Protection and Security in Operating System

Tej Bahadur Shahi
CDCSIT, TU

System Protection

- Protection refers to strategy for controlling the access of program, processes, and users to resources.
- It ensure that resources are used in consistent ways
- Further more, it ensure that each resource is accessed correctly and only by those process that are allowed to do so.

System Protection

- If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it.
- So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc.

Example of Protection

- **Privileged instructions** – the process must be executing in kernel mode in order to execute without causing an exception.
- **Memory protection** – the kernel address space is protected from user level instructions. Similarly one process' address space is protected from access by another.
- **File system** – one user's files are protected from access by another user.

Goals of Protection

- to prevent malicious misuse of the system by users or programs.
- To ensure that each shared resource is used only in accordance with system *policies.*
- To ensure that errant programs cause the minimal amount of damage possible.

System protection only provide the *mechanisms* for enforcing policies and ensuring reliable systems. It is up to administrators and users to implement those mechanisms effectively.

Principles of Protection

- ***Principle of least privilege :***
  - dictates that programs, users, and systems be given just enough privileges to perform their tasks.
  - Ensure a little damage even if system fails
- Example:
  - if a program needs special privileges to perform a task, it is better to make it a SGID program with group ownership of "network" or "backup" or some other pseudo group, rather than SUID with root ownership

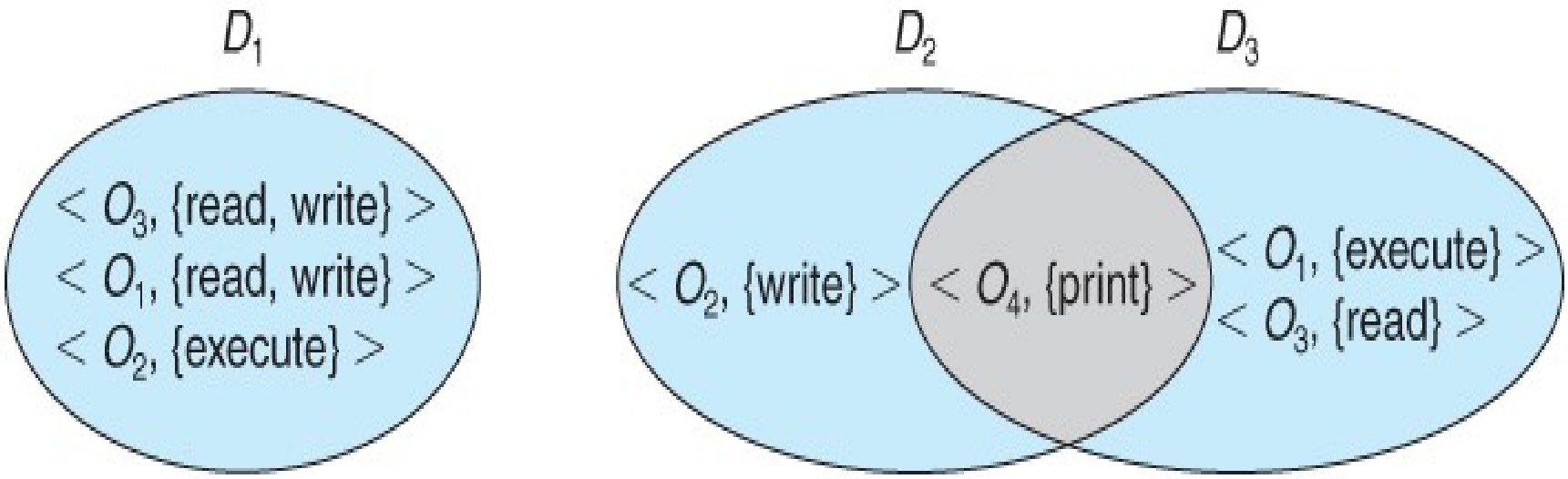
Domain of Protection

- Computer System is consists of objects and process
- Objects consists of both hardware objects (CPU, Memory)and software objects(files and programs)
- Each domain defines a set of objects and the types of operations that may be invoked on each object.

Domain of Protection

- An ***access right*** is the ability to execute an operation on an object.
- A domain is defined as a set of
  - < object, { access right set } > pairs
  - Example if domain D has the access right <file F, {read, write}> then a process executing on domain D can both read and write file F. However, it can't perform other operation on it.

Domain of Protection



Domain can overlap: Permission in overlap are available to both domains.

Protection Domain

- A domain can be realized in various way:
  - Each user may be a domain. In this case the set of object that can be accessed depends upon the identity of user.
  - Each process may be a domain. In this case the set of object that can be accessed depends upon the identity of process.

Crossing Domain

- We want users to have controlled access to resources they don't have direct access to.
- e.g. a database, particular hardware, networks
- So we give the user access to a program that does have access to the restricted resource.

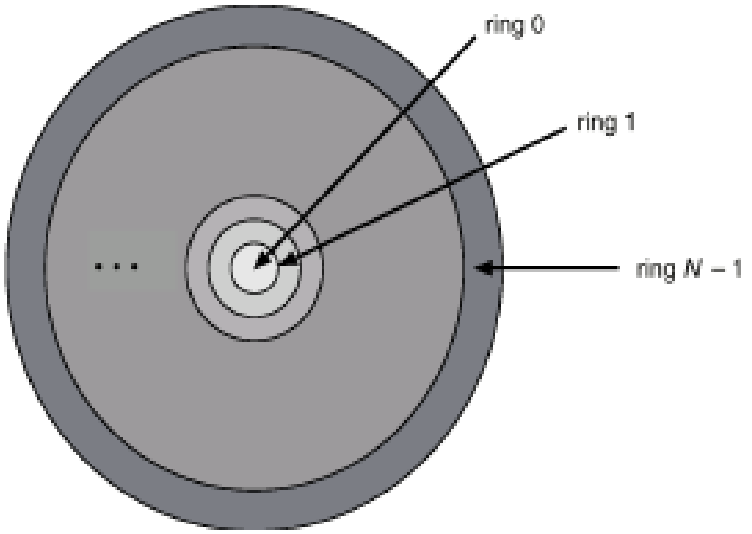
Crossing domains is dangerous and is commonly used to attack systems.

An Example: UNIX

- UNIX associates domains with users.
- Here users ID is used to identify the domains

Multics ring structure: Domain Protection

- Let *Di and Dj be any two domain* rings.
- If $j < i \Rightarrow Di \subseteq Dj$ *Then a* process executing in *Dj has more* privileges than one executing in *Di.*



Access Matrix

- ***A model of system protection called access matrix,***
  - Where columns represent different system resources and rows represent different protection domains.
  - Entries within the matrix indicate what access that domain has to that resource.

Example

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	

Domain Switching

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			

Access Matrix : Owner Right

- The ***owner*** right adds the privilege of adding new rights or removing existing ones:

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

(a)

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

(b)

Implementation of Access Matrix

- It Contains a set of ordered triples
  <domain,object,rights-set>
- Whenever an operation M is executed on an object Oj within domain Di the global table is searched for a triple **<Di,Oj,R_k>** where M Є R_k

- If this triple is found, the operation is allowed to continue or else an exception is raised.

Implementation of Access Matrix

- **Global Table**
  - The simplest approach is one big global table with < domain, object, rights > entries.
  - Unfortunately this table is very large (even if sparse) and so cannot be kept in memory (without invoking virtual memory techniques.)
  - There is also no good way to specify groupings - If everyone has access to some resource, then it still needs a separate entry for every domain.

Implementation of Access Matrix

- **Access Lists for Objects**
  - Each column implemented as an access list for one object
  - Resulting per-object list consists of ordered pairs < *domain, rights-set* > defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> If M ∈ default set, also allow access.

Access List (cont…)

- Each column = Access-control list for one object .
- Defines who can perform what operation

Domain 1 = Read, Write
Domain 2 = Read
Domain 3 = Read

Implementation of Access Matrix

- **Capability Lists for Domains**
  - Instead of object-based, list is domain based
  - **Capability list** for domain is list of objects together with operations allows on them
  - Object represented by its name or address, called a **capability**
  - Execute operation M on object O_j, process requests operation and specifies capability as parameter
    - Possession of capability means access is allowed
  - Capability list associated with domain but never directly accessible by domain
    - Rather, protected object, maintained by OS and accessed indirectly
    - Like a “secure pointer”
    - Idea can be extended up to applications

Capability List(cont..)

- Each Row = Capability List (like a key)
  For each domain, what operations allowed on what objects

  Object F1 – Read

  Object F4 – Read, Write, Execute

  Object F5 – Read, Write, Delete, Copy

Implementation of Access Matrix

A Lock-Key Mechanism

- Each resource has a list of unique bit patterns, termed locks.
- Each domain has its own list of unique bit patterns, termed keys.
- Access is granted if one of the domain's keys fits one of the resource's locks.
- Again, a process is not allowed to modify its own keys.

Comparison of Implementations

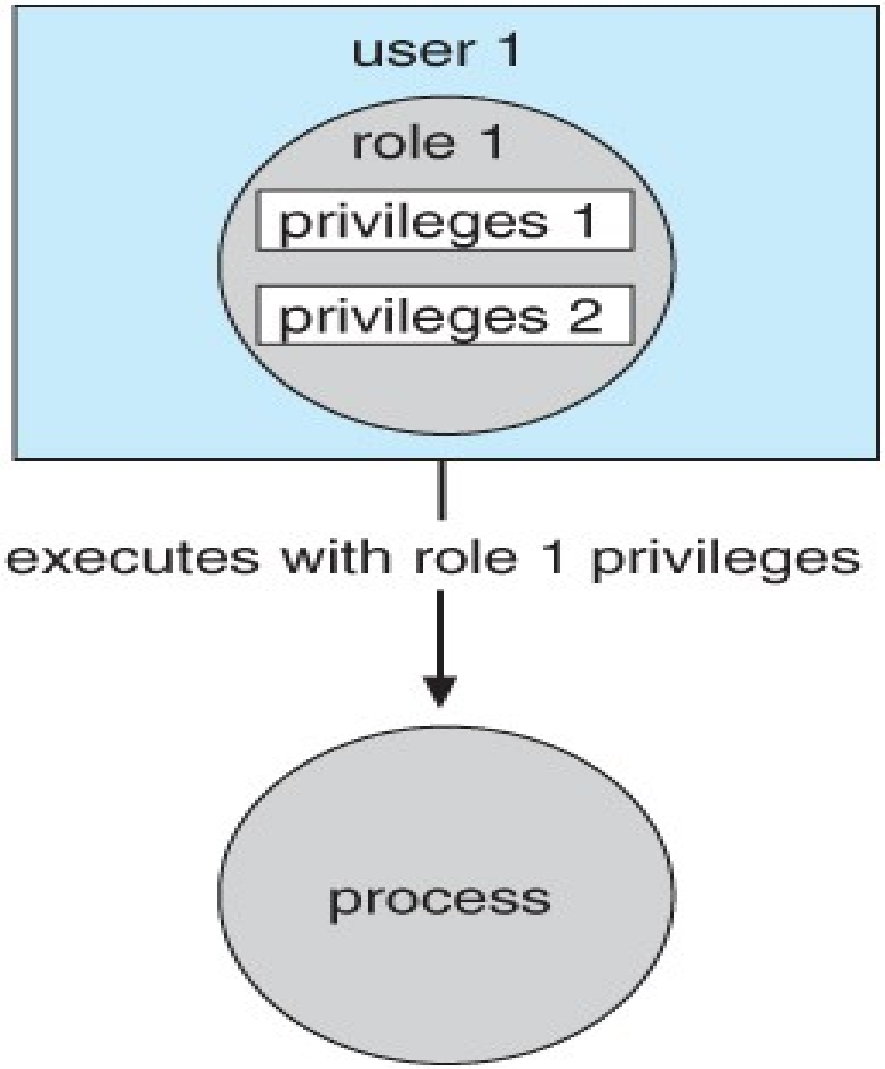
- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - Determining set of access rights for domain non-localized difficult
    - Every access to an object must be checked
      - Many objects and access rights -> slow
  - Capability lists useful for localizing information for a given process
    - But revocation capabilities can be inefficient
  - Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

Access Matrix Implementation

- Each method has advantage and disadvantages
- Many commercial system uses combination of these system

Access Controls

- ***Role-Based Access Control, RBAC,*** assigns privileges to users, programs, or roles as appropriate, where "privileges" refer to the right to call certain system calls, or to use certain parameters with those calls.
- RBAC supports the principle of least privilege, and reduces the susceptibility to abuse as opposed to SUID or SGID programs



Role based access control in solaris 10