Operating System Design

Dr. Jerry Shiao, Silicon Valley University

Protection

start [July 07th]

- Overview
- Protection refers to a mechanism to ensure that only the processes that have gained proper authorization from the Operating System can operate on resources defined by the Operating System.
  - Resources are: Memory Segments, CPU, I/O Devices, Files, Programs.
  - Mechanism must provide the means for specifying the controls to be imposed together, along with the means of enforcement.
- Is Protection the same as Security?
  - Security broader meaning:
    - Security needs Protection is an internal control mechanism: Protection ineffective when user authentication is compromised.
    - Security also must consider external environment: Compromised user authentication, malicious alterations by viruses, worms, Denial-of-Service.

Protection

- Overview
- In a protected environment, there are Domains and Objects.
- Access Matrix as a model of protection.
  - Access List, Capability List, Access-Rights.
- Revocation of Access Rights.
- Capability-Based Systems.
- Language-Based Protection.

Protection

- Protection
  - Mechanism for controlling the access to the Operating System resources by:
    - Programs.
    - Processes.
    - Users.

a lot of concurrency going on.

  - In multiprogramming Operating System:
    - Increase the reliability of any complex system that makes use of shared resources.

based on Access Matrix

  - What are the controls to be imposed and the means of enforcement?
  - Different than Security: The measure of confidence that the integrity of the system and its data will be preserved.

Protection

- Goals of Protection
  - Protection at the Subsystem: Prevent mischievous, intentional violation of access restriction by user.
    - System resources must be used consistent with stated policies.
    - Protect the interfaces between component subsystems.
    - Prevent contamination and malfunction of the subsystem.
  - What are mechanisms? ← static
    - Determines HOW something will be done. ← control by the kernel.
  - What are Policies? ← how the mechanism is interpretted. ← can change
    - Decides WHAT will be done.
  - Role of protection in a computer system is to provide a mechanism for the enforcement of the polices governing resource use.
    - Unprotected resource cannot defend itself against use by unauthorized or user.

Protection

- Principles of Protection
  - Key Principle of Protection: user should not have more privileges than what they actually need.
    - Principle of Least Privilege: Programs, users, and systems be given just enough privileges to perform their tasks.
  - Managing Operating System with Principle of Least Privilege:
    - Operating System features, programs, system calls, and data structures allows minimal damage when compromised.
    - Fine grain Access Control: Enable privileges when needed (passwd).
    - Audit Trail to trace protection and security activities in system.

    a way to control & grant the user privileges when needed
  - Managing users with Principle of Least Privilege: Role based Access Control
    - Operating System manages separate account for each user (RBAC).
    - User enabled with privileges needed.
  - Managing Computers with Principle of Least Privilege:
    - Limit a Computer System to specific services or remote services.
    - Enabling/Disabling services and using Access Control List.

Protection

- Domain of Protection
  - Computer System: Collection of Abstract Data Types.
    - Operations depend on the Object Type.
    - Hardware Objects:
      - CPU – Execute Op.
      - Memory – Read/Write Op.
      - CD-ROM, DVD-ROM – Read Op.
      - Tape Drives – Read/Write/Rewind Op.
    - Software Objects: ← abstract of disk.
      - Data Files – Cr/Op/Rd/Wr/Cl/Del Op.
      - Program Files – Cr/Rd/Wr/Ex/Del Op.

WHAT to be protected.

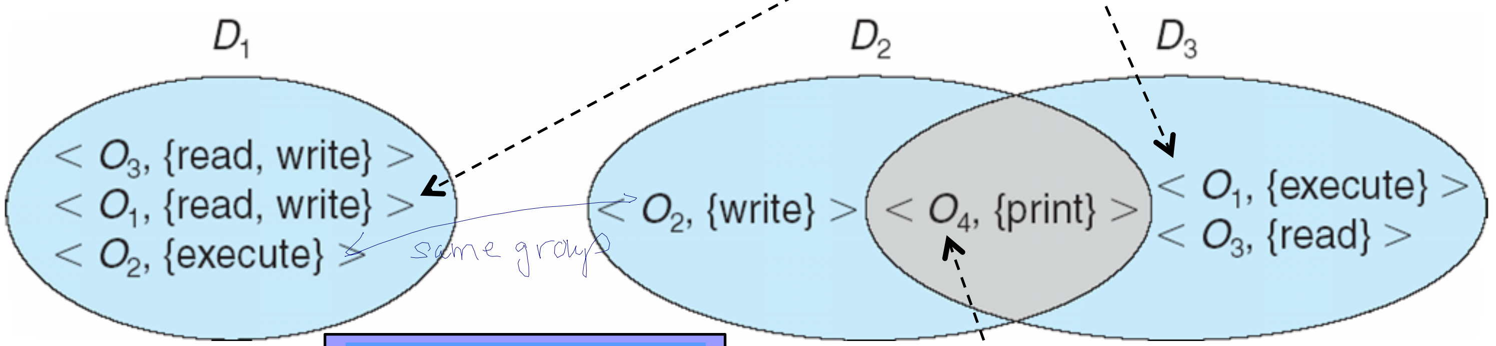
  - Process Access ONLY Authorized Resources and Resources Required to Complete Task.
    - Need-To-Know Principle (Principle of Least Privilege):
      - Procedure access only parameters passed by calling procedure (NOT all variables of calling procedure).
      - Compiler access subset of files (".c" files, NOT arbitrary files).
      - User cannot access private files used by the Compiler.

only allow the procedure to access local params belong to it

Protection (go to 10 first)

- Domain Structure
- Process executes within a Protection Domain:
- Protection Domain: Resources a process may access.
  - Access Rights = Ordered pair of Objects + { Rights-Set }
    - Object = File "F", Rights-Set = { read, write }
  - Domain = Collection of Access Rights

O1, Domain D1 allows {read, write} and Domain D3 allows {execute}.



Domain defines a set of Objects and Types of operations that may be invoked on each Object.

Domain D2 and D3 share Access Rights <O2, {print}>.

Protection

Protection Domain = Process + Domain
= Process + { access right 1
 ———— 2
 ———— 3
 . . .
= Process + | Obj 1, read
 | — 2, write
 | . . .

- Domain Structure
- Protection Domain: Association with Process
  - Static:
    - Set of Resources available to the Process is Fixed throughout Process' Lifetime.
    - Need-to-Know Principle
      - Process only needs the minimum Access Rights.
    - Mechanism must be available to change content of a domain.
      - Scenerio: Process may need Read Access in one period and Write Access in another.
        - Domain with both Read and Write Access violates Need-to-Know Principle (Process has more Access Rights then necessary).
        - Modify the contents of a Domain, Domain will always reflect minimum Access Rights.
  - Dynamic:
    - Mechanism is available for process to switch Domains.
    - Contents of Domain can also be allowed.
    - Create new Domain with changed content and switching to new Domain.

Protection

- Domain Structure
- What is a Domain?
  - User: ← focus on Domain as a User
    - Set of Objects associated with identity of User.
    - Domain switching when user logs out, another user logs in.
  - Process:
    - Set of Objects associated with identity of Process.
    - Domain Switching when one process sends message to another.
  - Procedure:
    - Set of Objects corresponds to local variables within Procedure.
    - Domain Switching when procedure call is made.
  - Operating System Dual Mode: ← domain switching
    - Monitor (Kernel) Mode: Execute privileged instructions and has control of Operating System.
    - User Mode: Execute unprivileged instructions,.
    - Domain Switching with System Calls.

Protection

- Domain Structure
- UNIX Domain

instead of associating with process as protection domain.

  - Domain Associated with User: User and Supervisor (Root).
    - File System allows temporary User (Domain) to be switched.
    - Special Access Bits: If the <u>SUID Bit</u> is set for an executable file (i.e. command or shell script), the process takes on the User privilege of the owner of the file when it executes.
    - **passwd Command**: Allows user to change user's password, but /etc/passwd file is owned by Kernel.

file owned by root.

    - sau@buildbed-vm:/usr/bin> ls -l /etc/passwd

      -rw-r--r-- 1 root root 2029 2012-10-02 15:43 /etc/passwd
    - sau@buildbed-vm:/usr/bin> ls -l /usr/bin/passwd

      -rwsr-xr-x 1 root shadow 80268 2011-07-29 12:55 /usr/bin/passwd

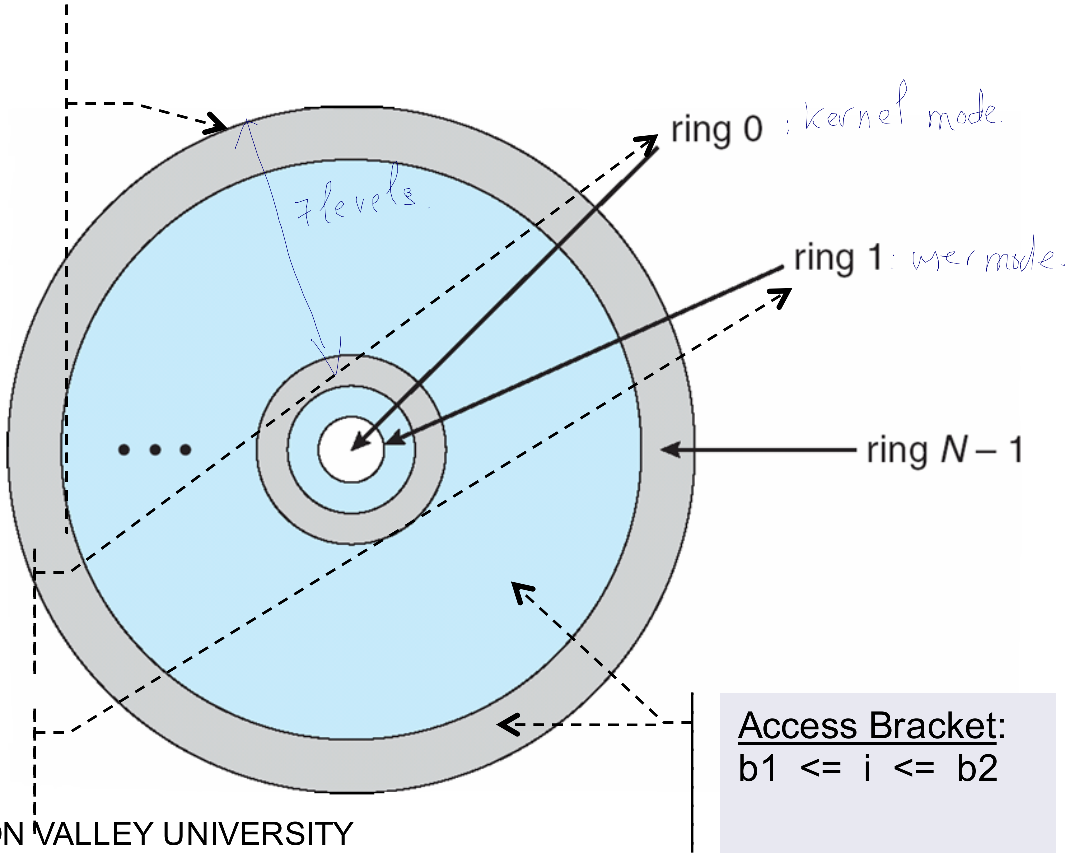
Bit: 12	11	10	9	8	7	6	5	4	3	2	1
SUID	SGID	Sticky	r	w	x	r	w	x	r	w	x
Bits for special access privileges			Owner's access privileges			Group's access privileges			Others' access privileges		

chmod 4xxx <file-name> OR chmod u+s <file-name>

Protection

- Domain Structure
- MULTICS Domain ← used by UNIX
  - D<i> and D<j> are Domain Rings, 0 to 7.
  - If <j> less than <i>, then D<i> is a subset of D<j>

-- Segmented address space, where each segment is a ring belonging to a domain.
-- Domain switching when a process calling a procedure in a different segment (ring).
-- Segment has Access Bracket, allowing a segment to call a lower numbered segment (higher privilege) or call a higher numbered segment (lower privilege).
-- Lower numbered segment can only be reached through predefined gates (entry points).



Ring 0 process has most privileges.

Ring 1 process has less privileges.

Access Bracket:
b1 <= i <= b2

Protection

- Access Matrix: Model of Protection
  - Rows: Represents Domains.
  - Columns: Represent Objects
  - Entry is a set of Access Rights.

Process executing in Domain D1 has "read" privilege with Files F1 and F3. Process executing in Domain D4 has the same privileges as D1, with additional "write" privilege with F1 and F3.

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	

Matrix grows bigger when system has more users. or more objects

Protection

- Access Matrix
  - <u>Matrix provides the mechanism </u>for specifying different policies.
    - Mechanism is implementing the matrix and following the properties.
    - If a process in Domain D<i> tries to do "op" on object O<j>, then "op" must be in the Access Matrix.
    - Example: Process executing in Domain D<i> can access objects in Row <i> ONLY as allowed by the Access Matrix entry's Operation.
  - Matrix provides the mechanism for implementing control for <u>dynamic association </u>between Process and Domain.
    - Domain Switching: <u>Adding Domains as Objects </u>of Access Matrix.
      - *Switch – Switch from Domain D<i> to D<j>.*
    - Matrix entry modified: <u>Adding Matrix as Object </u>of Access Matrix.
      - Use Special Access Rights in Matrix entry:
      - *Owner of O<j> - Add or Delete O<i> to O<j> (within column).*
      - *Copy Operation from O<i> to O<j> (within column).*
      - *Control – D<i> can modify D<j> Access Rights (within row).*

Protection

- Access Matrix

Policy:
-User Dictates Policy.
- Who can access what Object and in what Mode (Domain).
-Policy decision involve which Access Right are included in the Access Matrix Entry (whether the entry is blank or contains Rights).

Mechanism:
Process executing in Domain can Access ONLY Objects in row (Implements the Access Matrix Entry).

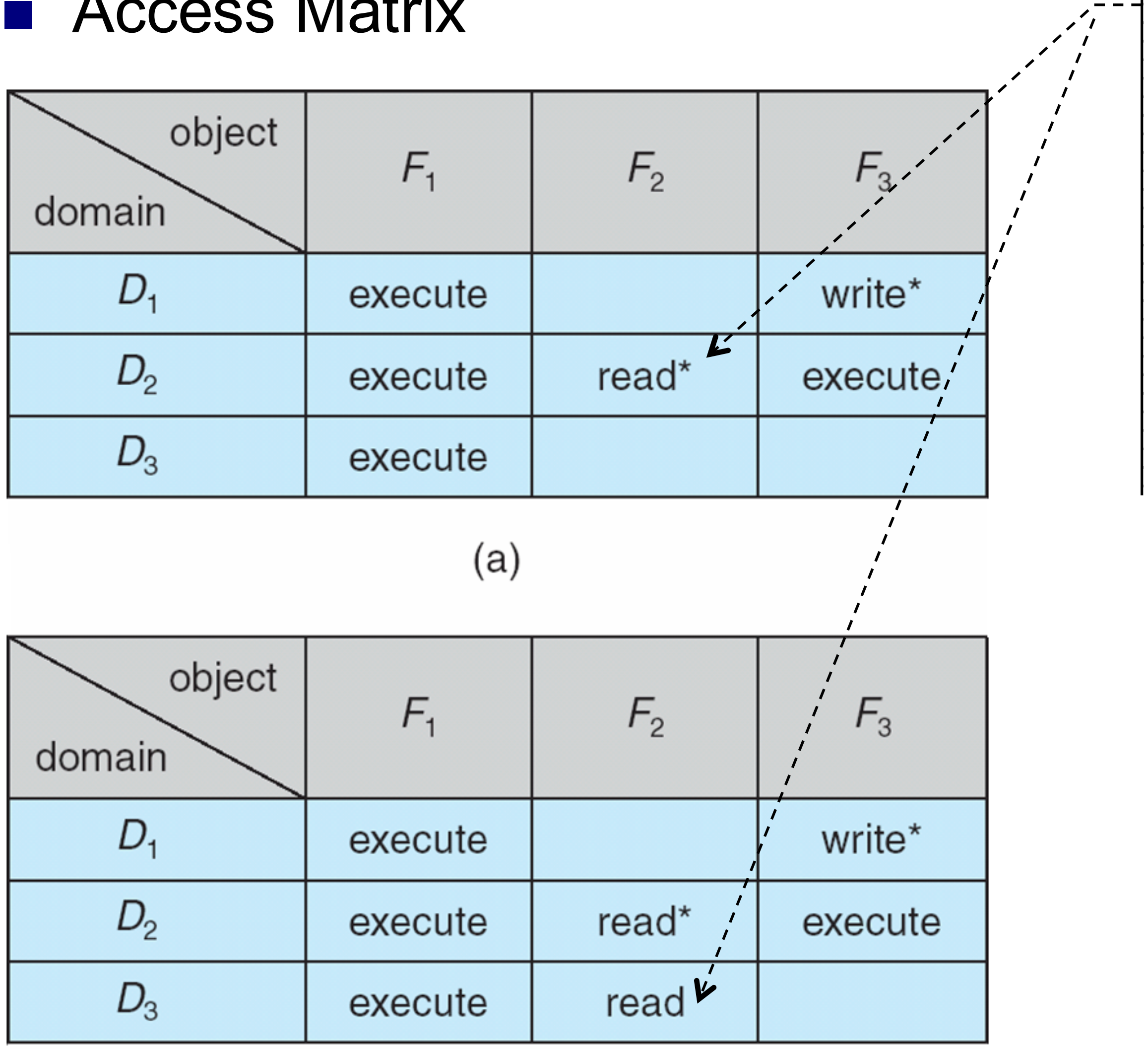
domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Domain as Objects.
A process executing in D1 can switch to D2.

Domain D1 can switch to D2

Protection

- Access Matrix



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

(a)

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

(b)

Copy Access Right denoted by asterisk (*) appended to the Access Right.
-Copy allows Access Right copied only within the column (for the Object).
- Copy can be a transfer (Access Right REMOVED from originating Object).
- Limited Copy allows ONLY the Access Right to be copied, NOT the ability to Copy (Example below).

Protection

- Access Matrix

Owner Access Right is mechanism to allow addition of new rights and removal of rights.
- Allow a process to change the entries ONLY in the Object's column.

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

(a)

owner can provide permission of write . to others.

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)

D_2 allows D_3 to have access right to F2/F3.

Protection

- Access Matrix

<u>Control Access Right</u> applicable to Domain Objects.
-Changes Access Matrix ONLY on the row.
- Access Matrix entry D<2>,D<4> Control Access Right allows changes to D<4>.
-Example removes "Read" Access.

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			

D_2 is [illegible] of su

Protection

refer p25.

- Implementation of Access Matrix
- How Access Matrix (sparse matrix) be implemented?
- Global Table
  - Ordered Triplets < Domain, Object, Rights-set >.
  - Executing operation on Object O<j> within Domain D<i>, the Global Table is searched for triplet < O<j>, D<i>, R<k> >.
  - If triplet found (operation in Rights-set, R<k>), operation allowed to continue.
  - Global Table very large and cannot be kept in memory, Virtual Memory technique required to manage the table.
  - Special groupings of Objects, where everyone (Domain) can read (operation in Rights-set) the Object, then the Object must have entry in every domain.

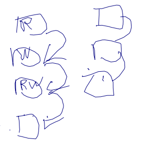
Protection

refer p25.

Obj i: Access List

F1 F2 F3

Object Centric



Linked List of the columns.

- Implementation of Access Matrix
- Access List for Objects
  - Each column in Access Matrix is an Access List for the Object
    - List for each Object is an ordered pair { Domain, Rights-set }.
      - Defines who can perform what operation on the Object.
      - Domain 1 = Read, Write
      - Domain 2 = Read
      - Domain 3 = Read
    - Domains contain access rights for the Object.
  - Define a default set of Access Rights.
  - Every access to the Object requires searching the Access List, which can be time consuming.

Protection

p25

Capability List

$$\begin{tikzcd} D_1 \arrow[r] & \boxed{F_1} \arrow[r] & \boxed{F_2} \arrow[r] & \boxed{F_3} \\ & \text{obj} \arrow[u] & \text{obj.} \arrow[u] & \end{tikzcd}$$

} Domain Centric

- Implementation of Access Matrix
- Capability List for Objects

Linked List

- Each row in Access Matrix, is a Capability List for a Domain.
  - Capability is an Object.
  - Capability List is list of protected Objects, maintained by Operating System, and accessed by the user indirectly.
    - List of Objects together with the operations allowed on those Objects.
    - Object 1 – Read
    - Object 4 – Read, Write, Execution
    - Object 5 – Read, Write, Delete, Copy
  - To provide protection, distinguish Capabilities from other kinds of Objects.
    - Each Object is tagged as Capability or accessible data.
      - All Object can be tagged with the Object's type (Integer, Pointers, Booleans,
    - Address space associated with program is split into two parts: one part contains program and other part contains the Capability List.

Protection

- Implementation of Access Matrix
- Comparision
- Global Table:
  - Simple, but table can be large and sparse: Virtual Memory often used to manage the table.
  - Cannot take advantage of special groupings of Objects or Domains.
- Access List: ← column ← Unix is more of Access List.
  - Users creates Access List by creating the Object and linking the Domains and operations allowed.
  - Access List Only: Access-Rights (Domain, Rights-set) not localized because Access-Rights spread is over many Domains.
- Capability List: ← row file descriptor table.
  - Process in the Domain creates localized list of capabilities (Object, Rights-set). priveleges that the OS provides
  - Capability Only: Useful for localizing information for a process.

Protection

- Implementation of Access Matrix
- Comparision
- Combination of Access Lists and Capabilities.
  - Combination: Process first access an Object, Access List is used and a Capability List for the Domain is created for the process.
    - Additional references uses the Capability List.
    - When Access List entry removed, the entry in Capability List also removed.
  - Lock-Key Mechanism: Using Combination, but each object has an unique bit pattern (Lock), and Domain has list of unique bit patterns (Key).
  - Example: File system using Access List and Capabilities.
    - The File (Object) has Access List that is used to validate permission.
    - File Table (Capability List) is created for the User (Domain) and additional Files (Objects) are linked.
    - An index is returned for the File Table (Lock-Key Mechanism) and the index is used to access the File Table.
    - When File is closed, the File Table entry is removed.

Protection

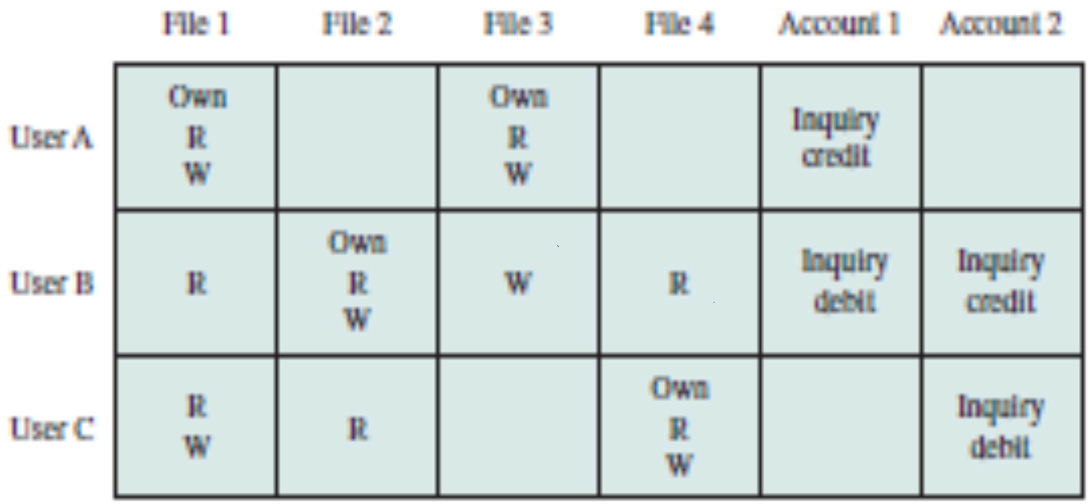
- Implementation of Access Matrix
- UNIX Database Management System makes decision on individual user access attempt.
  - Operating System only grants user permission to access a file or use an application.
  - Access Control mechanism is the Access Matrix.
    - Subject: Entity capable of accessing objects (process that represents User).
    - Object: Anything to which access is controlled (files, programs, segments of memory, software objects (Java objects)).
    - Access Rights: Way in which an Object is accessed by a Subject (read, write, execute, and functions in software objects).
  - Access Matrix implemented by decomposition into either:
    - Access Control Lists: Each Object, lists Subjects (Users) and their permitted Access Rights.
    - Capability Tickets: Authorized Objects and operations for a Subject (User).

Protection

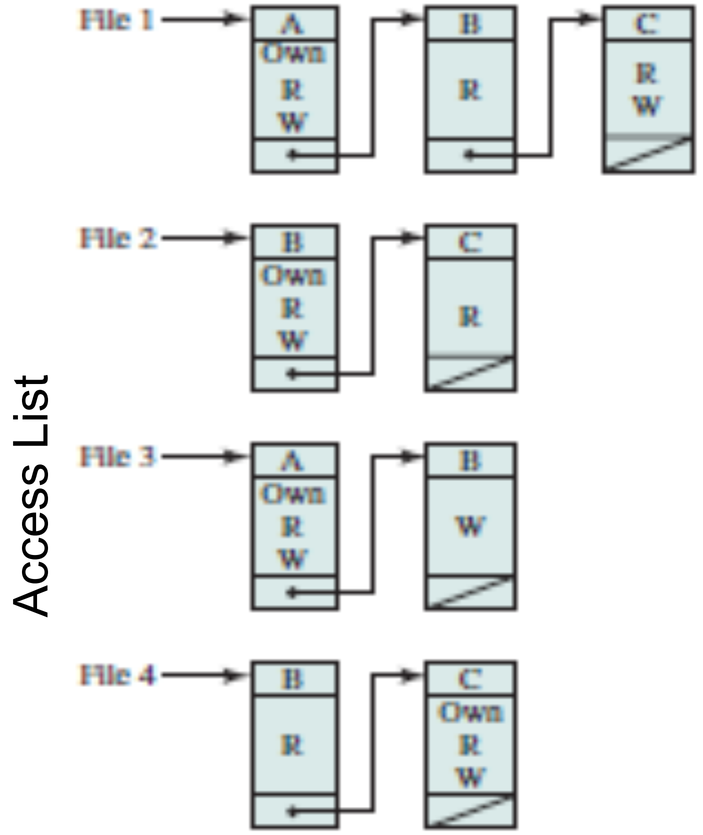
- File System or Database Management System

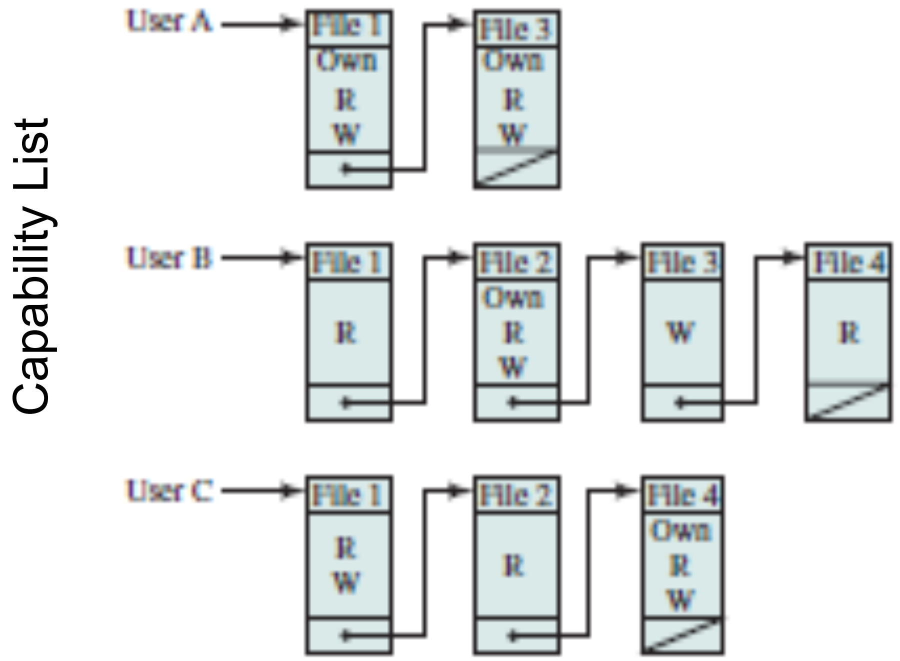
Base on objects

Access Matrix



	File 1	File 2	File 3	File 4	Account 1	Account 2
User A	Own R W		Own R W		Inquiry credit	
User B	R	Own R W	W	R	Inquiry debit	Inquiry credit
User C	R W	R		Own R W		Inquiry debit





Protection

- Access Control
- Role-Based Access Control (RBAC): Protection enhanced with concept of privileges.
  - Implements Principle of Least Privilege.
  - Definition of Privilege: Right to execute system call or use an option within the system call (i.e. Open File with Write access).
    - Privilege similar to Rights-Set.
  - Privilege assigned to process, limiting process to the access needed to perform the work.
- Concept of Role: Privileges (Rights-Set) and programs can be assigned to Role.
  - Role is a group of privileges (Rights-Set).
- Users are assigned to Roles, not Objects (i.e. File).
  - User takes Role, that enables a privilege, allowing User to access Files and run programs.
  - Multiple Users can be assigned to one Role.

Protection

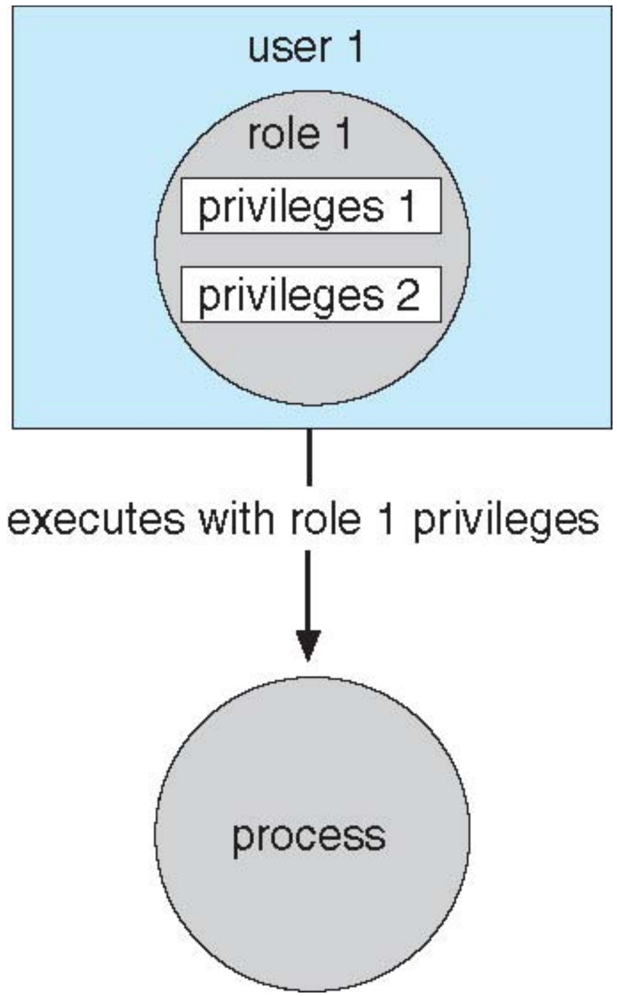
- Access Control
- Role-Based Access Control (RBAC): Protection enhanced with concept of privileges.
  - Implements Principle of Least Privilege.
  - Definition of Privilege: Right to execute system call or use an option within the system call (i.e. Open File with Write access).
    - Privilege similar to Rights-Set.
  - Privilege assigned to process, limiting process to the access needed to perform the work.
  - Controls access based on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles.
    - Not on the identity of the User.

$$\text{user} \rightarrow \text{Role} \Rightarrow \text{Access Right} \begin{cases} \text{object} \\ \text{rights} \end{cases}$$

- Concept of Role: Privileges (Rights-Set) and programs can be assigned to Role.
  - Role is a group of privileges (Rights-Set).

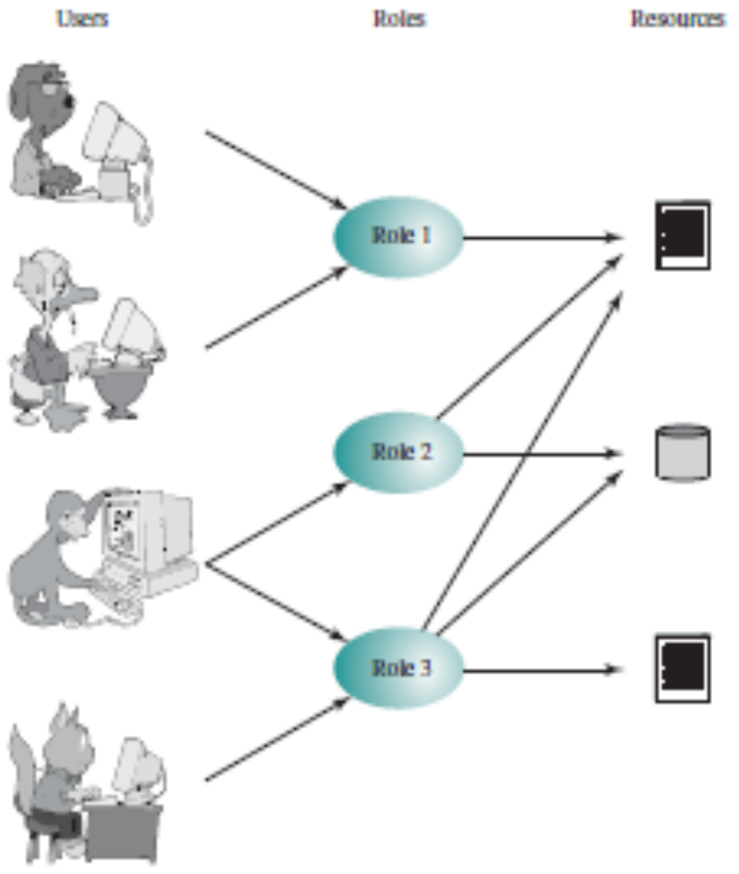
Protection

- Access Control
- User assigned Role(s), NOT Objects (i.e. File).
- User can take Role to enable a privilege, allowing the user to access an object (File with Access-Rights) or execute a programs.
- Multiple Users can be assigned to one Role.



Protection

- Access Control
- RBAC represented in Access Matrix.
  - Matrix depicts relationship of Users to Roles.
    - Many more Users than Roles.
    - Single User assigned to multiple Roles.
    - Multiple Users assigned to single Role.
  - Matrix depicts relationship of Roles to Objects.
    - Objects are hardware resources (CPU, Memory Segments, Printers, Disks, Tape Drives).
    - Objects are software resources (Files, Programs, Semaphores).



Protection

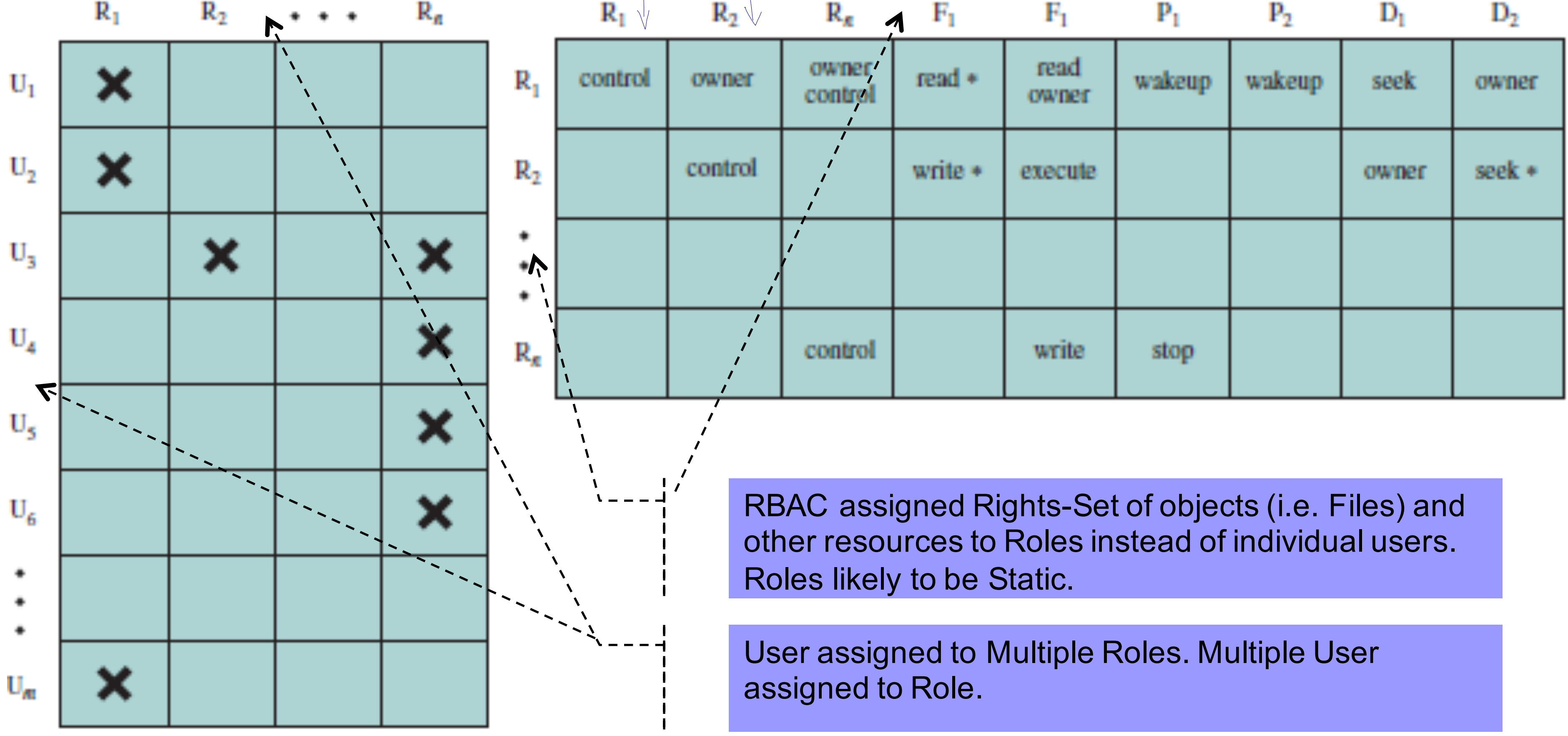
- Role-Based Access Control (RBAC)

User tied to roles

Role

Role.

Object privileges tied to Role.



	R_1	R_2	$\cdots$	R_n
U_1	✖			
U_2	✖			
U_3		✖		✖
U_4				✖
U_5				✖
U_6				✖
$\vdots$				
U_m	✖			

	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
$\vdots$									
R_n			control		write	stop			

RBAC assigned Rights-Set of objects (i.e. Files) and other resources to Roles instead of individual users. Roles likely to be Static.

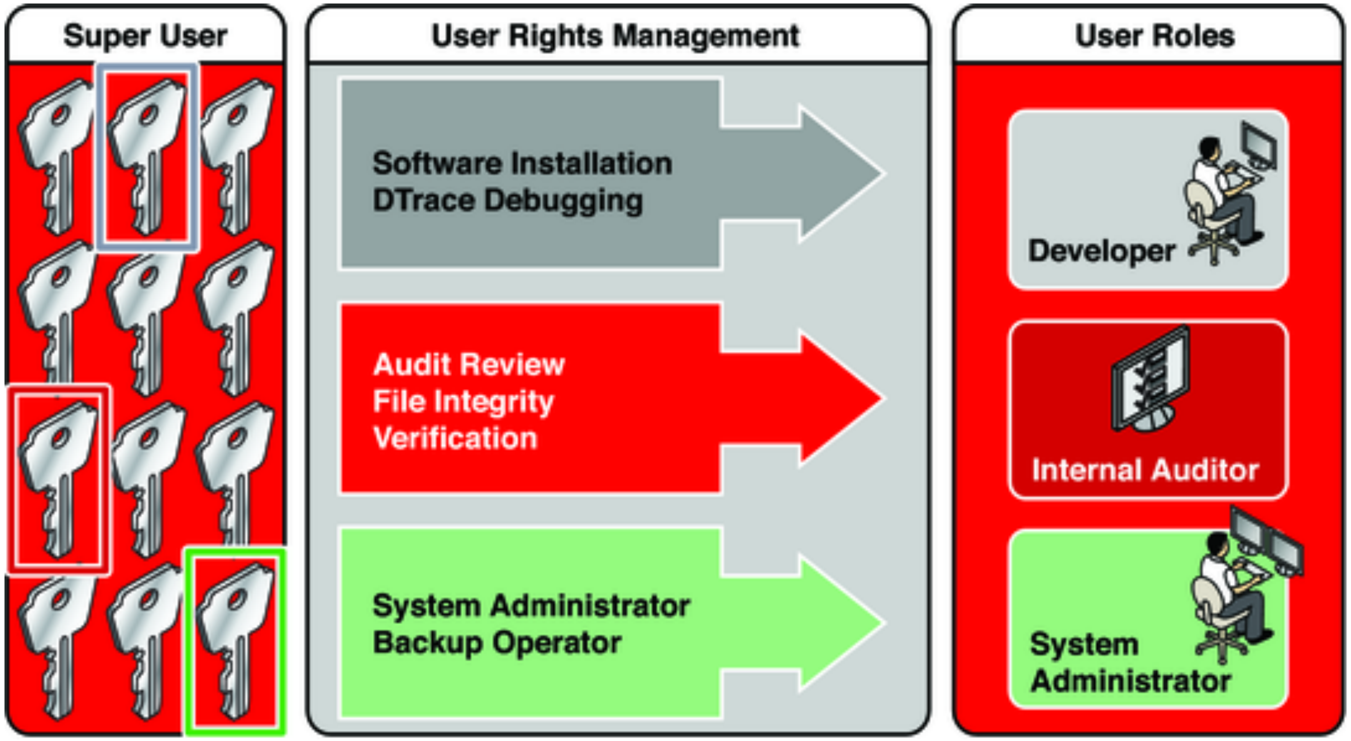
User assigned to Multiple Roles. Multiple User assigned to Role.

Protection

- RBAC in Solaris Environment
- Users are given the minimal amount of privilege to accomplish their given responsibilies.
- Applied to UNIX SuperUser (root) privileges.
  - Role with limited SU capabilties:
    - Primary Administrator: Full Root capability.
    - System Administrator: Less strong (role, not related to security (i.e. this role does not allow user to set passwords).
    - Operator: Need privileges for Backups, Restores, and Printer Management.
  - Role has attributes as user account (i.e. UID, Home Directory, Profile Shells, and set of unique capabilities).
    - User logs in as themselves and switch to the role with "su" command.
    - The Profile Shell understand RBAC and implement role capabilities.
  - RBAC Configuration Files:
    - /etc/user_attr: Defines which users can assume which roles and profile.

Protection

- RBAC in Solaris Environment
- Superuser creates roles:
  - Superuser assigns the roles to users who are trusted to perform the tasks of the role. After user login, users assume roles that can run restricted administrative commands and GUI tools.



Protection

- Revocation of Access Rights
- Protection Systems need to revoke Access-Rights to Objects shared by different Users.
- Revocation Issues:
  - Immediate Versus Delayed (when will it take place)?
  - Selective Users or All Users of an Object?
  - Partial Access-Rights or All Access-Rights?
  - Permanently revoked or can revoked Access-Rights be restored?
- Revoke by Access List: ← easy to remove object. access from a user
  - Simple, follow Access List Object to Domains and remove one or all Access-Rights.
  - Immediate, Selective or All Users, Partial or All Access-Rights, Permanent or Temporary.
- Revoke by Capabilities: ← easy to remove user access rights from object.
  - Capabilities distributed throughout the system, they must be found before revoked.

Protection

- Revocation of Access Rights
- Revoke by Capability List:
  - Capabilities has different Object types (i.e. Files, Domains, Devices (Printer)) and distributed throughout the system.
    - List of Pointers maintained with each Object. Pointers are to other capabilities (Objects).
      - Revoke capabilty by removing pointer from List of Pointers.
    - Global Table of Objects and Indirection. Searching the Global Table will locate all Objects for a Capability.
      - Revoke capability by removing Object Entry (Pointer) from Global Table of Objects.
    - Object associated with list of keys. Domain that owns the capability will have a set of keys (Objects).
      - Global Table of Keys and Indirection. Searching the Global Table of Keys will locate all Objects for a Capability.
      - Revoke capability by removing key from Global Table of Keys.

Protection

?

- Capability-Based Systems
- Hydra: Subsystem built on top of protection Kernel.
  - Subsystem interacts with Kernel through Kernel-defined primitives.
- Provides flexibility with User defined Object Types with User defined Access-Rights and System defined Access-Rights.
- Operations on Objects are through pre-defined procedures.
  - Object contains user-defined procedures that is used to perform an operation on the Object (Auxiliary Rights).
  - Process must have right to execute the procedure.
  - The Access-Rights of the procedure can exceed the rights held by the calling process.
    - Amplification allows the procedure to perform the action, but revert back to the calling procedure Access-Right when procedure returns.

Protection

- Language–Based Protection
- Protection usually achieved through Kernel: Acts as security agent to validate attempt to protected resource.
  - Overhead of access validation is high: Protection environment is large, should system designer compromise goals of protection?
- Protection can be provided by a programming-language implementation.
  - Policies might change over time on the Application Subsystem.
  - Access Control for a shared resource in a system is making a <u>declarative statement about the resource</u>.
  - The <u>Programming Language's typing facility</u> can be integrated with the declarative statement.
- Compiler-Based Enforcement
  - Protection needs are simply declared, rather than programmed as a sequence of calls on procedures of an Operating System.
  - A declarative notation is natural because <u>access privileges are closely related to concept of data type</u>.

Protection

- Language–Based Protection
- Relative Merits of Protection by Kernel as opposed to language-based?
  - Kernel provides greater degree of security than language-based, that has to use the underlying mechanism from the Operating System for protection. Language-based coding errors is a security risk.
  - Kernel is not as flexible in implementing protection. Language-based user-defined policy can be extended or replaced by recompiling Application Subsystem, without modifying the Operating System.
  - Software protection enforcement when hardware-supported checking not available.
  - Greater efficiency in verifying language-based protection off-line.
- Protection in programming languages greater concern to systems with distributed architectures and stringent requirements on data security.

Protection

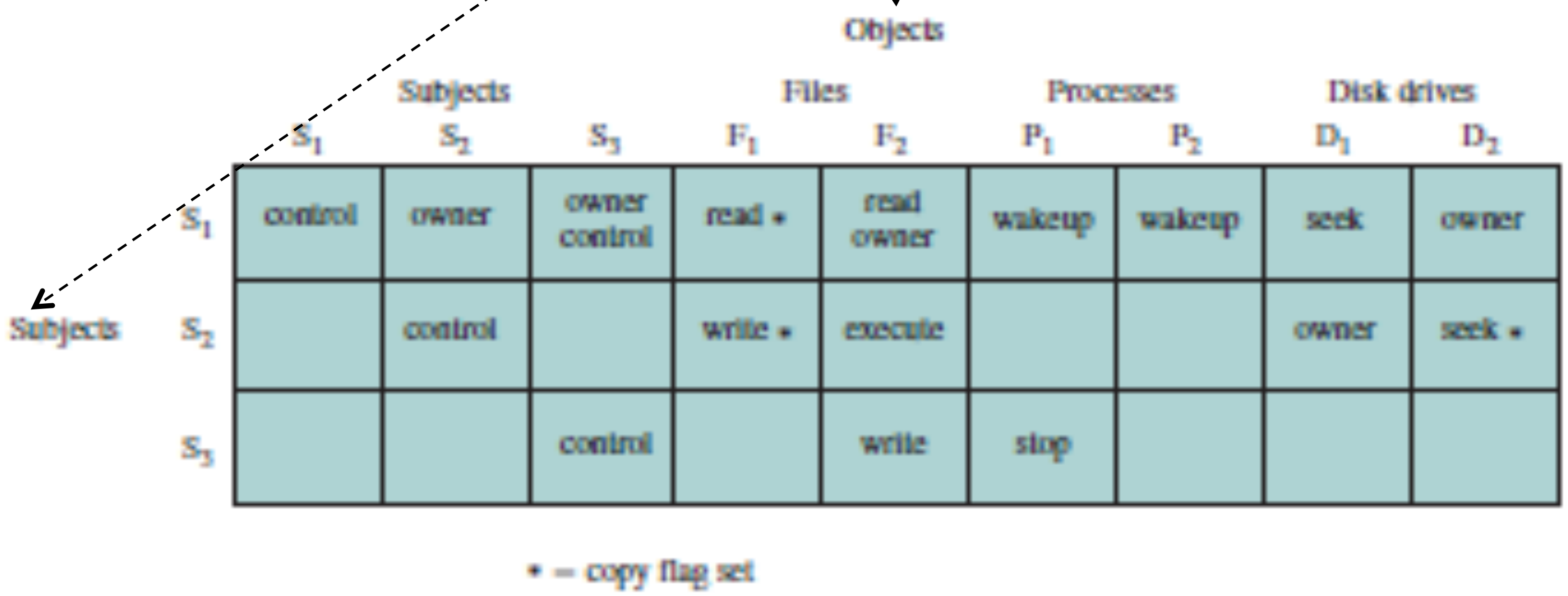
end on Jul 07.

- Summary
- Computer Systems contain objects that must be protected.
  - Hardware Objects (Memory, Real-Time Clock, I/O Devices).
  - Software Objects (Files, Programs, Semaphores).
- Access-Right: Permission to perform an operation on an Object.
- Domain: Set of Access-Rights.
- Process or User or Procedure is a Domain and may use any of the Access-Rights in the Domain to access and manipulate Objects.
  - Process can switch from one Domain to another.
- Access Matrix is a model used to illustrate the protection policy of a Computer System.
- Access List is implementation of the Access Matrix associated with each Object.
- Capability List is implementation of the Access Matrix associated with each Domain.
- Revocation of Access-Rights is implemented using an Access List.
- UNIX provides protection for File Objects:
  - Access-Rights are Read, Write, and Execution protection.
  - Domain is the User, as the Owner, Group, or Other of the File Object.
- Solaris implements a Role-Based Access Control (RBAS).
- Language-based protection is more flexible than the Operating System, when implementing protection.

Protection

- Access Matrix
- Access Control Policy as Access Matrix

Access Control Matrix specifying the access rights of subject “S” to object “X”.


Objects
Subjects

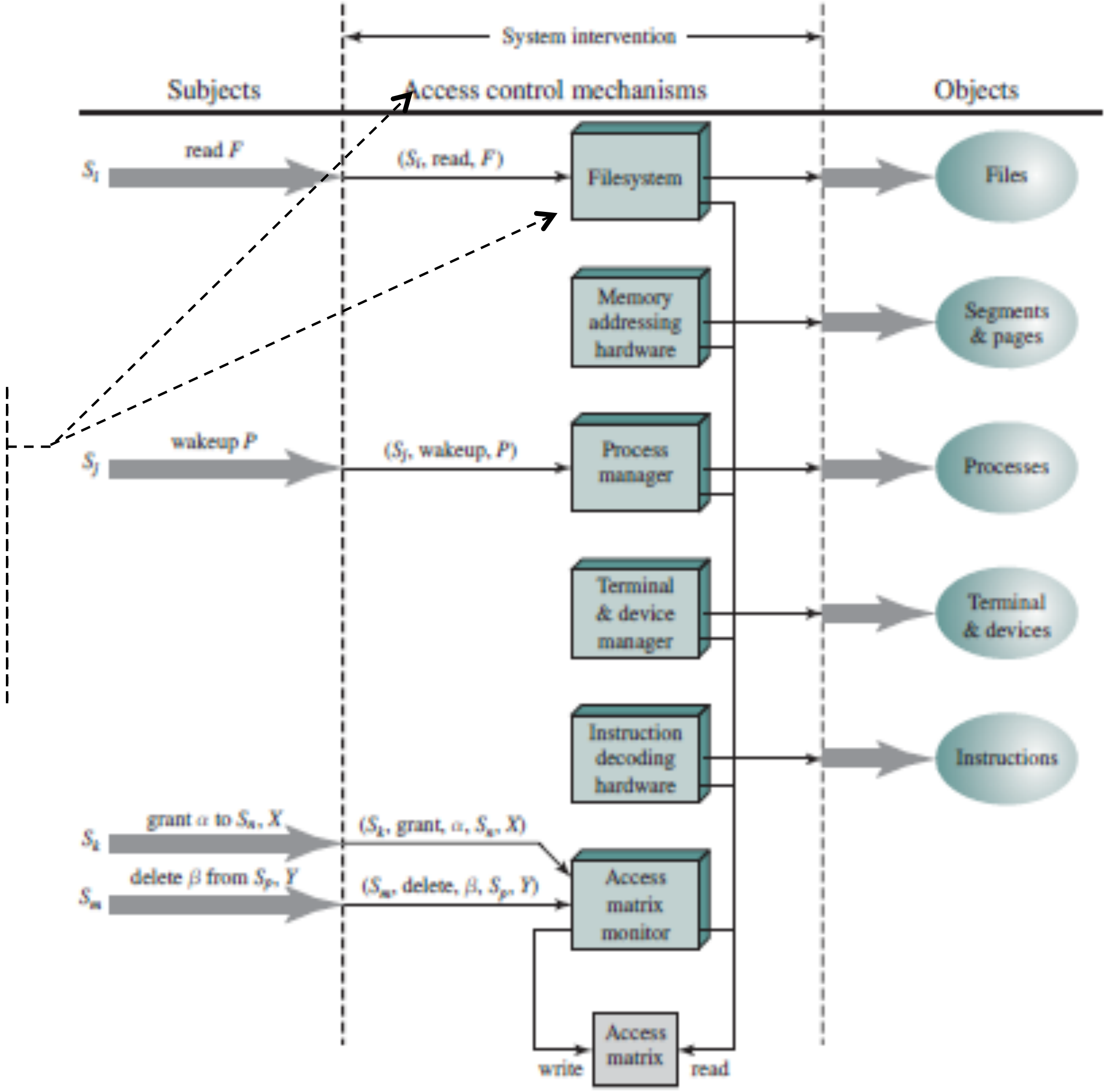
	Subjects			Files		Processes		Disk drives	
	S_1	S_2	S_3	F_1	F_2	P_1	P_2	D_1	D_2
S_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
S_2		control		write *	execute			owner	seek *
S_3			control		write	stop			

* = copy flag set

Protection

- Access Matrix
- Access Control Function

Functional View: A separate Access Control Module is associated with each type of Object. The module evaluates each request by a Subject to access an Object to determine if the access right exists.



Protection

- Capability-Based Systems
- Cambridge CAP System
  - Access to a Memory Segment or hardware required that the current process held the necessary capabilities.
- Simplier than Hydra: Using two kinds of capabilitites.
- Data Capability: Access-Rights provided are Read / Write / Execute of the storage segments associated with Object.
- Software Capability: Protected procedure that is part of a subsystem.
  - Process executing the procedure will acquire the right to Read or Write the contents of a software capability.

Protection

- Language–Based Protection
- Compiler-Based Enforcement
  - Protection available to the application program through use of Software Capability to system resources.
    - Storage reference must occur indirectly through a Software Capability: Prevents any process from accessing a resource outside of its protection environment.
  - Arbitrary restrictions on resource use during particular code segment.
  - Program can distinguish areas of code where protection is needed.
    - Mechanism must reply on more assumptions about the operational state of the Operating System.
  - Principle assumption is that the code generated by the compiler will not be modified prior or during execution.

Protection

- Language–Based Protection
- Protection in Java
- Java designed to run in a distributed environment.
  - JVM (Java Virtual Machine) has many built-in protection mechanisms.
  - Programs composed of classes: Collection of data fields and functions (methods) that operate on these fields.
  - Untrusted Classes loaded over network or within JVM:
    - Protection domain of loaded class depends on URL from which the class was loaded and digital signatures on the class file.
    - Trusted server places class in domain different from untrusted server.
  - Stack Inspection during access of protected resource:
    - A method in the calling sequence must assert the privilege to access the resource.
    - Stack inspection (checkPermissions() API) used to determine if the request should be allowed.
    - Examines stack frames on the stack, from current to oldest, to find the method that requested the access (doPrivileged() API).
    - If domain of a method in the stack without access permissions is found, the request is not allowed.

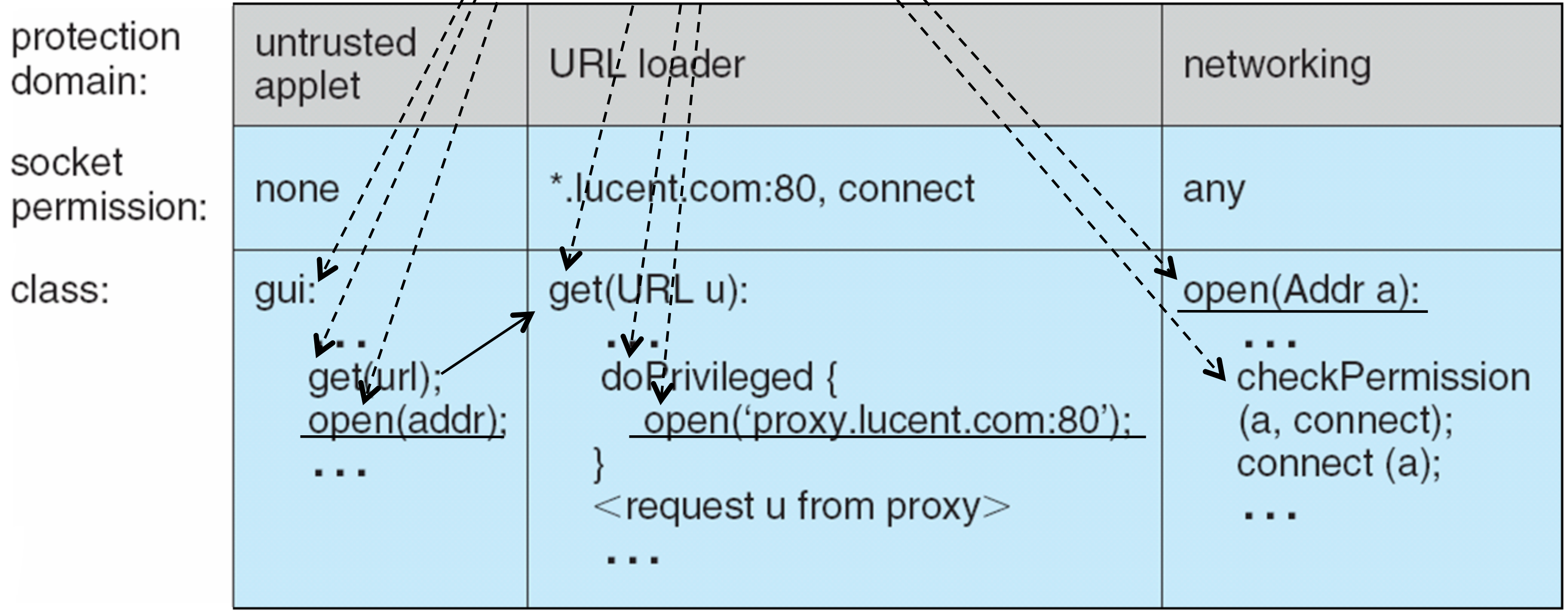
Protection

- Language–Based Protection
- Protection in Java

Get() executes doPrivileged() assert successfully privilege to access the resource.
Open() executes **successfully**.

Method, gui, from "untrusted" protection domain.
Gui() performs get() and open() operation.

Open() executes checkPermission() to perform stack inspection.
In untrusted applet open(), there is no evidence of privilege was asserted before request to open() the resource, open() execution fails.



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);
			…