CHAPTER 14 - SYSTEM PROTECTION

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
- Examine capability and language-based protection systems

GOALS OF PROTECTION

GOALS OF PROTECTION

- In one protection model, computer consists of a collection of objects, hardware or software
- 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

MECHANISM VS POLICY

mechanisms are distinct from **policies**.

- Mechanisms determine how something will be done
- Policies decide what will be done.

The separation of policy and mechanism is important for flexibility.

PRINCIPLES OF PROTECTION

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
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – domain switching, privilege escalation
  - “Need to know” similar concept with access to data

PRINCIPLES OF PROTECTION

- 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
    - File ACL lists, RBAC
- Domain can be user, process, procedure

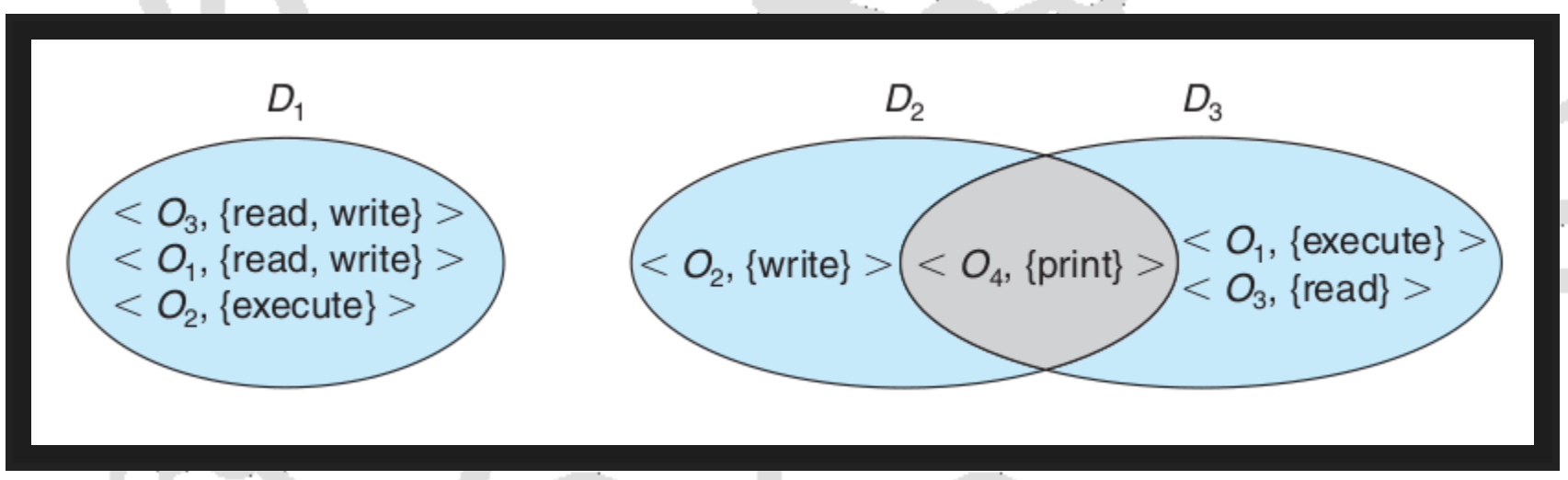
DOMAIN OF PROTECTION

RULE OF THUMB

- A process should be allowed to access only those resources for which it has authorization
  - **Need to know principle**: At any time, a process should be able to access only those resources that is currently required to complete its task

DOMAIN STRUCTURE

- 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



DOMAIN IMPLEMENTATION (UNIX)

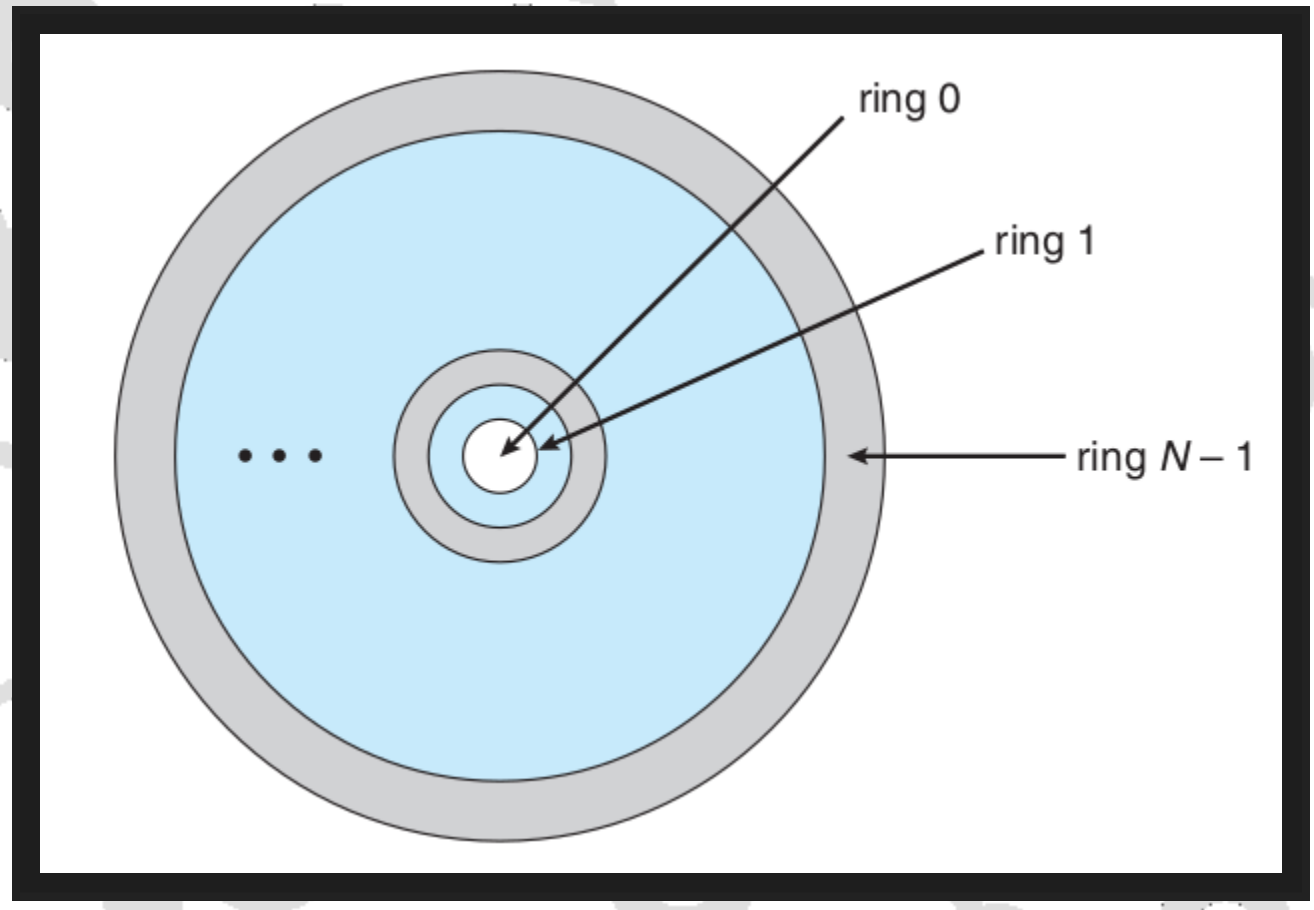
- Domain = user-id
- Domain switch accomplished via file system
  - Each file has associated with it a domain bit (setuid bit)
  - When file is executed and setuid = on, then user-id is set to owner of the file being executed
  - When execution completes user-id is reset

DOMAIN IMPLEMENTATION (UNIX)

- 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)

DOMAIN IMPL. (MULTICS)

- Let D i and D j be any two domain rings
- If j < I ⇒ D i ⊆ D j



MULTICS BENEFITS AND LIMITS

- Ring / hierarchical structure provided more than the basic kernel / user or root / normal user design
- Fairly complex → more overhead
- But does not allow strict need-to-know
  - Object accessible in D j but not in D_i , then j must be < i
  - But then every segment accessible in D_i also accessible in D_j

ACCESS MATRIX

ACCESS MATRIX

- View protection as a matrix (access matrix)
- Rows represent domains
- Columns represent objects
- Access(i, j) is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$

ACCESS MATRIX

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 access matrix
- User who creates object can define access column for that object

USE OF ACCESS MATRIX

Can be expanded to dynamic protection

- Operations to add, delete access rights
- Special access rights:
  - owner of O_i
  - copy op from O_i to O_j (denoted by “*”)
  - control – D_i can modify D_j access rights
  - transfer – switch from domain D_i to D_j

ACCESS MATRIX 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

- Copy and Owner applicable to an object
- Control applicable to domain object

ACCESS MATRIX W. 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)

ACCESS MATRIX W. 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)

ACCESS MATRIX W. CONTROL

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

- **Access matrix** design separates mechanism from policy
  - **Mechanism**
    - Operating system provides access-matrix + rules
    - If 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

GENERAL CONFINEMENT PROBLEM

- **General confinement problem**: Guaranteeing that that no information initially held by an object can migrate outside of its execution environment

General unsolvable

IMPLEMENTATION OF THE ACCESS MATRIX

IMPLEMENTATION OF AM

Generally, a sparse matrix

We will consider 4 ways

- Global table
- Access lists for objects
- Capability list for domains
- Lock-key

GLOBAL TABLE

- Store ordered triples < domain, object, rights-set > in table
- A requested operation M on object O_j within domain D_i → search table for $<D_i, O_j, R_k>$
  - with $M \in R_k$
- But table could be large → won't fit in main memory
- Difficult to group objects (consider an object that all domains can read)

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 LISTS FOR OBJECTS

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

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

- 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
```

CAPABILITY LIST 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 FOR DOMAINS

- 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

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
- Process in a domain can only access object if domain has key that matches one of the locks

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 so difficult
    - Every access to an object must be checked
    - Many objects and access rights → slow

COMPARISON OF IMPLEMENTATIONS

- 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

COMPARISON OF IMPLEMENTATIONS

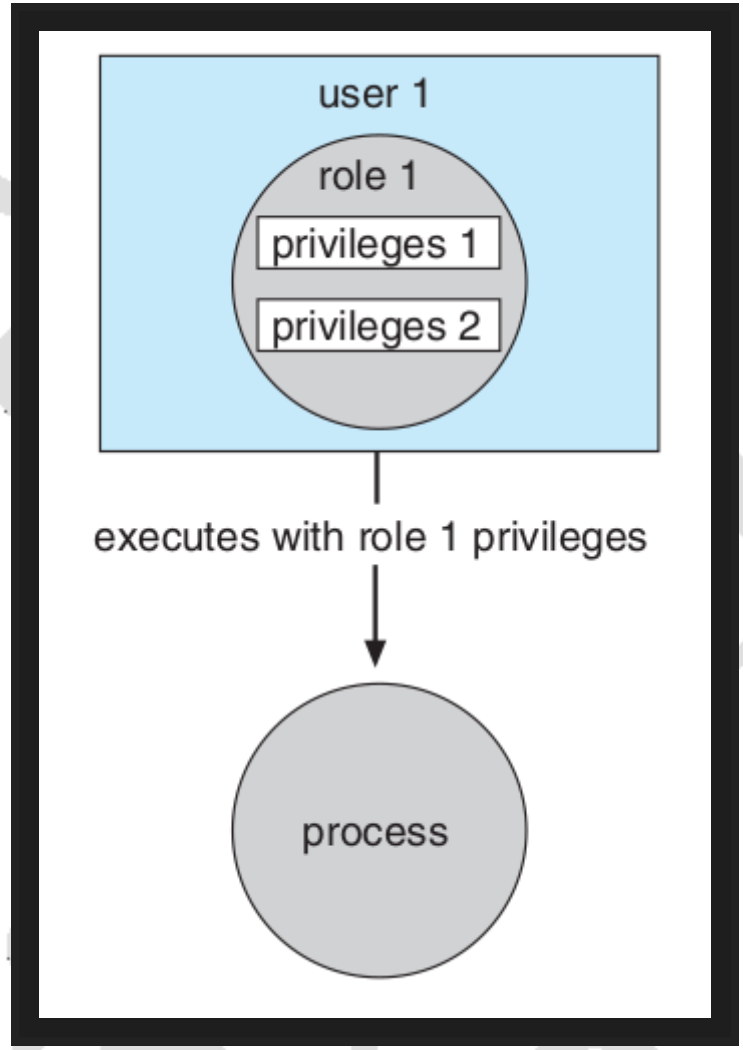
- Most systems use combination of access lists and capabilities
  - First access to an object → access list searched
    - If allowed, capability created and attached to process
      - Additional accesses need not be checked
    - After last access, capability destroyed

ACCESS CONTROL

ACCESS CONTROL

- Protection can be applied to non-file resources
- Solaris 10 provides role-based access control (RBAC) to implement least privilege
  - Privilege is right to execute system call or use an option within a system call
  - Can be assigned to processes
  - Users assigned roles granting access to privileges and programs
    - Enable role via password to gain its privileges
  - Similar to access matrix

RBAC IN SOLARIS 10



REVOCATION OF ACCESS RIGHTS

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

REVOCATION OF ACCESS RIGHTS

- Access List – Delete access rights from access list
  - Simple – search access list and remove entry
  - Immediate, general or selective, total or partial, permanent or temporary

REVOCATION OF ACCESS RIGHTS

- Capability List – Scheme required to locate capability in the system before capability can be revoked
  - **Reacquisition** – periodic delete, with require and denial if revoked
  - **Back-pointers** – set of pointers from each object to all capabilities of that object
  - **Indirection** – capability points to global table entry which points to object – delete entry from global table, not selective

REVOCATION OF ACCESS RIGHTS

- Keys – unique bits associated with capability, generated when capability created
  - Master key associated with object, key matches master key for access
  - Revocation – create new master key
  - Policy decision of who can create and modify keys – object owner or others?

CAPABILITY-BASED SYSTEMS

CAPABILITY-BASED SYSTEMS

- Hydra
- Cambridge CAP System

HYDRA

- Fixed set of access rights known to and interpreted by the system
  - i.e. read, write, or execute each memory segment
  - User can declare other **auxiliary rights** and register those with protection system
  - Accessing process must hold capability and know name of operation
  - **Rights amplification** allowed by trustworthy procedures for a specific type

HYDRA

- Interpretation of user-defined rights performed solely by user's program; system provides access protection for use of these rights
- Operations on objects defined procedurally – procedures are objects accessed indirectly by capabilities
- Solves the problem of mutually suspicious subsystems
- Includes library of prewritten security routines

CAMBRIDGE CAP SYSTEM

- Simpler but powerful
- Data capability - provides standard read, write, execute of individual storage segments associated with object – implemented in microcode
- Software capability -interpretation left to the subsystem, through its protected procedures
  - Only has access to its own subsystem
  - Programmers must learn principles and techniques of protection

LANGUAGE-BASED PROTECTION

LANGUAGE-BASED PROTECTION

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources
- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable
- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system

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
- The protection domain indicates what operations the class can (and cannot) perform
- If a library method is invoked that performs a privileged operation, the stack is inspected to ensure the operation can be performed by the library

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);
			...

QUESTIONS

BONUS

- Exam question number 10: **System Protection**