Chapter 14: Protection

Chapter 14: Protection

- Goals of Protection
- Principles of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix

Goals of Protection

- Operating system 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

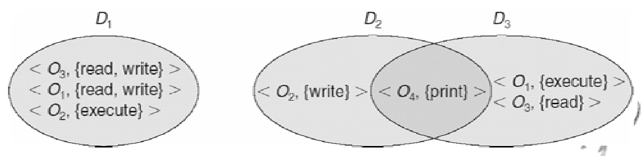
Domain of Protection

- A computer system is a collection of processes and objects.
- By *objects, we mean* both hardware objects(such as the CPU, memory segments, printers, disks, and tape drives) and software objects(such as files, programs, and semaphores).

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.
- A domain is a collection of access-rights
  - Domain need not be disjoint (as shown in fig below)

For example, if domain D has the access right <*file F,* {read, write}>, then a process executing in domain D can both read and write file *F; it cannot, however, perform any other operation on that object.*



Domain Structure

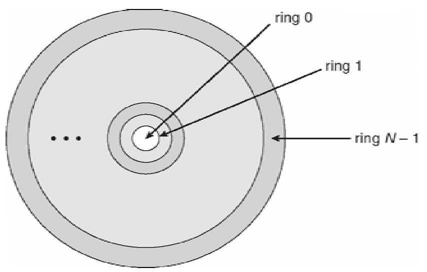
- A domain can be realized in a variety of ways:
  - Each *user may be a domain. In this case, the set of objects that can be* accessed depends on the identity of the user. Domain switching occurs when the user is changed -generally when one user logs out and another user logs in.
  - Each *process may be a domain. In this case, the set of objects that can be* accessed depends on the identity of the process. Domain switching occurs when one process sends a message to another process and then waits for a response.
  - Each *procedure may be a domain. In this case, the set of objects that can be* accessed corresponds to the local variables defined within the procedure. Domain switching occurs when a procedure call is made.

Domain Implementation (UNIX)

- UNIX System consists of 2 domains:
  - User
  - Supervisor (root user)
- Switching the domain corresponds to changing the user identification temporarily
- Domain switch accomplished via file system
  - Each file has associated an owner identification & a domain bit (also known as setuid bit)
  - When the setuid bit is *on, and a user executes that file, the user ID is* set to that of the owner of the file; when the bit is *off, however, the user ID does* not change.

Domain Implementation (MULTICS)

- In the MULTICS system, the protection domains are organized hierarchically into ring structure. Each ring corresponds to a single domain
- Let D_i and D_j be any two domain rings.
- If $j < i$ then D_i is a subset of D_j
- *That is, a process executing in domain D_j* has more privileges than does a process executing in domain D_i. A process executing in domain D_0 has the most privileges.



Domain Implementation (MULTICS)

- MULTICS has a segmented address space; each segment is a file, and each segment is associated with one of the rings.
- When a process is executing in ring *i, it cannot access a segment associated with ring j (j < i). It can access a* segment associated with ring *k (k ≥ i)*

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

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	

we consider the access matrix shown in Figure. There are four domains and four objects-three files (F1, F2, F3) and one laser printer. A process executing in domain D1 can read files F1 and *F3 . A* process executing in domain D4 has the same privileges as one executing in domain D1; but in addition, it can also write onto files F1 and *F3 . Note that the* laser printer can be accessed only by a process executing in domain D2.

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
- 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 (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

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			

Figure B

Switching from domain Di to domain *Dj is allowed if and only if the access right switch ∈* access(i,j). Thus, in Figure , a process executing in domain D2 can switch to domain D3 or to domain D4 . *A process in domain D4 can switch to D1, and* one in domain D1 can switch to D2.

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)

In Figure (a), a process executing in domain D2 can copy the read operation into any entry associated with file *F2 . Hence, the access matrix of Figure (a) can be* modified to the access matrix shown in Figure (b).

Access Matrix with *Copy* Rights

This scheme has two variants:

1. 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.*
2. Propagation of the *copy right may be limited. That is, when the right* 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.*

Access Matrix With *Owner* Rights

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)

In Figure (a), domain D1 is the owner of F1 and thus can add and delete any valid right in column F1. Similarly, domain D2 is the owner of *F2 and F3 and thus can add and remove* any valid right within these two columns. Thus, the access matrix of Figure (a) can be modified to the access matrix shown in Figure (b)

Modified Access Matrix of Figure B

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			

If access(i, *j) includes the control right,* then a process executing in domain Di can remove any access right from row *j.* For example, suppose that, in Figure B, we include the *control right in* access(D2, D4). Then, a process executing in domain D2 could modify domain D4, as shown in Figure above.

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 $M \in R_k$
  - But table could be large -> won't fit in main memory
  - Difficult to group objects (consider an object that all domains can read)
- 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

- 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

Implementation of Access Matrix (Cont.)

- 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**
  - 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
  - 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