PROTECTION

Protection is the mechanism for controlling access to computer resources.

Security concerns the physical integrity of the system and its data and is the subject of the NEXT chapter.

- Goals of Protection are:

  a) Increase reliability of systems that use shared resources

  b) Prevent mischievous activity

  c) Detect malfunctions before they contaminate the system.

MECHANISMS AND POLICIES:

- There's a difference between the **mechanism** and the **policy** it enforces:

  Mechanisms determine **how** to do something.
  Policies decide **what** to do.

- Mechanism must accommodate policies. When policies change, mechanisms should stay constant.

- Think of a system as processes and objects:

  ✓ The processes access objects (hard and soft objects that are abstract data types.)

  ✓ Accesses must be restricted (what can be accessed and how)

  ✓ Determined on need to know basis.

  ✓ When process P invokes procedure A, A should be allowed to access only its own variables, along with the parameters explicitly passed to it;

  ✓ A should NOT be able to access the variables of P.

DOMAIN OF PROTECTION:

- Access rights give the ability to perform an operation on an object.

- A domain is a set of objects and access rights in which a process operates.

- Domains can share access rights; multiple domains can have some access to the same object.

- **EXAMPLE OF UNIX IMPLEMENTATION:**

  System consists of two domains, **user**, and **supervisor**.

  Domain is determined by user ID. Domain switch is controlled by the file system:

  > Each file has associated with it a domain bit (setuid bit).
  >
  > When the 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.

ACCESS MATRIX:

For each domain, delineate all objects and the access rights for them.
<<< FIGURE 19.3 >>>

IMPLEMENTATION OF ACCESS MATRIX

A Simple Matrix

- A matrix is too sparse - it would take far too much space.

Global Table

- Triplet (domain, object, rights)
- Too lengthy; common rights must be duplicated for each domain.
- I/O is required to get the table.
- The needs of a process change over time. But we don't want to establish maximum privileges to start with, since this violates the need-to-know principle.
- We can either modify the domain, or change to a different domain (either an existing one or a just-created one.)
- Can consider a domain to be an object; the operation within a domain is to "switch" from one domain to another. **<<< FIGURE 19.4 >>>**
- We can allow a domain copy right - (transfer / copy / limited copy). Operates on a column of the access matrix and thus affects only ONE object.
  <<< FIGURE 19.5 >>>
- Can allow domain "owner" right (can add or delete to any entry in column.)
  <<< FIGURE 19.6>>>
- Can allow domain "control" right (modify the rights of another domain.)
  <<< FIGURE 19.7>>>
- Each resource object must:

  a) Have a manager
  b) Check capabilities of users
  c) Schedule use
  d) Preempt if necessary

ACCESS LISTS

- For each object, list of (domain, rights). This is the **column** in **<<<Figure 19.3>>>**
- Can also have default rights common to all domains.

CAPABILITY LISTS

- For each domain, list of (object, rights). This is the **row** in **<<<Figure 19.3>>>**
- The process executes an operation by specifying the capability (pointer) for the object.
- Possession of the capability means access is allowed.
- This list must be protected to avoid user modification. (For instance, the system can’t hand a user a pointer to a file; instead the handle must be tagged to indicate it is unchangeable, or it must be kept in read-only space.)

LOCK / KEY

- The object has bit patterns (locks); each domain has bit patterns (keys).
- A process executing in a domain can access an object only if that domain has the right key.
- Again, this system must be protected from user modification.

COMPARISON

Access List	Corresponds with needs of users.
	Hard to find the set of access rights for a domain.
	Every access must be checked - could be long search.
Capability List	Harder to set up, easier to check when used.
	No search of list is necessary; only verification of capability by the system.
Lock and Key	Compromise between the above.
	In addition, it's easy to distribute and revoke.

- Most systems use a combination of access and capability.
- When a process first references an object, an access list is checked.
- If successful, a capability is given to the domain so that the process can use it thereafter.

An access matrix is a **mechanism** upon which numerous **policies** can be built.

REVOCATION:

- It may be necessary to revoke/remove rights to objects shared by a number of users.

- Issues involved in revocation:

  a) Immediate or delayed (if delayed, when?)

  b) Selective or general (subset of users, or all users)

  c) Partial (subset of rights) or all of them.

  d) Temporary or permanent (once revoked, can they be reinstated?)

- With an access list, revocation is easy.

- With capabilities, revocation is much harder.

- Example of opened file - how can you suddenly stop user reads once you've handed out the capability?

- Possible Solutions:

  Periodic deletion from domain -- user needs to reacquire before reuse.

  Back pointers from an object to all capabilities pointing to it. Very general, but very expensive.

  Indirect: Maintain a list of pointers to objects; capabilities link to these pointers rather than the objects themselves.

  Then must search list of pointers in order to revoke; the method is not selective.

  Keys: Hand out master key for each object. Change locks to revoke.

  The method is not selective unless there are multiple keys associated with an object.

  Alternatively, can use the same key with several objects and use a pool of valid keys.

  Here domains may be user, process, or procedure.