OPERATING SYSTEMS

Access Matrix

Suresh Jamadagni
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

Access Matrix

- A general model of protection can be viewed abstractly as a matrix, called an **access matrix**.
- The rows of the access matrix represent domains, and thecolumns represent objects. Each entry in the matrix consists of a set of access rights.
- Because the column defines objects explicitly, we can omit the object name from the access right.
- The entry access(*i,j*) defines the set of operations that a process executing in domain *Di* can invoke on object *Oj* .

OPERATING SYSTEMS

Access Matrix 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	

- There are four domains and four objects - three files (*F*1, *F*2, *F*3) and one laser printer.
- A process executing in domain *D*1 can read files *F*1 and *F*3.
- A process executing in domain *D*4 has the same privileges as one executing in domain *D*1; but in addition, it can also write onto files *F*1 and *F*3.
- The laser printer can be accessed only by a process executing in domain *D*2.

OPERATING SYSTEMS

Access Matrix

- The access-matrix scheme provides us with the mechanism for specifying a variety of policies.
- The mechanism consists of implementing the access matrix and ensuring that the semantic properties we have outlined hold.
- More specifically, we must ensure that a process executing in domain *Di* can access only those objects specified in row *i*, and then only as allowed by the access-matrix entries.
- The access matrix can implement policy decisions concerning protection.
- The policy decisions involve which rights should be included in the (*i, j*)*th* entry.
- We must also decide the domain in which each process executes. This last policy is usually decided by the operating system.

OPERATING SYSTEMS

Access Matrix – Domain switching

- The access matrix provides an appropriate mechanism for defining and implementing strict control for both static and dynamic association between processes and domains.
- When we switch a process from one domain to another, we are executing an operation (switch) on an object (the domain).
- We can control domain switching by including domains among the objects of the access matrix.
- Similarly, when we change the content of the access matrix, we are performing an operation on an object: the access matrix.
- Again, we can control these changes by including the access matrix itself as an object
- Processes can switch from domain *Di* to domain *Dj* if and only if the access right switch $\in$ access(*i*, *j*)

OPERATING SYSTEMS

Access Matrix – 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			

- A process executing in domain *D*3 or to domain *D*4.
- A process in domain *D*4 can switch to *D*1, and one in domain *D*1 can switch to *D*2.
- Allowing controlled change in the contents of the access-matrix entries requires three additional operations: copy, owner, and control
- The ability to copy an access right from one domain (or row) of the access matrix to another is denoted by an asterisk (*) appended to the access right.
- The copy right allows the access right to be copied only within the column (that is, for the object) for which the right is defined

Implementation of the Access Matrix – Global table

- The simplest implementation of the access matrix is a global table consisting of 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* , *Rk*>, with $M \in Rk$.
- If this triple is found, the operation is allowed to continue; otherwise, an exception (or error) condition is raised.
- This implementation suffers from several drawbacks. The table is usually large and thus cannot be kept in main memory, so additional I/O is needed.
- Virtual memory techniques are often used for managing this table.
- In addition, it is difficult to take advantage of special groupings of objects or domains.
- For example, if everyone can read a particular object, this object must have a separate entry in every domain.

Implementation of the Access Matrix – Access list for objects

- Each column in the access matrix can be implemented as an access list for one object.
- Obviously, the empty entries can be discarded.
- The resulting list for each object consists of ordered pairs <domain, rights-set>, which define all domains with a nonempty set of access rights for that object.
- This approach can be extended easily to define a list plus a ***default*** set of access rights.
- When an operation *M* on an object *Oj* is attempted in domain *Di* , we search the access list for object *Oj* , looking for an entry *<Di , Rk>* with $M \in Rk$.
- 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, and an exception condition occurs

Implementation of the Access Matrix – capability list for domains

- Rather than associating the columns of the access matrix with the objects as access lists, we can associate each row with its domain.
- A **capability list** for a domain is a list of objects together with the operations allowed on those objects.
- An object is often represented by its physical name or address, called a **capability**.
- To execute operation *M* on object *Oj* , the process executes the operation *M*, specifying the capability (or pointer) for object *Oj* as a parameter.
- Simple **possession** of the capability means that access is allowed.
- The capability list is associated with a domain, but it is never directly accessible to a process executing in that domain. Rather, the capability list is itself a protected object, maintained by the operating system and accessed by the user only indirectly.

OPERATING SYSTEMS

Implementation of the Access Matrix – capability list for domains

- Capability-based protection relies on the fact that the capabilities are never allowed to migrate into any address space directly accessible by a user process (where they could be modified).
- If all capabilities are secure, the object they protect is also secure against unauthorized access.

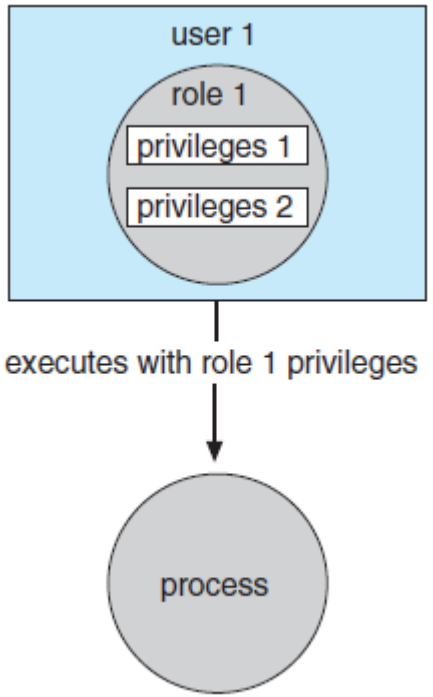
Implementation of the Access Matrix – lock-key mechanism

- The **lock–key scheme** is a compromise between access lists and capability lists.
- Each object has a list of unique bit patterns, called **locks**.
- Similarly, each domain has a list of unique bit patterns, called **keys**.
- A process executing in a domain can access an object only if that domain has a key that matches one of the locks of the object.
- As with capability lists, the list of keys for a domain must be managed by the operating system on behalf of the domain.
- Users are not allowed to examine or modify the list of keys (or locks) directly.

OPERATING SYSTEMS

Role based access control

- A privilege is the right to execute a system call or to use an option within that system call (such as opening a file with write access).
- Privileges can be assigned to processes, limiting them to exactly the access they need to perform their work.
- Privileges and programs can also be assigned to **roles**.
- Users are assigned roles or can take roles based on passwords to the roles.
- A user can take a role that enables a privilege, allowing the user to run a program to accomplish a specific task.
- This implementation of privileges decreases the security risk associated with superusers and setuid programs.



Revocation of access rights

In a dynamic protection system, we may need to revoke access rights to objects shared by different users.

- **Immediate versus delayed**. Does revocation occur immediately or later?
- **Selective versus general**. When an access right to an object is revoked, does it affect all the users who have an access right to that object, or can we specify a select group of users whose access rights should be revoked?
- **Partial versus total**. Can a subset of the rights associated with an object be revoked, or must we revoke all access rights for this object?
- **Temporary versus permanent**. Can access be revoked permanently (that is, the revoked access right will never again be available), or can access be revoked and later be obtained again?

Revocation of access rights

- With an access-list scheme, revocation is easy.
- The access list is searched for any access rights to be revoked, and they are deleted from the list. Revocation is immediate and can be general or selective, total or partial, and permanent or temporary.
- Capabilities, however, present a much more difficult revocation problem.
- Since the capabilities are distributed throughout the system, we must find them before we can revoke them.
- Schemes that implement revocation for capabilities include the following:
  - **Reacquisition**. Periodically, capabilities are deleted from each domain. If a process wants to use a capability, it may find that that capability has been deleted. The process may then try to reacquire the capability. If access has been revoked, the process will not be able to reacquire the capability

Revocation of access rights

- Schemes that implement revocation for capabilities include the following:
  - **Reacquisition**. Periodically, capabilities are deleted from each domain. If a process wants to use a capability, it may find that that capability has been deleted. The process may then try to reacquire the capability. If access has been revoked, the process will not be able to reacquire the capability
  - **Back-pointers**. A list of pointers is maintained with each object, pointing to all capabilities associated with that object. When revocation is required, we can follow these pointers, changing the capabilities as necessary. Implementation of this scheme is costly.
  - **Indirection**. The capabilities point indirectly, not directly, to the objects. Each capability points to a unique entry in a global table, which in turn points to the object. We implement revocation by searching the global table for the desired entry and deleting it

Revocation of access rights

Schemes that implement revocation for capabilities include the following:

- **Keys**. A key is a unique bit pattern that can be associated with a capability. This key is defined when the capability is created, and it can be neither modified nor inspected by the process that owns the capability. A **master key** is associated with each object. Revocation replaces the master key with a new value via the set-key operation, invalidating all previous capabilities for this object

THANK YOU

Suresh Jamadagni
Department of Computer Science Engineering

sureshjamadagni@pes.edu