Chapter 14: Protection

Zhi Wang
Florida State University

Content

- Goals of protection
- Principles of protection
- Access matrix
- Access control
- Capability-based systems
- Language-based protection

Objectives

- Discuss the 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
- Examine capability and language-based protection systems

Goals of Protection

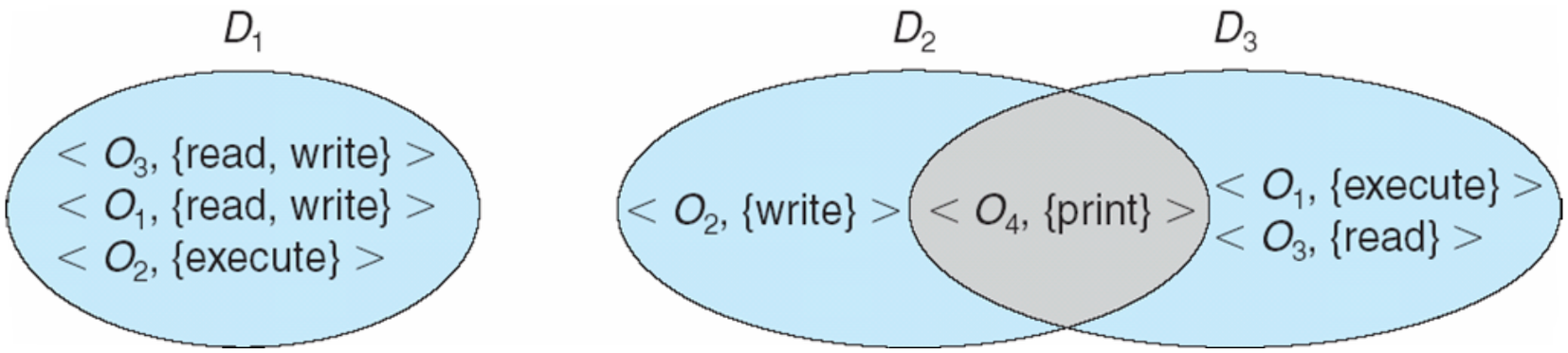
- **Protection**: ensure that OS object is accessed correctly and only by those processes that are allowed to do so
  - computer consists of a collection of objects, in hardware or software
    - each object has a unique name
    - each object can be accessed through a well-defined set of operations
    - e.g., directories and files, keyboard, network…

Principles of Protection

- Guiding principle: **principle of least privilege**
  - an entity should be given just enough privileges to perform their tasks
    - it limits damage if the entity has a bug, gets abused
    - it can be static (during life of system, during life of process)
    - or dynamic (changed as needed) – domain switching, privilege escalation
  - similar to “need to know”, an concept regarding access to data
- Privilege management can be coarse-grained or fine-grained
  - coarse-grained is simpler, but less precise
    - e.g., a Unix processes either have the role of the user or the root
  - fine-grained is more complex, higher overhead, but more securer
    - file ACL lists, RBAC (role-based access control)
- Domain of control can be user, process, procedure

Domain Structure

- Access-right: <object-name, rights-set>
  - **rights-set**: subset of all valid operations that can be performed on the object
- **Domain** is a set of access-rights associated with an subject (entity)

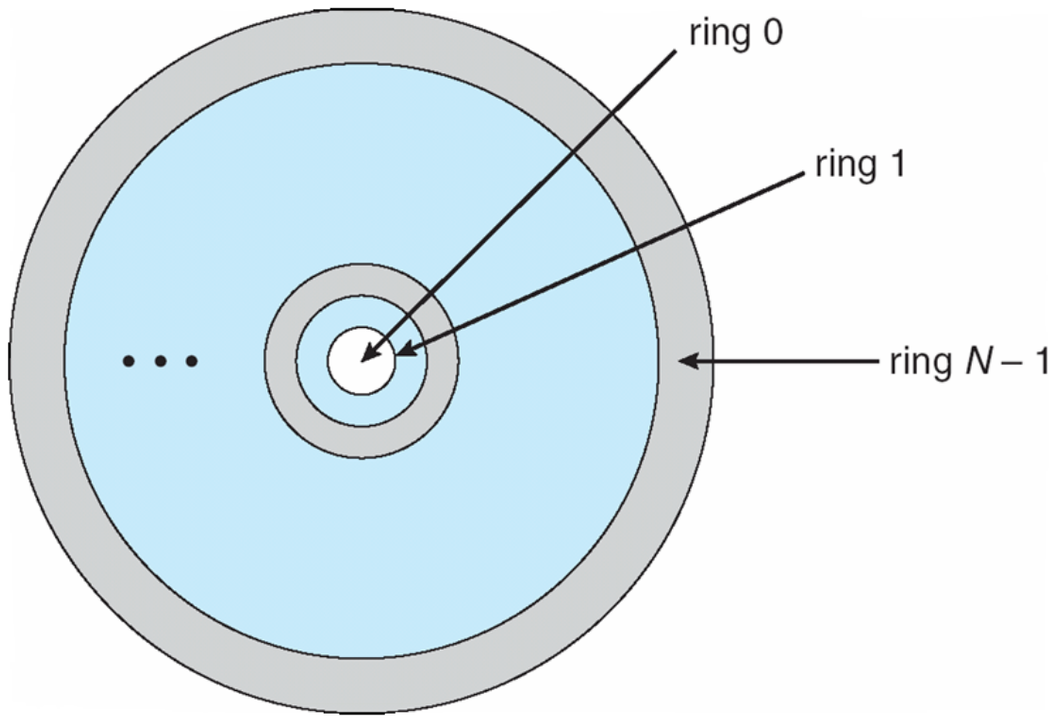


Domain Implementation (UNIX)

- In Unix, domain is based on the user-id
- Domain can be switched via:
  - **password**
    - switch user via the **su** command and his/her password
  - **file system** (file attribute)
    - executable file with **setuid** = on
    - when executed, setuid program will assume the user-id of the file owner
    - why it is necessary?
  - **command**
    - use sudo to executes a command in another domain, if original domain has privilege or password given

Domain Implementation (MULTICS)

- Let D_i and D_j be any two domain rings, if $j < i$ $D_i \subseteq D_j$



Multics Benefits and Limits

- Ring structure is more powerful than kernel/user or root/user design
- It is fairly complex ➠ more overhead
- It is flexible enough to provide strict least-privilege
  - object accessible in D_j but not in D_i, then j must be < i
  - then every segment accessible in D_i also accessible in D_j

Access Matrix

- View protection as a matrix (access matrix)
  - rows represent domains
  - columns represent objects
  - access(i, j): set of operations that a process executing in domain i can invoke on object j
- Access control can be **discretionary** or **mandatory**
  - **DAC**: user who creates object can define access column for that object
  - **MAC**: sys admin determines the access matrix, user cannot modify it

Access Matrix

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

- Validation of access right:
  - a process in domain i tries to do “op” on object j
  - “op” is allowed only if access(i, j) contains “op”
- Access matrix can be expanded to dynamic protection
  - define operations to add, delete access rights
    - **owner**: return the owner of o_i
    - **copy**: copy op from o_i to o_j
    - **control**: d_i can modify d_j access rights
    - **transfer**: switch from domain d_i to d_j
  - note: copy and owner is applied to an object, control and transfer to domain

Use of Access Matrix

- Access matrix design separates mechanism from policy
  - **mechanism**
    - kernel provides access-matrix + rules
    - it ensures matrix can only be manipulated by authorized entities
    - kernel strictly enforce the rules
  - **policy**
    - user define the matrix: who can access what object and in what mode

Access Matrix w/ Switch Rights

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 with Copy Rights

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)

copy right marked with *; a → b if d_2 copy read to d_3

Access Matrix With Owner Rights

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)

owner right marked with *; owner can change access to object

Access Matrix w/ Control Rights

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			

Implementation of Access Matrix

- Access matrix is usually a sparse matrix
- Option 1: **global table**
  - store ordered triples < domain, object, rights-set > in table
  - need to search the table
    - the table could be large ➠ won't fit in main memory
      - difficult to group objects (consider an object that all domains can read)
- Option 2: **per-object access list**
  - each column implemented as an access list for one object
  - access list: a list of domains with non-empty set of access rights to object

Implementation of Access Matrix

- Option 3: **per-domain capability list**
  - instead of object-based, capability list is domain based
  - capability list: a list of objects and operations the domain can operate with
    - object represented by its name or address, called a capability
  - capability list associated with a domain, but cannot be changed by it
    - otherwise, the domain can just add the capability to its list…
    - it can only be changed by a trusted entity
- Option 4: **lock-key**
  - compromise between access lists and capability lists
  - each object has list of unique bit patterns, called locks
  - each domain as list of unique bit patterns called keys
  - a domain can access a object if it has a key matching one of the locks

Implementation of Access Matrix

- For access matrix
  - each column defines an access-control list for the object: domain, operation
  - each row defines a capability list (like a key): object, operation

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	

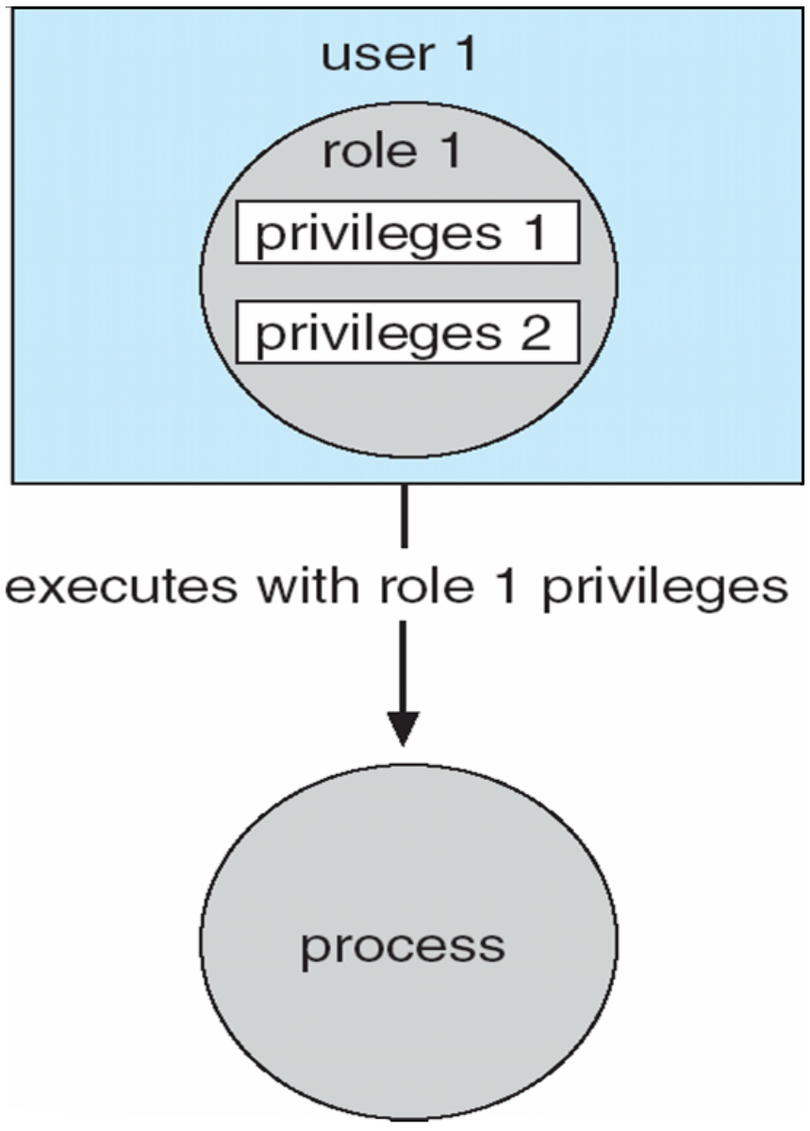
Comparison of Implementations

- Many trade-offs to consider:
  - global table is simple, but can be large
  - access lists focus on the object
  - capability lists useful for localizing information for a given process
  - …
- 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

Role-based Access Control

- Solaris 10 provides RBAC to implement least privilege
  - a role is a group of (closely) related privileges
  - users assigned roles granting access to privileges and programs
    - enable role via password to gain its privileges
  - roles can be assigned to processes

Role-based Access Control in Solaris 10



Revocation of Access Rights

- Various options to remove the access right of a domain to an object
  - immediate vs. delayed
  - selective vs. general
  - partial vs. total
  - temporary vs. permanent

Protection in Java 2

- Protection is handled by the Java Virtual Machine (JVM)
  - a class is assigned a protection domain when it is loaded by the JVM
  - protection domain indicates what operations the class can/cannot perform
  - if an invoked method requires privilege, the call stack is inspected to for access right check

Stack Inspection

protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any

class:	gui:
	...
	get(url);
	open(addr);
	...
		get(URL u):
		...
		doPrivileged {
		open('proxy.lucent.com:80');
		}
		<request u from proxy>
		...
			open(Addr a):
			...
			checkPermission
			(a, connect);
			connect (a);
			...

End of Chapter 14