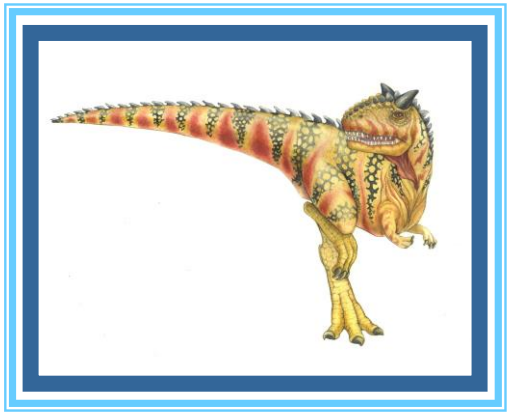
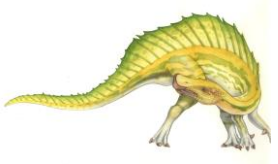
# Chapter 14: Protection



# Chapter 14: Protection

- Goals of Protection
- Principles of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Access Control
- Revocation of Access Rights
- Capability-Based Systems
- Language-Based Protection



# Objectives

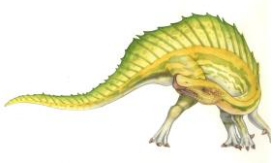
- Discuss the goals and principles of protection in a modern computer system
- Explain how protection domains combined with an access matrix are used to specify the resources a process may access
- Examine capability and language-based protection systems

# Goals of Protection

- 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

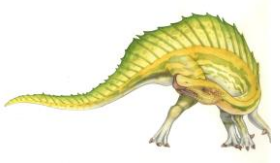
# Principles of Protection

- Guiding principle – **principle of least privilege**
  - Programs, users and systems should be given just enough **privileges** to perform their tasks
  - 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" a similar concept regarding access to data



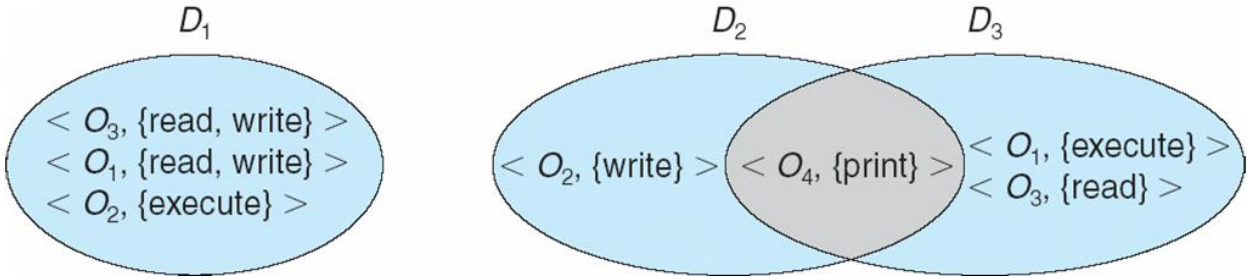
# Principles of Protection (Cont.)

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



# 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



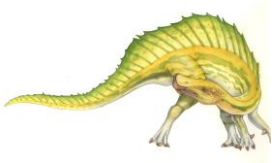

# Domain Implementation (UNIX)

- 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 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

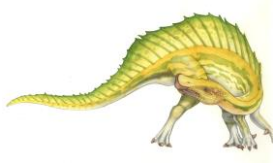
- View protection as a matrix (**access matrix**)
- Rows represent domains
- Columns represent objects
- **`Access(i, j)`** is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$

| object / domain | $F_1$ | $F_2$ | $F_3$ | printer |
|---|---|---|---|---|
| $D_1$ | read | | read | |
| $D_2$ | | | | print |
| $D_3$ | | read | execute | |
| $D_4$ | read<br>write | | read<br>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
- 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* $D_j$
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object



# Use of Access Matrix (Cont.)

- **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

# Access Matrix of Figure A 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 | | | |

# Access Matrix with *Copy* Rights

| 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

| object / domain | $F_1$ | $F_2$ | $F_3$ |
|---|---|---|---|
| $D_1$ | owner<br>execute | | write |
| $D_2$ | | read*<br>owner | read*<br>owner<br>write |
| $D_3$ | execute | | |

(a)

| object / domain | $F_1$ | $F_2$ | $F_3$ |
|---|---|---|---|
| $D_1$ | owner<br>execute | | write |
| $D_2$ | | owner<br>read*<br>write* | read*<br>owner<br>write |
| $D_3$ | | write | write |

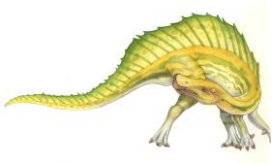
(b)

# Modified Access Matrix of Figure B

| 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 | | | |

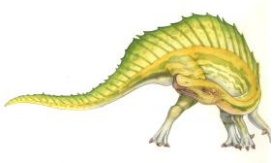
# Implementation of Access Matrix

- Generally, a sparse matrix
- Option 1 – 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 ∈ $R_k$
  - But table could be large -> won't fit in main memory
  - Difficult to group objects (consider an object that all domains can read)



# Implementation of Access Matrix (Cont.)

- Option 2 – 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



# Implementation of Access Matrix (Cont.)

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

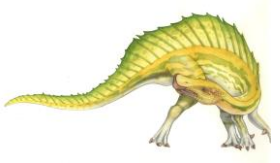
  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

- 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

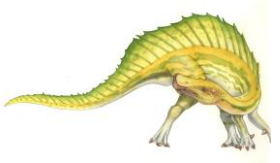


# Implementation of Access Matrix (Cont.)

- Option 3 – 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 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

# 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
  - 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

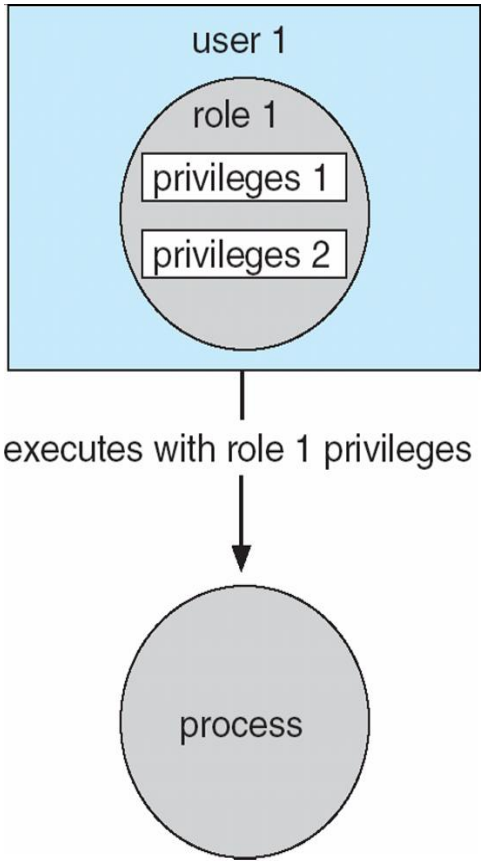


# Comparison of Implementations (Cont.)

- 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 ACLs per file

# Access Control

- Protection can be applied to non-file resources
- Oracle 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




# End of Chapter 14