OPERATING SYSTEMS

Input - Output Management and Security - 3

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Goals, Principles and Domains of Protection

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Course Syllabus - Unit 5

10 Hours

Unit-5:Unit 5: IO Management and Security

I/O Hardware, polling and interrupts, DMA, Kernel I/O Subsystem and Transforming I/O Requests to Hardware Operations - Device interaction, device driver, buffering System Protection: Goals, Principles and Domain of Protection, Access Matrix, Access control, Access rights. System Security: The Security Problem,Program Threats, System Threats and Network Threats. Case Study: Windows 7/Windows 10

OPERATING SYSTEMS

Course Outline

47	I/O Hardware, polling and interrupts	13.1,13.2
48	DMA	13.2.3
49	Transforming I/O Requests to Hardware Operations, Device interaction, device driver, buffering.	13.5
50	Goals, Principles and Domain of Protection	14.1-14.3
51	Access Matrix	14.4
52	Access control, Access rights	14.5-14.7
53	The Security Problem	15.1
54	Program Threats	15.2
55	System Threats and Network Threats	15.3
56	Case Study : Windows File System	17.5

Topic Outline

- Overview
- Goals of Protection
- Principles of Protection
- Domains of Protection
- Domain Structure
- Domain Structure - UNIX
- Domain Structure - MULTICS

Overview

- The processes in an operating system must be protected from one another's activities.
- To provide such protection, we can use various mechanisms to ensure that only processes that have gained proper authorization from the operating system can operate on the files, memory segments, CPU , and other resources of a system.
- Protection refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.
- This mechanism must provide a means for specifying the controls to be imposed, together with a means of enforcement.

Goals of Protection

- Modern protection concepts have evolved to increase the reliability of any complex system that makes use of shared resources.

- The most obvious reason is the need to prevent the mischievous, intentional violation of an access restriction by a user

- Another important reason is the need to ensure that each program component active in a system uses system resources only in ways consistent with stated policies.

- This requirement is an absolute one for a reliable system.

Goals of Protection

- Protection can improve reliability by detecting latent errors at the interfaces between component subsystems.
- An unprotected resource cannot defend against use or misuse by an unauthorized or incompetent user.
- A protection-oriented system provides means to distinguish between authorized and unauthorized usage.
- The role of protection in a computer system is to provide a mechanism for the enforcement of the policies governing resource use.

Goals of Protection

- Some policies are fixed in the design of the system, while others are formulated by the management of a system includes those defined by the individual users to protect their own files and programs.
- A protection system must have the flexibility to enforce a variety of policies.
- In one protection model, computer consists of a collection of **Objects**, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so

Goals of Protection

- Policies for resource use may vary by application, and they may change over time.
- For these reasons, protection is no longer the concern solely of the designer of an operating system.
- The application programmer needs to use protection mechanisms as well, to guard resources created and supported by an application subsystem against misuse.
- Note that mechanisms are distinct from policies.

Goals of Protection

- Mechanisms determine how something will be done; policies decide what will be done.
- The separation of policy and mechanism is important for flexibility.
- Policies are likely to change from place to place or time to time.
- In the worst case, every change in policy would require a change in the underlying mechanism.
- Using general mechanisms enables us to avoid such a situation.

Principles of Protection

- A key, time-tested guiding principle for protection is the **Principle of Least Privilege.**

- It dictates that programs, users, and even systems be given just enough privileges to perform their tasks.

- An operating system following the principle of least privilege implements its features, programs, system calls, and data structures so that failure or compromise of a component does the minimum damage and allows the minimum damage to be done.

- The overflow of a buffer in a system daemon might cause the daemon process to fail, for example, but should not allow the execution of code from the daemon process stack that would enable a remote user to gain maximum privileges and access to the entire system

Principles of Protection

- An Operating System also provides system calls and services that allow applications to be written with fine-grained access controls.
- It provides mechanisms to enable privileges when they are needed and to disable them when they are not needed.
- Creation of audit trails for all privileged function access.
- The audit trail allows the programmer, system administrator, or law-enforcement officer to trace all protection and security activities on the system.
- Managing users with the principle of least privilege entails creating a separate account for each user, with just the privileges that the user needs.

Principles of Protection

- Some systems implement **R**ole-**B**ased **A**ccess **C**ontrol - **RBAC** to provide this functionality
- The principle of least privilege can help produce a more secure computing environment.
- More Features More Vulnerable
- Less Features More Protection
- Optimal Features Optimal Protection

Domains of Protection

- A computer system is a collection of **Processes** and **Objects**.
- **Hardware Objects** => CPU , memory segments, printers, disks, and tape drives etc.
- **Software Objects** => Files, Programs, and Semaphores etc
- Each object has a unique name that differentiates it from all other objects in the system, and each can be accessed only through well-defined and meaningful operations.
- Objects are essentially abstract data types.

Domains of Protection

- The operations that are possible may depend on the object.

- For example,
  - On a **CPU** , we can only execute.
  - **Memory segments** can be read and written
  - **CD-ROM or DVD-ROM** can only be write once read many times.
  - **Tape drives** can be read, written, and rewound.
  - **Data files** can be created, opened, read, written, closed, and deleted
  - **Program files** can be read, written, executed, and deleted.

Domains of Protection

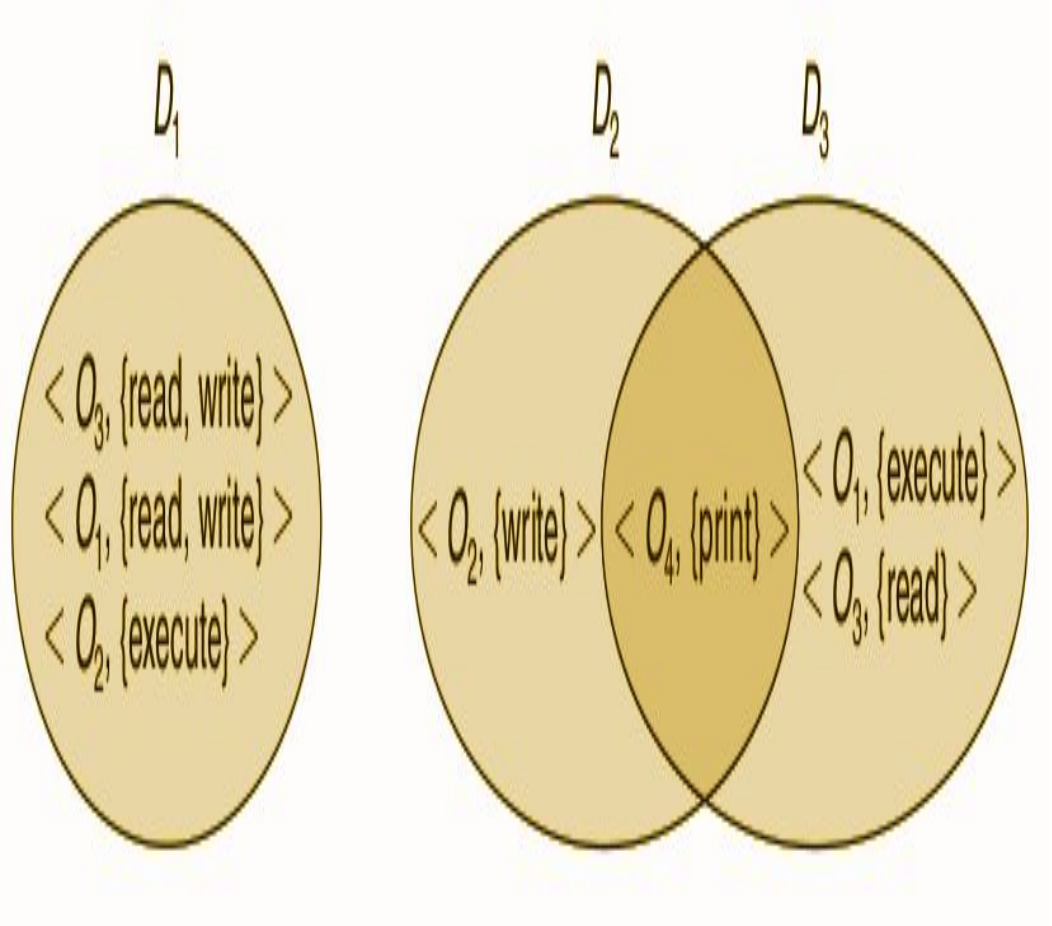
- **First Requirement:** A process should be allowed to access only those resources for which it has authorization.
- **Second Requirement:** A process should be able to access only those resources that it currently requires to complete its task.
- The second requirement, commonly referred to as the **Need-to-Know** principle, is useful in limiting the amount of damage a faulty process can cause in the system.
- For example, when process p invokes procedure A() , the procedure should be allowed to access only its own variables and the formal parameters passed to it; it should not be able to access all the variables of process p.
- Similarly, consider the case in which process p invokes a compiler to compile a particular file.
- The compiler should not be able to access files arbitrarily but should have access only to a well-defined subset of files (such as the source file, listing file, and so on) related to the file to be compiled.

Domain Structure

- A process operates within a protection domain, which specifies the resources that the process may access.
- Each domain defines a set of objects and the types of operations that may be invoked on each object.
- The ability to execute an operation on an object is an **Access Right**.
- A domain is a collection of access rights, each of which is an ordered pair < object-name, rights-set >
- For example, if domain D has the access right < file F , {read,write} >, then a process executing in domain D can both read and write file F.
- It cannot, however, perform any other operation on that object.

Domain Structure

- Domains may share access rights.
- For example, in Figure, we have three domains: D1 , D2 , and D3 .
- The access right <O4 , {print}> is shared by D2 and D3 , implying that a process executing in either of these two domains can print object O4 .
- Note that a process must be executing in domain D1 to read and write object O1 , while only processes in domain D3 may execute object O1 .

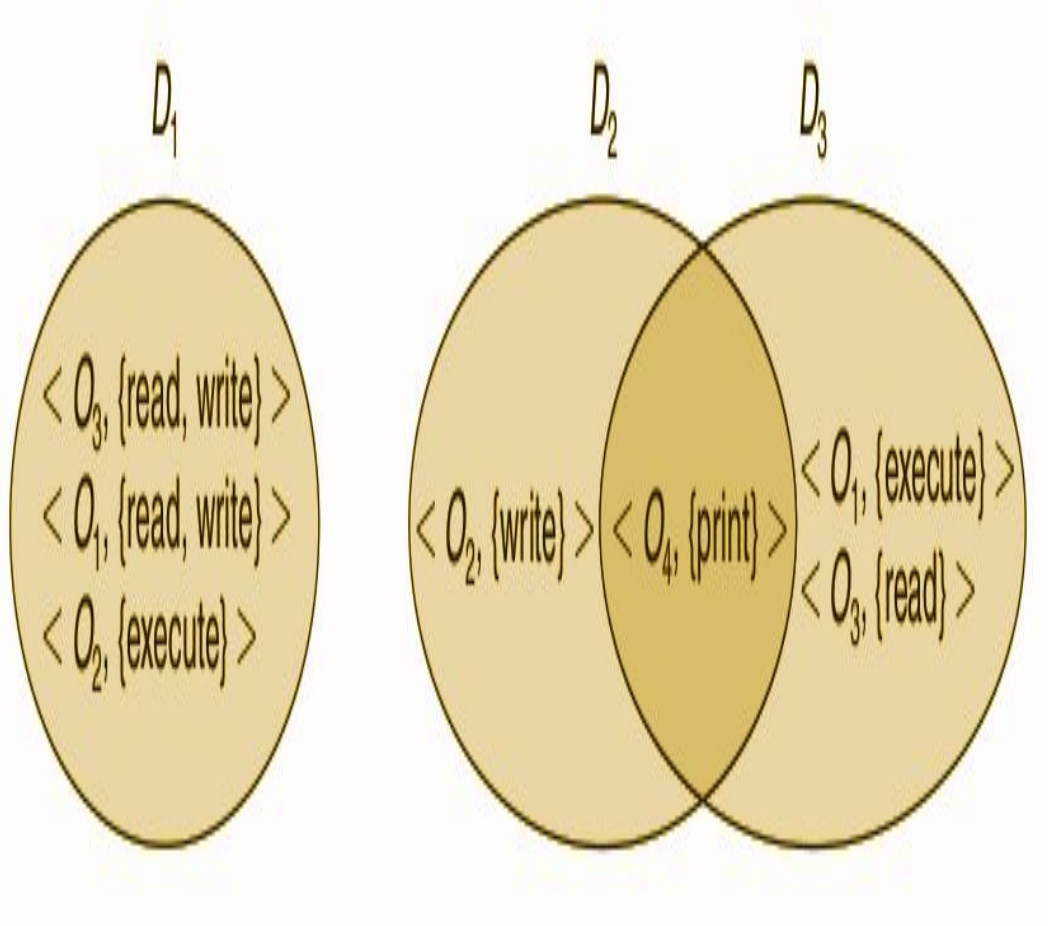


Domain Structure

- The association between a process and a domain may be either **Static**, if the set of resources available to the process is fixed throughout the process's lifetime, or dynamic.

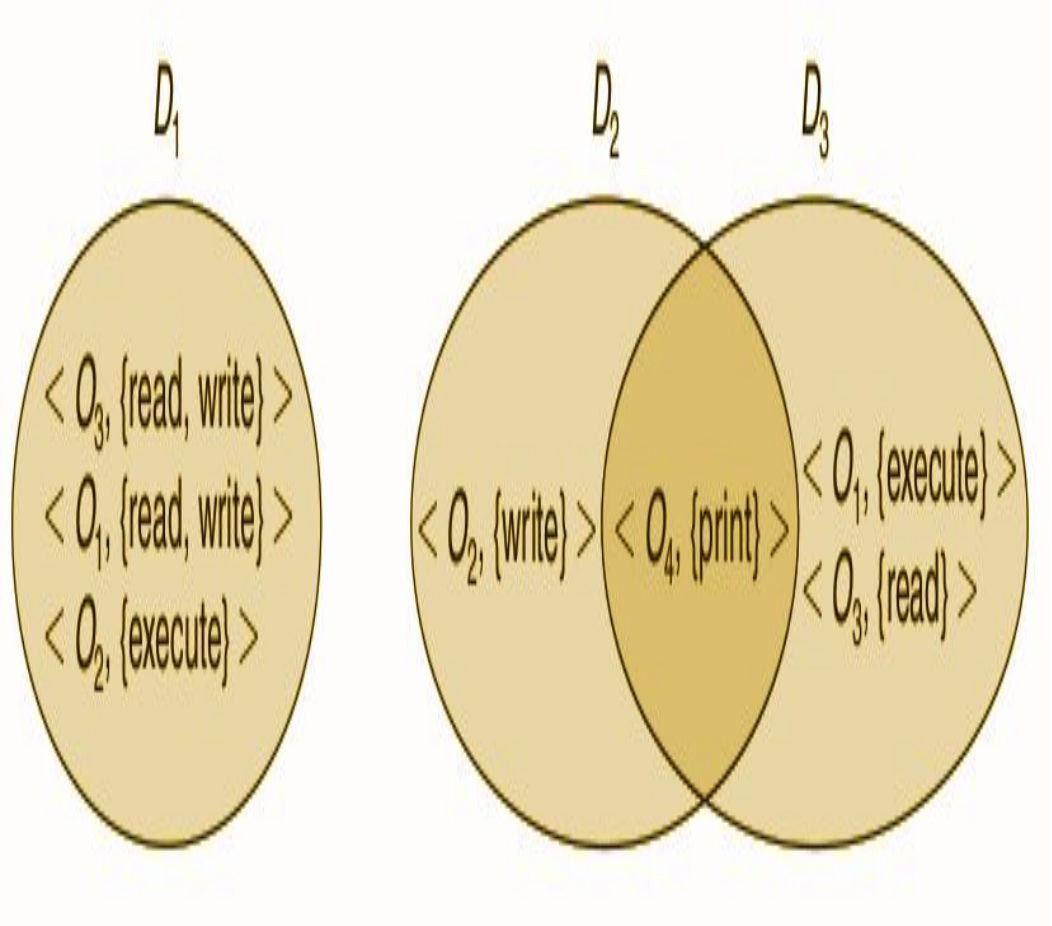
- As might be expected, establishing **Dynamic** protection domains is more complicated than establishing static protection domains.

- If the association between processes and domains is fixed, and we want to adhere to the need-to-know principle, then a mechanism must be available to change the content of a domain.



Domain Structure

- If the association is dynamic, a mechanism is available to allow domain switching, enabling the process to switch from one domain to another.
- We may also want to allow the content of a domain to be changed.
- If we cannot change the content of a domain, we can provide the same effect by creating a new domain with the changed content and switching to that new domain when we want to change the domain content.



Domain Structure

A domain can be realized in a variety of ways.

- Each **User** may be a domain. In this case, the set of objects that can be accessed depends on the identity of the user. Domain switching occurs when the user is changed —generally when one user logs out and another user logs in.

- Each **Process** may be a domain. In this case, the set of objects that can be accessed depends on the identity of the process. Domain switching occurs when one process sends a message to another process and then waits for a response.

- Each **Procedure** may be a domain. In this case, the set of objects that can be accessed corresponds to the local variables defined within the procedure. Domain switching occurs when a procedure call is made.

Domain Structure

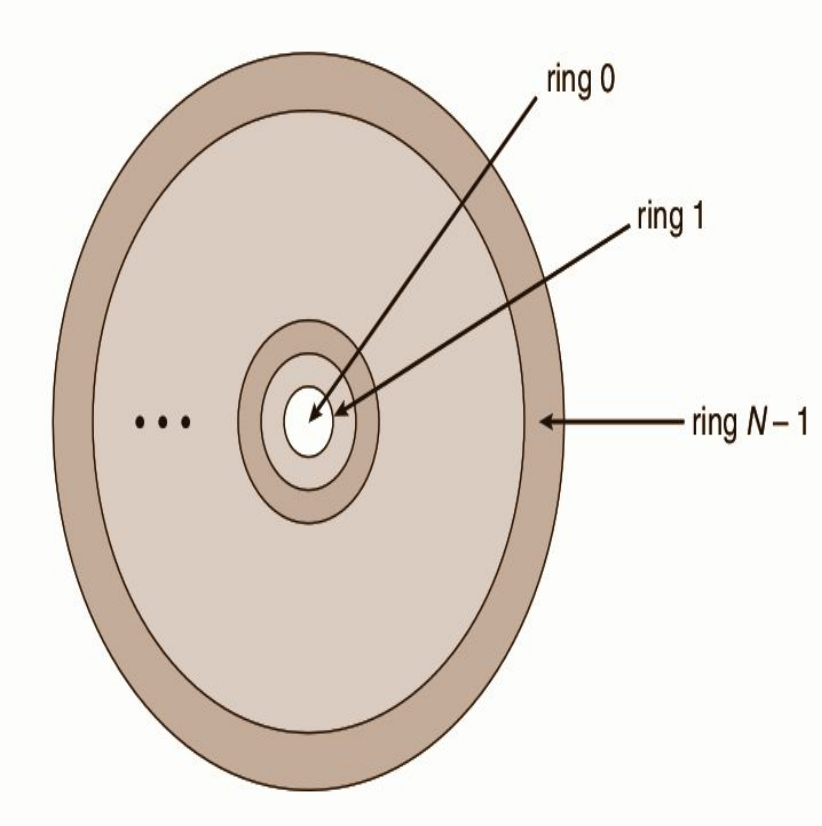
- When a process executes in monitor mode, it can execute privileged instructions and thus gain complete control of the computer system.
- In contrast, when a process executes in user mode, it can invoke only non-privileged instructions.
- Consequently, it can execute only within its predefined memory space. These two modes protect the operating system (executing in monitor domain) from the user processes (executing in user domain).
- In a multiprogrammed operating system, two protection domains are insufficient, since users also want to be protected from one another.
- A more elaborate scheme is needed.

Domain Structure - UNIX

- In the UNIX operating system, a domain is associated with the user.
- Switching the domain corresponds to changing the user identification temporarily.
- An owner identification and a domain bit known as the **setuid** bit are associated with each file.
- When the setuid bit is on , and a user executes that file, the user ID is set to that of the owner of the file.
- Even more restrictive, and thus more protective, are systems that simply do not allow a change of user ID . In these instances, special techniques must be used to allow users access to privileged facilities.
- For instance, a daemon process may be started at boot time and run as a special user ID . Users then run a separate program, which sends requests to this process whenever they need to use the facility. This method is used by the TOPS-20 operating system.

Domain Structure - MULTICS

- In the MULTICS system, the protection domains are organized hierarchically into a ring structure.
- Each ring corresponds to a single domain as shown in Figure.
- The rings are numbered from 0 to 7.
  - Let Di and Dj be any two domain rings.
  - If j < i, then Di is a subset of Dj . That is, a process executing in domain Dj has more privileges than does a process executing in domain Di .
  - A process executing in domain D0 has the most privileges. If only two rings exist, this scheme is equivalent to the monitor–user mode of execution, where monitor mode corresponds to D0 and user mode corresponds to D1 .



Topic Outline

- Overview
- Goals of Protection
- Principles of Protection
- Domains of Protection
- Domain Structure
- Domain Structure - UNIX
- Domain Structure - MULTICS

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com