CS343 - Operating Systems

Module-8A

Protection Services by Operating Systems

Dr. John Jose

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

http://www.iitg.ac.in/johnjose/

Overview

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

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 based protection systems

Goals of Protection

- 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**

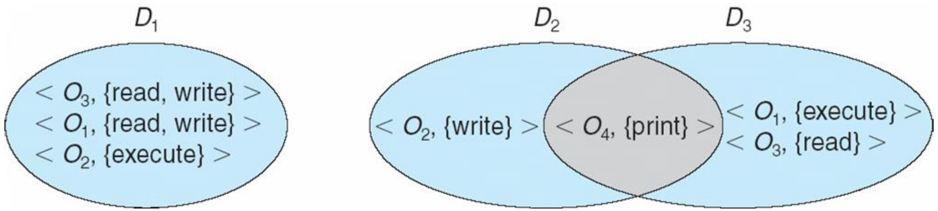
Principles of Protection

- 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 Access Control List (ACL), Roll Based Access Control (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) 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
- If a process in Domain D_i tries to do ***op*** on object O_j, then ***op*** must be in the 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

- 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
    - *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

- **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
    - 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

- A right is copied from access(i, j) to access(k, j); it is then removed from access(i, j). This action is a transfer of a right, rather than a copy.

- Propagation of the copy R* is copied from access(i,j) to access(k,j), only the right R (not R*) is created. A process executing in domain Dk cannot further copy the right R.

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 With Owner Rights

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

Modified Access Matrix with Control Rights

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			

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 control
D_3		read	execute					
D_4	write		write		switch			

- The control right is applicable only to domain objects.
- If access(i, j) includes the control right, then a process executing in domain Di can remove any access right from row j.

Implementation of Access Matrix

- Generally, access matrix is 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 $M \in R_k$.
  - If this triple is found, the operation is allowed to continue; otherwise, an exception (or error) condition is raised
  - 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

- **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
  - When an operation M on an object O_i is attempted in domain D, we search the access list for object O_i, looking for an entry $< D; R_k >$ with $M \in R_k$. If the entry is found, we allow the operation;
  - if it is not, we check the default set. If M is in the default set, we allow the access. Otherwise, access is denied

Implementation of Access Matrix

- Each column = Access-control list for one object
  Defines who can perform what operation
  - Domain 1 = Read, Write
  - Domain 2 = Read
  - Domain 3 = 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 control
D_3		read	execute					
D_4	write		write		switch			

- Each Row = Capability List (like a key)
  For each domain, what operations allowed on what objects
  - Object F1 – Read
  - Object F2 – Read, Write, Execute
  - Object F3 – Read, Write, Delete, Copy

Implementation of Access Matrix

- **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

Implementation of Access Matrix

- **Option 4 – Lock-key**
  - Compromise between access lists and capability lists
  - Each object has list of unique bit patterns, called **locks**
  - Each domain has 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

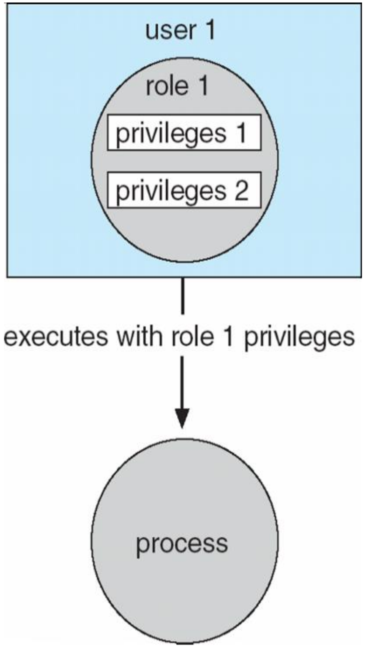
- Global table is simple, but can be large
- Access lists correspond to needs of users
  - Determining set of access rights for domain non-localized so difficult
  - Every access to an object must be checked
  - Many objects and access rights -> slow
- Capability lists useful for localizing information 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

- 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



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**
- **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 (CAL)
  - **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

Capability-Based Systems

- **Hydra - A capability based protection system**
  - 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
  - Includes library of prewritten security routines

Capability-Based Systems

- **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

Thank you

johnjose@iitg.ac.in
http://www.iitg.ac.in/johnjose/