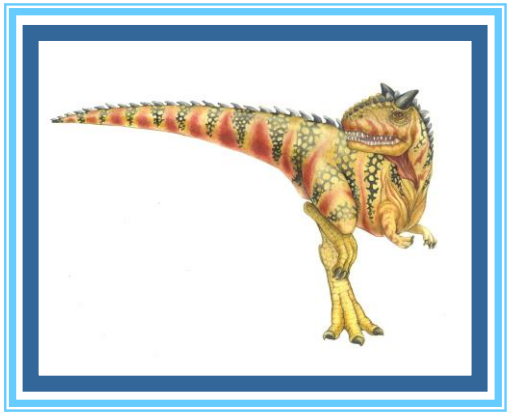
Chapter 17: Protection

Section 17.1-17.2,17.4-17.6



Goals of Protection

- A computer system consists of a collection of objects
  - **Hardware objects** (such as devices) and **software objects** (such as files, programs, semaphores)
- Each object has a unique name and can be accessed through a well-defined set of operations
- Protection involves controlling the access of processes and users to the resources defined by a computer system
  - OS ensures that each object is accessed correctly and only by those processes that are allowed to do so
- Protection mechanisms provide a means for specifying the controls to be imposed and a means of enforcement

Principles of Protection

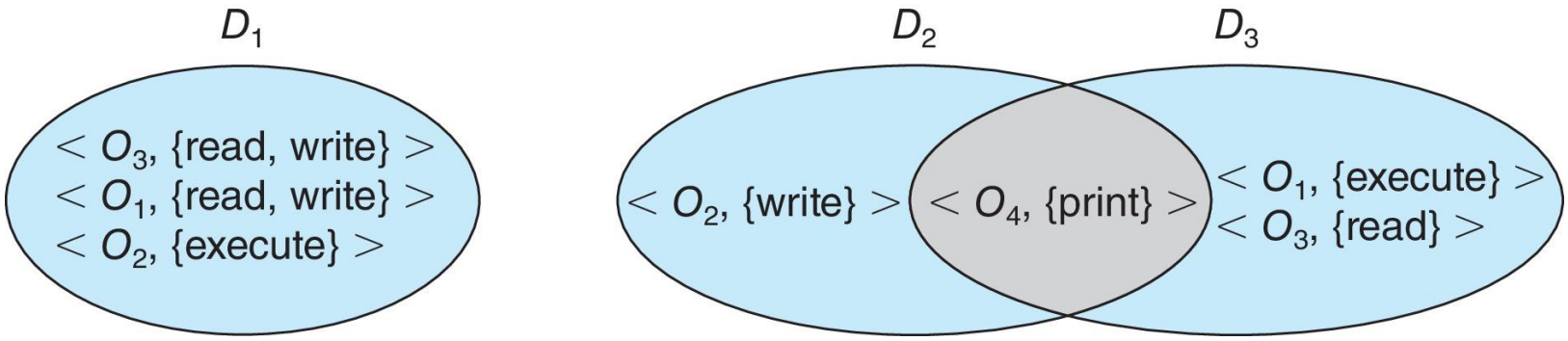
- Guiding principle – **principle of least privilege**
  - Processes and users should be given just enough **privileges** to perform their tasks
- **Compartmentalization** – a derivative of the principle of least privilege
  - Process of protecting each individual system component through the use of specific permissions and access restrictions
  - Properly set **permissions** can limit damage if entity has a bug, gets abused

Domain of Protection

- Process should only have access to objects it currently requires to complete its task – the **need-to-know** principle
- Implementation can be via process operating in a **protection domain**
  - Specifies resources process may access
  - Each domain specifies a set of objects and types of operations that may be invoked on each object
  - 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
  - A domain is a set of access-rights

Domain Structure

- Domain = set of access-rights
- Domains may share access rights
- Association between a process and a domain can be **static** or **dynamic**
  - If dynamic, a process can switch from one domain to another



Domain Structure

A domain can be realized in several ways:

- Each user may be a domain
  - The set of objects that can be accessed depends on the identify of the user
- Each process may be a domain
  - The set of objects that can be accessed depends on the identify of the process
- Each procedure may be a domain
  - The set of objects that can be accessed corresponds to the local variables defined within the procedure

Access Matrix

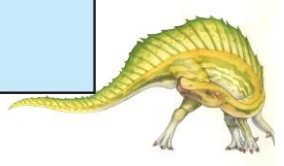
- **Access matrix** provides a mechanism for protection
  - Rows represent domains
  - Columns represent objects
  - Entry `access(i, j)` defines the set of operations that a process executing in domain D_i can invoke on object O_j

domain \ object	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

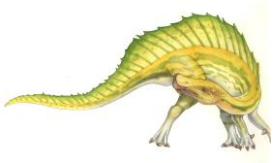
- If a process in domain D_i tries to do "op" on object O_j, then "op" must be in the entry **`access(i, j)`**
- User who creates an object can define access column for that object
- We can control domain switching by including domains among the objects of the access matrix
  - Switching from domain D_i to domain D_j is allowed if and only if the access right **switch** is in entry **`access(i, j)`**

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

- Processes can change contents of access-matrix entries
- Special operations: copy, owner, control
  - **Copy**: ability to copy an access right from one domain to another
  - **Owner**: ability to add or remove access rights
  - **Control**: ability to change the entries in a row
  - **Copy** and **Owner** rights applicable to an object
  - **Control** right applicable to a domain



Access Matrix with *Copy* Rights

- Ability to *copy* an access right from one domain to another is denoted by an asterisk(*) appended to the access right
  - Access right can only be copied within column

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

(a)

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

(b)

A process executing in domain D_2 can modify the access matrix of (a) to the access matrix of (b)

Access Matrix With *Owner* Rights

- If `access(i, j)` includes the *owner* right, then a process executing in 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)

The access matrix of (a) can be modified to the access matrix shown in (b)

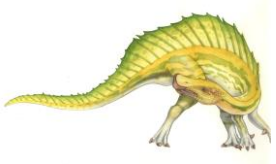
Access Matrix With *Control* Rights

- If `access(i, j)` includes the *control* right, then a process executing in D_i can modify any right from row D_j

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			

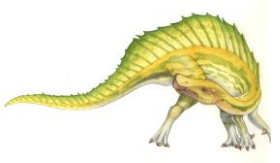
object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

If we include control right here, then a process executing in D_2 can modify D_4, as shown in the lower figure



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
    - System designers and users dictate policy
    - Which domains can access which objects in which ways



Implementation of Access Matrix

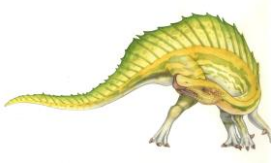
- Generally, a sparse matrix
- Option 1 – Global table
  - Store ordered triples **<domain, object, rights-set>** in table
    - E.g., <D_1, F_1, {read}>, <D_4, F_3, {read, write}>
  - A requested operation op on object O_j within domain D_i → search table for triple < D_i, O_j, R > with op ∈ R
    - If triple is found, the operation is allowed
    - Otherwise, an exception condition is raised
  - But table could be large → won't fit in main memory
  - Difficult to group objects or domains (consider an object that all domains can read)

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			

Implementation of Access Matrix (Cont.)

- Option 2 – Access lists for objects
  - Each column implemented as an access list for one object
  - Per-object list consists of ordered pairs **<domain, rights-set>** defining all domains with non-empty set of access rights for the object
    - E.g., access list for F_1= [<D_1, {read}>, <D_4, {read, write}>]
  - Operation op on object O_j is attempted in domain D_i → search the access list for O_j for entry < D_i, R > with op ∈ R
    - If entry is found, the operation is allowed. Otherwise, access is denied
  - Can be extended to contain default set of access rights → If op ∈ default set, also allow access

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			



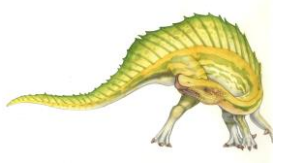
Implementation of Access Matrix (Cont.)

- Option 3 – Capability list for domains
  - Instead of object based, list is domain based
  - **Capability list** for a domain is a list of objects together with operations allowed on those objects

    Object F1 – read

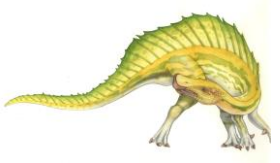
    Object F4 – read, write, execute

    Object F5 – read, write, delete, copy
  - Object represented by its name or address, called a **capability**
  - To execute operation op on object O_j, process requests operation op and specifies capability (or pointer to capability) for O_j as a parameter
    - Possession of capability means access is allowed
  - Capability list associated with domain but never directly accessible to a process executting in that domain
    - Capability list is a protected object, maintained by OS and accessed by user indirectly
    - Capabilities like a “secure pointer”



Capability-Based Systems

- Capabilities are used in Linux and Android
  - Essentially slices up root privileges into distinct areas, each represented by a bitmap bit
  - Fine grain control over privileged operations can be achieved by setting or masking the bitmap
    - Process or thread starts with the full set of permitted capabilities, voluntarily decreases set during execution
    - Essentially a direct implementation of the principle of least privilege
- List of Linux capabilities can be found at http://man7.org/linux/man-pages/man7/capabilities.7.html

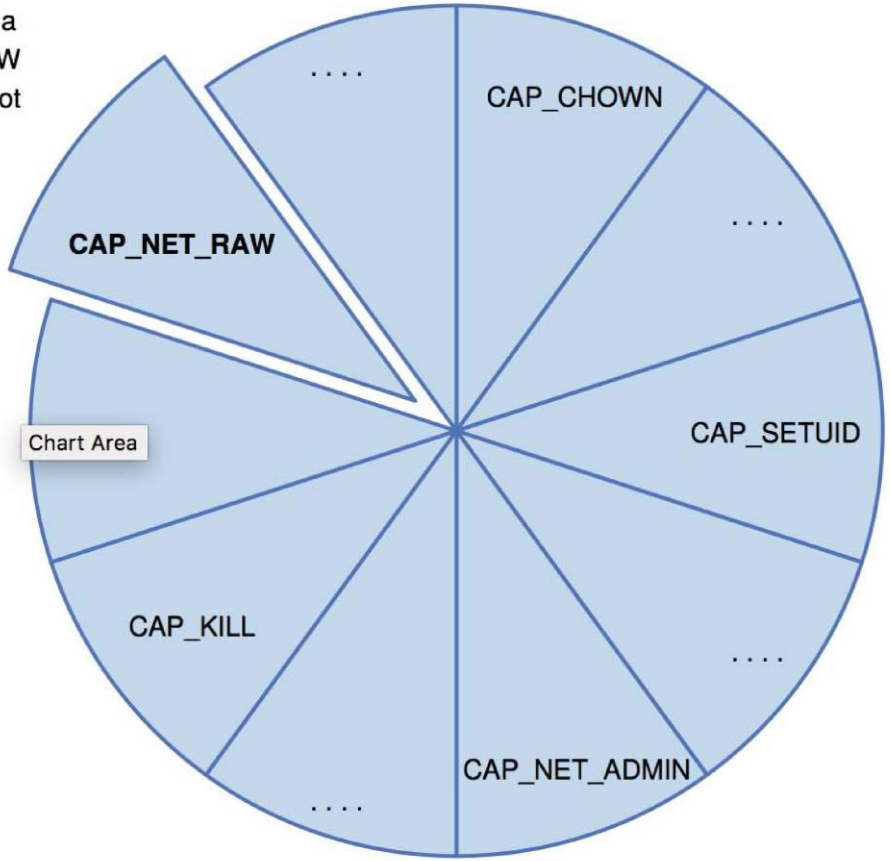


Capabilities in Linux

In the old model, even a simple `ping` utility would have required root privileges, because it opens a raw (ICMP) network socket

Capabilities can be thought of as "slicing up the powers of root" so that individual applications can "cut and choose" only those privileges they actually require

With capabilities, ping can run as a normal user, with CAP_NET_RAW set, allowing it to use ICMP but not other extra privileges



Comparison of Implementations

- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - When a user creates an object, he can specify which domains can access the object, and what operations are allowed
    - Determining set of access rights for each domain is difficult as information is non-localized
    - Every access to an object must be checked
      - When access list is long, search will be slow
  - Capability lists useful for localizing information for a given process
    - Process attempting access must present a capability for that access

Comparison of Implementations (Cont.)

- Most systems use combination of access lists and capabilities
  - First access to an object → access list searched
    - If allowed, capability created and attached to process
      - Additional accesses need not be checked
    - After last access, capability destroyed
  - Consider file system with an access list per file
    - When a process opens a file, access permission is checked
    - If access is allowed, per-process open-file table entry created, entry has capability for allowed operations (e.g., read-only, write-only, read-write)