,0 or the value 11111111,1 - which may be made unlikely or impossible by appropriate system design. (Thus, the most significant byte of a virtual address may be required to be other than 11111111 with the sacrifice of only 1/256 of the address space.)

Similarly an EDAC memory check system in which every bit transmitted participates in more than one parity identity, such as Hamming-related Single Error Correcting Double Error Detecting (SEC-DED) code, increases p(r) from 0.5 at least to 0.9375.

Implementing a set of paths covered by a single check, such as the individual bits of a byte and its parity, to use different integrated circuits, serviced by different power, clocking, and other service leads provides even greater probability of early detection of failures. For s independent circuits involved in a checking identity with probability p(d) that a failure in any one of them is detectable,

$$1/MTBE = (1/MTBF)(1-p(d))^{**}s/(1-(1-p(d))^{**}s)). \qquad \text{(Eqn. 6-2)}$$

Since (1 - p(d)) << 1/2 in practice, even a small s such as 2, 3, or 4 produces a remarkable lowering of the undetected error rate for a given p(d), as Table III shows.

Practical conditions may rule out universal application of this principle, but it may be specified as a design goal for the implementor.

TABLE III
Reliability Function for Independent Fault Detection

		X**s/(1-X**s)			
p(d)	X=(1-p(d))	s=1	s=2	s=3	s=4
0.6	0.4	0.667	0.1904	0.0685	0.0263
0.75	0.25	0.333	0.066	0.016	0.0039
0.8	0.2	0.25	0.0416	0.008	0.0016
0.9	0.1	0.11	0.0101	0.001	0.0001

Upper Bound, Probability of Breach of Security

We can estimate maximum probability of breach of security as a function of expected failure rate, and details of system implementation. Numerical values of an upper bound on the probability of security breach per unit time may be estimated by

$$1/MTBB = PS/(1 + P)(1 + S)MTBE \qquad \text{(Eqn. 6-3)}$$

where S is the ratio of system time spent in manipulating security related information to system time spent in processor and memory manipulations in an equivalent unprotected computer system, and P is the ratio of the amount of logic and supporting equipment in the protection subsystem to that in the unprotected system. The ratio

(S/(1 + S)) represents the probability that a hardware fault first affects security related information, rather than data or procedure. Presumably, using either gives the hardware error detecting redundancy an equal opportunity to indicate the presence of trouble which will cause the system to be stopped, or reconfigured and deflected to test and diagnostic actions.

Estimate of the MTBF p(d) factor for various integrated circuits, connections, and subsystems may be done by statistically valid simulated operation of each subsystem with simulated introduction of a (randomly selected) permanent failure in the hardware. It could also be done by analysis of performance of similar circuits with standard 'fault analysis' algorithms well known in the literature. (11)

A better method, usable only when a logic design is complete, estimates probability of security breach by full system logic analysis and simulation (12) methods used for generating hardware logic diagnostic tests. An efficient and thorough set of such tests may be used as a base for the generation of others which will test the system more thoroughly than any practical detailed test design.

The best method is provision of appropriate redundancy checks, and extensive subsystem MTBB analysis during the design phase. Expected ranges of MTBF for practical systems in the minicomputer range, using modern technologies, should be in the thousands of hours, and that for the subsystems of individual logic paths, germane in the above context, should be in the tens or hundreds of thousands of hours. It is best to design as many subsystems as possible to provide an estimated MTBE which satisfies MAXUSEMTBE in

$$\text{MAXUSEMTBE} \geq (\text{Ns})(\text{LIFE})/(\text{RISK}), \text{ where}$$

Ns = the number of systems expected to be in use, and

LIFE = the expected life of a system, and

RISK = the risk it is reasonable to assume acceptable for a security breach on any one of the Ns systems during their installed life.

As an example, consider a practical system of twelve boards of minicomputer electronics and an MTBF of 8000 hours. Adding to this

---

(11) c.f. Symposia on Fault-Tolerant Computing, FTC-1 to FTC-5 inclusive, published 1971 to 1975 by IEEE, New York.

(12) ibid.

an SPM of approximately 1-1/2 boards of logic changes the MTBF to

$$MTBF \approx [12/(12 + 1.5)][8000] \approx 7100 \text{ hours}$$

Then Equation 6-1 may be used to estimate that MTBE exceeds

$$MTBE \geq 7100\ [0.9/0.1] = 63{,}900 \text{ hours}$$

for the entire system.

In operation, the system carries security related (absolute) addressing information on the bus about 30% of the time any bus is in use. Thus $(S/(1+S)) = 0.3$ and $P/(1+P) = (1.5/(13.5)) = 1/9$, and

$$MTBB \approx 9/0.3(MTBE) = 30(63{,}900) \approx 1{,}920{,}000 \text{ hours.}$$

This technique must be applied to the designed system for an adequate combination of subsystems, using median, minimum, and maximum estimates for MTBF, p(d), and the ratio $S2/(1+S)(1+P)$. The resulting distribution of estimates gives a measure of the confidence. For greater assurance, a reasonable distribution may be assigned each subsystem's estimate, and MonteCarlo calculation of the entire system's MTBB estimate made.

Table IV outlines a minimum set of analyses required for a practical system. Where the MTBB estimate is too small, checking circuitry must be introduced into the design. The redundancy is the only way to increase p(d) so that the system MTBB may be adequate. There will be certain circuits critical to security for which this necessity is to be expected.

.sr space 3

TABLE IV

Subsystems in Analysis	Security-Related Processing
1. Outside SPM	
Address bus & checks and memory module address circuitry.	Absolute only, iff new address out of user's space
Address control	Absolute only, iff SA1 failure, (time out if SA0)
Data bus & checks	Only when passing descriptor parts, critical state information, or absolute device identification, and classes of errors as on A-bus
Data Control	SA1 failure, either A-bus or V-bus
Other bus controls	To be analyzed
Interrupt network priority resolver	Only for SPM security fault condition being transformed to other fault condition
Timing information	If withheld from kernel, or if "unique-name" generated from clock be repeated.
Power and ground	To be analyzed
2. Inside SPM	
Associator identifying descriptor	Only for false "hit" indication
Descriptor permission information: permission checking logic and storage	Only for false extension of permission
Address within descriptor	Iff altered base is out of user's space
Limit within descriptor	Iff limit be effectively increased, and overlaps another user's resource.

Current user_id, operating ring	Always potential breach

DESIGN RECOMMENDATIONS - RELIABILITY

To minimize the probability of security breach per unit time, it is recommended that the system implementation both of hardware and software make use of the following principles:

1. The hardware should make extensive use of redundancy coding wherever practical. All memories and busses should use parity and/or Single Error Correcting, Double Error Detecting Hamming code, or the equivalent, to increase the MTBF. This particularly includes parity in the descriptor cache, and on address and data busses.

2. To keep f(^r) and p(r) high in the address circuitry, which is security critical, hardware and software should be implemented to use the maximum internal virtual address space supportable by the internal physical address bus, even if the smaller size of the physical memory associated with a given practical system makes it tempting to wire some address leads as "always zero" both in hardware and in software. Note that software should assign virtual internal addresses with similar probability throughout the physically supportable address space, rather than starting at the same small quantity for all allocations. Sacrificing a small fraction of the virtual address potential for the sake of increased f(^r) is also a valuable opportunity to decrease security breach probability.

3. The MTBF of the security critical circuitry and firmware should be made very high. If the implementor's estimated MTBF for a critical subsystem does not exceed desired MTBB, he must either

   (a) provide redundant logic, raising p(d) to detect inconsistency in time to prevent damage to the system, or else

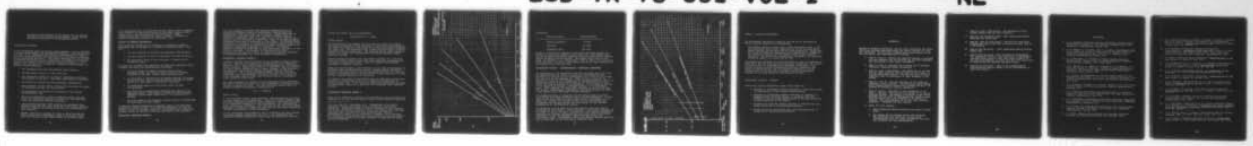
   (b) describe the frequency f of diagnostic error checks which must be introduced during system operation to guarantee that

$$1/f(\text{MTBF}) \leq \text{risk}$$

AD-A055 164 HONEYWELL INFORMATION SYSTEMS INC MCLEAN VA FEDERAL --ETC F/G 9/2
ANALYSIS OF SECURE COMMUNICATIONS PROCESSOR ARCHITECTURE. VOLUM--ETC(U)
NOV 75 J GILSON, J MEKOTA F19628-74-C-0205

UNCLASSIFIED ESD-TR-76-351-VOL-1 NL

2 OF 2
AD
A055164



END
DATE
FILMED
7-78
DDC

for these critical parts of the system, for the desired maximum risk of hardware caused security breach between successful operations of the diagnostic checks.

PERFORMANCE ESTIMATES

In the foreseeable state of the art, there appears no way to implement a secure system without some performance penalties. Some mention has been made of the implementation effects on performance, and some of our design recommendations have been made with the intent of avoiding severe performance penalties. This section discusses the size of the performance penalties which may practically be expected for a system using an SPM, compared with an ordinary unsecured system. It is assumed that the SPM is to be added to a system of approximately the following characteristics:

* The processor meets the requirements listed earlier.
* The main memory cycle is 1 usec ±250 nsec.
* The unprotected system has no data or instruction cache not controllable by the SPM. (The SPM's must manage all cache's.)
* The system is prepared for multiprocessor installation.
* Each processor of the system, except those connected to appear as peripheral devices, has its own SPM.
* All peripheral controllers are connected to the virtual ("uncertified") bus.
* There is no buffering of data or addresses within the SPM, except for descriptor working registers, and the descriptor cache and peripheral descriptor storage.
* Direct Memory Accesses (DMA) required by peripheral data transfers comprise less than 10% of the total memory accesses. (This is suggested by extensive metering and simulations of many systems.)
* Context switching is assumed to occur as often as once per 1000 to 2000 memory references, based on Schiller's MITRE PDP/11/45 security kernel design study, as an upper limit.

It is understood that the reference monitor software may be designed to be resident in every user's space at all times, requiring less effort than a full context switch for activation. These considerations will be ignored at first in favor of obtaining a grossly conservative estimate of performance loss.

Sources of Reduced Performance

Three principal causes may be discerned for inherently reduced performance due to presence of a Security Protection Module. They are:

* the time required to generate and manipulate descriptors;
* the time required to place a descriptor in cache to use it;
* the additional delay in each reference to memory due to descriptor cache.

To these must be added and subtracted the following components whose presence and extent are implementation dependent:

* the extra "skew" or "synch" delay in some two bus implementations, with certain types of logic and memory circuitry in the SPM; this is also associated with
* the time lost by distributing peripherals between two busses rather than one. The bus time is usually less than a linear function of the number of peripherals connected;
* the duplicated parts of a bus cycle, under the same circumstances;
* the time lost by descriptors contending for places in the descriptor cache. This factor is cache size and algorithm dependent; sound design should keep it to a fraction of a percent.
* the time saved if the necessary request storage is implemented to serve also as a data cache;

If multiple central processors are present, there is an additional increment to each of these. To a first approximation, it will be assumed this is in the same proportion for protected and unprotected systems, except as expressly indicated otherwise in the analysis.

Performance Reduction Source 1

The time required to generate and manipulate descriptors is an application dependent source of performance reduction believed substantially less than either of the others. Descriptors must be generated or manipulated only when a physical resource must be temporarily reassigned (page in or page out) or when a process' or use(r)'s work is initiated or terminated. Most of the work is memory management which must be performed in some manner. In communications particularly, the real time constraints frequently require much of the procedure to be resident in memory at all times, further reducing the demand for descriptor generation or alteration. No adequate quantitative model has been made of this performance increment, to the writer's knowledge. The presence of descriptors and their manipulations may even be a plus, by providing convenient discipline and hardware assistance for these necessary manipulations.

Performance Reduction Source 2

A process residency will be defined as the processing between events - such as a wait for I/O transfer, or assignment of a new peripheral resource - which cause enough delay to make dispatch of a new process highly probable. Assume three words of a descriptor must be placed in the cache before referencing can be done, although a fixed page size subdescriptor type might be defined requiring only two words of transmission. With a Read-Alter-Write memory, setting the U(sed), M(odified), and C(encacheable) bits takes no extra time, except for write references made on other than the first use of a descriptor; these require an extra cycle.

Let p(DW) represent the fraction of the descriptors first referenced for writing after their first use, and k represent the control time to initiate a descriptor fetch. The lost descriptor loading time in a process residency referencing n descriptors is

$$(3 + k + p(DW))n$$

memory cycles.

In an m-processor system, some descriptor modifications must be noted by other use(r)s of that descriptor. They will receive a message from the Clear Associative Memories (CAM) function to invalidate all copies of any descriptor "SIMILAR" to the one modified - by a conveniently built criterion of SIMILARity. The time lost is not only that assumed necessary to create the descriptor, but also the time spent refetching "SIMILAR" descriptors in other processors.

If p(D) represents the probability that a descriptor has been deleted from cache between residencies, and E(S) is the expectation per process of encached descriptors which match the SIMILARity criterion

of CAM, the extra time is approximately

$$p(D)mE(S)(3 + k + p(DW))$$

memory cycles.

The definition of the symbols p(D) and E(S) is a great deal easier than their calculation, which is heavily a function of the details of the SIMILARity algorithm, the number and nature of processes competing for space, etc. The confidence that useful operation exists is again provided by Multics experience. The proposed system should expect to operate with fewer page displacements per machine language program step than Multics.

There is reason to suspect that the product $p(D)E(S) \leq 1$ with many likely SIMILARity functions and with some reasonable assumptions for the order of magnitude of p(D). Figure 20 graphs this estimated performance reduction component.

Figure 20 also provides a rough estimate of the cost of increasing the expected number of descriptors used per thousand memory references. For estimating convenience, the "extra cycles" scale is ten times the percent performance loss in memory limited system operation, if there are one thousand references per residency.

In practice the number of descriptors used per 1000 memory references will tend to remain less than 12-14, without tremendous care in program design. (13) A low number of descriptors referenced per usage is particularly important in designing the frequently used reference monitor.

Performance Reduction Source 3

Delay in the descriptor evaluating and permission circuit checking was estimated by a trial logical design, using typical currently available

---

(13) c.f. Ref. 20 - Schroeder, M.D. "Performance of the GE-645 associative memory while Multics is in operation", for Multics experience. Practical studies in design of "based" computer systems such as the IBM 360/370, Honeywell 62 and 2000 series, etc. have also indicated that 8-16 bases are very convenient for defining addressability needed in the locality of a program, with less than five percent of the program's time spent managing its base registers. Careful program design is required, however, if less than six bases are to be utilized.

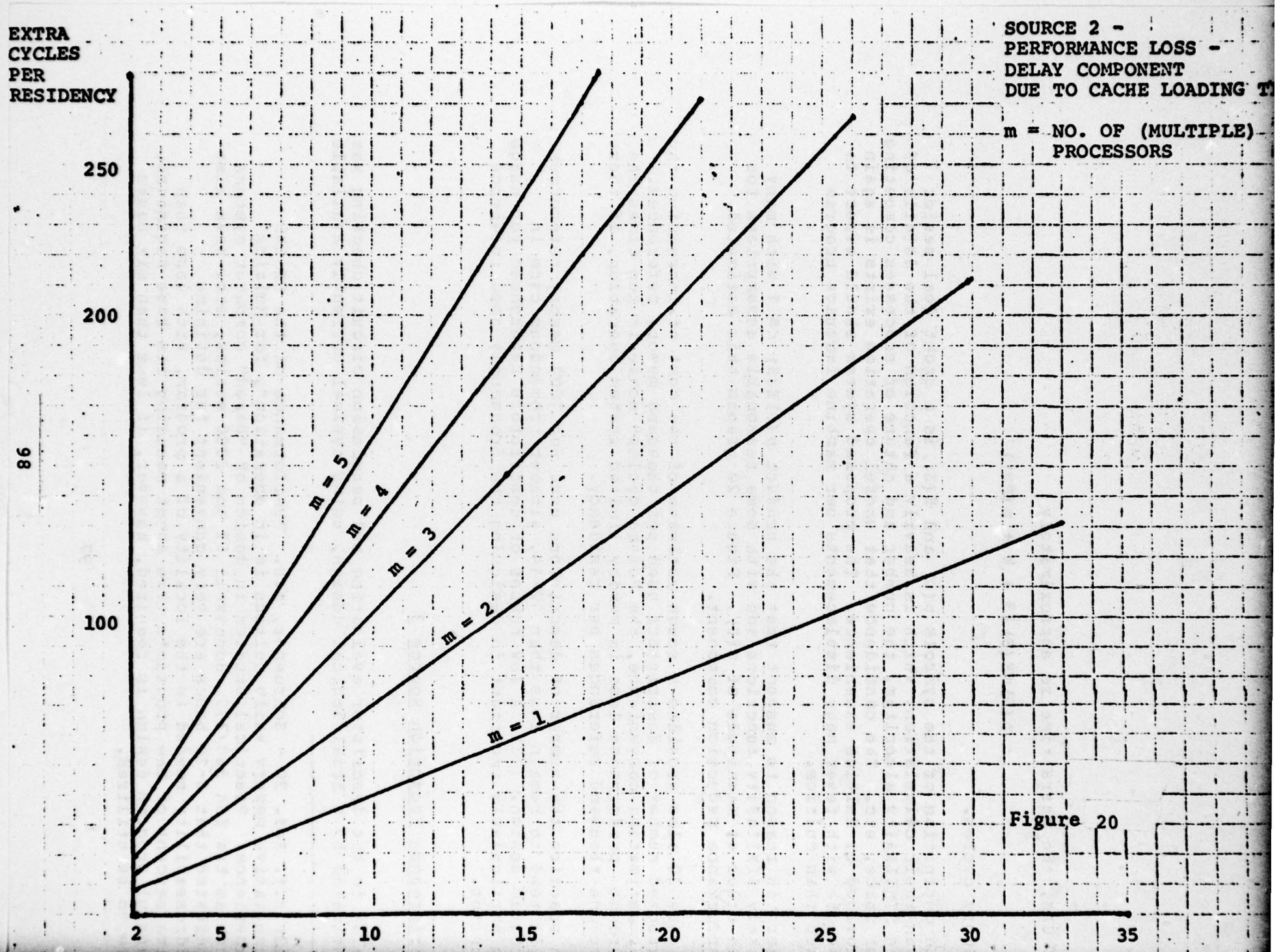

EXTRA CYCLES PER RESIDENCY
250
200
100
m = 5
m = 4
m = 3
m = 2
m = 1
2
5
10
15
20
25
30
35
SOURCE 2 -
PERFORMANCE LOSS -
DELAY COMPONENT
DUE TO CACHE LOADING T
m = NO. OF (MULTIPLE)
PROCESSORS

Figure 20

components.

Nature of Design	Estimated Delay
Highly optimistic	75 nsec
Realistic	125 nsec
Reasonable Goal	100 nsec

Figure 21 displays the percent of time added from this source, as a function of the memory module's peak access rate (horizontal scale) and the percentage of system operation which is memory access limited (choice of three vertical scales). For memory cycle times in the range of 1000nsec, and a typical 60% to 80% of system time limited by memory accesses, the memory performance loss due to this source could be expected to be less than 10%.

Performance Changes - Implementation - Dependent Components

The separation of the peripherals into two different groups in a two bus implementation is likely to imply that the resynching time required on a parallel bus is experienced twice - once on the virtual bus, and once on the absolute bus. This may not be true if the associative logic can be cascaded without resynchronization, and/or if special anticipative logic interfaces can be provided for all bus users, but this is unlikely to be practical. It is likely to be true if the memory does require a resynchronization at its addressing input. This component of delay might typically be an extra 25 nsec with present technologies, representing 2.5% extra system time with a 1 usec memory and a 100% memory limited operation.

If resynchronization is required, the SPM cannot begin its action until almost an entire virtual bus cycle has been completed. It must wait for the last element of skew delay on the V-Bus before it can either (a) initiate the actions on the V-Bus to acknowledge the information and release the bus for other usage, or (b) initiate consultation of the descriptor memory.

The situation is much worse if the bus response time also must be duplicated on both busses. This introduces a non-trivial component of the order of 100 to 150 nsec. Fortunately, a bus system organization requiring bus response duplication may well also imply the graceful opportunity to use the necessary request storage as data and instruction cache. References to this extra general purpose cache may save a great part of the performance otherwise lost.

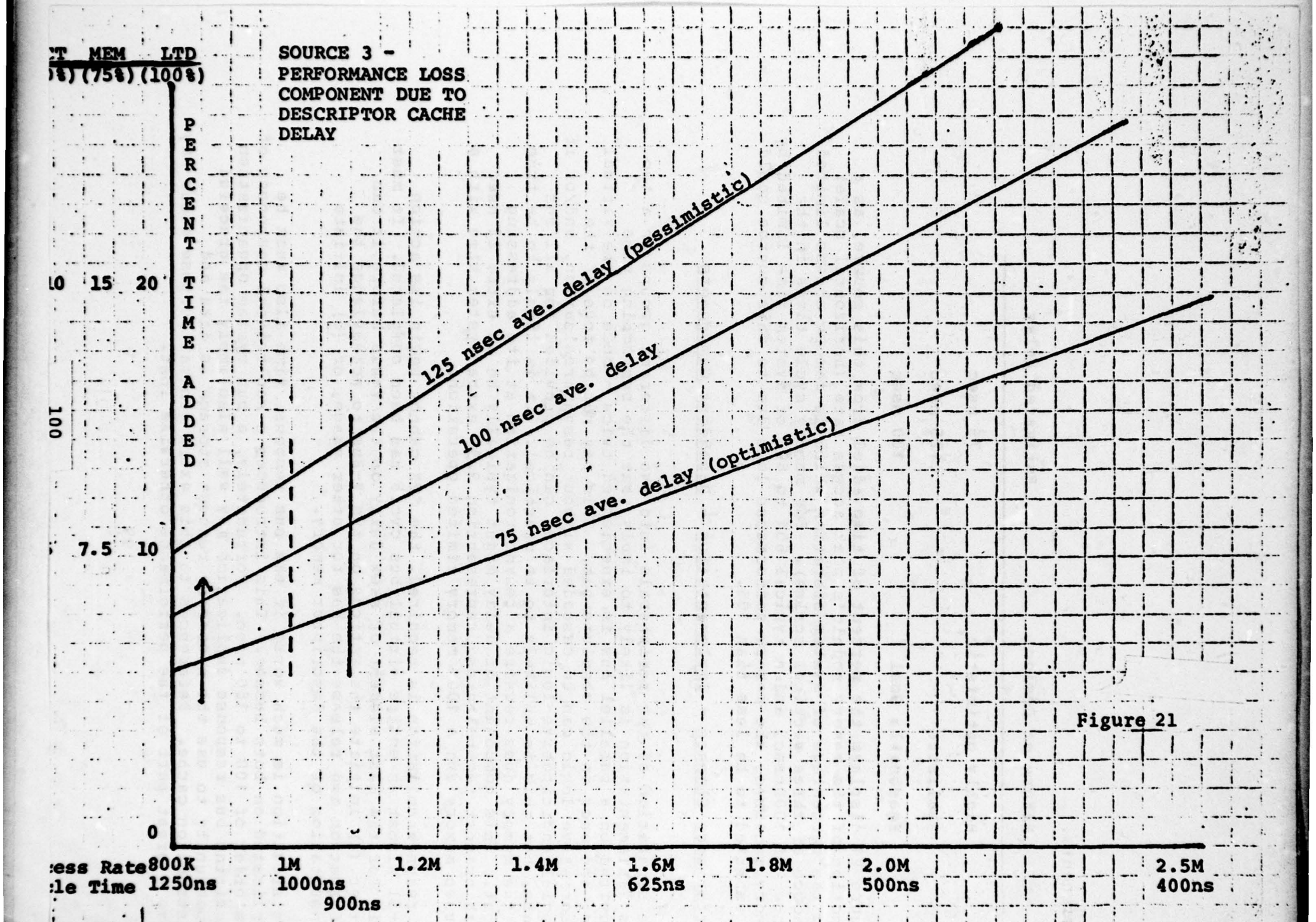

SOURCE 3 -
PERFORMANCE LOSS
COMPONENT DUE TO
DESCRIPTOR CACHE
DELAY
PERCENT TIME ADDED
125 nsec ave. delay (pessimistic)
100 nsec ave. delay
75 nsec ave. delay (optimistic)
0
7.5
10
15
20
800K
1M
1.2M
1.4M
1.6M
1.8M
2.0M
2.5M
1250ns
1000ns
900ns
625ns
500ns
400ns

Figure 21

SUMMARY - EXPECTED PERFORMANCE

The performance degradation caused by the SPM may be estimated by adding expected components of degradation.

For example, assume a system with a 1000 nsec memory cycle, a 100 nsec descriptor cache delay, approximately 75% of its time spent memory limited, and not more than two multiprocessors. Assume also that about 10 different descriptors are expected to be referenced per processor residency of 1000 memory cycles between context switching. The total system degradation, compared to an equivalent unsecured system, is of the order of 17%.

Experimentation indicates it is very difficult to set up circumstances which would lead to more than 25% performance degradation with likely assumptions for the "Implementation dependent" components. More precise estimates require implementation details, especially the amount of overlap between SPM and processor operations, and the relationships of the most used programs' working set reference patterns to the organization of the caches in the SPM.

PERFORMANCE ANALYSIS - SUMMARY

These notes provide estimating guides, which serve to:

* Indicate to programmers useful trade-offs in using many versus few descriptors in their program organizations.
* Indicate to implementors useful trade-offs in reducing the effective delay by clever design - a challenge when the designer of the protected miniprocessor is already attempting to push technology to its utmost.
* Establish the need for further analysis to describe the best design trade-offs in multiprocessor systems.
* Provide practical indications that the performance cost of protection is economically acceptable.

APPENDIX A

Because of funding limitations, the Air Force terminated the effort which this document describes before the effort reached its logical conclusion. The Air Force comments which were present at the time the effort was terminated are as follows:

1. Page 9, line 11 - Multics was selected because it provided the best hardware and its software was the most adaptable and not because it was "least vulnerable."

2. Page 13, line 1 - How were the criteria for a "properly organized" segmented memory determined?

3. Page 15, line 7 from bottom - The comparison of the SPM and the ideal reference monitor based on the figures is a gross oversimplification. The SPM functions are more important than its interfaces to other elements in the system.

4. Page 21, line 16 - The last sentence of the paragraph presents a faulty dilemma. One multilevel secure system could replace several single level systems and still not provide an information sharing capability.

5. Page 33, line 4 - The point of this paragraph is not clear. Function x should not be in kernel if it was not security related. The example illustrates a case where programs for one level of abstraction may reside in more than one domain and is not necessarily a conflict as claimed. The ring mechanism is not the only isolation mechanism. Processes can also isolate.

6. Pages 36 to 44, General -

   a. Many unexplained design decisions are found on these pages.

   b. The discussion of software levels was confusing. The reader may have difficulty at each level distinguishing what was being implemented from what was available for the implementation.

7. Page 50, line 4 from bottom - The definition of processes differs from the one on page 42.

8. Page 53, and following pages - More unexplained design decisions are found here.

9. Page 65, line 16 from bottom - The Multics clock which records time from 1 Jan 1900 to late in the year 2042 is a red herring.

10. Page 66 and following - More unexplained design choices appear here.

11. Page 72 and following - Many realization requirements are unsupported (nine bit byte potential), questionable (the SPM could provide a real time clock as easily as the base minicomputer) or wrong (the PDP 11/45 could have an SPM but has few reserved operation codes).

12. Page 55 and following - Many of the implementation considerations seem to assume a 16 bit minicomputer or even a HIS Level 6.

REFERENCES

1. J. P. Anderson, "Computer Security Technology Planning Study," ESD-TR-73-51, Volume I and II, James P. Anderson & Co., Fort Washington, Pennsylvania, October 1972.

2. R. R. Schell, P. J. Downey, and G. J. Popek, "Preliminary Notes on the Design of Secure Military Computer Systems," MCI-73-1, Electronic Systems Division (AFSC), L. G. Hanscom Field, Bedford, Massachusetts, January 1973.

3. D. E. Bell and L. J. LaPadula, "Secure Computer Systems," ESD-TR-73-278, Volume I-III, The MITRE Corporation, Bedford, Massachusetts, November 1973 - June 1974.

4. K. G. Walter, W. F. Ogden, W. C. Rounds F. T., Bradshaw, S. R. Ames, Jr., and D. G. Shumway,, "Primitive Models for Computer Security," ESD-TR-74-117, Case Western Reserve University, Cleveland, Ohio, January 1974.

5. W. P. Price, "Implications of a Virtual Memory Mechanism for Implementing Protection in a Family of Operating Systems," Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania, June 1973.

6. W. L. Schiller, "Design of a Security Kernel for the PDP-11/45," ESD-TR-73-294, The MITRE Corporation, Bedford, Massachusetts, December 1973.

7. W. L. Schiller, "The Design and Specification of a Security Kernel for the PDP-11/45," ESD-TR-75-69, The MITRE Corporation, Bedford, Massachusetts, May 1975.

8. M. D. Schroeder, "Cooperation of Mutually Suspicious Subsystems in a Computer Utility," MAC-TR-104, MIT Project MAC, Cambridge, Massachusetts, September 1972.

9. D. D. Clark, "An Input/Output Architecture for Virtual Memory Computer Systems," MAC-TR-117, MIT Project MAC, Cambridge, Massachusetts, January 1974.

10. D. Redell, "Naming and Protection in Extendible Operating Systems," MAC TR-140, MIT Project MAC, October, 1974.

11. W. A. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and S. F. Pollack, "HYDRA: The Kernel of a Multiprocessor Operating System," Communications of the ACM, Volume 17, Number 6, June 1974, pp. 337-345.

12. R. M. Needham, "Protection Systems and Protection Implementation," Proceedings AFIPS 1972 FJCC, Volume 41, AFIPS Press, Montvale, New Jersey, pp.571-576. Also Project MAC Protection Symposium, MIT, Cambridge, Massachusetts, 1974.

13. R. Fabry, "Capability-Based Addressing," Communications of the ACM, Volume 17, Number 7, July 1974.

14. M. D. Schroeder and J. H. Saltzer, "A Hardware Architecture for Implementing Protection Rings," Communications of the ACM, Volume 15, Number 3, March 1972, pp. 157-170.

15. E. I. Organick, The Multics System: An Examination of Its Structure, MIT Press, Cambridge, Massachusetts, 1972.

16. E. W. Dijkstra, "The Humble Programmer," Communications of the ACM, Volume 15, Number 10, October 1972.

17. E. W. Dijkstra, "The Structure of the 'THE' Multiprogramming System," Communications of the ACM, Volume 11, Number 5, May 1968, pp. 341-346.

18. B. H. Liskov, "The Design of the Venus Operating System," Communications of the ACM, Volume 15, Number 3, March 1972, pp. 144-149.

19. J. H. Saltzer, "Traffic Control in a Multiplexed Computer System," MAC-TR-30 (Thesis), MIT Project MAC, Cambridge, Massachusetts, July 1966.

20. M. D. Schroeder, "Performance of the GE-645 Associative Memory while Multics is in Operation," Proceedings ACM SIGOPS Workshop on System Performance Evaluation, Harvard University, April 1971, pp. 227-245

21. K. R. Kaplar, and R. O. Winder, "Cache-Based Computer Systems," Computer, IEEE, New York, New York, March 1973, p30.

22. P. W. Lampson, "Dynamic Protection Structures," Proceedings AFIPS Volume 35, AFIPS Press,Montvale, New Jersey, pp. 27-38.

23. R. C. Daley and J. B. Dennis, "Virtual Memory, Process and Sharing in MULTICS," Communications of the ACM, Volume 11, Number 5, May 1968, pages 306-312.

24. L. Smith, "Architectures for Secure Computer Systems," ESD-TR-75-51, The MITRE Corporation, Bedford, Massachusetts, April 1975.

25. Per Brinch Hansen, RC 4000 Software Multiprogramming System, A/S Regnecentralen, Copenhagen, February 1971.

26. T. Kilburn, et al., "One Level Storage System", IRE Transactions EC-11, Volume 2, pages 223-235. Also C. G. Bell and A. Newell, Computer Structures: Readings and Examples, McGraw-Hill, New York, 1971.

27. F. H. Sumner et al., "The Central Control Unit of the "Atlas" Computer," Proceedings IFIP Congress 1962, pages 657-662.

28. W. Lonergan and P. King, "Design of the B5000 System", Datamation, Volume 7, Number 5, May 1961, pages 28-32. Also C. G. Bell and A. Newell, Computer Readings and Examples, McGraw-Hill, New York, 1971.

29. P. J. Maher, "Problems of Storage Allocation in a Multiprocessor Multiprogrammed System", Communications of the ACM, Volume 4, Number 10, October 1961, pages 421-422.

30. L. Robinson, P. G. Neumann, K. N. Levitt, and A. R. Saxena, "On Attaining Reliable Software for a Secure Operating System," 1975 International Conference on Reliable Software, Los Angeles, California, April 1975, pp. 267-284.

31. A. N. Habermann, "On the Harmonious Co-operation of Abstract Machines," PhD thesis, Technische Hogeschool te Eindhoven, The Netherlands, 1968.

32. A. Bensoussan, C. T. Clinger, and R. C. Daley, "The Multics Virtual Memory: Concepts and Design," Communications of the ACM, Volume 15, Number 5, May 1972, pp. 308-318.

33. Digital Equipment Corporation, PDP-11/70 Processor Handbook, 1975.

34. S. M. Goheen, and R. S. Fiske, "OS/360 Computer Security Penetration Exercise," WP-4467, The MITRE Corporation October, 1972 (controlled distribution)

35. P. A. Karger and R. R. Schell, "Multics Security Evaluation: Vulnerability Analysis," ESD-TR-74-193, Volume II, Electronic Systems Divison (AFSC), L. G. Hanscom Field, Bedford, Massachusetts, June 1974.