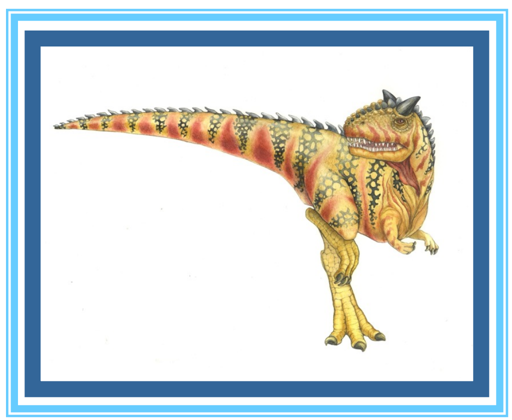
Chapter 14: Protection



Chapter 14: Protection

- Goals of Protection
- Principles of Protection
- Domain of Protection
- Access Matrix

Objectives

- Discuss the goals and principles of protection in a modern computer system
- Explain how protection domains combined with an access matrix are used to specify the resources a process may access

Goals of Protection

- In one protection model, computer consists of a collection of objects,
  - **hardware objects** (e.g., CPU, memory segments, printers, disks, and tape drives
  - **software objects** (e.g., files, programs, and semaphores).
- 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

- The role of protection in a computer system is to provide a mechanism for the enforcement of the policies governing resource use

  Note that ***mechanisms*** and ***policies are different***.
  - Mechanisms determine ***how*** something will be done.
  - Policies decide ***what*** will be done.

Principles of Protection

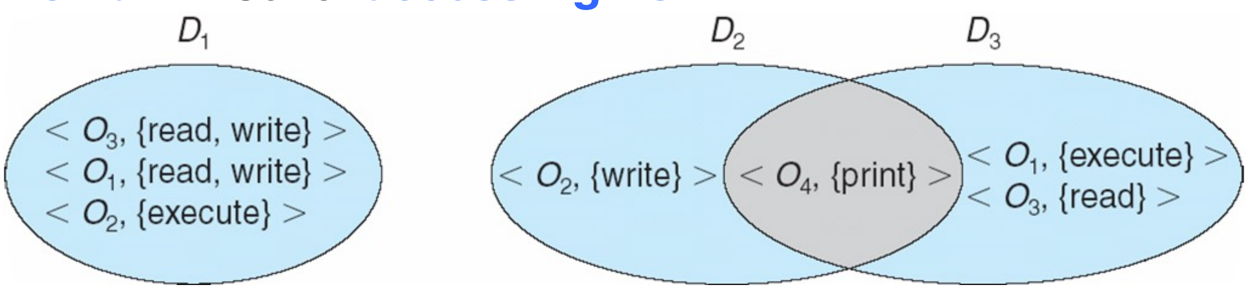
- Guiding principle – **principle of least privilege**
  - Programs, users and systems should be given just enough **privileges** to perform their tasks
  - Limits damage if entity has a bug, gets abused/lost
  - Managing users with the **principle of least privilege** entails creating a separate account for each user, with just the privileges that the user needs.
  - "**Need-to-know principle**" a similar concept regarding access to data (at any time, a process should be able to access only those resources that it currently requires to complete its task)

Principles of Protection (Cont.)

- Must consider "grain" aspect
  - Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
    - For example, traditional Unix processes either have abilities of the associated user, or of root
  - Fine-grained management more complex, more overhead, but more protective (It provides mechanisms to enable privileges when they are needed and to disable them when they are not needed).
    - File ACL lists, RBAC
- Domain can be user, process, procedure

Domain Structure

- A **protection domain** 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.**
- **Access-right** = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object
- **Domain** = set of **access-rights**



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.*

Domain Structure

- The association between a process and a domain may be either
  - **Static:** the set of resources available to the process is fixed throughout the process's lifetime
  - **Dynamic:** changed by process as needed – **domain switching** **(**enabling the process to switch from one domain to another), **privilege escalation**
- A domain can be realized in a variety of ways:
  - Each ***user*** may be a domain
  - Each ***process*** may be a domain
  - Each ***procedure*** may b e a domain

Domain Implementation (UNIX)

- Domain = user-id
- Domain switch accomplished via file system
  - Each file has associated with it a domain bit (*setuid bit*)
  - When file is executed and setuid = on, then user-id is set to owner of the file being executed
  - When execution completes user-id is reset
- Domain switch accomplished via passwords
  - `su` command temporarily switches to another user's domain when other domain's password provided
- Domain switching via commands
  - `sudo` command prefix executes specified command in another domain (if original domain has privilege or password given)

Access Matrix

- Protection model can be viewed as a matrix ➔ **access matrix**
  - Rows: represent domains
  - Columns: represent objects
  - Entry: **`Access(i, j)`** is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix

- The access matrix provides a mechanism for defining and implementing strict control for both static and dynamic association between processes and domains.
- When we switch a process from one domain to another, we are executing an operation (*switch*) on an object (the *domain*).

Access Matrix of with Domains as Objects

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Use of Access Matrix

- User who creates object can define access column for that object
- Can be expanded to dynamic protection (Allowing controlled change in the contents of the access-matrix entries)
  - Operations to add, delete access rights
  - Special access rights:
    - *owner of* O_i
    - *copy op from* O_i *to* O_j *(*denoted by asterisk *"*"* appended to the access right*)*
    - *control* – D_i can modify D_j access rights
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable only to domain objects

Access Matrix with *Copy* Rights

The copy right allows the access right to be copied only within the column (for the object) for which the right is defined.

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access Matrix With *Owner* Rights

We also need a mechanism to allow addition of new rights and removal of some rights. The owner right controls these operation

- If access(i, j) includes the *owner* right, then a process executing in domain D_i can add and remove any right in any entry in column j

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Modified Access Matrix with *control* Right

If access(i, j) includes the *control* right, then a process executing in domain D_i can add or remove any access right from row j.

control right in access (D_2, D_4).
→ a process executing in domain D_2 could modify domain D_4

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Use of Access Matrix (Cont.)

- **Access matrix** design separates mechanism from policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - It ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced
  - Policy
    - User dictates policy
    - Who can access what object and in what mode
- However, it does nott solve the general **confinement problem**
- The problem of guaranteeing that no information initially held in an object can migrate outside of its execution environment is called the **confinement problem.**

End of Chapter 14