OPERATING SYSTEMS

Protection

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

Protection - Introduction

- 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
- Protection was originally conceived as an adjunct to multiprogramming operating systems, so that untrustworthy users might safely share a common logical name space, such as a directory of files, or share a common physical name space, such as memory.
- Modern protection concepts have evolved to increase the reliability of any complex system that makes use of shared resources.

OPERATING SYSTEMS

Goals of protection

- Need to prevent mischievous, intentional violation of an access restriction by a user
- Need to ensure that each program component active in a system uses system resources only in ways consistent with stated policies.
- A protection-oriented system should provides means to distinguish between authorized and unauthorized usage
- Improve reliability by detecting latent errors at the interfaces between component subsystems. Early detection of interface errors can often prevent contamination of a healthy subsystem by a malfunctioning subsystem.
- Provide a mechanism for the enforcement of the policies governing resource use. A protection system must have the flexibility to enforce a variety of policies.

OPERATING SYSTEMS

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.
- An operating system should provide system calls and services that allow applications to be written with fine-grained access controls. It should provide mechanisms to enable privileges when they are needed and to disable them when they are not needed.

OPERATING SYSTEMS

Principles of protection

- Also beneficial is the 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.
- Some systems implement role-based access control (RBAC) to provide this functionality
- Computers implemented in a computing facility under the principle of least privilege can be limited to running specific services, accessing specific remote hosts via specific services, and doing so during specific times

OPERATING SYSTEMS

Domain of protection

- A computer system is a collection of processes and objects.
- ***Objects*** include both **hardware objects** (such as the CPU, memory segments, printers, disks, and tape drives) and **software objects** (such as files, programs, and semaphores).
- 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.
- The operations that are possible may depend on the object.
- Processes should be allowed to access only those resources for which it has authorization.
- At any time, a process should be able to access only those resources that it currently requires to complete its task. This 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

OPERATING SYSTEMS

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.
- Domains may share access rights

OPERATING SYSTEMS

Domain structure

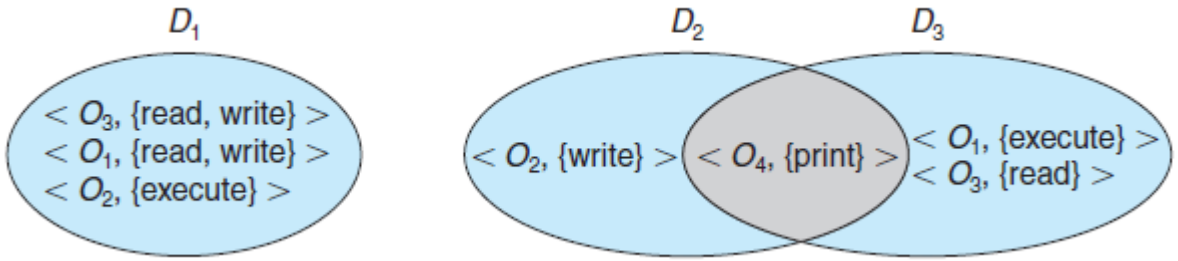
- The association between a process and a domain may be either **static**, if the set of resources available to the process is fixed throughout the process's lifetime, or **dynamic**.
- Establishing dynamic protection domains is more complicated than establishing static protection domains.
- If the association 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.
- The reason stems from the fact that a process may execute in two different phases and may, for example, need read access in one phase and write access in another.
- If a domain is static, we must define the domain to include both read and write access. However, this arrangement provides more rights than are needed in each of the two phases, since we have read access in the phase where we need only write access, and vice versa

Domain structure

- Thus, the need-to-know principle is violated. We must allow the contents of a domain to be modified so that the domain always reflects the minimum necessary access rights.
- 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

OPERATING SYSTEMS

Domain structure example



- We have three domains: *D*1, *D*2, and *D*3.
- The access right <*O*4, *{*print*}*> is shared by *D*2 and *D*3, implying that a process executing in either of these two domains can print object *O*4.
- A process must be executing in domain *D*1 to read and write object *O*1, while only processes in domain *D*3 may execute object *O*1

OPERATING SYSTEMS

Domain structure implementation

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.

OPERATING SYSTEMS

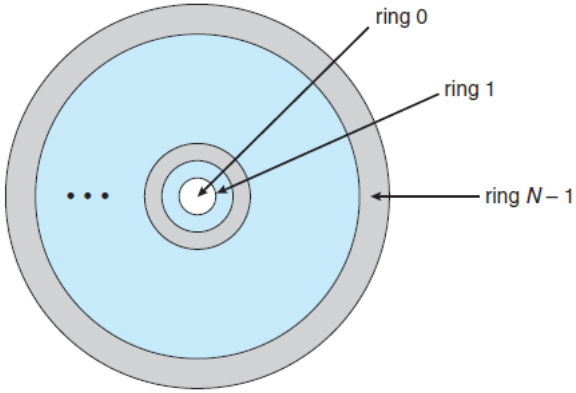
Domain structure implementation - UNIX

- In the UNIX operating system, a domain is associated with the user.
- Switching the domain corresponds to changing the user identification temporarily.
- This change is accomplished through the file system as follows.
  - 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 userID is set to that of the owner of the file. When the bit is off, however, the userID does not change.
  - For example, when a user *A* (that is, a user with userID = *A*) starts executing a file owned by *B,* whose associated domain bit is off, the userID of the process is set to *A.* When the setuid bit is on, the userID is set to that of the owner of the file: *B.*
  - When the process exits, this temporary userID change ends.

OPERATING SYSTEMS

Domain structure implementation - MULTICS

- In the MULTICS system, the protection domains are organized hierarchically into a ring structure. The rings are numbered from 0 to 7.
- MULTICS has a segmented address space; each segment is a file, and each segment is associated with one of the rings. A segment description includes an entry that identifies the ring number.
- Each ring corresponds to a single domain.
- 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 D_0 has the most privileges



OPERATING SYSTEMS

Domain structure implementation - MULTICS

- A current-ring-number counter is associated with each process, identifying the ring in which the process is executing currently.
- When a process is executing in ring i, it cannot access a segment associated with ring j ($j < i$). It can access a segment associated with ring k ($k \geq i$).
- Domain switching in MULTICS occurs when a process crosses from one ring to another by calling a procedure in a different ring
- The main disadvantage of the ring (or hierarchical) structure is that it does not allow us to enforce the need-to-know principle.
- In particular, if an object must be accessible in domain *Dj* but not accessible in domain *Di* , then we must have $j < i$. But this requirement means that every segment accessible in *Di* is also accessible in *Dj* .

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu