

**PROCEEDINGS**  
**OF THE**  
**FOURTH SEMINAR**  
**ON THE**  
**DOD COMPUTER SECURITY**  
**INITIATIVE**

**NATIONAL BUREAU OF STANDARDS**  
**GAITHERSBURG, MARYLAND**

**AUGUST 10 - 12, 1981**

APPROVED FOR PUBLIC RELEASE;  
DISTRIBUTION UNLIMITED

## TABLE OF CONTENTS

	<u>Page</u>
Table of Contents	i
About the DoD Computer Security Initiative	iii
About the Seminar	iii
Acknowledgments	iv
Program	v
List of Handouts	viii
"Welcoming Address," James H. Burrows, Director, Institute for Computer Sciences and Technology, National Bureau of Standards	A-1
"Keynote Address," Admiral Bobby Inman, Deputy Director of Central Intelligence, Washington, D.C.	B-1
"Introductory Comments," Stephen T. Walker, Director, Information Systems, Office of Deputy Under Secretary of Defense (C <sup>3</sup> I)	C-1
"Burroughs' Efforts in Computer Security," Chris Tomlinson	D-1
"CR80-A Fault Tolerant Computer for Implementation in Secure Systems," Asbjørn Smitt, Christian-Rovsing A/S, Ballerup, Denmark	E-1
"Computer Security and Control Data," Terry A. Cureton, Control Data Corporation	F-1
"SAC Digital Network Security Methodology," Mauro Ferdman, The MITRE Corporation	G-1
"COS/NFE Overview," Gary Grossman, Digital Technology, Incorporated	H-1
"WIS Security Strategy," Larry Bernosky, Defense Communications Agency	I-1
"Trusted Computing Research at Data General Corporation," Leslie DeLashmutt and Doug Wells, Data General Corporation	J-1
"The iAPX-432 Microcomputer System," George Cox, Intel Corporation	K-1

"ICL Efforts in Computer Security," Tom Parker, International Computers, Limited	L-1
"GNOSIS: A Progress Report," Bob Colten, TYMSHARE	M-1
"Computer Security Evaluation Center," George Cotter, Acting Director, DoD Computer Security Evaluation Center, National Security Agency	N-1
"Trusted Computer Systems," Rein Turn, The RAND Corporation	O-1
"The SDC Communications Kernel," David L. Golber, System Development Corporation	P-1
"The MITRE Trusted Packet Switch," Chris Hisgen, The MITRE Corporation	Q-1
"Experience with KVM," Tom Hinke, System Development Corporation	R-1
"SCOMP (KSOS-6) Development Experience Update," Lester Friam, Honeywell	S-1
"KSOS-11 Summary and Update," John Woodward, The MITRE Corporation	T-1
"ACCAT and FORSCOM Guard Systems," Mike Sogleglad, Logicon	U-1
"A Security Model for a Military Message System," Carl E. Landwehr, Naval Research Laboratory	V-1
"EUCLID and Verification", Ian Griggs, I.P. Sharp & Associates, Ltd.	W-1
"The Evaluation of Three Specification and Verification Methodologies," Richard A. Platek, Digicomp Research Corporation	X-1

## DOD COMPUTER SECURITY INITIATIVE SEMINAR - IV

August 10-12, 1981

### ABOUT THE DOD COMPUTER SECURITY INITIATIVE

The Department of Defense (DoD) Computer Security Initiative was established in 1978 by the Assistant Secretary of Defense for Communications, Command, and Control and Intelligence to achieve the widespread availability of "trusted" ADP systems for use within the DoD. Widespread availability implies the use of commercially developed trusted ADP systems whenever possible. Recent DoD research activities are demonstrating that trusted ADP systems can be developed and successfully employed in sensitive information handling environments. In addition to these demonstration systems, a technically sound and consistent evaluation procedure must be established for determining the environments for which a particular trusted system is suitable.

The Computer Security Initiative is attempting to foster the development of trusted ADP systems through technology transfer efforts and to define reasonable ADP system evaluation procedures to be applied to both government and commercially developed trusted ADP systems. This seminar is the fourth in a series which constitute an essential element in the Initiative's Technology Transfer Program.

Effective January 1, 1981, the Director of the National Security Agency was assigned responsibility for computer security evaluation for the Department of Defense. Plans for the transfer of the Computer Security Initiative activities to NSA are well underway.

The Institute for Computer Sciences and Technology, through its Computer Security and Risk Management Standards program, seeks new technology to satisfy Federal ADP security requirements. The Institute then promulgates acceptable and cost effective technology in Federal Information Processing Standards and Guidelines. The Institute is pleased to assist the Department of Defense in transferring the interim results of its research being conducted under the Computer Security Initiative.

### ABOUT THE SEMINAR

This is the fourth in a series of seminars to acquaint computer system developers and users with the status of trusted ADP system developments plans for the integrity evaluation of trusted systems. The three previous seminars have stressed user requirements for trusted systems throughout the government and the private sector, experience with design of production prototype trusted systems, and industry progress in computer security. The focus of this seminar is on trusted system efforts across the board.



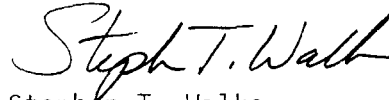
## ACKNOWLEDGMENTS

A number of people in and outside of the DoD Computer Security Technical Consortium have helped to make this seminar a success. At the MITRE Corporation, Pete Tasker helped to organize the speakers; Karen Borgeson and Dianne Mazzone managed registration; Charles McClure provided behind-the-scenes support. Finally Dr. Billy Claybrook handled the entire job of collecting and organizing the material for this Proceedings.

Also, we are grateful to Greta Pignone and Sara R. Torrence of NBS for arranging the splendid facilities.

## DISCLAIMER

The presentations in this proceedings are provided for your information. They should not be interpreted as necessarily representing the official view or carrying any endorsement, either expressed or implied, of the Department of Defense or the United States Government.



Stephen T. Walker  
Chairman  
DoD Computer Security Technical  
Consortium

DOD COMPUTER SECURITY INITIATIVE SEMINAR - IV

August 10-12, 1981

Monday, August 10

9:30 INTRODUCTION

Jim Burrows, Director  
Institute for Computer Sciences and Technology  
National Bureau of Standards

KEYNOTE SPEAKER

Admiral Bobby Inman  
Deputy Director of Central Intelligence

DOD Computer Security Initiative Status

Steve Walker, Chairman  
DoD Computer Security Technical Consortium

MANUFACTURERS' EFFORTS IN COMPUTER SECURITY

Chris Tomlinson  
Burroughs Corporation

Axel Hvidtfeldt  
Christian-Rovsing

Terry Cureton  
Control Data Corporation

2:00 ACQUISITION & DEVELOPMENT EXPERIENCE

SACDIN

Mauro Ferdman  
The MITRE Corporation

Communications Operating System/NFE

Gary Grossman  
Digital Technology Incorporated

WWMCCS INFORMATION SYSTEM COMPUTER SECURITY

Larry Bernosky  
WWMCCS System Engineering Office

Tuesday, August 11

9:15 NBS COMPUTER SECURITY EFFORTS

Dennis Branstad  
National Bureau of Standards

MANUFACTURERS' EFFORTS IN COMPUTER SECURITY (Continued)

Les DeLashmutt  
Data General Corporation

George Cox  
Intel Corporation

Tom Parker  
International Computers Limited

Bob Colten & Norm Hardy  
Tymshare

2:00 DOD COMPUTER SECURITY EVALUATION CENTER

George Cotter, Acting Director  
DOD Computer Security Evaluation Center  
National Security Agency

NON-DOD TRUSTED SYSTEM NEEDS

Rein Turn  
The Rand Corporation (Consultant)

COMMUNICATIONS EXPERIENCE

The SDC Communications Kernel

David L. Golber  
System Development Corporation

The MITRE Trusted Packet Switch

Chris Hisgen  
The MITRE Corporation

5:30 Wine & Cheese Reception  
Washingtonian Hotel  
(until 7:30 p.m.)

Wednesday, August 12

9:15           DEVELOPMENT EXPERIENCE UPDATE

KVM/370

Tom Hinke  
System Development Corporation

KSOS-6

Les Fraim  
Honeywell Information Systems

KSOS-11

John Woodward  
The MITRE Corporation

RESEARCH AND DEVELOPMENT UPDATE

ACCAT Guard & FORSCOM Security Monitor

Mike Soleglad  
Logicon

Security Model for a Military Message System

Carl Landwehr  
Naval Research Lab

2:00           RESEARCH AND DEVELOPMENT UPDATE (continued)

Euclid & Verification

Ian Griggs  
I. P. Sharp & Associates, Ltd

Evaluation of Specification & Verification Systems

Richard Platek  
Digicomp Research

WRAP-UP

LIST OF HANDOUTS

REIN TURN

TRUSTED COMPUTER SYSTEMS: NEEDS AND  
INCENTIVES FOR USE IN GOVERNMENT AND  
THE PRIVATE SECTOR, JUNE 1981



Welcoming Address  
Fourth Seminar on the DoD Computer Security Initiative  
August 10, 1981

James H. Burrows  
Director, Institute for Computer  
Sciences and Technology

I am pleased to welcome you to the Fourth Seminar on the Department of Defense Computer Security Initiative Program. As in the past, the National Bureau of Standards and the Institute for Computer Sciences and Technology are happy to collaborate with Office of the Secretary of Defense in bringing information about trusted computer systems to users and system developers. I am told that there is a plan to hold a fifth seminar in this series next Spring to continue these valuable information exchanges.

The program announcing this seminar also announced the establishment of the computer security evaluation center for the defense and intelligence communities at the National Security Agency, a subject to be addressed by our distinguished keynote speaker this morning. We are glad that this has come to fruition, and hope that we will be able to continue to work with the evaluation center through the security initiative in diffusing trusted system technology to the user community.

Computer security is no longer an exclusive concern of the defense and intelligence communities. These agencies, of course, have rigorous requirements for protecting the secrecy of data. However, as we become more dependent upon computers for handling financial, health, and other critical information, techniques for assuring the integrity and reliability of computer systems become essential throughout the government and private sector.

Not only do the defense agencies in the Federal Government need off-the-shelf solutions to their security problems, but so do the ADP users in the civil government agencies and the private sector. NBS can play a role in getting information about this needed technology to users through technical interchanges such as this seminar, through the publication of technical reports, and through the development of computer security standards and guidelines when the technology is appropriately developed.

The Paperwork Reduction Act of 1980 (P.L. 96-511) passed last year reflects Congress' concerns that computer security efforts be integrated into the overall information resources management concept. Among the responsibilities, centered on the Office of Management and Budget, in implementing the Act are the functions of developing and implementing policies, principles, standards, and guidelines on information disclosure and confidentiality, and on safeguarding the security of information collected and maintained by the agencies. With its emphasis on planning for information technology acquisition and use, the Act provides the impetus for including essential activities such as planning for computer security into agency long-range planning for information management activities.

I believe that computer security is a pervasive problem that needs top level attention from managers, as well as from technical staff. It is a problem that encompasses the entire information processing cycle from intake of data through the processing of data, the delivery of the information product, and the storage of data. While the need is pervasive, it is also clear that achieving a secure system is costly in both time and money.



Since the technology of computer security is not available in existing computer systems, we have tried to attack the problem of computer security through a variety of administrative and management controls which will continue to be essential elements for achieving secure systems. Trusted system technology, however, offers promising capabilities for maintaining the integrity and reliability of critical systems. That assuring integrity and reliability is important is evident in the estimates that problems associated with errors, omissions, and modifications of data occur ten times more frequently than intentional disclosures.

I, therefore, strongly support this R&D and technology transfer effort and hope that this is a successful and fruitful seminar.

I now have the honor of introducing our distinguished keynote speaker, Admiral Bobby R. Inman, who has broad experience both in the defense and intelligence communities. Admiral Inman, a graduate of the University of Texas, Austin, began his career in the U.S. Navy in 1952. Since then, he has held the positions of Director of Naval Intelligence, Vice Director of the Defense Intelligence Agency for Plans, Operations and Support, and the Director of the National Security Agency. He is currently the Deputy Director of Central Intelligence. Let's welcome Admiral Inman to this seminar.

KEYNOTE ADDRESS

COMPUTER SECURITY INITIATIVE

Admiral Bobby Inman  
Deputy Director of Central Intelligence  
Washington, D.C.

It is a pleasure to welcome you to this Seminar and to speak briefly with you about computer security, the recent developments within the Department of Defense and the Intelligence Community and the challenges that lie ahead.

As Dr. Gerald P. Dinneen, former Assistant Secretary of Defense for C<sup>3</sup>I defined at the first of these Seminars two years ago, a "trusted" computer system is one with sufficient hardware and software integrity to allow its use for the simultaneous processing of multiple levels of classified or sensitive information.

The need for trusted computer systems is very real and growing rapidly. Factors influencing this need are:

- the growing use of automated information handling systems throughout the DoD and the Intelligence Community and in particular the linking of these systems into major networks;
- increasing requirements for controlling access to compartmented and sensitive information;
- the requirement for broader dissemination of information both within and beyond the community;
- growing difficulties with obtaining required numbers of cleared personnel, both military and civilian.

Despite continuing internal efforts to develop special purpose trusted systems for unique needs, we already rely very heavily on the products of the computer industry to meet our information processing requirements, and this

dependence will continue to grow significantly in the future. It is therefore very gratifying to observe the progress being made by the computer industry in applying computer security technology as represented by the industry presentations at this and the previous Seminars.

It is very important, also, that the Department of Defense and the Intelligence Community develop sufficient expertise to be able to evaluate the integrity of computer software and systems developed by industry and government, and that we be able to determine suitable physical and administrative environments for their application. We have had scattered efforts over the past several years to evaluate specific systems for specific installations. But these efforts have always been more or less ad hoc, and because of the extensive technical background required, expensive to carry out.

I am very pleased therefore to announce today the establishment of a Computer Security Technical Evaluation Center for the Department of Defense and the Intelligence Community at the National Security Agency. Last fall, as Director of NSA, I enthusiastically endorsed the establishment of this Center at NSA as a new and separate function. I am very pleased with the progress being made in setting up the Center and I remain strongly committed to its success.

I would like to make several observations about the Center and some of its relationships:

- Because the private sector computer manufacturing community is the primary source of ADP systems, the Center's role will be to work with the manufacturers, deriving as much system integrity as possible from industry developed systems. This is a rather sharp contrast to the NSA's more traditional communications security role where the government has the dominant technical role.

- The Center will have a difficult task developing procedures which assure protection of sensitive portions of a system which the government does not own. Simply classifying security related portions of a system built by industry won't work since the government represents such a small portion of the overall market that the manufacturers may well decide not to sell to the government rather than accepting the limitations imposed by classification. This, in the end, might lead to a highly undesirable situation where private sector users (e.g., banks, insurance companies) have higher integrity systems than the government.
- But sensitive portions of systems and the known vulnerabilities that remain must be protected, in the interests of both the government and the manufacturers. It is quite likely therefore that the most sensitive portions of the government's analyses will be both classified and proprietary to the manufacturer. Careful, reasoned interaction between the government and industry will be needed to work out suitable working relationships.
- The Center will act in the interests and for the benefit of the Department of Defense and the Intelligence Community. Its evaluation will not be intended for use by other than the DoD. It will not make general product endorsements. But as with the Qualified Products List procedures (as prescribed in the DoD Defense Acquisition Regulations), the relative merit of a system in the hierarchy of evaluated products may be available publicly in order to provide incentive and encouragement for manufacturers to develop trusted systems and private sector users to employ them.

- Because of the wide range of sensitive environments that exist for information systems (ranging from privacy applications to compartmentation within the Intelligence Community, and from adjacent security levels (e.g., Secret and Top Secret) to full multi-level systems with Intelligence users and uncleared users), it will be vital for the Evaluated Products List to offer a range of technical categories and appropriate environments for specific systems. The approach of establishing levels of technical integrity which has evolved from the work of the Computer Security Initiative indicates the kinds of distinctions which will be made in evaluating systems. A range of suitable environments is possible with trusted systems because the security accreditation of ADP systems depends upon all of the aspects of the total system. The accreditation of a system to serve users cleared at both the Secret and the Top Secret level is not as difficult a problem as extending the use of such a system to uncleared users as well. The Department of Defense is now using Multics in such a limited environment serving both Secret and Top Secret cleared users. The Evaluated Products List should provide guidelines for implementing this type of operation where sufficient technical integrity of software products can be demonstrated.
- Finally, I would like to say that the establishment of an Evaluation Center, important as it is, must not be viewed as providing by itself the long sought answer to the computer security problem. Within the Department of Defense and the Intelligence Community, system builders will have to become aware of and properly employ the procedures for development of trusted system applications. The Services and Defense

Agencies are being encouraged to establish or enhance their own technical security test and evaluation capabilities to ensure widespread use and availability of trusted computer systems. The computer manufacturing community must work closely with the Center and these Service organizations to ensure that reasonable products are available for use in sensitive applications.

In conclusion, I would like to restate my awareness of the importance of this problem area, my enthusiasm for the establishment of the Evaluation Center, and my deep and continuing interest in its success. I encourage you participate fully in this Seminar, ask the tough questions, learn all you can and then go out and apply what you have learned so that we may all have trust worthy computers in the very near future.

## INTRODUCTORY COMMENTS

STEPHEN T. WALKER

DIRECTOR INFORMATION SYSTEMS

OFFICE OF DEPUTY UNDER SECRETARY OF DEFENSE (C<sup>3</sup>I)

Good Morning. It is indeed a pleasure to welcome you to the Fourth Seminar on the DoD Computer Security Initiative.

It was just three years ago that we began the Computer Security Initiative and just two years ago that we held the First Seminar here at NBS. We had two major goals when we started this effort and I am proud to announce that as of today I believe we have accomplished both of them.

As I described in my opening remarks at the last Seminar, our major external objective for the Initiative, that of getting the computer manufacturers involved in the development of trusted systems, had already come a long way as indicated by the five manufacturers who described their efforts at that seminar. This time, as you glance at your program you will see that we have eight manufacturers giving presentations; seven new ones including three European manufacturers and one giving an update from last time.

I must admit that I expected only two or three manufacturer presentations and as Pete Tasker and I were working out the program we had the pleasant task of frequently shuffling the program as more manufacturers accepted our invitation to speak.

I think it is obvious from the number and variety of manufacturers represented today and at the last Seminar that there is a strong interest in computer security and in trusted computer systems in the US and international computer manufacturing community. This external interest is most gratifying.

But just as exciting to me at least is the progress we have made to satisfy the major internal objective of the Initiative. At the last Seminar I hinted that within a year there would be a technical integrity evaluation process in being to serve the DoD.

In fact, as Admiral Inman has just announced, that goal has been met with the establishment of the DoD Computer Security Evaluation Center at NSA. The Deputy Secretary of Defense made it official as of January 1, 1981 and NSA has been hard at work pulling all the necessary pieces together to get the Center functioning. Tomorrow afternoon you will hear a status report on the Center from Mr. George Cotter of NSA.

I am personally very excited and pleased with our progress in just three years. It is clear to me that the time was right for what we have tried to

do. My personal thanks to everyone who has helped make this possible. I believe that the combination of rapidly growing interest on the part of the computer manufacturers and the existence of a DoD evaluation capability will profoundly influence the integrity of computer systems in the very near term and from now on.

It is vital that we start to take advantage of this improvement as soon as possible. In just a minute, I would like to propose a challenge to both the computer manufacturers and the computer users both in this audience and beyond.

Let me first describe a particular situation as I see it in right now.

Over five years ago the Air Force, after extensive testing and evaluation, installed a Honeywell MULTICS System at the Data Services Center in the Pentagon. That system has successfully operated in a Top Secret environment with some users cleared only for Secret access for several years. It is a general purpose system being used for all kinds of programming and administration support to the AF.

I am not recommending that everyone go out and buy a MULTICS System to satisfy all their needs. But as I review the efforts of the many manufacturers that I have talked with lately, I realize that there is a real potential for a number of systems with integrity similar to MULTICS to be available in the not so distant future.

So what, you say! A Top Secret-Secret environment is not fully multilevel secure. I can't have the highest levels of sensitive data on my system with unclassified users so it hasn't solved my problem.

In reality though, not very many applications require a system to operate over anything like the full range of sensitive information. This afternoon you will hear about the computer security aspects of the WWMCCS Information System Modernization effort, perhaps the largest, highly sensitive computer system upgrade that the DoD will perform this decade. There are multilevel security problems throughout WIS but as you will hear, the requirements exist over a reasonable range of sensitivity levels, not necessarily over the full range of possible levels.

If one couples the fact that the manufacturers could soon develop trusted systems with integrity levels similar to MULTICS and the realization that many of our security requirements can be met by systems that operate over a limited range of sensitivity, it is possible to see how solutions to at least these limited applications may be forthcoming very soon.

You may accuse me of advocating a less than perfect solution by what I've just said. Far from that, though, I am advocating seeking a reasonable, useful solution prior to seeking the perfect solution. Indeed if we do not make serious attempts to crawl before we run here, we very likely will never get anywhere near that perfect solution.



Now back to my challenge. I would like to challenge the users in this audience to seriously review their needs for trusted computer systems and determine, as Larry Bernosky has for the WWMCCS Information System, which needs could be met by systems able to operate over limited sensitivity ranges. To the extent that you can do this, I strongly urge you to convey this information to your local computer manufacturers representatives to help motivate them to develop systems to meet your needs, and then get involved in the evaluation of potential systems for your application.

I would similarly challenge all the manufacturers in the audience to study what has been done to date, understand the security design of systems like MULTICS, Kernelized Secure Operating System (KSOS) and Kernelized VM 370 System (KVM) and incorporate these ideas into your product lines, quickly. More and more users are beginning to realize not only that they need improved integrity within their computer systems but also that it is possible to build systems with these improvements, and that they can begin to demand such features. As you can tell from the manufacturers participation here, at least some of your competitors are taking this seriously.

We've come a long way in the last few years. We've completed the first tough phase of the Initiative, getting the various pieces in place. Now it's time to move into phase two. This will involve a lot of work by the manufacturers and you the users have the opportunity and the responsibility to get involved.

I know by your being here that you are interested. I challenge you to get seriously involved.

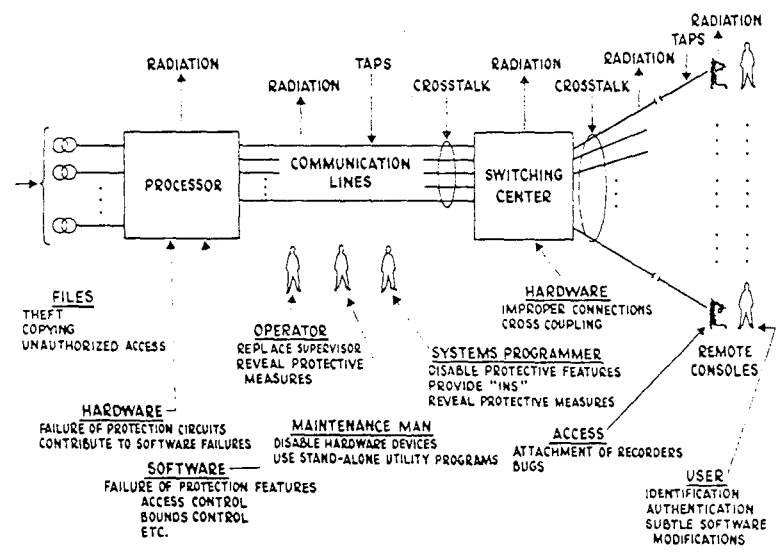
I would now like to summarize the activities of the Initiative on the next few slides.

DoD  
**COMPUTER  
 SECURITY INITIATIVE**

...TO ACHIEVE THE WIDESPREAD  
 AVAILABILITY OF TRUSTED  
 COMPUTER SYSTEMS

Stephen T. Walker  
 Chairman  
 DoD Computer Security  
 Technical Consortium

**COMPUTER NETWORK VULNERABILITIES**



## **COMPUTER SECURITY**

**PHYSICAL SECURITY**

**ADMINISTRATIVE SECURITY**

**PERSONNEL SECURITY**

**COMMUNICATIONS SECURITY**

**EMANATIONS SECURITY**

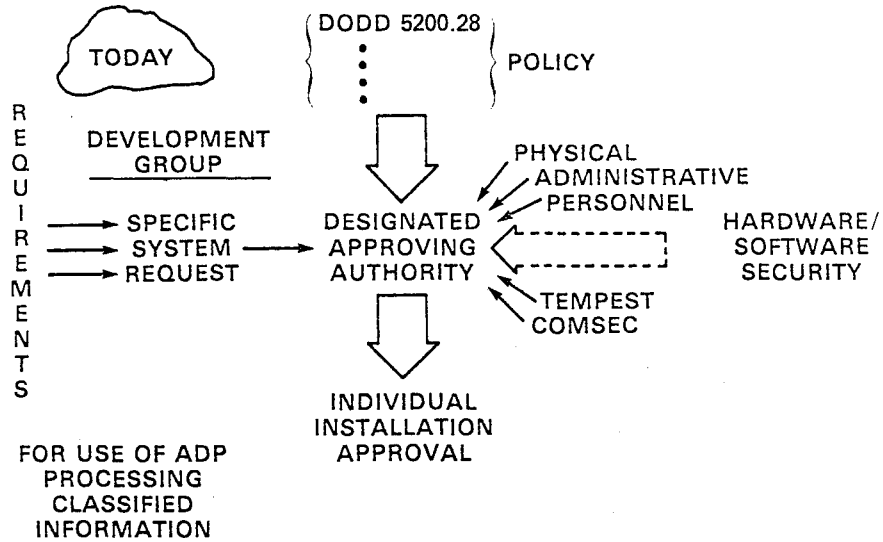
**HARDWARE/SOFTWARE  
SECURITY**

## **COMPUTER SECURITY INITIATIVE**

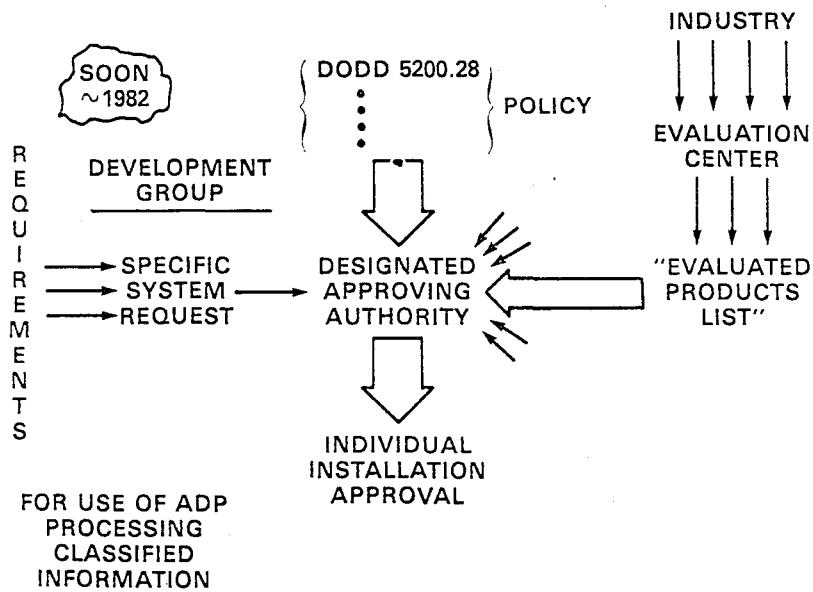
**TRUSTED: SUFFICIENT HARDWARE AND  
SOFTWARE INTEGRITY TO  
ALLOW SIMULTANEOUS USE  
AT MULTIPLE SECURITY/  
SENSITIVITY LEVELS**

**WIDESPREAD: COMMERCIALY SUPPORTED**

## APPROVAL FOR DoD USE



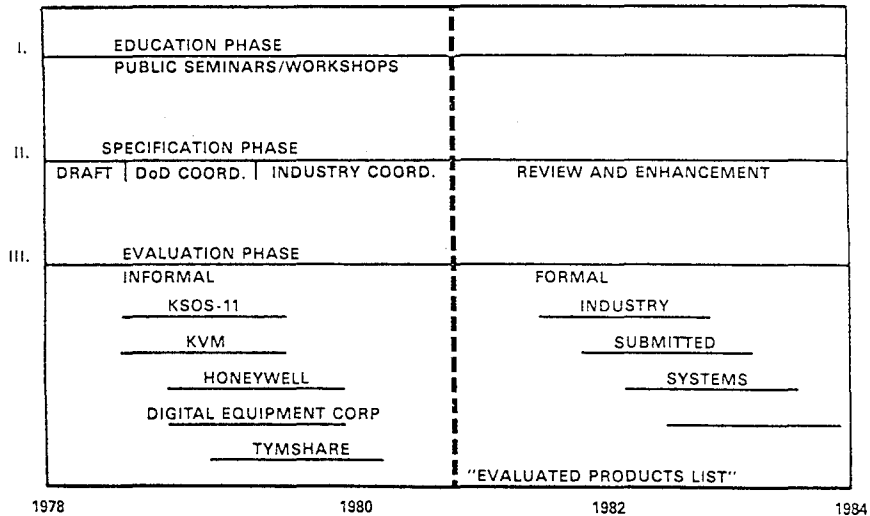
## APPROVAL FOR DoD USE



## EVALUATED PRODUCTS LIST

CLASS	TECHNICAL FEATURES	EXAMPLES	POSSIBLE ENVIRONMENTS
1	-	MOST COMMERCIAL SYSTEMS	DEDICATED MODE
2	FUNCTIONAL SPECIFICATION REASONABLE PENETRATION RESULTS	"MATURE" "ENHANCED" OPERATING SYSTEM	BENIGN, NEED TO KNOW ENVIRONMENTS
3	REASONABLE MODERN PROGRAMMING TECHNIQUES LIMITED SYSTEM INTEGRITY MEASURES	MULTICS	AF DATA SERVICE CENTER TS-S
4	FORMAL DESIGN SPECIFICATIONS SYSTEM INTEGRITY MEASURES		NO USER PROGRAMMING TS-S-C
5	PROVEN DESIGN SPECIFICATIONS VERIFIABLE IMPLEMENTATION LIMITED COVERT PATH PROVISIONS	KSOS KVM	LIMITED USER PROGRAMMING TS-S-C
6	VERIFIED IMPLEMENTATION AUTOMATED TEST GENERATION EXTENDED COVERT PATH PROVISIONS REASONABLE DENIAL OF SERVICE PROVISIONS		FULL USER PROGRAMMING TS-S-C

## COMPUTER SECURITY INITIATIVE



## **COMPUTER SECURITY INITIATIVE**

---

ON JANUARY 1, 1981 THE SECRETARY OF DEFENSE ASSIGNED RESPONSIBILITY FOR COMPUTER SECURITY EVALUATION FOR DOD TO THE DIRECTOR, NATIONAL SECURITY AGENCY.

## **COMPUTER SECURITY EVALUATION CENTER**

---

- ESTABLISH TECHNICAL EVALUATION CRITERIA
- EVALUATE INDUSTRY AND DOD SYSTEMS
- MAINTAIN EVALUATED PRODUCTS LIST
- SPONSOR R&D IN COMPUTER SECURITY
- ENCOURAGE DEVELOPMENT AND WIDESPREAD USE OF TRUSTED COMPUTER SYSTEMS

**CURRENT INITIATIVE EVALUATION  
EFFORTS**

**CONTROL DATA  
DIGITAL EQUIPMENT CORPORATION  
HONEYWELL  
INTEL  
TYMSHARE  
UNIVAC**

**UNDER DISCUSSION**

**BURROUGHS**

# Burroughs

FEDERAL AND SPECIAL SYSTEMS GROUP

CHRIS TOMLINSON

## RESEARCH AND DEVELOPMENT

### LOCAL NETWORK SECURITY

- ▶ POLICY
- ▶ DESIGN
- ▶ ACCEPTANCE



**NEED FOR INTERCONNECTION OF COMPUTERS AND PERIPHERALS**

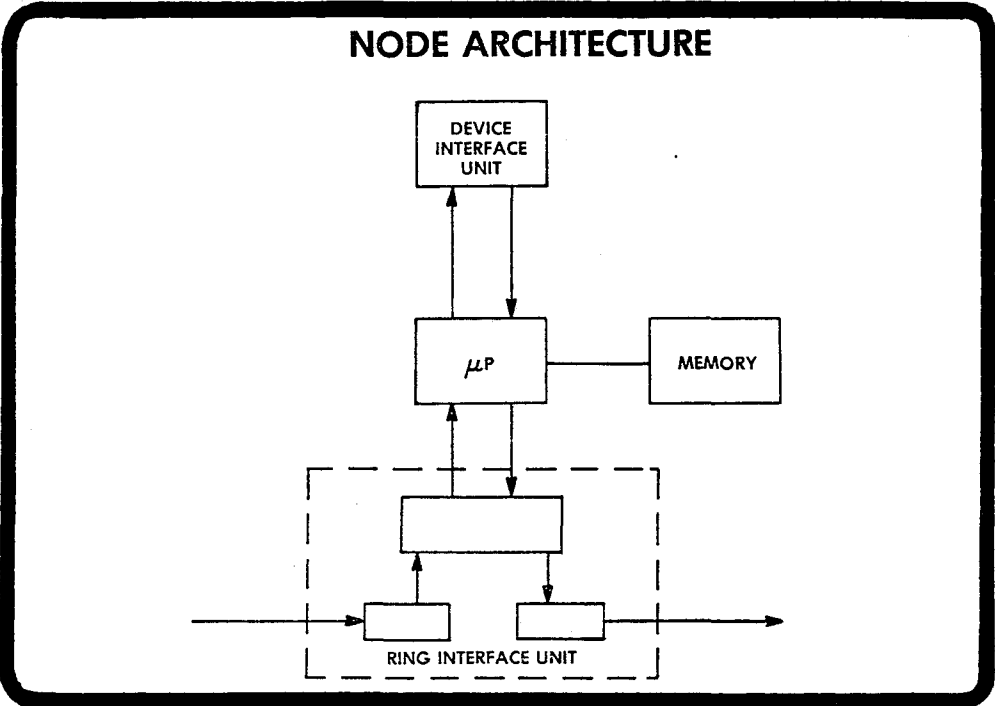
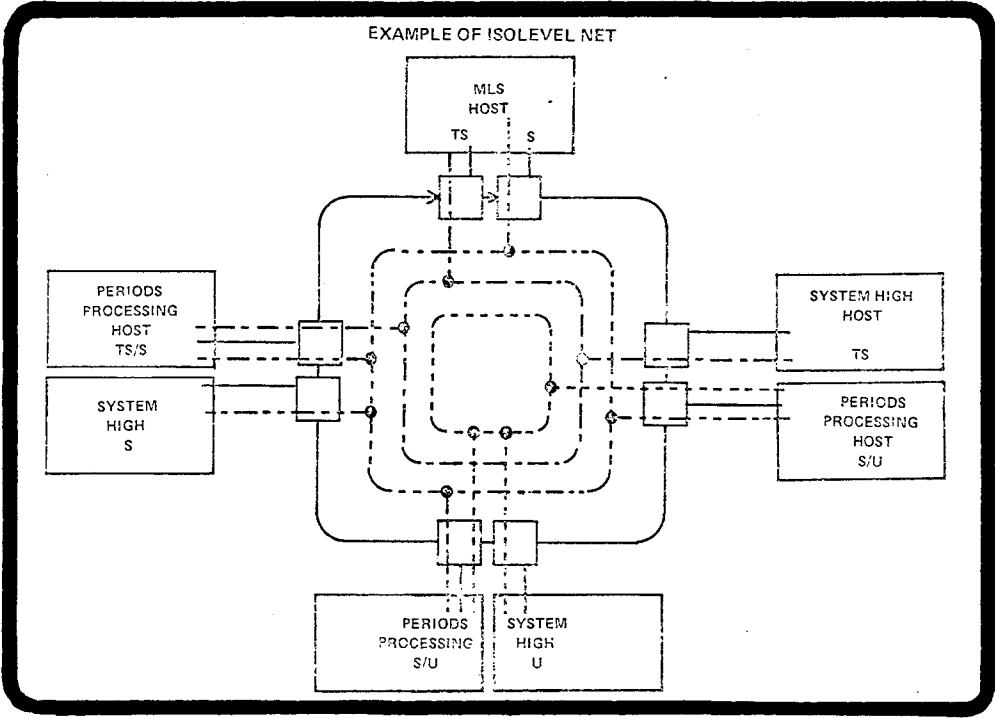
- ▶ **SYSTEM HIGH**
- ▶ **PERIODS PROCESSING**
- ▶ **MLS HOSTS**

**USING LOCAL NETWORK TECHNOLOGY**

**ISO-LEVEL POLICY**

**COMMUNICATION IS PERMITTED ONLY  
AMONG SUBSCRIBERS AT THE IDENTICAL  
SECURITY LEVEL (CLASSIFICATION,  
CATEGORY)**

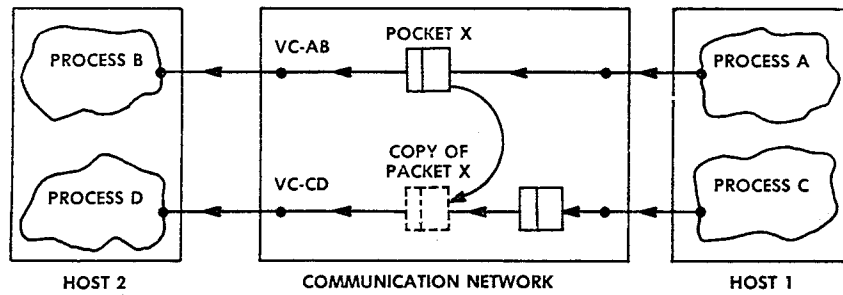
**NO OTHER COMMUNICATION CAN  
OCCUR**



## SECURITY PROBLEMS ADDRESSED

- WITHOUT  $E^3$

— PACKET CONTENTS CAN BE COPIED DIRECTLY BETWEEN SECURITY LEVELS

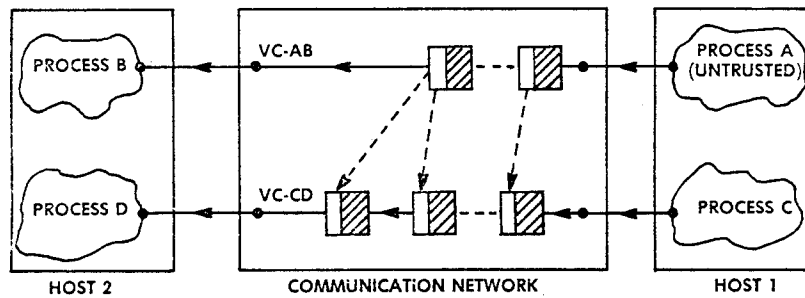


## SECURITY PROBLEMS ADDRESSED

- WITH  $E^3$

— PACKET HEADERS MUST REMAIN IN PLAIN TEXT

— UNTRUSTED PROTOCOL PROCESSES CAN DOWNGRADE CLASSIFIED DATA BY DECODING/ENCODING INFORMATION CONTAINED IN PACKET HEADERS



## POTENTIAL SOLUTIONS

### STORAGE CHANNELS

- ▶ ELIMINATE VARIABLES
  - ADDRESSES
  - LENGTH
  - OTHER HEADER FIELDS
- ▶ ENSURE THAT UNTRUSTED NETWORK PROCESSES CANNOT COMMUNICATE WITH ONE ANOTHER OUTSIDE THE POLICY

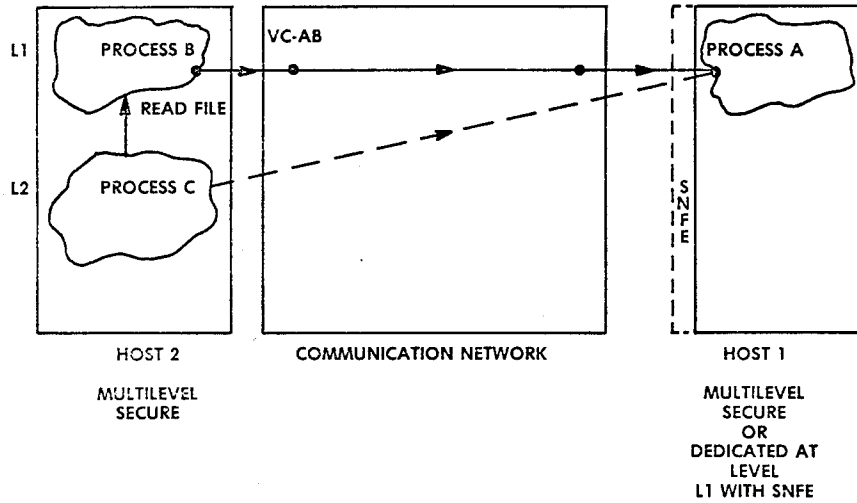
### TIMING CHANNELS

- ▶ TIME DIVISION MULTIPLEXING

## KERNEL SUPPORTS

- ▶ MESSAGE BASED IPC
- ▶ PROCESS ISOLATION
- ▶ POLICY PROCESSES

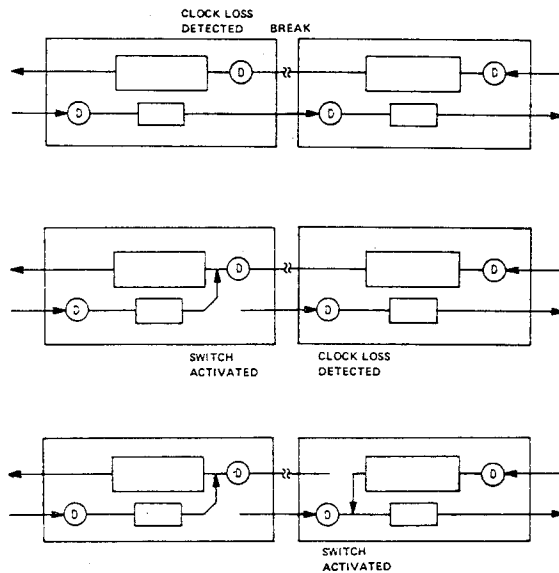
## INTERLEVEL INFORMATION FLOW



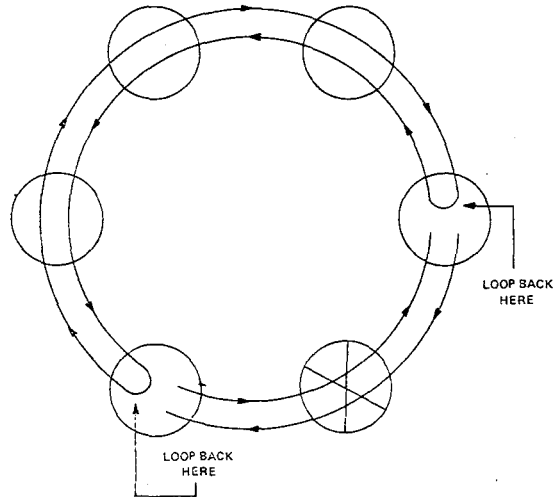
NOTE: L1 > L2

Burroughs

## LOOP BACK



## LOOP WITH FAILURE



## ADDITIONAL MEASURES

- END-TO-END ENCRYPTION
- TRAFFIC FLOW SECURITY

## **SUMMARY**

**EXPLOIT A SIMPLE AND USEFUL POLICY  
TO REDUCE THE EFFORT OF  
CONSTRUCTING A SECURE LOCAL  
NETWORK**

## CR80-A Fault Tolerant Computer for Implementation in Secure Systems

Asbjørn Smitt

Head of Research and Development  
Christian Rovsing A/S, Ballerup, Denmark

### 1.1 General

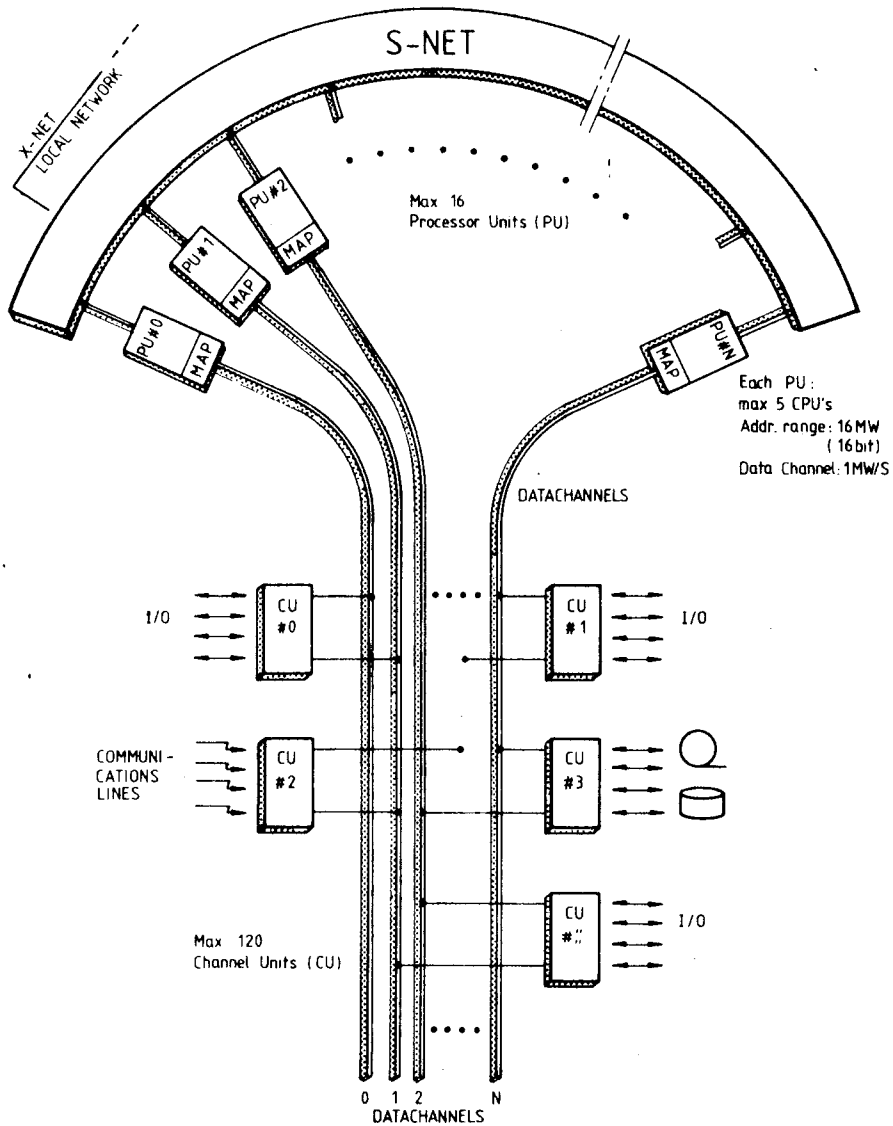
Christian Rovsing A/S with the CR80 MAXIM and FATOM virtual machines has introduced a new and powerful architecture for implementing secure systems on a ultra-reliable, easy to maintain and modular fail safe computer. The high speed memory mapped multiprocessor computers have been designed to provide modular growth in processing power and memory requirements to cope economically with the requirements of:

- General purpose computer systems
- Packet switches
- Message switches
- Control and Command Information
- Concentrators
- On-line systems
- Terminal systems
- Front end processors

The illustration overleaf shows that the CR80 FATOM computer tightly couples up to 16 Processing Units (Multiprocessors) together via the S-NET, and that each peripheral connects through individual channels to two Processing Units, one channel being the active connection for a connected peripheral, the other the back-up connection. Also it is seen that the CR80 MAXIM (Memory mapped Maxi-computer) is the single Processor Unit, non-redundant subset of the CR80 FATOM (Fault Tolerant multiprocessor) otherwise they have identical high performance characteristics.

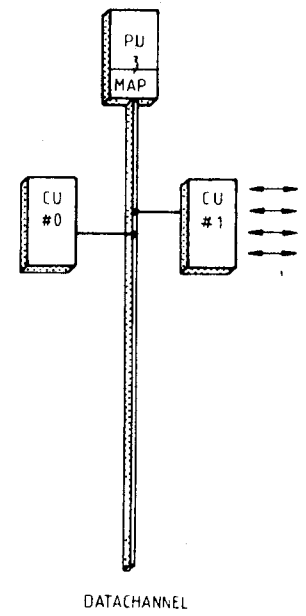
The CR80 FATOM fault tolerant computer differs from other computers (large, medium or small) in that it, based on a unique distribution of its memory providing nearby unlimited processing power, up to 50 Million instructions per second (MIPS) together with minimum added hardware to achieve its "self repair" features and 256 Mega word maximum memory size.





# CR80 FATOM

FAult Tolerant Multiprocessor



# CR80 MAXIM

Extensive hardware checks has been incorporated throughout the CR80 architecture, supporting integrity and security in execution of both application and system programs, ensuring that erroneous interaction among users, and with the system software, are prohibited. This is extremely important during software maintenance and development, once a fault tolerant system has been brought operational, as well as facilitating the initial software development and debugging.

The CR80 architecture and DAMOS system software supports modularly the total spectrum of virtual memory machines, from the 0.7-3.0 MIPS MAXIM multiprocessor computer with one or more CPUs, up to the 50 MIPS, N+1 redundant FATOM computer, incorporating the cost effective approach of only having 1 single spare unit, capable of backing up for any of N working units). The CR80 can be upgraded in the field, often without stopping operational use, due to its on-line maintainability and unique galvanic isolation between system elements at the card-magazine level.

A CR80 Processor Unit (PU) constitutes either a uni- or multiprocessor computer with from 1 to 5 CPUs (.7 to 3 MIPS). The CR80 FATOM connects up to 16 Processor Units (PUs) together via the extremely fast S-NET (up to 512 Mbit/sec.) into a tightly coupled multicomputer with up to 50 MIPS capability. In addition all lower levels of input/output processing is distributed to the I/O controllers (peripheral processors) in the Channel Units (CU), this further enhances the CR80 above the simple accumulated processing power of the CPUs.

The I/O Controllers (peripheral processors) communicates with PUs through one port of the triple ported controller memory, the two other ports allowing for this memory being part of the address space of two processing units (PUs), which ensure an alternative path, in case of a Processing Unit (PU) failure.

The CR80 computers also gain their strength from very fast intelligent multiplexed Direct Memory Access (DMA) channels between the distributed memory in PUs and CUs and that the imbedded channel processors (S-NET & DATA CHANNEL) with minimum interruption of the CPUs autonomously handle and ensure the integrity of hundreds of simultaneous active logical channels between programs and processes.

The CR80 FATOM basic system philosophy is to achieve N+1 redundancy on all levels, both processors and I/O controllers. A unified system approach to software in a redundant system, relieving application software as far as possible of mechanisms and functions necessary for fault tolerance, moving these to the system S/W. The CR80 FATOM Computer thus is designed to have no single points of failure on a system basis, this includes all parts of the system: Processors, busses, I/O devices, power supply, cooling and software in order to achieve a continuously available no-break computer. The on-line maintenance features, allows any failed module to be exchanged and tested, without interrupting system operation.

Also the CR80 modular packaging and integration system, ensures the capability for expansion of a CR80 FATOM Computer to virtually any physical size, using only a few standard types of modules and cables, as well as achieves the cost efficiency of both the single and fault tolerant CR80 Computers.

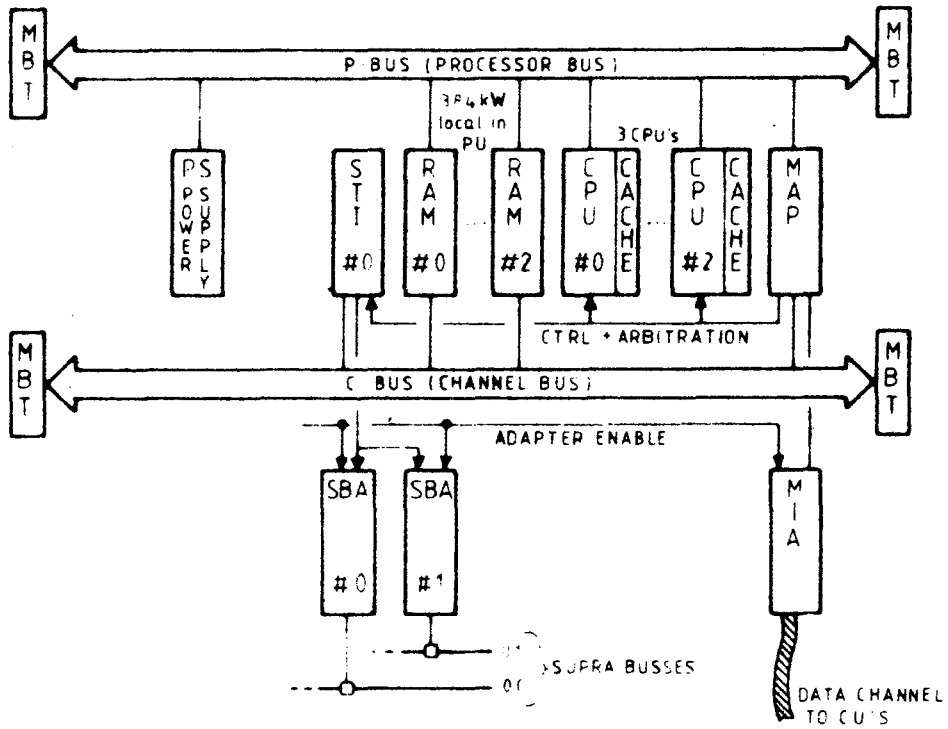
## 1.2 PU Logical Organisation

As an introduction to the features of the CR80 memory mapped PU a brief discussion of the CR80 Processor Unit Logical Organization, shown overleaf is given.

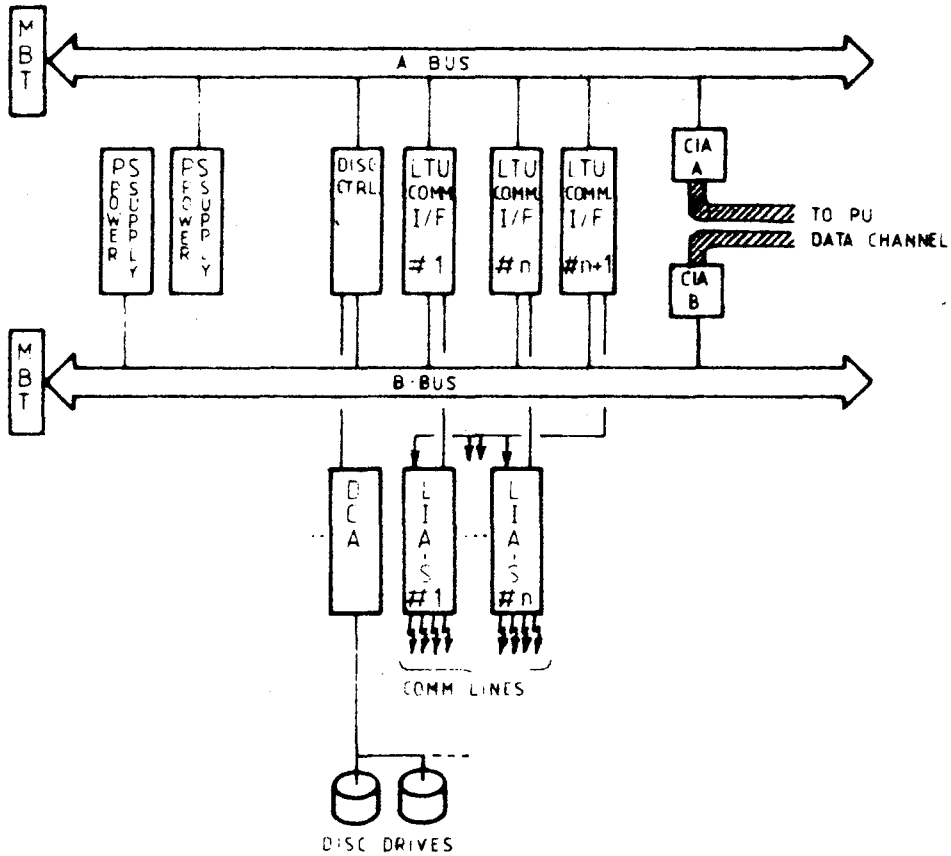
Interconnection of the PU modules is performed by means of two parallel transfer busses, the (Processor Bus) and the (Channel Bus) implemented as two backplane printed circuit boards. The busses have identical electrical and timing specifications with the following characteristics: transfer rate up to 4 mega word per second (16 bits + 2 parity bits), addressing of 1 mega word as word or byte. The Processor Bus performs as transfer bus for the Central Processor Units (CPUs), while the Channel Bus performs as transfer bus for the Channel Bus modules (DMAs).

The central processor units, CPUs, are general purpose processor units with a word length of 16 bits and the ability to address 64K word of instruction and 64K word of data. All data/instruction transfer performed by the CPU are via the processor bus and the memory MAP to the memory. Physically, the CPUs and the memory MAP are connected to the same Processor Bus, but logically the CPUs recognize the MAP as being located between the memory and the CPU.

The function performed by the memory MAP is to expand the addressable memory area to 16 mega word of which 1 mega word can be located in the PU as fast access, local storage, while the remaining 15 mega word can be located on the data channel. Besides the address translation, the MAP also includes memory read/write protection, the protection can be performed individually for each 1K page of the memory.



CR80 FATOM PU CONFIGURATION



CR80 FATOM CU CONFIGURATION

The functions performed by the MAP on the Processor Bus transfers are also performed on all Channel Bus transfers, meaning the Channel Bus Modules can access the complete 16 mega word memory area, but only the areas which are not protected.

Beside the address translation described above, the MAP module also includes the Channel direct memory access (DMA) function, interrupt preprocessing and Data Channel Interface.

The DMA is used for block transfer between shared memory with peripheral Controllers and PU local memory and is under control of the Input/output system software.

The interrupt preprocessing performed ensures that only interrupts (CPU or I/O) with sufficient priority will cause a context switch in one of the CPUs, while all other interrupts will be queued by the MAP, until the CPU status allows service of them.

Transfer on the Data Channel will be performed by the memory MAP when the addressed location is not within the PU Local Main Memory addressing space (1 Mw).

Security is supported by means of memory access protection and division of instructions into three privilege classes.

The CPU has 16 states of which one (state  $\emptyset$ ) is a user state and 15 (states 1 through 15) are system states. In user state only not privileged instructions may be executed. Medium privileged instructions can be executed at all system states while the most privileged instructions are reserved for execution at system state 15.

Attempt to illegally execute a privileged instruction in user state or system states 1 through 14 causes a local interrupt, upon which the CPU automatically invokes a supervisor routine.

The CPU state is changed by means of the MON\_instruction which is used to activate system procedures.

### 1.3. CR80 Security Mechanisms

The inherent logical and physical separation of programs and data in the CR80 architecture is well suited for preventing unauthorized access to data and programs and for preventing non-intended modification of programs.

The objectives of the protection mechanisms in the CR80 are:

- to protect data belonging to a process against unauthorized modification by other processes and against not intended reading;
- to protect programs against modifications, and,
- to prevent unauthorized execution of programs and system resources
- to prevent processes from monopolizing the processor.

Security is supported by means of memory access protection and division of instructions into three privilege classes.

The CPU has 16 states of which one (state  $\emptyset$ ) is a User state and 15 (states 1 through 15) are System States.

Higher states have more privileges than lower states. In user state only not privileged instructions may be executed. Medium privileged instructions can be executed at all system states while the most privileged instructions are reserved for execution at system state 15.

Attempt to illegally execute a privileged instruction in user state or system states 1 through 14 causes a local interrupt, upon which the CPU automatically invokes a supervisor routine.

The CPU state is changed by means of the MON\_ instruction which is used to activate system procedures.

In addition to the memory protection provided in USER STATE by the MEMORY MAP, each of the system states has its own memory bound register. Only data memory locations below or equal to this boundary value may be modified while all data memory locations available might be read in SYSTEM STATE.



The Memory Map protection mechanism which is active in user state is implemented by means of two access control bits for each 1 Kw page in memory. The protection values are:

access  
control  
bits:

- 00 Page absent
- 01 Full access
- 10 Read only
- 11 No access

As will be seen in the following all non-privileged (USER STATE) memory accesses (both from CPU's and DMA's) go through the Memory Map, and are checked by hardware not to violate the protection value. In the system state full access (read or write) is granted irrespective of the protection value.

If a not allowed access is attempted, the transfer is terminated without sending the physical address to the memory, and, a transfer error is signalled from the Memory Map.

The "Page absent" condition is used to invoke the demand paging feature of DAMOS. It indicates that the accessed page is not resident in main memory (or not mapped in), and will lead to suspension of the process until the page has been loaded into memory or relocated.

## 1.4 Security

The CR80 operating system DAMOS offers comprehensive data security features. A multilevel security system ensures that protected data is not disclosed to unauthorized users and that protected data is not modified by unauthorized users.

All memory allocatable for multiple users is erased prior to allocation in case of reload, change of mode, etc. The erase facility is controlled during system generation.

DAMOS is specified using the formal notation of the Vienna development method with the intention of making formal verification possible.

The security system is based on the following facilities:

- a. Hardware supported user mode/privileged mode with 16 privilege levels. Privileged instructions can be executed only when processing under DAMOS control.
- b. Hardware protected addressing boundaries for each process.
- c. Non-assigned instructions will cause a trap.
- d. Primary memory is parity protected.
- e. Memory bound violation, non-assigned instructions, or illegal use of privileged instructions cause an interrupt of highest priority.
- f. The hierarchical structure of DAMOS ensures a controlled use of DAMOS functions.
- h. A general centralized addressing mechanism is used whenever objects external to a user process are referred to.
- i. A general centralized access authorization mechanism is employed.

Centralized addressing capabilities and access authorization are integral parts of the security implementation. User processes are capable of addressing Kernel objects only via the associated object descriptor table. The following types of DAMOS objects are known only via object descriptors:

- a. Processes
- b. Synchronization elements
- c. Segments
- d. Devices
- e. PUs
- f. CPUs
- g. Ports

The object descriptor forms the user level representation of a DAMOS Kernel object. It contains the information necessary for the Kernel to locate its low level representation and to ensure its security and integrity:

- a. Host PU
- b. Object type
- c. Object control block index for use by the Kernel to locate the corresponding object control block.
- d. A sequence number which must match a number in the object control block (to prevent reallocated blocks from being erroneously accessed).
- e. A capability vector specifying the operations which may be performed on the object by the process which has the object descriptor.

The access right information concerning the various DAMOS objects is retained in a PU directory of object control blocks. Each control is associated with a single object.

When the access right of a process to a segment is verified and the segment is included in the logical memory space of the process, the contents of that segment may be accessed on a 16-bit word basis at the hardware level subject to hardware access checks.

Authorization of access to an object is based on

- a general security policy, and
- a discretionary access checking

The security policy is based on a multilevel -multicompartment security system.

Objects are associated with a security classification level for each compartment (i.e., set of data with the same kind of information) and subjects (processes) are associated with a security clearance level for each compartment. Both entities are described in a common type:

- the security profile

Discretionary access checking is based on

- identification of access classes of subjects (processes), and
- statements of access capabilities for explicitly enumerated access classes of subjects vis a vis a given object.

Access to an object is authorized if the following conditions are both fulfilled:

- the access operation requested is allowed according to the capability vector in the object descriptor
- the combination of process security profile, object security profile and operation (read or write) agrees with the security policy.

The security policy is:

- A process may read from objects with classification not higher than that of the process. An untrusted process may write to objects with classification not lower than that of the subject.
- A trusted process may write to objects with any classification.

A process can only obtain access rights (i.e., an object descriptor) to a DAMOS object in the following ways:

- a. By inheritance from a parent process
- b. By creating the object.
- c. By successful look-up in the PU directory.

Similarly, a process can only distribute access rights to objects registered in its object descriptor table. This may be done:

- a. By inheritance when creating a child process
- b. By entering the object into the PU directory by a symbolic name.

When an object is entered into the directory it is specified by whom it may be looked up and what capabilities they should have vis a vis the object.

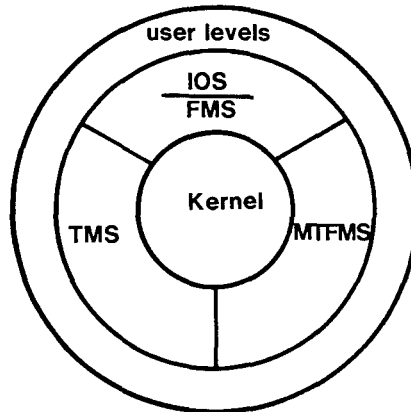
The object descriptor table and Security profile of a process is kept in a memory which is accessible by that process when it is executing in privileged mode, but protected against modification by the process when executing in user mode.



## DAMOS SECURITY

---

Layered design



## DAMOS SECURITY

---

### Kernel

- Directory functions
- CPU management
- Process management
- Memory management
- Inter process communication
- Device management
- Device handlers
- Error processing
- Real time clock
- PU management
- PU service module
- Transfer module
- Basic transport service



## DAMOS SECURITY

---

### Objectives:

#### Data security

- Protection of data against disclosure to unauthorized users
- Controlled update of data

#### Availability of service

- Protection against denial of service

### Measures:

- Capability based design
- Resource management



## DAMOS SECURITY

---

### HW security features

Memory protection embedded in memory mapping

- 16 Privilege levels  
Each with an associated memory boundary
- Privileged instructions
- Non-assigned instruction codes trap
- Parity on memory



**DAMOS SECURITY**

---

**Objects**

Security is based on controlled access to objects

**Kernel objects**

- PUS
- CPUS
- Processes
- Synchronization elements
- Virtual memory segments
- Devices
- Communication ports



**DAMOS SECURITY**

---

**Objects**

**File management objects**

- Devices (disk drives)
- Volumes (disk packs)
- Files
- Users

**Magnetic tape file management system**

- Devices (tape decks)
- Volumes (tapes)
- Files
- Users

**Terminal management objects**

- Devices (communication controllers)
- Lines
- Units (terminal, LP, VC,)
- Users

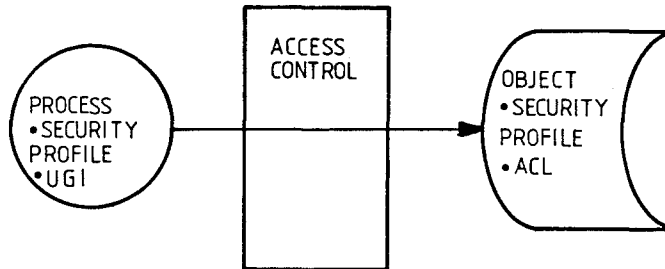




## DAMOS SECURITY

---

### Access authorization



- Security check
- Discretionary access right verification



## DAMOS SECURITY

---

### Security check

- A process may read from objects with a classification not higher than that of the process
- A process may write to objects with a classification not lower than that of the process
- A trusted process may write to objects with any classification



## DAMOS SECURITY

---

### Security profile

Defines a classification for each of a set of compartments

### Type profile record

A class min A class..max A class

- 
- 
- 
- 

N class min N class..max N class

End:

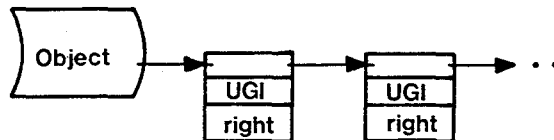


## DAMOS SECURITY

---

### Discretionary access right

- Subject identified by a user group identifier
- Object has an access control list



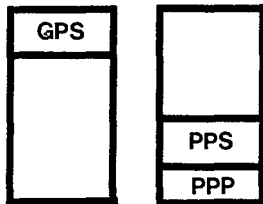


**DAMOS SECURITY**

---

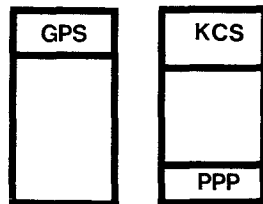
Damos processing domains

**User view**



64 kw

**Kernel views**



- PMD/PCF
- DF/BTS
- PM
- DVM (3)



**DAMOS SECURITY**

---

Process parameter segment

Level areas 1-15

Level 15:

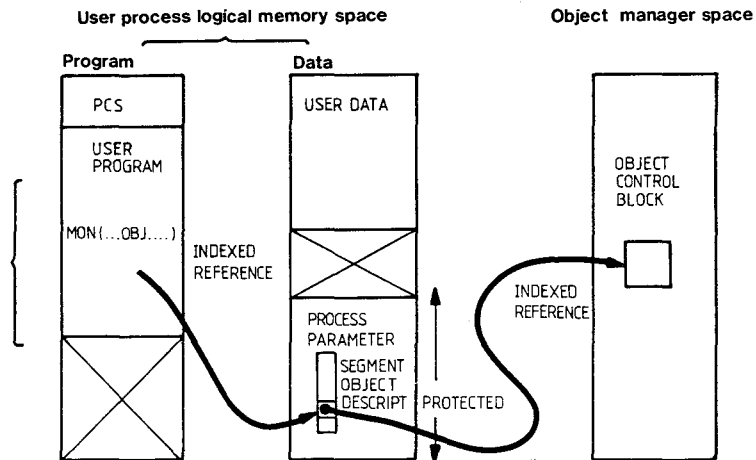
- Object descriptor array

Process parameter page

- Parameter stack
- Context stack
- Translation table
- Segment table
- Security profile
- Process level



## DAMOS SECURITY



## DAMOS SECURITY

### Object descriptor

#### Information contained in OD:

- Host PU
- Object type and subtype
- Index to object control block
- Object control block sequence number
- Object access level
- Capability vector

#### Object descriptors may be obtained via:

- Creation of object
- Inheritance from parent process
- Lookup in PU directory



**DAMOS SECURITY**

---

**Change of execution level**

- MON instruction
- RTM instruction
- Interrupt
- RTI

**Change of view (processing domain)**

- CALL instruction
- RET instruction
- Interrupt
- RTI

} Only at level 15

## Computer Security and Control Data

Terry A. Cureton  
Program Manager, Security Systems  
Control Data Corporation

August 10, 1981

[SLIDE: CDC Logo]

It is a pleasure for me to represent Control Data at this seminar. We have been observing the activities of the DoD Computer Security Initiative for some time, and are impressed with your progress. Until recently, our participation in the Initiative has been silent. For the most part, this has been due to the largely theoretical or experimental nature of the material presented. However, the Initiative has given us an opportunity to look at our own experiences in computer security from another viewpoint. We can now see the parallels and principles common to both the theoretical work and our experience as practitioners of computer security. The message we would like to share with you today is that we at last see a convergence between the theory and practice.

[SLIDE: Topics]

To begin, I must start with what Control Data is, and why we are involved in computer security. Then, I would like to dispel a myth about security and performance, by relating that to our unique machine architecture. Next, I will briefly describe how that architecture is reflected in our operating system design. A comparison of commercial versus government security requirements will show how we plan to meet both. Another comparison of formal and informal design methodologies will show how we think they are converging. Lastly, I will describe our involvement with the DoD Initiative and our view of its impact on the industry as a whole.

[SLIDE: Control Data Reputation]

What kind of company is Control Data?

- Many of you know Control Data is in the large-scale scientific and engineering computer business.
- That is our tradition and our legacy, since the company was founded in 1957 - since the days of the 1604 and the 6600.

[SLIDE: Control Data Today]

But you may not be aware of the range of Control Data's business today. Yes, we still make super-scale computers for our systems business, but we are also the industry's leading supplier of peripherals - both OEM and plug-compatibles, in addition to our own label. The next time you walk into a room full of disks, there's a good chance (65%) that we made them, since we supply OEM peripherals to all but one of the major manufacturers. We are also deeply committed to education with our unique PLATO system. PLATO is winning acceptance in uses ranging from teaching grade school fundamentals, to training airline pilots and nuclear safety engineers. But it is in Data Services that we are the world-wide leader.

[SLIDE: DATA SERVICES]

Our Data Services Company operates both commercial and scientific data centers around the world, around the clock. It's a more than half billion dollar business, reaching from Main Street to Wall Street. And - whether it is a small businessman dealing with our Service Bureau Company in Cleveland - or an engineering firm dealing with our CYBERNET Services in Copenhagen - the two questions we always get are:

[SLIDE: DS Customers Ask]

"How much will it cost?" and

"How secure will my data be?".

[SLIDE: Security (1)]

Clearly, security is a customer concern, and for Control Data it is a hard-nosed, hard-headed business need. It is here that Control Data learned about computer security in a day-to-day pragmatic way. We have been addressing that need for more than 20 years, since the beginnings of Data Services.

Now, Data Services is a large chunk of our business, in fact they are our largest Systems "customer". Their needs have a major impact on system design and development. Simply put - Security has been essential to our largest systems marketplace for more than 20 years. That's why Control Data has been involved in computer security. We will have to look at the data services environment to see how it relates to computer security.

[SLIDE. Timesharing Environment]

From a security viewpoint, it is the timesharing environment where the needs are greatest. The first need of course, is to simply keep it running, since users have little patience for system downtime. That requires a good deal of system integrity, in the first place. By definition, timesharing means multiple users sharing system resources. Those resources and the users' data are real and tangible assets which must be protected. Then, resources have to be controlled so that all may share equitably, and if you want to get paid, they have to be accounted for. Finally, users have to be kept separate, since they might be competitors.

Control Data met those needs by developing a system specifically designed for the timesharing environment. Over time, security flaws were discovered and corrected, and new security mechanisms evolved into the system design. We built up a great deal of practical experience with that system, and that system evolved into our standard system of today. But it wasn't until the DoD Initiative that we fully recognized the unique advantages of the CYBER 170 architecture regarding security.

What is so unique about the CYBER 170 architecture and security? The answer in a word is - Performance. There seems to be a growing supposition in the industry, that security can only be obtained at the expense of performance. We would like to dispel that myth, by showing how the CYBER architecture and hardware can provide security without penalizing performance. To understand why, we have to first examine the relationship between security and performance, and then how that relates to design.

[SLIDE: Performance]

When considered in a broad sense, performance over the long term requires both speed and endurance - that's why the Indianapolis 500 is so tough. It isn't worth much to be the fastest in the race, if you can't keep it running long enough to finish. In computing terms, endurance is a combination of reliable hardware and software, and the total system's ability to recover when something does break.

[SLIDE: Security (2)]

In that sense, the concept of integrity as a security requirement, is just another way of describing endurance for performance. Thus the emphasis on system integrity, as described in these DoD Seminars, is consonant with our experience in computer security. That's one sign of a convergence between the theory and practice.

Given that endurance and integrity are just different views of the same set of requirements, then those hardware and software features which contribute to the endurance of a system are, in fact, contributing to both performance and security. Here's another way to look at it.



[SLIDE: Implementation]

From this viewpoint, we can see how security and performance should be mutually beneficial - synergistic if you will - rather than conflicting goals. How these features are implemented, - and which are in hardware - is where conflicts arise. If security features must be implemented in software - at the expense of performance, - then the software designer is forced to make a tradeoff decision. Historically, the choice has been in favor of performance, simply because that's what sold computers. But that tradeoff is beginning to shift the other way now.

[SLIDE: Hardware Security/Performance]

Specifically, there are four key hardware characteristics which are contribute to both performance and security:

- o Machine Architecture,
- o Memory Protection,
- o Context Switching, and
- o Reliability Features.

Let's look at each, beginning with the architecture.

[SLIDE: Architecture]

This is the general architecture of the Control Data Cyber 170 series computers. What is unique in this block diagram is the Peripheral Processor Units (PPUs) in the middle. These are up to 20 separate, independent computers, which operate concurrently with, but independent of, the Central Processor. Note also that all I/O operations must be performed by the PPUs. Already we see the principle of separation of functions implemented in hardware. I'll come back to the performance aspects of this later. Let's just see how that architecture is reflected in the system design.

[SLIDE: System Layout]

I must explain that only system software executes in the PPUs. In fact, most of the operating system consists of modules to be executed in a peripheral processor. The PPUs also have primary control of the operating system. The one at the top, labeled MTR (Monitor) is the real boss of the system. The executive shown in central memory is just a fast assistant to MTR. User jobs also reside in central memory and only execute in the CPU. Again, we see a separation of functions. When a user program requests I/O, or other services, from a PPU, it validates the request and performs the operation completely independent of the CPU. The CPU program is thus isolated from I/O operations and cannot directly participate in error handling and so

on.

On performance, it should be noted that concurrent operations in the PPU's also means that the software designer need not make a tradeoff between security and performance. While a PPU module is laboriously checking parameters or validating a user's authority to perform an I/O function, the CPU can be producing useful computations for another user. This hardware separation pays off directly in performance, and at the same time, establishes a solid base for security.

Let's move on to memory protection. Actually, memory protection also starts with the architecture. What better isolation can there be than between physically separate memories? Each Peripheral Processor has its own independent memory, separate from the other PPU's, and more important - from Central Memory where users must reside. Again, hardware design provides the separation and isolation necessary for security.

But notice, there are some system tables and software sharing central memory with the user jobs. Here separation is maintained by the CPU memory protection scheme.

[SLIDE: Memory Protection]

This scheme is simply a base and bounds hardware register pair. The Reference Address (RA) is the starting address of memory assigned to an executing program. The Field Length (FL) is the length of that area. These hardware registers are part of the CPU, but are not accessible to the executing program. Their use is completely transparent. To the user, all memory addresses are relative to assigned memory and the hardware precludes any other access. Thus the CPU program does not handle real memory addresses, which is one characteristic in common with virtual memory systems. This eliminates user participation - or observance - of memory management. Since only the Reference Address changes when a program is moved or reloaded into memory, usage can be highly dynamic and efficient. Doing it entirely in hardware provides even greater efficiency, due to the simplicity of the mechanism. Here we see both security and performance as a result of how memory protection is implemented.

[SLIDE: User/System Interface]

Another critical security/performance concern is the need for safe and fast context switching between programs. The actual context switching mechanism is provided by a hardware feature, which has been characterized as the "ultimate interrupt" but officially known as the Exchange Jump operation.

An Exchange Jump can be triggered by either a PPU or a CPU instruction. This single instruction stores the complete set of CPU registers, including RA and FL, into memory and reloads them from the same memory block. Yes, it sounds like magic, but it does go both ways in the same operation. The result is a complete two-way swap of

August 10, 1981

- the execution state of the current CPU program - with the memory image of the state of another program. The whole thing is transparent to the program and the hardware insures that nothing is lost - or gained - in the exchange.

The exchange operation is very fast. For comparison, it is roughly the same time as a floating point divide operation. In some processor models it is even faster. In that case, it could be said that "a swap is faster than a FLOP." Again the intent was performance, but the result is security since it is implemented in the hardware.

A CPU triggered exchange is part of the normal user/system interface. In this case the user program merely relinquishes the CPU to the operating system. On completion of the request, the CPU is returned in a similar manner.

The system PPU monitor however, can independently trigger a context switch at any time. This is how a PPU module can both monitor and control the time-slicing of the CPU among many jobs. It is also the mechanism for "pulling the plug" on programs consuming too much of a resource or hung up, and becoming a "denial of service" threat to others. It effectively eliminates of any form of user lockup, as the PPUs always have the ultimate control. Thus a hardware context switching capability can provide not only performance and security but resource control as well.

[SLIDE: Reliability Features]

Finally, we come to those reliability features usually thought to be interesting only to engineers. Error detection and correction features are the most basic elements of hardware integrity. An adequate set insures that the hardware will yield just two results - either a correct result, or a signal that it cannot perform the function properly. In addition, the diagnostic data produced by these and other maintenance controls contribute to long term stability, reliability and recoverability. My point is that they are not to be overlooked when considering security. We are all aware that most system flaws are exposed when operating in crisis mode - usually in response to an error.

[SLIDE: Hardware Security/Performance (Result)]

In short, there are four key hardware characteristics which contribute heavily to both performance and security:

- o Machine Architecture
- o Memory Protection
- o Context Switching
- o Reliability Features

All of these establish the base on which software must rely, to provide both security and performance in the broad sense.

[SLIDE: Network Operating System]

At this point I should introduce you to our Network Operating System, (N.O.S. or NOS as you will). The name makes it clear that NOS is network oriented. It not only supports access via communications networks, but also supports multi computer networks both locally and remotely. NOS is a multi mode system offering a full range of processing modes including local and remote batch, database managers and transaction processing, and a variety of interactive programming environments. Obviously it is a multi user system as well, and that's where security becomes a key requirement.

[SLIDE: NOS Characteristics]

One of the outstanding characteristics of NOS is that it is a capabilities based system. It all begins with the built-in concept of individual users. Each user must be known to the system, and their capabilities defined on an individual basis. From this is built an accounting system where every activity in the system is attributable and traceable to a user. Users are totally isolated from each other, and the operating system. NOS relies heavily on the hardware separation and memory protection features for this isolation. For NOS users, the only means of sharing data is via the file system. The file system is built around individual ownership of files, and access is, - by default - restricted to the owner. If the owner chooses, other users' access to a file may be specified on the basis of user identity and mode of access. NOS has file passwords too, but they are seldom used since they are independent of identification.

Interestingly, the file system carries the memory addressing concept much further, and exhibits most of the characteristics of a virtual memory system. Space allocation is dynamic, on an as-needed basis, and does not require pre-allocation. That makes it very space-efficient and avoids deadlocks. All I/O references are relative to the logical file name, and the system (a PPU module) does the mapping to real device addresses. Thus, NOS can preclude access outside of a file, and to unwritten space.

Users and their files are also grouped into higher level FAMILYS with no access to files across FAMILY groups. This is particularly valuable in a university environment, to separate students from faculty. Families are then divided into sub-families by storage device to provide further physical separations. The result is that a population of NOS users can be easily managed dynamically and without inconvenience to the user. Both Families and Sub-Families may be controlled as a group via operator commands.

In summary, NOS benefits from both a solid hardware security base, and a design intended for commercial timesharing, which has withstood the test of time and emerged robustly healthy.

[SLIDE: Security Requirements (1 of 2)]

But what of the DoD's security requirements? Although the words may differ, there is a strong similiarity between commercial and government security requirements. When you speak of a kernelized system, it must be as simple a possible - to allow provability - and by definition must be modular. It would be interesting to compare this concept to our system PPU modules. A self-protecting system doesn't fall apart when a user goofs. Though not permissive, it must expect and tolerate user errors. We have already discussed how integrity relates to reliability. User privacy-by-default is a more precise description of isolation, and provides protection from accidental access.

[SLIDE: Security Requirements (2 of 2)]

Actually, access controls are a subset of resource controls. Resource controls also deal with the denial-of-service threat. Controlled sharing is where security is the name of the game, but need-to-know access controls are only one form of control. Access based on the identity of the user, and control based on ownership is another. Auditability is of course, more narrowly directed toward resource accountability. But it also provides a very effective user surveillance capability.

The one listed government security requirement without a commercial equivalent is the concept of security levels and categories. Actually, they are just different sets of criteria for the access controls mentioned above. The unique aspect is that levels and categories are independent of data ownership and subject to a mandatory policy. That's the hole we intend to fill.

With all of these similarities, it should not be surprising then, that a system meeting one set of requirements, should be easily adapted to the other. In fact, while adapting the NOS design to support levels and categories - we found that essentially all of the control mechanisms were already in place. The mechanisms only have to be extended to consider levels and categories and the mandatory security policy in the access control decision. It is clear that not only are the requirements similar, but are convergent on a common set of mechanisms. Simply put - form follows function. Thus we believe there is a common, generic set of control mechanisms which can be adapted to specific security policies. There's a bonus too - With those generic mechanisms already in place, we are confident that the Multilevel Security extensions will result in no significant performance degradation.

[SLIDE: NOS Multilevel Security]

With the NOS Multilevel Security extensions, we will have the functional capability to support Multilevel Mode operations. This will be a standard, fully equipped operating system, for use with our large scale, high performance computers. It will be compatible with the full line of CYBER 170 computers, and most predecessor machines. It will offer the full set of standard software products, and will be software compatible with existing NOS user applications. NOS with MLS will also be available not only to new customers, but to installed customers as well, which goes a long way toward the goal of "widespread availability."

That's what we are doing as practitioners of computer security. But how does that relate to the DoD Initiative and the theoretical work?

[SLIDE: Computer Security Approaches]

As you can see, Control Data has been approaching computer security from a practitioner's viewpoint. Our first concern has to be functional requirements, since we are selling not just hardware and software but capabilities. Design evolution recognizes the fact that we must maintain compatibility with previous systems and the user's applications. Marketability is, in fact, not the least concern, but the one driving all other concerns.

From a theoretical approach, it is clear that computer security must begin with the design methodology, with the objective being provability. The idea of a formal evaluation and on-the-shelf certification is also important, and a pragmatic concern as well. But what really drives a manufacturer is marketability. In this case, it seems our concerns are markedly different. But let's look at the respective methodologies to see if that difference holds up on examination.

[SLIDE: Development Methodologies]

Here we can compare the formal design methodologies with those used by informal practitioners like Control Data. Obviously, both processes begin with some form of requirements. Formally, the security model serves as a target requirement. But as usual, a manufacturer is driven by market requirements, which are often conflicting and subject to internal constraints as well. Eventually, requirements are agreed upon and functional specifications are created. These are roughly equivalent to Top Level Specifications and here the two processes are very similar. In the formal process, the specs are then verified to the security model, while informally a Design Review occurs. A Design Review can be just as tough to do as a logical verification, and a lot more emotional. Where a detailed design is done formally, coding specs emerge informally. Now formal design correspondence may be compute-intensive, but peer review of all generated code is people-intensive. We're not sure which is more expensive, but neither is cheap! We have been told that complete code verification is beyond the

state-of-the-art, well complete system testing may be also - but we keep on trying. In penetration analysis we are doing essentially the same thing. At Control Data, we call it Malicious User Group or MUG system testing. Its fun, and occasionally very exciting! Finally, there is an evaluation of the resulting system by someone whose opinion is important to the developer. For commercial systems, it is simply market acceptance by the user. It would be nice however, to have a formal stamp of approval before shipping the system.

The objectives of these methodologies differ markedly however. For formal methodologies, it is Provability, but for commercial systems it is Functionality. In most other respects they are not only similar, but appear to Converge on a common set of developmental functions.

This convergence has encouraged Control Data to look into applying some of these formal methods to our system. As a first step in that direction, we have requested a DoD evaluation of our NOS system and Multilevel Security design. That process is underway, and so far it looks very promising. On the matter of formal design verification, we understand the benefits, but will have to develop the means of applying the theory to our practices. We are currently exploring some alternatives in that area.

[SLIDE: DoD Initiative Impact]

In conclusion, we at Control Data applaud the progress of the DoD Computer Security Initiative. We would especially like to congratulate you -

- o On increasing industry awareness of the need for security. (Some non-DoD people have helped too - by getting caught.)
- o We thank you for fostering - and occasionally funding - the development of computer security technology.
- o Thanks too, for focusing computer security requirements for those not so knowledgeable in computer security. This directly benefits manufacturers by limiting the ingenuity of those who write technical specifications for procurements.
- o And finally, we thank you for providing an evaluation framework which places greater emphasis on functional capabilities than on technical specifications.

We look forward to a fruitful dialog on our common objectives of advancing the state-of-the-art, and achieving the widespread availability of Trusted Computing Bases.

Thank you for the opportunity to address this forum.

[SLIDE: CDC Logo or DoD slide]



# CONTROL DATA CORPORATION

## TOPICS

- CONTROL DATA AND SECURITY
- SECURITY AND PERFORMANCE
- ARCHITECTURE AND SYSTEM DESIGN
- DOD AND DEVELOPMENT METHODS
- DOD INITIATIVE

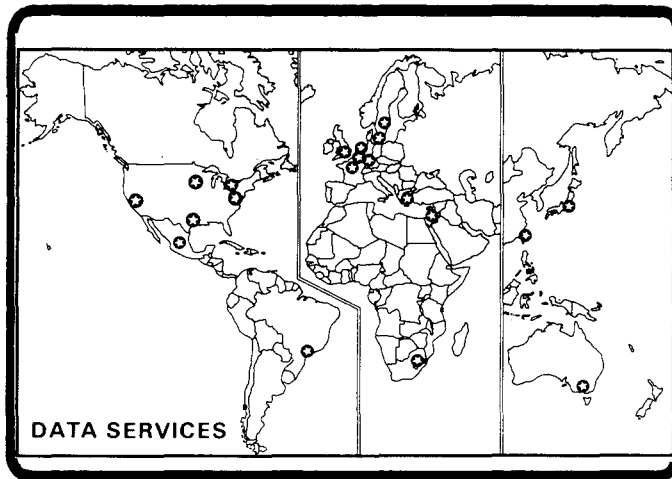
## CONTROL DATA REPUTATION

- LARGE SCALE COMPUTERS
- SCIENTIFIC/ENGINEERING
- SINCE 1957



## **CONTROL DATA TODAY**

- **SUPER SCALE COMPUTERS**
- **PERIPHERALS**
- **EDUCATION – PLATO**
- **DATA SERVICES**



## **DATA SERVICES**

**CUSTOMERS ALWAYS ASK:**

**"HOW MUCH DOES IT COST?"**

**"HOW SECURE WILL MY DATA BE?"**

## **SECURITY**

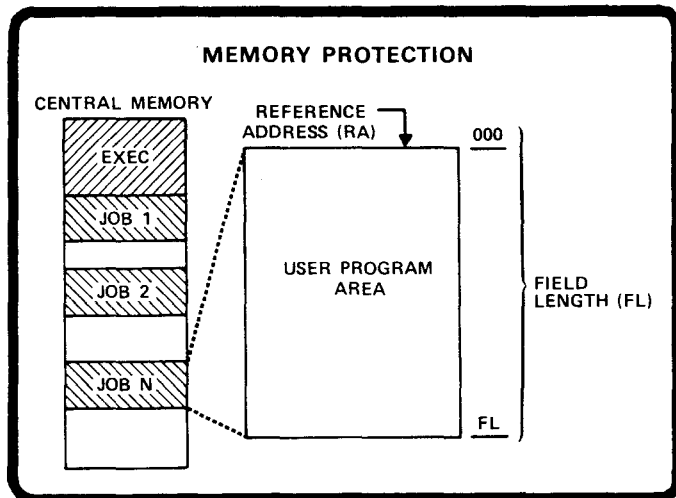
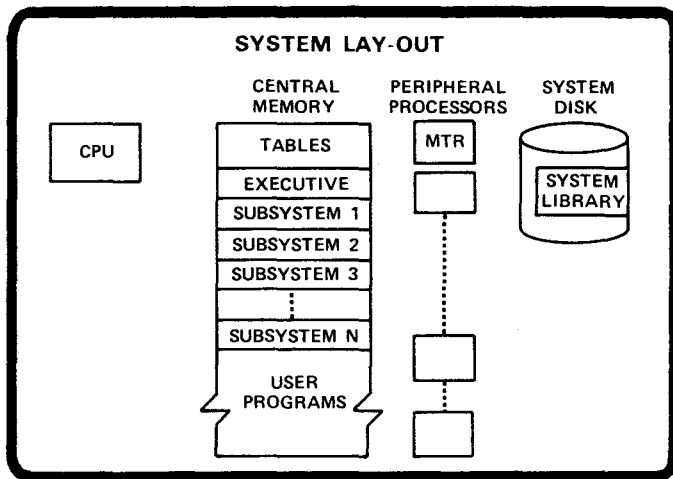
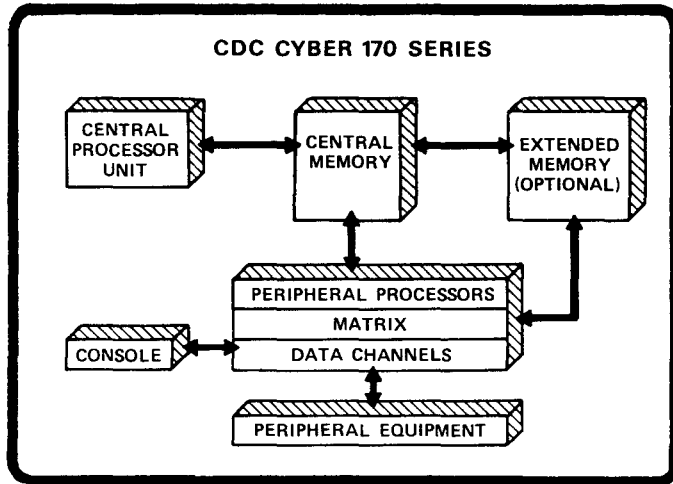
- CUSTOMER CONCERN
- BUSINESS NEED
- 20 YEAR HISTORY
- OUR LARGEST MARKETPLACE

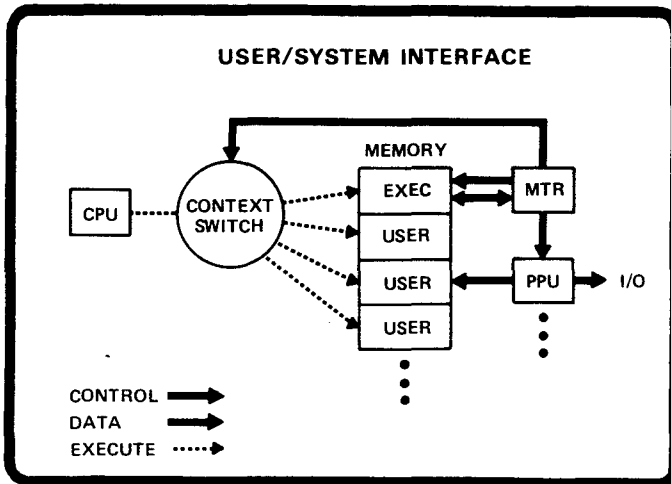
## **TIMESHARING ENVIRONMENT**

- SYSTEM INTEGRITY
- MULTI USER
- ASSETS PROTECTION
- RESOURCE CONTROLS
- ACCOUNTABILITY
- USER ISOLATION

## **PERFORMANCE**

- COMPUTE POWER
- ENDURANCE
  - RELIABILITY
  - RECOVERABILITY





### RELIABILITY FEATURES

- ERROR DETECTION/CORRECTION
  - CORRECT RESULT
  - ERROR SIGNAL
- MAINTENANCE FEATURES
  - HARDWARE CONTROLS
  - DIAGNOSTIC DATA

### HARDWARE SECURITY/PERFORMANCE

- MACHINE ARCHITECTURE
- MEMORY PROTECTION
- CONTEXT SWITCHING
- RELIABILITY FEATURES

RESULT: SECURITY WITH  
PERFORMANCE

## **NETWORK OPERATING SYSTEM**

- NETWORKS
- MULTI COMPUTERS
- MULTI MODE
- MULTI USERS

## **NOS CHARACTERISTICS**

- CAPABILITIES BASED
- USER CONCEPT
- ACCOUNTING CONTROLS
- USER ISOLATION
- FILE SYSTEM
- LOGICAL FILE I/O
- FAMILY OF USERS CONCEPT

## **SECURITY REQUIREMENTS**

### COMMERCIAL SYSTEMS

- SIMPLICITY, MODULARITY
- FAULT TOLERANT
- RELIABILITY
- USER PRIVACY

### GOVERNMENT SYSTEMS

- KERNELIZED
- SELF PROTECTING
- INTEGRITY
- ISOLATION

## SECURITY REQUIREMENTS

### COMMERCIAL SYSTEMS

- RESOURCE CONTROLS
- CONTROLLED SHARING
- AUDITABILITY

### GOVERNMENT SYSTEMS

- ACCESS CONTROLS
- NEED-TO-KNOW ACCESS
- SURVEILLANCE
- LEVELS/CATEGORIES

## NOS MULTILEVEL SECURITY

- STANDARD SYSTEM
  - LARGE SCALE SYSTEMS
  - PERFORMANCE
- COMPATIBILITY
  - HARDWARE
  - SOFTWARE
- AVAILABILITY
  - NEW CUSTOMERS
  - INSTALLED CUSTOMERS

## COMPUTER SECURITY APPROACHES

### THEORETICAL

DESIGN METHODOLOGY  
DESIGN VERIFICATION  
FORMAL EVALUATION

### PRACTICAL

FUNCTIONAL REQUIREMENTS  
DESIGN EVOLUTION  
MARKETABILITY

## DEVELOPMENT METHODS

### THEORY

SECURITY MODEL  
TOP LEVEL SPECIFICATIONS  
DESIGN VERIFICATION  
DESIGN CORRESPONDENCE  
CODE VERIFICATION  
PENETRATION ANALYSIS  
FORMAL EVALUATION

### PRACTICE

MARKET REQUIREMENTS  
FUNCTIONAL SPECIFICATIONS  
DESIGN REVIEW  
PEER REVIEW OF CODE  
UNIT/SYSTEM TESTING  
IN-HOUSE USE/TESTING  
USER ACCEPTANCE

## DOD INITIATIVE IMPACT

- AWARENESS
- TECHNOLOGY STIMULUS
- FOCUS FOR REQUIREMENTS
- EVALUATION FRAMEWORK

SAC Digital Network  
(SACDIN)  
Security Methodology

Mauro Ferdman  
The MITRE Corporation

PRESENTATION OUTLINE

- Slide 1 : SACDIN will be used to support command and control functions of the Strategic Air Command.
- Slide 2 : Present status of the project is full-scale engineering development. Prime contractor is ITT and the major subcontractors are IBM for software and BDM for systems.
- Slide 3 : SACDIN is a large scale network covering all SAC units throughout the continental U.S. It is a packet-switched network and it uses AUTODIN II as a backbone. One of the characteristics of SACDIN that is important for this seminar is that it is designed to be mutli-level secure.
- Slide 4 : The security requirements are very strict and as it was mentioned before, they include requirements for simultaneous transmission of messages of different classification. It provides protection against compromise of information, integrity and denial-of-service.
- Slide 5 : The IACM provides total mediation between subjects, which are the users of information, and the objects which are the repositories of information. The IACM mediates every single access of subjects to objects.
- Slide 6 : The IACM mediates all accesses so it must be some assurances that it was designed and implemented correctly. This has required that a specialized software design methodology be used and it will be described later. In addition, there must be some ways of protecting the IACM from being altered by other software.
- Slide 7 : SACDIN has three tiers of protection provided by the applications processes which are used for user support, the trusted processes, which are used for I/O Interfaces and the IACM or Internal Access Control Mechanism, which also serves as the Operating System. The next slides will explain in more detail the security enforcement mechanism of the IACM.
- Slide 8 : The methodology used for development of the IACM consisted of creating a mathematical model to formally represent DOD security policies, followed by a formal description of the IACM design in a formal language which was formally verified not to violate the math model. Lower level specifications were only correlated in a less formal way.



- Slide 9 : In a more graphical way, the bottom line shows the standard DOD Procurement practices for software, from user requirements to code, with the proper test and evaluations. Our methodology has added the upper part in parallel to provide a better assurance of a correct design.
- Slide 10: We quickly found that the IACM by itself was not enough to protect against compromise. There were problems in these cases where information must be transferred into or out of the Central Processor, such as network communications, peripheral devices, etc. The following slides will deal with these problems.
- Slide 11: A host or node contains an IACM and it is fully capable of handling multi-level communications such as from A to B or access to the Multi-Level Data Base, marked as MLDB in the slide.
- Slide 12: If we have a network, and now A attempts to communicate with C or B with D, we are dealing with multiple IACM's, one in each node, so it is important that the last software process that handled the message be trusted.
- Slide 13: The situation is more complicated through the use of a back-bone. See in the slide the path from A to C and B to D.
- Slide 14: The solution that we adopted was to create specialized software that serves to authenticate one node to another and to serve as I/O transmission control. It earned the name trusted because it used the same design methodology as the IACM.
- Slide 15: The problem with peripheral devices are similar, because the IACM does not have direct control of the information going outside the Central Processor. The solutions adopted were similar to the ones adopted for communications, namely to use trusted software to handle the printer and user interface.
- Slide 16: As far as integrity protection is concerned, it was based on using good software practices as shown in the slide.
- Slide 17: The Central Processor that we used is a modified off-the-shelf computer, the IBM Series/1, with several security features added. They consisted of an expanded relocation translator, a security controller to monitor accesses to core and an expanded instruction set. The terminal was specially developed and it includes a special security field.
- Slide 18: Summary and conclusions.
- Slide 19: Lesson learned.

**SAC Digital Network  
(SACDIN)**

**Security Methodology**

**Mauro Ferdman  
MITRE Corporation**

**Purpose**

---

**Provide Data Communications Support  
for Command and Control of SAC Forces**

**Status**

---

**Presently Under Full-Scale Engineering  
Development**

**Prime Contractor: ITT**

**Main Subcontractors: IBM, BDM**

## **Characteristics**

---

**Large Scale Network (About 200 Nodes)**  
**Packet-Switched Network**  
**Uses AUTODIN II As Backbone**  
**Multi-Level Secure**

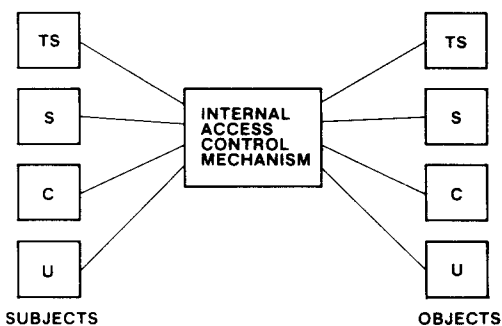
## **Security Requirements**

---

**Strict Overall Security Requirements**  
**Multi-Level Capabilities**  
**Compromise, Integrity and Denial of Service Protection**

## **Security Architecture**

---



## Internal Access Control Mechanism (IACM)

Mediates All Access

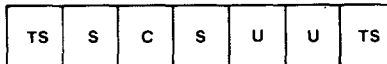
Formally Proven Secure

Protected From Modification

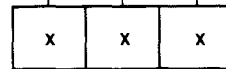
Serves As OS

## Software Architecture

APPLICATION PROCESSES



TRUSTED PROCESSES



IACM

INTERNAL ACCESS CONTROL MECHANISM

HARDWARE

PERIPHERAL DEVICES AND INTERFACES

## IACM Development Methodology

**Formally Represent Security Policies (Math Model)**

**Prepare Formal Top Level Specifications (B-5)**

Formally prove specifications

**Intermediate Language Representation**

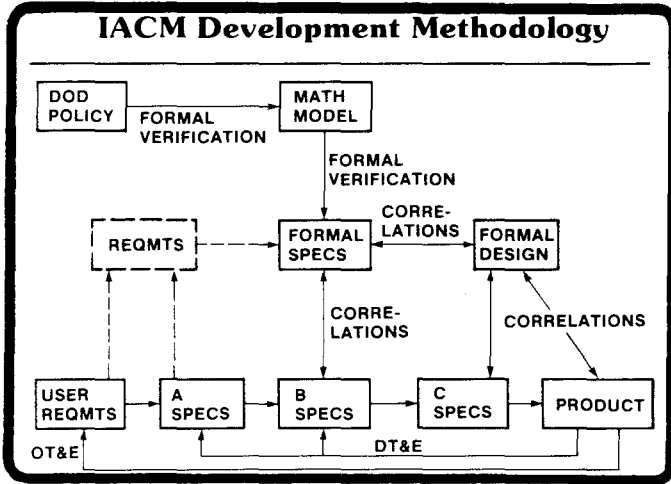
Correlations proofs

**Stepwise Refinements**

Correlation proofs

**Implementation Code**

Correlations proofs



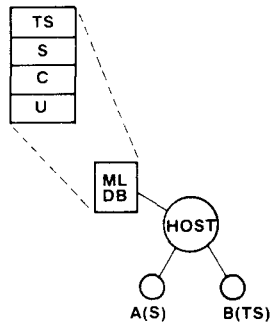
### IACM Not Enough To Protect Against Compromise

Network Problems

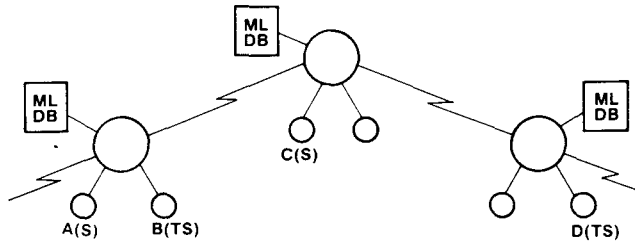
Peripheral Devices Problems

Multi-Level Files

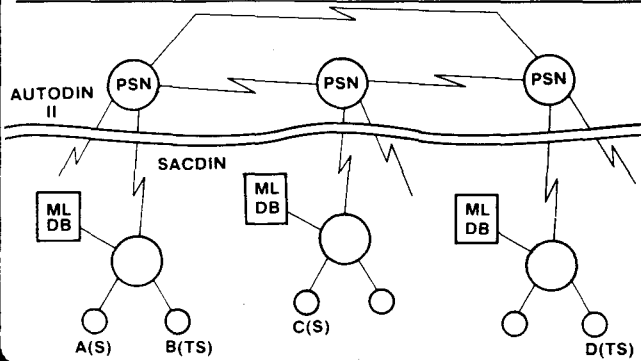
### Multi-Level Problems In Networks Host Problem



### Multi-Level Problem In Networks SACDIN Lines



### Multi-Level Problems in Networks SACDIN/AUTODIN II Links



### Network Solutions

#### Trusted Software Required For

- Node authentication
- Output transmission control
- Input transmission control

### **Other Trusted Functions**

**Printer Manager**  
**User Interface**  
**File Manager**

### **Integrity Protection**

**Single Computer Program Architecture**  
**Top Down Design/Structured Programming**  
**Strict Accountability and Journaling of Messages**  
**Error Detection Mechanisms**

### **Hardware Security Features**

**Node's Central Processor**  
Modified IBM/Series I processor  
Relocation translator  
Security controller  
Extended instruction set  
**Specialized Terminal**  
Special security fields

## **Summary**

---

**SACDIN Is First Multi-Level Network  
With Strict Security Requirements From  
Program Inception**

**Uses Specialized Software Development  
Methodology Reaching As Far As the  
Practical State-of-the-Art Will Go**

**Thorough Security Analysis Throughout  
Design and Development**

**Collaborative Effort**

## **Lessons Learned**

---

**Large Amount of Trusted Software  
Required Over and Above Basic Kernel**

**Largest Security Problem Is the Handling  
of Peripherals and Communications  
Lines, Not the Internal Handling of Data**

**Multi-Level Security Can Be Achieved If  
System Is Carefully Planned, Designed  
and Developed**



# **COS/NFE OVERVIEW**

Gary Grossman  
Digital Technology Incorporated

August 10, 1981

DTI

## **Preview**

- COS/NFE Program
- COS/NFE Technical Description
- HUB™ Executive
- Security Methodology
- Experience

DTI

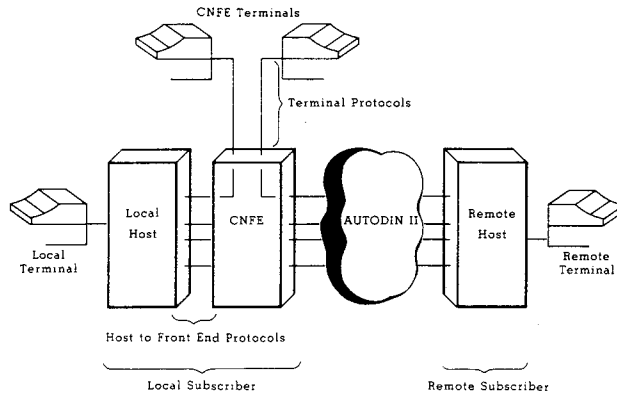
**C**ommunication  
**O**perating  
**S**ystem  
/  
**N**etwork  
**F**ront  
**E**nd

DTI

COS/NFE

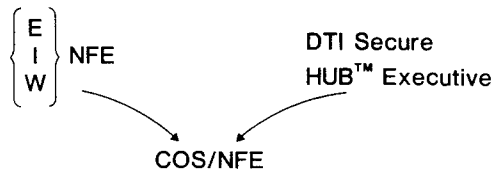
- Verifiably secure
- Prototype NFE
- For AUTODIN II

DTI



81-11

Lineage



DTI

### Precursors

ENFE	Network UNIX + IIPC	ARPANET	U of I
INFE	UNIX + Attach I/O	AUTODIN II	DTI
WNFE	UNIX + Attach I/O	WIN	DTI

DTI

### Participants

- DCA - Sponsor
- DTI - Prime design & implementation
- SDC - Sub formal specs., verification, & security analysis
- ISET - Security Watchdog

DTI

### Goals

- Security
  - Overt channels
  - Covert channels
  - Denial of service
- Performance
  - "Significantly" better than INFE

DTI

**Bases**

- Hardware - PDP-11/70
- Software - Secure HUB™ Executive PASCAL

DTI

**Schedule**

- Completion - March '83
- Trusted security control - soon

DTI

**COS/NFE Functions**

- Identical to INFE + security
- Interfaces
- Protocols

DTI

**COS/NFE Interfaces**

- AUTODIN II  
ACC UMC-Z80
- WWMCCS H6000  
ACC LH/DH-11 - ABSI
- Terminals  
DH-11 Asynch  
DV-11 Synch VIP

DTI

**COS/NFE Protocols**

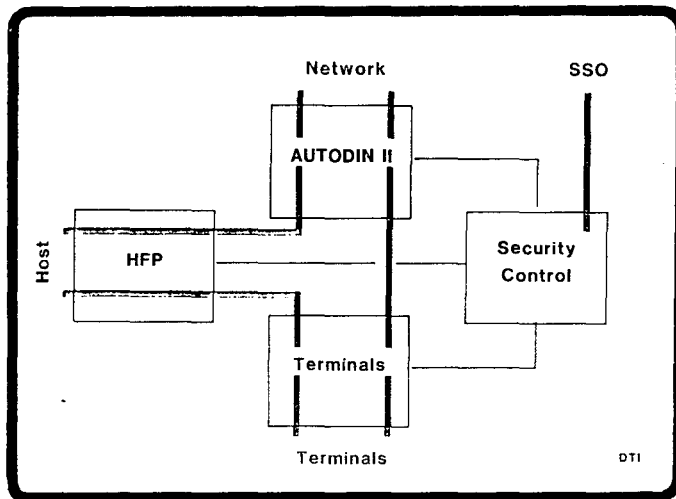
- AUTODIN II  
THP, TCP, IP, SIP, Mode VI
- WWMCCS H6000  
HFP: SAP's, Channel, Link
- Terminals  
Asynch      Character/Start-stop  
Synch        Honeywell VIP

DTI

**COS/NFE Modules**

- Protocol processing  
From INFE
- Admin. & security  
CI      - designed, coded  
TH      - designed, coded  
Others - designed, being coded
- HUB™ Executive  
Designed, coded, tested to usefulness  
for measurements

DTI



**COS/NFE Security Policy**

- Preserves security labelling
- Level (data)  $\subseteq$  level (line)

DTI

**COS/NFE Multi-level Users**

- AUTODIN II
- Terminals
- (Hosts)  
(Modifications to HFP)

DTI

### Secure HUB<sup>TM</sup> Executive

- Stand-alone
- Verifiably secure
- Communications-oriented
- Portable  
PASCAL, 11/70, 11/780,  $\mu$ P
- Proprietary

DTI

### HUB<sup>TM</sup> Security Policy

- Separation of Domains
- Flexibility in supporting more sophistication

DTI

### HUB<sup>TM</sup> Sizes (PDP-11)

- 2838 lines of PASCAL
- 236 routines
- 32 primitives
- 1200 lines of assembler (bootstrap, dump)
- 70K bytes on PDP-11 ( $\approx$  54K code,  
 $\approx$  16K data)

DTI

**Relative Speed of IPC Operations**

- Includes all related primitive calls
  - Buffer allocation
  - Sending message
  - Receiving message
  - ⋮
- Attach I/O 6.95ms
- HUB™ 3.9ms
- Ratio 1.75
- Functions may not be comparable DTI

**HUB™ Primitives**

• Resource management	10
• Process management	3
• Address space management	2
• IPC	7
• Flow control	5
• I/O	4
• Timing	1
	<hr/>
	32 DTI

**HUB™ Concepts**

- Stages (processes)
- IPC
- Sessions (domains)

DTI



### HUB™ Stages

- Program - sharable
- Memory
  - Private - unshared
  - Buffers - serially shared
- Ports for IPC

DTI

### HUB™ IPC

- Connections between ports
- Via buffers only

DTI

### HUB™ Sessions

- Execution control
- Connection control
- Resource control

DTI

**HUB™ Security**

- Overt channels
- Covert channels
- Denial of service

DTI

**Overt Channels**

- Formal control
- Definition of "trusted"
- Communication rule
- Execution rule

DTI

**Covert Channels**

- Engineering
- Few shared resources
- Strict controls on resources
- Only trusted software can move resources

DTI

Denial of Service

- Engineering
- Similar to covert channels

DTI

Security Methodology

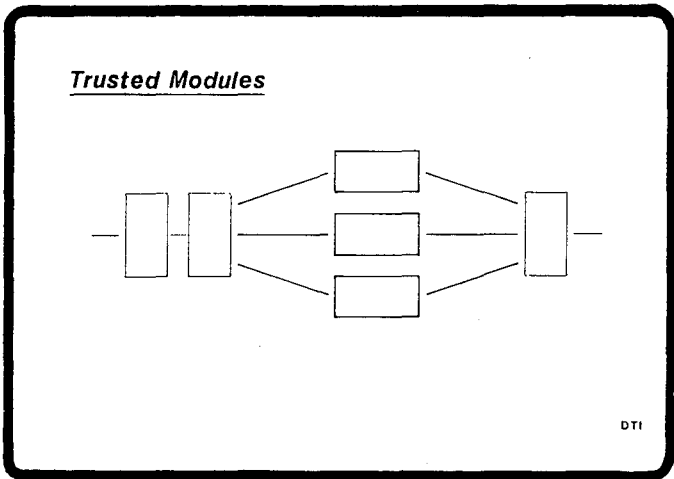
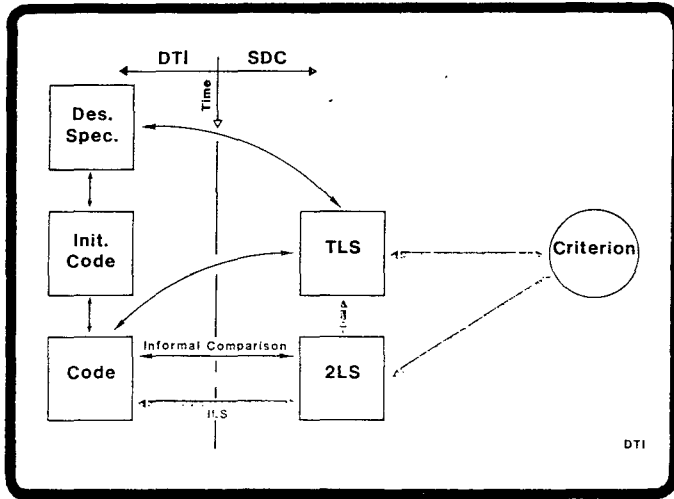
- Verification plan
- ISET evaluation
- Overt channels – formal verification
- Covert channels – engineering analysis & solutions
- Denial of service – same

DTI

Overt Channels

	<u>DTI</u>	<u>SDC</u>
• Identify trusted modules	X	
• Correctness criterion	X	X
• Write & prove TLS		X
• Write & prove 2LS		X
• Compare 2LS & code	X	

DTI



- Correctness Criterion
- One for each trusted module
  - Relatively simple
  - Security-related only
- DTI

### Top-Level Specification

- Correctness criterion
- Initial conditions
- Variables
- Transforms

DTI

### Second-Level Specification

- Mapping to TLS
- Refinement
  - Variables
  - Transforms

DTI

### Comparison of 2LS & Code

- EG: HUB™
  - 2LS: 2400 lines of INA JO
  - Code: 2838 lines of PASCAL

DTI

**Covert Channels**

- SDC analysis
- Identify channels
- Construct scenarios
- B/W  $\leq$  5000 baud worst case  
20 baud typical
- Limited by engineering

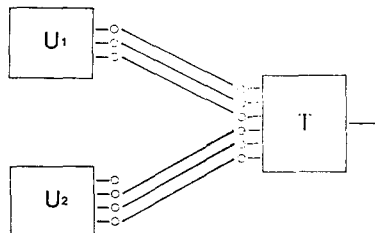
DTI

**Identify Channels**

Resource Attribute	Primitive	READ	WRITE	SEEK	CREATE FILE	DELETE FILE
Files	Existence				<i>RW</i>	<i>RW</i>
	File Length	<i>R</i>	<i>W</i>	<i>R</i>	<i>W</i>	
	Current Location	<i>RW</i>	<i>W</i>	<i>RW</i>	<i>W</i>	
Disk Device	Arm Position	<i>RW</i>	<i>RW</i>			
	Free Space		<i>RW</i>		<i>R</i>	<i>W</i>

DTI

**Scenario**



DTI

### Performance Experiment

- TCP: HUB™ vs. INFE UNIX™
- Security with HUB™  
Resource allocation: 46% of CPU
- More IPC with HUB™
- HUB™: PASCAL; UNIX: C
- HUB™ 17% faster

DTI

### Security Experience

- SDC analysis
- ISET evaluation
- Proof of HUB™ 2LS, 2000 pages

DTI

### Things to Come

Verification continuing  
INFE protocols to HUB™  
HUB™ to other processors

DTI

*Examples from  
HUB<sup>TM</sup> Executive Top Level Specification  
in INA-JO<sup>TM</sup>*

*INA-JO is a Trademark of System Development Corporation*

DTI

**HUB<sup>TM</sup> Executive Security Criterion**

*From HUB Executive INA-JO<sup>TM</sup> Top Level Specification*

Criterion

```
A"B:BUFFER,SESS:SESSION(
  B <: BUFFERS_OF(SESS)
-> SLS_OF_BUFFER(B) <<= SLS_OF_SESSION(SESS))
& A"E:BUFFER,DEV:DEVICE(
  B <: DEVICE_BUFFERS(DEV)
-> SLS_OF_BUFFER(B) = SLS_OF_DEVICE(DEV))
& A"P:SEPS,SESS:SESSION(
  P <: SET_OF_SEPS_OF(SESS) -> DOMINATES(P,SESS))
```

DTI

**HUB<sup>TM</sup> Executive Initial Condition**

*From HUB Executive INA-JO<sup>TM</sup> Top Level Specification*

Initial

```
A"SESS:SESSION(
  (SESS = ADM & ACTIVE_SESSION(ADM)
  | -ACTIVE_SESSION(SESS)
  & SET_OF_SEPS_OF(SESS) = EMPTY)
  & BUFFERS_OF(SESS) = EMPTY)
& A"DEV:DEVICE(
  -ACTIVE_DEVICE(DEV) & DEVICE_BUFFERS(DEV) = EMPTY)
& A"P:SEPS, SESS:SESSION(
  P <: SET_OF_SEPS_OF(SESS) -> DOMINATES(P,SESS))
```

DTI



**HUB™ Executive Transform Communicate**

*From HUB Executive INA-JO™ TOP LEVEL Specification*

TRANSFORM COMMUNICATE(B:BUFFER,SI,SJ:SESSION)  
EXTERNAL EFFECT

```
SLS_OF_BUFFER(B) <<=
  SLS_OF_SESSION(SI) && SLS_OF_SESSION(SJ)
& B <: BUFFERS_OF(SI)
& ACTIVE_SESSION(SJ)
& SI = SJ
& A"SESS:SESSION(
  N"BUFFERS_OF(SESS) =
    (SESS = SI => BUFFERS_OF(SESS) -- S"(B)
    <> SESS = SJ => BUFFERS_OF(SESS) || S"(B)
    <> BUFFERS_OF(SESS)))
| NC"(BUFFERS_OF)
```

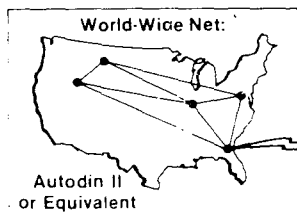
DTI

# WIS Security Strategy

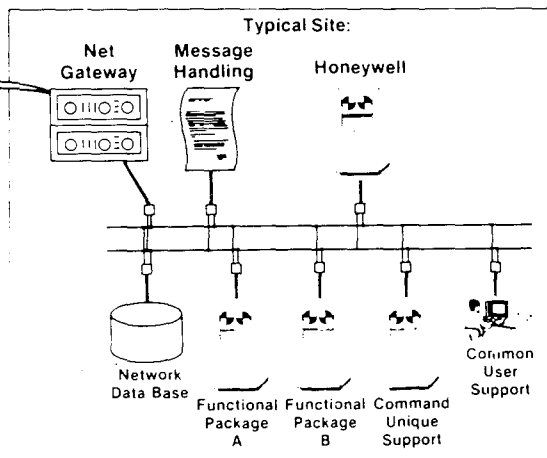
Larry Bernosky  
Defense Communications Agency  
WWMCCS System Engineering

## WWMCCS Information System: Target Architecture

Goal: Reliable Connectivity Among All Sites



Goal: Improved Local Flexibility and User Support



## **DOD Security Regulations**

- DOD Directive 5200.28
- JCS Publication 22
- Army Regulation 380-380
- DIA Manual 50-4
- ...
- ...
- ...

## **Current Security Control Techniques**

- System High
- Dedicated Systems
- Periods - Processing

## **Characteristics of Current Controls**

- **Static**
- **Long Lead Time to Implement**
- **Expensive**
- **Limited Extensibility**

## **WWMCCS Environment Trends**

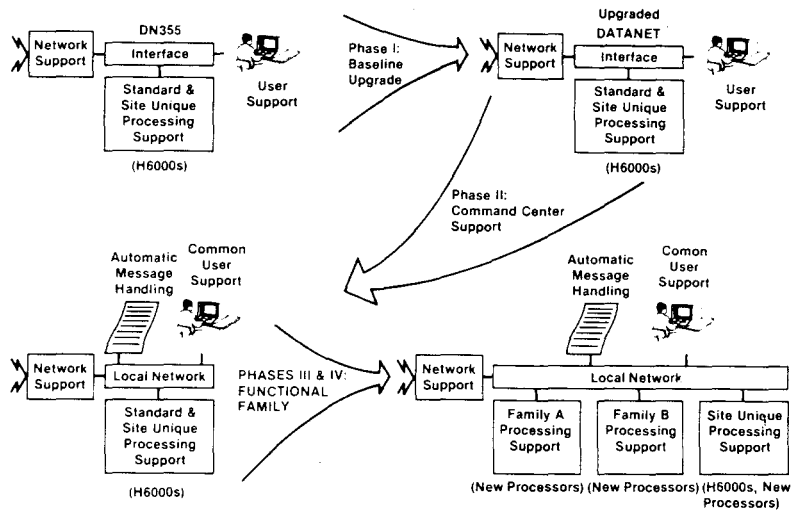
- **Increasingly Complex Processing Needs**
- **Extensive Internetting and Intranetting**
- **Evolution Toward Distributed Control**
- **Temporary Reliance on Monolithic Machines**

## WIS Security Goals

**Objective: Provide "Adequate" Security for WIS**

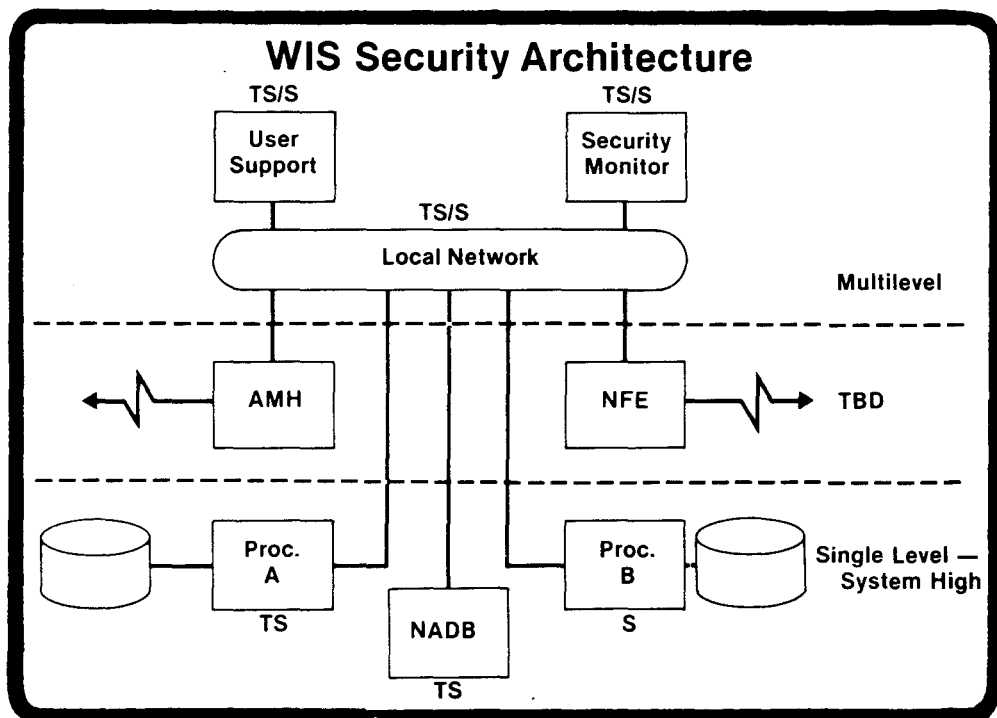
- Satisfy the Security Policy
- Allow WIS to Perform Its Required Functions
- Make Controls Transparent to the User
- Allow for Evolutionary Upgrades

## ARCHITECTURE PHASES



## Security Architecture Overview

- Develop General Scenarios
- Summarize Requirements for Specific Components
- Derive Security Architecture
- Overlay Scenarios on WIS Architecture



## **Security Architecture Components**

### **Category 1**

- **Local Network (Multilevel Mode Essential)**
- **User Support System (Multilevel Mode Essential)**
- **Security Monitor (Multilevel Mode Essential)**

### **Category 2**

- **Automated Message Handler (Multilevel Mode Desirable)**
- **Long Haul Network (Multilevel Mode Desirable)**
- **Network Front End (Multilevel Mode Desirable)**

### **Category 3**

- **Applications Processors (System High/Dedicated)**
- **Data Base (System High/Dedicated)**

## **General WIS Security Principles/Assumptions**

- **Not All WIS Components Need Be Multilevel Secure**
- **Priority Attention to Multilevel Secure Local Network**
- **Multilevel Security Required Only Over Limited Range of Security Levels (Controlled Mode)**

## **WIS Operational Scenarios**

- 1) **Support for Homogeneous User Access**
- 2) **Support for Low to High User Access**
- 3) **Support for High to Low User Access**
- 4) **Message Receipt and Distribution to WIS Users**
- 5) **WIS Multilevel Long-Haul Connections**

## **Support for Homogeneous User Access**

### **Description**

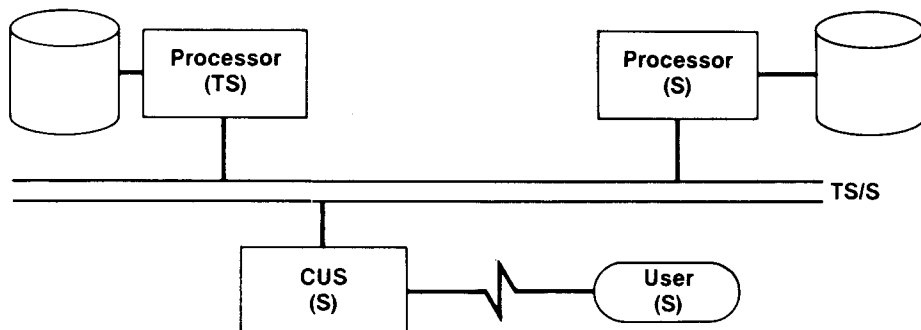
- **Secret Remote User Requires Access to Data on a Secret Processor**

### **Requirements**

- **Local Network Needs to Support Remote Terminals**
- **Local Network Needs to Support Communication Between Devices at the Same Security Level**
- **Local Network Needs to Maintain Separation of Data Having Different Security Classifications**



## Support for Homogeneous User Access



- User (S) Requests Access to Data on a Secret Processor
- CUS (S) Validates User Identity and Access Request
- CUS (S) Forwards Request to Processor (S) via Trusted Multilevel Local Net
- Processor (S) Validates Request and Forwards Data to CUS (S)
- CUS (S) Queues Data for User (S)

## Support for Low to High User Access

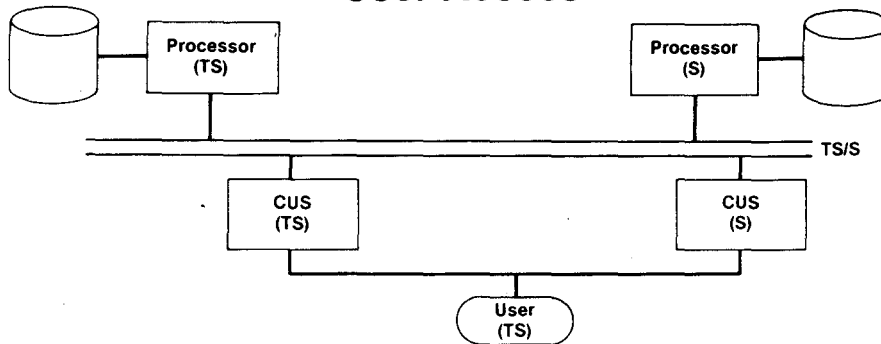
### Description

- Confidential (or Secret) Remote User Requires Access to Selected Data from a TS System High Processor
- Access Control Mechanism is Needed to Screen Request and Validate User Identity
- Information from TS Processor Must be Reviewed/Sanitized Before Delivery to Low User

### Requirements

- Local Support is Needed for Users at Different Classification Levels
- Local Network Needs to Support Remote Terminals
- Data Base Needs to Contain Material at Different Classification Levels
- Data Base Needs to be Accessed by Authorized Users Having Differing Security Clearances

## Support for High to Low User Access



- User (TS) Initially Connected to Top Secret CUS
- User (TS) Disconnects (Physical Switch or Trusted SW) from CUS (TS) After Storing Working Files in CUS (TS)
- User's Terminal is Sanitized Automatically
- User (TS) Connects to Secret CUS
- Access to Secret Processor is Made via CUS (S)
- User (TS) May Switch Back to Top Secret CUS Without Sanitizing Terminal

## Support for High to Low User Access

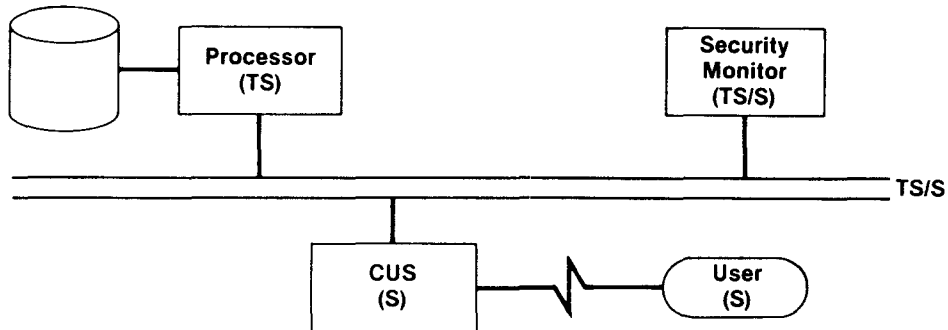
### Description

- Top Secret User Requires Access to Data on a Secret Processor
- Secret Data is Released to Top Secret User

### Requirements

- CUS Needs to Provide Multilevel Support for a TS User
- Mechanism is Needed to Prevent Release of TS Data into an S Environment
- User Performance Must Not be Adversely Affected by Security Controls

## Support for Low to High User Access



- User (S) Requests File Controlled by TS System High Processor
- CUS (S) Validates User Identity and Access Request
- CUS (S) Forwards Request to Processor (TS)
- Processor (TS) Validates Request and Forwards File to Security Monitor
- Security Monitor Forwards Reviewed File to CUS (S)
- CUS (S) Queues File for User (S)

## Message Receipt and Distribution

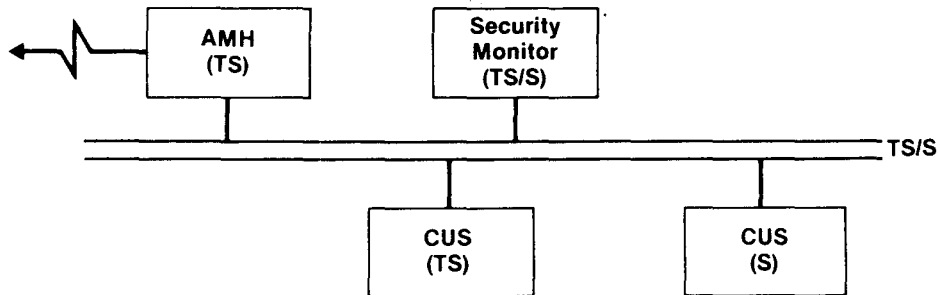
### Description

- AMH Receives Secret Labeled Message Over TS Communication Line for Distribution to Selected Local TS and S Users
- Message Must Be Reviewed/Sanitized Since TS Data May Have Been Mixed with Message on Long Haul Net
- Message is Queued to Common User Support (CUS) for TS and S Users

### Requirements

- Local Network Needs to Maintain Separation of Classified Material While on the Net and When Entering or Leaving the Net
- CUS Needs to Support Terminals Operating at Different Security Levels
- Message Handling and Distribution Functions Need to Support Different Security Levels
- Selected Classified Information Needs to be Reviewed or Sanitized

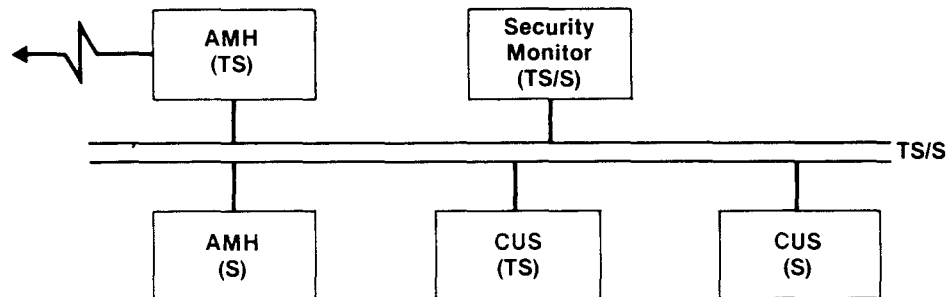
## Message Receipt and Distribution



### Essential 1

- AMH (TS) Receives Message (S)
- Message Queued for TS Users at CUS (TS)
- AMH Forwards Message to Security Monitor With Addresses of S Users
- Security Monitor Queues Reviewed Message at CUS (S)

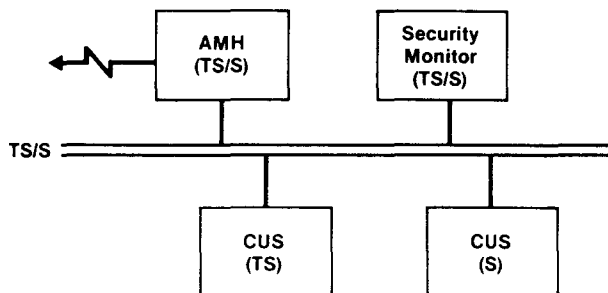
## Message Receipt and Distribution



### Essential 2

- AMH (TS) Receives Message (S)
- Message Queued for TS Users at CUS (TS)
- AMH Forwards Message to Security Monitor
- Security Monitor Sends Reviewed Message to AMH (S)
- AMH (S) Queues Message for S Users at CUS (S)

## Message Receipt and Distribution



### Desirable

- If Message Could Contain TS Data, the AMH Routes Message to Security Monitor
- Security Monitor Sends Sanitized Message Back to AMH
- AMH Queues Message (S) to Both TS and S Users via Appropriate CUS

## WIS Multilevel Long-Haul Connections

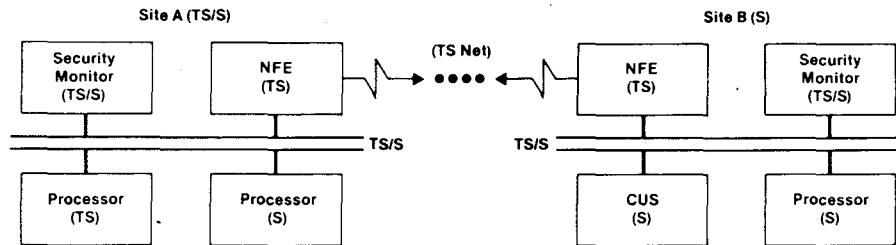
### Description

- Two WIS Sites Operate at Different Maximum Security Levels
- Sites Need to Exchange Information
- TS to S Message Flow Must be Reviewed/Sanitized

### Requirements

- Long-Haul Network Needs to Support Local Users Operating at Different Security Levels
- Long-Haul Network Needs to Connect WIS Nodes Having Different Ranges of Classified Information
- Material with Different Classification Levels Needs to be Transmitted Over the Long-Haul Network

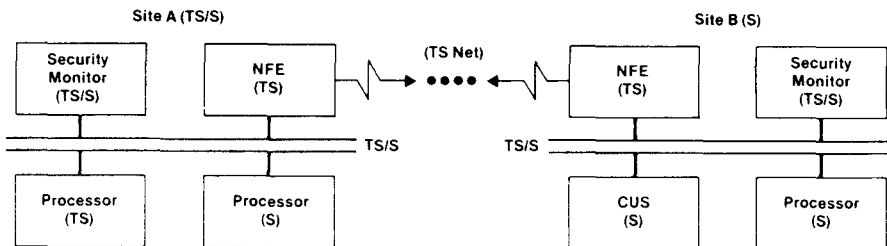
## WIS Multilevel Long-Haul Connections



### Essential 1

- Site B (S) Requests File from Site A (TS/S) via NFE (TS)
- If File at A is on TS Processor then File is Passed Through Security Monitor at A to Verify File Contents Are at S Level
- Security Monitor Forwards File (S) to NFE (TS)
- NFE (TS) at B Receives File Which is Sent to Security Monitor at B to Verify That No Modifications Occurred on the Long Haul Net (Perhaps by a More Rigorous CRC Type Authentication)
- Security Monitor Forwards File (S) to Appropriate Locations on Local Net B

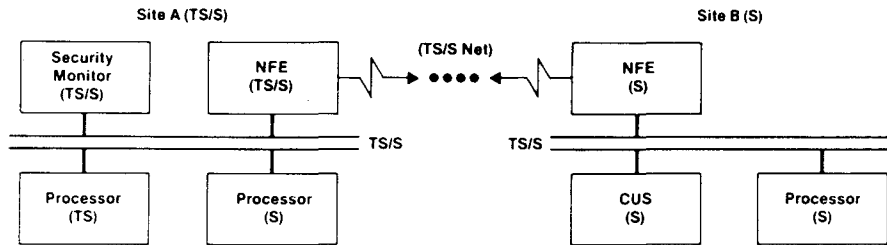
## WIS Multilevel Long-Haul Connections



### Essential 2

- Site B (S) Requests File from Site A (TS/S) via NFE (TS)
- Site A Forwards File to Site B via NFE (TS)
- NFE (TS) at B Forwards File to the Security Monitor at B for Review/Sanitization to S Level
- Security Monitor Forwards File (S) to Appropriate Locations on Local Net B

## WIS Multilevel Long-Haul Connections



### Desirable

- Site B (S) Requests File from Site A (TS/S) via MLS Long-Haul Network and MLS NFE
- If File at A is on TS Processor Then File is Reviewed/Sanitized by Security Monitor at A
- Security Monitor Forwards File (S) to S Portion of NFE (TS/S)
- Long-Haul Net (TS/S) Guarantees File Received at B is S Level
- NFE (S) at Site B Forwards File (S) to Appropriate Location on Local Net B

### Interconnection

- Protocols
- Long-Hauls
- Gateways
- Subnetworks

### Trusted Software

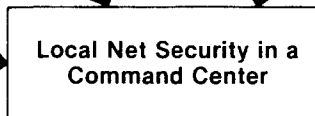
- Verification
- Kernels

### Encryption

- E<sup>3</sup>
- Link

### Policy

- 5200.28
- JCS Pub. 22



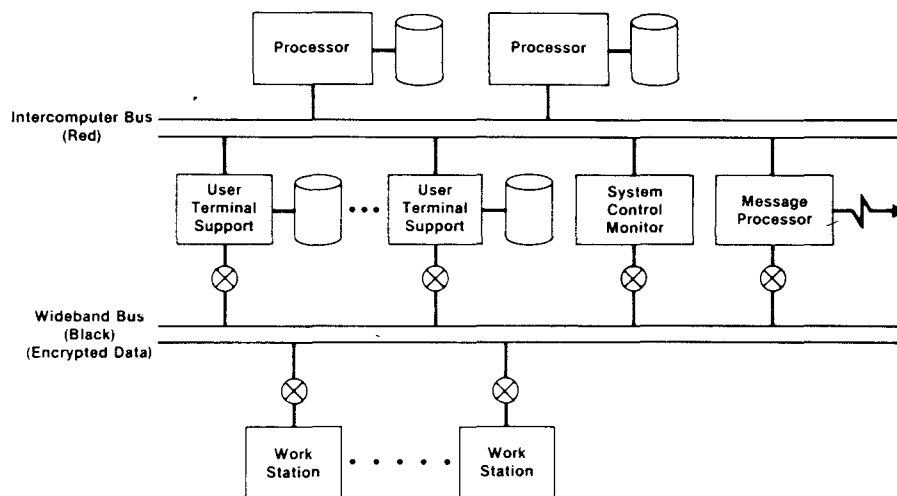
### Recommendations

- Assessment
- Experiments
- Action Items

## Security Flexibilities in the Local Network

- Reduced Need to Share Hardware
- Can Support Several Different (Tailored) Security Approaches
- Use of Specialized Solution Approaches
- Evolutionary Implementation and Upgrade Possibilities

## SAFE Description

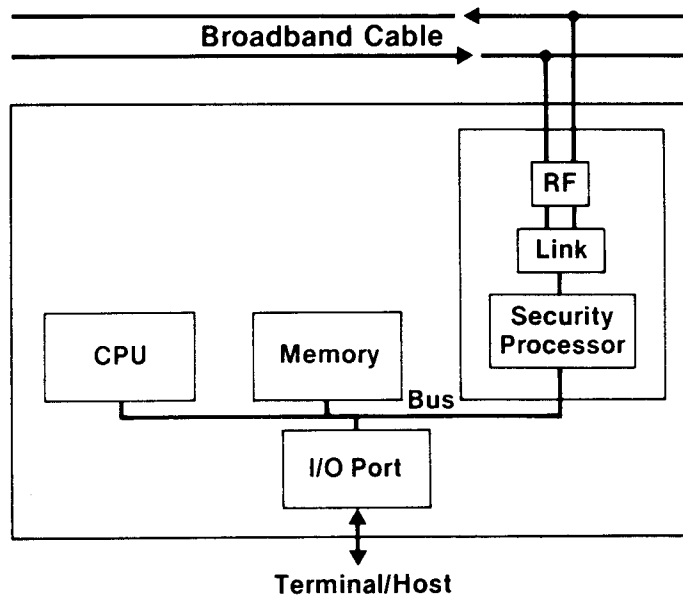




## SAFE - WIS Summary

- Similar High Level Design; Many Specifics Differ
- Need to Analyze Traffic Characteristics Impact
- E<sup>3</sup> Protocol Analysis and BIU Development Will Benefit WIS
- SAFE-Type Crypto Modules Can be “Easily” Incorporated in Reston Testbed
- NSA Will Develop Crypto Devices if WIS Requirements Are Clearly Specified in Time
- Need to Continue Tracking SAFE Effort

## Trusted Interface Unit



## **Progress**

- **Security Requirements Have Been Refined**
  - Scenarios Addressing Known Security Problem
  - Inputs to WIS Requirements Survey
- **Local Net Security Task Force**
  - Evaluate Issues of Encryption, Trusted Software, Security Protocols
  - Examine Technologies Within WIS Context
- **Security Architecture for WIS Has Been Developed**
  - Operating Mode for Transition Components Defined
  - Mandatory and Optional Requirements Identified
  - Technology to Support Security Requirements for Components Identified

## **WIS Security Summary**

- **More EFFICIENT SECURITY Controls Are VITAL to WIS**
- **NOT Seeking ABSOLUTE Multilevel Security**
- **LOCAL NET Architecture Affords More FLEXIBILITY in Solving Problem**

**TRUSTED COMPUTING RESEARCH  
AT  
DATA GENERAL CORPORATION**

Leslie DeLashmutt  
Doug Wells

Research Triangle Park  
North Carolina

**Goal**

Controlled sharing of information in a  
distributed, multi-user environment

## Overview

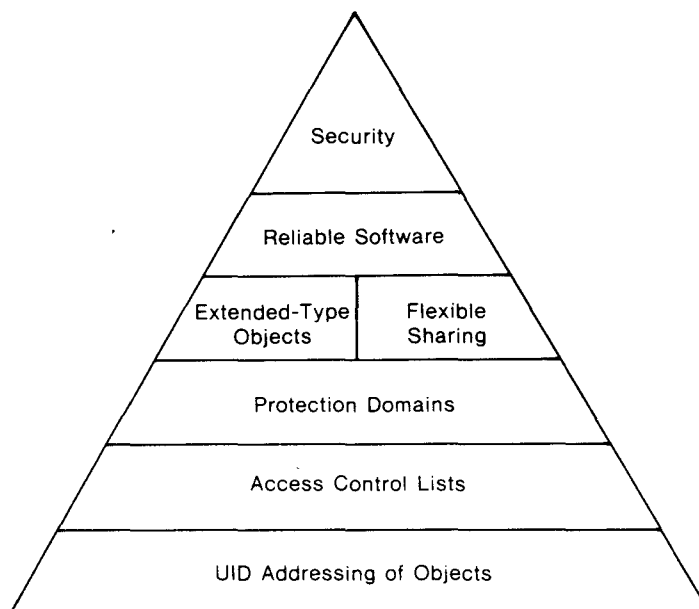
### Access control approaches

- Capabilities
- Access control lists (ACLs)

### Confinement approach

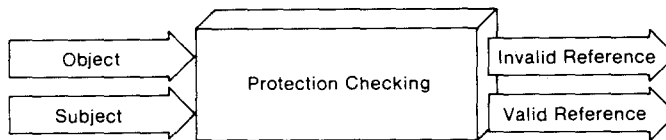
- Domains

### Extended types



**Protection Model**

- Active subjects
- Passive objects
- Access rights



**Access Matrix**

		Objects				
		38846	Proc1	38820	19926	...
Subjects	Jones	Execute	Read Execute	Read Write	Read Write	...
	Smith	Execute		Read Write	Read Write	...
	Lewis	Read Write Execute Non-Data		Read Write	Read Write	...
		⋮	⋮	⋮	⋮	⋮

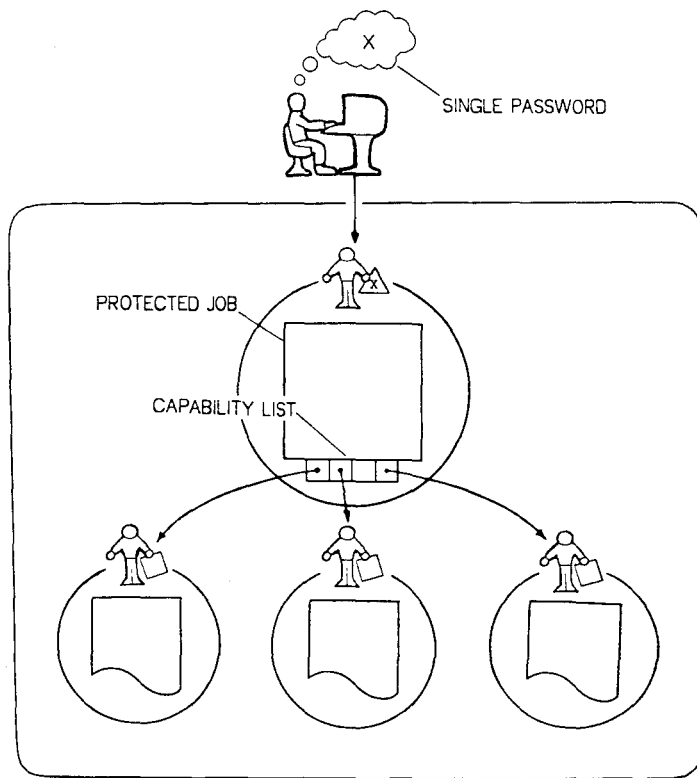
### Design Considerations

- Number of subjects and objects may be large
- No protection attributes for some subject/object pairs
- Matrix may be sparse
- Identical protection attributes for subjects or objects
- Only small part of matrix necessary at any one time

### Capability Systems

		Objects					
		38846	Proc1	38820	19926	...	Jones
Subjects	Jones	Execute	Read Execute	Read Write	Read Write	...	38846 E Proc1 R,E 38820 R,W 19926 R,W
	Smith	Execute		Read Write	Read Write	...	Smith 38846 E 38820 R,W 19926 R,W
	Lewis	Read Write Execute Non-Data		Read Write	Read Write	...	Lewis 38846 R,W,E,Non-Data 38820 R,W 19926 R,W
	⋮	⋮	⋮	⋮	⋮		

# PROTECTION USING CAPABILITIES



1975 IEEE. Reprinted by permission Clark, D.D., and Redell, D.D.,  
*Protection of Information in Computer Systems*, p.15

### Evaluation of Capabilities

#### Virtues

- Protection
- Simplicity
- Flexibility
- Efficiency

#### Problems

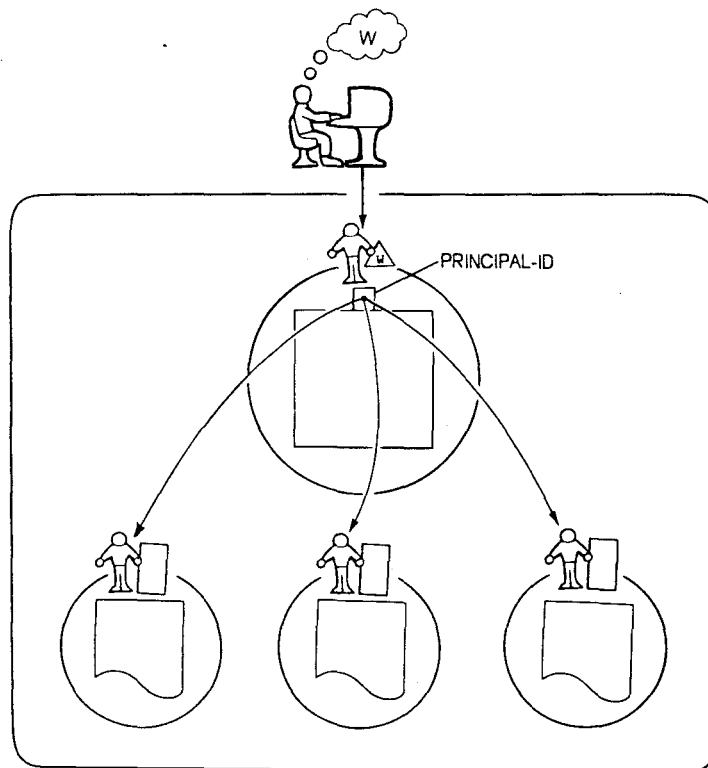
- Forgery of keys
- Accountability
- Revoking access
- Controlling propagation
- Access review

### Access Control List Systems

		Objects					
		38846	Proc1	38820	19926	...	
Subjects	Jones	Execute	Read Execute	Read Write	Read Write	...	38846 Jones E Smith E Lewis R,W,E,Non-Data
	Smith	Execute		Read Write	Read Write	...	Proc1 Jones R,E
	Lewis	Read Write Execute Non-Data		Read Write	Read Write	...	38820 Jones R,W Smith R,W Lewis R,W
	⋮	⋮	⋮	⋮	⋮	...	19926 Jones R,W Smith R,W Lewis R,W



## PROTECTION USING ACCESS CONTROL LISTS



1975 IEEE. Reprinted by permission. Clark, D.D., and Redell, D.D.,  
*Protection of Information in Computer Systems*, p.18.

### **Positive Features of ACL Systems**

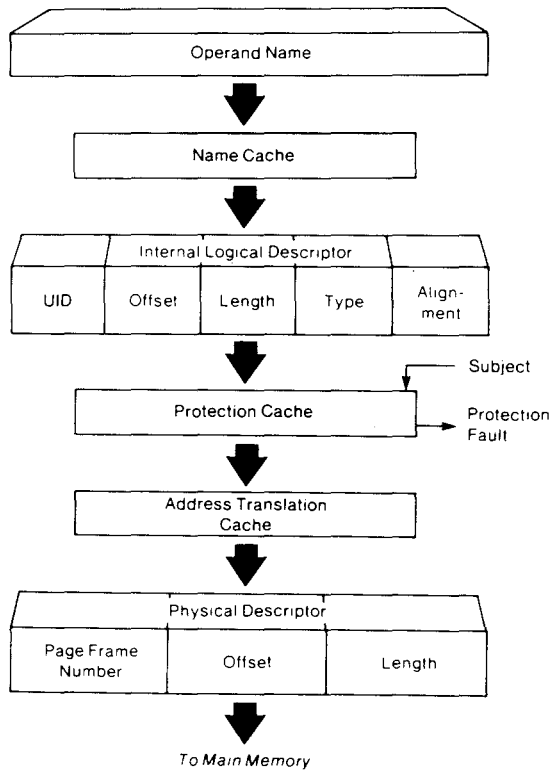
- Granting access has known, auditable consequences
- ACLs directly implement verification of an access request
- Access revocation is manageable
- Each ACL lists authorized users of an object
- Break association between data organization and authorization
- Natural to the user
- Minimal hardware implementation costs
- Readily adapted to heterogeneous networks
- Natural primitive for a high-level security language
- Provide top-down view of security

### **Drawbacks of ACL Systems**

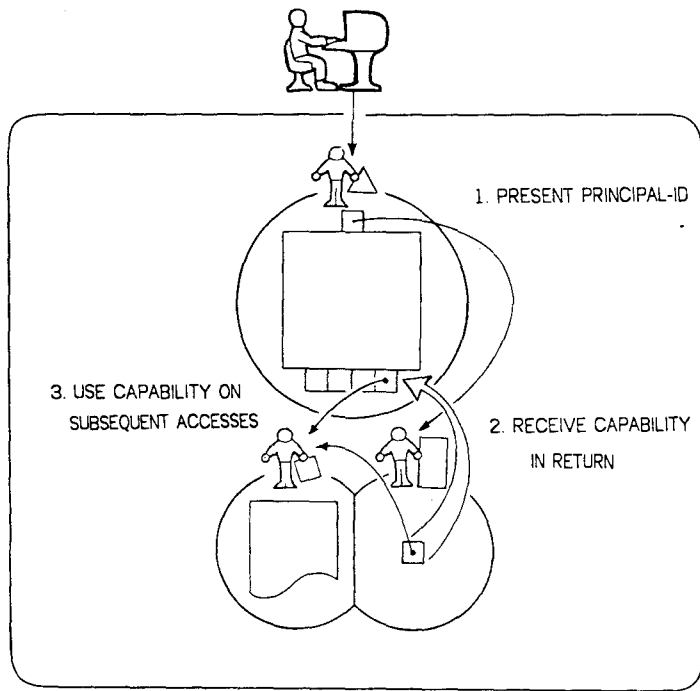
- ACL search
- Allocation of space for ACL
- ACL check at access time

The purpose of an ACL is to establish authorization—not to mediate every detailed access.

### Memory Addressing in an ACL/Descriptor System



# CAPABILITY ACQUISITION IN A HYBRID SYSTEM

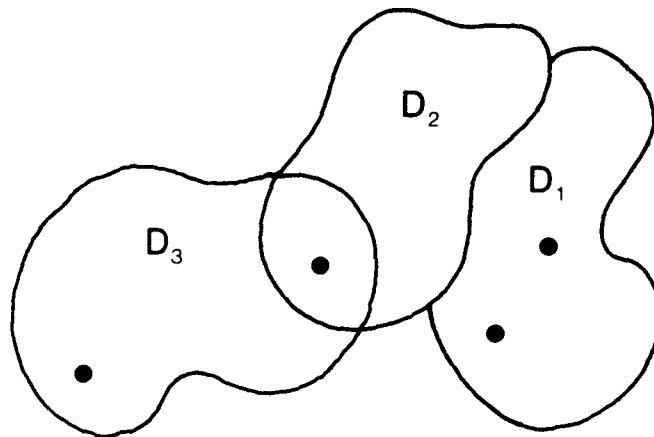


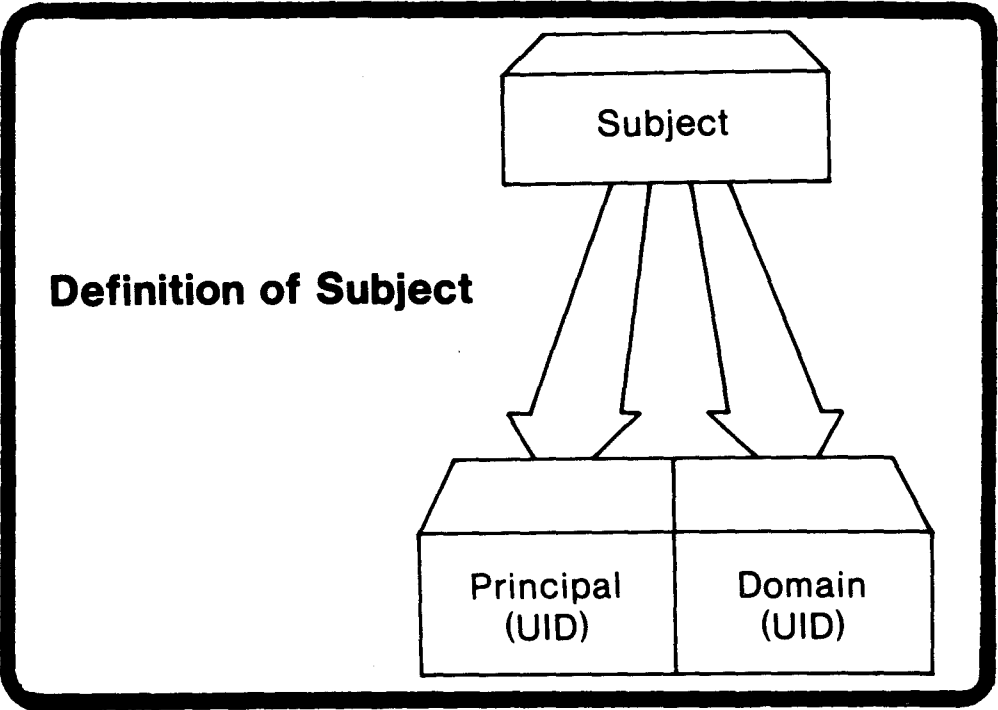
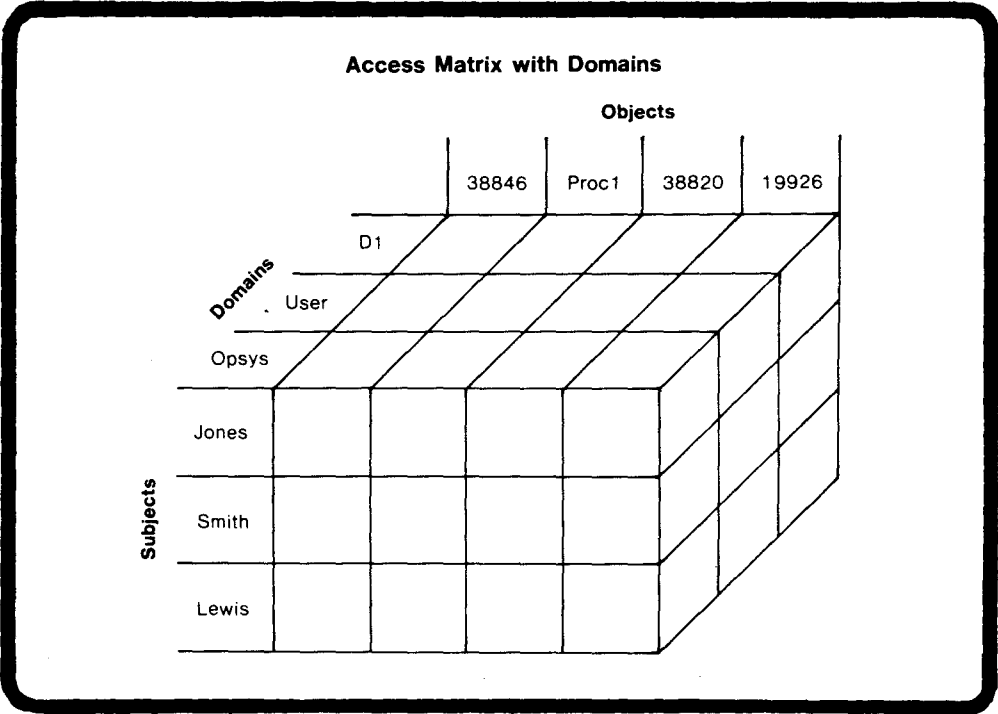
1975 IEEE. Reprinted by permission. Clark, D.D., and Redell, D.D.,  
*Protection of Information in Computer Systems*, p.21.

### Limitations

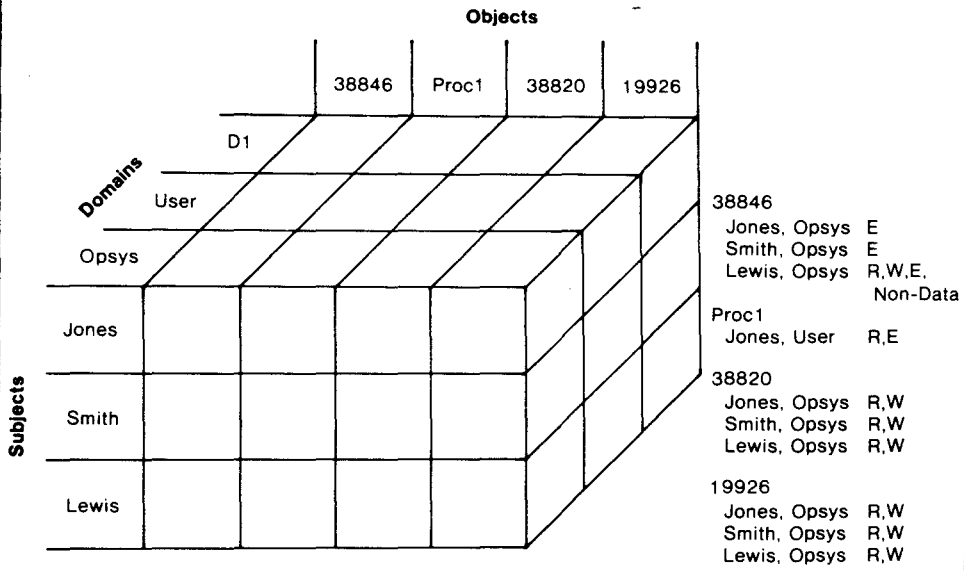
- Only system-defined access restrictions enforced
- No protection of user from borrowed program "trojan horse"
- No protection of borrowed program from user

### Beginning Domain Model

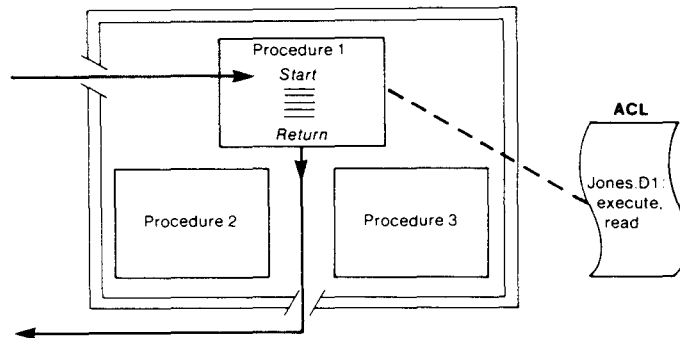




### Access Control List with Domains

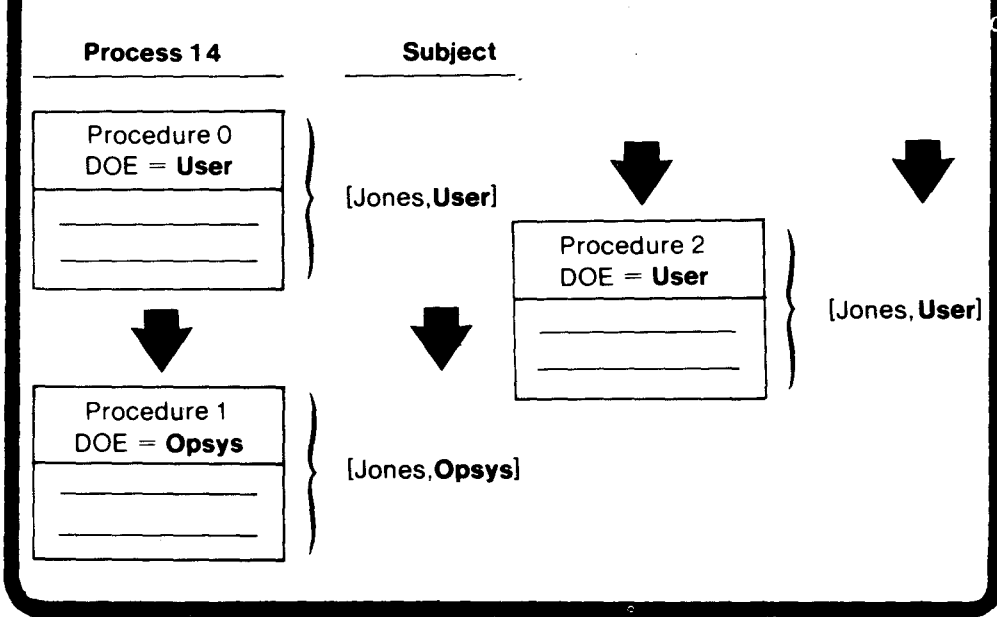


### Gates into Domains

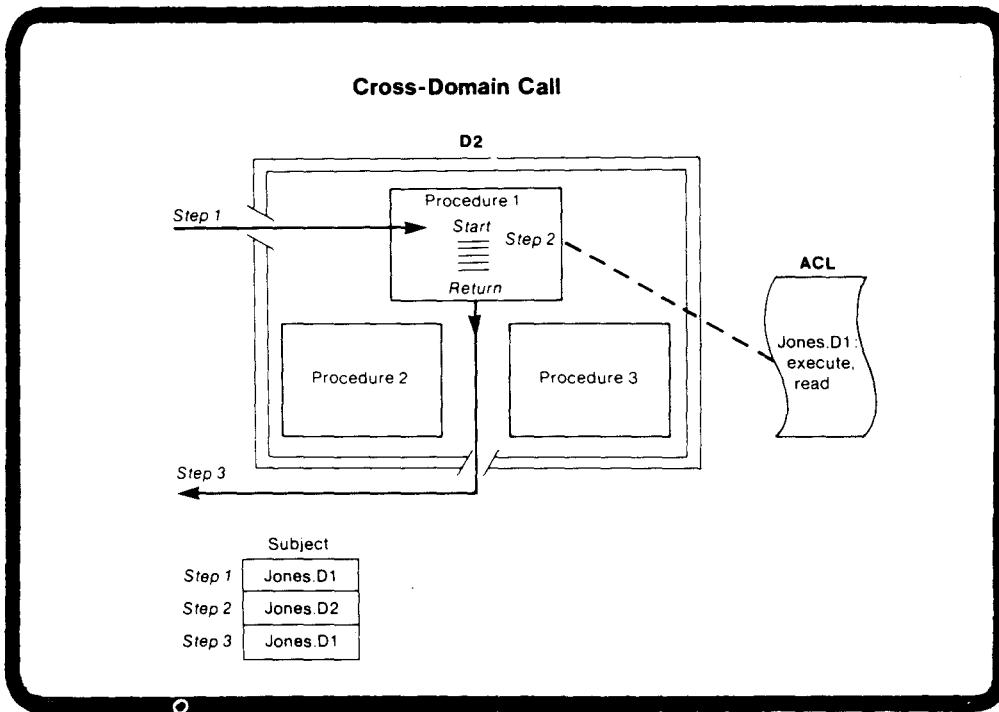




## Simplified Cross-Domain Call Example



## Cross-Domain Call



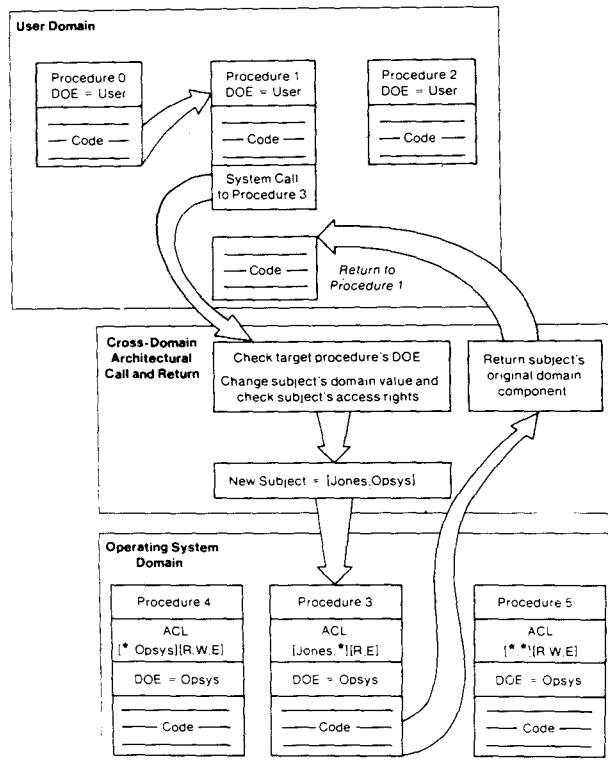
### **Potential Implementation of Domains**

- Interprocedure call and return
- Problem: no architectural assurance that a procedure can access its arguments when called in a new domain
- One solution: dynamic access capabilities on cross-domain calls

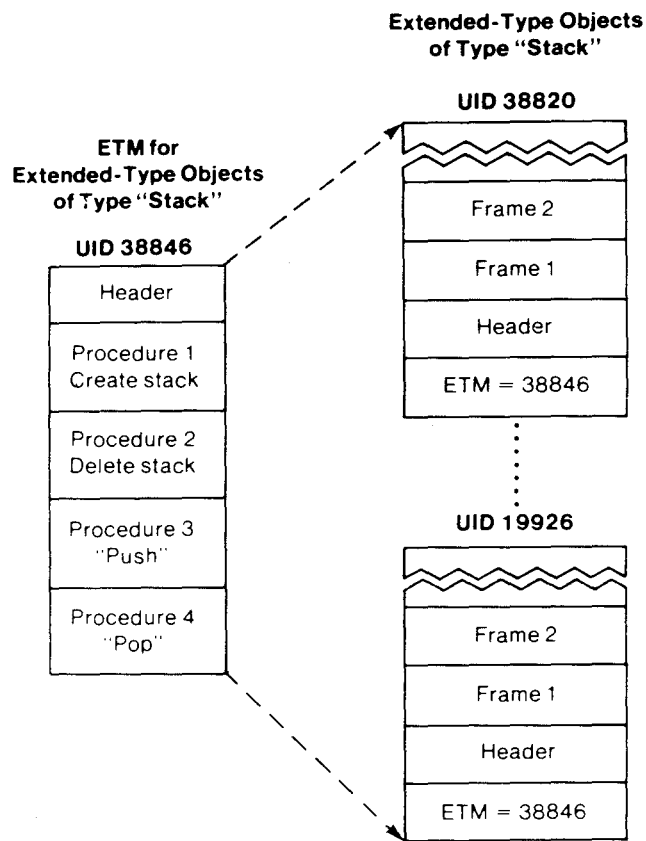
### Cross-Domain Call Example

Initial Subject  
Bound to Process 14

Principal	Domain
Jones	User

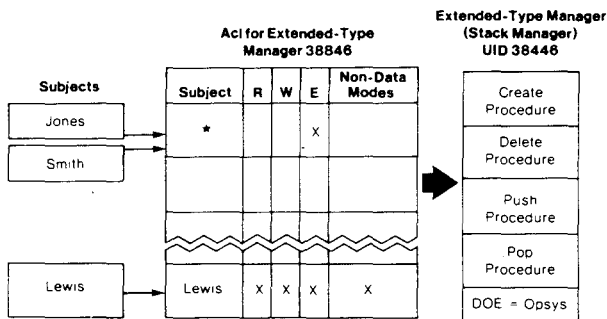


### Extended-Type Object Example



### Extended-Type Manager Example

STEP 1



STEP 2

**Extended Access Control Lists for Extended-Type Objects of Type "Stack"**

Subject	Pop	Push	Create	Delete
Jones	X	X		
Lewis	X	X	X	X

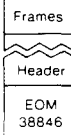
EACL for Object 38820  
(Jones' stack)

Subject	Pop	Push	Create	Delete
Smith	X	X		
Lewis	X	X	X	X

EACL for Object 19926  
(Smith's stack)

STEP 3

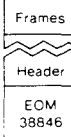
**Jones' Stack UID 38820**



Subject	R	W	E
* Opsys	X	X	

ACL for Object 38820

**Smith's Stack UID 19926**



Subject	R	W	E
* Opsys	X	X	

ACL for Object 19926

### **Future Directions**

- Military security model
  - Flow control
  - \*-property (prevention of write-down)
- Formal specification of design
- Formal model for security in our architecture
- Fault tolerance
- Encryption
- Sophisticated authentication mechanisms

### **Summary of DG/RTP Activities to Date:**

- Made critical survey of primitives available to support a trusted computing base
- Selected the best concepts to support such a base
- Integrated these concepts into a coherent architecture

IEEE Computer Society Fall Comcon '81

"Research in High-Level Computer  
Architecture"

John F. Pilat  
Data General Corporation/Research Triangle Park

# **THE iAPX-432 MICROCOMPUTER SYSTEM**

GEORGE COX  
INTEL CORPORATION

- **VLSI COMPONENTRY**
- **ARCHITECTURE**
- **OPERATING SYSTEM**
- **SYSTEM LANGUAGE**

## **432 MOTIVATIONS**

- **HIGH PERFORMANCE MOS/VLSI (HMOS)**
- **RECENT COMPUTER SCIENCE RESEARCH**
- **ADVANCING MICROCOMPUTER APPLICATIONS**



## **432 DESIGN OBJECTIVES**

- **LARGE SCALE COMPUTING POWER**
- **INCREMENTAL PERFORMANCE CAPACITY**
- **INCREASED PROGRAMMER PRODUCTIVITY**
- **DEPENDABLE HARDWARE AND SOFTWARE**

## **KEY CONCEPT: DATA ABSTRACTION**

- **MODULAR DATA STRUCTURES AND PROCEDURES**
- **WELL-DEFINED MODULE INTERFACES**
- **OBJECT-ORIENTED PROGRAMMING METHODOLOGY**

**KEY CONCEPT: HIGH LEVEL FUNCTION**

- LANGUAGE-ORIENTED RUN-TIME ENVIRONMENTS
- HARDWARE-CONTROLLED RESOURCE MANAGEMENT
- OBJECT-BASED INTERFACES AND SERVICES

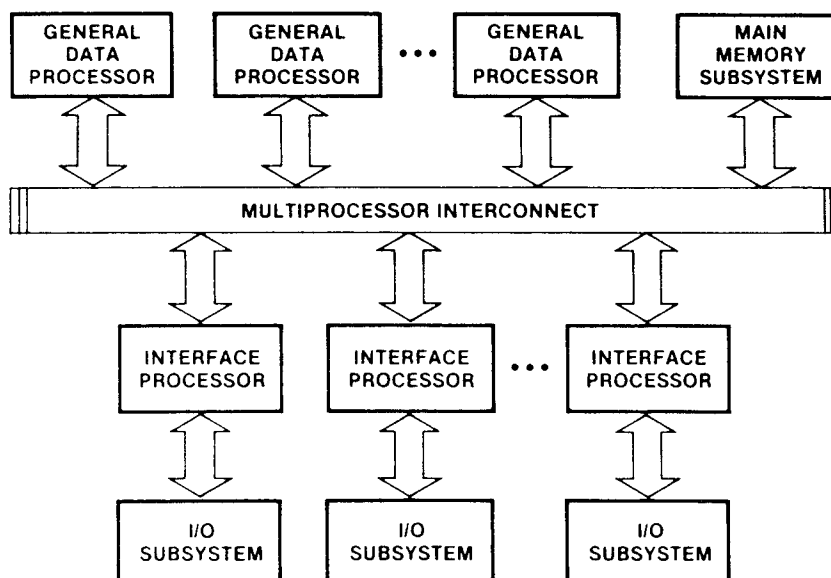
**KEY CONCEPT:  
DOMAIN-BASED PROTECTION**

- INDEPENDENT MODULE ADDRESS SPACES
- ADDRESS SPACE SWITCHING ON PROCEDURE CALL
- CAPABILITY ADDRESSING AND ACCESS CONTROL

## KEY CONCEPT: OBJECTS AS UNIFIED DESIGN FRAMEWORK

- INTEGRATING HARDWARE AND SOFTWARE
- MINIMIZING CONCEPTUAL DIFFERENCES
- CLARIFYING AND SIMPLIFYING OVERALL DESIGN

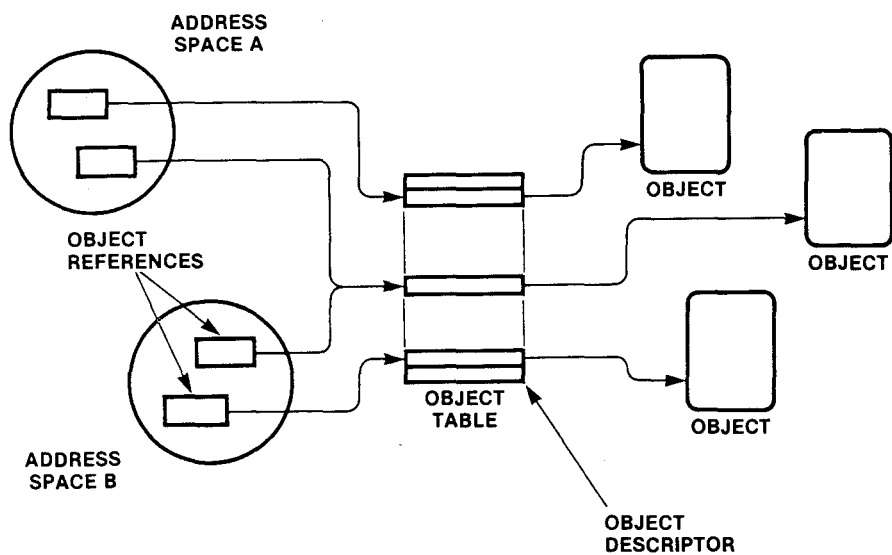
## KEY CONCEPT: MULTIPROCESSING



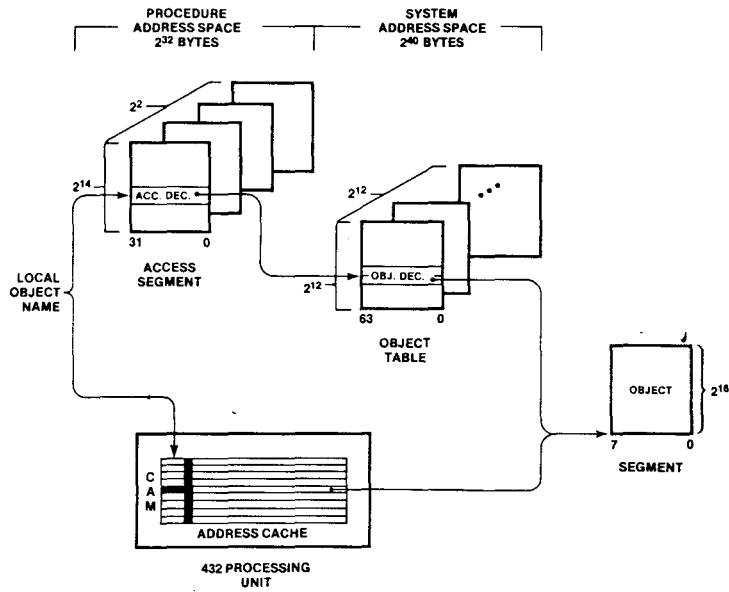
## ELEMENTS OF THE ARCHITECTURE

- OBJECT-BASED ADDRESSING AND PROTECTION
- BASIC COMPUTATIONAL FACILITIES
- PROGRAM EXECUTION ENVIRONMENTS
- OBJECT-ORIENTED PROGRAMMING SUPPORT
- INTERPROCESS COMMUNICATION
- SYSTEM RESOURCE MANAGEMENT

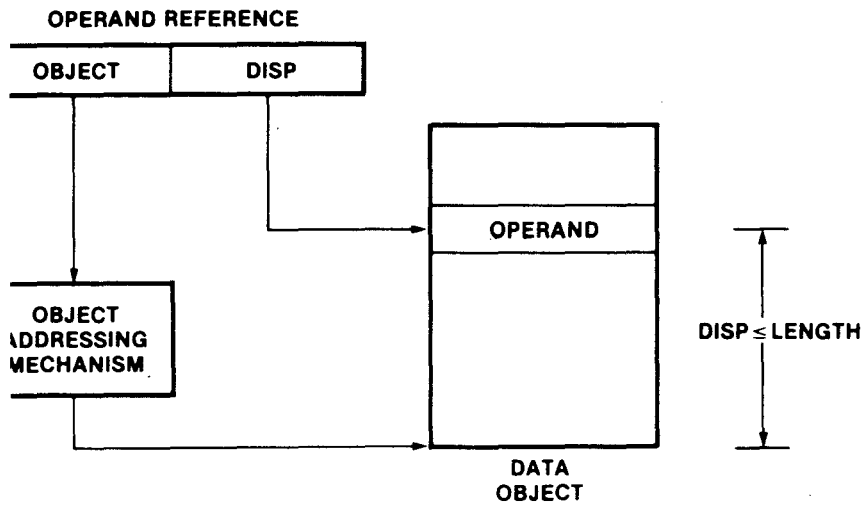
## A CONCEPTUAL VIEW OF OBJECT ADDRESSING



## 432 OBJECT ADDRESSING MECHANISM



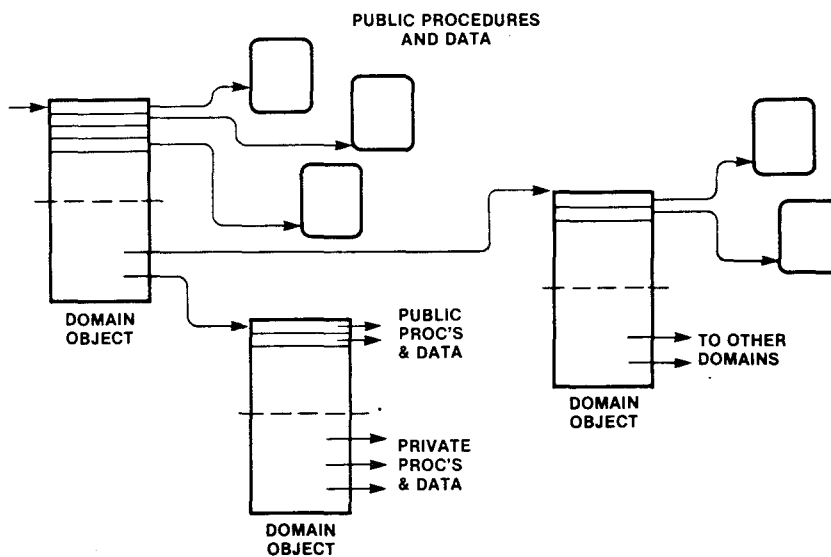
## SIMPLE OPERAND ADDRESSING



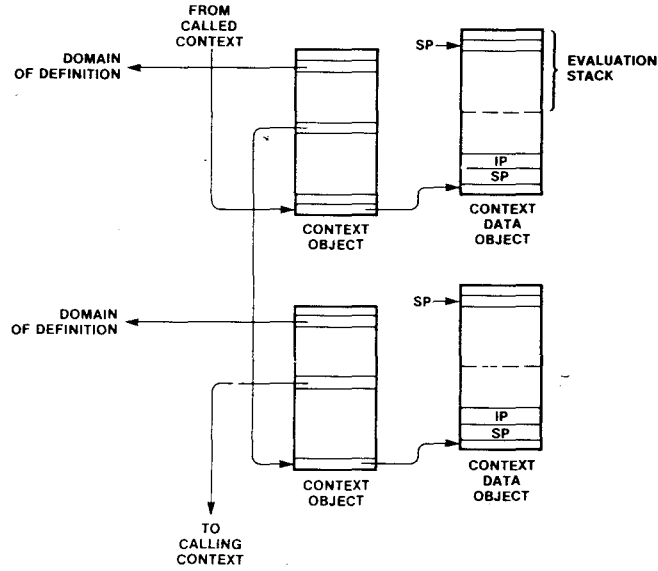
## COMPILER-ORIENTED INSTRUCTION FORMATS

- ZERO TO THREE OPERANDS PER INSTRUCTION
- SYMMETRIC OPERAND ADDRESSING MODES FOR:
  - SCALARS (BASE + DISPLACEMENT)
  - VECTOR (BASE + INDEX)
  - RECORD ELEMENTS (BASE + INDEX + DISPLACEMENT)
- REGISTER-FREE
  - OPERANDS IN MEMORY
  - OPERANDS ON TOP OF STACK
  - ANY MIXTURE OF MEMORY AND STACK OPERANDS
- BIT VARIABLE

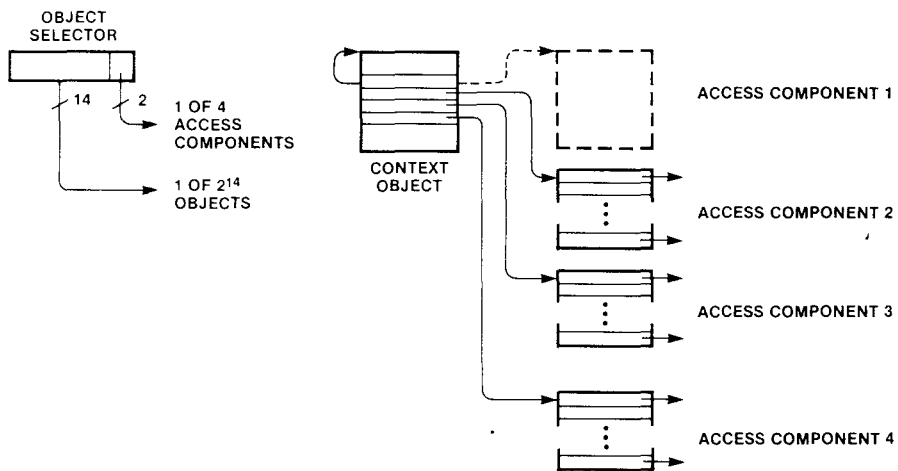
## PACKAGING PROGRAM MODULES



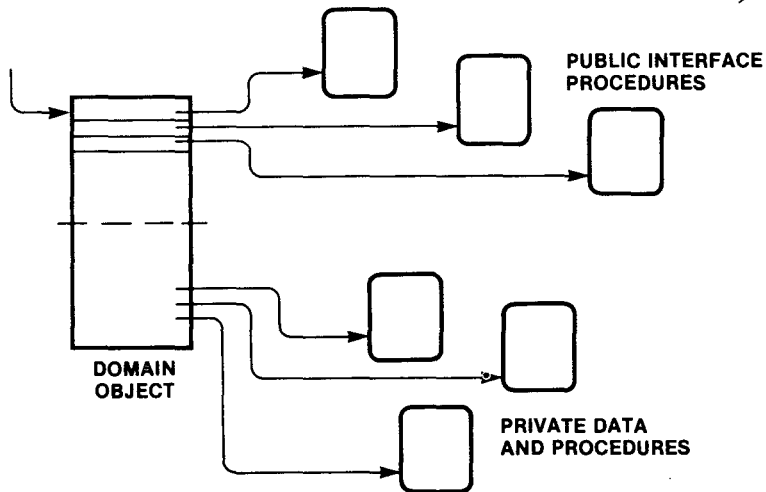
## CONTEXTS AS PROCEDURE ACTIVATIONS



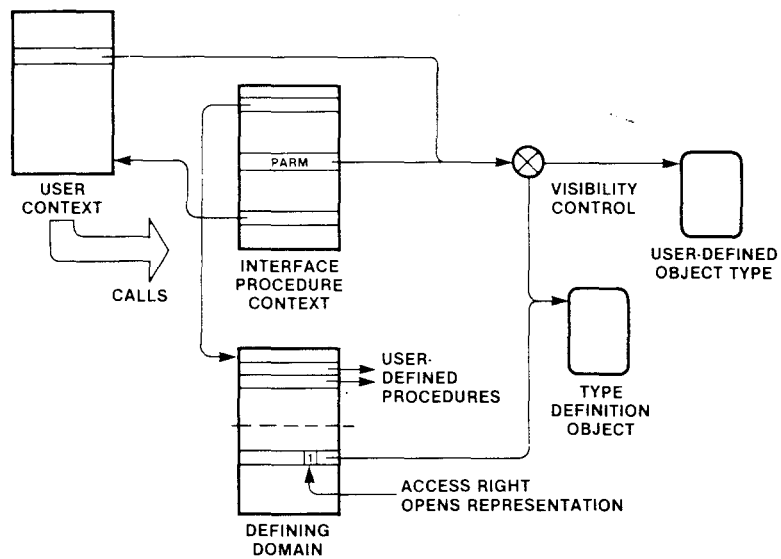
## CONTEXTS DEFINE THE INSTANTANEOUS LOGICAL ADDRESS SPACE



## DOMAINS AS USER-DEFINED OBJECT TYPES



## PROCEDURE-FREE REPRESENTATIONS OF USER-DEFINED OBJECTS

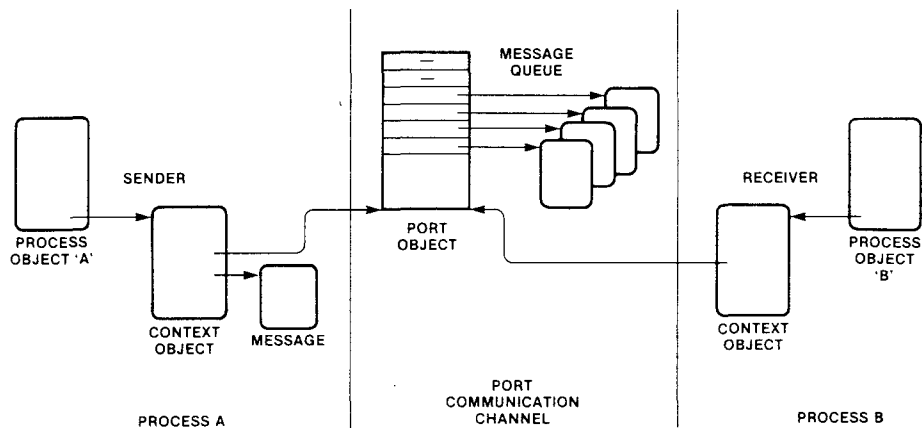




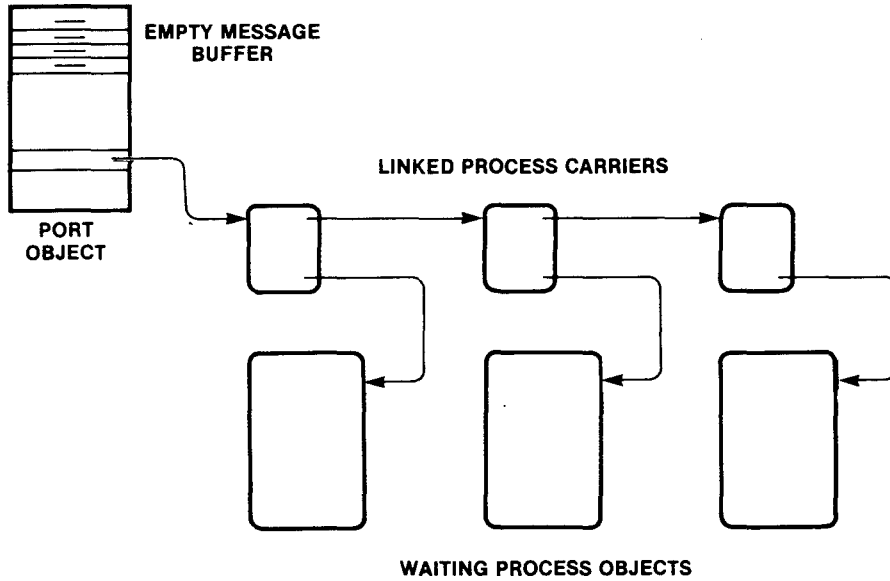
## 432 SYSTEM OBJECTS ARE BASIS OF THE SILICON OS

- KEY HARDWARE-DEFINED OBJECTS CONTROL SYSTEM FUNCTIONS, e.g.:
  - PROCESS OBJECT
  - STORAGE RESOURCE OBJECT
  - PORT OBJECT
- HARDWARE PROVIDES THE KEY OPERATIONS
  - TIME CRITICAL
  - SECURITY SENSITIVE
  - COMPLEX
- SOFTWARE AND HARDWARE COOPERATE TO MANAGE THESE OBJECTS

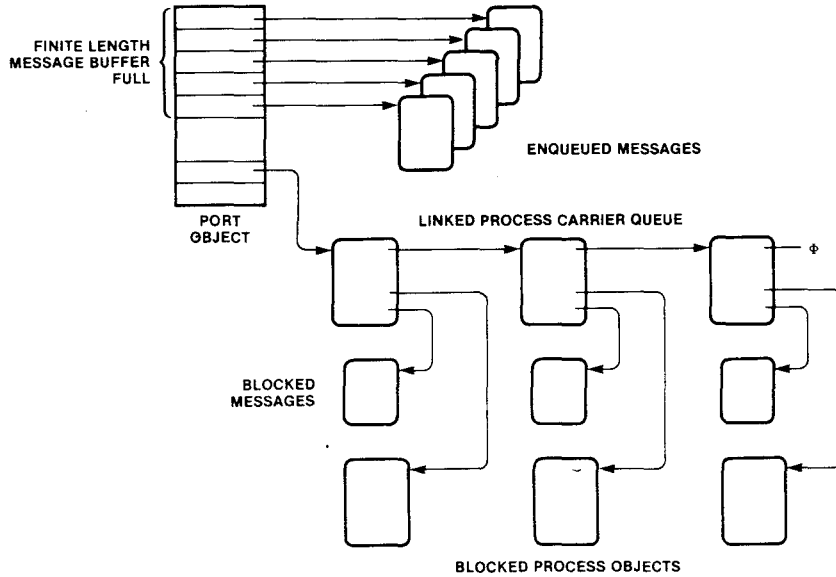
## OBJECT-BASED INTERPROCESS COMMUNICATION



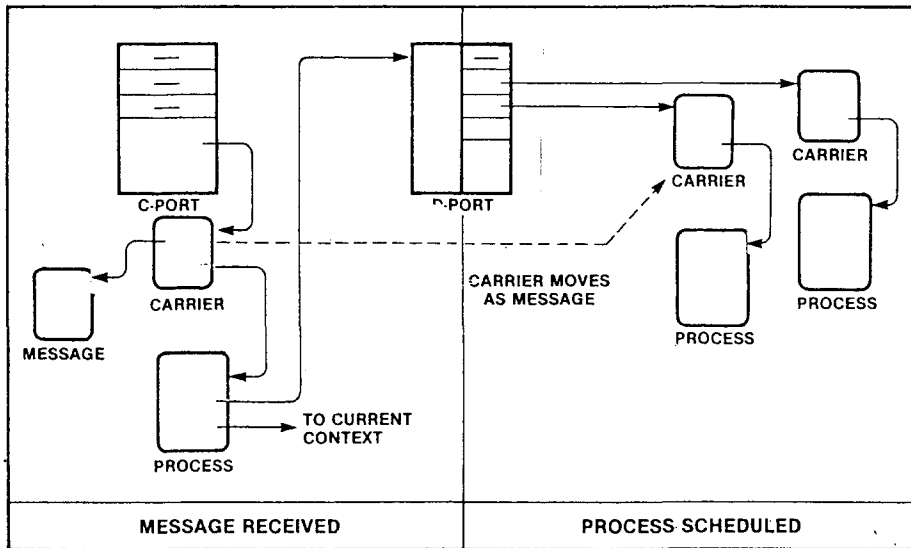
## CARRIERS ENQUEUE WAITING PROCESSES



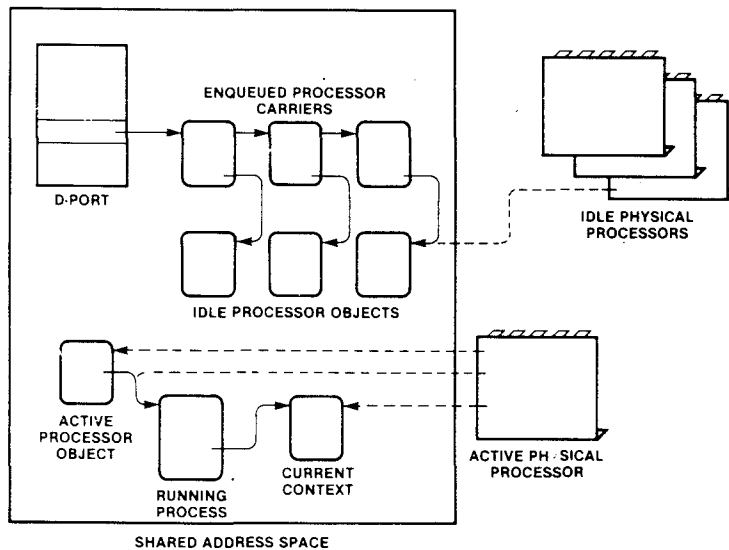
## CARRIER OBJECTS RESOLVE MESSAGE QUEUE OVERFLOW



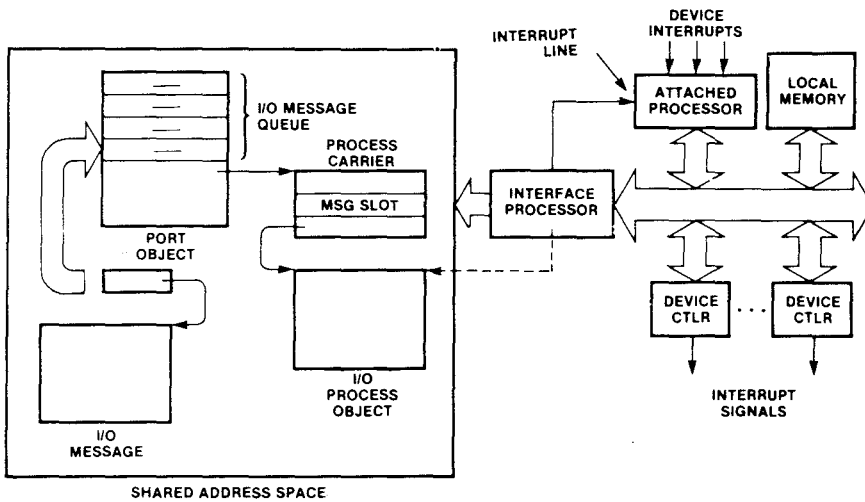
### A MESSAGE-PASSING MODEL OF PROCESS SCHEDULING



### MESSAGE-PASSING MODEL OF PROCESSOR DISPATCHING



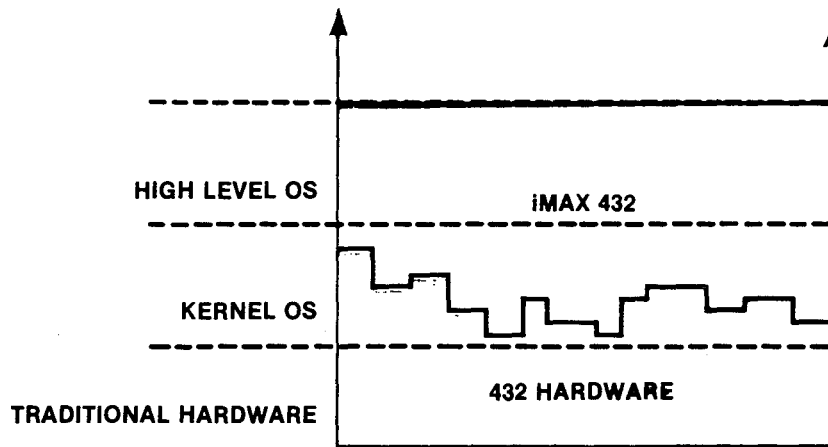
## I/O IS A SPECIAL CASE OF INTERPROCESS COMMUNICATION



## iMAX 432: MULTIFUNCTION APPLICATIONS EXECUTIVE

- **COOPERATES WITH HARDWARE TO MANAGE OBJECTS**
  - OBJECT CREATION
  - OBJECT MAINTENANCE
  - OBJECT DISPOSAL
- **PROVIDES A UNIFORM VIEW OF AN OBJECT**
  - e.g., FOR PORTS: CREATE PORT (SOFTWARE)
  - SEND MESSAGE (HARDWARE)
  - RECEIVE MESSAGE (HARDWARE)

## iMAX 432 AND THE SILICON OS



REBALANCING THE HARDWARE/SOFTWARE INTERFACE

## ADA: THE IDEAL IMPLEMENTATION LANGUAGE FOR 432

- **ADA MATCHES THE 432 DESIGN METHODOLOGY**
  - BASED ON THE CONCEPT OF OBJECTS
  - SIGNIFICANT SUPPORT FOR MODULARIZATION
  - AIMED AT REDUCING PROGRAMMING COSTS
- **ADA CONSTRUCTS MAP THE ARCHITECTURE AND OS**

OBJECT	OBJECT
PACKAGE	DOMAIN
ACCESS	ACCESS DESCRIPTOR
SUBPROGRAM ACTIVATION	CONTEXT
- **ADA-432 FEATURES PROVIDE DIRECT ACCESS TO THE HARDWARE**
  - 432 SPECIFIC OPERATIONS ARE IN ADA'S STANDARD MACHINE ACCESS PACKAGE
  - SIMPLE 432 EXTENSIONS TO ADA SUPPORT DYNAMIC SYSTEMS

# ICL EFFORTS IN COMPUTER SECURITY

Tom Parker

International Computers Limited

This presentation covers a subject which is becoming a bit of a Cinderella in the secure computing world. I am talking about the much criticized business of making a real life, big machine, practical, commercially acceptable operating system as secure as possible.

I am talking about the kind of system for which the use of formal verification technology is beyond the state-of-the-art; for which restructuring along the TCB lines proposed by Grace Nibaldi must be very expensive, and for which there are no absolute guarantees of security, a system for which the attainment of the magic Nibaldi level 6 is a fairytale fantasy, but a system which nevertheless occupies an important niche in the total spectrum of secure data processing requirements.

## SLIDE 1

The system I shall be talking about is a large general purpose operating system from a European manufacturer. It is called VME/B, and it is marketed by ICL on our 2900 range of computer mainframes.

As some of the audience here today may not know much about ICL, I think I'd better start by giving a brief description of who we are, and give a bit of background to the development of the 2900 series. I shall then go on to describe the hardware architecture of 2900, concentrating of course on those features that are most relevant to security. VME/B is the largest of a number of ICL operating systems that run on this architecture; it is a system that has received a lot of attention from the point of view of security in ICL and I shall be describing some of its protection features. It has also been the target for much of the security enhancement work that has been undertaken by ICL, mainly with the objective of satisfying the needs of customers with especially stringent security requirements. I shall outline some of this work at the end of the presentation.

## SLIDE 2

"ICL" stands for International Computers Limited, and not only is ICL the only indigenous UK mainframe manufacturer, but also we are one of the few to produce computers with an architecture fundamentally different from IBM's. To give you some idea of the size of the Company, our

turnover last year exceeded 1.5 billion dollars and we employ over 26,000 people, about 20,000 of whom are in the UK.

The value of ICL's world-wide customer base is well over 4.5 billion dollars in 86 different countries. Apart from the ubiquitous IBM this is the biggest customer base outside America and Japan of any computer manufacturer.

ICL was formed in 1968 as a result of a merger between what was then two major, and competing British computer companies: ICT and English Electric. At that time it was realized that the new Company would soon need a range of new machines to replace the many, varied and incompatible ones inherited from the merger. Also of course these inherited machines had architectures and a hardware technology dating from the 50's and early 60's. Both hardware and software technology had moved on a lot since then.

So out of all this after an appropriate gestation period, came the first of the 2900 series. This was a revolutionary rather than an evolutionary step. A rare thing in the commercial computer world.

### SLIDE 3

The design was of course influenced by that of existing in-house systems and obviously the architectures of other machines in the marketplace at the time were also examined, for example a number of MULTICS concepts were very influential, particularly in the protection sphere, where some aspects still represent state-of-the-art, even 12 years later.

For those of you who would like to know more about this history, I would recommend John Buckle's book on the subject, which also appears in condensed form in the November 78 issue of the ICL Technical Journal.

### SLIDE 4

#### 2900 ARCHITECTURE

So what kind of beast did we produce? Let's have a look at some of the architectural features of the 2900. This is a list of the ones we shall look at. I'll describe each one then bring this slide back and collect them altogether at the end.

Central to the architecture of 2900 are the complementary concepts of virtual store and virtual machines, and their common base of virtual addressing.

### SLIDE 5

All addressing is in terms of virtual addresses mapped onto real addresses by hardware using segment and page tables. The real addresses can be in real store or on secondary store on drum or disc. In order words, we have a straightforward virtual store implementation. Each process runs in its own "virtual machine" in which it has its own unique local segment table and shares a public segment table with all other virtual machines. It can optionally also have "global" segments which it shares with chosen other virtual machines. I think that it is generally accepted that this kind of hardware-supported process address-space separation is important to good system security.

#### SLIDE 6

The primitive instruction code makes extensive use of 'descriptors' for indirect addressing. A descriptor is a 64-bit entity which formally describes an item of information in store. One half of the descriptor contains the base address of the item being described in terms of segment number and displacement. In other words, this half of the descriptor contains a 'virtual' address. The other half contains information relating to the unit size of the item, the number of units it contains, whether modifiers added to the item's address should be appropriately scaled or not, and so on. Descriptors are also typed according to what kind of information they are addressing. This slide shows a "Descriptor Descriptor" pointing to a row of three "Byte-Vector" descriptors, each of which is pointing at a bounded area of virtual store. Some other examples of descriptor types are Code Descriptors, Semaphore Descriptors and System Call Descriptors. One obviously important 'correctness' feature present in the 2900 is automatic bound checking on modification.

#### SLIDE 7

Next in the list, are the features needed to control basic input/output and other primitive privileged operations. The totality of addressable hardware registers is called the "image store". This divides into two parts: the visible and invisible registers, and the distinction between them is critical to the system's security.

Visible registers are accessible by normal unprivileged instructions, and consist of such things as Program Counter, local namebase pointer, Real Time Clock, and so on. Access to the invisible registers is by using what is called "image store operand format," and this requires privilege.

Access to the invisible registers is required to perform Input/Output operations, to activate a new process and to perform other privileged functions.

Privileged status is obtained only by a hardware interrupt mechanism. When the VME/B Operating System is present, such interrupts cause entry to the most trusted part of the operating system (the



'Kernel' of VME/B).

SLIDE 8

A process's level of trustedness is defined by the contents of an 'invisible' register: the Access Control Register, called ACR for short. The ACR register is actually a part of the Program Status Register shown on the previous slide. The level of trustedness is called the ACR level. The lower the value of its ACR, the more trusted a process is.

Segment protection occurs in that on access to store, the ACR level of the process and the mode of access are compared with access permission fields that are associated with each segment. "Change Access" on the slide refers to the ability to change the access permission fields themselves. There is also an Execute Permission bit which is used to prevent the accidental execution of data.

Entry to a procedure running at a different ACR level must be via a hardware-supported 'system call' mechanism which polices the availability of the called procedure from the caller's ACR level. An important feature of the mechanism is its enforcement of entry to the procedure at the proper entry point. So you can see that what we have here is a ring protection system. There are sixteen possible different levels, that is: 16 ACR levels.

Critical to the security of the system is the proper validation by a trusted procedure of reference parameters passed to it when being called by less trusted code. A special primitive instruction is provided for this purpose which we call the 'Validate' instruction. I shall be saying more about parameter validation later.

So, let's pause for a minute and look at the major items so far.

SLIDE 9

We have:

virtual addressing, supporting  
 virtual store and  
 virtual machines - providing protection between processes  
 descriptors with automatic bound checking,  
 a protected I/O mechanism;  
 and a 16 level ring protection system with an  
 associated mechanism for policing the transfer of control  
 between the rings - providing protection with a process.

These are all basic architectural hardware supported features present in the raw machine.

I should quickly mention one further architectural feature: the "process stack." It has no direct security connotations, but makes such an important contribution to the overall flavour of the 2900 architecture that it would be misleading to miss it out.

The instruction code at the primitive level is based on the use of a LIFO or "Last In First Out" stack. The stack is used for parameter passing and local name space purposes and each Virtual Machine has its own stack. Nested procedure calls will cause the usual succession of name spaces to be built up on the stack, which are deleted on a "last in, first out" basis as the procedures exit. We have found the stack mechanism on 2900 to be an elegant and natural aid to the procedure call mechanism. Those then are the main architectural features of the 2900 series machines. When it was introduced it was quite an advanced system for its time. In fact many of ICL's early development problems stemmed from the fact that we were breaking new ground in so many areas. Even today, operating system technology in commercially available systems is only just catching up with the 2900 architecture which forms a very adequate basis for the development of a secure operating system.

#### SLIDE 10

One such development is VME/B. VME stands for Virtual Machine Environment. B stands for B.

VME/B is a large mixed workload operating system catering for Batch, Multi-Access and Transaction Processing applications.

The smallest machine on the 2900 range on which a full system is at present intended to be run is the 2956, though there are subset options that can run on a smaller machine. Comparisons are difficult but the 2956 is very roughly equivalent to an IBM 4331 Model 2, or somewhere between a DEC VAX 11/750 and VAX 11/780. A practical full VME/B system including typical user application code needs a real store size of at least 2 megabytes, so it's a big operating system.

The operating system divides into three quite distinct parts, separated by error handlers as shown on the slide. At the most trusted level is Kernel, which handles real system resources like store and devices. It runs mainly 'out of process' on a public stack and helps to support the virtual store/virtual machine image of the basic 2900 architecture.

Director is responsible for the handling of a more abstract view of the system's resources. At this level are the block level file managers, and the major security related operating system functions like the loader, name handler and privacy controller. Director occupies ACR levels 4 and 5.

Uncontrolled communication between virtual machines is prohibited above ACR level 5 by disallowing public segments with write access keys

greater than ACR 5 and by controlling the availability of global segments.

Level 7 to 9 contain the Above Director software as shown on the slide. From the security point of view it could be considered as a sort of "trusted superstructure." Above ACR 9 is the real enemy - the user. Facilities are provided for user installations to structure the levels at which the various applications run within ACR levels 10 to 15, and these can be used by installation management to cut normal unprivileged users off either partially or completely from direct use of the facilities of the operating system if so required. I shall say a bit more about this later.

Notice that unlike in some contemporary machines, compilers, and general utilities are no more trusted by the operating system than user code.

All operating system code segments are established with a write access key of zero and so all operating system code is necessarily pure.

#### SLIDE 11

The slide shows a typical selection of operating system procedures and VME/B's use of the system call mechanism. The small boxes are the procedures. I have drawn them at their execution ACR level. The lines with blobs on the end show the highest ACR level from which they can be called.

In the actual system the vast majority cannot be called at all from outside their own level, but there still remains a substantial number that are directly callable from user ACR levels.

The proper validation of parameters passed across the user/operating system interface is well-known to be critical to the security correctness of operating systems, and VME/B is no exception. A lot of time and energy has been spent ensuring that reference parameter validation is complete. In particular an object code analysis package has been written that searches out discrepancies for manual analysis and correction. It examines actual loaded code, and so detects flaws that might be introduced by post compilation patches or repairs which at that stage have been fully applied. The checking is therefore as near to the 'engine' as possible.

Another approach has been to reduce the number of procedures available to the user at particular secure installations. A package has been produced to monitor usage of system code with a view to making unused interfaces unavailable, or restructuring the availability of little used interfaces on particular secure sites.

The package has a very low performance overhead and can be permanently left in the system.

Installations have a considerable degree of control over operating system called is defined at system load time in a load control file interestingly called the 'recipe' file, and this file is amendable by installation management. A mechanism also exists whereby trusted users can access a smaller number of additional procedures. The availability of these more powerful interfaces can be tailored to the specific functional requirements of the chosen trusted user classes. Standard ones are for example Support Engineers, Operators, or the System Manager.

There are in fact a number of areas in VME/B into which hooks and options have been put. This gives installation security authorities a great deal of flexibility in deciding on what they want for their system.

Indeed, the extent to which particular secure installations can bend VME/B to suit their individual security requirements is itself a major security feature of VME/B.

To give a crude example, a class of multi-access user can be defined whose only commands are, say,

```
INPUT
EDIT
COBOL COMPILE
COBOL RUN,
```

with no low level code or direct use of operating system interface being allowed at all.

#### SLIDE 12

All major system objects are recorded in a central filestore database known as the VME/B Catalogue. It is controlled from ACR level 5. The Catalogue is organized in terms of nodes and relationships. Entries for named objects are located at nodes which are connected by relationships. Objects catalogued include Devices, Volumes, Files and other specialized VME/B objects. One example of a VME/B object is shown on the slide - that of a job profile. A user's access to job profile nodes determines the kind of work that he can run on the system.

Privacy controls can be applied to all of these objects and access can be constrained on a general, specific and hierarchic basis. A wide variety of success types is supported, distinguishing for example between access to a file's contents and access to its name and description. All attempted privacy violations are logged to a security journal. An installation can arrange that such messages are output immediately to the journal, or held in a buffer and then output when the buffer is full.

Particularly important in the access control features is the ability, by means of device access settings, to prevent users other than chosen individuals accessing the system using a particular terminal.

Alternatively all users except named individuals can be allowed to use a particular device. This is useful for example in preventing the system manager's username being used at all terminals except a particular one. Such protection would of course be additional to the protection provided by the System Manger's password.

### SLIDE 13

Users are identified by a catalogued username which can take one of three security levels: low, medium and high. A high security user may not submit batch jobs; high and medium security users must submit a password when logging in for a multi-access session.

Multi-access, or MAC passwords can be up to 12 characters long and are irreversably encrypted when stored at the catalogue user 'node' for login comparison. In this way sight of the stored version is made useless to the would-be penetrator.

The login sequence is very tightly controlled.

The would-be MAC user once having started the sequence will either obtain legitimate access within a certain time or cause the terminal to be locked out with an immediate security alarm at the Master Operator's terminal. Line breakdowns for example at this stage cause security violations. So pulling the plug out won't do him any good.

A reverse password facility is also available with which the system can be made to identify itself to the user.

Another feature is program controlled access to files using ACR levels. By this means, installation management can force access to chosen files through installation written software which can perform auxiliary protection checks, for example file passwords.

ICL markets the IDMS Database system (developed from the Cullinane Corporation design). By making use of ACR access control, IDMS has become one of the few secure databases systems commercially available today.

### SLIDE 14

Well, these are some of the main VME/B security features. Here they are, collected together.

We have:

- full use of the ACR ring protection system both by the operating system and potentially by the installation security management
- in store code unmodifiable - pure code

- extensive installation tailoring facilities and hooks
- central access control to all system objects

... and so on as shown here.

And, one final thing on VME/B's features:

We have just started to investigate the possibility of the provision of the ability to police a mandatory security policy, and it is looking reasonably straightforward to integrate into the existing security structure.

#### DESIGN AND PRODUCTION METHODOLOGY

##### SLIDE 15

From our experience of producing earlier operating systems we realized at the outset that one cannot simply treat an operating system as a collection of programs and then farm out the development of these programs to separate groups of programmers, hoping that they will all fit nicely together when the doomsday of integration approaches.

So we designed and built a system called CADES.

##### SLIDE 16

CADES is a methodology and a set of mechanisms to support that methodology. The VME/B design is top down data driven and hierarchic, and the prime objective of CADES was that the product was designed before it was implemented. We all know how difficult that is in the pressures of a commercial production environment.

The design methodology is then supported by mechanisms which may consist either of well-established rules for human actions and interactions (we call them the CADES Design Rules), or software products to be used as tools by the designers and implementors.

The hierarchies of modules and data structures with their attributes and relationships are stored in the CADES database, and this forms an authorized description of the product as it is being developed. The final content of the database is the product itself so there is no break in continuity between design and production. The ultimate objective of CADES was to support the total software development cycle from initial design right through to successive releases of the system with supporting documentation.

VME/B is a result extensively documented in a structured manner in a microfiched multi-volume library known as the "Project Log." For example, systems wide cross reference listings of data object usage and procedure calling structure are available, and can be automatically reproduced for all new releases using the CADES database.

A very good description of CADES can be found in the May 1980 edition of the ICL Technical Journal.

It is important to say at this point that CADES does not have the richness of design language nor the degree of formalization enjoyed by the formal languages that everybody here is familiar with. It was never intended to be used as a basis for later design correctness verification.

Nevertheless, ICL finds CADES an invaluable practical tool, and we are continuously developing, enhancing, and, possibly most importantly, using it.

The implementation language of VME/B is called S3. I haven't the faintest idea why. It is a development of ALGOL 68. In other words, it is a well-structured high level language with moderately strong typing and a block structure very suitable for the 2900 stack architecture.

The production teams, however, actually Code VME/B in an implementation level enhanced System Description Language, or SDL, which is automatically converted into S3 source by the CADES system. Niggling little things like complex data mode declarations, interfaces parameter specifications, constant and failure code values, macro expansions, and so on, are thereby automatically looked after by CADES.

#### SLIDE 17

The slide shows an example of some implementation level SDL. It is actually some SDL for a module which is part of the CADES system itself. We now use CADES to design and build CADES!

I won't describe the slide in detail, it's there just to give you a flavour of the language. At the top are the EXT and IO sections which list the procedures that this procedure calls, and the external data areas referenced. The interface definitions and modes of these items are of course all held in the CADES database. The asterisks at the beginning of some of the lines in the FUNCTION section trigger off various substitution and validation actions that occur when the system is converting this code into S3.

The existence of centrally held definitions of system-wide objects like data mode declarations and interface specs and so on automatically reduce, of course, the problem of mismatches in all of these areas.

An important current CADES development is the provision of an enhanced SDL/PASCAL back end. ICL holds the view that whenever possible, software

products should be written in high level languages, and PASCAL is one that we have chosen to be heavily used, particularly in the production of non-mainframe software. The compiler has been structured to enable a number of different target object codes to be produced. In developing the PASCAL aspect of CADES we have incorporated a number of the good features of the ADA development environment, for example the separation of package specs from package bodies.

One final random point of interest, to do with the CADES design there are no "GO TO's" in VME/B, well, hardly any!

#### SLIDE 18

I would just like to finish off now with a brief survey of some of the additional security work that has been and is being undertaken on VME/B.

Obviously some of our customers have special security requirements, so the first thing to say is that a substantial number of extra security features have been developed to satisfy them. Another objective has been to 'harden' VME/B not only from the point of view of extra facilities but also from the point of view of correctness.

Of course everybody benefits from correctness improvements, but also some of the additional facilities have since become standard product line items.

To further our 'correctness' objective, we have been subjecting the primitive architectural features and low level operating system features to 'theoretical' analysis, backed up where appropriate by actual tests. This is a continuous process, since new releases of the operating system are continually providing new areas to be examined. For this reason we are attempting to automate the analysis as much as possible.

Most of the tools developed in this work are incorporated into a "security test package" which also incorporates tests of the standard user visible security facilities. The package is now being applied during the acceptance testing phase of each new release of the product.

We also maintain a close relationship between ourselves and our major secure users and conduct regular meetings devoted to examination and discussion of security issues at both technical and non-technical levels.

Every so often, we stand back and look at the overall security structure of VME/B. One current example is a development of the security control object dependency graph idea developed by Linde at SDC, which we hope will be found useful in identifying areas requiring most attention. Another example is an examination of the feasibility of restructuring the operating system in minor ways to enable the control of security to be more localized (note I might add to the extent of a security kernel's localization).



An early hardened version of VME/B was subjected to a 'tiger team' attack a few years ago with encouraging results. In that attack, the system demonstrated a reasonable degree of security in that the attack team failed to achieve their major penetration objectives.

I should add that at the time there were a small number of known defects declared as 'no go' areas, and others that had to be compensated for by appropriate rather restrictive procedural controls. We have, of course, since cleared these defects.

I would be foolish though to claim that the system is now therefore totally secure, but it at least shows that the claim that it is 'easy' to penetrate a modern well structured commercial operating system has to be examined very carefully. The great majority of successful penetrations have been by teams consisting of top class systems penetration specialists. It has been said that the ideal qualification for a member of a penetration team is that he should be "a negative thinking anarchist with an IQ of 150 and the patience of Job." Such people are hard to find. What is easy for them might well prove impossible for ordinary mortals.

A system that has been penetrated by specialists, and VME/B might well be one day, cannot be dismissed as being insecure. Security is not a binary property that is either present or not, and this has of course been clearly recognized in Grace Nibaldi's valuable work on this subject.

#### SLIDE 19

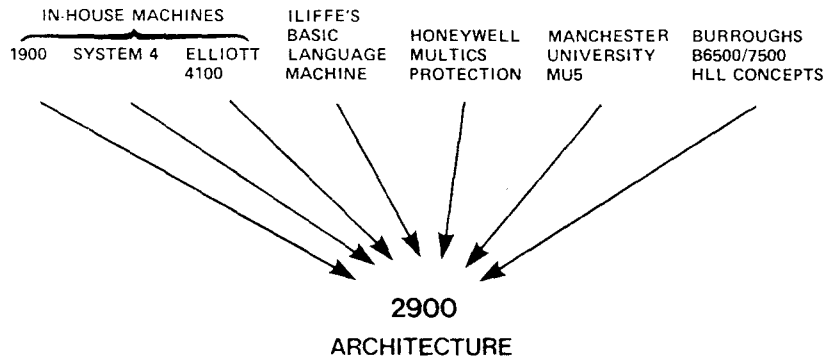
Well that's about it. As you can see we take a pragmatic approach to security; it has to be pragmatic on a system as large as VME/B. We make no claims of absolute security. All we can do is fill as many holes in the colander as our expertise and the state-of-the-art, allows.

The architectural bedrock on which VME/B lies is sound. The operating system itself has been produced using modern software engineering techniques, and the VME/B user has always been considered 'malicious'! We know of no comparable but more secure system.

**ICL** MENU

- BACKGROUND AND ORIGINS
- 2900 ARCHITECTURE - PROTECTION FEATURES
- VME/B SECURITY FEATURES
- SECURITY ENHANCEMENT WORK

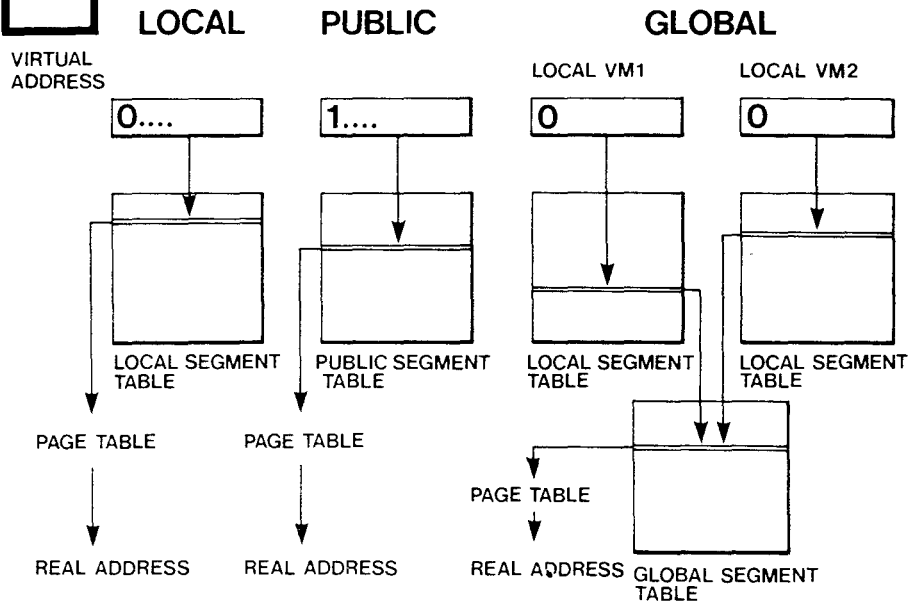
**ICL** ORIGINS AND INFLUENCES



**ICL** FEATURES OF THE 2900 ARCHITECTURE

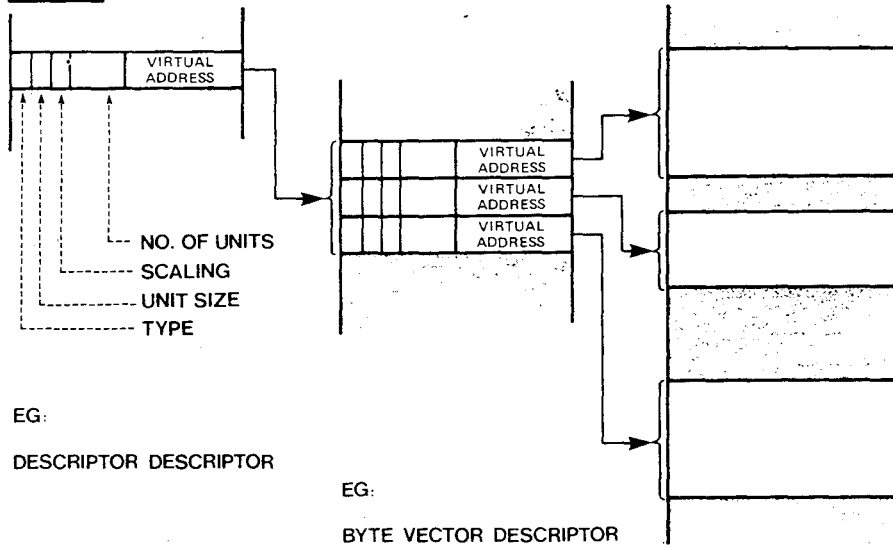
- VIRTUAL ADDRESSING
- DESCRIPTORS
- IMAGE STORE AND INPUT/OUTPUT CONTROL
- ACR LEVELS AND THE VALIDATE INSTRUCTION
- SYSTEM CALL MECHANISM
- THE STACK

**ICL** VIRTUAL ADDRESSING





# DESCRIPTORS



## IMAGE STORE

### VISIBLE REGISTERS

PC PROGRAM COUNTER  
LNB LOCAL NAME BASE  
RTC REAL TIME CLOCK  
DR DESCRIPTOR REGISTER  
ACC ACCUMULATOR  
~  
ETC  
~

### INVISIBLE REGISTERS

SSR SYSTEM STATUS REGISTER  
PSR PROGRAM STATUS REGISTER  
LSTB LOCAL SEGMENT TABLE BASE  
PSTB PUBLIC SEGMENT TABLE BASE  
~  
ETC  
~  
EXTERNAL DEVICE REGISTERS  
~  
ETC  
~



## PROTECTION LEVELS

- ACCESS CONTROL REGISTER (ACR)
- SEGMENT ACCESS CHECKS
  - READ ACCESS
  - WRITE ACCESS
  - CHANGE ACCESS
  - EXECUTE PERMISSION BIT
- SYSTEM CALLS
  - CALLING ACR LEVEL CONTROLS
  - ENFORCED ENTRY AT PROPER ENTRYPOINT
  - HARDWARE SUPPORTED PARAMETER VALIDATION

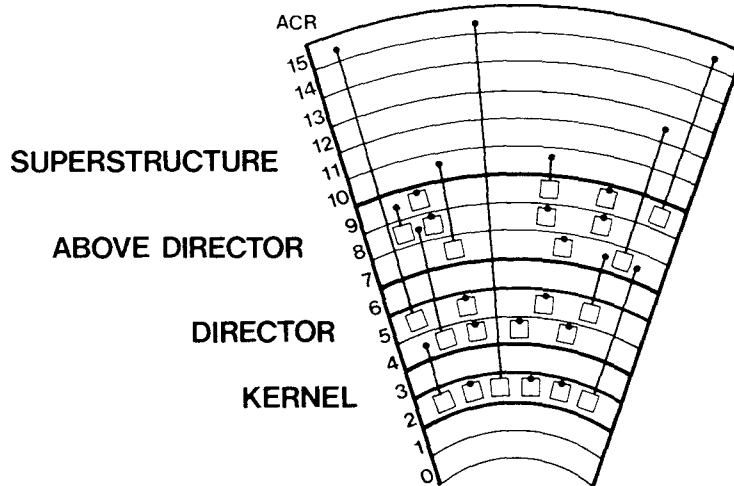
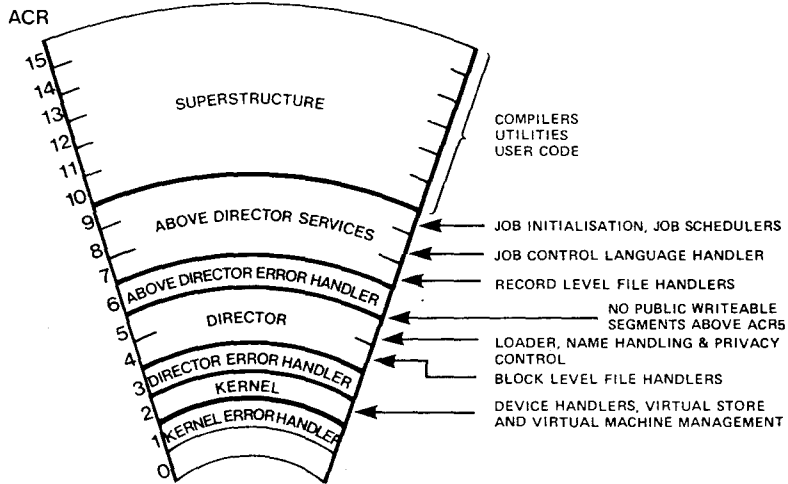


## FEATURES OF THE 2900 ARCHITECTURE

- VIRTUAL ADDRESSING
- DESCRIPTORS
- IMAGE STORE AND INPUT/OUTPUT CONTROL
- ACR LEVELS AND THE VALIDATE INSTRUCTION
- SYSTEM CALL MECHANISM
- THE STACK

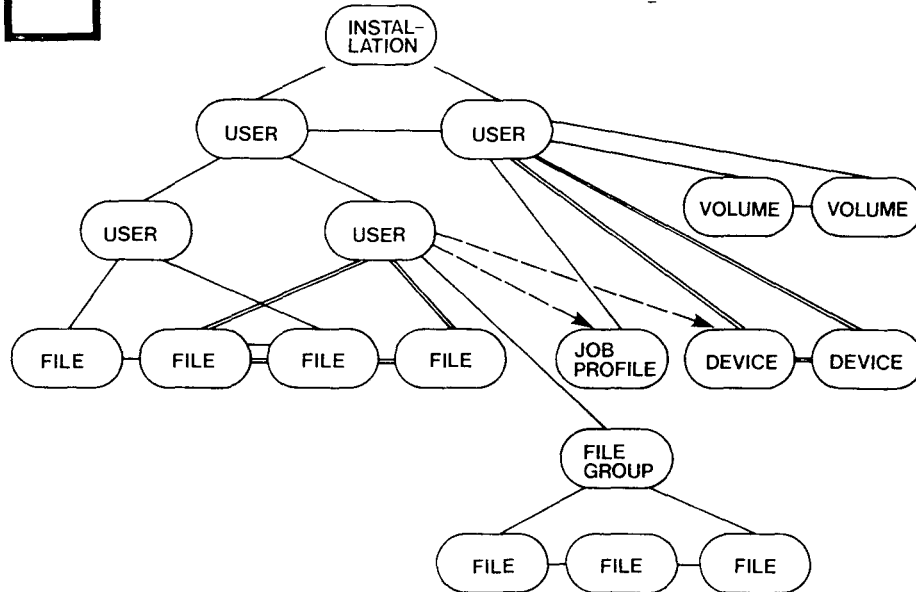


# VME/B STRUCTURE





# THE CATALOGUE



## MORE VME/B PROTECTION FEATURES

- USER AUTHENTICATION
  - 12 CHARACTER PASSWORDS
  - ONE WAY ENCRYPTION
  - SUCCESS OR LOCKOUT WITH ALARMS
  - TIMEOUT DURING LOGIN
- REVERSE PASSWORDS
- ACR PROTECTED FILES
  - SECURE IDMS DATABASE SYSTEM



## MAJOR VME/B SECURITY FEATURES

- STRUCTURED USE OF ACR PROTECTION WITH:
  - KERNEL
  - DIRECTOR
  - ABOVE DIRECTOR
  - SUPERSTRUCTURE
- SUPERSTRUCTURE (USER'S CODE) SUBDIVIDABLE BETWEEN 6 ACR LEVELS
- PURE CODE
- INSTALLATION ABILITY TO DEFINE WHICH O/S FACILITIES ARE TO BE AVAILABLE TO WHICH USERS
- ADDITIONAL INSTALLATION HOOKS
- CENTRAL TOTAL ACCESS CONTROL VIA FILESTORE CATALOGUE OF SYSTEM OBJECTS
- LOGGING
- STRINGENT USER AUTHENTICATION PROCESS
- REVERSE PASSWORDS
- INSTALLATION PROGRAMMABLE FILE ACCESS CONTROLS



# COMPUTER AIDED DEVELOPMENT and EVALUATION SYSTEM





## **CADES**

### **CHARACTERISTICS**

- **A METHODOLOGY AND A MECHANISM**
- **TOP DOWN, DATA DRIVEN, HIERARCHIC DESIGN**
- **FORMAL CAPTURE OF DESIGN ON AN IDMS DATABASE**
- **COMPLETE DEVELOPMENT CYCLE SUPPORT**
- **SDL LANGUAGE**



```
1  HOLON : EN_PARAMETER_VALIDATOR;
2  VERSION:007;
3  EXT  : COMMON : CHECK,
4          MOVE,
5          SCANUNQ,
6          ;
7          SCLMAC : TRANSLATE_HIERARCHIC_NAME,
8          ;
9          : EN_OUTPUT_MESSAGE,
10         EN_EXTEND_HEAP,
11         ;
12  IO   : E :
13         EN_OUTPUT_PHASE,
14         EN_OUTPUT_LIBRARY_OPEN,
15         EN_TRUSTED_USERNAME,
16         EN_TRUSTED_USERNAME_BUFFER,
17         EN_TRUSTED_USERNAME_CURRENCY,
18  P   : HOLON_NAMES,
19         SELECTORS,
20         NON_STD_PHASES,
21         GP,
22         ENV_DE,
23         ACTNAME;
24  FUNCTION :
25  BEGIN
26  (32)BYTE ANB :=
27  (L) EN_ALPHANUMERIC_BREAK,
28         AN :=
29  (L) EN_ALPHANUMERIC,
30         NRMC :=
31  (L) EN_NUMERIC;
32  RESULT_CODE :=
33  (L) SUCCESS;
34  UNLESS
35  HOLON_NAMES IS NIL
36  THEN
37  FOR I TO BOUND HOLON_NAMES
38  DO
39  REF()BYTE HN IS HOLON_NAMES(I);
40  UNLESS
41  LENGTH HN LE 32
42  AND
43  CHECK(ANB,HN,D,NIL)
44  AND
45  IF
46  LENGTH HN = 0
47  THEN
48  TRUE
49  ELSE
50  HN(0) GE 'A' - ALPHABETIC 1ST CHAR ?
51  AND
52  HN(0) LE 'Z'
53  FI
```



## **FURTHER SECURITY WORK**

- **ADDITIONAL SECURITY FEATURES**
- **THEORETICAL ATTACK**
- **SECURITY TEST PACKAGE**
- **REGULAR REVIEWS OF USER REQUIREMENTS**
- **SECURITY STRUCTURE APPRAISAL**
- **ATTACK EXERCISE**

GNOSIS: A PROGRESS REPORT

BOB COLTEN  
TYMSHARE

Thank you, Steve. We must have done something right last year to get invited again.

Before my prepared remarks, I'd like to briefly comment on the subjects raised by the preceding speakers: We fully agree with the message expressed by Steve, by Terry Cureton, and others. You have to walk before you run. We also agree with Axel Vidtheldt that availability is a part of security; and with CDC that you can't forget performance or customer need. With that, let me proceed to this presentation.

This year it is our intention to bring you up to date on the progress we have made since last years presentation.

For those of you who weren't here last year, we'll briefly review Gnosis.

This will be only the 25¢ tour. Then we'll tell you about the benchmark.....

- . Why we selected it
- . How it was implemented
- . What was the performance
- . We'll tell you what we learned and....
- . Where we are going from here and why

Then Norm Hardy, the Architect of Gnosis will discuss some of our concerns with the mechanisms and approaches of the secure system evaluation effort.

- . We disclosed Gnosis to this audience at the third DOD security initiative seminar.
- . Gnosis was started by Tymshare as an in-house research program in 1975.
- . It is a capability based operating system designed to run on 370 type architecture. It was started by a tiny team which has expanded to a small team which now consists of six people.

The Gnosis design objectives were and still are to:

1. Protect proprietary applications, both programs and data.
2. Provide a high performance environment for transaction oriented applications.
3. Simplify and reduce the cost of maintaining applications.
4. Provide an operating environment in which applications can be easily maintained.
5. Improving programmer productivity in developing new applications.
6. Provide a facility for developing fault tolerant applications.

In summary, we wanted to develop a product to enable us to enter new markets. An operating system that is easy to:

- |            |                |
|------------|----------------|
| 1. Learn   | 3. Debug       |
| 2. Program | 4. and Protect |

For those of you who weren't here last year, what follows is the 25¢ tour.

First, let me briefly contrast the Gnosis architecture with the architectures of systems which we are all familiar with and love dearly.

In most existing systems, applications are located in the same memory space. On the slide that is shown on the left side, state pack are all in the same space. If one part of the application has a bug in it, it is not unlikely that it will impact the entire application or even other applications.

This type of breakdown is not only inconsiderate but downright rude. It also tends to actively promote paranoia.

In Gnosis, we keep not only applications but components of applications separated in separate domains. In Gnosis, each element can be totally isolated from every other element. Each of these separate elements is a domain. The only way a user or another program can access a domain is through an explicitly authorized capability.

If you want to know more about Gnosis you can find more in last years proceedings or the Mitre evaluation or write to me or Norm at Tymshare.

Let me now briefly contrast our situation today with our situation when we last met in 1980.

Today our development objectives are different from those we had in 1980.

At that time we were---

- . Looking for external applications
- . Looking for some kind of support

We obtained moral support from the evaluation center team.... and this support has been a factor in motivating us to accelerate the Gnosis development effort. More about the contribution of the evaluation center team from Norm.

What we are focusing on now are internal applications. We are not looking for external financial support.

We are now focusing on:

- Developing new tools to facilitate application development
- Producing documentation to help users implement applications and actually implementing new application, as well as,
- Measuring the performance of the new applications.

Now I'd like to briefly talk about the benchmark.

As you probably know, one of the most frequently cited reasons for not using capability based systems in the past, was that their performance was rotten.

We therefore, had to find an application that was both real, as well as typical of a class of applications in which many users performed a small number of activities simultaneously.

We wanted the test to be a multi-purpose test. The individual selected to do the evaluation was someone outside the project who works for another division. The management of that division wanted to find out if Gnosis would run the kinds of applications they wanted to implement.

- . Thus we and they both wanted to test the reliability of the system under stress.
- . We both wanted to know how the system would perform under various loads.
- . They wanted to know if Gnosis had the functionality to run their applications.
- . They were concerned that Gnosis was so different that normal people couldn't use it.

- . They wanted to find out how good the documentation really was, namely, could their development people use it.
- . They also wanted to know how much sharing was feasible and how easy it was to implement in an application.
- . Finally they didn't want to spend a lot of money or time to find out if the system was viable and useful on an application development environment.
- . And, we wanted an application that was fun to implement.

Because of all these reasons, we selected "Adventure", a program written in PL/1, as the first test program to be implemented in Gnosis. "Adventure" is a game similar to "Dungeons and Dragons" with a specific cave called the colossal cave.

Now I'd like to show you what functions were involved in building the selected application and how much code you need to trust.

This part of the system predates adventure and this is the only part needed to implement the adventure application.

It consists of the kernel and two separate domains, the terminal interface and the receptionist.

The kernel, unlike most familiar operating systems, is small and very simple; it functions more as a control program rather than as a conventional operating system. It runs in supervisor mode, it is unswappable. The kernel maintains the extended machine architecture, provides the basic building blocks and performs operations on them on behalf of the user. In Gnosis, capabilities and data are isolated from each other so that capabilities which only the kernel can access directly cannot be forged or manipulated without authority. We wanted to test the kernel under stress.

The terminal interface system provides the path for a terminal to communicate with the adventure game; it converts ASCII code to EBDIC and EBDIC to ASCII.

The receptionist verifies the identity of the caller and the destina-



tion on the domain that is being called.

Next, we built the adventure domain and hooked it to the terminal interfact domain.

The adventure domain contained the same PL/1 program we ran under CMS.

There was an unresolved situation at this point. When the line hung up we were left hanging, so we built the line monitor to:

- A). Recognize the event
- B). Take the appropriate action

Well, in the crudest sense, this is all you need to have a single adventure.

But, we assume that like most people, this audience is jaded and having developed a taste for adventure you would like to repeat the experience and maybe have some friends join you in the game.

The slide shows that if you want more than one adventure, you need to duplicate the adventure and the line monitor domains.

It is important to point out however, that each line monitor and each adventure share most of their code and their data in read only mode.

Thus, when more than one player plays simultaneously, it is necessary to create multiple instances of adventure. Code is shared in each instance of adventure between the terminal interface and the receptionist.

The adventure domains can also share data which is common to all users. For example, the description of the cave.

Thus, the entire amount of storage space required for each instance of adventure is only 10 pages of real memory per user, most of it is PL/1 variable storage. An interesting note--the adventure program was not modified to run on Gnosis even though it was not designed for sharing.

To build the multiple instances of adventure and the line monitor, and to implement policy of what to do when a line disconnects, a new mechanism had to be built. This mechanism we called the adventure controller.

The adventure controller creates more instances of adventure, gives them keys, (no one else has keys), namely rights to access.

The adventure controller also implements the policy for disconnecting terminals. When a line is dropped, the line monitor notifies the adventure controller. The adventure controller then destroys the instance of adventure and reclaims all resources owned by that instance.

In addition to its existing functions, the adventure controller could be used to:

- . Monitor resource consumption by each user.
- . Insert debugging tools.
- . Insert auditors for each user.

As is perhaps more evident in this slide, the entire structure of adventure, operating within the Gnosis environment, is very simple. A total of only 600 new lines of code had to be written to make it work. 500 lines for the adventure controller and 100 lines for the line monitor. In this application one must trust the kernel, the terminal interface, the receptionist, and the adventure controller and nothing else.

There is no operating system as such which needs to be trusted.

- No virtual machine
- No command language
- No loading of programs
- No file system
- No editor
- No system libraries

Another way of saying this is that when playing adventure in the Gnosis environment, "The tail does not wag the dragon." If Marv Shaefer were here, he'd say that was a troll remark. I would call him a bad gnome.

Now that you know the architecture of the adventure benchmark, let's look at how long it took to implement it.

In reality, the whole thing took a little over one month, if you don't count the first month to get oriented.

It took 1 man week to do the controller  
the line monitor and  
the linkages  
and two more man weeks to do the multiple version of adventure. Then two more man weeks to do the driver and the scripts.

Now I'd like to share with you some of the comments made by the developer of the benchmark in his report to his manager. The developer is a senior applications programmer. Although he had experience on many different operating systems, he had no:

- . Capability-based system experience
- . Gnosis experience
- . Experience with Gnosis debugging tools
- . Prior contact with the Gnosis team

### GNOSIS FUNCTIONALITY

- \* "Application programmers can learn to use Gnosis in a relatively short time"
  
- \* "PL/1 programs can be run under Gnosis with very little source code modification"

He also noted a few shortcomings:

- . Documentation as presently available is unfit for man or beast.
- . System still needs work.

Now let's look at the results of the benchmark.

First note the CMS baseline: We used CMS because it was available. Note: The vertical axis should read cumulative or total transaction rate.

The test was conducted on a 370/158 MOD I with 5 megabytes of memory located in Dallas.

A transaction generator was used to generate one transaction per second per user. (That is shown by the 45 degree line.)

Each CMS transaction needed 77 pages of memory per user and used up 150 milliseconds of CPU time allowing a maximum of 6.4 transactions per second.

In this slide, we use the CMS line for comparison. We ran 3 tests on subsequent weekends.

- . The 1st test labeled 6/1/81 looked pretty bad.
- . In the 2nd test on 6/7/81, we reduced resources required for each transaction.
- . The original Gnosis version used PL/1 reformatted I/O to write each line on the terminal.

PL/1 terminal I/O took up more than half the CPU cycles. We wrote a subroutine to replace the PL/1 language call to a subroutine.

Compatible to the one is CMS.

Test 2 performance dropped off precipitously due to thrashing (fixed tables in kernel were not balanced).

In test 3, on 6/13/81, we understood and partially resolved thrashing by better balancing of tables in the kernel.

While it may appear that Gnosis and CMS performed about equally, it is important to note that the CMS tests were running with 100% CPU utilization while Gnosis tests ran with much excess CPU capacity.

The 6/13/81 line in this slide shows what the total transaction rate would have been if the transaction driver had been able to run fast enough to saturate the CPU.

At this point, we stopped making changes in the system since we had met our objective to beat CMS.

We are now convinced that we can further improve Gnosis performance and that Gnosis can be competitive with other I M transaction systems.

We are aware of many other improvements which could be made by reducing system overhead as well as the cost per transaction.

The top line in this figure could be achieved with 1 man month of work to reduce system overhead by removing additional thrashing bottlenecks in the kernel.

In addition, we could increase the transaction rate another 50% if we optimized terminal transactions by developing a high performance terminal interface and by optimizing adventure to allow even more sharing.

These actions would take 2 additional man months and would reduce transaction time to 30 MS, reduce memory per transaction to 10 pages and yield 30 transactions per second on a 370/158 CPU. On an IBM 3033 this would mean a potential of between 100 and 150 transactions per second.

This is comparable to all, but the fastest IBM transaction processing systems.

Tymshare is increasing the level of support for Gnosis. We have been authorized to hire more people immediately.

- . Gnosis is moving from R & D to development status.
- We have two applications.

The first is:

- \* A switch which is designed to enable users to access applications which run on multiple computers without effort or awareness on the users part.

The second is:

- \* A transaction processing system for a transportation agency which will be continuously updated and accessed by many users simultaneously.

In summary.... what we plan to do during the next year is the following:

- \* Implement more complex systems.
- \* Have a system which is continuously and routinely operational.
- \* Develop tools to facilitate the implementation, debugging monitoring and operating the new applications.
- \* Insure that the new tools are general purpose and that they enhance programmer productivity performance and the reliability of Gnosis.

Finally, we are now convinced that Gnosis will evolve into a system which will be ready for general use in two to three years.

It is important to note, especially for this audience that:

- \* The Gnosis architecture inherently provides a base for a trusted environment.

However, Tymshare's approach to achieve the trusted system objective is different from the traditional approach. And it is not clear at this time how the currently accepted trusted system model can be mapped into the Gnosis architecture.

- \* Norm Hardy, the architect of Gnosis, along with some Mitre people perceives a knowledge gap in this area.

- \* Norm is going to briefly address our concerns with the mechanisms and approaches of the secure system evaluation effort, not as a criticism of the process, but rather as a search for a broader set of perspectives.

Thank you and please help me welcome Norm Hardy to the podium.



GNOSIS :  
A PROGRESS REPORT

PREPARED FOR:  
THE FOURTH SEMINAR ON THE DOD  
COMPUTER SECURITY INITIATIVE PROGRAM

AUGUST 11, 1981  
TYMSHARE INCORPORATED  
CUPERTINO, CALIFORNIA

PRESENTATION OUTLINE

- \* GNOSIS REVIEW
- \* INITIAL BENCHMARK
  - SELECTION
  - ARCHITECTURE
  - PERFORMANCE
- \* FUTURE DIRECTIONS
- \* CONCLUSIONS AND CONCERNS

## INTRODUCTION

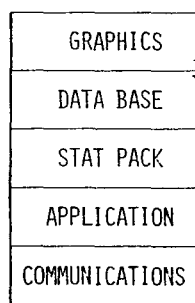
\* GNOSIS DISCLOSED 1980

\* DESIGN GOALS:

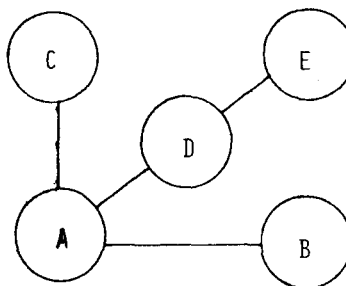
SECURITY  
PERFORMANCE  
SIMPLICITY  
MAINTAINABILITY  
PRODUCTIVITY  
FAULT TOLERANT

## APPLICATION ARCHITECTURE

CONTEMPORARY  
APPLICATION  
ARCHITECTURE



GNOSIS  
APPLICATION  
ARCHITECTURE

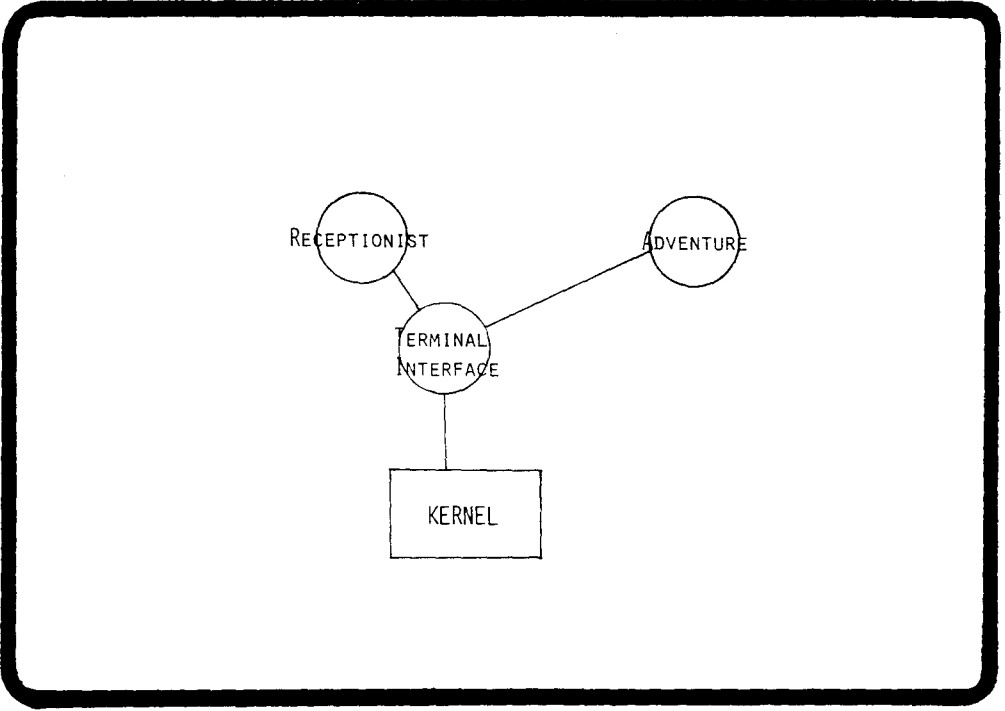
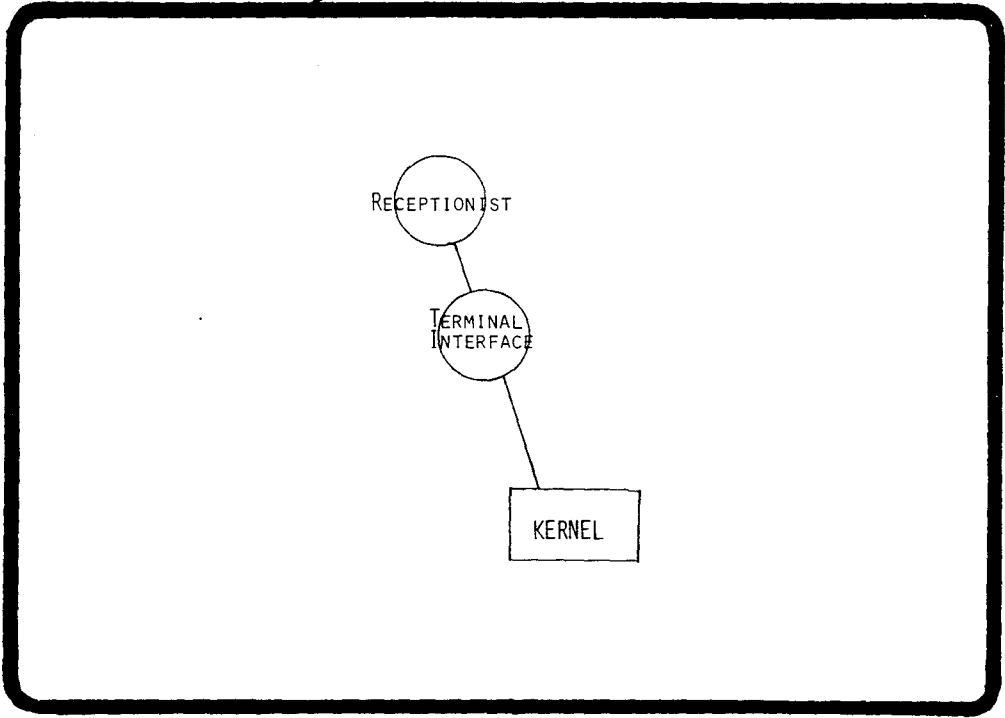


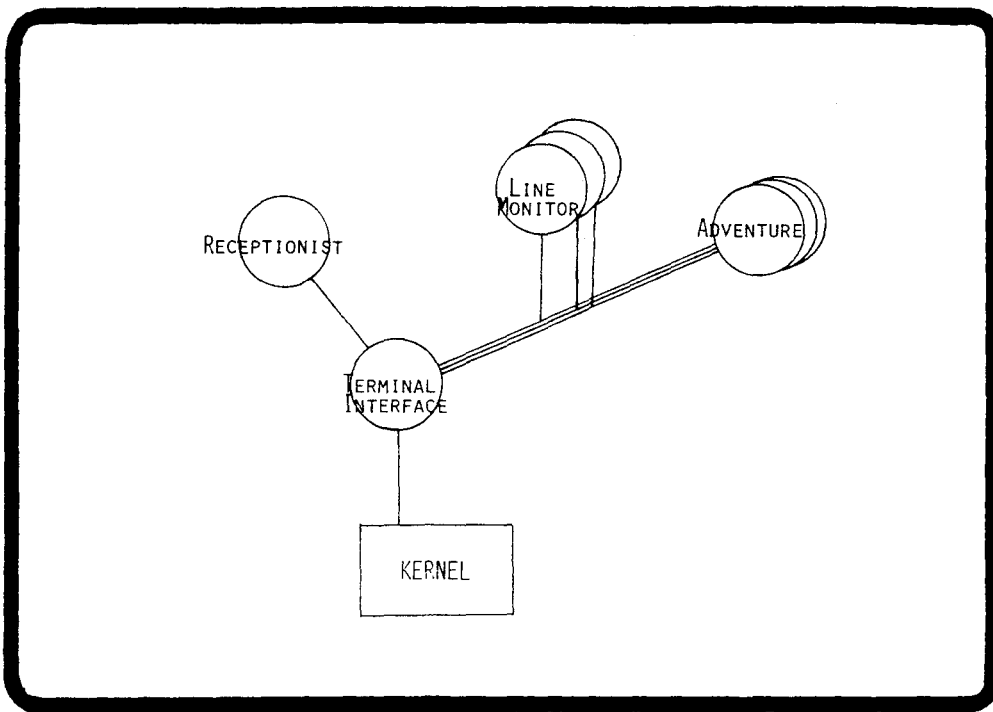
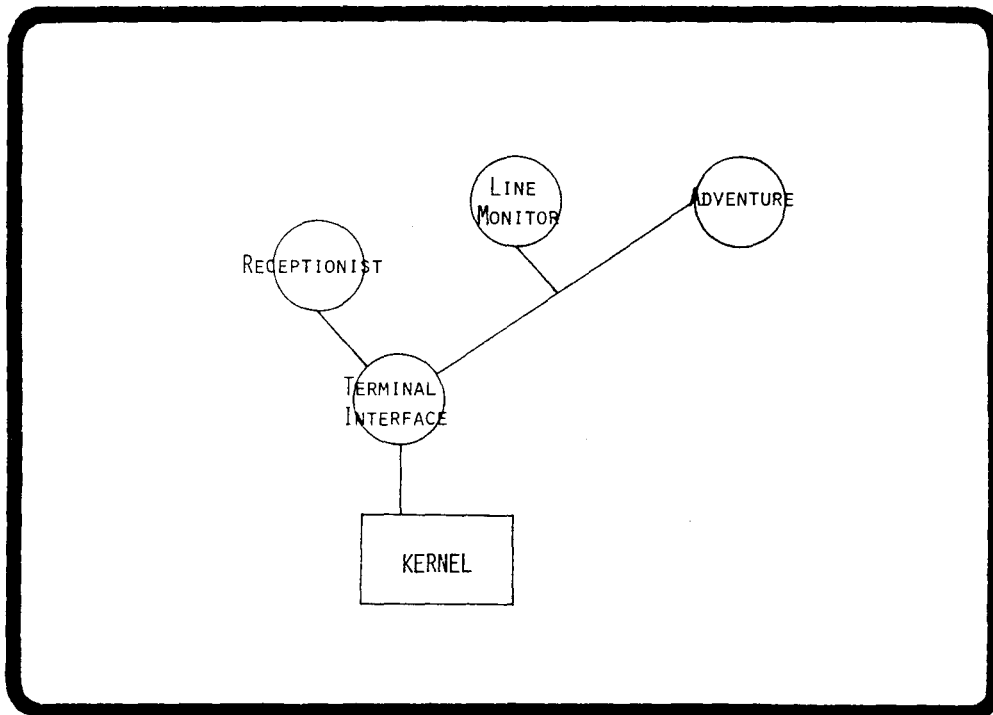
1981 STATUS

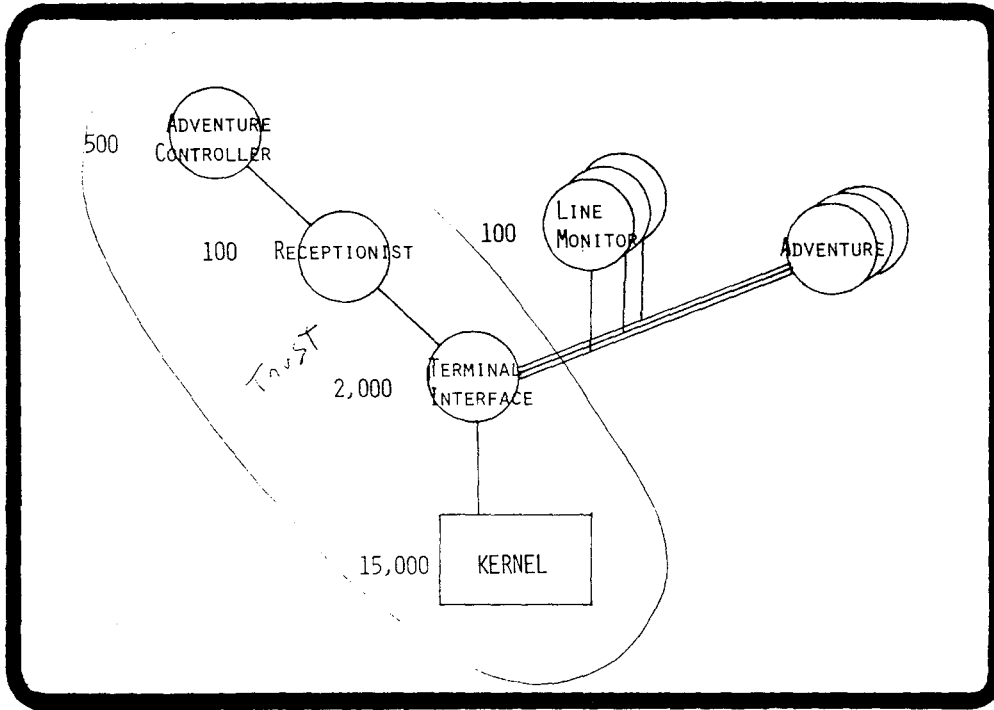
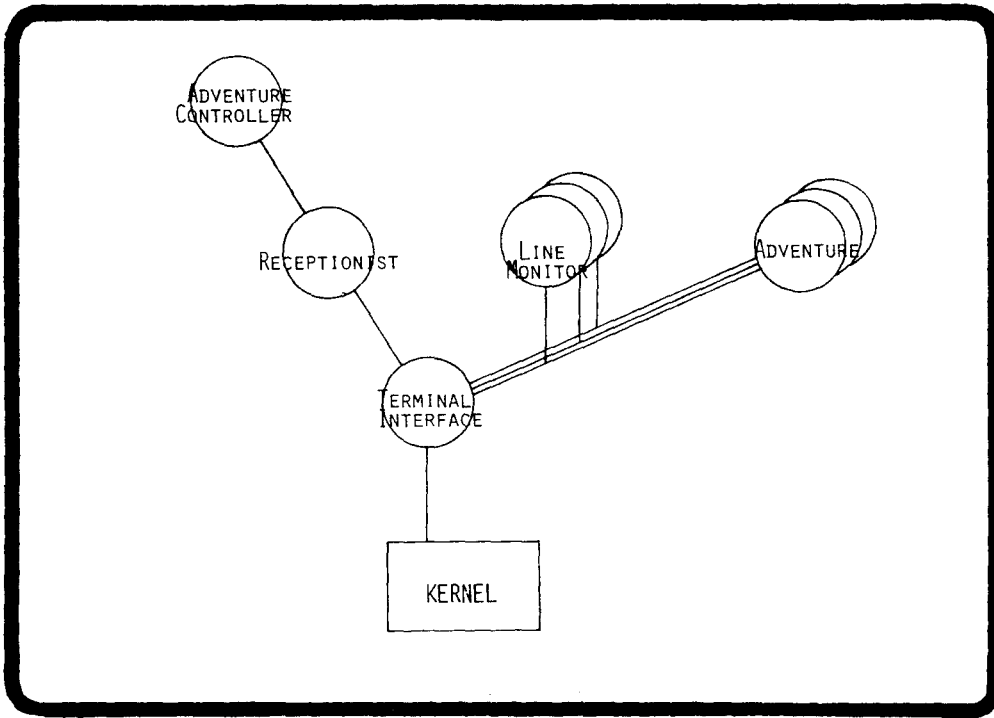
- \* CLEAR DEVELOPMENT OBJECTIVES
- \* FOCUS ON:
  - DEVELOPMENT TOOLS
  - DOCUMENTATION
  - IMPLEMENTATION OF APPLICATIONS
  - MEASUREMENT

SELECTING THE BENCHMARK

- \* REPRESENTATIVE
- \* MULTIPLE PURPOSE TEST
  - STRESS
  - PERFORMANCE
  - FUNCTIONALITY
  - USABILITY
  - DOCUMENTATION
- \* RESOURCE SHARING DEMONSTRATION
- \* INEXPENSIVE







BENCHMARK IMPLEMENTATION CALENDAR

- \* READING, LEARNING 1 MONTH
- \* SINGLE ADVENTURE 1 WEEK
- \* CONTROLLER FOR MULTIPLE ADVENTURES 2+ WEEKS
- \* BENCHMARK DRIVER & SCRIPTS 2+ WEEKS

GNOSIS FUNCTIONALITY

- \* "GNOSIS IS ALIVE & WELL AND CAN BE USED AS A BASE FOR MULTIUSER APPLICATIONS"
- \* "GNOSIS DOES NOT CONSTRAIN APPLICATION DESIGN - HIGHLY ADVANTAGEOUS TO THE DEVELOPMENT OF COST COMPETITIVE APPLICATIONS."

GNOSIS FUNCTIONALITY

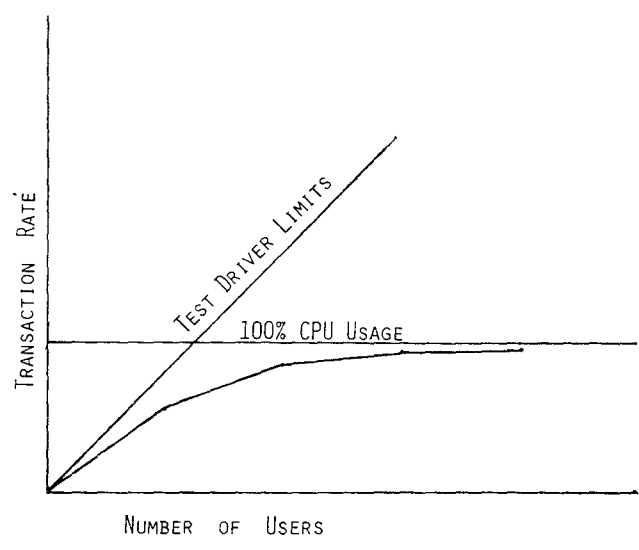
- \* "APPLICATION PROGRAMMERS CAN LEARN TO USE GNOSIS IN A RELATIVELY SHORT TIME"
- \* "PL/1 PROGRAMS CAN BE RUN UNDER GNOSIS WITH VERY LITTLE SOURCE CODE MODIFICATION"

GNOSIS SHORTCOMINGS

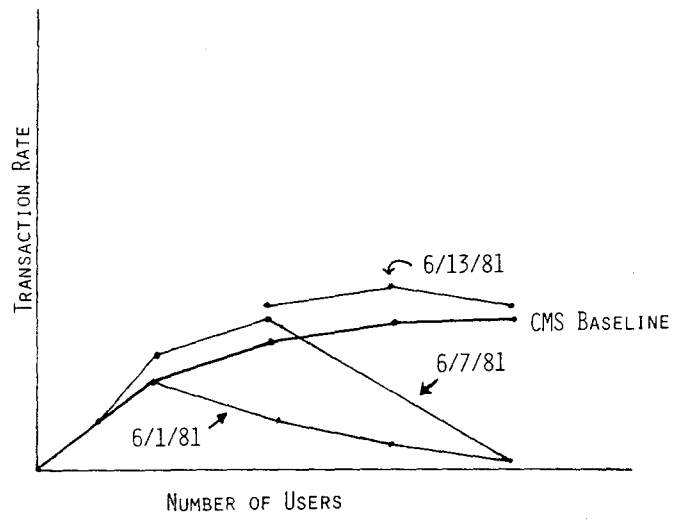
- \* "DOCUMENTATION IS UNSUITABLE FOR APPLICATION DESIGNERS AND PROGRAMMERS."
- \* "SYSTEM IS INCOMPLETE AND MORE FACILITIES ARE NEEDED TO IMPLEMENT NEW APPLICATIONS."



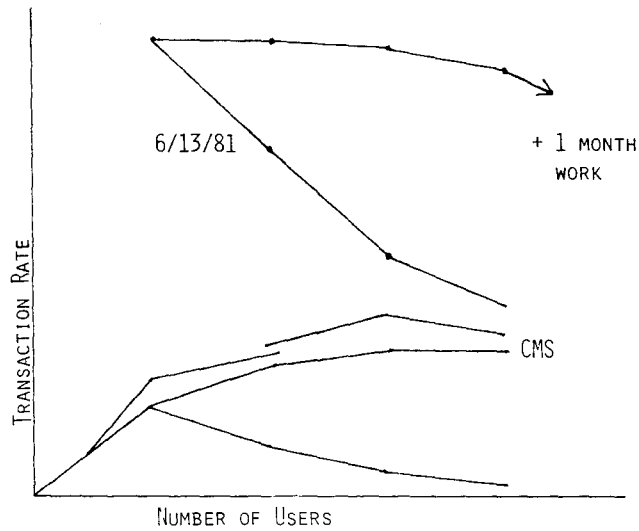
CMS PERFORMANCE TEST



GNOSIS BENCHMARK RESULTS



MAXIMUM GNOSIS CAPACITY

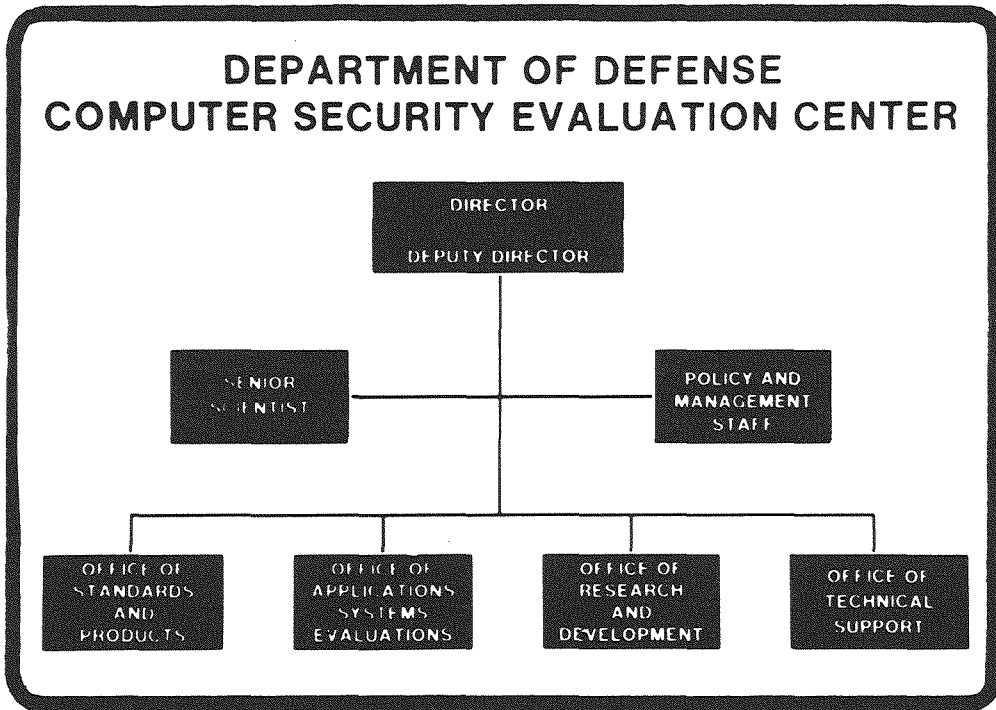
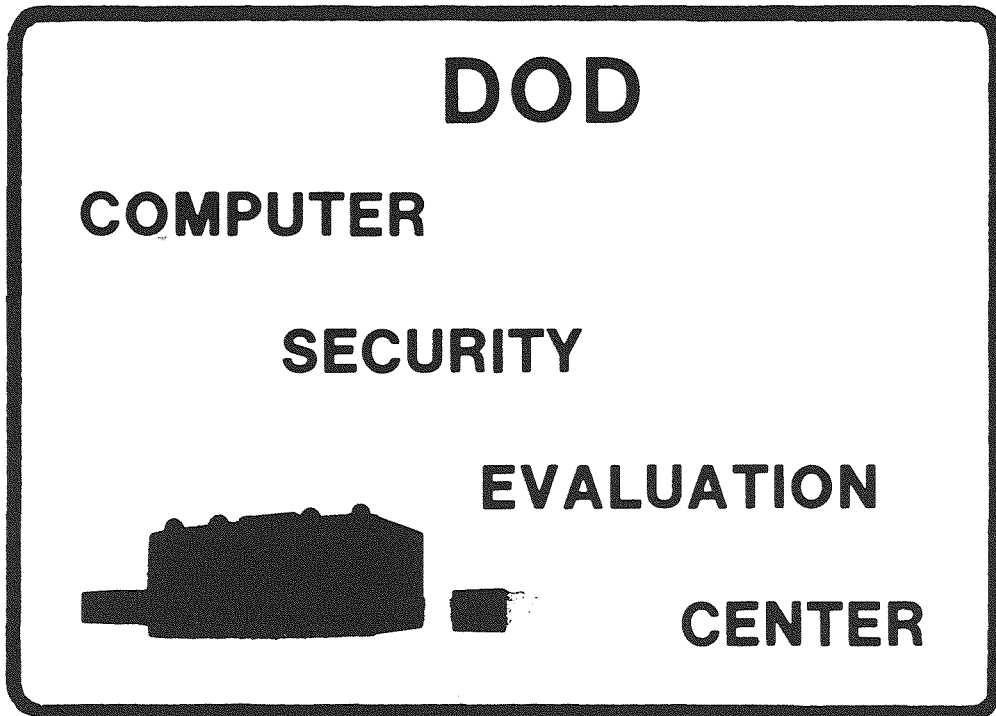


FUTURE PLANS AND DIRECTIONS

- \* INCREASED SUPPORT
- \* FROM R & D TO DEVELOPMENT
- \* TWO TEST APPLICATIONS
  - SWITCH
  - TRANSACTION PROCESSING SYSTEM
- \* CONTINUOUS SYSTEM OPERATION
- \* NEW GENERAL PURPOSE TOOLS

George Cotter

DoD Computer Security Evaluation Center



## **A LOOK AHEAD . . .**

- **PRODUCT EVALUATIONS**
  - **CONTRACTUAL SUPPORT**
  - **CDC CYBER NOS**
  - **UNIVAC 1100 OS**
- **SYSTEMS EVALUATIONS**
  - **CONTRACTUAL SUPPORT**
  - **COMMUNICATIONS RELATED**
  - **DEFENSE SYSTEMS**
  - **INTELLIGENCE SYSTEMS**
- **RESEARCH & DEVELOPMENT**
  - **COMPLEX ENVIRONMENTS**
  - **NETWORKS**
  - **KERNELS**
  - **DBMS**
  - **MICROPROCESSORS**
  - **TOOLS**

## **CURRENT ACTIVITIES . . .**

- **PRODUCT EVALUATION**
- **SYSTEMS EVALUATION**
- **RESEARCH AND EVALUATION**

## **DOD DIRECTIVE . . .**

- **RESPONSIBILITIES OF DIRECTOR NSA AND OTHERS**
- **OTHER APPLICABLE DIRECTIVES**
- **COMPUTER SECURITY POLICY**
- **CENTER CHARTER**

## **STATUS . . .**

- |                                    |                     |
|------------------------------------|---------------------|
| • <b>DOD DIRECTIVE</b>             | - <b>IN DRAFT</b>   |
| • <b>ORGANIZATION</b>              | - <b>APPROVED</b>   |
| • <b>MANNING</b>                   | - <b>IN PROCESS</b> |
| • <b>FACILITIES</b>                | - <b>IDENTIFIED</b> |
| • <b>COMPUTER SECURITY PROGRAM</b> | - <b>FY83 POM</b>   |
| • <b>CURRENT ACTIVITIES</b>        | - <b>CONTINUING</b> |

## **FUNCTIONS . . .**

- **TECHNICAL INTERFACE AND SUPPORT**
- **CONDUCT EVALUATIONS OF INDUSTRY, GOVERNMENT PRODUCTS**
- **MAINTAIN EVALUATED PRODUCT LIST**
- **ESTABLISH AND MAINTAIN EVALUATION STANDARDS AND CRITERIA**
- **CONDUCT SELECTED COMPUTER SECURITY EVALUATIONS**
- **CONDUCT AND SPONSOR RESEARCH AND DEVELOPMENT**
- **CHAIR DOD COMPUTER SECURITY TECHNICAL CONSORTIUM**
- **SPONSOR COOPERATIVE EFFORTS, SEMINARS, WORKSHOPS**
- **DEVELOP CONSOLIDATED COMPUTER SECURITY PROGRAM**

## **THOUGHTS EN ROUTE . . .**

- **COOPERATION IS THE KEY INGREDIENT**

## **THOUGHTS EN ROUTE . . .**

- **COOPERATION IS THE KEY INGREDIENT**
- **CENTERS FUNCTIONS ARE TECHNICAL**

## **THOUGHTS EN ROUTE . . .**

- **COOPERATION IS THE KEY INGREDIENT**
- **CENTERS FUNCTIONS ARE TECHNICAL**
- **CUSTOMER SERVICE MUST DOMINATE ACTIVITIES**

## **THOUGHTS EN ROUTE . . .**

- COOPERATION IS THE KEY INGREDIENT
- CENTERS FUNCTIONS ARE TECHNICAL
- CUSTOMER SERVICE MUST DOMINATE ACTIVITIES
- **MUST PROVOKE COMMERCIAL DEVELOPMENT**

## **THOUGHTS EN ROUTE . . .**

- COOPERATION IS THE KEY INGREDIENT
- CENTERS FUNCTIONS ARE TECHNICAL
- CUSTOMER SERVICE MUST DOMINATE ACTIVITIES
- MUST PROVOKE COMMERCIAL DEVELOPMENT
- **EPL SHOULD NOT BE A THREAT**



## **THOUGHTS EN ROUTE . . .**

- **COOPERATION IS THE KEY INGREDIENT**
- **CENTERS FUNCTIONS ARE TECHNICAL**
- **CUSTOMER SERVICE MUST DOMINATE ACTIVITIES**
- **MUST PROVOKE COMMERCIAL DEVELOPMENT**
- **EPL SHOULD NOT BE A THREAT**
- **DOD RESEARCH TO PLUG THE GAPS**

## **THOUGHTS EN ROUTE . . .**

- **COOPERATION IS THE KEY INGREDIENT**
- **CENTERS FUNCTIONS ARE TECHNICAL**
- **CUSTOMER SERVICE MUST DOMINATE ACTIVITIES**
- **MUST PROVOKE COMMERCIAL DEVELOPMENT**
- **EPL SHOULD NOT BE A THREAT**
- **DOD RESEARCH TO PLUG THE GAPS**
- **PACE AND PRIORITIES SET CAREFULLY**

## TRUSTED COMPUTER SYSTEMS

### REIN TURN

#### THE RAND CORPORATION

Since June 1978 the DoD Computer Security Consortium has conducted a Computer Security Initiative program, with the goal of achieving widespread availability of "trusted ADP systems"\* for use within the Department of Defense (DoD), in other government agencies, and in the private sector. For the government, "widespread availability" means the use of commercially developed trusted systems whenever possible. Effective January 1, 1981, the Director of the National Security Agency (NSA) was assigned responsibility for the evaluation of computer security for the DoD and thus will serve as Executive Agent for the Computer Security Initiative. One of his functions will be the compilation of a DoD Evaluated Products List of trusted systems.

To date, the three major activities of the Initiative have been (1) coordination of DoD research and development efforts in computer security, (2) identification of efficient evaluation procedures for trusted operating systems and their uses, and (3) identification of incentives for the computer industry to develop trusted systems as part of its standard product lines. This report addresses the third task. It analyzes the needs for trusted computer systems in the civilian agencies of the federal government, in state and local governments, and in the private sector.

Protection is needed in computer systems to (1) safeguard assets or resources, (2) comply with certain laws and regulations, (3) enforce management control, and (4) assure the safety and integrity of computer-controlled processes or systems. Additional incentives for implementing trusted systems might be to realize operational economies, to achieve marketing advantages, and to enhance an organization's public image.

Protection of programs and data in computer systems involves a variety of physical, personnel, and hardware/software security techniques; administrative and operational procedures; and computer-communication security techniques. The most difficult task to date has been the development of trusted operating systems--a necessity

---

\*A "trusted" ADP (automated data processing) system is one that employs sufficient hardware and software integrity measures to allow its use for simultaneous processing of multiple levels of classified and/or sensitive information. See the Glossary of Technical Terms in Appendix A for other definitions.

in resource-sharing, multiuser systems to prevent users from interfering with each other and to control access to sensitive data files or processing operations. The trusted operating systems sought by the Computer Security Initiative Program have a high potential for providing a solution to many of these problems.

In general, the use of current computer security techniques entails some reduction of system throughput, as well as some modification of existing application software or data bases. Some potential users of trusted systems are concerned about these impacts on their existing computer applications. However, there is a clear trend in computer hardware architectures and in software development toward including features that would be very useful for implementing performance-effective trusted systems; thus, performance loss is likely to be far less of a problem in the future. Conversion requirements for application software can also be reduced by designing trusted systems to be compatible with existing operating systems (as has been done, for example, in the KVM and KSOS efforts). A data-base conversion may be necessary (e.g., to include sensitivity-level information), but this is usually a one-time effort.

Computer security is needed in the civilian agencies of the federal government primarily for asset and resource protection and for regulatory compliance. Many agencies are responsible for financial disbursements or collections and thus are subject to attempts to perform unauthorized transactions. Trusted systems with appropriate operational and administrative controls can protect against unauthorized actions, unless these actions are performed by malicious or untrustworthy authorized users. Here, additional controls must be designed into the application programs.

All civilian agencies of the federal government are subject to the requirements for data security and integrity of Transmittal Memorandum #1 of Office of Management and Budget Circular A-71. Personal information on individual citizens that is maintained by these agencies is also subject to the confidentiality requirements of the Privacy Act of 1974. Trusted operating systems can provide a tool for effectively meeting these requirements.

Protection needs in state and local government computer systems are similar to those in federal government systems, although they are on a smaller scale and there is considerable variation from state to state. Financial disbursements and collections account for a large part of state and local governments' computer use, but regulatory requirements for security are less stringent; indeed, many states have not enacted fair information practices laws, and some do not have laws requiring confidentiality of computerized criminal-

history or public health information. Although these state agencies may have less compelling needs for trusted systems and they may be more constrained by economic considerations, trusted operating systems can greatly enhance the controllability and auditability of state and local government computer systems, and as a consequence, they would increase public trust in government operations.

In the private sector, business information that is stored and processed in nearly all corporate computer systems is, or represents, a valuable asset that must be protected. The need for effective management control over all operations, particularly those that involve computers, is self-evident. Strong accountability requirements have been established by the Foreign Corrupt Practices Act of 1977, and requirements for ensuring confidentiality of personal employment, medical, and financial information are included in state laws. In addition, federal privacy protection requirements are pending that will affect insurance, health care, and financial industries in the private sector. Thus there is a strong rationale for protection of data and programs in private-sector computer systems. Trusted operating systems could provide that protection, as well as certain collateral benefits in the areas of safety and integrity, marketing, and public relations.

The widespread availability of effective and economical trusted operating systems is predicated on computer system vendors' perceptions of an adequate market for these systems. The government alone cannot provide enough user demand to be attractive; the market must also include the private sector. Thus, the situation is somewhat circular: A market will develop along with availability, but availability is influenced by the size of the market. The trusted system technology has been developed and is not being demonstrated by the Computer Security Initiative, so the technical risk to vendors appears relatively small. However, the perceived need to maintain compatibility between trusted systems that use new architectural and design concepts and the existing equipment and software bases causes vendors to be cautious about undertaking such development efforts.

Given the trend in new operating systems and software packages toward inclusion of stronger controllability and auditability features, it appears that development may evolve naturally toward trusted operating systems. A demonstration of a credible rationale for acquisition and implementation of trusted systems, as attempted in this report, may provide the additional increment of incentive for vendors to submit their systems for evaluation and inclusion in the Computer Security Initiative's Evaluated Products Lists.

Trusted systems can contribute effectively to the solution of the growing problems of protection of assets and resources,

compliance with laws and regulations, assurance of safety and integrity, and implementation of full management control. In addition, trusted systems may provide operational economies, marketing advantages, and public-image enhancement. They are needed in a variety of applications that constitute a market that should be of considerable interest to vendors and that should strongly encourage participation in trusted system development efforts. Their use could serve the interests of private business and industry, as well as public policy, public safety, and national welfare.

## **NON-DOD TRUSTED SYSTEM NEEDS**

R. TURN

THE RAND CORPORATION, SANTA MONICA, CA.

AUGUST 1981

### **SOURCE DOCUMENT**

R-2811-DR&E

Trusted Computer Systems: Needs and Incentives for  
Use in Government and the Private Sector

R. Turn

The Rand Corporation, Santa Monica, Ca. 90406

Prepared for The Office of The Undersecretary for  
Defense Research and Engineering

August 1981

The logo for The Rand Corporation, consisting of the word "Rand" in a stylized, bold, sans-serif font.

### **OUTLINE**

1. Trusted systems
2. Generic needs for trusted systems
3. Civilian agencies of the federal government
4. State and local governments
5. Private sector
6. Concluding remarks

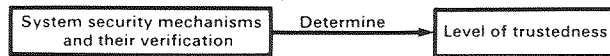
## TRUSTED COMPUTER SYSTEMS

“Systems that have sufficient hardware and software integrity to allow their use for simultaneous processing of multiple levels of classified and/or sensitive information.”

DoD Computer Security Initiative Program

Rand

## TRUSTED SYSTEM EVALUATION PROCESS

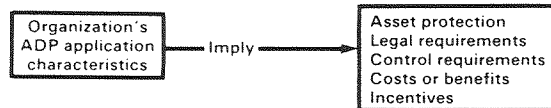


Mechanisms:  
Prevention  
Detection  
Recovery  
Operations  
Support

Assurance:  
Specification  
Design  
Implementation  
Verification  
Testing

Rand

## TRUSTED SYSTEM SELECTION



Environment considerations:  
Processor coupling  
User capability  
User data classification relationship  
Developer user trust

Rand

## GENERIC CLASSES OF NEEDS AND INCENTIVES

- Protection of assets and resources
- Compliance with regulations
- Management control
- Systems' safety and integrity
- Operational economy
- Marketing advantages
- Public image

RAND

## EFFECTIVE MANAGEMENT CONTROL

- Important goal in any organization
- Internal control and auditing in computer environment
- Trusted systems to enhance control implementation
- Tradeoffs
  - Control vs. efficiency and innovation
  - Risk of loss vs. economic pressures

RAND

## ASSURANCE OF SAFETY AND INTEGRITY

- Reliability and integrity of real-time control
  - Hardware reliability
  - Software correctness
  - Resistance to errors and tampering
- Computer-aided design and modeling

RAND



## POTENTIAL OPERATIONAL ECONOMY

- Realization depends on context and situation
- Elimination of "make-shift" security procedures
- Reduced duplication, personnel clearances, downtime losses
- Reduced insurance premiums

Rand

## OTHER CONSIDERATIONS

- Cost-effectiveness of trusted systems
- Impacts on performance
- Interoperability and compatibility
- Security policy versatility

Rand

## TRUSTED SYSTEM COST-EFFECTIVENESS

- Performance losses will be reduced
  - Use of hardware features
  - New architectures support trusted systems
  - Lessons learned are being applied
- Software conversion can be minimized
  - Compatibility will be a design goal
- Data base conversion may be required
  - Additional data fields
  - DBMS conversion may be needed

Rand

**TRUSTED SYSTEM NEEDS:  
FEDERAL CIVILIAN AGENCIES — 1**

- Protection of assets and resources
  - Massive financial disbursements or collections
  - Vulnerable to fraud
  - Trusted systems improve access control, audit trails
- Management control
- Safety and integrity
- Operational economy

**Rand**

**TRUSTED SYSTEM NEEDS:  
FEDERAL CIVILIAN AGENCIES — 2**

- Regulatory compliance
  - Transmittal Memo #1, OMB Circular A-71  
Physical, technical, administrative safeguards required
  - GSA regulations
    - FPMR 101-35.3
    - FPMR 101-36.7
    - FPR 1-4.1107-21
  - Privacy Act of 1974
  - Federal Personnel Manual, Ch. 293, 297
  - Freedom of Information Act

**Rand**

**TRUSTED SYSTEM NEEDS:  
FEDERAL CIVILIAN AGENCIES — 3**

- Other considerations
  - Funding of security requirements
  - Enforcement
  - Mission-oriented agencies
  - Cost-effectiveness of security mechanisms
  - Physical and administrative security

**Rand**

### TRUSTED SYSTEM NEEDS: STATE AND LOCAL GOVERNMENTS — 1

- Protection of assets and resources
- Regulatory compliance
  - Information confidentiality statutes
  - Fair information practices laws
  - Criminal justice systems
  - Pending federal legislation regarding social services
- Management control

Rand

### TRUSTED SYSTEM NEEDS: STATE AND LOCAL GOVERNMENTS — 2

- Safety and integrity
- Operational economy
- Other considerations
  - Cost is important
  - Consolidated systems
  - Public perceptions

Rand

### TRUSTED SYSTEM NEEDS: PRIVATE SECTOR — 1

- Rationale
  - Computers are a necessity
  - Concern with interruption and consequences
  - Trusted systems needed
  - Cost and risk tradeoffs important
- Protection of assets and resources
  - Business records, accountings of assets and receivables
  - Planning, marketing, manufacturing
  - Computer-related crime and fraud
  - Disgruntled employees
  - "Grass-roots" growth of threats

Rand

## TRUSTED SYSTEM NEEDS: PRIVATE SECTOR — 2

- Regulatory compliance
  - Fair Credit Reporting Act of 1969
  - Family Educational Rights and Privacy Act of 1974
  - Financial Privacy Act of 1980
  - Pending federal laws
    - H.R. 1059 Privacy of medical information
    - H.R. 1061 Privacy of public assistance records
    - Amendments to Fair Credit Reporting Act
  - State laws on personnel records

Rand

## TRUSTED SYSTEM NEEDS: PRIVATE SECTOR — 3

- Regulatory compliance
  - Foreign Corrupt Practices Act of 1977
    - Accurate record-keeping
    - Management control over access
    - Accountability established
  - International Data Protection
    - National laws: Austria, Canada, Denmark, France, Germany, Israel, Luxembourg, Norway, and Sweden
    - OECD guidelines
    - Council of Europe convention
- Management control
- Safety and integrity
  - Real-time systems
  - Computer-aided design and modeling

Rand

## TRUSTED SYSTEM NEEDS: PRIVATE SECTOR — 4

- Operational economies
  - Reduced personnel security costs
  - Enhanced auditability
  - Reduced security enforcement, training costs
  - Savings on insurance, bonding
- Marketing advantages
  - Security assurance to clients
  - Demonstration of concern about confidentiality and privacy
  - Reduction of victimization potential
  - Enhanced public image
- Other considerations
  - Cost-effectiveness
  - Risk tradeoffs
  - Management support

Rand

## CONCLUDING REMARKS

- Need exists for trusted computer systems
- Incentives are there for trusted system use
- Potential market is growing
- Incentives exist for vendors to produce trusted systems
- Implementation and operational questions can be resolved satisfactorily

**Rand**

DAVID L. GOLBER  
SYSTEM DEVELOPMENT CORPORATION  
THE SDC COMMUNICATIONS KERNEL

The SDC communications kernel is intended to support secure communications applications, such as secure front ends and terminal access systems. It is a minimal operating system, capability-based, and has a basic structure that we hope will ease the problem of formal specification and verification. [1]

The kernel is oriented towards support of communications systems in that it offers extensive facilities for interprocess communications. Because of its restricted aim, it does not support dynamic changes, such as creation of processes.

The SDC communications kernel has been operational for a number of years in an ARPANET-like DoD system. We feel that the capabilities and speed of the kernel are well-adapted to such a system, and are competitive with other systems not using a kernelized architecture.

The kernel was developed under the primary direction of Dr. Richard Mandell. The design and coding were done by Karl Auerbach, David Clemans, and Jay Eaglstun.

### 1.0 In General

The SDC communications kernel is a descendant of the UCLA Data-Secure Unix [2] operating system. The SDC communications kernel remains similar to the UCLA kernel in the following major areas:

- a. The SDC kernel is a minimalized operating system. It is a small amount of code which exists to provide environment and services to processes. The processes may be regarded as "application" code; there is no partitioning of the kernel itself into processes. The kernel is the only code in the machine which accesses hardware features of the machine such as memory protection registers, device registers, etc. In a PDP 11/70, the kernel

-----  
[1] The question of verification is discussed at the end of section 2.

[2] "Unix" is a trademark of Bell Laboratories.

consists of exactly that code which runs in hardware "kernel" mode, the privileged mode of the machine. Processes run in non-kernel hardware mode.

- b. The SDC communications kernel is intended to be a verifiable operating system. That is, it should be possible to formally state the services and protections that it supplies and to formally prove that it does what it is intended to do and no more.
- c. It is generally felt that operating system code which is interruptable is very hard to verify. Therefore it is preferable for a verifiable operating system to run with interrupts completely locked out. This is the policy in the case of the SDC communications kernel.
- d. The SDC communications kernel is a capability-based operating system. That is, it keeps track of processes' allowed accesses to various objects by maintaining for each process an array of data structures called capabilities, each of which describes an object and an allowed access to that object.
- e. The kernel is entered for one of two reasons:

An interrupt is received from a device. This can only occur while a process is running. Or

A kernel call (request for some kernel action) is made by some process.

In either case, the kernel code is entered via a trap or interrupt while a process is running, runs straight through without interruption and then exits. The kernel exits by causing the resumption of the execution of the code of some process (which may or may not be the process which was running when the kernel was entered).

On the other hand, the SDC communications kernel has been modified so as to be appropriate for a communications environment rather than for a general user-support environment. For this and other reasons, the SDC communications kernel differs from the UCLA kernel in a number of important ways:

- a. The SDC communications kernel does not provide for the dynamic creation or destruction of processes.

All processes exist from the time that the CPU is booted until it is halted.

- b. The SDC communications kernel does not provide for swapping of processes in and out of memory. All processes are permanently resident in memory.
- c. The UCLA system runs on a CPU (11/70, 11/45, etc) with three hardware modes: kernel, supervisor and user. The kernel runs in kernel mode, while the supervisor mode contains code called the "unix emulator" which provides an environment very like that of standard Unix to "application" code running in user mode. In distinction, "application" code written for the SDC system runs in supervisor mode and makes kernel calls directly. (User mode is unused.) SDC software thus can run in CPUs with only two hardware modes (11/34 and 11/23). (This is perhaps more a difference in usage than in the kernels themselves. The SDC communications kernel on an 11/70 or 11/45 could support some sort of emulator in supervisor mode, which could in turn provide some sort of standard environment to code in user mode.)
- d. The SDC communications kernel incorporates very extensive provisions for interprocess communications.
- e. In the UCLA system, a "Scheduler" process is responsible for choosing the next process to run. In the SDC kernel, processes are not swapped out, so scheduling is much simplified and has been made part of the kernel.
- f. In the UCLA system, a "File Manager" process is responsible for giving capabilities to processes. In the SDC system, most capabilities of processes (for instance, the capability to access a certain peripheral) are assigned statically at the time the system is configured, by a program called the "Superlinker", running under normal Unix. The Superlinker assembles the CPU memory image and gives static capabilities to processes as instructed by the "superlinker control file", which is prepared by a human being. It is this human being who is ultimately responsible for deciding what processes are allowed to communicate, etc. (Some capabilities are given to and taken away from a process dynamically as part of the interprocess communication facilities; this is discussed in more



detail below.) A separate File Manager process is not used.

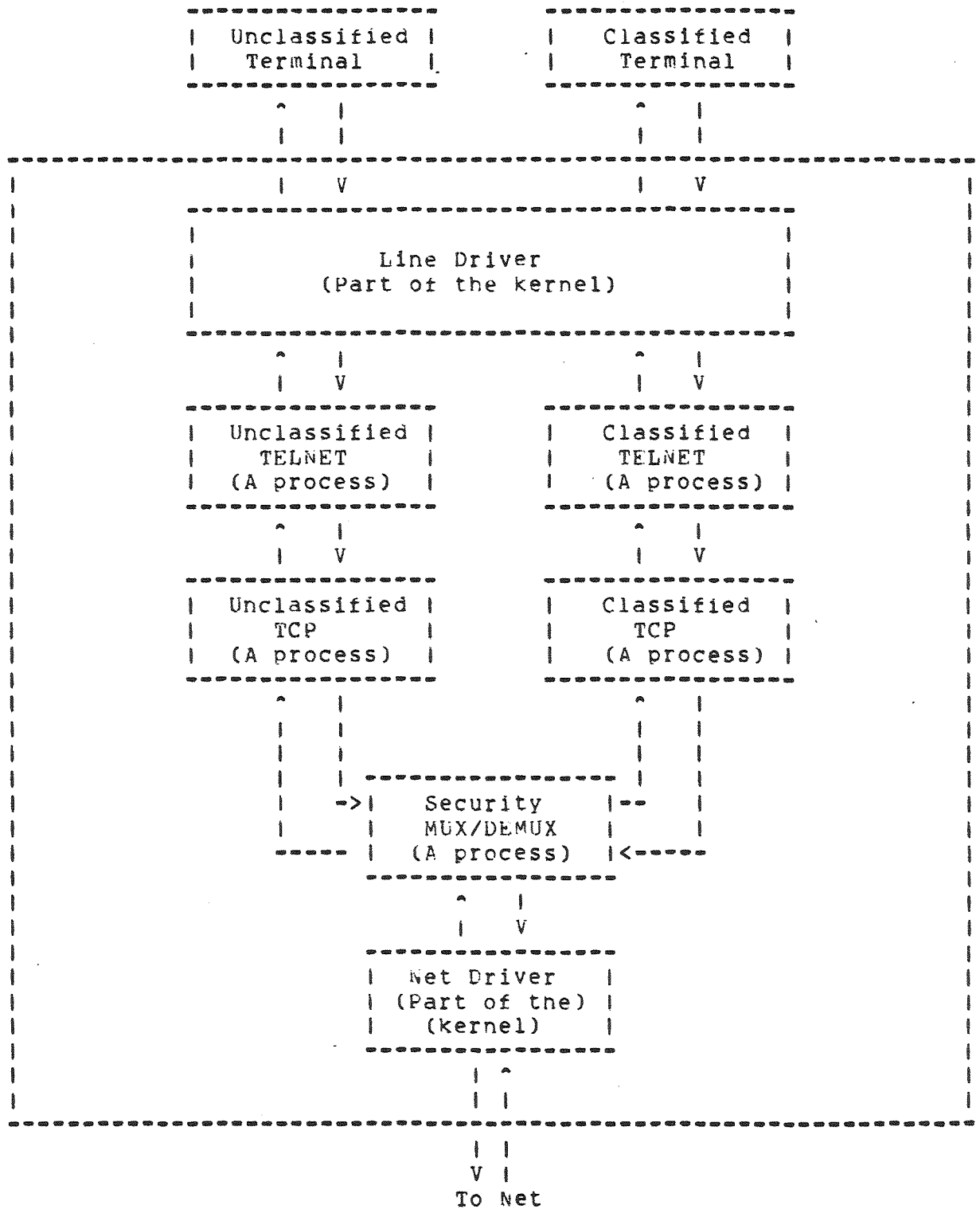
The SDC communications kernel is written in a version of Pascal, augmented to provide certain extensions necessary for the use of Pascal in an operating system. The UCLA Pascal-C translator translates this into C, which is then compiled normally. The code is written in a top-down, highly modular and methodical method, which is intended to facilitate verification, although no verification or formal specification has been done as yet.

## 2.0 Security Policy ... An Example

The SDC kernel does not itself implement a security policy. In a typical communications system using the kernel, the total security policy would be a result of the properties of various parts of the system, of which the kernel is only one part. The kernel by itself does not guarantee that the security policy is correctly implemented. The kernel is only responsible for maintaining and separating process environments, and providing and regulating interprocess communications. Thus, the properties of the kernel are related to the total security policy as a lemma is to a theorem.

An example may help to make this clear.

Consider a CPU which is to act as a sort of terminal concentrator. The CPU is to support two terminals, one of which is to carry unclassified traffic only, and the other of which is to carry classified traffic only. The TCP and TELNET protocols are to be used to provide services to each of these terminals. In order to provide separation between the classified and unclassified traffic, the TCP and TELNET processes are duplicated. The internal situation in the CPU can be pictured as followed:



In this figure, the "drivers" are collections of subroutines; they are within the kernel, since they must manipulate the physical device registers.

The Security MUX/DEMUX process is a process whose responsibility it is to separate classified and unclassified traffic streams (on reception) and to merge the streams on transmission. We do not speculate here on what basis this is done. But it is clear that this process is performing a highly security-relevant function. Therefore, the code of this process must be appropriately verified. However, it is important to point out that the verification of the functioning of this process is quite distinct from the verification of the kernel. The process is not part of the kernel.

The TELNET and TCP processes are likewise processes, not part of the kernel. Because of the scheme diagrammed above, we can hope to be able to show that the malfunctioning of any of these processes would not be able to violate security constraints. (Note that this diagram represents only one example of a system which might be build on the kernel.)

Note that the drivers handle unseparated data; therefore they too would need to be verified. However, this is true even before we make the observation that they handle unseparated data: They must be verified because they are part of the kernel, and all of the kernel must be verified.

Now we are in a position to discuss the role of the kernel itself. what are the services and protections that the kernel provides?

First of all, the kernel provides and separates the environments of the processes. For example, the kernel sets the machine mapping registers when one process runs so that the code and data of that process are accessed, and so that the code and data of some other process are not accessed.

Second of all, the kernel provides interprocess communications facilities as specified when the system is configured. In the figure above, for example, the various arrows represent interprocess communications mechanisms called "queues". (These will be discussed in more detail below.)

when the system was configured, the responsible person specifies what processes are to exist, and what communications paths between them are to exist.

The tool by which this is done is the "superlinker"

mentioned above. The responsible person prepares a "superlinker control file". For instance, for the system pictured above, the superlinker control file will specify that there are to be five processes. Each of these processes has previously been compiled, and its object code is ready and waiting. The control file specifies where this object code is to be found. Furthermore, the control file specifies exactly what queues are to exist in the system, what processes are allowed to place information on a given queue, and what processes are to be allowed to take information off of a given queue.

This superlinker control file is processed by the superlinker program, which is running under whatever development system is in use. (Not under the kernel.) The superlinker prepares the complete memory image of the CPU. In particular, it prepares the kernel tables which establish the existence of the various queues and what processes are allowed to enqueue to and dequeue from each one of them. (This will be discussed in more detail below.)

Now we can describe what it is that the kernel is trusted to do: the kernel is trusted to correctly implement and administer the system described by the superlinker control file. For example, if the superlinker control file describes the system shown in the figure above, then verification of the kernel will ensure that the unclassified TELNET process will not be able to dequeue information from the queue which is shown as leading from the classified TCP to the classified TELNET.

In order to correctly understand the nature of the security policy of the kernel as shown in the example above, it is very important to understand: The MUX/DEMUX process may be described as "trusted" in that it is trusted by the human beings who design, configure and use the system. However, it is inappropriate to describe this process as "trusted" by the kernel. The kernel does not have a notion of "trusted" process. In particular, there is no "trusted" boolean in the per-process table maintained by the kernel. The kernel knows only what communications paths each process has been authorized to use.

In the example, the queue from the net driver to the MUX/DEMUX process carries both classified and unclassified information, while the queue from the unclassified TCP to the unclassified TELNET carries only unclassified information. Thus, from a security point of view, these queues are very different. However, there is nothing in the kernel corresponding to this difference in the nature of these queues.

We can describe the philosophy here as this: the "real" security policy is executed by the person who prepares the superlinker control file. The kernel is responsible only for seeing that that person's descisions are enforced. Note that this is appropriate for the purpose for which the SDC kernel has been designed. That is, since the system is static, there is no need to burden the kernel with code, algorithms, etc, for making security-related descisions. Instead, these descisions are made beforehand, and the kernel is only responsible for enforcing them.

Note that in the example, the Line Driver software in the kernel handles both terminals. There is no reason to provide two copies of this software; both copies would have to reside in the kernel, so as to access the hardware device registers, and would have be verified to function properly, as is true for any part of the kernel and the kernel as a whole. There would be no hardware separation between the two copies.

Note that as part of its functioning, the driver must take data from the queue from the unclassified TELNET process, and place it on the line to the unclassified terminal. Similarly for the classified terminal and for the other direction of flow. It must be verified that this function is performed correctly; but this is covered by the requirement that all the kernel functioning must be verified to perform correctly.

If the kernel were to be verified, what is it that would be verified? what would be the formal properties that would have to be verified to hold?

The kernel is responsible for

- a. Maintaining and separating process environments.
- b. Providing and regulating interprocess communications.
- c. Operating devices.

Verification of the kernel would require formally stating the nature of these responsiblites. (These statements would probably include formal statements of the effects of the various kernel calls.) Then it would be necessary to formally prove that the kernel code does properly carry out these responsiblites.

As already emphasized, verification of the kernel would be

only part of what would have to be done to verify that a given system satisfies some security policy. Various non-kernel parts of the system, as well as various aspects of the total system architecture, would also have to be verified.

Certain parts of the support software used to produce the system would also have to be appropriately verified. Clearly, an important part of this software is the superlinker. The output of the superlinker is source code versions of the kernel tables, which are then compiled, linked, and built into a total memory image. These kernel tables could be human-inspected, but this would be a very difficult task, which itself would use many machine aids. If there were any chance of having to do this tedious human inspection repeatedly, verification of the superlinker would be the proper thing to do instead.

The kernel was developed in a context which emphasized the production of working code in a relatively short time. For this reason, it was decided neither to formally specify the properties of the kernel, nor to attempt to formally demonstrate anything about it. Some such effort may be made in the future.

It is of course the case that code which was not developed from formal specifications may be quite difficult to formally verify after the fact, and will almost certainly have to be modified in order to be verified. This may be true because actual security flaws are found by the formal analysis, or because some aspects of the existing code are particularly unamenable to verification. However, there are some aspects of the existing kernel - the capability orientation in particular - which we hope will ease formal verification.

### 3.0 The Environment of a Process

To begin with, we emphasize that a "process" is not part of the kernel, but rather an "application" program for which the kernel provides environment and services. No part of the kernel is described as a process.

In a PDP 11/34, the virtual address space of a process comprises 64K bytes - each process produces 16-bit addresses

as it accesses memory. These virtual addresses are translated to physical addresses by the memory management hardware. This hardware manages the process' virtual memory space in eight pieces, each of which contains 8K bytes. These pieces are the "pages" of a process' virtual address space.

These pages are used as follows:

- a. One page accesses the "library". This is a collection of commonly useful subroutines. A typical routine would be a routine for converting between a machine clock, which might read in seconds past January 1, 1970, to human time (date and time). The library is read-only to all processes.
- b. One or more pages are used to access the process' text ... that is, its executable code. This access is read-only.
- c. One or more pages are used to access the process' data area ... that is, the area in which initialized variables are kept. This area is normally read-only, but may be made writeable, by special instructions to the superlinker.
- d. One or more pages are used to access the process' so-called "bss" area ... that is, the area in which variables which are initially zero are kept. This area is read-write.
- e. The last page (page seven) accesses the process' "communications block". This is an area of memory snared by the process and the kernel and used for communications between a process and the kernel.
- f. The remaining pages (there are at most three) are free to be used to "map in" blocks of data passed from process to process using the interprocess communications mechanisms described below. These are referred to as mappable pages.

In an 11/70, the situation is similiar, except that an 11/70 has "separate I and D space", and so has twice as many pages for each process as the 11/34.

When an event occurs which affects a process, the kernel posts a notification of the event in the process' communications block, which the process looks at in the course of its main loop, which is described below. (Section

4 discusses traps and interruptions in more detail.)

In the SDC system, programmers write code which makes kernel calls directly. There is no "emulator" to provide the running process with an environment like that of some familiar operating system. (This is in distinction to the UCLA Data Secure UNIX system.) Since the programmer is writing code to run in an environment which is unfamiliar to him, we have taken the approach of providing a standard top-level structure for every process. (This also makes understanding a process written by another programmer considerably easier.)

This standard top-level structure is implemented by providing each programmer with the same "main" routine. (Again: we emphasize that this "main" is part of the process, not part of the kernel.) The entire code of a process consists of subprocedures called from this highest level procedure "main". (In particular, there are no "interrupt handler" or "completion" routines which are initiated directly by the kernel.) The outline of main is as follows:

```

procedure main;
begin
  initialize;
  while (true) do
  begin
    Set "summary" flag in communications block to false.
    while (some external event remains unprocessed) do
    begin
      Call procedure to process that external event.
    end;
    K_SLEEP;
  end;
end;

```

The procedure "main" is caused to begin executing when the system is booted. Main never exits.

The process begins by calling an initialization subroutine, and then enters an infinite loop. This loop basically does nothing except process external events. ("External" here means external to the process.) The process detects that there are external events to be processed by examining its communications block. When there are no external events to be processed, the process makes the system call K\_SLEEP to give up the CPU until some external event occurs. When an external event does occur, the kernel awakens the process, which resumes execution just as though the K\_SLEEP call had returned immediately.



From the ordinary programmer's point of view, writing a process to run under the SDC kernel consists in coding various procedures which are called from main, the procedures which they in turn call, etc.

The "summary" flag in the communications block is used in conjunction with the K\_SLEEP call to avoid a possible race condition.

(If the summary flag were not used, the following might be possible:

A process has processed all previously pending external events, has decided that there is nothing more to do, but has not yet made the K\_SLEEP call. Now an external event occurs. The kernel posts a notification of the event in the process' communication block. However, the process has already decided to go to sleep. The process now makes the K\_SLEEP call. As far as the kernel can see, the process has disposed of the new event. Thus the process goes to sleep without handling the event, and might even sleep forever.)

The summary flag avoids this race condition as follows: Any time that the kernel posts an external event to a process, it sets the summary flag in the process' communications block to "true". If the summary flag is true when the K\_SLEEP call is made, then the process is not put to sleep; the K\_SLEEP call returns immediately. It is easy to see that this mechanism, and its usage as in "main" above, avoids the race condition.

#### 4.0 Interrupts and Traps

The kernel operates with all interrupts locked out (PDP-11 priority 7). Thus, if a device wishes to interrupt while the kernel is executing, the interrupt will remain pending until the kernel exits and a process starts to execute. Then that process will be immediately interrupted.

Suppose that an interrupt occurs while a process is executing. The CPU will be interrupted and the kernel will handle the interrupt. When the process resumes executing, it will resume at exactly the place at which it was when the interrupt took place. In this sense, the interrupt is transparent to the process.

If the interrupt implies that some process should be

notified of a certain external event, then the kernel posts a notification in the communications block of that process. The process is awakened if it was previously asleep. If the notified process happens also to be the process that was running when the interrupt took place, then the process finds out about the event when it returns to its "main" routine and examines its communications block.

Thus a process runs without interrupts visible to that process. The only possible race conditions that might affect a process are concerned with the reception of notifications of external events. These problems are handled by the summary flag and the provision of a standard "main". Thus, a programmer can produce code for a process without considerations of race conditions, critical areas, etc. This is clearly of great benefit in a security-oriented system which is also production-oriented.

The only trap used in the system is the so-called "EMT" trap, which is used by a process to make a kernel call. The occurrence of a trap while in kernel mode would indicate a bug in the kernel code. In this case the kernel halts the machine. A trap other than the EMT trap while a process is running indicates a bug in the code of the process. The kernel handles this by causing the process to be re-entered and restarted at a low virtual process address.

### 5.0 The Capability List

The kernel maintains for each process a "capability list". This is an array of records, called "capability slots". An index into this array is called a "capability index". A capability slot, if not empty, contains a "capability". A capability names some "object" and describes an allowed "access" to that object. Some examples:

- a. A (statically defined) section of a disk is an object. Reading and writing are the two important accesses.
- b. The central clock maintained by the kernel is an object. The only access which may be given by a capability to the clock is the ability to set the clock. (Any process is allowed to read the clock without having an explicit capability to do so.)
- c. A block of memory is an object. Reading and writing are the two important accesses.

The capability list for each process is maintained by the kernel. Some capabilities are placed in the list by the superlinker at the time that the CPU memory image is prepared, while other capabilities are placed in or removed from the list in response to kernel calls. The process gets no access to its capability list, either read or write.

A capability serves not only to define what accesses a process has to a given object; it serves to actually identify that object. For example, suppose that one process communicates with another process via a "queue", as discussed further below. When enqueueing information to the other process, the process names the queue by giving the capability index to the capability which gives the process' access to its end of the queue.

As another example, suppose that a certain process is to be allowed to set the system clock. The superlinker control file will contain lines instructing the superlinker to set into the process' capability list a capability to set the clock. The superlinker control file, in the part describing the capabilities which the process is to have, will contain a line such as

```
clock capability on 12
```

This specifies that the process is to have a capability to set the system clock, located at index 12 in its capability list. When the process makes the `K_SET_TIME` system call, one of the parameters will be the number 12. In fact, the call is

```
K_SET_TIME(12, new_time)
```

When this call is made, the kernel will check slot number 12 of the process' capability list to see if it contains a capability to set the clock. Since it does, the kernel will do what the call asks it to do, namely to set the clock. Note that the kernel does not search the capability list of the process for a capability allowing the process to do what it has asked to do.

If the process by mistake made the call `K_SET_TIME(13, new_time)`, the kernel will look in slot number 13 of the process' capability list. Since this slot does not contain a capability to set the clock, the call will fail. That is, the kernel will give the process a return indicating that the call failed because of a "bad capability" - that is, the capability at the indicated capability index was not what was required. Also, the clock will not be set.

Notice that although the process cannot either read or write its capability list, since that list is maintained entirely by the kernel, the process must know what is in each capability slot. Capabilities are placed in the capability list of a process either statically by the superlinker, like the capability to set the clock, or else as a result of kernel calls made by the process, as in the case of getting a data block as described below. Thus the process can keep track of the entries of its capability list without in fact being able to read it.

### 6.0 Interprocess Communications

This section describes the major method of interprocess communications under the SDC kernel, namely the enqueueing and dequeueing of blocks. (There are other methods of interprocess communications which are not described here.)

The kernel maintains a pool of free memory blocks. These are blocks of 128 bytes of memory (in our current implementations). The blocks are clear as kept in the free pool. When one process wishes to send a message to another process, the sequence of events is as follows:

- a. The first process gets a block, and writes information in it.
- b. The first process places the block on a queue to the second process.
- c. The second process takes the block off the queue and reads the information from it.
- d. The second process returns the block to the kernel, which clears it and puts it back in the free pool.

In more detail, the steps are as follows:

The first process makes a `K_GET_DATA_BLOCK` kernel call. An argument to this is a capability index. This must be the index to a currently empty capability slot. The kernel will remove a block from the free pool and place a read-write capability to the block in the specified slot.

The process then makes a `K_MAP` call. This specifies the capability index where the capability to the block is located, and one of the process' virtual pages, which must be unused. The kernel in response sets the memory management hardware to make the block appear at the

beginning of that page of the process' virtual address space, giving the process read and write access to the block. This is called "mapping the block in".

The process can now read and write the block, using references to a data structure which is forced to reside at the appropriate location in the process' virtual address space.

Now, this sequence of operations is a natural pair: when a process gets a data block, it will almost certainly want to "map the block in" to access it. Thus, these two calls can be combined for greater efficiency. This is in fact what has been done. That is, the `K_GET_DATA` block call has additional parameters which will allow the calling process to map the block in as part of the call.

The process then makes a `K_ENQUEUE` call. The parameters here are the capability index naming the block, and the capability index naming the enqueue end of the queue. (The queue is defined, and the capability to the queue is given, by the superlinker.) In response, the kernel removes the capability to the block from the first process' capability list, puts the block on the queue (which is maintained entirely by the kernel), and unmaps the block, so that the process no longer has access to it. It posts a notification to the second process that the queue has something on it, and wakes the second process if it is asleep.

The second process makes a `K_DEQUEUE` call. The parameters here are a capability index to the dequeue end of the queue, and a capability index to an unused slot in its capability list. The kernel removes the block from the queue, and puts a capability to that block in the specified slot. The normal sequence of events is that a receiving process will first dequeue a block and then map it in, similar to the situation in the case of the `K_GET_DATA_BLOCK` call. Therefore the `K_DEQUEUE` call has optional parameters by which the calling process asks the kernel to map the dequeued block in to a specified virtual page.

The second process can now read the data in the block.

The second process finally makes a `K_RELEASE_DATA_BLOCK` kernel call, specifying the capability index at which the capability referring to the block is located. The kernel removes the capability, unmaps the block from the process' virtual memory space, clears the block and returns it to the free pool.

The above description is one of the simplest of the

interprocess communications mechanisms provided by the SDC kernel. One of the more interesting variations is the ability of the kernel to regulate write-access by a process to the contents of a block on a basis of a finer granularity than the whole block itself.

This facility might be useful if there were a process that should be allowed to modify certain fields in a block, but not other fields. It might be the case that some process receives a block from another via a queue, and should be allowed to modify a "header" field within the block, but no other part of the block.

This can be achieved in the SDC kernel as follows: Special instructions are placed in the superlinker control file. These instructions include a specification (namely, a bit-mask) of which bytes of the blocks dequeued from a certain queue the process in question is to be able to alter. The superlinker then configures the kernel's tables in a special way. Now when the process dequeues a block from the queue in question, the process gets a read-only capability to the block. When the process uses the `K_MAP` call to "map the block in", the kernel sets the hardware mapping registers so that the process gets only read-access to the block. The process sets the fields it is permitted to set by making a `K_WRITE_BLOCK` call. The parameters to this call are the capability index to the block, along with (the address of) a buffer of 128 bytes in the process' data space. The kernel will then copy from that buffer to the block those bytes which are indicated by the bit-mask supplied to the superlinker by the superlinker control file.

This kind of fine-granularity control must be implemented by the kernel software, since the 11/70 memory management hardware does not have the necessary capabilities.

## 7.0 Time

The kernel maintains a 48-bit "fast" clock which is incremented every 10 microseconds, using the DEC KW11-P clock device. This can be read by a process, using the `K_GET_TIME` kernel call.

The kernel also maintains for each process a "slow" clock. This is a counter in the process' communications block which is incremented every half-second. By setting variables in its communications block, a process can arrange for the kernel to give it (the process) an "alarm" notification after a specified number of half-second ticks.

By using the slow clock and the associated alarm mechanism, a process can implement any sort of facilities for maintaining multiple named timers, as it chooses. Note that using the slow clock and the alarm mechanism do not require system calls. The associated system overhead is thus quite low.

The kernel allocates time among processes by time slicing at one-tenth second intervals.

### 8.0 Implementations and Results

The SDC kernel has so far been implemented on the PDP 11/70, 11/34, 11/23 and 11/03. (The 11/23 and 11/03 implementations are modifications: the 11/23 version allows interrupts, while the 11/03 version is more properly viewed as a kernel emulator.)

The code is written in a modified version of Pascal, as used in the UCLA kernel, with small amounts of assembly language.

The 11/70 version of the code comprises approximately 2500 Pascal statements, including drivers for the DH11, DL11, RX01, RP05, TE16, and other devices. This becomes approximately 30000 bytes of instructions. (Total kernel size, including all tables, is extremely dependent on the system being configured: the number of processes, the sizes of their capability lists, the number of queues, etc.)

The times required for some kernel calls is shown below.

	11/70	11/34	
	-----	-----	
K_GET_TIME (Read fast clock.)	0.81	1.8	milliseconds
K_GET_DATA_BLOCK (Get block and map it in.)	1.5	2.7	milliseconds
K_ENQUEUE (Put block on queue.)	1.7	3.1	milliseconds
K_DEQUEUE (Get block off queue and map it in.)	1.9	3.5	milliseconds
K_RELEASE_DATA_BLOCK (Clear block and return to pool.)	2.0	4.4	milliseconds

The only one of these calls which has an equivalent in the Unix system is the K\_GET\_TIME call. The Unix "time" system call takes .31 milliseconds on the 11/70. [3]

We can use these numbers to get estimates of the bandwidth of the enqueue/dequeue interprocess communication path under several assumptions.

First of all, consider the following situation:



Here, we are supposing that the blocks are prepared by A, processed by B, and consumed and released by C. The total kernel call overhead associated with B receiving and transmitting one 128-byte block is

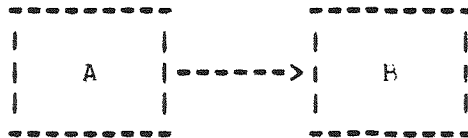
-----  
[3] All times discussed below will be for the 11/70, unless specified otherwise.



time for K_DEQUEUE	1.9 ms
time for K_ENQUEUE	1.7 ms
	-----
total	3.6 ms

This corresponds to a throughput of about 35K bytes per second.

A second situation is the following:



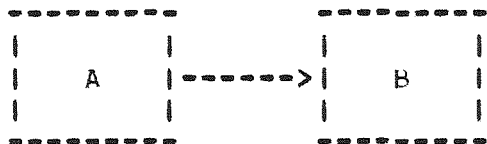
Here, we suppose that A gets a block, prepares a message, and enqueues the block to B. B dequeues the message, reads it, and then releases the block. The total kernel call time per 128-byte block here is

time for K_GET_DATA_BLOCK	1.5 ms
time for K_ENQUEUE	1.7 ms
time for K_DEQUEUE	1.9 ms
time for K_RELEASE_DATA_BLOCK	2.0 ms
	-----
total	7.1 ms

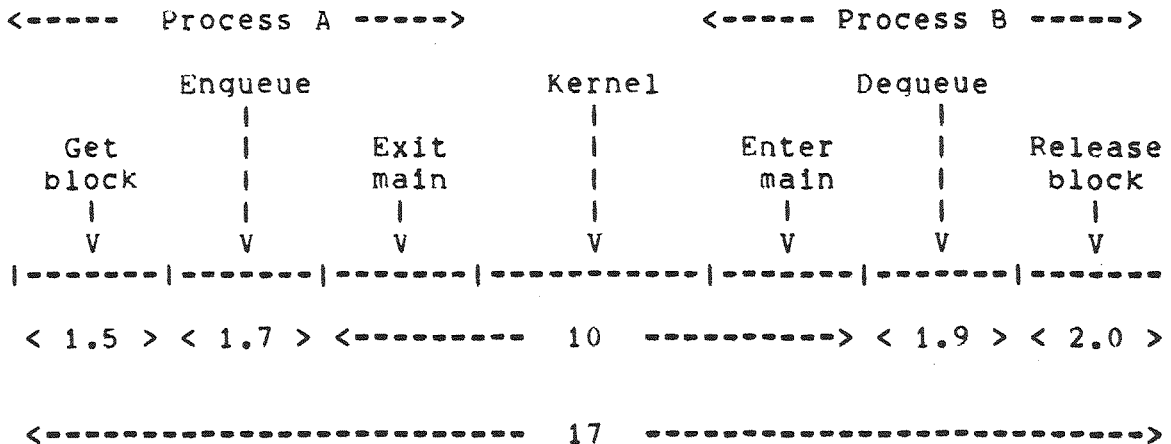
This corresponds to a throughput of about 18K bytes per second.

This calculation does not allow for the time necessary to switch processes so as to allow A and B to both run, and thus may seem unduly optimistic. However, it is actually quite realistic. When a system is heavily loaded, as is the case of interest for throughput calculations, process A would typically have a number of external events to process when it wakes up. A will then process all of these events, producing a number of blocks which it enqueues to B, before it (A) goes to sleep, letting B run. B then will process all these input blocks at one scheduling. Thus the time required to switch from A to B is divided among this number of blocks, and so does not greatly affect the throughput.

The time required to switch processes is, however, of some interest. The experiment



was re-run in such a way as to force B to be scheduled every time A sent one block. The following figure shows the times used to send one block. Note that both processes used the standard "main"; the test did not separate out the time in main from the time actually in the kernel.



This shows a worst-case time of 17 milliseconds for a 128-byte block. This corresponds to a throughput of 7.5K bytes per second.

It should be remembered that these throughputs are based on the use of 128-byte blocks, as in our current implementations. The use of larger blocks would be a minor change, and would result in proportionally larger bandwidths, since kernel call times are independent of the size of the block. [4] For example, if 256-byte blocks were used, the throughputs above would nearly double, giving values of 70K, 36K, and 15K bytes per second.

In considering these speeds and throughputs, it should also be pointed out that the SDC kernel, although it has been in use for some time, has not been extensively worked over to increase its speed. Effort in this area would undoubtedly pay off.

-----  
 [4] with the exception of the K\_RELEASE\_DATA\_BLOCK call.

In comparison, the throughput of a Unix pipe, on an otherwise-idle 11/70, is about 25K bytes per second. This is the rate when a sending process sends in units of 128 bytes. Increasing the send unit to 1280 bytes leaves the throughput rate approximately unchanged.

### 9.0 Denial of Service

The SDC kernel does not attempt to deal with denial-of-service threats. That is, a malicious process could cause CPU usefulness to be so degraded that the CPU could perform no useful work. For example, a process could (potentially but improbably) get and keep a large number of blocks. (This threat is somewhat limited: a process cannot get more blocks than it has slots in its capability list. This is a per-process parameter in the superlinker control file.)

Facilities could be added to the SDC kernel to address some denial-of-service issues, but it should be pointed out that it is consistent with the objects of the SDC kernel not to worry about denial-of-service issues. The reason is that there are no "optional" processes in a communications processor of the kind that the kernel was constructed to support. That is, the correct functioning of each process is necessary for the system to provide correct service. If any process is not performing its tasks correctly, service will be denied, and the kernel cannot do anything about it. However, security is preserved regardless of service denials.

### 10.0 Current Status

The SDC kernel was coded several years ago. It is currently operational for the Department of Defense on a number of CPUs functioning as special communications controllers and network front ends for ARPANET-like packet network terminal and host interfaces. Our experience so far shows that the resulting system provides throughput which is competitive with other systems not using a kernelized architecture.

11.0 References

Kampe, M., et al. The UCLA Data Secure Operating System. Tech. Rep., UCLA, July 1977.

Popek, G., and Farber, D. A Model for Verification of Data Security in Operating Systems. Comm. ACM, 21 9 (Sept 1978) 737-749. (Contains other pertinent references.)

Walton, E. The UCLA Kernel. Master's Th., Comptr. Sci. Dept., UCLA, 1975.

## THE SDC COMMUNICATIONS KERNEL

- AIMED AT SUPPORTING SECURE COMMUNICATIONS APPLICATIONS:
  - FRONT-ENDS
  - TERMINAL ACCESS SYSTEMS
  - ETC.
- TYPICAL PROCESSES RUNNING SUPPORTED BY THE KERNEL WOULD BE COMMUNICATIONS PROTOCOLS, ETC:
  - TCP
  - IP
- SUBJECTS OF THIS TALK:
  - SECURITY POLICY
  - SOME KERNEL MECHANISMS
  - CURRENT STATUS AND RESULTS

System Development Corporation

## FEATURES

### GENERAL "KERNEL" FEATURES:

- MINIMAL OPERATING SYSTEM, PROVIDES
  - PROCESS ENVIRONMENT
  - SCHEDULING
  - HARDWARE OPERATION
  - DEVICE DRIVERS
  - INTERRUPT HANDLING
- NON-INTERRUPTABLE
- INTENDED TO BE VERIFIABLE
- CAPABILITY-BASED

System Development Corporation

## FEATURES (Cont'd)

### SPECIALIZED FOR COMMUNICATIONS ENVIRONMENT

- NO DYNAMIC CREATION OF PROCESSES
- NO SWAPPING
- EXTENSIVE INTERPROCESS COMMUNICATIONS FACILITIES
- SECURITY POLICY

HUMAN DECIDES "NEED TO KNOW" AMONG PROCESSES WHEN LOAD IMAGE IS BUILT

ALLOWABLE INTERPROCESS PATHS SPECIFIED BY HUMAN WHEN LOAD IMAGE IS BUILT

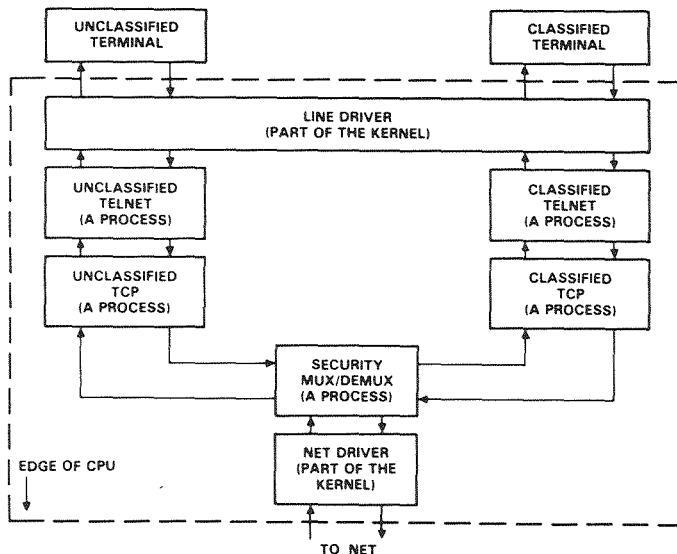
KERNEL SUPPLIES AND REGULATES INTERPROCESS COMMUNICATIONS PATHS AS SPECIFIED

KERNEL DOES NOT DISTINGUISH BETWEEN "TRUSTED" AND "UNTRUSTED" PROCESSES, ALTHOUGH HUMAN MAY

KERNEL DOES NOT KNOW ANYTHING ABOUT SECURITY POLICIES SUCH AS THE "STAR PROPERTY"

System Development Corporation

## A TYPICAL APPLICATION



System Development Corporation

## HOW THE HUMAN SPECIFIES COMMUNICATION PATHS

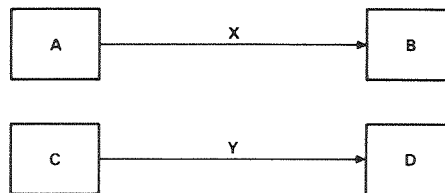
(Placing into the kernel his decisions about need-to-communicate  
among the processes)

### • THE SUPERLINKER CONTROL FILE

SAMPLE SITUATION:

PROCESSES A, B, C, D

"QUEUES" X, Y



System Development Corporation

## HOW THE HUMAN SPECIFIES COMMUNICATION PATHS

### THE SUPERLINKER CONTROL FILE

```
system DEMO
cpu is an 11/70 with ... bytes of memory
...

process A
wants
  enqueue-access to queue X ...
...
code
  /.../a.obj

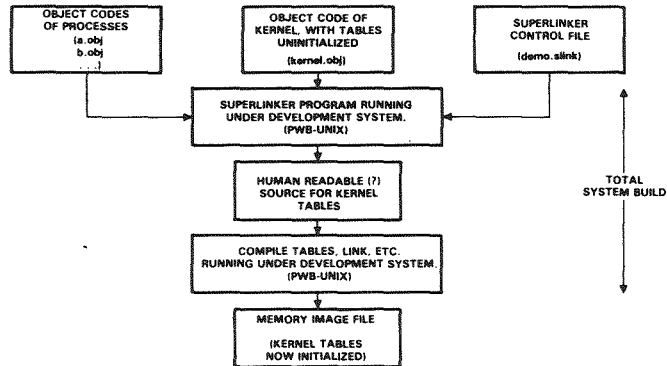
process B
wants
  dequeue-access to queue X ...
...
code
  /.../b.obj

process C
wants
  enqueue-access to queue Y ...
...
code
  /.../c.obj

process D
wants
  dequeue-access to queue Y ...
...
code
  /.../d.obj
```

System Development Corporation

## SUPERLINKER FUNCTION



- INITIALIZED TABLES TELL THE KERNEL
  - WHAT PROCESSES EXIST.
  - WHAT QUEUES EXIST.
  - WHAT PROCESSES HAVE ACCESS TO WHAT QUEUES.
  - ETC., ETC.
- SUPERLINKER AND OTHER SYSTEM-BUILD UTILITIES TRUSTED IN SAME SENSE AS COMPLIER.

System Development Corporation

## THE BOTTOM LINE

### HUMANS ARE RESPONSIBLE FOR:

- DESIGNING SYSTEM
- DECIDING WHICH PROCESSES MUST BE TRUSTED (i.e. BY THE HUMAN, NOT THE KERNEL)
- APPROPRIATELY VERIFYING (OR?) THESE PROCESSES
- WRITING THE SUPERLINKER CONTROL FILE

### KERNEL IS RESPONSIBLE FOR

ALLOWING PRECISELY THE INTERPROCESS COMMUNICATIONS SPECIFIED BY THE SUPERLINKER CONTROL FILE

- THE KERNEL DOES NOT ITSELF IMPLEMENT ANY PARTICULAR SECURITY POLICY ("STAR PROPERTY" OR . . .).
- THE KERNEL HAS NO NOTION OF "TRUSTED" OR "NON-TRUSTED" PROCESSES. (CERTAIN PROCESSES MAY BE TRUSTED BY THE HUMAN.)
- THE KERNEL DOES NOT PROTECT AGAINST "DENIAL-OF-SERVICE" THREATS.

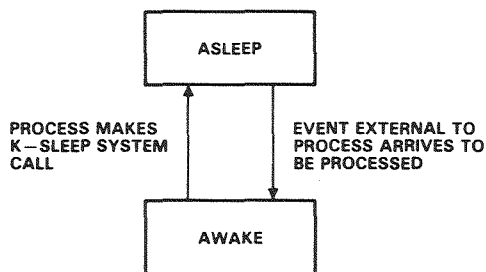
System Development Corporation



## PROCESS STATES

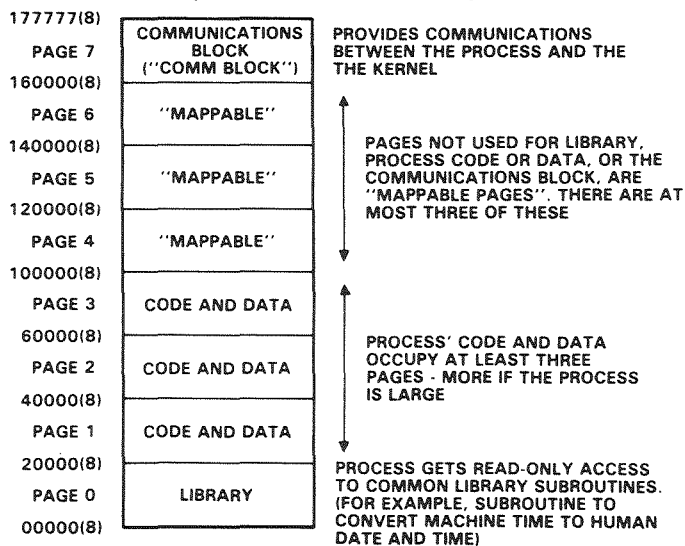
**ASLEEP:** THE PROCESS HAS NOTHING TO DO AND GETS NO CPU TIME UNTIL SOME EVENT EXTERNAL TO THE PROCESS CAUSES THE KERNEL TO AWAKEN THE PROCESS

**AWAKE:** THE PROCESS WILL GET CPU TIME. THE AVAILABLE CPU TIME IS ALLOCATED AMONG ALL THE AWAKE PROCESSES IN A ROUND-ROBIN FASHION, IN 1/10-TH SECOND SLICES



System Development Corporation

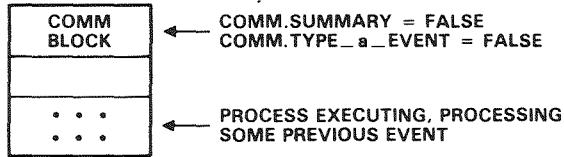
## PROCESS VIRTUAL ADDRESS SPACE (IN A PDP 11/34)



System Development Corporation

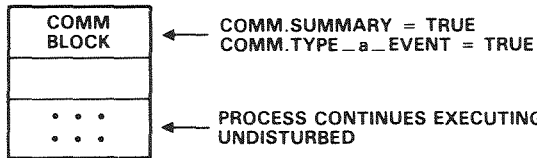
# KERNEL-TO-PROCESS COMMUNICATIONS

(1) BEFORE EXTERNAL EVENT

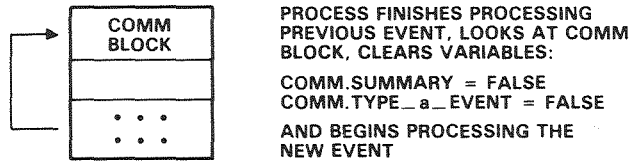


(2) EVENT OF TYPE a, EXTERNAL TO THE PROCESS, OCCURS

KERNEL SETS VARIABLES. IF PROCESS WERE ASLEEP, KERNEL WOULD AWAKEN IT.

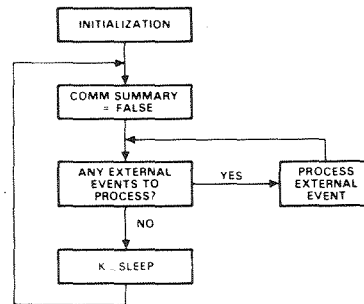


(3) PROCESS LOOKS AT ITS COMM BLOCK



System Development Corporation

# PROCESS TOP-LEVEL LOOP



```

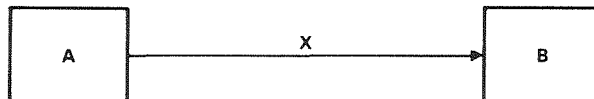
PROCEDURE MAIN:
BEGIN
  INITIALIZE:
  WHILE (TRUE) DO
  BEGIN
    COMM SUMMARY := FALSE;
    WHILE (THERE ARE UNPROCESSED EXTERNAL EVENTS) DO
    BEGIN
      IF (COMM.TYPE_a_EVENT) PROCESS_a;
      IF (COMM.TYPE_b_EVENT) PROCESS_b;
    END;
    K_SLEEP;
  END;
END;

```

IF COMM SUMMARY IS TRUE WHEN THE K\_SLEEP CALL IS MADE, THE PROCESS DOES NOT ACTUALLY SLEEP; THE K\_SLEEP CALL RETURNS IMMEDIATELY

System Development Corporation

## INTERPROCESS COMMUNICATIONS BY QUEUES



STEPS:

1. PROCESS A MAKES K-GET-DATA-BLOCK KERNEL CALL TO GET A BLOCK FROM THE CPU-WIDE FREE POOL.  
PROCESS A WRITES MESSAGE IN THE BLOCK.
2. PROCESS A MAKES K-ENQUEUE KERNEL CALL TO PLACE THE BLOCK ON THE QUEUE.  
PROCESS A LOOSES ALL ACCESS TO THE BLOCK.  
THE KERNEL MAINTAINS THE QUEUE.
3. PROCESS B MAKES K-DEQUEUE KERNEL CALL TO GET THE BLOCK FROM THE QUEUE.  
PROCESS B READS THE MESSAGE.
4. PROCESS B MAKES K-RELEASE-DATA-BLOCK KERNEL CALL.  
PROCESS B LOOSES ALL ACCESS TO THE BLOCK. THE KERNEL  
CLEARS THE BLOCK AND RETURNS IT TO THE FREE QUEUE.

System Development Corporation

## ACCESSING A BLOCK

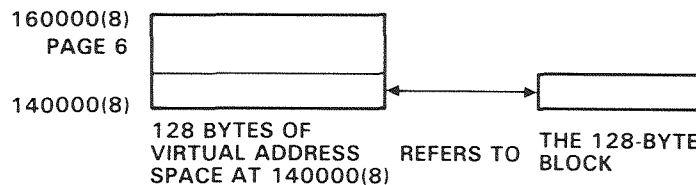
SUPPOSE: PROCESS HAS GOTTEN A BLOCK  
CAPABILITY REFERRING TO BLOCK SPECIFIED BY SOME INDEX,  
SAY 17

PROCESS: K-MAP(17,140000(8));

"17" SPECIFIES BLOCK

"140000(8)" SPECIFIES ONE OF THE PAGES OF THE  
PROCESS' VIRTUAL ADDRESS SPACE

KERNEL: IN RESPONSE TO THE CALL, SETS HARDWARE MAPPING REGISTERS  
OF THE PROCESS SO THAT



NOTE: SOMEWHAT DEPENDENT ON PDP-11 MEMORY-MANAGEMENT SCHEME

BUT: HAS BEEN EMULATED ON 11/03!

System Development Corporation

## PROCESS PROGRAMMER'S SETTING

STANDARD "MAIN" PROVIDED TO EVERY PROGRAMMER

PROGRAMMER WRITES THE PROGRAM UNITS CALLED FROM MAIN:

INITIALIZE	(INITIALIZE PROCESS)
PROCESS—a	(PROCESS EXTERNAL EVENTS OF TYPE a)
PROCESS—b	(PROCESS EXTERNAL EVENTS OF TYPE b)

PROGRAMMER'S CODE IS NEVER (LOGICALLY) INTERRUPTED. THE PROGRAMMER DOES NOT HAVE TO DEAL WITH RACE CONDITIONS, ETC

THE PROGRAMMER'S CODE MAKES KERNEL CALLS DIRECTLY. THERE IS NO ATTEMPT TO PROVIDE A FAMILIAR PROCESS ENVIRONMENT, SUCH AS A "UNIX EMULATOR"

THE PROGRAMMER MUST HAVE SOME UNDERSTANDING OF MEMORY MANAGEMENT IN ORDER TO USE THE INTERPROCESS COMMUNICATIONS MECHANISMS

*System Development Corporation*

## STATUS

WRITTEN IN MODIFIED PASCAL.  
NOT FORMALLY SPECIFIED  
INTENDED TO BE "VERIFIABLE"

VERSIONS FOR

PDP 11/70  
PDP 11/34  
PDP 11/23 (INTERRUPTABLE)  
PDP 11/03 (KERNEL "EMULATOR")

SIZE

11/70 VERSION, INCLUDES DRIVERS FOR  
DH, DL (SERIAL LINE INTERFACES)  
RP (DISK)  
RX (FLOPPIES)  
TE (TAPE)  
AND MORE

~2500 PASCAL STATEMENTS  
~30,000 BYTES OF CODE

DATA SPACE SIZE EXTREMELY DEPENDENT ON THE NATURE OF THE SYSTEM:  
NUMBER OF PROCESSES, QUEUES, ETC.

IN USE SUPPORTING FRONT-ENDS, ETC, FOR A SPECIAL DoD ARPANET-LIKE SYSTEM

*System Development Corporation*

## INDIVIDUAL CALL TIMINGS

ALL TIMES IN MILLISECONDS

	11/34	11/70	11/70 UNIX
K-GET-TIME	1.8	0.81	0.31
K-GET-DATA-BLOCK	2.7	1.5	
K-ENQUEUE	3.1	1.7	
K-DEQUEUE	3.5	1.9	
K-RELEASE-DATA-BLOCK	4.4	2.0	

KERNEL CALL TIME TO SEND ONE 128-BYTE DATA BLOCK

	11/34	11/70	11/70 UNIX
K-GET-DATA-BLOCK	2.7	1.5	
K-ENQUEUE	3.1	1.7	
K-DEQUEUE	3.5	1.9	
K-RELEASE-DATA-BLOCK	4.4	2.0	
	<u>13.7</u>	<u>7.1</u>	

DEDUCED BANDWIDTH (BYTES/SEC)

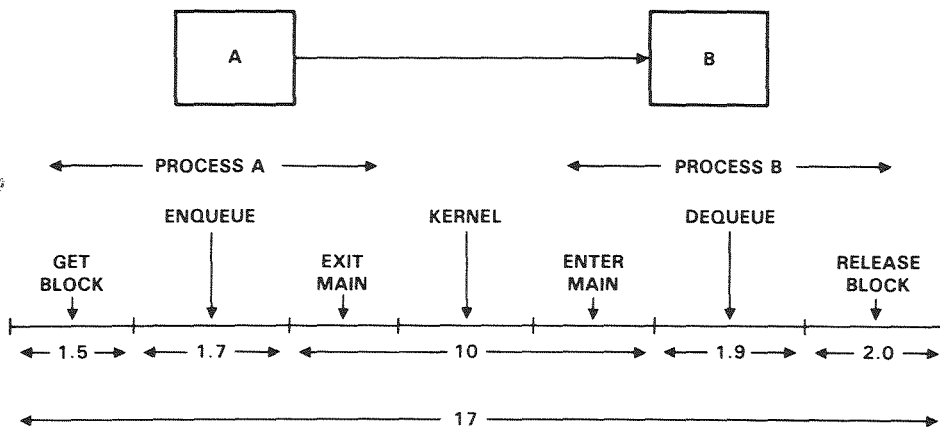
9.3K                      18K                      ~ 25K (PIPE)

KERNEL BANDWIDTH WOULD INCREASE PROPORTIONALLY IF BLOCK SIZE WERE INCREASED PAST 128

UNIX PIPE BANDWIDTH SAME IF READS AND WRITES WERE DONE IN UNITS OF 1280 BYTES INSTEAD OF 128

System Development Corporation

## SCHEDULING SPEED



(ALL TIMES FOR 11/70, IN MILLISECONDS)

System Development Corporation

## **SUMMARY**

- **OPERATIONAL FOR A NUMBER OF YEARS**
- **HOPEFULLY VERIFIABLE OPERATING SYSTEM KERNEL.**
- **REASONABLE SPEED IN CLASSIFIED DoD APPLICATIONS,  
COMPETITIVE WITH NON-KERNEL SYSTEMS.**
- **REASONABLY HOSPITABLE ENVIRONMENT FOR A  
COMMUNICATIONS SYSTEM**

*System Development Corporation*

---

**MITRE IR&D  
Project 95130  
Secure Packet Switch**

---

Chris Hisgen  
The MITRE Corporation

**Motivation**

- Survey of Commercial Architectures
- Exploration of Multiprocessor Machines and the Impact of Security Kernels on Them
- Impact of Multiprocessors on Security Kernels

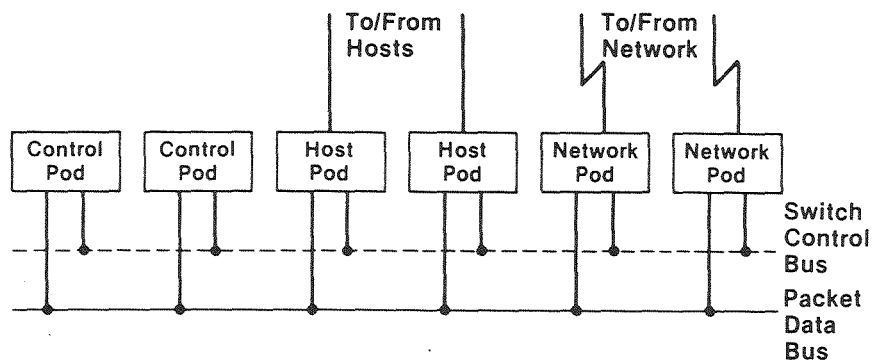
**Problem**

- Verification of Large Amounts of Software
- Performance Overhead of the Security Kernel
- Economics of Minicomputer Based Switch
- Survivability of Few Node Network

## Approach

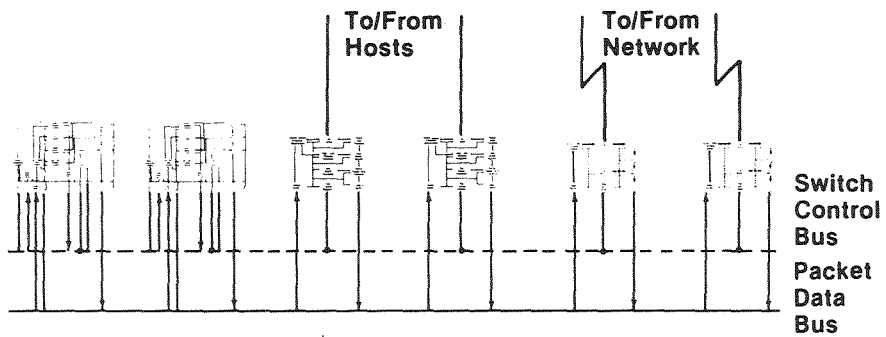
- Functional Partitioning of Packet Switching Tasks
- Assignment of One Processor per Function
- Interprocessor Communication Minimized
- Processors that Handle More Than One Packet Simultaneously Will Have Their Code Verified
- Most Processors Handle One Packet at a Time and Then Have Their Memory Scrubbed Before Handling the Next Packet
- The Functional Partitioning and Communication Limitations Enforce the Security Policy
- Modular Node with Many Microprocessors Ensures Survivability at Lower Cost

## Model Switch

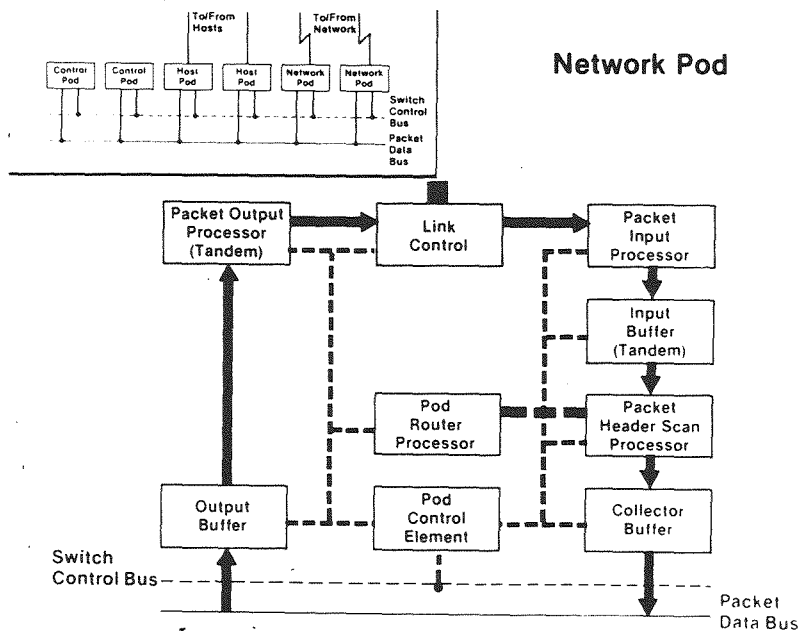


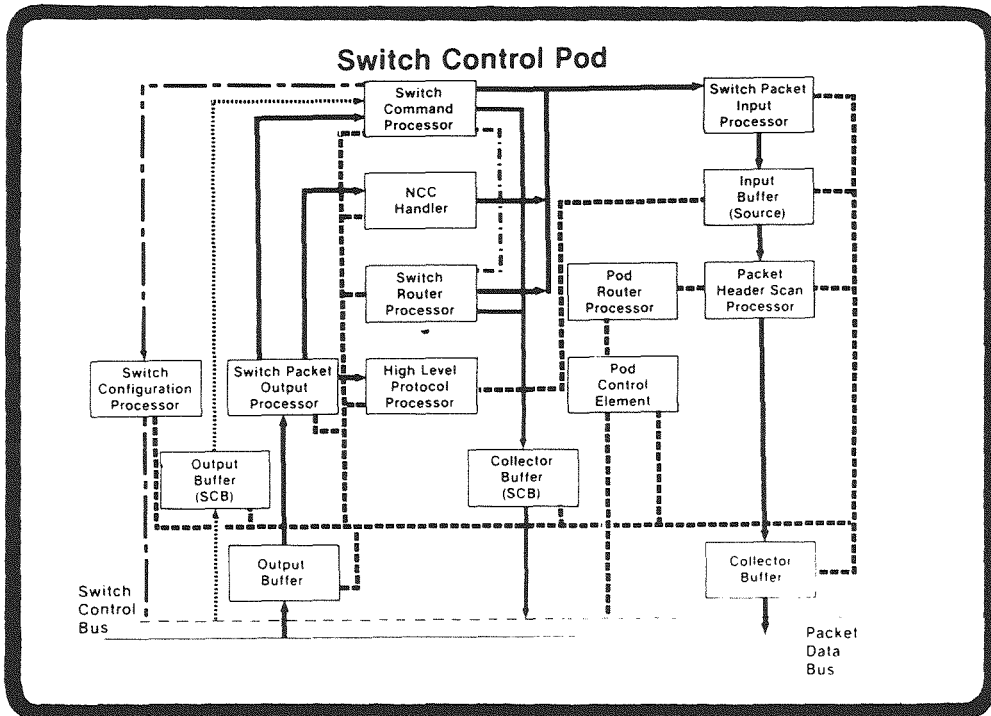
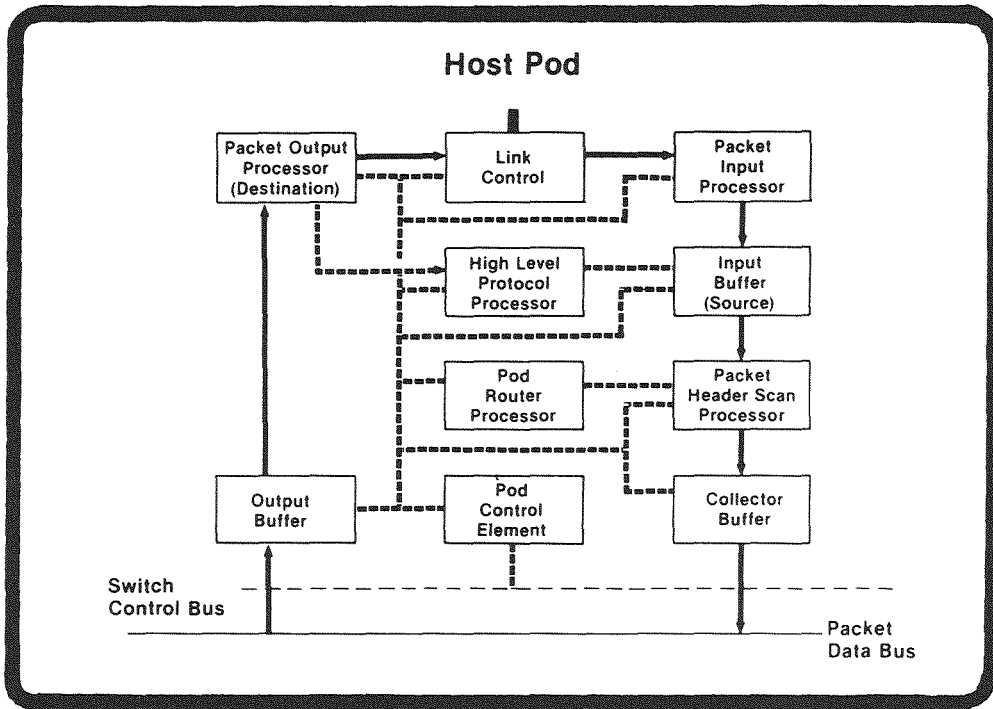


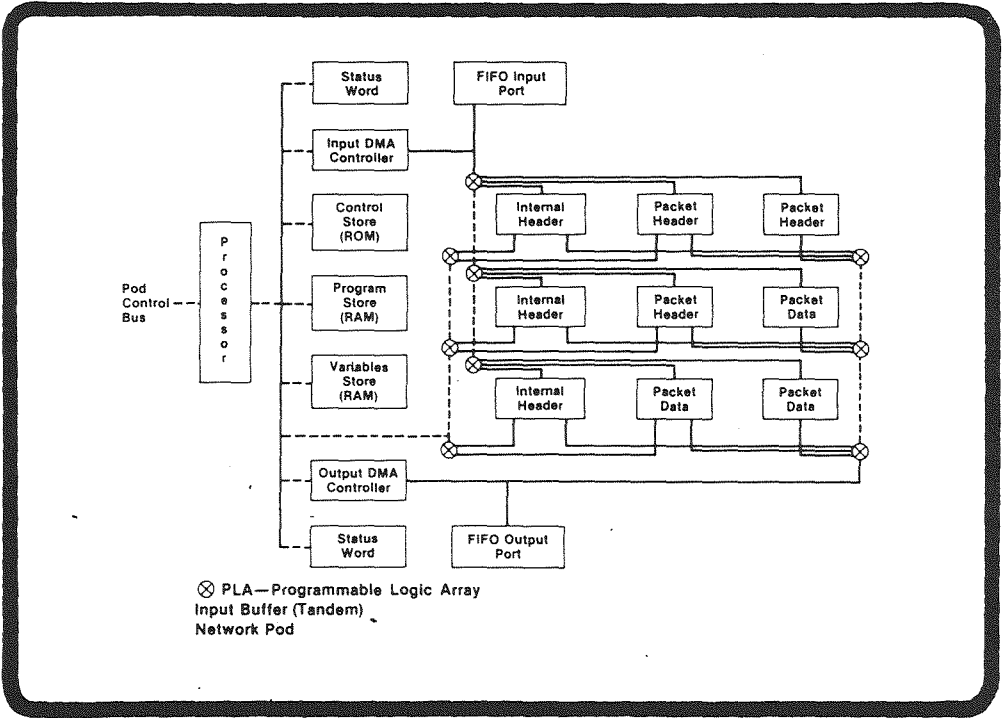
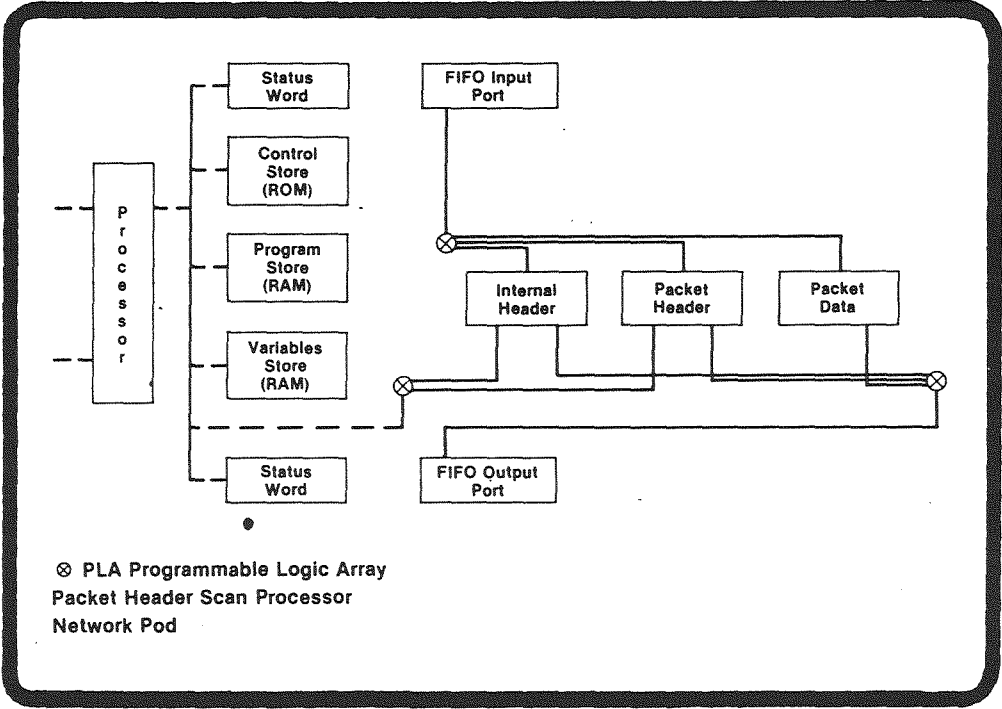
## Model Switch



## Network Pod







## Microprocessor Usage

---

- **Microprocessors Per Pod**

- Network Pod: 9
- Host Pod: 10
- Control Pod: 15

- **There Are Only Four Classes of Processor**

- Packet Processor
- Packet Buffer
- Control Elements
- Fake Hosts

## Microprocessor Usage, Continued

---

- **Microprocessors for an Arpanet Style Packet Switch Node**

- 4 Trunk Lines
- 4 Host Lines

**Translates To**

— 6 Network Pods	— 6 × 9 =	54 Microprocessors
— 6 Host Pods	— 6 × 10 =	60 Microprocessors
— 2 Control Pods	— 2 × 15 =	<u>30 Microprocessors</u>
	<b>Total</b>	<b>144 Microprocessors</b>

## Open Technical Issues

---

- Serial Bus or Parallel Bus
- 16/32 Bit Bus (Motorola VersaBus, Zilog Z-Bus, Intel Multibus)
- Performance as Function of Load for Various Access Protocols (e.g., Polling, Contention, TDMA)
- Bus Choice Must Satisfy Requirements for Control, Addressing, and Data Transfer

## Conclusions

---

- The Design is Feasible
- The Design Benefits from AUTODIN II Experience
- The Design is a Hardware Casting of the Trusted Computing Base
- The Design Has Less Software to Verify than a Comparable Switch
- Special Purpose Multi-Microprocessor Switches Have Been Built Commercially

## EXPERIENCE WITH KVM

TOM HINKE

SYSTEM DEVELOPMENT CORPORATION  
SANTA MONICA, CALIFORNIA

**System Development Corporation**

## OVERVIEW

- KVM IS A GENERAL USE SYSTEM
- KVM IS DESIGNED FOR LEVEL 4 CERTIFICATION
- KVM ARCHITECTURAL STATUS
- KVM OPERATIONAL STATUS

**System Development Corporation**

KVM IS A COMPLETE SYSTEM

- COMPATIBLE WITH POPULAR UNMODIFIED OPERATING SYSTEMS

O/S	DOS
CMS	MVS
MVT	ETC.

- SUPPORTS EXISTING APPLICATIONS

FORTRAN	JOVIAL
PL/I	ASSEMBLER
TEXT EDITORS	DATA MANAGEMENT SYSTEMS

**System Development Corporation**

KVM IS DESIGNED FOR LEVEL 4 CERTIFICATION

- KERNELIZED ARCHITECTURE
- ENFORCES DoD SECURITY POLICY
- FORMAL VERIFIED SPECIFICATIONS
- CORRESPONDENCE BETWEEN SPECIFICATIONS AND CODE

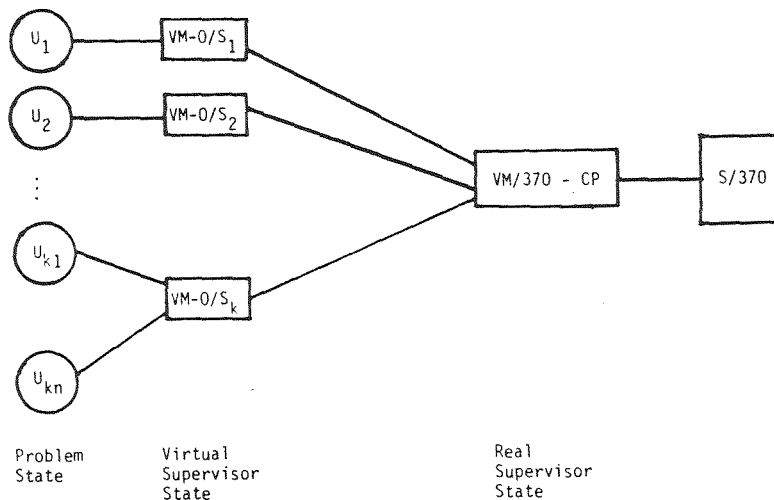
**System Development Corporation**

## KERNELIZED ARCHITECTURE

- KERNEL & TRUSTED PROCESSES
  - FORMALLY SPECIFIED AND VERIFIED
  - INTERPRET & ENFORCE SECURITY POLICY
- AUDITED GLOBAL PROCESSES
  - CONTROL SHARED SYSTEM RESOURCES
  - CONFINED AND UNPRIVILEGED
- NON KERNEL CONTROL PROGRAM
  - SUPPORTS USER VIRTUAL MACHINES
  - REENTRANT, UNPRIVILEGED, UNTRUSTED

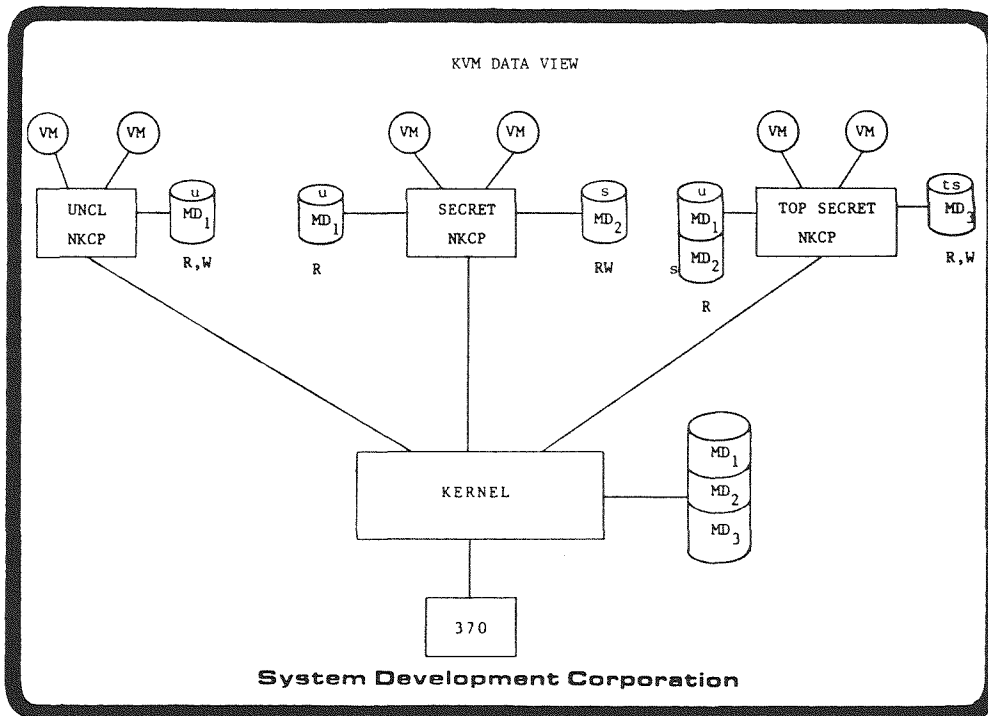
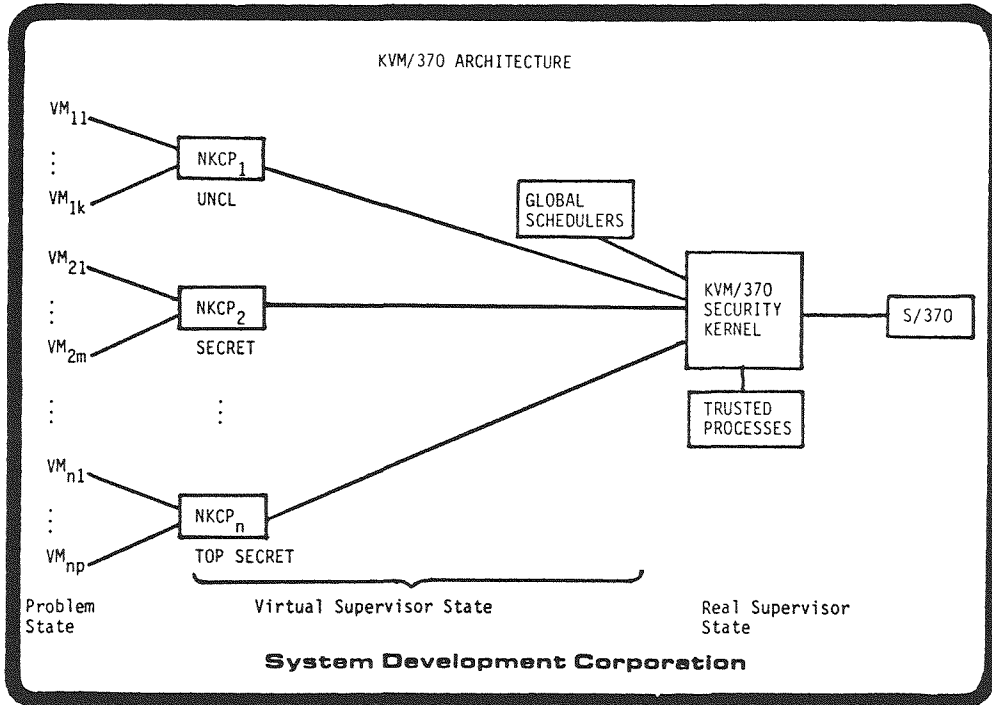
System Development Corporation

## VM/370 ARCHITECTURE



System Development Corporation





KVM ENFORCES DoD SECURITY POLICY

- MANDATORY  
4 HIERARCHICAL LEVELS + 62 COMPARTMENTS
- DISCRETIONARY  
ACCESS CONTROL LISTS + PASSWORDS
- MULTI-LEVEL ACCESS TO MINIDISKS

**System Development Corporation**

VERIFIED FORMAL SPECIFICATIONS

- SPECIFICATIONS WRITTEN IN INA JO™ FOR  
ALL TRUSTED CODE
- VERIFIED TOP LEVEL SPECIFICATIONS
  - 780 LINES OF SPECIFICATIONS
  - 494 PAGES OF PROOF EVIDENCE
- VERIFIED SECOND LEVEL SPECIFICATIONS
  - 2,910 LINES OF SPECIFICATIONS
  - PROOFS ARE IN PROGRESS

**System Development Corporation**

### SPECIFICATION-TO-CODE CORRESPONDENCE

1. MAP INA JO CONSTANTS & VARIABLES TO JOVIAL DATA .
2. MAP INA JO TRANSFORMS TO JOVIAL PROCEDURES .
3. MAP INA JO ASSERTIONS TO JOVIAL SECURITY CHECKS .
4. MAP INA JO TRANSITIONS TO JOVIAL ASSIGNMENT STATEMENTS .
5. RESOLVE DISCREPANCIES .
6. VERIFY ALL SECURITY CHECKS ARE PERFORMED BEFORE ANY ASSIGNMENTS ARE MADE .
7. AUDIT UNMAPPED SOURCE CODE FOR SECURITY-RELEVANT CODE .

**System Development Corporation**

### ARCHITECTURAL STATISTICS

VM/370 REL 3 PLC 15

134 MODULES 130,000 LINES ASSEMBLER CODE

---

FUNCTIONAL AREA	MODULES	TOTAL LINES	
COMPOOLS	47	3,139	JOVIAL
KERNELS	107	11,590	JOVIAL/ASMB
AUTHORIZATION	32	3,637	JOVIAL
*ACCOUNTING	2	204	JOVIAL
*OPERATOR	22	2,017	JOVIAL/ASMB
*UPDATER	<u>3</u>	<u>264</u>	
→ TRUSTED	213	20,821	
NKCP	116	129,754	
GLOBAL PROCESSES	<u>20</u>	<u>17,230</u>	
→ UNTRUSTED	126	146,984	

\* UNDER DEVELOPMENT

**System Development Corporation**

## KVM ARCHITECTURAL STATUS

### KVM IMPLEMENTS

- MESSAGE PROTOCOL DRIVEN SYSTEM
  - INTERNAL COMMUNICATION
- MULTI-LEVEL RELATIONAL DBMS
  - USER, DEVICE, PROFILE DIRECTORIES
- CAPABILITY BASED SYSTEM
  - ACCESS PERMITTED ONLY IF USER HAS A "GRANT"
- ABSTRACT DATA TYPE MONITORS
  - NO CENTRAL SYSTEM TABLES

**System Development Corporation**

## KVM OPERATIONAL STATUS

- UNDERGOING FORMAL DETAILED SYSTEM TESTING
  - SYSTEM DEVELOPMENT CORPORATION IBM 4331-II
  - NAVAL AIR TEST CENTER, AMDAHL V7/A
- IN PROGRESS & CONTINUING NEXT 12 MONTHS WITH NEW FEATURES

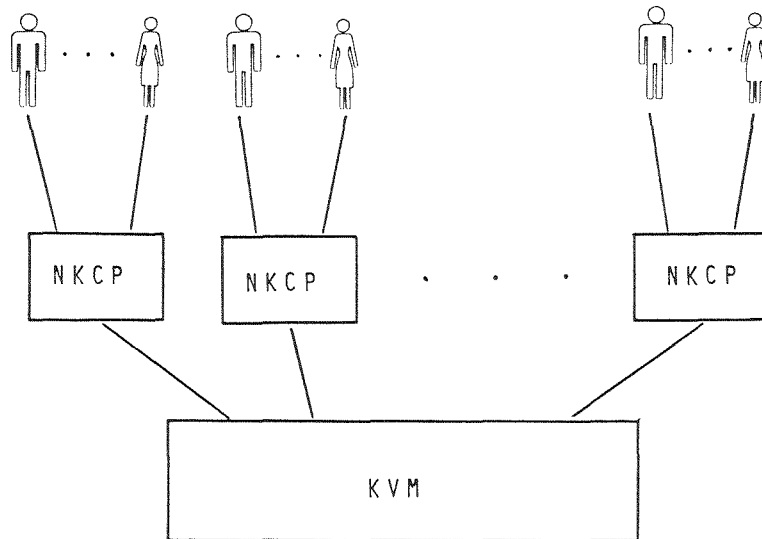
**System Development Corporation**

OPERATIONAL PERFORMANCE

- CONTRACTUAL MEASUREMENT TASK
- ESTABLISH MEANINGFUL BENCHMARKS
- CONTRAST THROUGHPUT OF VM vs KVM

System Development Corporation

PERFORMANCE EXPERIMENTS



System Development Corporation

- KVM IS A COMPLETE GENERAL USE SYSTEM
- KVM WILL BE A LEVEL-4 SYSTEM
- WE CAN LEARN ABOUT KVM WHILE USING KVM

**System Development Corporation**

**SCOMP (KSOS-6)  
DEVELOPMENT EXPERIENCE UPDATE**

LESTER FRAIM  
HONEYWELL  
FEDERAL SYSTEMS OPERATION  
AUGUST 12, 1981

81.25.1

**Honeywell**

**TOPICS**

- PROJECT OBJECTIVES
- HARDWARE DESIGN OVERVIEW
- SOFTWARE DESIGN OVERVIEW
- PERFORMANCE EXPERIENCE
- VERIFICATION EXPERIENCE

81.25

**Honeywell**

**PROJECT SUPPORT**

NAVELEX  
- KERNEL AND HARDWARE DEVELOPMENT  
- TRUSTED SOFTWARE  
- TCP  
HONEYWELL  
- KERNEL INTERFACE PACKAGE  
- 1822 IMPLEMENTATION  
- HARDWARE PRODUCT DEVELOPMENT

**Honeywell**

### **PROGRAM OBJECTIVES**

- DEVELOP ADD-ON HARDWARE TO COMMERCIAL LEVEL 6 WHICH MAKES IT EASIER TO BUILD SECURE SYSTEMS
- DEVELOP TRUSTED COMPUTER BASE (TCB) SOFTWARE
  - ENFORCES DOD SECURITY POLICY
  - FORMALLY PROVABLE (TLS ONLY)
  - SUPPORTS VARIOUS APPLICATIONS
- DOD CERTIFICATION
- DEVELOP THE SCOMP PRODUCT

H1 26 4

### **POTENTIAL APPLICATIONS**

- FREE-STANDING TIME SHARING SYSTEM
- "GUARD" BETWEEN TWO NETWORKS AT DIFFERENT SECURITY LEVELS
- SECURE NETWORK FRONT-END
- SECURE DATA BASE MACHINE
- MILITARY MESSAGE SWITCH
- SECURE WORD PROCESSOR

H1 26 5

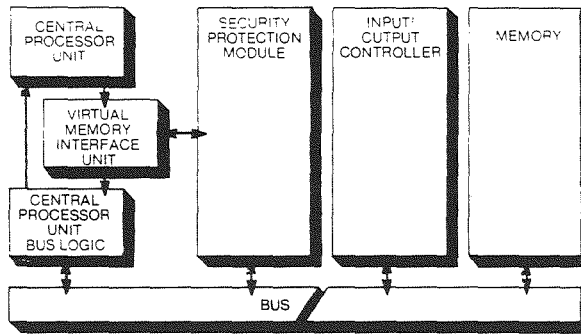
**Honeywell**

### **SCOMP HARDWARE BASE**

- LEVEL 6 MINICOMPUTER
- SECURITY PROTECTION MODULE (SPM)
- VIRTUAL MEMORY INTERFACE UNIT (VMIU)
- 1822 ACLA LINE ADAPTER
- STANDARD LEVEL 6 PERIPHERALS



### SPM + LEVEL 6 MINICOMPUTER = SCOMP



### SECURITY PROTECTION MODULE FEATURES

- FAST PROCESS SWITCHING
  - PROCESS DESCRIPTOR TREE DEFINITION VIA DESCRIPTOR BASE ROOT
  - AUTO LOAD OF DESCRIPTORS
- 1-3 LEVEL MEMORY DESCRIPTOR SYSTEM
  - R, W, E CONTROL AT ANY LEVEL
  - SEGMENTS: 2K WORDS (512)
  - PAGES: 128 WORDS
- I/O MEDIATION
  - CPU TO DEVICE
  - DEVICE TO MEMORY
- MULTICS-LIKE RING STRUCTURE
  - 2 PRIVILEGED, 2 NON-PRIVILEGED RINGS
  - READ, WRITE, EXECUTE, AND CALL BRACKETS
  - RING CROSSING SUPPORT INSTRUCTIONS
- PAGE FAULT RECOVERY SUPPORT

### KSOS-6 SOFTWARE

- SECURITY KERNEL
- TRUSTED SOFTWARE
- SCOMP KERNEL INTERFACE PACKAGE
- TRANSMISSION CONTROL PROTOCOL (TCP)

4-28-9

**Honeywell**

### SYSTEM DESIGN

- NON-FILE SYSTEM I/O OUTSIDE KERNEL
- FILES CONSTRUCTED EXTERNALLY USING SEGMENTS
- DEMAND PAGING VIRTUAL MEMORY
- NON-DISCRETIONARY ACCESS CONTROL - BELL AND LAPADULA
  - PRIVILEGE
  - ACCESS ATTRIBUTES NOT FIXED

81.25.10

**Honeywell**

### SYSTEM DESIGN (CONT)

- DISCRETIONARY ACCESS CONTROL
  - UNIX R, W, E FOR OWNER, GROUP, OTHER
  - RING BRACKETS FOR OWNER, GROUP, OTHER
  - SUBTYPES
- KERNEL INTERRUPTIBILITY
  - KERNEL OPERATIONS MAY BLOCK
  - KERNEL OPERATIONS NOT INTERRUPTED
  - NO PROCESS SWITCH
  - SEGMENT ACCESS RECHECK

81.25.11

**Honeywell**

### SYSTEM DESIGN (CONT)

- INFORMATION CHANNEL CONTROL
  - UPGRADED ARGUMENT
  - READABILITY DETERMINES RESPONSE
  - SYSTEM HIGH GARBAGE CAN SEGMENT
  - DELAY ON RESOURCE EXHAUSTION

81.25.12

**Honeywell**

**KSOS-6  
TRUSTED SOFTWARE**

- USER SERVICES
  - SECURE INITIATOR
  - SECURE SERVERS
  - ACCESS AUTHENTICATION FUNCTIONS
  - LOGIN
    - CHANGE GROUP
    - SET ACCESS LEVEL
    - CHANGE DEFAULT ACCESS LEVEL
  - LOGOUT
  - FILE DISPLAY AND ACCESS MODIFIER
  - PASSWORD MODIFIER

81 26 13

**Honeywell**

**KSOS-6  
TRUSTED SOFTWARE (CONT)**

- OPERATIONS SERVICES
  - SECURE STARTUP
  - AUDIT COLLECTION
  - SECURE LOADER
  - OPERATOR COMMANDS
    - SET SYSTEM CLOCK
    - SWITCH ACCOUNTING FILES
    - CHANGE DEVICE ACCESS
    - SET DISK DEVICE STATUS
    - SYSTEM SHUTDOWN

81 26 14

**Honeywell**

**KSOS-6  
TRUSTED SOFTWARE (CONT)**

- MAINTENANCE SERVICES
  - MAKE FILESYSTEM
  - TRUSTED DATABASE EDITORS
    - USER ACCESS
    - GROUP ACCESS
    - TERMINAL ACCESS
    - SECURITY MAP
  - MOUNTABLE FILESYSTEMS
  - FILESYSTEM DUMP
  - FILESYSTEM RESTORE
  - FILESYSTEM CONSISTENCY CHECK

81 26 15

**Honeywell**

## **SCOMP KERNEL INTERFACE PACKAGE (SKIP)**

- PURPOSE
  - PROVIDE AN EFFICIENT LOW LEVEL INTERFACE FOR USE BY APPLICATIONS SOFTWARE
  - PROVIDE A HIERARCHICAL FILESYSTEM
  - PROVIDE PROCESS CONTROL
- ATTRIBUTES
  - CODE RESIDES IN KERNEL ADDRESS SPACE WITH RING 2 EXECUTE PERMISSIONS
  - ACTS AS A FILTER FROM USER RING TO KERNEL GATES TO PROVIDE FILESYSTEM AND PROCESS CONTROL INTEGRITY

81 26 16

**Honeywell**

## **SKIP FILE SYSTEM FEATURES**

- ENTRY NAMING SYSTEM
- MONOTONICALLY INCREASING SECURITY
- INCREASE SECURITY LEVEL THROUGH UPGRADED DIRECTORY OR FILE
- MULTICS LIKE LINK SUPPORT
- FILESYSTEM INTEGRITY MAINTAINED IN RING 2
- NO PATHNAME AWARENESS
- FILE DATA MANIPULATION IN USER RING

81 26 17

**Honeywell**

## **SKIP PROCESS CONTROL FEATURES**

- PROVIDE CLASSICAL EVENT WAIT/NOTIFY SYNCHRONIZATION
- ALLOW SPAWNING OF CHILD PROCESSES
- PROVIDE MECHANISM BY WHICH USER RING CODE CAN HANDLE INTERRUPTS AND FAULTS

81 26 18

**Honeywell**

**KSOS-6  
TRANSMISSION CONTROL PROTOCOL  
(TCP)**

- BASED ON BBN-TCP-4
  - 1822 ASYNCHRONOUS LINE ADAPTER
  - WILL USE THE SKIP
- 81 26 19

**Honeywell**

**KSOS-6**

**KERNEL PERFORMANCE ENVIRONMENT**

- LEVEL 6/43
  - HARDWARE MONITOR
  - TOTAL EXECUTION TIME
  - I/O TIME
  - NUMBER OF DMA TRANSFERS
- 81 26 20

**Honeywell**

**SAMPLE KERNEL  
PERFORMANCE RESULTS**

<b>GATE</b>	<b>EXECUTION TIME (MS)</b>
READ_SYSTEM_CLOCK	1.06
GET_SYSTEM_PARAMETER	1.46
GET_PROCESS_ACCESS	2.62
GET_PROCESS_STATUS	1.46
SEND_MESSAGE	4.35
RECEIVE_MESSAGE	1.74
MAP_SEGMENT	16.20
UNMAP_SEGMENT	1.70
CREATE_PROCESS	488.29
RELEASE_PROCESS	1.36
CREATE_SEGMENT	19.79

**Honeywell**

### **SAMPLE KERNEL PERFORMANCE RESULTS**

TEST	EXECUTION TIME
MISSING SEGMENT FAULT RECOVERY	16.20
CONTEXT SWITCHING	1.85

81 26 22

**Honeywell**

### **KERNEL VERIFICATION STATUS/RESULTS**

- PROOF OF DESIGN COMPLETE
- TWO MODULES CAUSE STORAGE FAULTS IN  
FORMULA GENERATOR
  - CREATE\_\_PROCESS
  - INVOKE\_\_PROCESS
- FALSE THEOREMS CHANNEL MINIMIZED BY
  - DELAY ON RESOURCE EXHAUSTION
  - EXCEPTION REPORTING ON WRITE-UPS
  - PRIVILEGE CHECKS

81 26 23a

**Honeywell**

### **KERNEL VERIFICATION STATUS/RESULTS (CONT)**

- DIFFERENCES FROM IMPLEMENTATION
  - PRIVILEGE IS REMOVED
- TOOLS
  - ENHANCEMENT REQUIRED
  - ISOLATING REASONS FOR FALSE THEOREMS  
IS TEDIOUS

81 26 23 B

**Honeywell**

## **SUMMARY**

- **HARDWARE**
  - PROTOTYPE DEVELOPMENT COMPLETE
  - PRODUCTION DEVELOPMENT
- **SOFTWARE**
  - KERNEL
  - TRUSTED SOFTWARE
  - SKIP
- **TEST SITE DELIVERY FIRST QUARTER 1982**

11-20-84

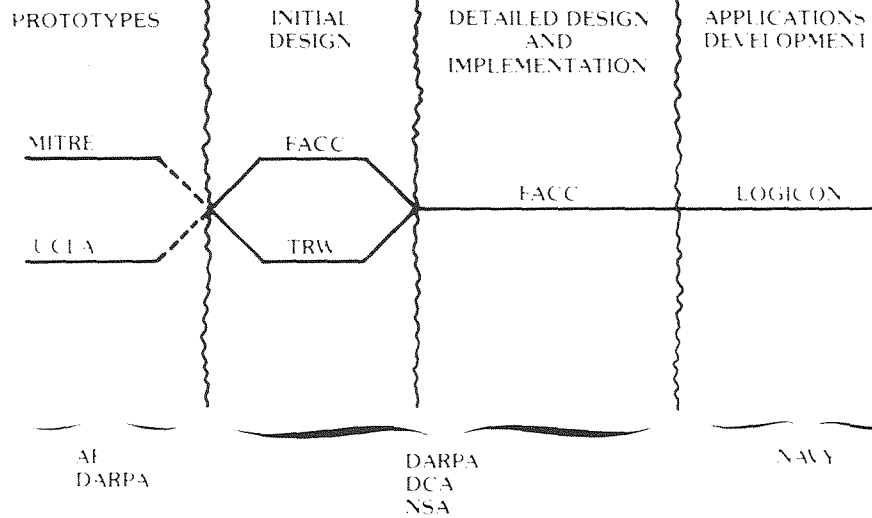
**Honeywell**

# KSOS-11

## Summary And Update

John Woodward  
The MITRE Corporation

### KSOS-11 History





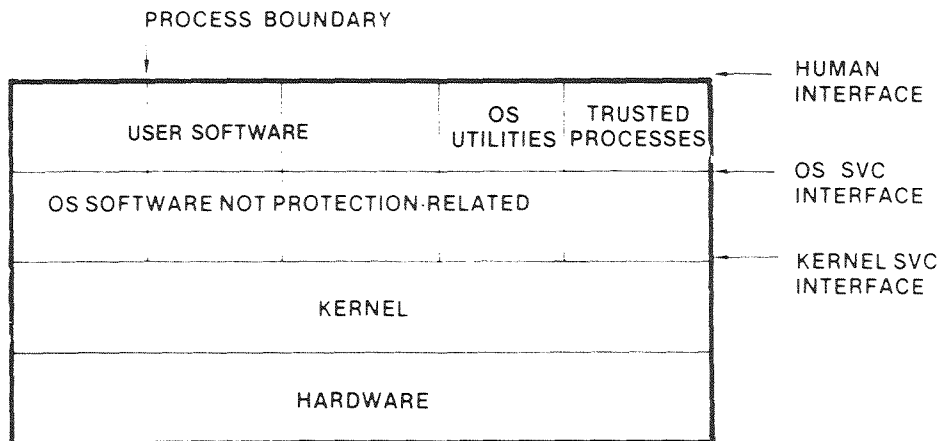
## KSOS Summary and Update — Overview

---

**Project Goals**  
**Project Status**  
**Insights Into Trusted Computing**

## Kernelized System

---



## **Project Goals — KSOS Requirements Summary**

---

**Production - Quality System**

**Provable Security**

**UNIX Compatibility**

**Efficiency Comparable With UNIX**

**Administrative Support Features**

**General-Purpose Kernel**

**Broad Applicability**

## **Project Goals — KSOS Kernel Architecture**

---

<b>Functions</b>		
<b>Processes</b>	<b>Segments</b>	<b>I/O</b>
fork invoke spawn	build release	device function
release	remap	mount/unmount
post receive message	rendezvous	create file
signal		open/close
interrupt return		link/unlink file
walk process table		read write block
nap		
boot		
halt		
get set status	get set status	get set status
get set level	get set level	get set level

## Project Goals — KSOS Kernel Architecture

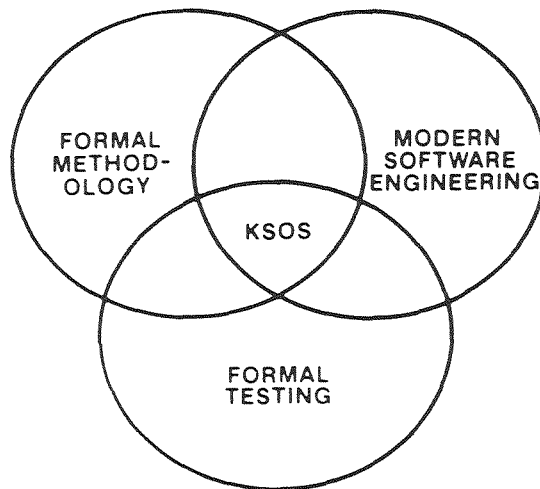
---

### Non-Kernel System-Related Software

User Services	Operations & Maintenance	Administration
secure initiator	file system dump restore	immigration
secure server	pack initialization	user control
login logout	extent initialization	privilege control
file access modifier	modify control entry	security map
change access level	consistency checkers	terminal profile
change group	boot copy	device profile
level preserving copy print	directory manager	system profile
secure mail	network controllers	audit capture
	system startup shutdown	
	system generation	
	process bootstrapper	
	mount unmount	
	assign deassign device	
	line printer spooler	
	kernel-to-pathname mapper	

## Project Goals — KSOS Security Assurance

---



## **Project Status — Versus Requirements**

---

**Production - Quality System**

**Provable Security**

**UNIX Compatibility**

**Efficiency Comparable With UNIX**

**Administrative Support Features**

**General-Purpose Kernel**

**Broad Applicability**

## **Project Status — Provable Security**

---

### **Design Proofs**

Spec Checking

Theorem Proving

Analysis of False Theorems

Flow Analysis

### **Code Proofs**

Example module only

## **Project Status — KSOS Efficiency**

---

**Performance**

**Size**

## **Project Status — KSOS Implementation Completion**

---

<b>Kernel</b>	<b>95%</b>
<b>Emulator</b>	<b>90%</b>
<b>NKSR</b>	<b>65%</b>
<b>TCP</b>	<b>?</b>

## **Insights Into Trusted Computing**

---

**Modula As the Implementation Language**

**Multiple Representations**

**Formal Methods**

**Hardware Base**

**Security Model**

## **Insights Into Trusted Computing**

---

**It Can Be Done!**

**Importance of Corporate Commitment**

**Utility and Benefits of Formal  
Specifications**

**Need for More Experience in Code Proofs**

**Need for Additional Tools and Concepts**

Mike Soleglad

Logicon

# ACCAT AND FORSCOM GUARD SYSTEMS

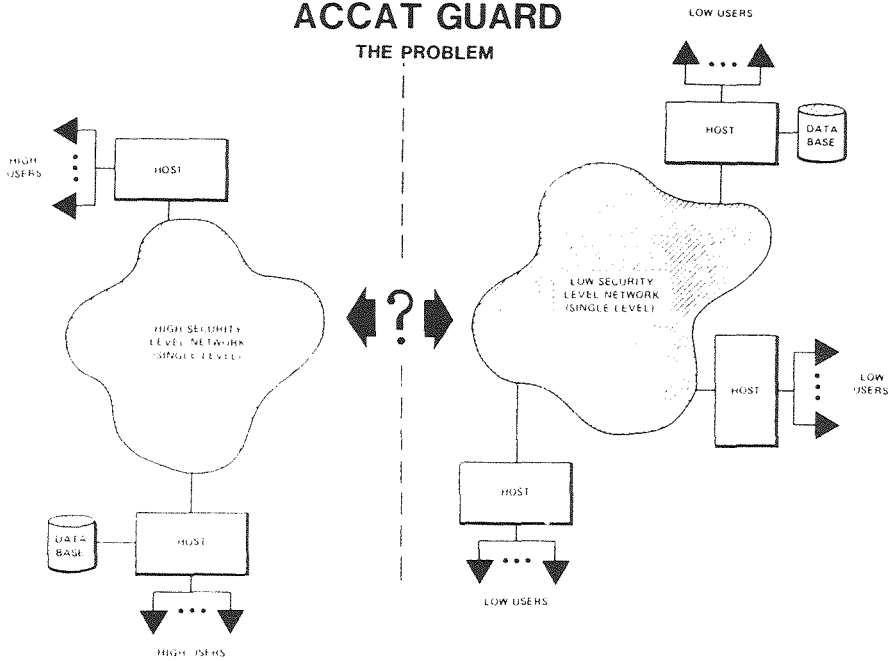
LOGICON

## ACCAT/FORSCOM GUARD PRESENTATION

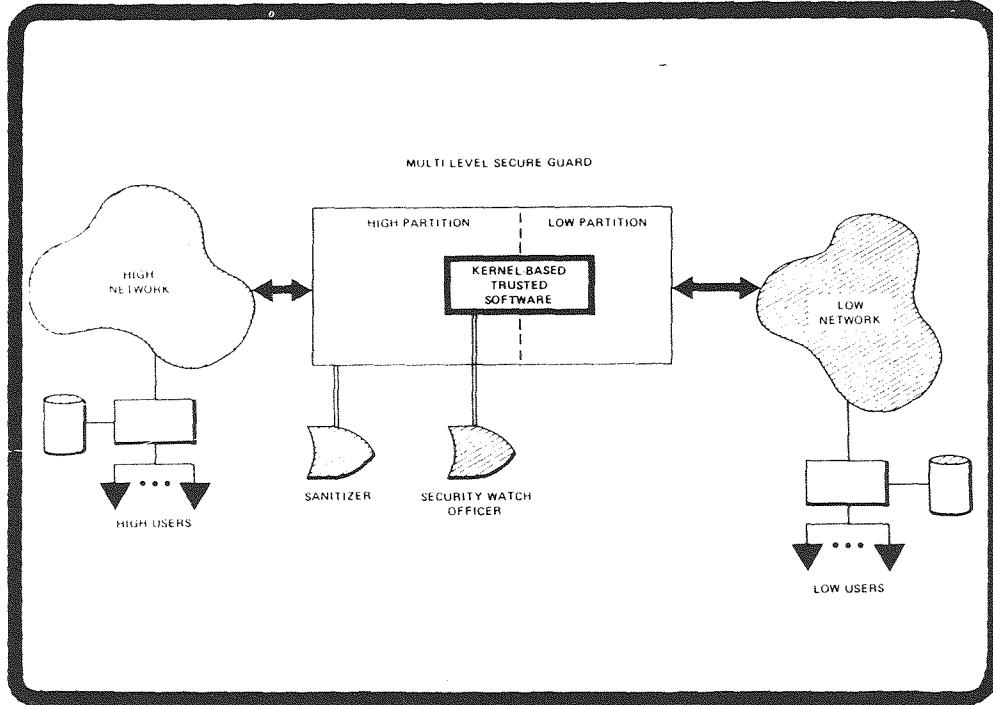
- THE PROBLEM
  - THE SOLUTION
  - HARDWARE CONFIGURATION
  - SOFTWARE MECHANISMS
  - STATUS
- } ACCAT GUARD  
AND  
FORSCOM GUARD

# ACCAT GUARD

## ACCAT GUARD THE PROBLEM







## ACCAT GUARD

### FUNCTIONAL DESCRIPTION - TRANSACTIONS

- "TRANSACTION" ORIENTED
  - ALL TRANSACTIONS ARE SUBMITTED VIA "NETWORK MAIL"
  - ALL RESULTS ARE RETURNED VIA "NETWORK MAIL"
  
- SIX TRANSACTION TYPES
  - LOW-TO-HIGH
    - MAIL
    - "CANONICAL" QUERY
    - "ENGLISH" QUERY
  
  - HIGH-TO-LOW
    - MAIL
    - "CANONICAL" QUERY
    - "ENGLISH" QUERY

# ACCAT GUARD

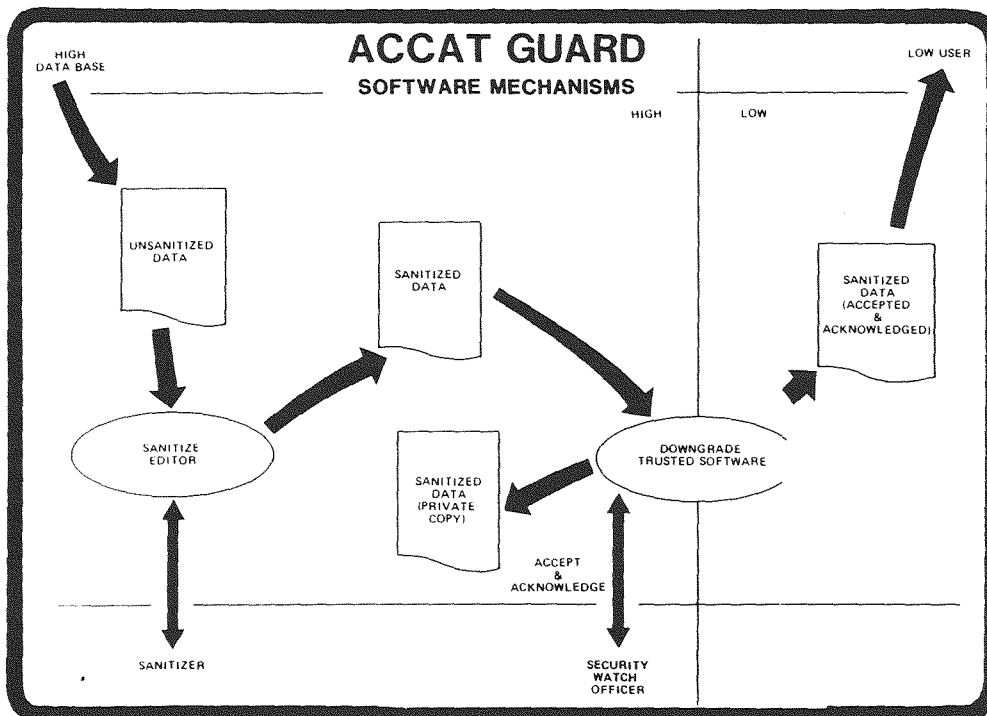
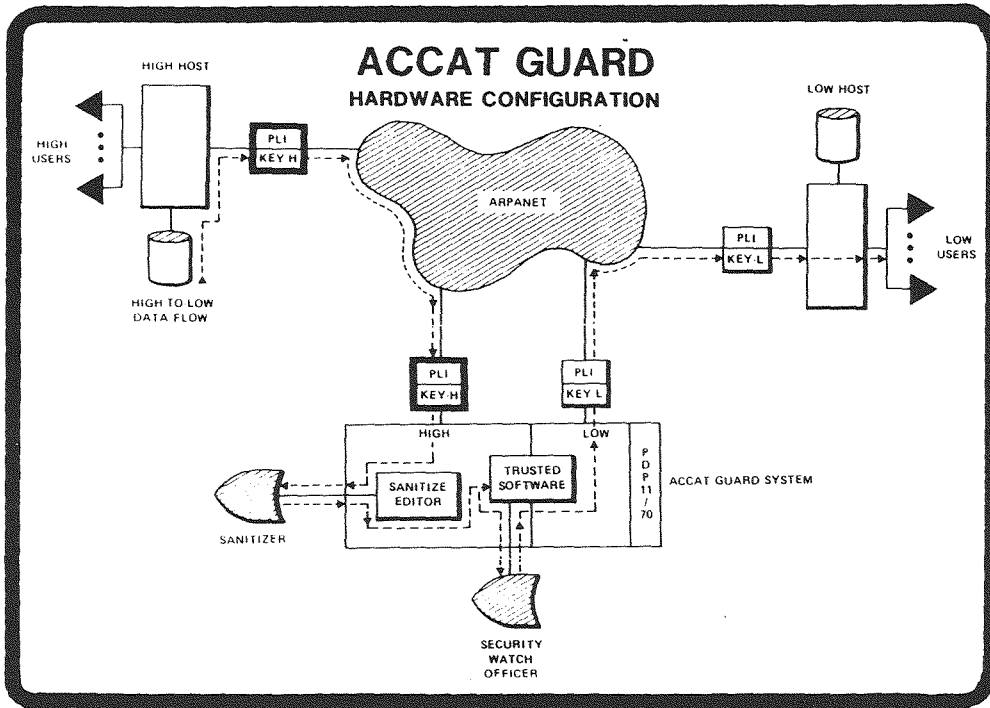
## FUNCTIONAL DESCRIPTION - PERSONNEL

- SECURITY WATCH OFFICER (SWO)
  - VIEWS ALL HIGH-TO-LOW DATA TRANSFERS
  - INTERFACES WITH "TRUSTED SOFTWARE" FOR "DOWNGRADING" OF DATA
  
- SANITIZATION PERSONNEL (SP)
  - SANITIZES LOW-TO-HIGH QUERY RESULTS
  - TRANSLATES ENGLISH QUERIES TO "CANONICAL" FORM
  - INTERFACES WITH "HIGH SIDE" UNTRUSTED SOFTWARE

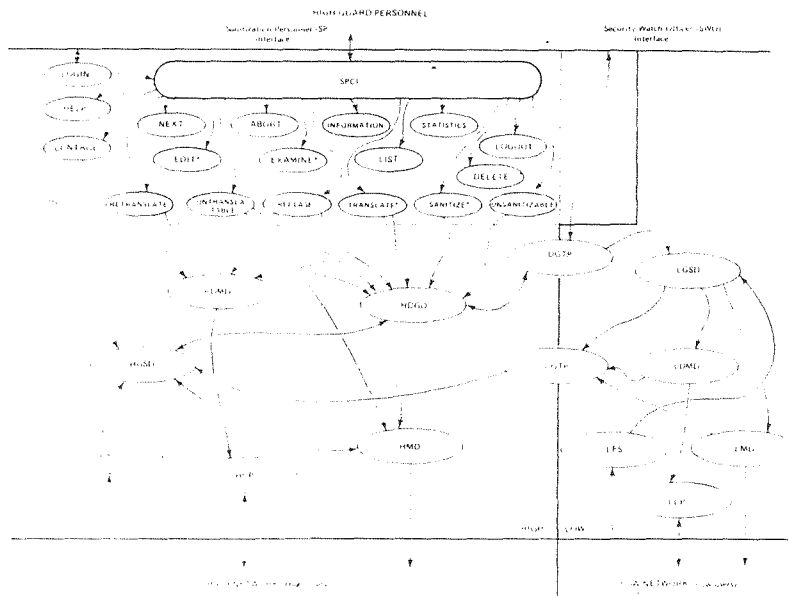
# ACCAT GUARD

## SECURITY POLICY

- DATA SEPARATION (DoD SECURITY MODEL)
    - SIMPLE SECURITY CONDITION ("READ" RULE)
    - \*-PROPERTY CONDITION ("WRITE" RULE)
    - TRANQUILITY CONDITION ("ALTER" RULE)
  - DATA INTEGRITY ("DUAL" OF DoD SECURITY MODEL)
  - DISCRETIONARY ACCESS (A LA UNIX)
  - MANUAL DOWNGRADE POLICY (VIOLATES \*-PROPERTY)
    - SECURITY WATCH OFFICER (SWO) VIEWS ALL DATA
    - SWO ACCEPTS DOWNGRADE
    - SWO CONFIRMS DECISION
  - AUDIT ALL HIGH-TO-LOW DOWNGRADES
- } KSOS ENFORCED
- } TRUSTED SOFTWARE ENFORCED



## ACCAT GUARD SOFTWARE CONFIGURATION



## ACCAT GUARD

STATUS: PRESENT AND FUTURE

### PRESENT

- HARDWARE INSTALLED AT NAVAL OCEAN SYSTEMS CENTER (NOSC)
- ALL SOFTWARE COMPLETED – DEMONSTRABLE UNDER UNIX
- TRUSTED SOFTWARE FORMALLY SPECIFIED AND VERIFIED
- THREAT/VULNERABILITY ANALYSIS COMPLETED
- KSOS 11 INSTALLATION UNDERWAY

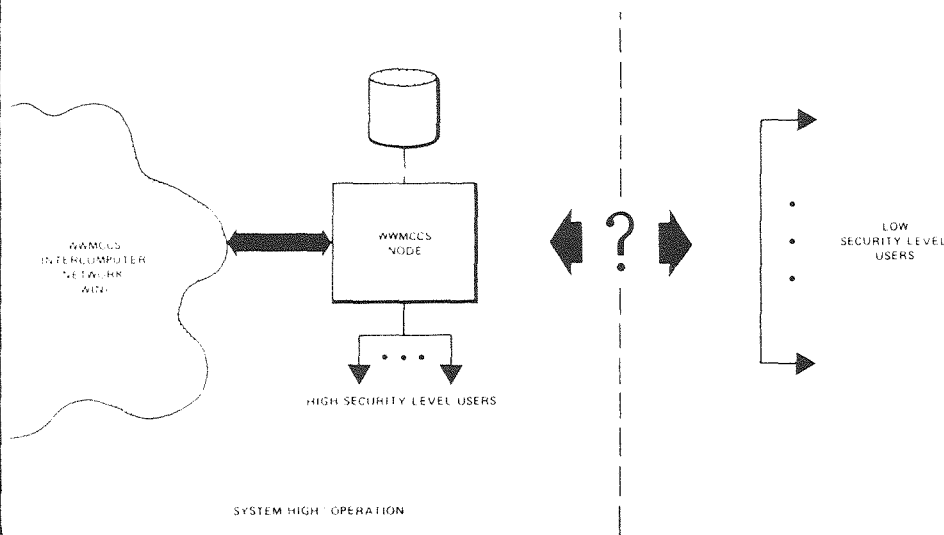
### FUTURE

- KSOS 6 INSTALLATION PLANNED
- AUTOMATED SANITIZATION/TRANSLATION – ELIMINATES SANITIZER
- VERIFICATION OF AUTO-SANITIZATION – ELIMINATES SWO
- OTHER LOW/HIGH HOST SUPPORT PLANNED

# FORSCOM GUARD

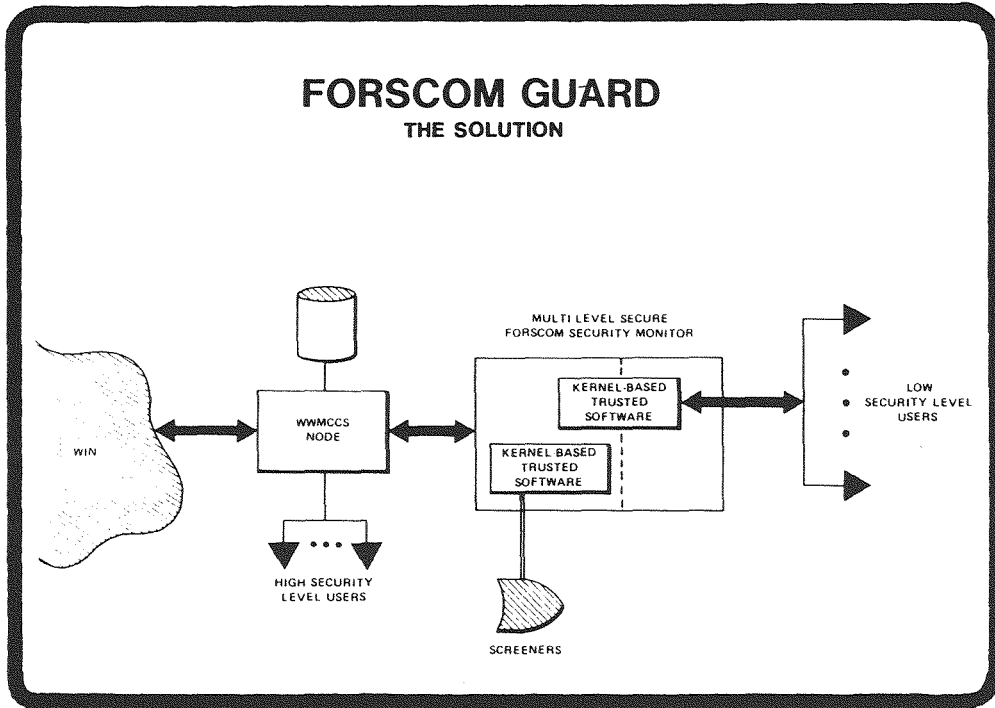
## FORSCOM GUARD

### THE PROBLEM



## FORSCOM GUARD

THE SOLUTION



## FORSCOM GUARD

### FUNCTIONAL DESCRIPTION

- "INTERACTIVE" ORIENTED
  - MEDIATES BETWEEN ALL LOW USER AND HIGH SYSTEM DIALOGUES
  - PROVIDES BOTH "MANUAL" AND "AUTOMATIC" DOWNGRADE MECHANISMS
  - PROVIDES LOW USER INPUT "FILTER" MECHANISM
- SCREENER PERSONNEL
  - VIEWS ALL "MANUAL" HIGH-TO-LOW DATA TRANSFERS
  - INTERFACES WITH "TRUSTED SOFTWARE" FOR "DOWNGRADING" OF DATA

# FORSCOM GUARD

## SECURITY POLICY

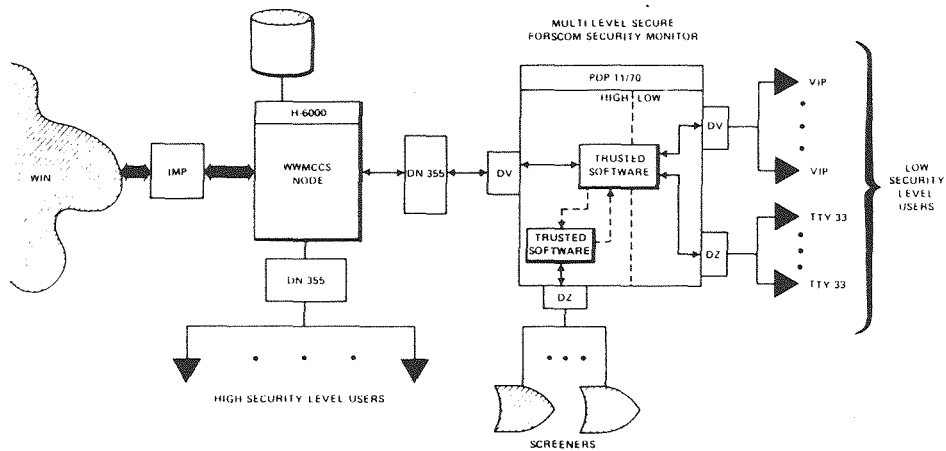
- DATA SEPARATION (DoD SECURITY MODEL)
  - SIMPLE SECURITY CONDITION ("READ" RULE)
  - \*-PROPERTY CONDITION ("WRITE" RULE)
  - TRANQUILITY CONDITION ("ALTER" RULE)
- DATA INTEGRITY ("DUAL" OF DoD SECURITY MODEL)
- DISCRETIONARY ACCESS (A LA UNIX)
- MANUAL DOWNGRADE POLICY (VIOLATES \*-PROPERTY)
  - SCREENER VIEWS ALL DATA
  - SCREENER ACCEPTS DOWNGRADE
  - SCREENER CONFIRMS DECISION
- AUTOMATIC DOWNGRADE POLICY (VIOLATES \*-PROPERTY)
  - ALL DATA IS RECOGNIZABLE IN PROPER CONTEXT
  - "BANDWIDTH" NOT EXCEEDED
- ACCEPT USER INPUT POLICY (A "FILTER")
  - DATA IS RECOGNIZABLE IN PROPER CONTEXT
- AUDIT ALL HIGH-TO-LOW DOWNGRADES

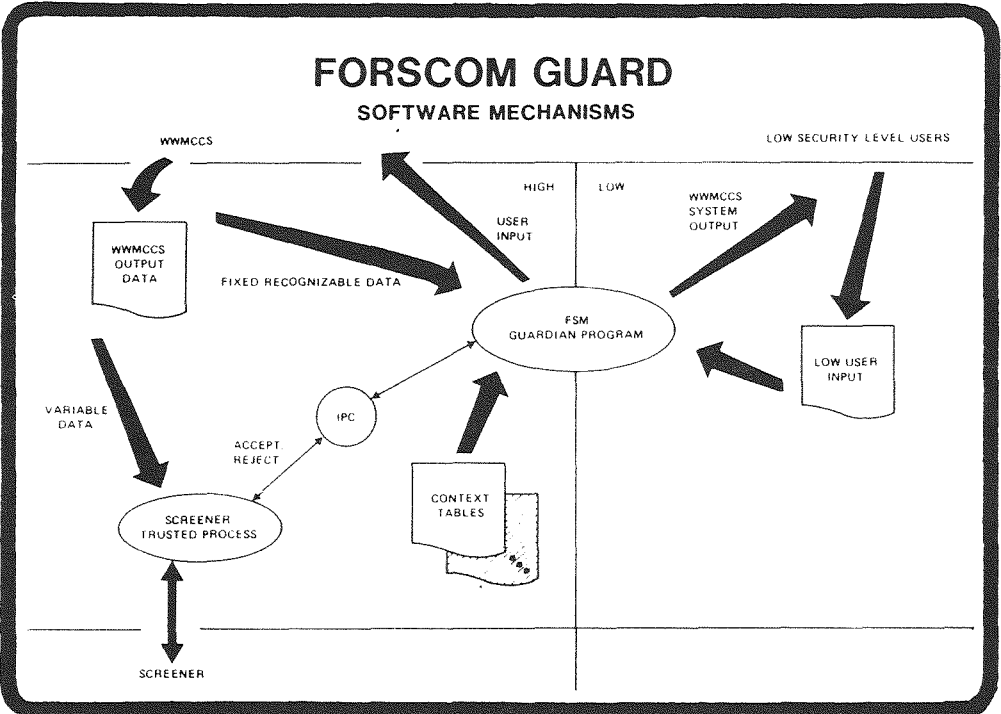
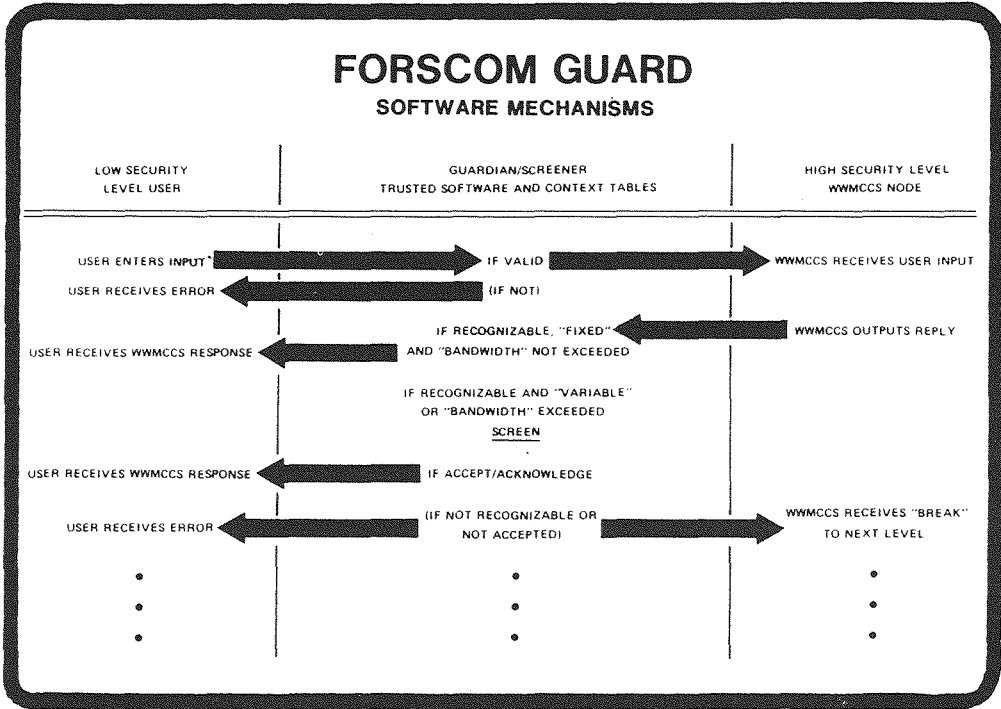
KSOS ENFORCED

TRUSTED SOFTWARE ENFORCED

# FORSCOM GUARD

## HARDWARE CONFIGURATION

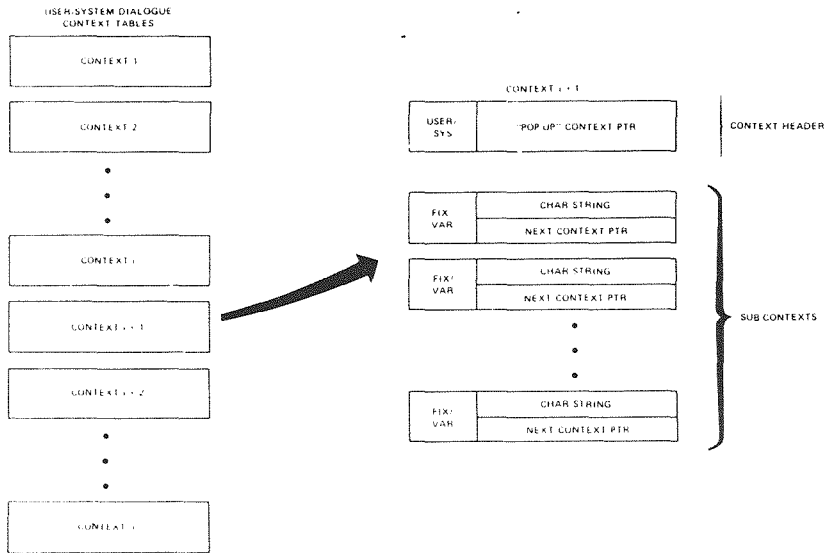






## FORSCOM GUARD

### SOFTWARE MECHANISMS



## FORSCOM GUARD

**STATUS: PRESENT AND FUTURE**

### PRESENT

- HARDWARE INSTALLED AT FORCES COMMAND, FT. GILLEM
- SOFTWARE OPERATIONAL – DEMONSTRATABLE UNDER UNIX
- FORMAL SPECIFICATION OF TRUSTED SOFTWARE UNDERWAY

### FUTURE

- KSOS-11 INSTALLATION PLANNED
- VERIFICATION OF TRUSTED SOFTWARE
- OTHER WWMCCS APPLICATIONS PLANNED

# A Security Model for a Military Message System

Carl E. Landwehr

Computer Science and Systems Branch, Code 7590

Information Technology Division

Naval Research Laboratory

Washington, D.C. 20375

[Portions of this work were sponsored by  
the Naval Electronics Systems Command,  
Code 8144, H. O. Lubbes.]

## Outline

What security models are good for

History of security models

Experience with Bell and LaPadula model

An application-based approach

Security model for a military message system:

Current version

Definitions

Model of operations

Security Assumptions

Security Assertions

Regimes for accessing objects within containers

Outstanding Issues

Plans

What security models are good for

Define what "security" means in a given system

Provide basis for understanding system operation

Provide basis for proofs

History of security models

Operating system protection models

Models incorporating DoD security

Access Control (Bell and LaPadula)

Information Flow (Denning)

Revised Bell and LaPadula

Experience with Bell and LaPadula model

MME - trusted job

KSOS - NKSR

Guard - trusted processes

An application-based approach

User's view of the system

Components of an application-based model

Definition of terms

Model of operations

Assumptions

Assertions

How the model can be used

Current version of the MMS model

Definitions

Classification - disclosure and modification levels

Clearance - user disclosure level

User ID - one per user

Role - function performed by user

Access control list - pairs (UserID or Role, Access mode)  
access modes include read, write, execute,  
may be attached to objects, containers

Object - smallest unit with explicit classification  
(single level)

Container - has classification and may contain objects  
or other containers (multi-level)

Entity - object or container. Each entity can be  
designated by unique ID or pathname

Program - sequence of machine-executable instructions  
may have an associated clearance and UserID

Message - a particular type of container

Examples of objects

Date-time group  
Subject  
Precedence

Examples of containers:

Text  
Message  
Message File

Entities that might be containers in one system and objects in another

Address list  
Comments

Some operations applicable to messages

Compose	Edit
Output	Update
Send	Release
Forward	Distribute
Coordinate	Chop
Readdress	Reclassify
Delete	Undelete
Destroy	Assign-action

#### Model of operations

- User gives UserID and is authenticated by the system
- User invokes programs to perform the functions of the message system
- The programs a user may invoke depend on the user's role
- A user with the role of System Security Officer controls the clearances and roles assigned to UserIDs
- Programs a user invokes may read, write, or invoke objects or containers
- The system enforces the security assertions listed below (prevents users from performing operations that would contradict them)

#### Security assumptions

- A1. Security officer assigns clearances and roles properly to users.
- A2. User enters appropriate classification when composing, editing, or reclassifying text.
- A3. User exercises proper control of access control lists.

#### Security assertions

##### Disclosure of information

- D1. A user can only view objects with disclosure level less than or equal to  $glb(\text{UserID}, \text{Role}, \text{Output Device})$ . For objects within containers, either the container's disclosure level or the object's disclosure level will be checked, depending on the type of the container and the mode of access (by unique ID or pathname).

Security assertions (cont'd)

Modification of information

- M1. Users can only modify objects with modification level less than or equal to the glb of User, Role, and Input Device modification levels.
- M2. The disclosure level of any container is always at least as great as the maximum of the disclosure levels of the objects and containers within it.
- M3. No classification marking can be downgraded except by a user with the role of downgrader.
- M4. The clearance recorded for a UserID can only be set or changed by a user with the role of system security officer.
- M5. No message can be released except by a user with the role of releaser.
- M6. No user can invoke a program for which his UserID or role is not on the access control list with an access mode of execute.

Noteworthy aspects of the model:

Multi-level objects (containers) are defined

Simple security condition is reflected in D1.

\*-property is not included, but "write-downs" are controlled via M2 and M3

Integrity is included as modification level

Login level is not included, but I/O device abstractions can provide this effect

Programs, not processes, are included because they are more recognizable to users

Implementation concepts (e.g., capabilities) are avoided, but model is designed to be implementable

### Example regimes for accessing objects within containers

1. Access to object is allowed only if the user and role clearances equal or exceed the classification of the container. If data is copied from the object to another entity, that data is treated as though it had the same classification as the container.

[Apply this regime to aggregation-sensitive data.]

2. Like (1), but data copied from the object is treated as though it has the same classification as the object, regardless of the container's classification.

[Apply this regime to extraction of a paragraph of text from a message.]

3. Like (2), but only the user's clearance must equal or exceed that of the container.

[Apply this regime to viewing of messages within a message file.]

### Outstanding issues

Mathematical properties of the model

Possible abstraction of model for proofs

Development of design and implementation from model

Detailed design questions --

Determine whether each abstraction is  
an object or a container

Determine appropriate regime for each  
type of container

Determine mappings between family members  
that make different container/object  
choices for a given entity



## Plans

Refine/revise the security model

Integrate with MMS Intermediate Command Language Specification

Consider man-machine interface questions

Design and develop prototype system based on this model

## Bibliography

1. Landwehr, C.E., "Formal Models for Computer Security," to appear, ACM Computing Surveys, September, 1981. Also available as NRL Report 8489.

A comprehensive survey of previous formal models.

2. Miller, J.S., and Resnick, R.G., "Military Message Systems: Applying a Security Model," IEEE Symposium on Security and Privacy, April, 1981.

A discussion of an earlier version of the MMS security model, with an application to a message system based on an Intermediate Command Language specification. Introduces three regimes for accessing entities within containers.

3. Landwehr, C.E., "Assertions for Verification of Multilevel Secure Military Message Systems," Verification Workshop, SRI, 1980, reprinted in ACM SIGSOFT Software Engineering Notes, Vol. 5 No. 3, July 1980, pp.46-47.

Presents the motivation for application-based models and the first version of a security model for military message systems. Still useful, but somewhat dated, as the version of the model presented does not include the concept of roles and leaves several issues unresolved.

4. Heitmeyer, C.L. and Wilson, S.H., "Military Message Systems: Current Status and Future Directions," IEEE Trans. on Comm., Vol COM-28, No. 9, Sept. 1980, pp.1645-1654.

Discusses the family of message systems for which the security model is defined. Describes the application of the program family principle to the design of message systems.

## **THE (ORIGINAL) EUCLID LANGUAGE**

- **MAJOR APPLICATION:  
PROVABLY SECURE SOFTWARE**
- **SYSTEM IMPLEMENTATION LANGUAGE**
- **ALLOWS VERIFIABLE PROGRAMS  
TO BE WRITTEN**

## **EUCLID AND VERIFICATION**

- **THE EUCLID LANGUAGE**
- **INTEGRATED VERIFICATION SYSTEM  
(EUCLID + VERIFICATION TOOLS)**
- **FUTURE DIRECTIONS**

## **HISTORY**

- **DESIGN COMMISSIONED BY DARPA**
- **DESIGNED BY EUCLID COMMITTEE**
- **PASCAL + VERIFICATION FEATURES**
- **PDP-11 COMPILER FOR TORONTO EUCLID  
SUBSET IMPLEMENTED BY:**
  - I.P. SHARP ASSOCIATES
  - UNIVERSITY OF TORONTO C.S.R.G.
- **TORONTO EUCLID BOOTSTRAPPED TO VAX**

## **MODULES**

- RECORDS WITH ATTACHED ROUTINES
- INTERFACE TO OUTER PROGRAM EXPLICITLY SPECIFIED
- SUPPORT INFORMATION HIDING, ABSTRACT DATA TYPES

## **VISIBILITY AND ACCESS CONTROL**

- MODULES AND ROUTINES IMPORT GLOBAL NAMES
- MODULES EXPORT INTERFACE NAMES
- READ / WRITE OR READONLY ACCESS

## **ANNOTATIONS**

- ASSERTIONS
- PRE, POST FOR ROUTINES
- MODULE INVARIANT

## **RESTRICTIONS TO HELP VERIFIER**

- NO ALIASING: ONE NAME FOR EACH DATA ITEM
- NO OVERLAP
- NO GO TO STATEMENT
- LEGALITY ASSERTIONS

## **PDP-11 TORONTO EUCLID COMPILER**

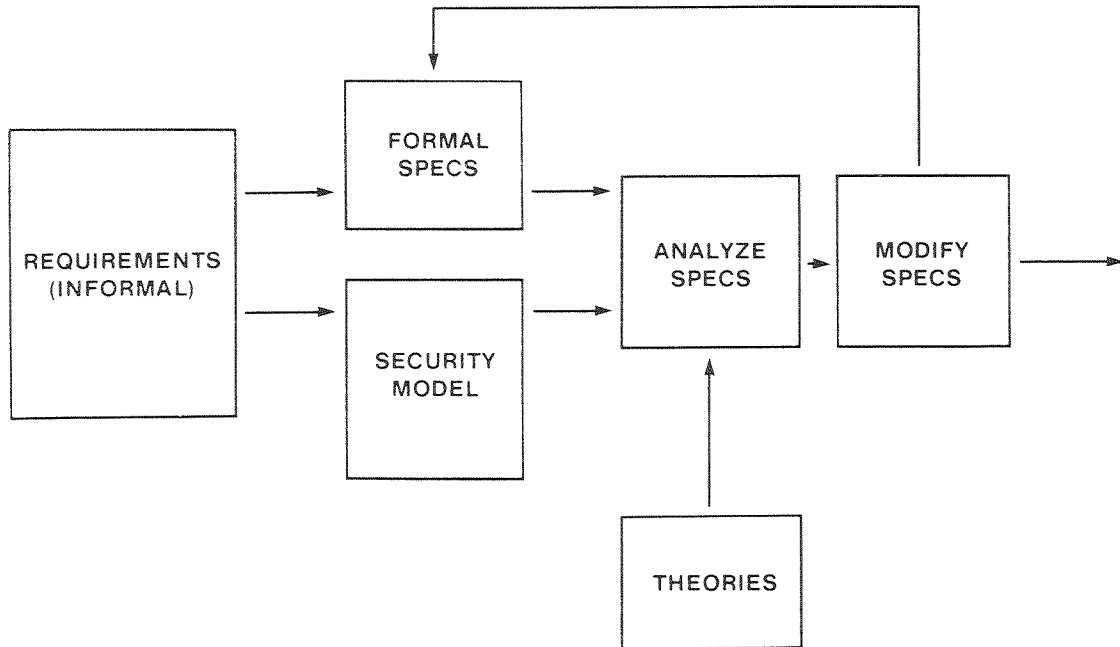
- OBJECT CODE EFFICIENCY: VERY GOOD
- COMPILER SPEED: SLOW
  - STRICT CHECKING TAKES TIME
- PROGRAMMER EFFICIENCY: VERY GOOD
  - STRICT CHECKING SPEEDS UP PROGRAMMING
- AVAILABLE: NOW, FROM IPSA

# INTEGRATED VERIFICATION SYSTEM

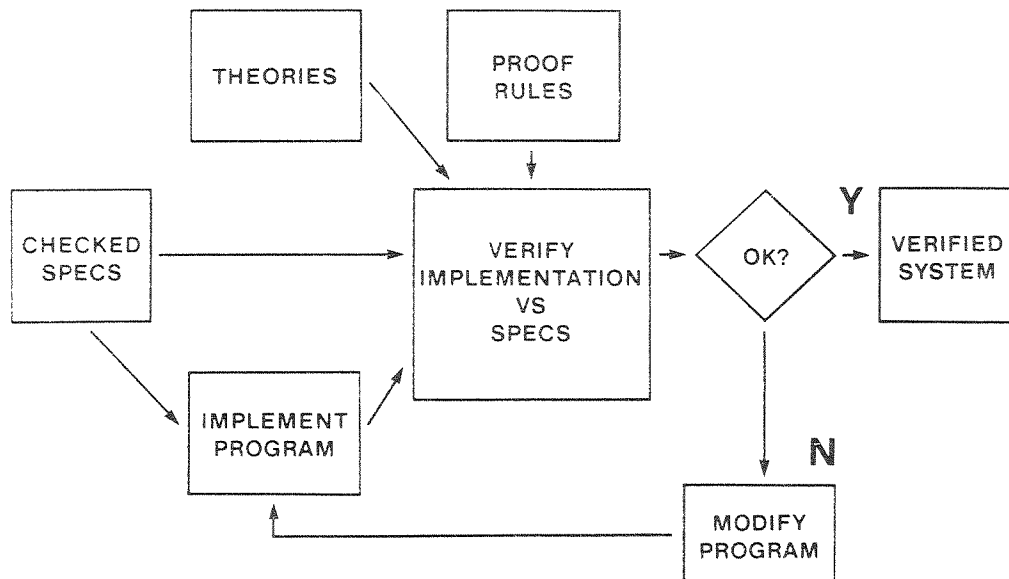
## OBJECTIVES:

- EUCLID AS IMPLEMENTATION LANGUAGE
- INTEGRATE EXISTING VERIFICATION TECHNOLOGY
- USER-FRIENDLY CONSISTENT SYSTEM
- RE-USABLE VERIFIED SOFTWARE MODULES
- MAJOR APPLICATION:  
PROVABLY SECURE SYSTEMS

## STEPS IN VERIFICATION — SPECIFICATIONS



## STEPS IN VERIFICATION — IMPLEMENTATION



# OTTAWA EUCLID

## IMPLEMENTATION EUCLID

### TORONTO EUCLID

- EXISTING  
PDP-11  
COMPILER

MORE  
EUCLID  
FEATURES

SPEC  
+  
THEORY  
EXTENSIONS

SEPARATE  
VERIFICATION  
+  
COMPILATION

# OTTAWA EUCLID

## IMPLEMENTATION EUCLID

### TORONTO EUCLID

- EXISTING  
PDP-11  
COMPILER

MORE  
EUCLID  
FEATURES

SPEC  
+  
THEORY  
EXTENSIONS

SEPARATE  
VERIFICATION  
+  
COMPILATION

# **TORONTO EUCLID RESTRICTIONS REMOVED**

- FUNCTIONS CAN RETURN STRUCTURES
- PARAMETERIZED TYPES
- LEGALITY ASSERTIONS CHECKED

## **ENHANCED ASSERTION LANGUAGE**

- QUANTIFICATION
- IF EXPRESSIONS
- SPECIFICATION FUNCTIONS AND VARIABLES
- LEMMAS AND AXIOMS

## **SEPARATE VERIFICATION / COMPILATION**

- STUB FOR EXTERNAL MODULE  
= SPECIFICATION
- LINK-TIME CHECK OF  
STUB VS. IMPLEMENTATION

## **ADVANTAGES**

- ONE CONSISTENT LANGUAGE
- TYPE-SAFE SPECIFICATIONS  
AND THEORIES
- RE-USE EXISTING COMPILER  
SOFTWARE FOR TYPE CHECKING

## **FUTURE DIRECTIONS**

- LSI GUARD
- CONCURRENCY
- OTTAWA EUCLID COMPILER
  - FIRST IMPLEMENTATION: VAX II
  - AVAILABLE: MID 1983
- ADAPT EXISTING TOOLS TO  
OTTAWA EUCLID

THE EVALUATION  
OF THREE  
SPECIFICATION  
and  
VERIFICATION  
METHODOLOGIES

by

Richard A. Platek

Digicomp Research Corp. Ithaca, N. Y.



Digicom Research Corp. is presently under contract with DoD through the Rome Air Development Center (RADC) to study and evaluate three specification and verification methodologies. They are HDM (SRI International), FDM (or Ina Jo, SDC) and Gypsy (UTexas). This thirty month effort which began Sept. 1980 has three main phases:

a. Impartial, critical analyses of the methodologies with special attention paid to their present state of usability by persons not directly associated with the developers and an evaluation of the expertise required in such a technology transfer.

b. Recommendations for enhancements some of which will be subcontracted to the major developers through Digicom (subject to government approval) while others will be used to drive further funding through other agencies.

c. The design, implementation and verification of a secure data base management system using each of the methodologies. The mathematical model of such a secure DBMS is based on previous work by I. P. Sharp.

The first and most of the second of these phases have been completed while the third is underway. In this talk I would like to describe some of our findings so far. The work has been performed by Tanya Korelsky, Len Silver and myself. As an indication of our backgrounds I should state that all three of us have Ph.D.s in Mathematics but no prior experience in verification.

Like many developing software systems these methodologies' documentation sometimes contain features which have not yet been implemented. Considering the fact that these implementations are ongoing our remarks could best be treated as time-stamped snapshots of evolving systems. Furthermore, since these tools have not been subjected to extensive use outside of their places of origin it is important to obtain independent evaluations based on sustained hands-on experience. The comparative method that has been chosen seems to us and our sponsors to be the best technique for revealing the strengths and weaknesses of the existing methodologies and for making recommendations that could be incorporated in future specification and verification work.

Although we will briefly review the paradigms which underlie each of the methodologies our time constraint forces us to assume that the hearer has been exposed to more detailed descriptions of the methodologies as they have been described by their developers at these and similar meetings.

#### I. HDM

The present situation with HDM is quite complicated due, in our opinion, to the large turnover in extremely talented personnel at SRI who have been involved over the years with the HDM project and the absence of a central authority who would have had the power to curtail creativity in the interests of consistency. While such a production

system orientation is inconsistent with research goals and verification as a whole has benefited from SRI's experiments it is a fact that HDM presently consists of several well thought out and engineered components that lack integration. Although SRI is aware of this problem and it is currently being addressed by ongoing work it is fair to say that at present an outsider can not use HDM to design, implement and verify a program from beginning to end. Although our study was completed last March and reflects the system as it was then we've kept abreast of the more recent changes.

HDM specifications are written in SPECIAL a non-procedural strongly typed assertional language based on first-order logic. The unit of specification is the module which is an encapsulated abstract data type. Following Parnas modules are described as abstract automata defined in terms of states and state transforms. (We prefer the terms "state" and "transform" to the awkward "V Function" and "O Function" terminology; unfortunately SPECIAL maintains the original Parnas nomenclature which is confusing to new users. Ina Jo uses the terms "variable" and "transform", the state being the values of all the variables at any given moment.) These modules are grouped together to form virtual machines which in turn are levels in a hierarchy. The top level of this hierarchy is the user interface while the bottom is the "machine" on which the system is to run; the latter is not necessarily a physical machine but can be a combination of hardware and software, for example a PASCAL

or ADA machine. Adjacent levels are related by mappings which are of two kinds. The state or data mappings provide an image,  $Image(S)$ , on the upper level for each lower level state  $S$ . These are described using SPECIAL expressions. The transform or procedure mappings express each higher level state transform  $T$  as a program  $P(T)$  which "runs" on the lower level machine and calls lower level state transforms.  $P(T)$  is correct if whenever it drives the lower state  $S1$  to the lower state  $S2$  then  $T(Image(S1))$  is  $T(Image(S2))$ . Said simply this means that the program  $P(T)$  simulates  $T$  on the lower machine. The program  $P$  is written in the target HOL. In the original HDM conception a new programming language, ILPL, was designed for this purpose but this approach has been abandoned. When all the  $P(T)$  have been verified to be correct the transform mappings can be composed to yield a complete verified implementation of the top level virtual machine on the bottom. The composition of transform mappings is reflected in the resulting program by procedural call nesting; the depth of this nesting being essentially the length of the hierarchy.

Unfortunately the specification language SPECIAL comes in several variants. First there is the original version of SPECIAL which we will call Handbook SPECIAL. The publicly available HDM automated tools which check for syntax correctness, hierarchical consistency and certain forms of completeness are written to this SPECIAL. These tools contain bugs which were discovered in the course of our

testing. These bugs have not been corrected for reasons outlined below.

Handbook SPECIAL contains many features which its designers thought would be useful in specifying complex systems. This compounding of features led to a language without a clear semantics (or perhaps a better way to say it would be a language susceptible to several overlapping semantics). For example, a. First order logic is used to express system states before and after transforms are called, b. A New operator creates new objects of a certain kind of type when called, c. Exception conditions for transforms must be evaluated in a certain order, d. There are unusual constructs like "Delay Until". Because of the incompatibility of the various semantics of these languages it was found to be necessary to subset SPECIAL whenever any design or code verification issues arose. For example, SRI produced a multilevel security information flow analysis tool. This works on the top level SPECIAL spec and uncovers information flow. The tool is very conservative and considers an information flow to occur between variables whenever the former is referenced in any way by the latter (e.g., in the assignment statement  $v1 := 0*v2$  information is assumed to have flown from  $v2$  to  $v1$ ). In order to make this analysis it was found necessary to restrict the kind of expressions that occur in specs. This gives rise to MLS SPECIAL. Every variable at the top level is assigned a security level and the MLS tool checks that information only

flows upward in level. To do this it produces formulas which are handed over to the Boyer-Moore theorem prover. The latter has its own language designed according to very different principles than those that govern SPECIAL.

HDM's original attempt at code verification involved the use of a pseudo-assembly language CIF (Common Internal Form) set up within Boyer-Moore theory. A MODULA translator translated MODULA code into CIF and the latter was proved correct within Boyer-Moore theory. In order to do this the SPECIAL specs had to also be translated into Boyer-Moore. A very impoverished subset of SPECIAL was developed called VSSL. No automated tools were provided, the program had to be respecified in VSSL. There are many discrepancies between SPECIAL and VSSL. VSSL and CIF for example understand integer to be non-negative while SPECIAL and MODULA understand integer to be positive or negative. VSSL does not allow any existential quantifiers in the effects section of a transforms spec. Furthermore all the code verification required large amounts of manual intervention to add statements necessary to achieve a proof. The smallest programs took an enormous amount of time to verify and when done it was not clear what had been verified since VSSL was not SPECIAL and CIF was not MODULA.

As part of the SIFT project SRI is developing a PASCAL verification system for HDM. This has involved a new version of SPECIAL, Pascal SPECIAL, and tools to check it.

It is largely because of this effort to build new tools that the above mentioned bugs in the existing SPECIAL tools have not been fixed. The concept of CIF has been abandoned. Instead a Meta-Vcg has been built which accepts an axiomatic definition of a programming language, code in that language and specs and then generates VCs directly. Originally the VCs were in Boyer-Moore theory. Boyer and Moore have both recently left SRI to join the University of Texas. As a result SRI has begun adapting HDM to run with the Shostak theorem prover. The Meta-Vcg for example will output Vcs in either Boyer-Moore or Shostak theory. We have not had any experience with the latter. Unfortunately Pascal SPECIAL does not contain MLS SPECIAL as a subset so that the Pascal system as it now stands can not be used for security proofs.

As a result of these uncertainties Digicomps subcontracted with Sytek to do a study of the SPECIAL dialects and make recommendations for standardization. This study is being directed by Rich Feiertag a former SRI member and principal designer of the MLS tool. It will be completed in Sept. We hope someone will be in a position to act on the recommendations in the report which from what we've seen is very thoughtful.

At SRI work is proceeding in matching the HDM tools more closely with the Shostak theorem prover, in developing proof rules to deal with full concurrency, and with developing a new specification language called ORDINARY.

## II. FDM

FDM is superficially similar to HDM although in many ways this similarity is misleading. Like HDM, FDM specifies a system using a series of levels each described in terms of states and transforms with adjacent levels related by mappings. But unlike HDM the top level is not the user interface. Instead it is to be thought of as either an incomplete abstract description of the final system which omits certain design decisions and contains others or, more mathematically, the top level is a specification of a family of systems one of which is the intended final system. Each subsequent level also defines a family of systems. As in HDM the mappings between levels relate the states of the lower to the states of the higher and the transforms of the higher to the transforms of the lower. They are said to be correct for each adjacent pair if the family of systems specified by the lower is a subset of the family of systems specified by the upper relative to the mappings which act like a dictionary enabling one to translate higher level expressions into lower. Thus each level can be thought of as a refinement of its predecessor. This refinement proceeds by adding further detail and concretion.

Levels are described using Ina Jo, a specification language similar to SPECIAL but cleaner in syntax and semantics. Unlike SPECIAL only one kind of semantics is involved, namely first order logic. This leads to



simplicity in expression and ease of provability. The cost is lack of expressiveness but Ina Jo is presently being upgraded to include more extended expressiveness.

Ina Jo provides a means for proving that the top level spec satisfies user supplied design criteria. These are written in Ina Jo and are syntactically part of the top level spec. Only experience will show whether this approach is adequate for the expressing of interesting security properties. In order to prove these design criteria for the top level one submits the spec to the language processor which produces candidate theorems the truth of which imply that the criteria holds of all systems that satisfy the top level spec. These theorems are proved using the associated ITP (Interactive Theorem Prover). The latter is simple to use and well integrated into the system. It is not very powerful but is continually being upgraded. It contains, for example, very little arithmetic since there has been very little need for it in the projects SDC has used Ina Jo on.

State mapping between adjacent levels are essentially the same as HDM but the transform mappings are logical rather than procedural. If  $T$  is an upper level transform then the mapping of  $T$ ,  $M(T)$ , is essentially

IF COND1 THEN D1 ELSE

IF COND2 THEN D2 ELSE

IF COND3 THEN D3 .... where the COND's are Ina Jo Boolean expressions describing subsets of the lower level state space and the D's are lower level transforms. With respect to these mappings one proves that each subsequent level is a refinement of its predecessor. As in the case of the proof of the top level design criteria candidate theorems are produced by the language processor from each pair of adjacent levels and these are proved using the ITP.

All that we have described so far is implemented but we should remark that when all this proving is completed one still has not verified any HOL code let alone written it. This is not meant to imply that describing, refining, and proving the specifications in this way is without value. We did not encounter serious difficulty in using the system. Since the ITP is weak many "self-evident" axioms had to be manually added to finish proofs. This is obviously a dangerous procedure in verification since "self-evident" sometimes turns out to be false. We have made SDC aware of all bugs and unimplemented details we have come across and they intend to act on them. FDM was produced primarily with internal SDC funds and is a proprietary product. Since it is fashionable in Washington nowadays to extol the spiritual values of capitalism one should remark here that private property tends to be kept up by its owners.

We now describe SDC's intentions with respect to code verification. In FDM all code verification will be done after all levels have been designed and proved. Beneath the bottom level Ina Jo spec there will be an implementation level which relates the bottom level Ina Jo variables and transforms to the names of HOL variables and transforms. An extension of the language processor not yet implemented will take this level and generate entry and exit conditions for the HOL procedures. These together with HOL code will yield VCs when passed through a VCG (a MODULA VCG for Ina Jo entry and exit conditions is near completion). In addition in order to show that the resulting program is an instance of the family of systems specified by the Ina Jo spec it will be necessary to check that in the resulting master program the entry conditions for each HOL procedure hold whenever it is called. The reason the mappings of transforms between levels are restricted to the form we described is to make it possible to assemble mechanically the entry conditions for each HOL procedure. This is a subtle point not mentioned in SDC documentation and was the cause of some misunderstanding among outside students of Ina Jo.

Much work remains to be done to complete the code verification aspects of Ina Jo and Digicomp expects to fund some of it. It is premature to make judgements but our experience with HDM leads us to suspect that code verification is not as simple as some would maintain. For example since the present version of FDM does not admit

modularity the amount of work to be done at code verification time may be inordinately large and various means to structure it may have to be devised. The theorem prover will need to be upgraded to handle the full spectrum of mathematics that occurs in program verification.

### III. GYPSY

Unlike HDM and FDM Gypsy is both a programming and specification language. Gypsy text appears like a Pascal program in which specifications are interspersed at key points. For example, every procedure and function has entry and exit assertions and every loop is broken by at least one assertion. Verification conditions are generated from these specifications and code, and these VCs are submitted to an integrated theorem prover. This is the central loop of the Gypsy Verification Environment (GVE). This environment is quite congenial. It contains a library manager which keeps track of the various parts of the verification process and their status, an internal structured editor and links to external editors like emacs, facilities to incrementally write and prove code, etc.

As a result of the integrated language there is no strict separation of design and implementation in Gypsy. The user can shade a Gypsy source text heavily towards the one or the other. It could be pure specification with no code or pure code with no specification. The latter is compilable with PDP-11 object code.

As mentioned above the Gypsy environment lends itself to incremental usage. Pieces of program are written and verified. Some of these pieces are high-level routines and some low-level. The bodies of the latter may be left pending while their entry and exit assertions are used to prove the correctness of the high level routines. The system could be used as a vehicle for many design strategies such as "stepwise refinement", top-down", etc.

Gypsy also provides a limited form of concurrency through the use of buffers that simultaneous running processes can send to or get from. Proof techniques have been developed to handle the logic of this kind of concurrency.

Gypsy's claim to fame is that one can actually produce verified code. The main caveat seems to be that since the specification is so close to the implementation level it is not simple and abstract enough to get a firm grasp on what has been proved about a large system. It lacks for example Ina Jo's facility of expressing a design criteria at a high level and then using mappings as a dictionary to unabbreviate it down to a condition on actual program variables. Gypsy does recognizes the need to provide mechanisms of abstraction so that the intent of the code becomes more transparent. But it seems that this goal was given a lower priority than the the admirable one of producing a system in which one can develop verifiable and

compilable code. One such abstraction mechanism is a form of abstract data types using access lists. It is described in the Gypsy language manual but hitherto not implemented. This is one of the enhancements currently -being funded by Digicomp.

Although Don Good and his senior assistant Rich Cohen have been with the system since its inception many people have worked on Gypsy as graduate students at UTexas. This is reflected in a certain unevenness in the components. An embarrassing example is that although unproved lemmas sometimes have to be added to the knowledge base in order to complete proofs their status as unproved can be forgotten by the system. Digicomp is funding a top level reimplementaion which will address some of these issues. This will be done in a yet to be finalized dialect of LISP with portability a major concern in the choice. The present version is written in UCI LISP running under Tops-20 and runs into space problems when verifying medium size programs.

From a logician's point of view the major criticism of the system is that it deals only with partial rather than total correctness as these terms are used in the field of program verification. This means that is no mechanism is provided to prove termination of subroutines. All functions are dealt with by the theorem prover as if they were total and in this way an unsoundness could be introduced. The

MODULA-CIF version of HDM attempted to deal with this problem through the use of a user supplied clock function. Ina Jo has not faced this issue yet.

I would now like to mention some areas for possible research.

1. There is a need for an understandable, intelligible specification language. The present specification languages are like the "machine language" of specification languages. They are difficult to read, too homogeneous. It seems the proper constructs peculiar to specification remain to be discovered.

2. Theorem proving is the big bottleneck in code verification. Proof checkers are too pedestrian and the automatic ones run away. The ideal would be a system which could take a sketch of a proof and expand it into a real proof. I don't believe this is an impossible goal, I do believe it is a necessity if large scale code verification is to become a reality but I also feel it requires a significant breakthrough in the field of automatic theorem proving. The latter is a pure research area involving mathematics, logic, and artificial intelligence.

3. Integration seems to be the key to success in this area. Ina Jo is weaker piece by piece than HDM but the integration the system has compensates. Gypsy is the most satisfying to work with because of its integration.

4. Finally I would like to mention the possibility of using approaches to code verification other than VC generation. There are several such methods available which allow one to use the program text itself in proofs rather than translations of it into another language. The advantages of the vcg approach is that one can use general purpose theorem provers since the VCs are statements in ordinary mathematical language. The advantage of the non VCG approach is further integration and unity. The less translation from one language to another the greater chances for success.



## COMPUTER SECURITY RELATED PUBLICATIONS

Listed below are titles and accession numbers of some computer security related publications which are now available from the Defense Technical Information Center (DTIC), Defense Logistics Agency, Building 5, Cameron Station, Alexandria, Virginia 22314 (Phone 202-274-7633, AUTOVON 284-7633).

Firms or individuals registered with the DTIC may obtain copies for a flat fee per document. Those who are not registered with the DTIC may obtain copies from the National Technical Information Service, 5285 Port Royal Road, Springfield, Virginia 22161. Orders may be placed or price quotations may be obtained for each document by calling 703-487-4650.

AD A101 996	Proceedings of the Third Seminar on the DoD Computer Security Initiative
AD A101 997	Proceedings of the Second Seminar on the DoD Computer Security Initiative
AD A101 998	Proceedings of the Seminar on the DoD Computer Security Initiative - (First Seminar)
AD 076 617	Security Controls for Computer Systems Report of DSB Task Force on Computer Security (Rand Ware Report, October 1979)
AD A103 399	TRUSTED COMPUTER SYSTEMS - Needs and Incentives for Use in Government and the Private Sector (Rand Turn Report, June 1981)
AD A095 409	Modernization of the WWMCCS Information System (WIS) (DCA) January 1981
AD A108 829	Trusted Computer Systems-Glossary (Huff, MITRE), March 1981
AD A108 827	Computer Security Bibliography (Discepolo, MITRE) November 1980
AD A108 828	Industry Trusted Computer System Evaluation Process (Trotter and Tasker, MITRE) May 1980
AD A108 830	History of Protection in Computer Systems (Tangney, MITRE), July 1980
AD A108 831	Specification of a Trusted Computing Base (TCB) (Nibaldi, MITRE) November 1979

AD A108 832 Proposed Technical Evaluation Criteria for Trusted Computer Systems (Nibaldi, MITRE) October 1979

AD A109 317 Formal Specifications of KVM/370 Kernel and Trusted Processes (Gold and Thompson, SDC) May 1978

AD A109 316 Final Report VM/370 Security Retrofit Program-Detailed Design and Implementation Phase (Gold and others, SDC) May 1978)

AD A109 318 Semi-Formal Description of KVM/370 Trusted Processes (Thompson, SDC) December 1977