Silberschatz, et al. Topics based on Chapter 18

Protection

Protection

- Goals of protection schemes
- Domain of protection
- Mechanisms
  - access matrix
    - implementation of access matrix
    - revocation of access rights
  - Capability-based systems
  - Language-based protection

Goals of protection schemes

- Operating system consists of a collection of hardware and software objects
  - CPU, memory segments, printers, disks, tape drives
  - files, programs, semaphores
- Each object has a unique name; is accessed through a well-defined set of operations
  - Essentially *abstract data types*
- Purpose of protection: to ensure that each object is accessed correctly and only by those processes that are allowed to do so
  - *need to know* principle

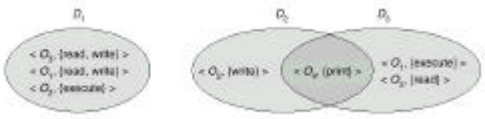
Goals of protection schemes

- Why protection?
  - Prevent mischief
  - Make sure that program components use resources in compliance with policies for resources
  - Protect from certain user errors
- Separation of *policy* from *mechanism*
  - Policy: *what* will be done
  - Mechanism: *how* it will be done
- Separating policy from mechanism allows change to policy without requiring changes to underlying mechanism (reconfiguration instead)

Protection domain structure

- *Protection domain*--specifies the resources that a process may access. Defines a set of objects and the operations that may be invoked on each object. A domain is a set of access rights
- *Access right*--the ability to execute an operation on an object; a subset of all valid operations that can be performed on the object
  - *<object-name, rights-set>*
- Domains can share access rights

Protection domain structure



Protection domain structure

- Association between a process and a domain may be static or dynamic
  - static: set of resources available to the process is fixed through the process' lifetime
  - static is easier to implement than dynamic
  - Static association plus need-to-know requires mechanisms to change the content of a domain
    - one phase may require read access but not write access
    - another may require only write access
    - need-to-know implies that we provide only the minimum needed access rights at all times
  - Dynamic association provides these means

Protection domain structure

- What defines a domain?
  - Each user is a domain
    - access depends on user's identity
    - domain switching occurs when users change (login/ logout)
  - Each process is a domain
    - access depends on process' identity
    - Domain switching occurs when process sends a message to another and then waits for answer
  - Each procedure is a domain
    - set of objects that can be accessed corresponds to local variables
    - Domain switching occurs when procedure call made

Domain implementation examples

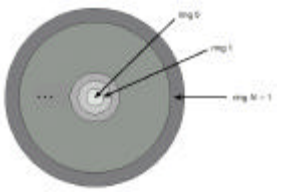
- System consists of 2 domains:
  - User
  - Supervisor

Domain implementation examples

- 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.
  - Some other systems do not allow change of user id. Here, user access to protected objects has to use different mechanisms. For example, a daemon process that mediates access to the object.

Domain implementation examples

- Multics rings (MIT late 1960's)
  - Let D_i and D_j be any two domain rings.
  - If $j < i$ Þ $D_i \subseteq D_j$



Domain implementation examples

- Multics system
  - Ring D_0 corresponds to monitor mode; has the most privileges
  - each memory segment includes ring number and access bits to control reading, writing, and execution
  - process can only access segments associated with rings with greater than or equal number, restricted according to the access bits
  - Domain switching is procedure oriented--procedure called in a different ring. Further controls on how those switches can occur (see following)

Domain implementation examples

- Multics domain switching
  - Makes use of the following
    - access bracket: a pair of integers, b1 and b2, such that b1 ≤ b2
    - limit: an integer b3, such that b3 > b2
    - list of gates: identifies entry points (gates) at which segments may be called
  - Process in ring *i* calls a procedure (segment) with access bracket (b1,b2)
    - Call allowed if b1 i b2
    - Current ring number of the process remains *i*
    - Otherwise, see following

Domain implementation examples

- Multics domain switching
  - When the caller's ring number is not in the callee's access bracket
    - i < b1
      - Call allowed since this is a transfer to a ring with fewer privileges
      - Parameters may need to be copied into an area accessible to the called procedure
    - i > b2
      - Call permitted only if b3 i (b3 is the *limit*) and the call has been directed to one of the designated entry points in the list of gates
      - This is a call to a procedure with higher privileges, but in a controlled manner

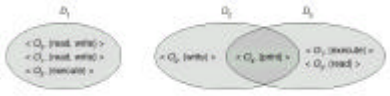
Domain implementation examples

- Multics domain model
  - Does not enforce need-to-know (as you have access to all segments in higher numbered rings)
  - More general models (which are also simpler) used in modern computer systems

Access Matrix

- Rows: domains
- Columns: objects
- Access(i,j) defines the set of operations that a process, executing in domain D_i can invoke on object O_j
- Process in Domain D_i can execute operation *op* on Object O_j only if there is a corresponding entry in the access matrix

Access matrix



	O1	*O2*	*O3*	*O4*
D1	read write	execute	read write	
D2		write		print
D3	execute		read	print

Access Matrix

- Allowing processes to switch among domains
  - Can be controlled by including domains in access matrix
  - "switch" access right allows switching to the specified domain

	O1	*O2*	*O3*	*O4*	*D1*	*D2*	*D3*
D1	read write	execute	read write			switch	
D2		write		print			
D3	execute		read	print	switch	switch	

Access Matrix

- Allowing controlled change to the access matrix
  - Operations to add, delete access rights.
  - Special access rights:
    - owner of object O_i
      - Can add/remove operations in column i
    - copy *op* from D_i to D_j
      - Copy within column (i.e., to additional domains for object for which the right is defined)
      - Variant: transfer of right, not copy
      - Variant: limit propagation (copy cannot be copied)
    - control – D_i can modify D_js access rights
      - D_i can remove access rights from row j

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 Implementation

- Want to implement efficiently but usually matrix is sparse
- Simple implementation: global table of triples <domain, object, rights set>
  - Problems: large table, hence too big for memory (has to be on secondary storage either explicitly or via virtual memory). Requires added I/O
  - Difficult to take advantage of special groupings of objects--for example if an object allows everyone to read it, it must have separate entry in every domain

Access Matrix Implementation

- Object-centric implementation
  - Access list implementation. Columns in access matrix are implemented as an access list, kept by the object (list of <domain, rights-set> pairs)
  - Easy extension also provides default set of access rights (search local list, if operation on object not found check default set)

Access Matrix Implementation

- Domain-centric implementation
  - each row in the access matrix can be implemented as a *capability list* kept by the process (<object, access-rights> list)
  - simple possession of capability means that specified rights are granted
  - manipulation and passing of capabilities has to be implemented by OS--capability-based protection assumes that capabilities never migrate into user space.

Access Matrix Implementation

- Lock-key mechanism
  - Objects have list of unique bit patterns, called *locks*
  - Domains have list of unique bit patterns, called *keys*
  - Process executing in a domain can access an object only if the domain has a key that matches one of the locks of the object
  - As with capabilities, users cannot examine or manipulate locks and keys directly

Revocation of Access Rights

- Access List – Delete access rights from access list.
  - Simple (access list kept in a centralized location)
  - Immediate
- Capability List – Scheme required to locate capability in the system before capability can be revoked (capabilities distributed throughout system).
  - Reacquisition
    - Require reacquisition of capabilities from time to time
  - Back-pointers
    - Keep list of capability holders
  - Indirection
    - Don't give out capabilities; give out pointers to capabilities
- Keys
  - Selectively change locks

Capability-Based Systems

- Hydra
  - CMU, ~1981
  - Fixed set of access rights known to and interpreted by the system.
  - Interpretation of user-defined rights performed solely by user's program; system provides access protection for the use of these rights.

Capability-Based Systems

- Cambridge CAP System
  - Cambridge ~1977
  - *Data capability* – provides standard read, write, execute of individual storage segments associated with object.
  - *Software capability* –interpretation left to the subsystem, through its protected procedures.

Language-Based Protection

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources.
- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable.
- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system.