LEVEL II

MITRE-Bedford Division

MTR-3999

81 12 22 109

AD A108830

History of Protection in Computer Systems

(12) 82

John D. Tangney

15 JULY 1980

235090

MITRE

MITRE Technical Report

MTR-3999

History of Protection in Computer Systems

John D. Tangney

15 JULY 1980

CONTRACT SPONSOR OSD (C3I)
CONTRACT NO AF19628-80-C-0001
PROJECT NO 8420
DEPT. D75

THE MITRE CORPORATION
BEDFORD, MASSACHUSETTS

Department Approval: _____

MITRE Project Approval: _____

ABSTRACT

This report documents a lecture delivered at the NSA Computer Security Workshop on 19 March 1980. The subject of the lecture was the evolution of information protection features in commercially available general purpose computing systems. The lecture covered features built into both computer hardware and operating systems in an attempt to prevent programs of one user from stealing, modifying, or destroying programs of other users and of the operating system.

This report is more than just a script for the lecture. In explaining certain features, it goes into much greater detail than is covered during the actual lecture, especially when presenting those hardware features considered essential to the development of trusted operating systems.

This report should be useful to readers interested in the multilevel computer security problem. An appendix is included which provides contextual information about the multilevel security problem and the need for trusted operating systems.

ACKNOWLEDGEMENTS

This lecture has evolved from one developed by Ed Burke, entitled "Computer Security Technology Introduction, History, and Background", given originally at the MITRE Computer Security Workshop in January 1979, and which was revised and redelivered by Ed with the title "Protection in Operating Systems: A Technical History," at the NBS Seminar on DOD Computer Security Initiative Program in July 1979.

Comments and suggestions by Pete Tasker were very helpful in bringing this lecture to its present form.

History
Of
Protection
In
Computing
Systems

The intent of this briefing is to consider the evolution of information protection features in commercially available computing systems. We will examine those features designed into the computer hardware and into operating systems in an attempt to prevent programs of one user from stealing, modifying or destroying the programs and data of other users and of the operating system.

Outline

Computer generations

Operating system support

Protection features

Residual problems

In order to structure this presentation, we will consider the evolution of information protection features starting from the first generation of computer systems and continuing right up to state-of-the-art or current generation computer systems.

Our purpose in doing so is to examine the principles underlying the provision of effective information access controls within a computer system. We shall see when people first became concerned about the protection of information and the provision of internal access controls, and we shall look at some of the hardware and software mechanisms developed to address the protection problem.

We shall see that the protection of information was not really a problem during the first and second generations because they were single-user-at-a-time systems. Protection became a cause for concern with the third computer generation when systems were designed to share processor and storage resources among several concurrent user programs.

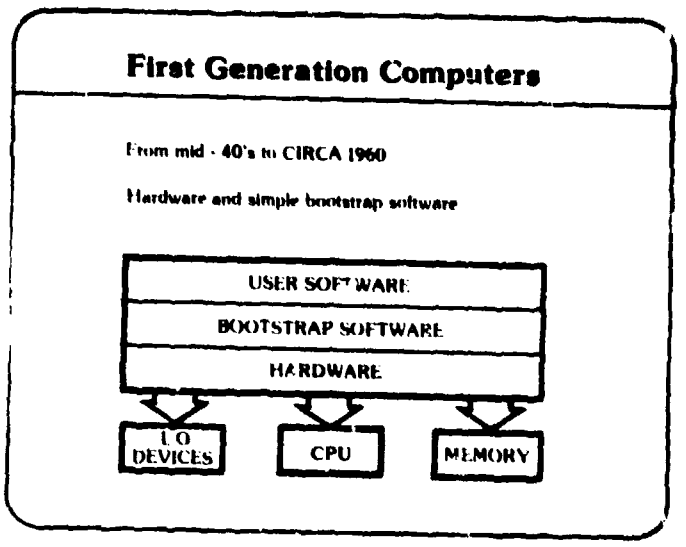
As we move through each generation, we shall first examine the nature of the operating systems typical of that generation. We will look at the types of services provided to the programmer by the operating system and consider briefly its underlying structure.

Then we will discuss the types of protection features designed into typical systems of that generation by both the hardware and operating system designers. We shall emphasize hardware features

because a sound hardware base is essential to the implementation of effective and efficient information access controls. We shall see that hardware is not the problem, that the types of hardware features we consider important are presently available with most contemporary computer systems. (As you will discover during the remainder of the lectures, the problem is really one of designing and implementing reliable operating systems software.)

After discussing typical protection features of each generation, we will see how requirements for the processing of classified information were satisfied.

Finally, we will note how protection problems not solved by a particular generation influenced the nature of hardware and software protection features built into systems of the succeeding generation.



The first computer generation began with the invention of the electronic digital computer just after the second world war and lasted until around 1960. The technology of these machines was relays and vacuum tubes. The ENIAC, EDVAC, and UNIVAC I were early members of the first generation. The IBM 704 (scientific) and IBM 705 (commercial) introduced core memories, with capacities on the order of 16 and 32 K words. The I/O devices were card reader/punch, printers, and magnetic tape for secondary storage.

First generation computers had no operating systems as we now know them. They were essentially single user systems -- no resources were shared. The programmer programmed in symbolic assembly language or FORTRAN, using simple bootstrap software to load the assembler or compiler into the machine to assemble or compile his program, later using the same bootstrap to load his machine code for execution. The programmer was also the computer operator.

First Generation Protection

No sharing of resources

User could destroy bootstrap software

No protection features

First generation systems were not resource sharing systems. They were used by only one programmer at a time, who had access to the entire system, all its resources, until finished or until his allotted time had expired, whereupon a new programmer had the system. Under this environment the only protection problem was that of trying not to destroy the bootstrap software. But this happened quite often as programs were being debugged (and even after as they were used on a regular basis); it was a well accepted fact of life. The programmer would just have to reload the bootstrap, usually by first manually entering (through console switches) a very simple bootstrap loader which would subsequently be used to load a more useful punched card or magnetic tape loader.

There were, however, no protection features designed into either the hardware or the bootstrap software.

First Generation Security

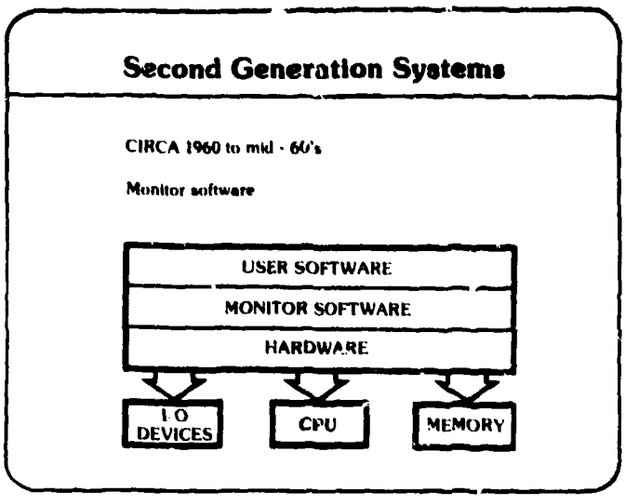
User had access to all physical resources

System protected at highest level

Most computation done at single level

First generation computers were primarily used for number crunching. If there were requirements for classified computations, the system would be operated in the system high mode. The system was located in a facility secured to the appropriate level and all programmers were cleared to that level.

There was no provision at that time for a periods processing mode of operation; most computations were done at the single, system high level. If there was a need for unclassified processing -- the bowling scores, for example -- these were treated as classified and the output -- the scores -- had to undergo a downgrading procedure before they could be removed from the facility and posted in a hallway in an uncleared area.



The second generation started shortly before 1960 and lasted only about until 1964, with IBM's intorduction of System/360. It was a transitional period in many respects.

Computers of this generation were constructed out of discrete solid state components -- transistors and resistors -- and were smaller, faster, cheaper, and more reliable than their vacuum tube predecessors. Representatives of this generation were the IBM 7000 series -- characterized by the 7090 (scientific) and 7070 (commercial) -- the IBM 1400 series -- with the 1401 (commercial) most prominent -- and the Burroughs B5000 series.

Main memories were still magnetic core based, but were double the capacity of first generation main memories, and memory access speeds improved almost by an order of magnitude.

Punched cards and magnetic tapes were still the primary peripheral storage media, however, random access magnetic disks and magnetic drums were first introduced on some second generation systems. These disks and drums were controlled by an independent I/O channel -- a small processing unit -- operating in parallel with the central processor. It wasn't until the third generation, though, when the performance advantages inherent in simultaneous operation of the central processor and I/O processors were fully exploited.

The second generation saw the advent of operating system software in the form of a collection of system routines commonly

called a monitor. The monitor was a set of useful machine services which enhanced and extended the machine architecture and provided a more useful and productive programming interface. A significant second generation development was the distinction between systems programmers, who wrote and maintained monitor routines, and application programmers, who developed application systems using the monitor. These application systems also utilized monitor facilities at run time.

Typical of second generation operating systems was the Fortran Monitor System of the IBM 7090, which provided the programmer with facilities for the compilation, debugging, and execution of Fortran programs.

Second Generation Systems

Notion of "logical" I/O devices

Memory space overlayed

Library functions

User at a time

The beginnings of time sharing/information processing

Some of the more notable services provided by these monitors were logical I/O devices, memory overlaying, and libraries of commonly used routines.

The notion of logical I/O devices was an important development. The user no longer had to worry about dealing directly with physical devices. The monitor provided a number of logical devices and operations for the manipulation of them. The monitor performed the mapping of logical device accesses to physical device accesses transparently to the user.

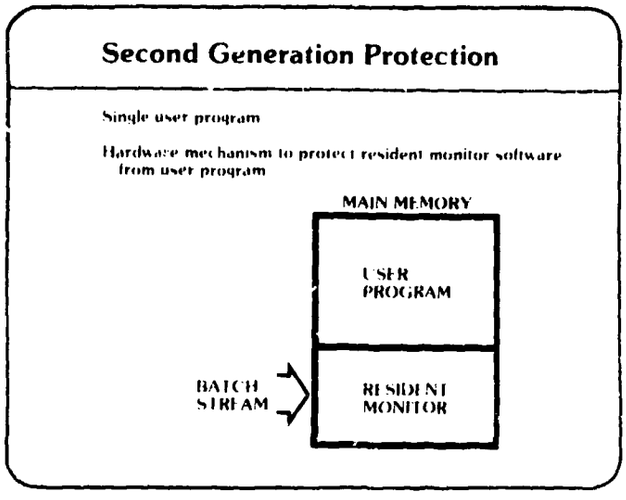
Another feature of these monitors was support for overlaying of main memory. Memory capacity was on the order of 64K words, some of which was reserved for the monitor. User applications were expanding and pressing the available memory capacity, so the idea of overlaying was conceived whereby the user program was partitioned into pieces, one piece resident and executing in main memory, the other pieces residing on auxiliary storage. When the first piece finished executing, the monitor would fetch the second piece into main memory, overlaying it on the first piece, and execute it.

The concept of library functions was developed. Examples were mostly useful mathematical routines -- trigonometric function, square root -- which could be referenced in a FORTRAN program; the link editor/loader routines of the monitor would retrieve these routines from the library. Various magnetic tape utilities, such as sort/merge, were also provided.

Second generation systems were single-user-at-a-time batch processing systems. Programmers would submit their batch jobs to an operator who would feed them into the system. However, only one user program would reside in main memory at a time and it would execute until completion. The operator would mount any tapes needed by the user's program.

In the second generation, though, systems programmers were already making some interesting observations about processor utilization. Although this was generally not a problem with CPU-bound scientific programming, it was observed that commercial applications tended to spend much of the time performing I/O operations. Some of this commercial processing tended to be information processing rather than computation.

Also, in the early 60's researchers on Project MAC at MIT began examining the idea of multi-access, interactive processing. Already researchers were recognizing the relative inefficiency of single-user-at-a-time batch processing systems and were searching for alternative modes of processing. The MIT group developed the Compatible Time Sharing System (CTSS), supporting three interactive users and running the Fortran Monitor System in the background, on an IBM 7090 modified to support multiprogramming, a technique we will see universally employed on third generation systems.



Second generation systems were still one user at a time. At most only one user program resided in main memory at a time and it executed until completion. The only protection problem was that of protecting the resident monitor programs from the resident user program. The special hardware feature was merely a fixed protected area at the bottom end of main memory which contained the monitor. Let us say that the monitor resided in main memory starting at location 40000. Now, whenever the processor fetched an instruction from a location greater than 40000, it knew that monitor software was executing and it would allow any location in main memory to be read or written by that instruction. However, whenever the processor fetched an instruction from a location less than 40000, it knew that a user program was executing and would allow only locations less than 40000 to be read, written, or branched to by that instruction. The one exception was that the user program could branch to location 40000 only. This was the mechanism by which the user program could "trap" to the monitor and request some particular service.

Some systems permitted only the monitor to execute I/O instructions; that is, the processor would perform an I/O instruction only if it had been fetched from location 40000 or greater. This forced the user to trap to the monitor for I/O and was the hardware protection mechanism by which the monitor could protect the library programs (stored on system tape or disk) from accidental or malicious destruction.

We shall see this notion of a "two state" processor -- a privileged state and unprivileged state -- extended somewhat in the third generation to address protection problems in multiprogramming computing system.

Second Generation Security

User had access to most resources

System, users at highest level

Some manual review for lower classified runs

With respect to the processing of classified information, there was very little difference from that of the first generation. System high was the prevalent mode of operation. The output of any lower classified jobs had to be reviewed and downgraded to its true level of classification.

Third Generation Systems

General purpose computing systems

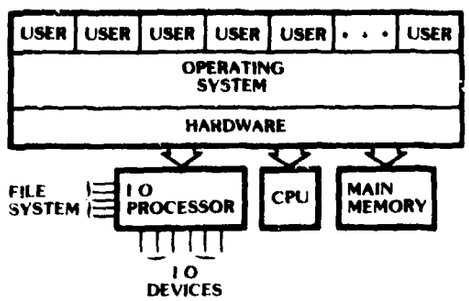
Common architecture over several models

Integration of autonomous I O processors

Main concern of operating system: maximum resource utilization

Multiprogramming

Third Generation Systems



The third generation is widely regarded as having begun in 1964 with IBM's introduction of System/360. Honeywell's 600 series is also characteristic of the third generation. The technology of the third generation was solid state circuitry, offering improvements in speed, capacity, and reliability, and permitting even smaller packaging. Memories, still of core memory technology, began to reach capacities of 128K words and larger.

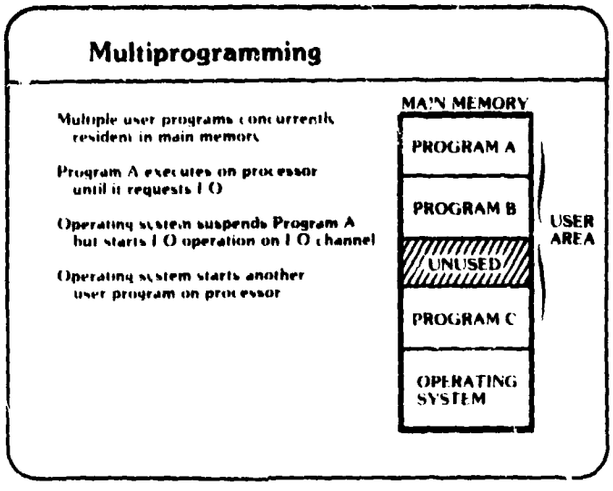
The major third generation systems were designed to be general purpose computing systems. The manufacturers decided to move away from the notion of one machine model for scientific processing and another for commercial processing. Instead they wanted an architecture which would support both numerical computation and information processing. Support for data management services was recognized as a very desirable and useful feature.

Another distinguishing characteristic of third generation systems was that they provided a common architecture over various models ranging in performance and capacity. Models at the lower end of the range presented the same architecture as did models at the upper end. The different models were instruction set compatible.

The third generation saw full employment of independent I/O processors (or channels) operating in parallel with the central processor. A significant factor influencing the third generation was the rapidly escalating demand for computer usage. As the basic technology improved, computers were becoming cheaper and a computer

was becoming essential to more and more businesses and government agencies. Even so, a computer represented a significant investment and it was important that it be utilized as much as possible. Fortunately more and more applications were being identified as candidates for computerization.

The primary concern of the third generation operating system was maximizing utilization of the computer's various resources. The goal was to keep the central processor and I/O processor as busy as possible. The technique employed by third generation operating systems to achieve maximum resource utilization was known as <u>multiprogramming</u>.



The idea behind multiprogramming is that the operating system keeps more than one user program resident in main memory at a time. One user program runs on the central processor until it terminates, exhausts its allotment of central processor time, or requires I/O, whereupon it makes an I/O request to the operating system. The operating system suspends that user program from running, initiates the I/O request on one of the I/O processors, and gives the central processor to one of the other resident user programs (which runs until it terminates, needs I/O, or runs out of time).

Multiprogramming is the technique by which the operating system shares the resources of the computing system among several resident user programs. It is this concurrent sharing of resources that distinguishes the third generation from earlier generations and introduces a number of new protection problems.

Third Generation Services

Resource allocation
CPU scheduling
I/O device scheduling
Memory space

File system
Virtualization of file resources

Program Library
Languages
Subroutine packages

The operating system became a major component of the third generation computing system. It carried the significant responsibility of seeing to it that the resources of the system were maximally utilized.

Most third generation computer systems were batch systems, receiving a stream of user jobs and processing several of these at a time. Several user programs are resident in main storage at a time. The task of the operating system is to multiplex the system's main resources -- central processor, main memory, and I/O channels - among the resident jobs in the most efficient manner.

During the third generation, the advantages of on-line storage for programs and data files -- much more efficient than off-line cards or magnetic tape -- were well recognized. Therefore some centralized means of organizing and storing user information, and managing the retreival of this information, was delegated to the operating system. This mechanism is known as a file system.

A wider variety of programming languages were supported by these general purpose systems. Compilers for Fortran and Cobol were available; IBM tried to be all things to all men with PL/I. The notion of on-line libraries containing often used routines and useful utility programs, introduced during the second generation, was greatly expanded during the third generation.

Third Generation Protection

Mechanisms needed to support user multiprogramming safely

User programs vs user program

User programs vs operating system

User programs vs system resources

As already noted, the introduction of multiprogramming presented both the hardware and operating system designers with some serious protection problems. Mechanisms were needed to support multiprogramming safely. With several user programs concurrently resident in main memory, mechanisms were needed to protect one user program from interference by another, and to protect resident operating system software from interference by user programs. Thirdly, mechanisms were needed to protect the various resources managed by the operating system -- the file system, the program libraries -- from interference by user programs.

The first two protection concerns are addressed by features providing memory protection -- the ability to protect areas or partitions of main memory from unwarranted access by user programs. The third protection concern is a matter of controlling a given program's access to certain central processor instructions and to I/O devices.

Third Generation Protection

Memory protection

Lock key

Base and bounds registers

CPU and I/O protection

Two state processor

Privileged instructions

Supervisor/user domains

Let us examine some of the hardware features built into third generation computing systems to provide memory, central processor, and I/O device protection.

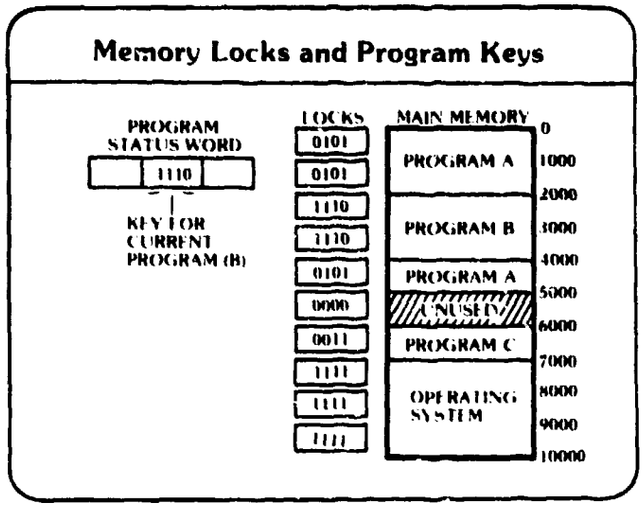
One of two mechanisms was employed to protect one resident user program from destroying another resident user program, and to protect the operating system resident software from resident user programs: lock and key, or base and bounds registers.

(See the next two view graphs entitled "MEMORY LOCKS AND PROGRAM KEYS" and "BASE AND BOUNDS REGISTERS".)

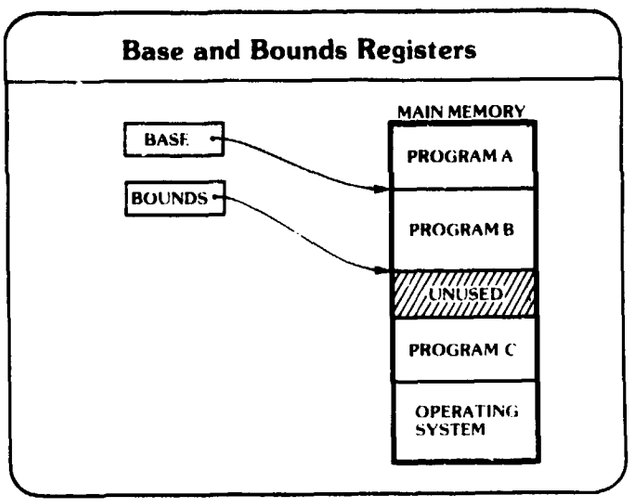
The most significant mechanism was that of the two-state central processor, meaning that the processor could operate in one of two states: privileged or unprivileged. Certain of the processors instructions are designated as privileged, meaning they can only be executed when the processor is operating in privileged state. Example of privileged instructions are those that perform I/O, control the interrupt mechanism, and set base and bounds registers or locks and keys.

The two processor states create two domains of execution, which have been called supervisor and user domains. User programs execute in user domain, with the processor in unprivileged mode, meaning that any attempt to execute one of the privileged instructions is trapped by the processor. Also, the lock and key, or base and bounds, memory protection mechanism, whichever is used, is enforced on memory accesses.

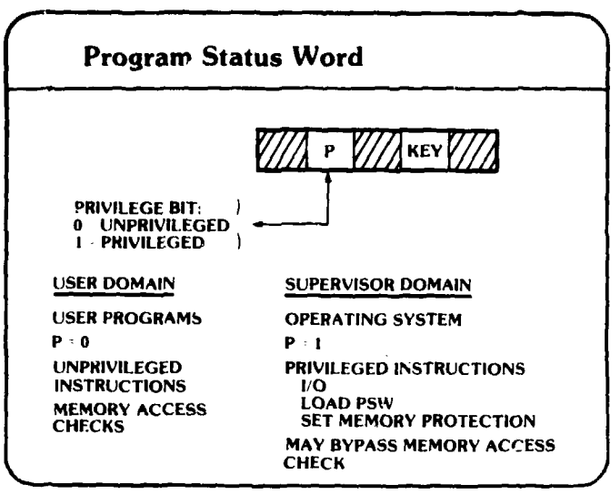
The operating system executes in supervisor domain, with the processor in privileged mode, able to execute the privileged instructions. Typically any memory protection mechanisms are disabled, meaning any memory location is accessible from supervisor domain. Only the operating system has uncontrolled access to all of the system's resources; user programs have indirect access to system resources.



With the lock and key mechanism, each location in main memory carries with it a lock -- for example, a 4-bit integer, which can be set by the operating system -- and each resident user program is assigned a key -- for example, another 4-bit integer. When the operating system assigns a user program an area in main memory, it sets the locks of the locations in that area to contain the 4-bit pattern assigned as key to the program. Now, whenever a program accesses main memory, the processor checks to see that the program's key matches the value stored in the lock associated with the area accessed.



With the base and bounds mechanism, the processor contains a pair of registers called a base register and a bounds register. The base register contains the starting location in main memory of the currently executing user program; and the bounds register contains the ending location of the user's program in main memory. On each memory reference made by the currently executing user program, the processor checks that the reference is within the area in main memory defined by the base and bounds register. When a different user program is given the processor, the base and bounds register are changed, by the operating system, to describe the new program's memory area.



Third generation systems were multiprogramming systems: meaning more than one user program was resident in main memory at a time and the central processor was multiplexed among these resident user programs (and the operating system).

Therefore, the hardware protection mechanisms centered about protecting one program from another (and protecting the system from user programs).

The key hardware feature was the program status word (or PSW), a hardware register within the central processor. It defined the characteristics of the currently executing program. One bit in the PSW indicated the particular state in which the processor was executing -- privileged or unprivileged -- and thereby indicated the domain in which the program was executing.

If the executing program was a user program, a field within the PSW held the (4-bit or so) key assigned to the program to govern its legal memory area.

Another field within the PSW was the interrupt mask. Depending upon the setting of this field, certain I/O devices were prevented from interrupting the central processor.

It is important to note that the PSW can only be loaded or modified by software executing in supervisor domain. The machine instructions to load or modify the PSW are privileged instructions.

By permitting only operating system software to run in supervisor domain, the operating system can control the use of system resources.

As noted, user software runs in user domain. Transfer into supervisor domain, and privileged processor mode of execution, is also hardware controlled. User programs invoke the operating system by executing a trap (or supervisor call) instruction, which is unprivileged. The effect of the trap instruction is to change the mode of the processor to privileged and to commence execution at a predefined memory location containing operating system code.

Transfer to the operating system occurs asynchronously by means of an I/O interrupt. When a device raises its interrupt flag, the central processor recognizes this and automatically sets the processor mode to privileged and initiates an interrupt handler at a predefined location within the operating system region of main memory.

Third Generation Protection Features

Software features

Checks legality of supervisor calls

Performs I/O processing

Provides privilege protection (passwords)

Audit mechanisms

Software mechanisms were included within the operating system itself to augment the protection mechanisms provided by the hardware.

Some systems attempted to verify the legality of supervisor calls issued via the trap instruction from user domain. In particular, some attention was given to checking the validity of parameters supplied with the system call. The operating system would check the number of parameters supplied and assure that each parameter was of the proper type. For example, the operating system might wish to check that a parameter defining the starting address of an I/O buffer was indeed within the memory area allocated to the program making the request.

Users had little facility for performing I/O directly. Rather, I/O was provided as a service by the operating system. This was good from a protection standpoint because the misuse of an I/O channel by a user program could destroy other resident user programs or information stored on shared I/O devices.

Most third generation systems employed passwords to control access to files within the file system. The owner of a file would assign a password to the file upon its creation. Anyone wishing to access that file in a job step would have to submit its password on one of the control cards for the job step. The system would verify correctness of the password before granting the user access. The owner could thus share the file with fellow programmers by informing them of the file's whereabouts and its password.

Most systems incorporated an audit log mechanism to maintain a record of user operations and system resource usage (information essential to billing users for system usage). Of importance to information protection were audit log entries resulting from file system usage. For example, the system would create an entry whenever a file was read or modified. The entry would indicate the user accessing the file, the type of access, and time of access. File access audit information was generally made available to the owner of a file so that the owner could determine whether some unauthorized user was using the file (by illegally obtaining or guessing its password).

Third Generation Security

Users and information at different levels

Concurrent multilevel use desirable

Requires effective operating system access control mechanisms

Operating systems were unreliable

Had to revert to traditional techniques

System high operation

Periods processing

There was an increasing demand for the services provided by third generation computer systems. More and more applications were being considered for implementation on a computer. Many of these potential applications had requirements for the concurrent processing of information at multiple classification levels, and some required the servicing of users of differing levels of clearance.

As we have noted, in order to satisfy these multilevel requirements the operating systems must support effective access control mechanisms to guarantee the separation of multilevel information. The computer system must be able to thwart attempts by malicious users to gain access to information for which they do not possess sufficient clearance.

It was evident, though, that third generation operating systems were too unreliable and they could not be trusted to effectively protect information. They were just not constructed with security as a primary design goal.

So it was not possible to satisfy the demand for true multilevel processing. Instead system architects had to revert to the traditional techniques of either system high operation, with its disadvantage of proliferating classified information, or periods processing, with its inefficient utilization of system resources.

Third Generation Problems

Hardware and software features were not effective

No well conceived design and development strategy

Resulting systems were large and complex

The hardware and software protection features of third generation operating systems were ineffective because of the unreliable nature of the operating system software. There were both design and implementation bugs in the software which could be exploited by the knowledgeable, malicious user to subvert the protection features. After all, it is the responsibility of the operating system to utilize the underlying hardware protection features correctly; if it is possible to subvert the operating system it is possible to subvert the hardware protection features.

The operating systems were so unreliable because they were not developed through a well conceived design and implementation strategy. The term "design by committee" or "ad hoc design" were unfortunately so characteristic of those systems. It is fair to say that the primary objective of third generation operating system designers and implementators was simply to get the system working, to keep the resources utilized, and maximize throughput. The responsibilities of the third generation operating system were an order of magnitude greater than its second generation predecessor. The resulting operating systems were very large in size, written mostly in assembly language for efficiency, and consequently very complex. (I'm sure you have all heard stories about the hundreds of people involved in designing and building OS/360, and its hundreds of thousand of assembly language instructions, and its near constant number of one thousand bugs throughout its many releases!)

Third Generation Penetrations

OS/360: MITRE

GCOS: Government

OS/VS: SDC

As evidence of the protection weaknesses of third generation operating systems, a number of them have been successfully penetrated and documentation of these penetrations is in the public domain. Let us briefly discuss three of them.

A couple of people at MITRE were able to penetrate OS/360, specifically the MVT (or Multiprogramming with Variable number of Tasks) operating system. A particular Air Force organization made an attempt at achieving a multilevel mode of processing on OS/360 MVT. They devised a security software package which they felt would permit the processing of classified data while both cleared and uncleared user programs were concurrently resident in main memory. The package would be used by people running classified jobs and what it did was to destroy all I/O buffers and the program's area in main memory upon termination of the classified job, so that subsequent (probably uncleared) user programs could not "scavenge" any of the classified information. The classified jobs were constrained to use only tapes for their classified data to avoid having to rely on the protection mechanisms of MVT's disk access software.

This approach sounds pretty good at first glance, but note that it is an attempt to provide security through an "add-on" application package. Because the MITRE penetrators were able to penetrate MVT itself, upon which the package ran, they were able to circumvent the package's security provisions.

The security package was located on the system disk and the penetrators were able to replace the package with their own version, which continued to destroy the buffers and working areas of classified jobs, but before doing so wrote a copy of the classified information onto a disk file owned by the penetrators.

The penetrators were able to replace the security package because they were able to "take over the system." While examining the program listings of MVT, they noticed that the software mechanism controlling the execution of MVT programs in supervisor domain were too dispersed; MVT did a lot of branching among routines in supervisor domain and they found it relatively easy to fool MVT into either a) returning to the user program after completion of a supervisor call without resetting the processor mode to unprivileged, or b) simply branching directly to user program while still in privileged mode. The net effect in either case is to return to the user program in privileged mode. The user program <u>is now the operating system</u>.

The penetrators also found a significant I/O design flaw. MVT would store I/O control blocks in the user program area, unprotected from the user program. A knowledgeable user could write his program to exploit this, modifying the control information set up by MVT so that the user could effectively read and write memory areas not belonging to him. The user could read information belonging to other users or, more ominously, write areas of memory reserved for the operating system. The latter would permit the user to "rewrite" portions of the operating system, inserting so-called "trap doors" into the operating system which could be exploited only by that user.

The Defense Intelligence Agency (DIA) was able to penetrate the GCOS operating system running on the Honeywell 635. As do most penetration teams, the DIA group found not one but a number of flaws; one relates to the checkpoint/restart feature of GCOS.

Checkpoint/restart is a useful program debugging tool. The idea is that a user program executes until it encounters a checkpoint, at which time GCOS is invoked to dump the user program region onto a disk file. GCOS would dump both user data and control information onto this file. GCOS would make this dump file available to the user; the user could read and modify this dump file and then request GCOS to restart the suspended program. GCOS would read in the dump file and restart the program.

One of the pieces of control information stored on the dump file is the program status word. Incredibly, the user could modify the PSW, setting the processor mode to privileged, before restarting

the program. The user could exploit this design flaw to take over the system.

The OS/VS operating system running on IBM 360 was penetrated by a combined group from IBM and System Development Corporation. OS/VS evolved from MVT and unfortunately inherited many of its flaws.

One of the penetrations perpetrated by IBM/SDC exploited flaws in the handling of I/O channel programs. OS/VS permitted user programs to write their own channel programs. However, the user program had to invoke OS/VS in order to run the channel program and OS/VS would endeavor to check the validity of the channnel program before running it. OS/VS would check to see that the user was only doing I/O to areas of memory assigned to the user program. Unfortunately, there were a number of schemes whereby the user could write a self-modifying channel program, particularly when the channel program consisted of a number of channel commands chained together. The channel program, once blessed by OS/VS, resided in the user's area of main memory. One of the early channel commands would read in a new channel program -- unchecked by OS/VS -- which would overwrite later channel commands which had been checked by OS/VS. These new commands could read or write memory areas not belonging to the user.

It is probably safe to say that there were few, if any, unsuccessful penetration efforts of third generation systems when the penetration team was knowledgeable and determined. It is also safe to say that most penetration efforts found not one but several exploitable flaws, both design and implementation flaws.

Current Generation Systems

Interactive, multiple access, timesharing

Virtual memory architectures

Examples

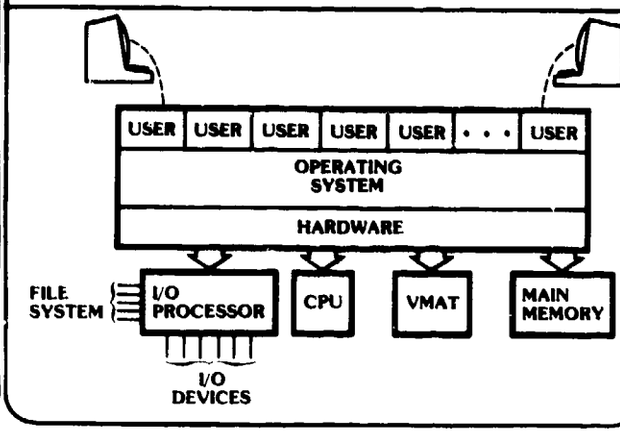
MULTICS

TENEX

VM/370

VAX/VMS

Current Generation Systems



Let us now look at current generation computing systems. It is probably reasonable to label these systems fourth generation systems, but I haven't seen that term used to refer to modern systems so I prefer to call them current generation systems.

Perhaps the most distinguishing characteristic of the current generation relates to mode of usage. Current generation systems are interactive systems where many users access the system concurrently using terminal devices. Instead of submitting a deck of punched cards containing a series of job steps, as was the pattern of use during the third generation, current generation users sit at a terminal and issue commands to the computer using the terminal's keyboard.

The technique employed by current generation operating systems to support multiple concurrent user access is called timesharing. The operating system allocates the central processor to user programs for very short periods of time (e.g., 100 milliseconds), usually in a simple round-robin fashion, in an attempt to distribute processor time evenly among all users desiring service. Since each user is guaranteed a slice of processor time, say, every three or four seconds, the user is given the illusion of having the whole machine to himself.

If the operating system is to timeshare efficiently the processor among multiple user programs, as many user programs as possible should be concurrently resident in main memory. In the ideal case,

all user programs would be memory resident and timesharing would be a simple matter of suspending one user program -- after it exhausted its allotted time-slice -- and starting another. Unfortunately main memories are just not large enough to accommodate, all at once, the programs of all users desiring service, particularly if each user's entire program must reside in main memory, as was the case on third generation systems.

A distinguishing architectural feature of current generation hardware is virtual memory. Virtual memory permits more flexible operating system allocation of main memory to user programs, making the operating system better equipped to support interactive access by multiple users through timesharing. The important characteristic about virtual memory architectures in this regard is that the address space of a user program is partitioned into a set of independently allocated units, some of which are main memory resident when the user program is executing, and some of which are not. Because only the more active (i.e, most recently executed) units of a user program are resident, and not the entire program (as was the case on third generation systems), the operating system can fit a greater number of user programs in memory at once, and hence a greater number of users can be serviced efficiently through timesharing.

The characteristics of virtual memory are treated in greater detail later.

Here are some examples of current generation systems. MULTICS is a system developed jointly by MIT, Bell Laboratories, and Honeywell. This group was one of the first to recognize the information protection problems inherent in interactive, multiple access, timesharing systems. The MULTICS operating system was first implemented on a Honeywell 645. Later, it was reimplemented on a Honeywell 6180. Many of the protection features specific to the MULTICS philosophy of information sharing and protection were implemented in software on the 645, but migrated into the hardware of the 6180.

TENEX is an operating system developed by Bolt, Beranek, and Newman for the PDP-10 line of DEC computers. The first version of TENEX was implemented on a third generation KA-10 processor enhanced by BBN with a hardware device which provided a paged virtual memory. TENEX has since been reimplemented on the KI-10 and KL-10 processors, both incorporating a paged virtual memory as a standard feature.

VM 370 is a virtual machine operating system developed by IBM for those IBM 370 models which include a

paged virtual memory. VM 370 provides a set of virtual machines, each one looking like a complete IBM 370 as described in the Principles of Operation of System/370. Each virtual machine runs its own IBM 370 operating system and supports a set of users. VM 370 isolates one virtual machine and its set of users from the other virtual machines and their users.

VAX VMS is the operating system developed by DEC for its VAX-11/780. The VAX machine is a segmented-paged virtual memory architecture.

Current Generation Operating Systems

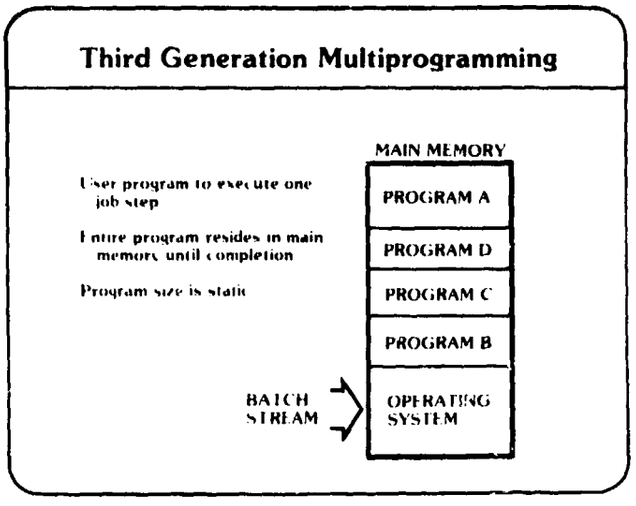
Two influences

Third generation multiprogramming model inappropriate for fourth generation

Operating system design emphasized

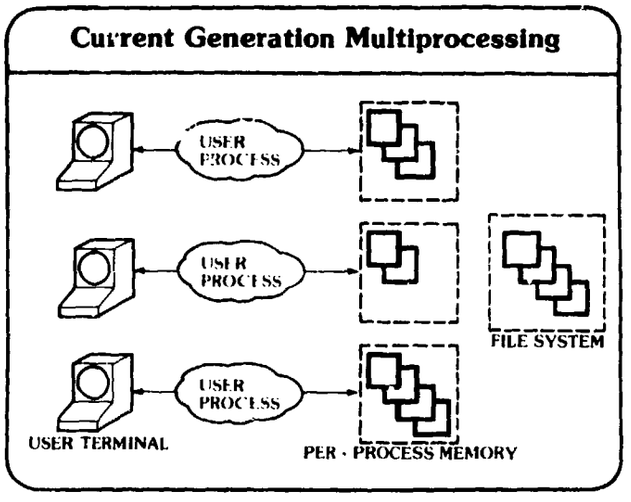
Two major factors shaped the development of current generation operating systems. First, operating system designers recognized that the multiprogramming model employed during the third generation was inappropriate for satisfying the response requirements of interactive processing. The concept of the process was developed as the computational entity to support the interactive user's processing requirements. The operating system's provision of multiple concurrent processes to support multiple interactive users is known as multiprocessing.

Secondly, the designers were well aware of the unreliability of third generation operating systems. They knew how easy it was to penetrate them and subvert whatever hardware and software protection mechanisms existed. They realized that the crux of the problem was the almost ad hoc manner in which third generation systems were developed. So, they began to consider better strategies for designing, implementing, and testing current generation operating systems. They learned a lot about the inadequacies of third generation software engineering and also about the flaws exploited to penetrate third generation systems.



Let us consider further the multiprogramming model of the third generation. As already noted, users submitted batch jobs on card decks. Each user job consisted of a number of job steps; for example, a typical job might consist of a compilation step, a link edit step, and a load and execute step. The operating system would read in a job and perform each job step sequentially. For the compilation step, the operating system allocated an area in main memory in which the compiler executed as a user program. This compilation program resided in main memory and executed until completion. For the linkage editing step, the operating system again allocated an area in main memory (perhaps the same area) in which the linkage editor ran as a user program until completion. Finally, the operating system performed the load and execution step by again allocating a main memory area, loading in the link edited user program, and executing it.

The important points about this model is that each job step ran independently as a user program which resided entirely in main memory and executed until completion, abnormal termination, or exhaustion of its allocated processor time, and that the main memory allocated to the user program was fixed in size during program execution.



Now, in the current generation, instead of having just one (or a few) batch job stream(s), the operating system must handle many job streams concurrently, one stream for each user logged on the system. Analogous to the individual steps of a batch job, the user issues a series of commands to the operating system. Each command is just like the job step of the previous generation. Now, however, the user at his terminal must be serviced responsively by the system. Acceptable "turn-around" or response time is now measured in seconds instead of minutes.

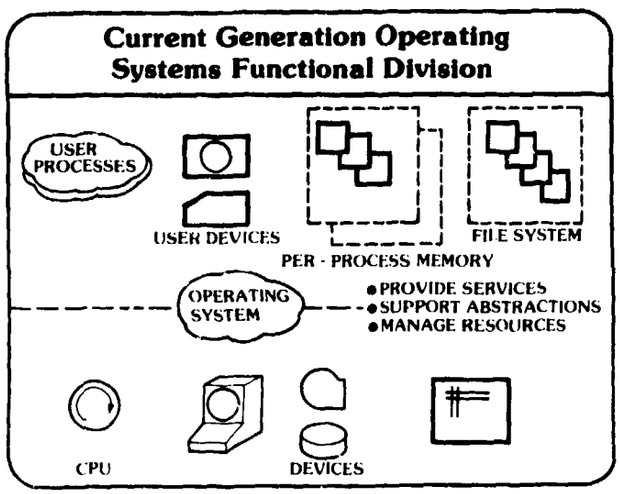
Hardware and software designers developed the notion of the process to service each user. The process is a computational entity, an environment in which the individual programs requested by the user are executed. The operating system supports these individual processing environments, providing them as needed to users of the system.

Current generation systems support a number of user processes concurrently and are therefore called multiprocessing systems. The multiprogramming model of the third generation has evolved into the multiprocessing model of the current generation to better support interactive processing by multiple concurrent users. Current generation systems use virtual memory addressing techniques to squeeze as many user processes as possible into main memory to achieve fast response to user requests.

The process is a virtual processing environment. It consists of a virtual address space of some fixed maximum size. On most current generation systems, the virtual address space of a user's process may contain many programs simultaneously. This is true of systems such as MULTICS, TENEX, and UNIX. Included in the virtual address space of each process is the operating system. (This doesn't mean that there are many copies of the operating systems, one for each process; rather, only one physical copy exists and is shared by all of the processes; each process has its own virtual copy of the operating system.)

Since each process includes a virtual address space and an operating system -- which provides a file system, I/O devices, and other services -- the process represents the user's virtual processing environment or virtual computing system. We say that current generation operating systems provide a <u>per-process virtual environment</u>.

The operating system should be capable, as a matter of course, of isolating and protecting processes from each other, while permitting the sharing of programs and information among processes which desire to cooperate.



As has already been stated, the other major factor shaping current generation operating systems is the emphasis on operating system design. Designers realized that, if their operating systems were to become more reliable, they had to adopt a better design strategy.

One of the first things they came to realize is that an operating system is not a big monolith -- the operating system. Rather, it was possible to distinguish the various functions performed by the operating system.

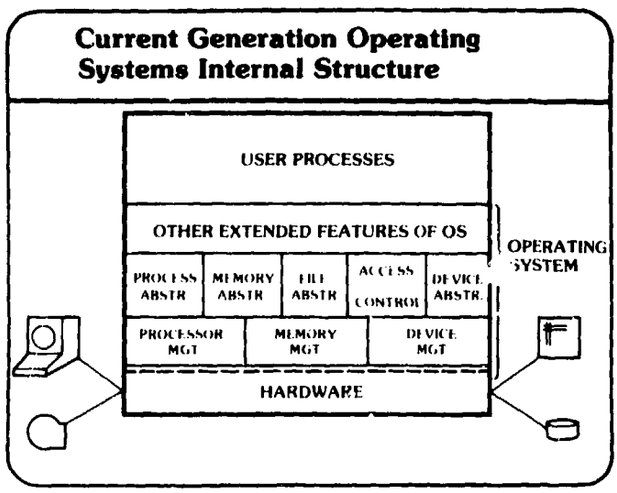
This slide depicts a three-part functional division.

One thing that an operating system does is provide services. It makes available to users such things as compilers, text editors, data base management services, loaders, debuggers, and mail services. In the context of the current generation, the operating system, at the user's request, moves the programs and data which constitute a particular service into the user's process virtual environment for execution.

Next, the operating system implements a number of abstractions. The most significant is the process abstraction, which we have explained above. The process itself is constructed out of a number of other abstractions which include 1) user or logical I/O devices, 2) the per-process virtual memory, and 3) the file system.

The operating system implements these various abstractions out of the underlying physical resources of the computing system. Processes are run, in some order, upon the central processor. References to logical I/O devices are mapped into references to physical I/O devices. The file system is constructed on (typically) random access I/O devices, such as magnetic disks. And the per-process virtual memory is mapped, with the assistance of some virtual address translation hardware, onto main memory and some fast, random access backing store, such as magnetic drum or fixed-head disk.

The operating system must incorporate various resource allocation and management strategies in order to support the abstraction-resource mapping in an efficient manner.



Some designers translated the functional division into an internal structuring, or layering of the operating system. It was during this period that the idea of structuring the internal design into a hierarchy of modules or layers was first conceived. Modules at the lowest layers of the hierarchy are concerned with the allocation and management of the physical resources. This slide shows three modules in the bottom layer: processor management, memory management, and I/O device management.

The next layer in the hierarchy consists of modules which construct abstractions out of resources and export these abstractions to users of the system. The slide depicts four abstraction modules: process, memory, file, and device. There is an additional module at this level, the access control module. Ideally, such a module would exist at this level and serve as the sole mechanism in charge of system-wide access control. All references by user processes to memory, file, and I/O device abstractions would be monitored by this module, which would determine whether a given reference is to be allowed.

These two layers together have been called the kernel of the operating system. The kernel is the critical core of the operating system and is that part which should be emphasized in order to achieve reliable information protection.

The highest layer of the operating system consists of modules which provide the extended features of the operating system. Some

of these features are constructed out of the basic abstractions of the operating system; these include mail facilities, record management or data management facilities, network access facilities, line printer spoolers, and command language interpreters.

Current Generation Protection Problems

User process vs. user process

User process vs. operating system

User process vs. system resources

The protection problems faced by current generation operating system designers were more complex than those faced during the third generation because the process is a more complex entity than the program. The process may consist of a number of programs. Whereas the third generation program was static in size, the process is dynamic -- its memory and resource requirements varying as programs are moved in and out of the per process virtual environment.

The problem now is the incorporation of hardware and software mechanisms to support multiprocessing safely. The problem has aspects similar to those encountered in the third generation, yet more complex.

Mechanisms are needed to protect one user process from another. The operating system must supply per process virtual environments which, on the one hand, isolate processes as a matter of course yet permit cooperation or controlled sharing of information between or among processes when desired. This problem is one of providing memory protection and inter-process communication features.

Mechanisms are needed to protect the operating system software from user processes. Again this is largely a matter of memory protection, setting aside main memory areas for operating system software and controlling access to it by user processes.

Lastly, the resources of the computer system must be protected from user processes. Users should be able to make use of the

system's resources, but only in a manner controlled by the operating system. As we saw in the third generation, this can be accomplished by structuring the hardware operations of the system into privileged and unprivileged operations, the former being used solely by the operating system to control and manage the system resources.

Current Generation Protection Features

Virtual memory

Execution domains

Hierarchical domains

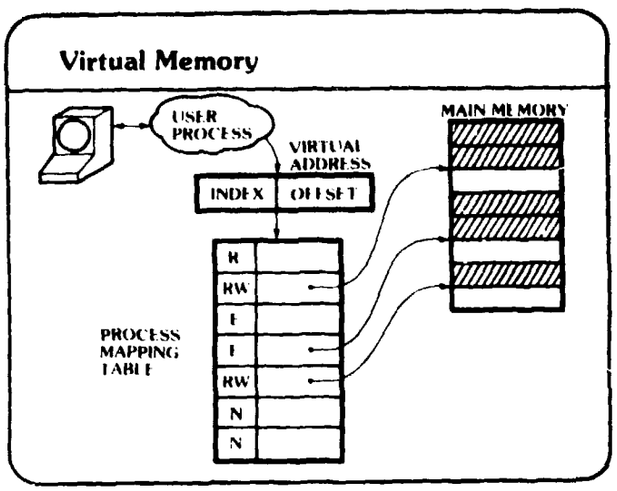
Concentric rings

I/O access controls

Here are the types of hardware mechanisms used to deal with the protection problems of multiprocessing systems.

Virtual memory is both a memory management and memory protection feature, and we saw earlier the importance of a virtual memory architecture to the support of multiprocessing and interactive access by multiple users.

Execution domains, hierarchical execution domains, concentric rings, and I/O access controls are some of the types of hardware mechanisms employed to separate privileged and unprivileged operations.



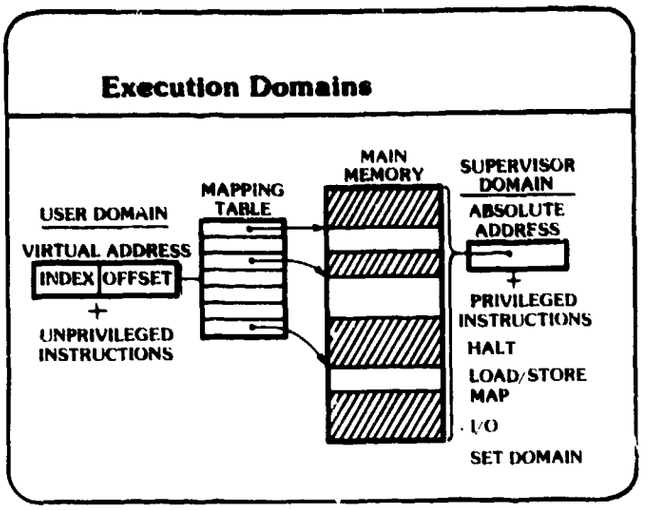
Here is a rather generalized description of virtual memory. It is meant to characterize both true virtual memory systems, such as MULTICS and VAX-11/780, and so-called mapped memory systems like the PDP-11/45 and 11/70.

The virtual address space of a process consists of a collection of either variable sized segments or fixed sized pages. This slide depicts a paged virtual memory system. Descriptors for the pages of a process are collected together into a mapping table. Each descriptor contains information defining both the location of the page and the process' access permissions to the page. A flag within the descriptor typically indicates whether the page is resident in main memory. If the flag is set, the location information is the starting location of the page in main memory; if reset, the starting location of the page on auxiliary memory. In the slide, the second, fourth, and fifth pages of the process' virtual address space are in main memory.

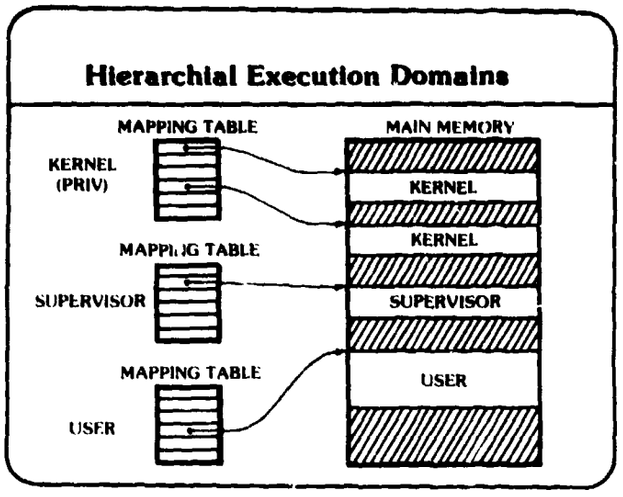
Access permissions are read (R), write (W), execute (E), and null (N). The process has read and write permission to the second and fifth pages, and execute permission to the fourth page. Null access means that no page is described by a particular descriptor; it is used to indicate that a mapping table entry does not contain a value descriptor — an unused page of the process' virtual address space.

Addresses generated by programs within the process are two-component vitual addresses. The first component is an index used by the virtual memory hardware to locate within the mapping table a descriptor for the page to be referenced. The second component is an offset used to locate a specific byte or word within the object. The virtual memory hardware first checks the access permission field to verify that the attempted access is permitted to the process. If it isn't, an access fault is generated by the hardware and the operating system is initiated to take appropriate action. If access is permitted and the page is in main memory, the hardware adds the offset to the starting location of the page to form the effective physical address. If the page is not in main memory, a page fault is generated by the hardware and the operating system is initiated to move the page into main memory off of auxiliary memory.

Because the operating system creates and manages processes, it is the responsibility of the operating system to create and manage the process mapping tables. By doing so in a correct and judicious manner, the operating system can effectively isolate processes where required and allow them to cooperate and share information where desired. Descriptors are added to mapping tables as processes request that pages be brought into their virtual memory. Whether or not the operating system honors the request depends upon the protection policy enforced by the operating system.



We saw in the third generation how execution domain hardware was used to partition operations into privileged and unprivileged ones. This slide shows how a two-domain hierarchy is implemented in a virtual memory architecture. The only distinction is that user software executing in user domain (with the processor in unprivileged mode), is constrained to using virtual addresses which are translated by the virtual memory hardware into physical main memory addresses. The operating system, executing in supervisor domain (with the processor in privileged mode), is privileged, if it so wishes, to use absolute physical memory addresses (and, in some systems, can thereby access any main memory location). All the other characteristics of execution domains remain the same as they were in the third generation.



We have noted how current generation operating system designers recognized that certain elements of the operating system were more critical than others and required greater emphasis in design and implementation in order to achieve overall reliability. Some hardware designers realized that hardware support was needed to isolate and protect these more critical operating system components.

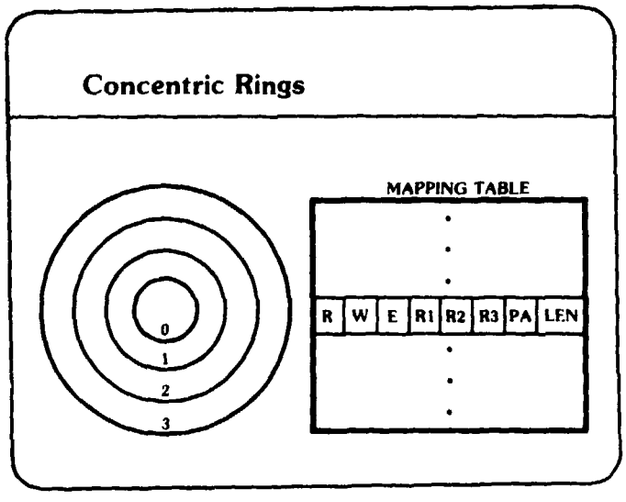
This slide illustrates the concept of hierarchical execution domains, as provided by the PDP-11/45 and 11/70 computer systems. There are three modes of processor operation -- kernel, supervisor, and user -- and a mapping table for each mode of execution. A process therefore consists of three separate address spaces, or three domains of execution. At any point in time during process execution, the particular mode of processor operation determines which of the three mapping tables is used to translate virtual addresses.

The kernel address space contains those critical operating system components which manage the physical machine resources and implement abstractions out of these resources. The supervisor address space contains non-kernel operating system components which provide the extended services. And the user address space contains user software.

The three domains of execution are linearly ordered in terms of privilege. When a process is executing in kernel domain, kernel operating system software is executing and information in all three address spaces is accessible to it. Also, any privileged machine

instructions can only be executed in kernel domain, i.e., when the processor is operating in kernel mode. When a process is executing in supervisor domain, non-kernel operating system software is executing and only the supervisor and user address spaces are accessible. And when a process is executing in user domain, user software is executing and only the user address space is accessible. Trap instructions are used by software in user domain to invoke software in supervisor domain, and by software in supervisor domain to invoke software in kernel domain.

With this hierarchical execution domain mechanism, critical kernel operating system software is protected from non-kernel and user software, and non-kernel operating system software is protected from user software.



Concentric ring hardware architectures are a generalization of hierarchical execution domain architectures. The MULTICS 6180, Honeywell SCOMP, and PRIME 400 and 500 are examples of concentric ring architectures. This slide illustrates the general characteristics of a ring architecture supporting a segmented virtual memory.

Conceptually, rings of execution are arranged concentrically, with ring 0 innermost, most privileged, and most protected. Rings 1, 2, and 3 are peripheral to ring 0 and of decreasing privilege and protection. Ring 0 is analogous to kernel domain in a hierarchical domain architecture, ring 1 is analogous to supervisor domain, and rings 2 and 3 are analogous to user domain. (Ring 2 can be thought of as user domain 1, with greater privilege and protection than ring 3, or user domain 2.) A field within the PSW defines the current ring of execution of the currently executing process.

Segments of the process virtual memory are assigned to particular rings of execution just as segments may be assigned to particular domains in a hierarchical domain architecture. But unlike hierarchical domain architectures, where a segment is assigned to a particular domain by the placement of its descriptor into the mapping table associated with the domain, a segment is assigned to a particular ring by the setting of certain values within ring bracket fields of its descriptor. Furthermore, instead of defining the address space of a process with a mapping table per domain, the address space is defined by a single mapping table.

The value of ring brackets R1, R2, and R3 define the ring(s) of execution from which a segment may be read, written, and executed. In this example, R1 defines the write bracket. A process may write a data segment providing it possesses a descriptor for it in which the write permission bit (W) is set and the current ring of execution of the process is between 0 and the value in R1. Similarly, the combination of R2 and the read permission bit (R) defines the ring(s) of execution from which a data segment may be read. And, in a somewhat different fashion, the combination of R1, R2, and R3, and the execute permission bit (E) defines the ring(s) of execution from which a code segment may be invoked and executed.

PA contains the physical memory address, either in main or auxiliary memory, of the start of the segment, and LEN contains information defining the length of the segment.

On a concentric ring architecture, code and data segments of the kernel components of the operating system would be assigned to ring 0. Each user process would include in its mapping table segment descriptors for these kernel code and data segments. The R1, R2, and R3 values for certain kernel code segments would be set such that they could be called by code segments executing in higher rings using a CALL machine instruction; the CALL instruction would set the ring field in the PSW such that the kernel code segments executed in ring 0. Other kernel code segments might have ring bracket values such that they could only be called by other kernel code segments executing in ring 0. The R1 and R2 values of descriptors for kernel data segments would be set such that they could only be accessed by kernel code segments (i.e, R1 = 0 and R2 = 0); they would be inaccessible to code segments operating in higher rings.

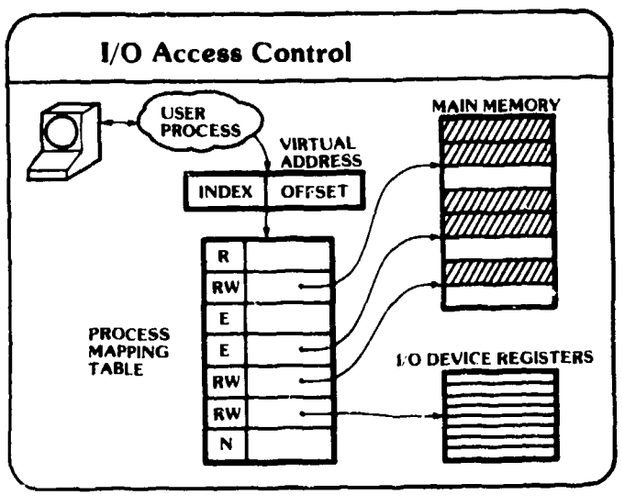
To continue further, code and data segments for non-kernel components of the operating system would be assigned to ring 1, and the mapping table for each user process would include descriptors for these segments. Again, some of these non-kernel segments would be made accessible to user code segments executing in rings 2 and 3, others would be accessible only to code segments operating in ring 1.

Finally, user code and data segments would be assigned to ring 2 or ring 3 depending upon the protection needs of the user.

Note that the access rights of the process tend to increase as the process' ring of execution decreases. Code segments executing in ring 0 — the kernel of the operating system — have access to all segments of the address space, whereas code segments executing in, say, ring 2 can only access segments assigned to rings 2 and 3. Certain ring 0 and 1 segments <u>may</u> be accessible to these ring 2 code

segments, however, depending on the protection concerns of the designers.

Any privileged machine instructions would be executable only by ring 0 code segments.



We have already discussed how most hardware designers have defined I/O operations to be privileged machine instructions, thereby permitting only the operating system access to I/O devices. Here is a mechanism whereby specific I/O devices can be made accessible to user programs and manipulated using unprivileged machine instructions. It is a mechanism provided by the PDP-11/45 and 11/70 computer systems and is called memory mapped I/O.

The idea is that the control, status, and data registers used to manipulate an I/O device are addressed just like memory locations. I/O device registers are assigned bus addresses as are memory locations. The status and data registers of a device can be read using a load instruction; control and data registers can be written using a store instruction.

The significance of this feature is that an I/O device can be included within the virtual address space of a process. A descriptor defining the control, status, and data registers for the device can be added to the process mapping table. In this slide, the sixth page descriptor maps to the device registers of an I/O device. The process has read and write access to the device registers and can read and write the I/O device by loading and storing, say, the first, second, and third words of the sixth page of its virtual address space.

Note that without some additional control mechanism only programmed I/O devices (e.g., terminals) can be made accessible to user

processes. It would be unwise to give a process access to a Direct Memory Access (DMA) I/O device, or an I/O channel, because the process might instruct the device to read or write memory areas not belonging to the process. One mechanism which would permit user processes to access DMA devices would be to have these devices operate using virtual addresses when reading and writing memory. Virtual addresses presented by the DMA device would be mapped into physical memory addresses using the mapping table of the process which "owned" the device.

Current Generation Protection Features

- Passwords for user authentication
- File system access controls
  - Self, group, others
  - Access control lists
- Audit mechanisms

Let us now examine some of the types of protection features of current generation operating systems.

One of the most common is the use of passwords for user authentication. When a user attempts to logon to the system, he must identify himself to the system and submit a password. The system assures that the user is who he says he is by checking the password against a list of valid users and their associated passwords. It is important that the system authenticate the user in some fashion, lest a malicious user masquerade as some other user and gain access to information to which he is otherwise not entitled.

Most current generation operating systems support mechanisms to control access to information in the file system. File system access controls must accommodate both the protection and controlled sharing of information. Systems such as TENEX and UNIX support file system protection based on the notion of self, group, and others. This means that the creator or owner of a file can specify what type of file access (e.g., read, write, execute, delete) is allowed to himself (the owner), to other users in the same group as the owner, and to all other users of the system. A much more flexible file protection mechanism is access control lists, as supported by MULTICS, wherein the creator or owner of a file can specify that particular users can have certain access to the file. For example, the owner may specify that Jones has read access, Smith has read and write access, Brown has read access, and everyone else has no access.

Finally, most current generation systems maintain an audit log of significant events. For example, whenever a user logs on, the operating system makes an entry into the audit log recording the user's identification and the date and time of logon. The system can then read the audit log and display to the user information describing when he last logged on. Should the user determine that he didn't really logon previously, as the system says he did, he can infer that someone knows his password. The system may also record in the audit log information describing for each file: time of last access, time of last modification, and by whom. The system should make this information available to the owner of the file so that he can be assured that the file is being used only as he wishes.

Current Generation Security

Operating systems still unreliable

Multilevel use not possible

Traditional security techniques used

System high operation

Periods processing

Well, with all of these interesting hardware and software protection features, and the greater emphasis on operating system design, you might conclude that current generation systems are by and large reliable enough to permit their use for processing multilevel information. Unfortunately this is not the case. Current systems are still too unreliable and the traditional security techniques of system high operation and periods processing must be used in order to process classified information.

Current Generation Penetrations

VM/370 SDC

TENEX BBN

MULTICS ESD MITRE

MTS University of Michigan

As evidence of the continuing lack of reliable access control mechanisms, consider the following successful penetrations of current generation systems.

A group consisting of representatives from IBM and System Development Corporation was able to penetrate the IBM Virtual Machine operating system, VM/370, which runs on the System/370. VM/370 differs from conventional operating systems in that the interface presented to the user is that described in the Principles of Operation of System/370. VM/370 implements a number of virtual System/370 machines; i.e., each user has the illusion of having a stand-alone 370 consisting of a CPU, main memory, and I/O devices. VM/370 endeavors to isolate each virtual machine. By sharing virtual devices, data sharing with VM/370 is analogous to the way in which data is typically shared among real machines.

The basis of all penetrations perpetrated by IBM/SDC was the complexity of the VM/370 Control Program (CP) in mapping virtual machine I/O onto real machine I/O. Each virtual machine user can write a virtual machine channel program which the CP, for performance reasons, would analyze for legality and then generate a real I/O channel program to accomplish the virtual machine I/O. The analysis was able to catch attempts at self-modifying virtual machine channel programs, but the penetrators were able to introduce "puns" in the real channel programs generated by CP. These puns were based on the ability to chain I/O commands on the real machine. The penetrators were able to seize control of the real machine and

have their own code run in supervisor domain. They were thus able to access files belonging to all VM/370 users. Also, they found that the CP was unable to control excessive demands by a particular virtual machine, and they were able to monopolize real machine resources, denying service to other virtual machines.

The TENEX operating system was penetrated in several ways by a group at Lawrence Livermore Laboratories. The most interesting penetration path was dubbed the "Password Information Leak." It is a very subtle and complex flaw and characterizes a very stubborn problem common in probably every system: the release of sensitive information through timing or other unrelated information channels. In the case of TENEX, the channel is the users ability to determine a page fault occurrence, i.e., a reference to a page in the user process' virtual address space which is not resident in main memory.

TENEX processes can submit passwords for checking in several flexible ways. For instance, the process can invoke a system call to gain access to some directory in the file system. Directories are password protected, so the process must know the password for the desired directory. The process would include the password as a parameter to the system call. The process is able to put the password in user space and submit a pointer to it as the system call parameter.

TENEX would do character-at-a-time checking of the password and the knowledgeable programmer could exploit this fact to guess passwords. The programmer would position the submitted password so that it straddled a page boundary. For example, the first character would reside in the last byte of a resident virtual page and the remaining characters would reside in the leading bytes of the next page, which the programmer would arrange to be non-resident. The programmer would issue the system call and subsequently determine whether the non-resident page was referenced by TENEX. If so, the programmer knew that the first character of the password was correct and he would rearrange the position of the password and issue the call again to guess the next character.

Another flaw the LLL group found was that typescript files of user sessions were automatically assigned a protection mode of read access to everyone. These files could contain very sensitive information, like passwords.

A very extensive penetration analysis of the MULTICS system was performed by people from the Air Force Electronic System Division (ESD), with support from individuals at The MITRE Corporation. This group was successful in finding an exploitable weakness in the MULTICS hardware (the early Honeywell 645) and several weaknesses in the MULTICS operating system.

The hardware problem concerned the bypassing of access checking of operand addresses of the "execute" instruction. The trick was to issue an execute instruction which had to resolve a series of indirect addresses through different segments before the ultimate operand was fetched. The hardware failed to perform access checking on the final address in the sequence. The penetrators were able to read or write a segment without the hardware checking the access permissions in the final segment descriptor.

A somewhat similar flaw was discovered in the parameter validation routines of the MULTICS operating system. The ring 0 validation routines performed insufficent validation of parameters supplied by outer ring procedures on system calls to ring 0. The outer ring procedure could fool the ring 0 validator by supplying a pointer as a parameter which indirected through several segments before the ultimate parameter was reached. The validator neglected to perform access checking before retrieving or storing the ultimate location.

These access checking flaws in the hardware and software could be exploited in a number of ways. The malicious user could change his user ID stored in his process data segment in ring 0 to be somebody else and gain access to their files. Or the user could modify ring 0 code and plant a trap door, or do just about anything else. (Note: the flaw in the execute instruction has been remedied in the new Honeywell 6180 hardware; also, the ring 0 parameter validator is now done by the 6180 hardware -- correctly!)

Finally the Michigan Terminal System (MTS) was successfully penetrated by a group of University of Michigan graduate students in advanced operating system principles. MTS runs on an IBM System/370 plug-compatible Amdahl 470V/6 and was designed to be secure from penetration by student users -- even if they had full access to all system documentation and listings. In fact, the penetration project was undertaken at the invitation and full cooperation of the University of Michigan Computer Center!

MTS is a general purpose operating system providing both batch and interactive service. It supported over 25,000 user accounts and over 250 terminal users per day.

The class used SDC's Flaw Hypothesis Methodology and was very successful. The flaws they found and exploited are similar to those already discussed for other systems. One flaw in the system's parameter checking routine allowed the penetrators to trick the system into storing arbitrary bit strings into system data bases. The

penetrators could alter accounting data, assume the identity of other users, and run their programs in privileged mode. The parameter checking flaw was that the system could be made to alter, unwittingly, pointer parameters after they had been checked for legality, but before they were actually used during the system call.

The Michigan group also found it easy to cause system routines to branch to user code without changing the processor mode back to unprivileged.

Summary conclusions in their report accurately characterize the general reliability problem. "A large operating system frequently depends on a number of control structures, each of which assumes that the others function correctly. However, a flaw in one such component may render the others useless. [The operating system designer]... must distinguish security relevant control structures from non-security relevant structures and concentrate on the former."

Summary

Protection problems arose when resources were shared

Third generation operating systems were not designed with protection as a primary goal

Current generation operating systems

Have added hardware support for protection

Centralize protection mechanisms

But suffer inadequate software design and development methodology

Let us briefly summarize this examination of the evolution of protection in operating systems.

We saw that the protection of information by the operating system did not become of concern until the third computer generation when resources were shared by a number of user programs concurrently resident in main memory. By and large, however, third generation operating systems were not designed with information protection as a primary goal. Rather, the goals were efficient utilization of resources and system throughput.

Current generation hardware and software designers were well aware of the unreliable nature of third generation protection and endeavored to make improvements. The hardware designers were successful. Intuitively, one can see that the hardware protection mechanisms provide adequate hardware support. The operating system designers and implementors were less successful. Although they clearly recognized the importance of concentrating on a better structuring of the internal software design -- and some did indeed centralize those components which dealt with resource utilization, the implementing of abstractions, and information protection -- the resulting software still tended to be large, complex, and bug prone.

In order to produce operating systems with more reliable information protection, better design and development methodolcgies are needed. Now that the importance of structuring the internal design of a system and of kernelizing components responsible for

information access control is well accepted, what are needed now are improved techniques for unambiguously specifying the design, and for constructing programs which can be shown to implement correctly the design. This requirement can be satisfied by evolving software design, development, and verification methodologies incorporating formal specification languages, verifiable programming languages, and verification techniques which strive to demonstrate via mathematical proof the correspondence between program specification and program code.

APPENDIX

This appendix includes lecture slides providing background information on the multilevel computer security problem and the requirement for trusted computing systems.

COMPUTER SECURITY PROBLEM

The protection of information

as it is stored within or processed by

a computer system serving a community of users

This opening slide attempts to define the computer security problem as it exists today. We state the overall problem rather simply as: the protection of information as it is stored within or processed by a computer system serving a community of users.

Our emphasis is on systems designed to support a community of users through the sharing of resources — the central processor, storage devices, communications lines. Concern about the ability of computing systems to protect the information they store and process became most acute with the advent, about a dozen years ago or so, of interactive systems designed to support multiple users simultaneously. The Department of Defense naturally desired to employ such systems for a variety of applications; many of these, however, had a requirement for the concurrent processing of information classified at multiple levels, and some potential applications had a requirement for supporting simultaneously users of multiple clearances.

Great concern arose, at that time, over whether the operating systems of this new breed of computer system could effectively maintain the separation of multilevel information and, further, whether they could prevent the malicious user from gaining access to information to which he was not entitled. As many of you know, it was quite evident these systems could not be trusted to enforce the separation of information, or deny the malicious user access to classified information.

With that unfortunate realization began the long process of research and development programs into the construction of operating systems incorporsting information protection mechanisms to permit the simultaneous storage and processing of multilevel information.

And this is what we are primarily here to examine: the provision of information protection mechanisms, and assurances of their correct implementation, in modern operating systems.

INTRODUCTION

SOLUTION : CENTRAL ISSUE IS ACCESS CONTROL

- EFFECTIVELY CONTROLLING ACCESS TO

a) THE COMPUTER SYSTEM ITSELF

b) INFORMATION CONTAINED WITHIN IT

One of the first groups to intensively examine the computer security problem was the Defense Science Board's Task Force on Computer Security, formed in 1967 to study and recommend hardware and software safeguards that would satisfactorily protect classified information in multi-access, resource-sharing computer systems.

Essentially, the central issue in solving the problem, so the Board concluded, is one of access controls -- effective, non-circumventable access controls. Mechanisms must be adopted which effectively control access to the computer system itself, and to the information contained within the system.

Adequate mechanisms had already existed to effectively control access to the system. After all, classified information was being stored and processed on computer systems at that time -- not multilevel information on multi-access systems, however -- and techniques did exist to control access to these systems.

It was the second aspect, that of effectively controlling access to information contained within the system, by the users of the system, for which effective mechanism did not exist and for which much work needed to be done.

INTRODUCTION

POLICY DICTATES ACCESS CONTROL RULES

DODD 5200.1R CHOSEN AS OUR STANDARD

MANDATORY PROTECTION POLICY

- INFORMATION HAS CLASSIFICATION
- USERS HAVE CLEARANCES
- CLEARANCE <= CLASSIFICATION

DISCRETIONARY PROTECTION POLICY

Some policy must form the basis of the access control rules. Department of Defense Directive 5200.1R, which outlines the policy to be followed in the realm of people and paper documents, can be naturally extended to the computer system domain.

The Directive establishes a mandatory protection policy to govern the handling of classified documents. It states that all information is assigned a classification which identifies the sensitivity of the information by ascertaining the potential level of damage to the interests of the United States were the information to be divulged to an unfriendly foreign agent. There are three formal levels of classification: Top Secret, Secret, and Confidential. However, when the policy is extended to the computer system domain, it is useful to consider Unclassified as a fourth level of classification.

Information may also carry a Special-Access Category or Compartment, if that information is segregated and entrusted to a particular agency or organizational group for safeguarding. For example, information pertaining to nuclear matters is entrusted to the Atomic Energy Commission. Note that compartments create a further structuring within classification levels.

The Directive also stipulates that all personnel are to be assigned a clearance, which is the privilege granted to an

individual on the basis of a background investigation to access classified information necessary to his work. There are three formal national clearances: Top Secret, Secret, and Confidential. Again, when extended to the realm of computer systems, it is useful to include a fourth category, Uncleared.

The general access control rule is that in order to be granted access to classified information, an individual's clearance must be equal to or greater than the classification of the information, where the clearances and classifications are linearly ordered Confidential, Secret, Top Secret. Further, if the information carries a compartment, the individual must also be cleared to access information of that compartment.

A clearance, however, is a necessary but not sufficient condition to have access to classified information. DODD 5200.1R also establishes a discretionary protection (or need-to-know) policy. Need-to-know is an administrative action certifying that a given individual requires access to specified classified information in order to perform his assigned duties. The combination of a clearance and a need-to-know constitutes the necessary and sufficient conditions for granting access to classified information.

CLASSIFIED PROCESSING

DODD 5200.28 REGULATES PROCESSING OF CLASSIFIED INFORMATION

- ESTABLISHES POLICY FOR PROTECTING CLASSIFIED INFORMATION STORED AND PROCESSED
- OUTLINES MODES OF OPERATION

As many of you know, computer systems are indeed used to store and process classified information. The formal requirements on computer systems that must process any form of classified information are set forth in Department of Defense Directive 5200.28, "Security Requirements for Automatic Data Processing (ADP) Systems."

This directive establishes policy for protecting classified information stored, processed, or used within an ADP system. It provides for the application of administrative, physical, and personnel security measures required to protect ADP equipment, material, and installations from inadvertant or deliberate compromise.

It states that security controls should prevent deliberate or inadvertant access to classified material by unauthorized persons, and unauthorized manipulation of the computer and its associated peripheral devices.

It also establishes a policy outlining the modes of operation in which classified data may be processed.

CLASSIFIED PROCESSING

DOD 5200.28-M IS ADP SECURITY MANUAL

OUTLINES CONTROLS

- PHYSICAL
- PERSONNEL
- ELECTROMAGNETIC
- COMSEC
- OPERATING SYSTEM ACCESS CONTROLS
- PROCEDURAL

DOD 5200.28-M, "Techniques and Procedures for Implementing, Deactivating, Testing, and Evaluating Secure Resource-Sharing ADP Systems," is also known as the ADP Security Manual. It expands upon the policies and requirements of DODD 5200.28 and outlines specific measures and controls to be established for compliance with the policy.

Sections in the manual include: physical, personnel, electromagnetic, communications, operating system, and procedural safeguards.

Physical security controls safeguard the computer system itself and access to it. It is the first line of defense, so to speak. The central computer installation and terminal locations should be in areas of restricted access. Types of measures employed here are vaults, locked doors, armed guards. The intent, of course, is to deny improperly cleared individuals access to the computer itself and to any of its terminals. Other issues relating to physical security are:

- fire, flood, earthquate safeguards
- hardware maintenance
- file backup
- recovery plans
- control of documentation

Personnel controls relate to establishing the trustworthiness of individuals permitted access to the computer system. Naturally, in the DOD environment personnel security relates to attaining clearance levels for these individuals commensurate with the classification and special categories of the information to be processed. It is very important to educate personnel as to their responsibilities in handling classified information in the computing domain.

Electromagnetic emanations are released during the transmission of information over communications lines and during the operation of computing equipment. It is possible to intercept and monitor these waves and determine the information being transmitted or processed. Generally, the field of strength of these emanations increases as the voltage and current increases. Therefore high voltage devices like CRTs and high current devices like core memories and electomechanical devices (card readers, printers) are of major concern. Techniques exist for analyzing emanations released by devices and for reducing these emanations.

Communications security addresses the passive monitoring of electromagnetic emanations and the active wire tapping of information during transmission. Communications security employs various cryptographic techniques to combat these threats. Commercially-available equipment exists for effectively and efficiently encrypting information transmitted over channels of small and large bandwidth.

Operating systems security relates to any security and protection mechanisms supported by the operating system or by any application subsystem. Types of controls considered here are passwords to support user authorization and access to files, the labelling of output, and the logging of user operations.

Finally, procedural controls refer to the coordinated administration of all of the above types of safeguards to satisfy the DODD 5200.28. Generally, a system security officer is assigned responsibility for administering these safeguards. Responsibilities include the registration of users, overseeing the mode of operation, establishing erasure and declassification procedures, reviewing audit log information.

CLASSIFIED PROCESSING

UNILEVEL PROCESSING

- IS NOT PART OF THE PROBLEM
- CAN BE HANDLED BY PROPER APPLICATION OF THE VARIOUS SAFEGUARDS
- DEDICATED SYSTEM

We distinguish between two types of classified processing: unilevel and multilevel processing. Unilevel processing is where the computer system stores and process information of a single classification level. The system is dedicated to processing at a single, fixed security level. This type of processing is not part of the problem; it is well understood and can be handled by the judicious application of the safeguards and procedures stipulated in DOD 5200.28-M.

CLASSIFIED PROCESSING

MULTILEVEL PROCESSING

- SYSTEM HIGH OPERATION
- PERIODS PROCESSING

Multilevel processing is a situation where processing of several different levels of classification must be done on a single computer system. The ADP Security Manual identifies two modes of operation to support multilevel processing.

The first is system high operation. This is a mode of operation where all users are cleared for the highest level of data being processed in the system, and all processing takes place at that level. All jobs must be upgraded to the system high level.

An implication of system high operation is that an unclassified job run on such a system would have to have its output treated as classified. This is unfavorable because it proliferates classified information. Before the output of the unclassified job can be handled again in an unclassified manner, it must go through a standard declassification procedure. Further, because we cannot trust the computer to separate the data of a real classified job and an upgraded classified job, we are not sure that the output of the upgraded job is really unclassified. It must be thoroughly reviewed.

This mode of operation is useful only if all users can be cleared to the system high level.

The second accepted mode of multilevel operation is periods processing, or color change operation. With this mode the computer is dedicated for use at specific security levels for different periods of time. For example, at SECRET in the morning and UNCLASSIFIED in the afternoon.

When the system transists from one level to another, certain procedures must be followed to change the level (color) of the system from one level (color) to another. This is called a color change.

Color change consists of three phases -- quiescence, changeover, restart -- and may take upwards of two hours from peak processing activity at one level to peak processing at the new level. Changeover phase involves removing storage media, clearing all memories and processors, removing all printer ribbons, and loading a fresh copy of the operating system.

The advantage of periods processing over system high operation is that there is no proliferation of classified information. The disadvantage is the time wasted during the color change procedure.

CLASSIFIED PROCESSING

TRUE MULTILEVEL PROCESSING

- CONCURRENT PROCESSING OF MULTILEVEL INFORMATION BY USERS OF DIFFERING LEVELS OF CLEARANCE
- COMPUTER ASSURES SEPARATION OF INFORMATION

REQUIRES TRUSTED OPERATING SYSTEM

- PROTECTION FEATURES IMPLEMENTING EFFECTIVE INFORMATION ACCESS CONTROLS
- ASSURANCE OF COMPLETE AND CORRECT OPERATION

What we are striving to achieve, however, is true multilevel classified processing, which we define to be the concurrent processing of information of more than one level of classification by users of more than one level of classification. A prerequisite for supporting this mode of operation is a computer system which can be trusted to maintain the separation of information classified at different levels.

The system must maintain a clearance attribute for each user and a classification attribute for all information contained within the system; and, before a user (i.e., his program) may access any piece of information, the system must examine the clearance of the user and the classification of the information.

Trust means that the system is reliable enough not to accidentally compromise information, and robust enough to thwart the intentions of the malicious user.

True multilevel processing requires a trusted computing system. A trusted computing system is defined to be one supporting hardware and software protection features implementing effective information access controls. In addition, we must be assured of the complete and correct operation of the information access controls.