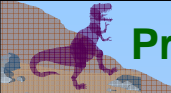# Chapter 18: Protection

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

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.

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 = set of access-rights

| D_1	D_2	D_3
< O_3, {read, write} > < O_1, {read, write} > < O_2, {execute} >	< O_2, {write} > < O_4, {print} >	< O_1, {execute} > < O_3, {read} >

Domain Implementation (UNIX)

- System consists of 2 domains:
  - ☞ User
  - ☞ Supervisor

- 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.

Domain Implementation (Multics)

- Let D_i and D_j be any two domain rings.
- If $j < I \Rightarrow D_i \subseteq D_j$

ring 0

ring 1

ring $N - 1$

...
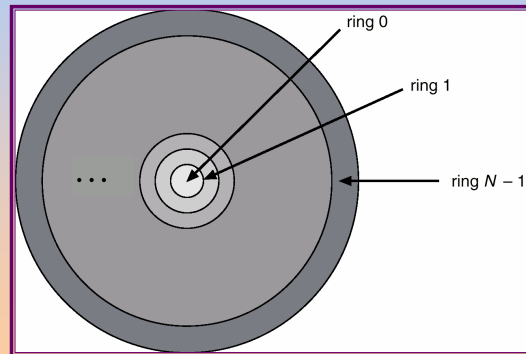
Multics Rings

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	

Figure A

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*

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.

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)
  Fore each domain, what operations allowed on what objects.

  Object 1 – Read
  Object 4 – Read, Write, Execute
  Object 5 – Read, Write, Delete, Copy

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

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)

Access Matrix With *Owner* Rights

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

(a)
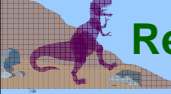
object domain	F_1	F_2	F_3
D_1	owner execute		
D_2		owner read* write*	read* owner write*
D_3		write	write

(b)

Modified Access Matrix of Figure B

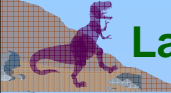| object domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Revocation of Access Rights

- *Access List* – Delete access rights from access list.
  - ☞ Simple
  - ☞ Immediate

- *Capability List* – Scheme required to locate capability in the system before capability can be revoked.
  - ☞ Reacquisition
  - ☞ Back-pointers
  - ☞ Indirection
  - ☞ Keys
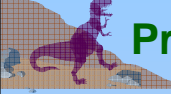
Language-Based Protection

- 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.
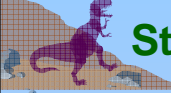
Protection in Java 2

- 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);
			  . . .