Unit VI: Protection in Operating System

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

# Introduction

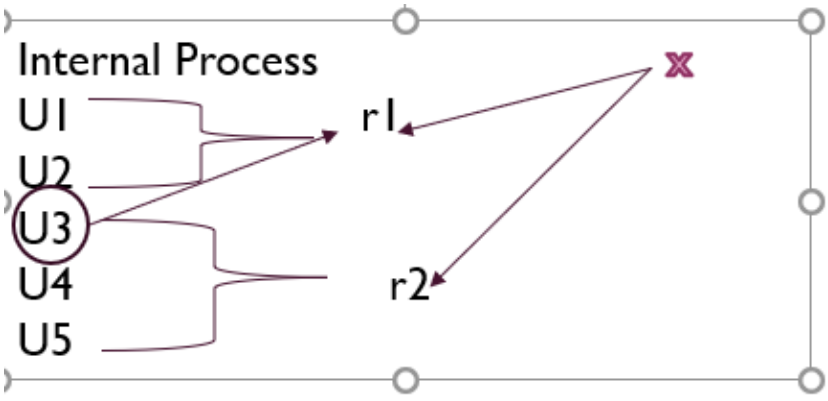
Protection and security Are implemented to prevent interface with the use of files. Both logical and physical

There are two terms

1. Protection
    - All kinds threats are comes under the protection mechanism
    - Providing mechanism for controlling the access to prog, process, user to resource
2. Security
    - It deals with external threats(for security purpose firewalls, encryption are used.)

There are two types of threads

- Internal (Threats to information are internal)
- External (External user trying to access the resources called External threats)




# Goals of Protection

- As computer systems have become more sophisticated and universal in their applications, the need to protect their integrity has also grown.
- Protection was originally considered as an adjunct to multiprogramming operating systems, so that unreliable users might safely share a common

logical name space, such as a directory of files, or share a common physical name space, such as memory.

- Modern protection concepts have evolved to increase the reliability of any complex system that makes use of shared resources.
- We need to provide protection for several reasons. The most obvious is the need to prevent the mischievous, intentional violation of an access restriction by a user.
- Safe sharing of a common logical address space(directory of files) or common physical address space(memory). If any one want to share a file or directory, it should be shared safely.
- Fair and reliable resource usage. Program in the system resource can use the reliable resource only accordince to the policies which are stated that particular process
    - E.g. Process P1--→ granted to use only Printer,. P1→ is not granted to access using printer, using scanner, and etc. that will be unfair

## Principles of Protection,

- The ***principle of least privilege*** dictates that programs, users, and systems be given just enough privileges to perform their tasks. This ensures that failures do the least amount of harm and allow the least of harm to be done.
- Consider the similarity of a security guard with a passkey. If this key allows the guard into just the public areas that she guards, then misuse of the key will result in minimal damage. If, however, the passkey allows access to all areas, then damage from its being lost, stolen, misused, copied, or otherwise compromised will be much greater.
- An operating system following the principle of least privilege implements its features, programs, system calls, and data structures so that failure or compromise of a component does the minimum damage and allows the minimum damage to be done.
- The overflow of a buffer in a system daemon might cause the daemon process to fail, for example, but should not allow the execution of code from the daemon process's stack that would enable a remote user to gain maximum privileges and access to the entire system

## Domain of Protection,

- A computer system is a collection of processes and objects. By ***objects,*** we mean both **hardware objects** (such as the CPU, memory segments,

printers, disks, and tape drives) and **software objects** (such as files, programs, and semaphores).

- Each object has a unique name that differentiates it from all other objects in the system, and each can be accessed only through well-defined and meaningful operations
- The operations that are possible may depend on the object.
- For example,
  - on a CPU, we can only execute.
  - Memory segments can be read and written,
  - CD-ROM or DVD-ROM can only be read.
  - Tape drives can be read, written, and rewound.
  - Data files can be created, opened, read, written, closed, and deleted
  - program files can be read, written, executed, and deleted.
- A process should be allowed to access only those resources for which it has authorization.
- Furthermore, at any time, a process should be able to access only those resources that it currently requires to complete its task.
- This second requirement, commonly referred to as the **need-to-know principle**, is useful in limiting the amount of damage a faulty process can cause in the system.
- For example, when process *p* invokes procedure A(), the procedure should be allowed to access only its own variables and the formal parameters passed to it; it should not be able to access all the variables of process *p.*

## Domain Structure

- To facilitate the scheme just described, 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>. For example, if domain *D* has the access right <file *F*, *{*read,write*}*>, then a process executing in domain *D* can both read and write file *F.* It cannot, however, perform any other operation on that object.
- A domain can be realized in a variety of ways:
- Each ***user*** may be a domain. In this case, the set of objects that can be accessed depends on the identity of the user. Domain switching occurs

when the user is changed—generally when one user logs out and another user logs in.

- Each ***process*** may be a domain. In this case, the set of objects that can be accessed depends on the identity of the process. Domain switching occurs when one process sends a message to another process and then waits for a response.
- Each ***procedure*** may be a domain. In this case, the set of objects that can be accessed corresponds to the local variables defined within the procedure. Domain switching occurs when a procedure call is made.

## Access Matrix

- Our general model of protection can be viewed abstractly as a matrix, called an **access matrix**.
- The rows of the access matrix represent domains, and the columns represent objects.
- Each entry in the matrix consists of a set of access rights. Because the column defines objects explicitly, we can omit the object name from the access right.
- The entry access(*i,j*) defines the set of operations that a process executing in domain *Di* can invoke on object *Oj* .
- To illustrate these concepts, we consider the access matrix shown in following Figure
- There are four domains and four objects—three files (*F*1, *F*2, *F*3) and one laser printer.
- A process executing in domain *D*1 can read files *F*1 and *F*3. A process executing in domain *D*4 has the same privileges as one executing in domain *D*1; but in addition, it can also write onto files *F*1 and *F*3. The laser printer can be accessed only by a process executing in domain *D*2.

| domain \ object | $F_1$ | $F_2$ | $F_3$ | printer |
|---|---|---|---|---|
| $D_1$ | read | | read | |
| $D_2$ | | | | print |
| $D_3$ | | read | execute | |
| $D_4$ | read<br>write | | read<br>write | |

- The access-matrix scheme provides us with the mechanism for specifying a variety of policies for the implementation of protection model.
- Any matrix have two entries i.e. row(i) and column(j).
- Each entry in the matrix consist of a set of access rights.
- The entry access (i,j) defines the set of operations that a process, executing in domain can invoke an objects oj.
- It must be ensured that a process executing in domain Di can access only those objects specified in row i

## Implementation of Access Control

- What is Access Control?
  - Access control is a security technique that can be used to regulate who or what can view or use resources in a computing environment. There are two main types of access control: Physical and Logical.
  - Physical access control limits access to campuses, buildings, rooms and physical IT assets.
  - Logical access limits connections to computer networks, system files and data.
  - Access controls are security features that control how users and systems communicate and interact with other systems and resources.
  - Access is the flow of information between a subject and a resources. A subject is an active entity that requests access to a resource or the data within a resource.
- E.g. : USER, PROGRAM, PROCESS etc.
- A resource is an entity that contains the information. E.g. Computer, Database, File, Program, Printer etc.
- Access control gives organization the ability to control, restrict, monitor, and protect resource availability, integrity and confidentiality.
- Access control is the process of enforcing the required security for a particular resource.
- Once we know who a user is, and we know what authorization level they have and what we should and should not give them access to, we need to physically prevent that user from accessing anything that they should not be able to.
- Access control can be seen as the combination of authentication and authorization plus additional measures, such as clock- or IP-based restrictions.

- In the context of a web or software application, access control may be implemented using bespoke logic, security features of the development framework being used, file permissions, URL access lists, or many other mechanisms.
- Note that lack of adequate access control is more often the cause of security vulnerabilities in applications than faulty authentication or authorization mechanisms, simply because access control is more complex to implement and becomes more complex as the application being secured becomes more complex itself.

## Revocation of Access Rights

- In a dynamic protection system, we may sometimes 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? If revocation is delayed, can we find out when it will take place?
- **Selective versus general**. When an access right to an object is revoked, does it affect all the users who have an access right 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 the 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?
- Schemes that implement revocation for capabilities include the following:
- **Reacquisition**. Periodically, capabilities are deleted from each domain. If a process wants to use a capability, it may find that that capability has been deleted. The process may then try to reacquire the capability. If access has been revoked, the process will not be able to reacquire the capability.
- **Back-pointers**. A list of pointers is maintained with each object, pointing to all capabilities associated with that object. When revocation is required, we can follow these pointers, changing the capabilities as necessary
- **Indirection**. The capabilities point indirectly, not directly, to the objects. Each capability points to a unique entry in a global table, which in turn points to the object. We implement revocation by searching the global table for the desired entry and deleting it. Then, when an access is attempted, the capability is found to point to an illegal table entry. Table

entries can be reused for other capabilities without difficulty, since both the capability and the table entry contain the unique name of the object.

- **Keys**. A key is a unique bit pattern that can be associated with a capability. This key is defined when the capability is created, and it can be neither modified nor inspected by the process that owns the 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

## Capability-Based Systems

- These systems vary in their complexity and in the types of policies that can be implemented on them.
  - **HYDRA**
    - Hydra is a capability-based protection system that provides considerable flexibility
    - A fixed set of possible access rihts is known to and interpreted by the system.
    - These rights include such basic forms of access as the right to read, write, or execute memory segment.
    - Also, a user (of the protection system) can declare other rights.
    - Operations on objects are defined procedurally.
    - The Procedure that implement such operations are themselves a form of object, and they are accessed indirectly by capabilities.
    - When the definition of an object is made known to Hydra, the names of operations on the type become auxiliary rights.
- Hydra also provides rights amplification.
- This scheme allow a procedure to be certified as trustworthy to act on a formal parameter of a specified type on behalf of any process that holds a right to execute the procedure.
- A hydra subsystem is built on top of its protection kernel and may require protection of its own components.
- A subsystem interacts with the kernel through calls on a set of kernel-defined primitives that define access rights to resources defined by the subsystem.
- A user of the Hydra system would explicitly incorporate calls on these system procedure into the code of programs or would use a program translator that had been interfaced to Hydra.

## Cambridge Based Systems

- Cambridge CAP System capability is similar and superficially less powerful than that of Hydra.
- CAP has two kind of capabilities.
  - The ordinary kind is called a **data capability**. It can be used to provide access to objects, but the only rights provided are the standard read, write, and execute of the individual storage segments associated with the object.
  - The second kind of capability is the **Software capability,** which is protected but not interpreted, by the CAP microcode but by application programmers.
  - When executing the code body of such a procedure, a process temporarily acquires the right to read or write the contents of a software capability itself.
  - The specific kind of rights amplification corresponds to an implementation of the seal and unseal primitives on capabilities.