10 - Protection in General-Purpose Operating Systems

Acknowledgments to Dr. Ola Flygo, Växjö University, Sweden for the original slides
Control of Access to General Objects

Outline

- Introduction to access control for general objects
- Directory-like mechanism for access control
- Access control lists
- Access control matrices
- Capabilities for access control
- Procedure-oriented access control
- Conclusions
Introduction to access control for general objects

- Objects and subjects accessing them
  - **General objects in OS that need protection** (examples)
    - Memory / File or data set on auxiliary storage device
    - Program executing in memory / Directory of files / Hardware device
    - Data structure / OS tables / Instructions, esp. privileged instructions
    - Passwords and authentication mechanism / Protection mechanism
  - **Subjects**
    - User / Administrator / Programmer / Program
    - Another object / Anything that seeks to use object
Introduction to access control for general objects

- **Goals in access control:**
  1) Check every access
     - Access is not granted forever—can be suspended or revoked
  2) Enforce least privilege
     - Give subject access to the smallest number of objects necessary to perform subject’s task
  3) Verify acceptable use
     - e.g., verify if requested kind of access is acceptable
       - e.g., R is OK, W/X is not
Introduction to access control for general objects

- Complexity of access control depends on:
  1) Object homogeneity
     - Homogeneous memory objects vs. heterogeneous h/w device
  2) Number of points of access
     - Access via memory manager vs. access via different device drivers
  3) Existence of central access authority
     - Central memory manager vs. different device drivers
     - 4) Kind of access
        - R/W/X vs. big set of possible kinds of access

- In general:

  Access control for more uniform objects with fewer kinds of access is simpler (e.g., simpler for memory than h/w devices)
Introduction to access control for general objects

- Growing complexity of access control mechanisms
  - Directory
  - Access Control List
  - Access Control Matrix
  - Capability
  - Procedure-Oriented Access Control

[cf. B. Endicott-Popovsky and D. Frincke]
Directory-like mechanism for Access Control

- File directory mechanism to control file access
  - Unique object **owner**
    - Owner controls access rights: assigns/revokes them
      - Access rights (ARs): read, write, execute (possible others)
  - *Each user has its own access rights directory*
Figure 4-10  Directory Access.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Access Rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROG1.C</td>
<td>ORW</td>
</tr>
<tr>
<td>PROG1.EXE</td>
<td>OX</td>
</tr>
<tr>
<td>BIBLIOG</td>
<td>ORW</td>
</tr>
<tr>
<td>HELP.TXT</td>
<td>R</td>
</tr>
<tr>
<td>TEMP</td>
<td>ORW</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File Name</th>
<th>Access Rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIBLIOG</td>
<td>R</td>
</tr>
<tr>
<td>TEST.TMP</td>
<td>OX</td>
</tr>
<tr>
<td>PRIVATE</td>
<td>ORW</td>
</tr>
<tr>
<td>HELP.TXT</td>
<td>R</td>
</tr>
</tbody>
</table>

O - owner  
R - read  
W - write  
X - execute
Directory-like mechanism for access control

- Directory-like mechanism to control access to general objects
  - Analogous to file directory mechanism
- Advantage: Easy to implement
  - Just one list (directory) per user
- Difficulties
Access control lists

- Access control list
  - A list attached to an object
    - Specifying ARs for each subject (who accesses this object)
      - For some subjects specified individually, for others — via being member of a group

Note: This “reverses” the directory approach where:

- lists are attached to a subject
- specifying ARs for each object (accessed by this subject)

- Example
  - Subjects: A, B, C, D, E
  - Use of wild card (*) for ‘any’ (any subject other than B can R/W Object 4)

Object1:  \{\{A: OWR\}, \{B: R\}, \{C: R\}, \{D: R\}\}
Object2:  \{\{A: R\}, \{B: OWR\}, \{C: R\}, \{E: R\}\}
Object3:  \{\{A: OWR\}\}
Object4:  \{\{B: OWR\}, \{*: WR\}\}
Object5:  \{\{B: OWR\}, \{E: R\}\}
Access control lists

- Significant advantages over directory approach
  - Can have default ARs for subjects w/o specific ARs

- Example 2: Unix approach
  - File ARs for: user (owner) / (owner’s) group / others (default)
    - e.g.: drwxr-xr-x 34 jones faculty 1476 Oct 17 08:26 secClass

- Example 3: Multics OS approach
  - user / group / compartment
    - user – ARs for individual subject
    - group – ARs for a group of subjects (e.g., for all project members)
    - compartment – confines untrusted objects or collects related objects
  - Use of wild cards: any user / any group / any compartment
    - Object1: { {Sanjay—Web_Proj—Midwest: X} }
      - Only Sanjay can execute O1 within the ‘Midwest’ compartment when working on the ‘Web’ project.
    - Object2: { {Sanjay—*: RW}, {*:*: R} }
      - Only Sanjay can write O2, but everybody can read it.
Access control matrices

- Previous access control mechanism used lists
  - Directory – subject’s list of ARs for objects accessible by the subject
  - Access list – object’s list of ARs for subjects that can access the object

- Access Control Matrix
  - A sparse matrix (a table)
    - Rows — subjects / columns — objects
    - Cell (i, j) — subjects i’s ARs for access to object j
Capabilities for access control

- Capability mechanism
  - Subjects access objects only via capabilities
  - Capability — a kind of token/ticket/pass giving to a subject certain ARs for an object
    - To see (kind of access) a movie (object), a moviegoer (subject) must have a ticket (capability)
  - Capability to transfer ARs — allows subject to pass copies of its capabilities to other subjects
    - Subject 1 can copy its capability to access Object 1 and transfer it to Subject 2
    - If Subject 1 omits ‘transfer’ rights for Object 1 in capability passed to Subject 2, Subject 2 can’t transfer these rights to any other subject
  - Capability is limited by its domain (= local name space)
    - Not all capabilities passed from caller domain to subroutine domain
    - Subroutine can have capabilities that its calling program doesn’t
Figure 4-13  Process Execution Domain.
Figure 4-14 Passing Objects to a Subject.
Capabilities for access control

- Capabilities help OS keep track of ARs during execution
  - Backed up by more detailed table (e.g., access control matrix)
  - Capabilities for objects accessed by current process are kept readily available (for speed)

- Protecting capabilities
  - Capabilities in memory are accessible to OS only
    - e.g., stored in protected memory
  - Capability are unforgeable - two basic ways:
    - 1) Only OS holds and writes capabilities
      OS issues to subjects only pointers to capabilities
    - 2) Capability is encrypted
      Key known only to OS’s access control mechanism
Example: Kerberos

- Servers: Authentication (AS), Ticket-granting (TGS)
- Implements *single sign-on*.

- Step 1: Joe provides password to AS
- Step 2: AS returns token to Joe (if authentication succeeded)
- Step 3: Joe wants to print to network printer
  - Joe sends *token* to TGS and request for printer R
  - TGS looks up access control matrix/list/directory to determine if Joe should have access to R
  - If yes, send Joe two *tickets*
    - Tell Joe that his access has been authorized
    - Ticket that Joe must present to printer R.
Procedure-oriented access control

- Need to control *actions* that subject can do on object
  - More actions than just R or W or X
    => procedure-oriented access control

- Procedure-oriented access control mechanism:
  - Procedure encapsulates object
    - Controls accesses to object
    - Provides trusted interface to object
    - Implements information hiding

- Example: Accessing table of users
  - Add, delete, validate functions only allowed

- Use of P-OAC results in an efficiency penalty
Conclusions

- Growing flexibility — but also complexity and overhead
  - Directory-like mechanism
  - Access control lists
  - Access control matrices
  - Capabilities for access control
  - Procedure-oriented access control

Flexibility
Complexity
Overhead