Chapter 14: Protection

Chapter 14: Protection

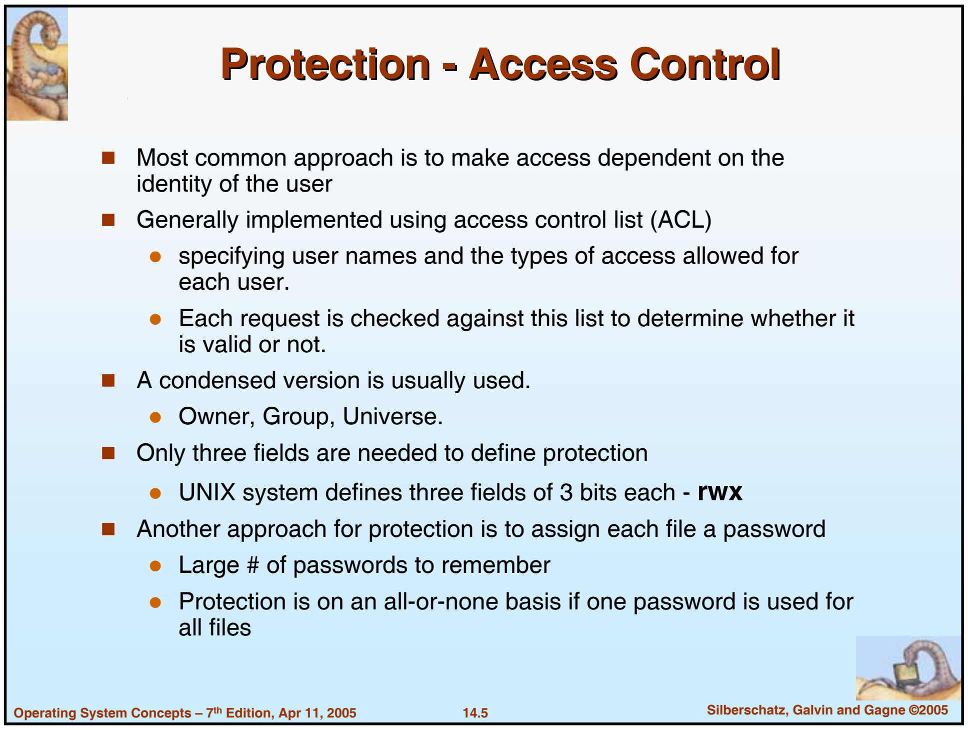
- Goals of Protection
- Principles of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Access Control
- Revocation of Access Rights
- Capability-Based Systems
- Language-Based Protection

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 and language-based protection systems

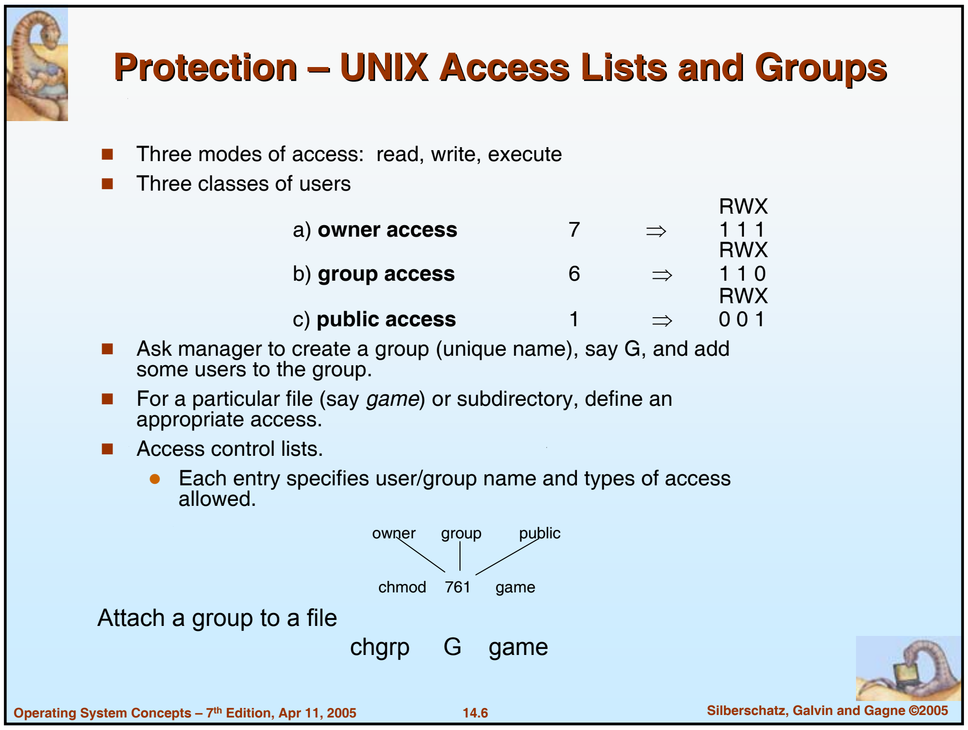
Protection

- The need to protect files is a direct result of the ability to access files.
  - complete protection by prohibiting access➔ not useful !!!!
  - Or, free access with no protection.
  - What is needed is **<u>controlled access</u>**.
- Types of access
  - Read (“r” in Unix)
  - Write (“w”)
  - Execute (“x”)
  - Append (“w”)
  - Delete (owner)
  - List (“r” for a directory)
  - Search (“x” for a directory)
- Other operations (renaming, copying, and editing) use these basic set of operations



Protection - Access Control

- Most common approach is to make access dependent on the identity of the user
- Generally implemented using access control list (ACL)
  - specifying user names and the types of access allowed for each user.
  - Each request is checked against this list to determine whether it is valid or not.
- A condensed version is usually used.
  - Owner, Group, Universe.
- Only three fields are needed to define protection
  - UNIX system defines three fields of 3 bits each - **rwx**
- Another approach for protection is to assign each file a password
  - Large # of passwords to remember
  - Protection is on an all-or-none basis if one password is used for all files



Protection – UNIX Access Lists and Groups

- Three modes of access: read, write, execute
- Three classes of users

			RWX
a) **owner access**	7	⇒	1 1 1
			RWX
b) **group access**	6	⇒	1 1 0
			RWX
c) **public access**	1	⇒	0 0 1

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.
- Access control lists.
  - Each entry specifies user/group name and types of access allowed.

owner group public

chmod 761 game

Attach a group to a file

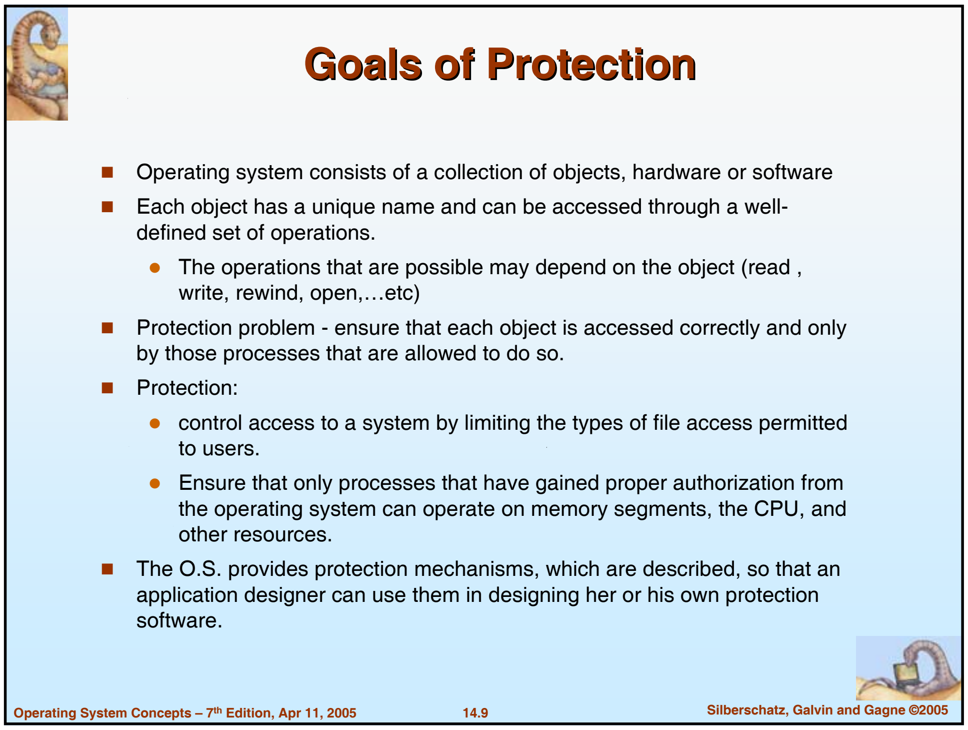
chgrp G game

Protection - A Sample UNIX Directory Listing

- -rwx------ 1 cs4520 users 20096 Sep 15 14:31 a.out
- -rw-r--r-- 1 cs4520 student 1764 Aug 30 16:05 assign1.c
- -rw------- 1 cs4520 users 4624 Aug 30 15:48 assign1.lis
- drwxr-xr-x 2 cs4520 users 8192 Jun 17 2003 bin
- drwx------ 2 cs4520 users 8192 Aug 31 2004 gtest
- -rw-r--r-- 1 cs4520 users 853 Aug 31 08:28 hello.c
- -rw------- 1 cs4520 users 104 Jan 13 2005 nums.bin
- -rw------- 1 cs4520 users 3351 Sep 15 14:32 class.list
- drwxr-xr-x 3 cs4520 student 8192 Dec 9 2004 projects
- drwxr-x--- 4 cs4520 faculty 8192 Oct 30 21:31 test

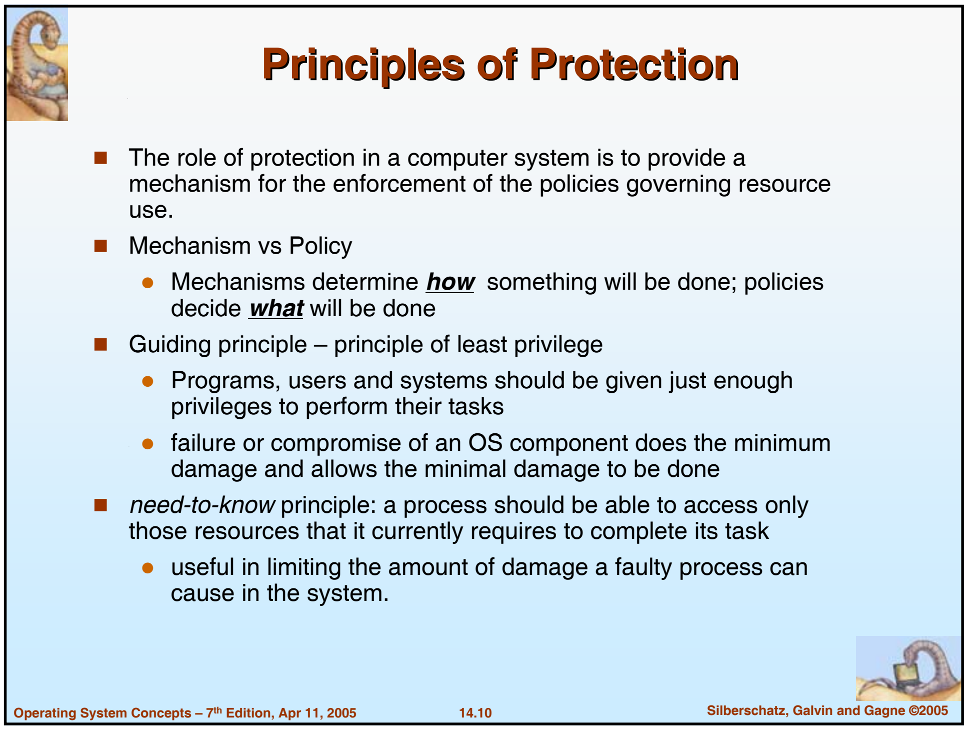
Protection - summary

- File systems implement some kind of protection system
  - Who can access a file
  - How they can access it
- More generally…
  - Objects are “what”, subjects are “who”, actions are “how”
- A protection system dictates whether a given action performed by a given subject on a given object should be allowed
  - You can read and/or write your files, but others cannot
  - You can read “/etc/motd”, but you cannot write it



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.
  - The operations that are possible may depend on the object (read , write, rewind, open,…etc)
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.
- Protection:
  - control access to a system by limiting the types of file access permitted to users.
  - Ensure that only processes that have gained proper authorization from the operating system can operate on memory segments, the CPU, and other resources.
- The O.S. provides protection mechanisms, which are described, so that an application designer can use them in designing her or his own protection software.



Principles of Protection

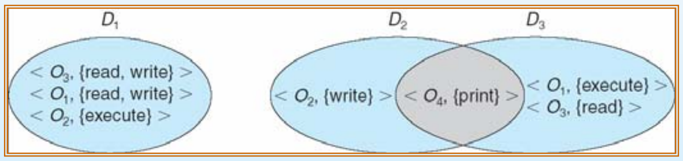
- The role of protection in a computer system is to provide a mechanism for the enforcement of the policies governing resource use.
- Mechanism vs Policy
  - Mechanisms determine ***how*** something will be done; policies decide ***what*** will be done
- Guiding principle – principle of least privilege
  - Programs, users and systems should be given just enough privileges to perform their tasks
  - failure or compromise of an OS component does the minimum damage and allows the minimal damage to be done
- *need-to-know* principle: a process should be able to access only those resources that it currently requires to complete its task
  - useful in limiting the amount of damage a faulty process can cause in the system.

Domain of Protection

- A process operates within a protection domain, which specifies the resources that the process may access.
- Each domain defines a set of objects and the types of operations that may be invoked on each object.
- The ability to execute an operation on an object is an ***access right***.
- A ***domain*** is a collection of access rights, each of which is an ordered pair: <object-name, rights-set>
- Example: If domain D has the access right: **<file F, {read, write}>,** then a process executing in domain D can only read and write file F.

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 = a collection of access-rights
- A protection domain specifies the resources that the process may access



- Domains may share access rights.
  - A process executing in either D2 or D3 can print O4
  - A process must be executing in D1 to read and write O1. Also, only process in D3 may execute O1

Domain Implementation (UNIX)

- System consists of 2 domains:
  - User
  - Supervisor
- UNIX
  - Domain = user-ID
  - Domain switching corresponds to user ID switching
  - Domain switching is accomplished through file system as follows:
    - Each file has associated with it a domain bit (setuid bit) and an owner ID.
    - When a user A starts executing a file owned by user B and the setuid bit is off, the user ID of the process is set to A
    - When setuid = on, then user-id is set to owner of the file being executed: B. When execution completes user-id is reset.

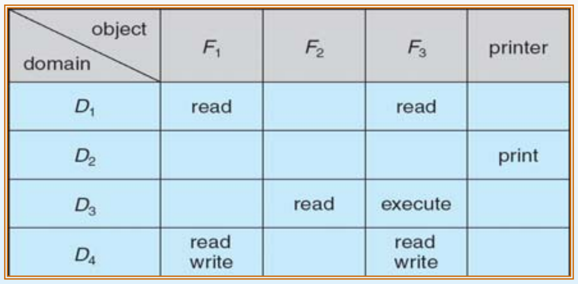
Access Matrix

- View protection as a matrix (*access matrix*)
  - Rows represent domains
  - Columns represent objects
  - Each entry in the matrix consists of a set of access rights.
  - *Access(i, j)* is the set of operations that a process executing in Domain$_i$ can invoke on Object$_j$
- Summary: access list keeps track of which object belongs to which domain
- Note: Most domains have no access at all to most objects, so storing a very large, mostly empty, matrix is a waste of disk space
  - ACL
  - Capability List

Access Matrix - contd

Access Matrix illustration:

- 4 domains and 4 objects (3 files and one printer).
- When a process executes in D1, it can read files F1 and F3.
- A process executing in D4 has the same privileges as it does in D1, it can also write onto files F1 and F3.
- Printer can be accessed by a process executing in D2.



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	

Access Matrix - contd

- Users decide the contents of the access-matrix entries.
  - When a user creates a new object Oj, the column Oj is added to the access matrix with the appropriate initialization entries.
  - The user may decide to enter some rights in some entries in column j and other rights in other entries.
- Access matrix provides mechanism for defining and implementing strict control for both static and dynamic association between processes and domains.
  - Controls changing content of access-matrix entries.
  - Controls switching between domains.

Access Matrix - contd

- 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 objects of the access matrix.
- When we change the content of the access matrix, we are performing an operation on an object which is the access matrix.
  - We can control these changes by including the access matrix itself as an object.
- Processes should be able to **switch** from one domain to another.
  - Domain Switching from Di to Dj is allowed to occur iff access right **switch** Є access(i,j).

Domain Switching: Access Matrix 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			

• A process executing in D4 (row) can switch to D1.
A process executing in D2 (row) can switch to D3 or D4.
A process executing in D1 (row) can switch to D2.

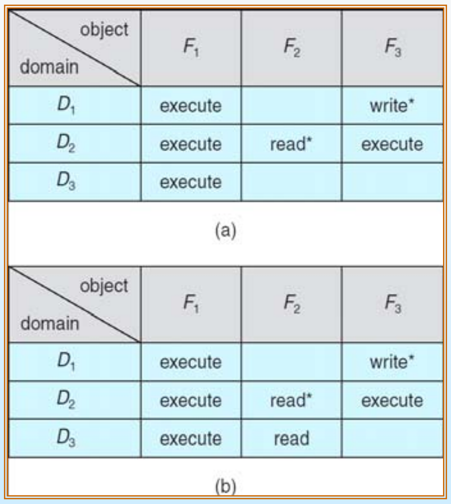
Access Matrix with *Copy* Rights

- Allowing controlled change to the contents of the access-matrix entries requires 3 additional operations:
  - Copy, Owner, and Control.
- The ability to copy an access right from one domain (row) to another is denoted by an asterisk (*) appended to the access right.
- **Copy** right allows the copying of the access right only within the column (that is, for the object) for which the right is defined.

Access Matrix with *Copy* Rights - contd

- A process executing in D2 can copy the read operation into any entry associated with F2.
- The ***copy*** scheme has 3 variants:

1. A right is copied from access(i,j) to access(k,j) is not limited: This action is called ***copy***.
   - When the right Read* is copied from access(i,j) to access(k,j), the Read* is created.
   - So, a process executing in Dk can further copy the right Read*.



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	

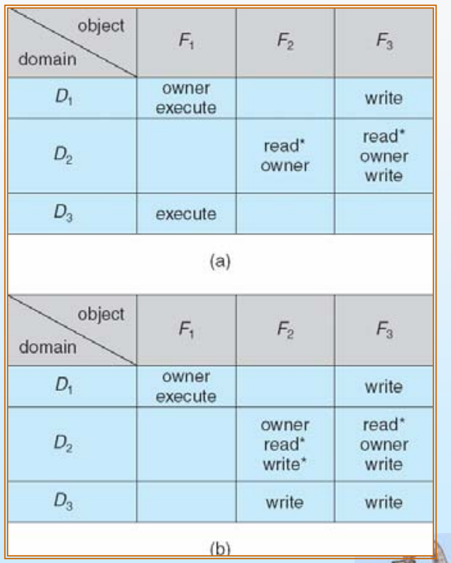
(b)

Access Matrix with *Copy* Rights - contd

2. Propagation of the copy right may be limited: This action is called ***limited copy***.
   - When the right Read* is copied from access(i,j) to access(k,j), only the Read (not Read*) is created.
   - So, a process executing in Dk cannot further copy the right Read.
3. A right is copied from access(i,j) to access(k,j); it is then removed from access(i,j).
   - This action is called a ***transfer*** of a right, rather than a copy

Access Matrix With *Owner* Rights

- We need a mechanism to allow addition of new rights and removal of some rights.
  - The ***owner*** right controls these operations.
  - If access(i,j) includes the owner right, then a process execution in Di can add and remove any right in any entry in column j.
- D1 is the owner of F1, and can add and delete any valid right in column F1.
- D2 is the owner of F2 and F3, and can add and delete any valid right within these 2 columns.



object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

object / domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Access Matrix With *Control* Rights

- The ***copy*** and ***owner*** rights allow a process to change the entries in a column.
  - limit the propagation of access rights
  - However, they do not give us the appropriate tools for preventing the propagation (or disclosure) of information.
- So, a mechanism is needed to change the entries in a row.
  - The ***control*** right is applicable only to domain objects (rows).
  - If access(i,j) includes the ***control*** right, then a process executing in Di can remove any access right from row j.

Access Matrix With *Control* Rights

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 control
D_3		read	execute					
D_4	write		write		switch			

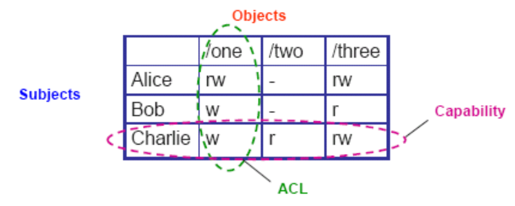
- We include control right in access(D2, D4).
- Then, a process executing in D2 (row) could modify D4 (row).

Access Matrix Implementation

1. Global table
   - Row= <domain,object,rights-set>
   - Whenever an operation M is performed on an object Oj within domain Di,the table is searched for a match
   - Drawbacks:
     - Large table, thus not kept in memory, so additional I/O is needed
     - No advantage for special grouping of domains or object
2. Access lists for objects
3. Capability Lists for Domains

Access Control Lists (ACL)
- For each object, maintain a list of subjects and their permitted actions

Capabilities
- For each subject, maintain a list of objects and their permitted actions



	/one	/two	/three
Alice	rw	-	rw
Bob	w	-	r
Charlie	w	r	rw

ACLs and Capabilities

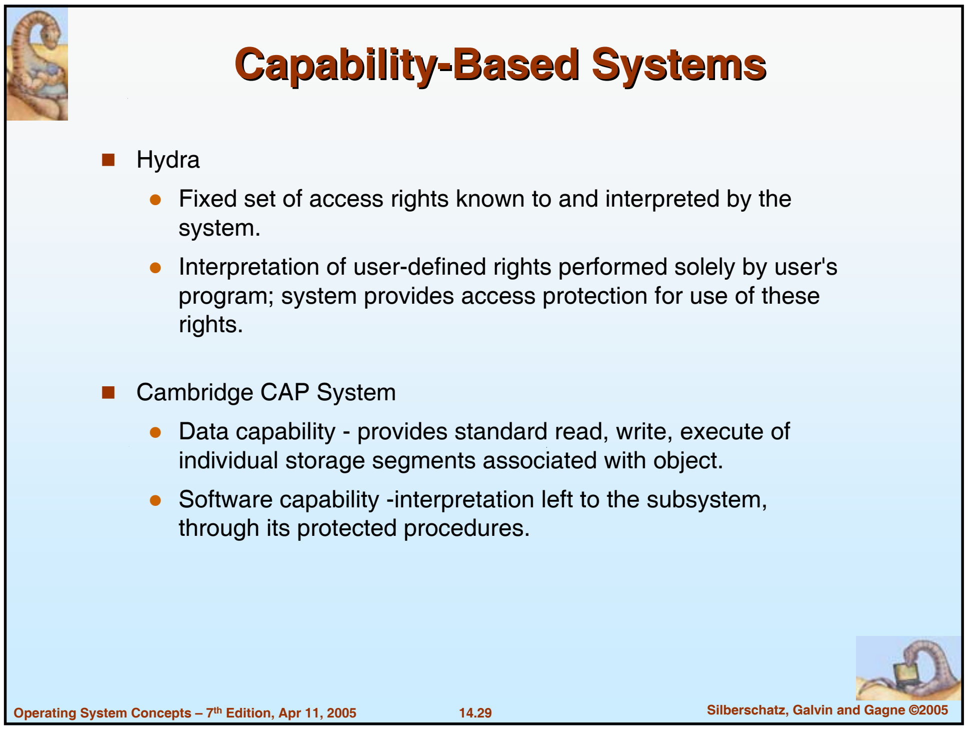
- The approaches differ only in how the table is represented
- Capabilities are easier to transfer
  - They are like keys, can handoff, does not depend on subject
  - Process protection capabilities is kept locally. Process provides a capability for each access, which only need to be verified➔ no search
- In practice, ACLs are easier to manage
  - Corresponds directly to the needs of the users
  - Object-centric, global, easy to grant, revoke
  - Every access must be checked, requiring a search for the list.
  - To revoke capabilities, have to keep track of all subjects that have the capability – a challenging problem
- ACLs have a problem when objects are heavily shared
  - The ACLs become very large
  - Use groups (e.g., Unix)
- Most systems use a combination of both. When accessing an object, the access list is searched. If successful, a capability is created and attached to the process. Otherwise, an error is raised.
  - Future access through the capability
  - After the last access, the capability is destroyed.

Revocation of Access Rights

- In dynamic protection system, we may need to revoke access rights to objects shared by different users.
- Various questions about revocation may arise:
  - ***Immediate*** versus ***Delayed***: Does revocation occur immediately, or is it delayed?
  - ***Selective*** versus ***General***: When an access right to an object is revoked, does it affect all users who have an access 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 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?

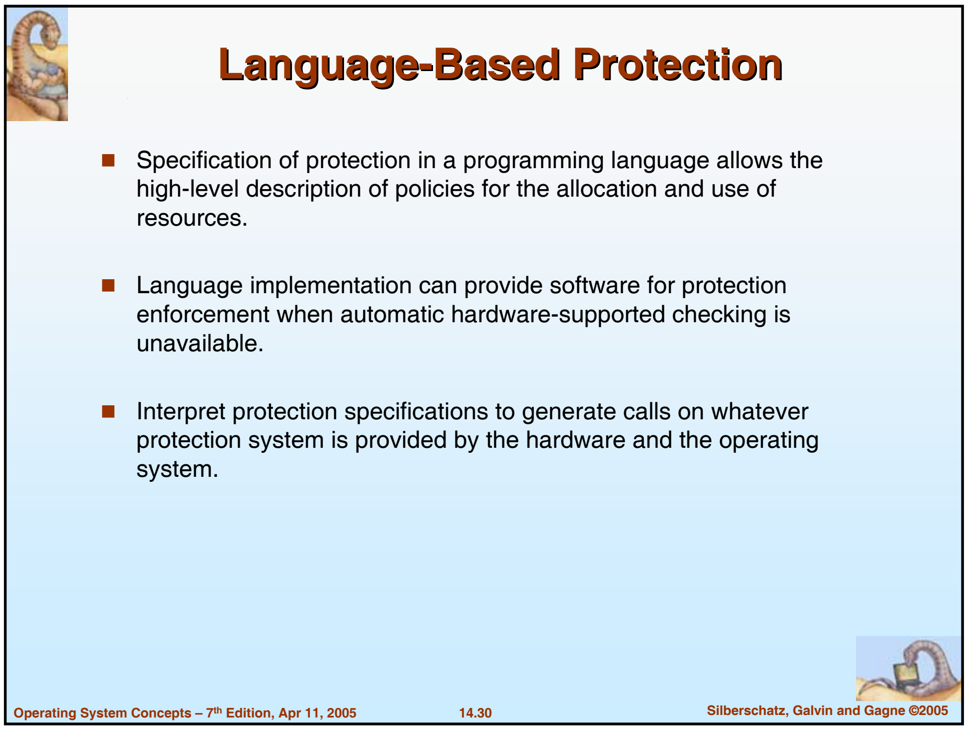
Revocation of Access Rights contd

- With an access-list scheme, revocation is easy.
  - Access list is searched for 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.
- *With a Capability List* scheme it is difficult
  - required to locate capability in the system before capability can be revoked since capabilities are distributed throughout the system.
    - Reacquisition: Periodically capabilities are deleted from each domain.
    - Back-pointers: follow pointers that points to all capabilities associated with that object, then delete as necessary.
    - Indirection: The capabilities point indirectly, not directly, to the objects.
    - Keys: a unique bit pattern that can be associated with a capability. A master key is associated with each object; it can be defined or replaced with the set-key operation. When a capability is created, the current value of the master key is associated with the capability. When the capability is exercised, its key is compared with the master key. If the keys match, the operation is allowed to continue; otherwise, an exception condition is raised. Revocation replaces the master key with a new value via the set-key operation, invalidating all previous capabilities for this object.
      - NO selective revocation, since only one master key is associated with each object



Capability-Based Systems

- Hydra
  - 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 use of these rights.
- Cambridge CAP System
  - 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.



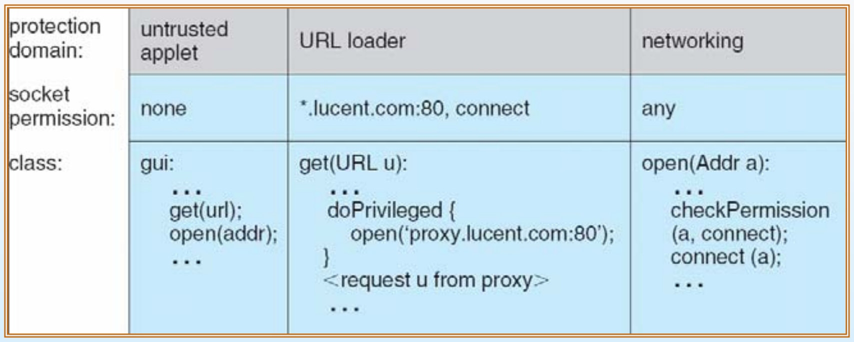
Language-Based Protection

- 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.

Protection in Java 2

- Protection is handled by the Java Virtual Machine (JVM)
- A class is assigned a protection domain when it is loaded by the JVM.
- The protection domain indicates what operations the class can (and cannot) perform.
- If a library method is invoked that performs a privileged operation, the stack is inspected to ensure the operation can be performed by the library.

Stack Inspection



protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any
class:	gui: . . . get(url); open(addr); . . .	get(URL u): . . . doPrivileged { open('proxy.lucent.com:80'); } <request u from proxy> . . .	open(Addr a): . . . checkPermission (a, connect); connect (a); . . .

End of Chapter 14