

# Protection

Protection – the mechanism of controlling access to resources for programs, processes and users.

Subjects – the active components in a system that can use resources (users, programs, processes). Also referred to as principals.

Objects – the resources being used (programs, processes, files, memory, communication channels, devices, databases, semaphores)

Objects can also be subjects.

We will assume that we have authenticated the subjects, so what we are concerned with here is how to ensure subjects only access objects in permitted ways.

We will look at authentication in the security section.

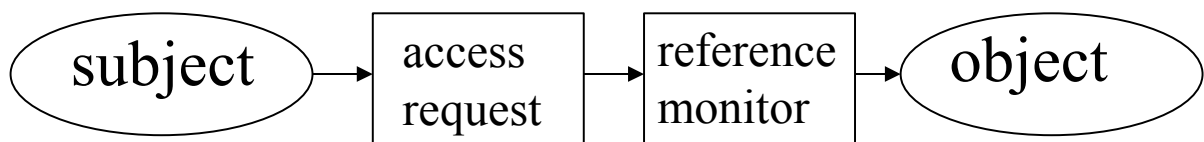
# Goals

Protect against

- malicious intent
- stupidity
- accident
- errors

Each object in the system has a number of operations that can be performed on it. Not only do we not want any other access than the permitted operations we would also like to limit access to the minimum required to achieve the allowed goals – the **need to know** principle.

All accesses to objects should be mediated by a **reference monitor**.



## Examples of protection

We have already seen several examples of protection in this course:

- Privileged instructions – the process must be executing in kernel mode in order to execute without causing an exception.
- Memory protection – the kernel address space is protected from user level instructions. Similarly one process' address space is protected from access by another.
- File system – one user's files are protected from access by another user.

What is the reference monitor in each of these cases?

# Protection Domains

Access rights are commonly associated with protection domains.

A process executes inside a protection domain. The process then has the rights and privileges of the domain.

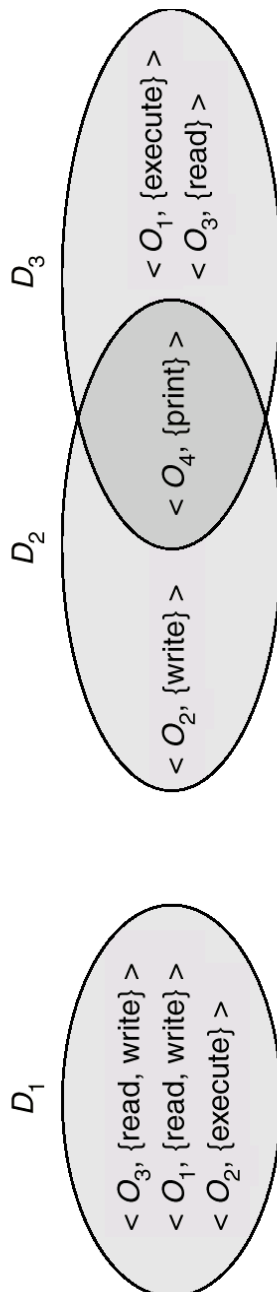
Thus many processes can have the same rights if they execute in the same domain.

We will see that there are too many subjects, objects and access rights in a normal system to explicitly keep information about all of them.

So this combining is the first attempt to decrease the amount of protection information which the system needs to maintain.

A domain is a collection of ordered pairs  
<object, rights> - Java has permissions like this.

# Intersection of domains



Domains can overlap.

In this case the permission in the overlap is available to both.

Our programs frequently have to move from one domain to another.

This switching can only be allowed if the start domain has the permission to change to another domain.

Domains can be associated with users, locations (e.g. URLs), programs, processes ...

# Crossing domains

Crossing domains is dangerous and is commonly used to attack systems.

## Why do we need it?

We want users to have controlled access to resources they don't have direct access to.

e.g. a database, particular hardware, networks

So we give the user access to a program that does have access to the restricted resource.

The users domain allows access to the program, the programs domain allows access to the resource.

## UNIX

Domains are associated with users (and the groups they belong to).

When a program is run it takes on the permissions of the user (both individual and group permissions).

We can set programs to take on the permissions of the group or owner of the program file instead.

The program becomes a setuid or setgid program.

## How to setuid

After these changes anyone (not in my group) can run the *program* file.

When they do so, the process uses my permissions. If it wasn't setuid then the process would have run with their permissions.

This is really dangerous, especially if I am the superuser. If they can start another process from within the *program* process that new process would have my permissions as well.

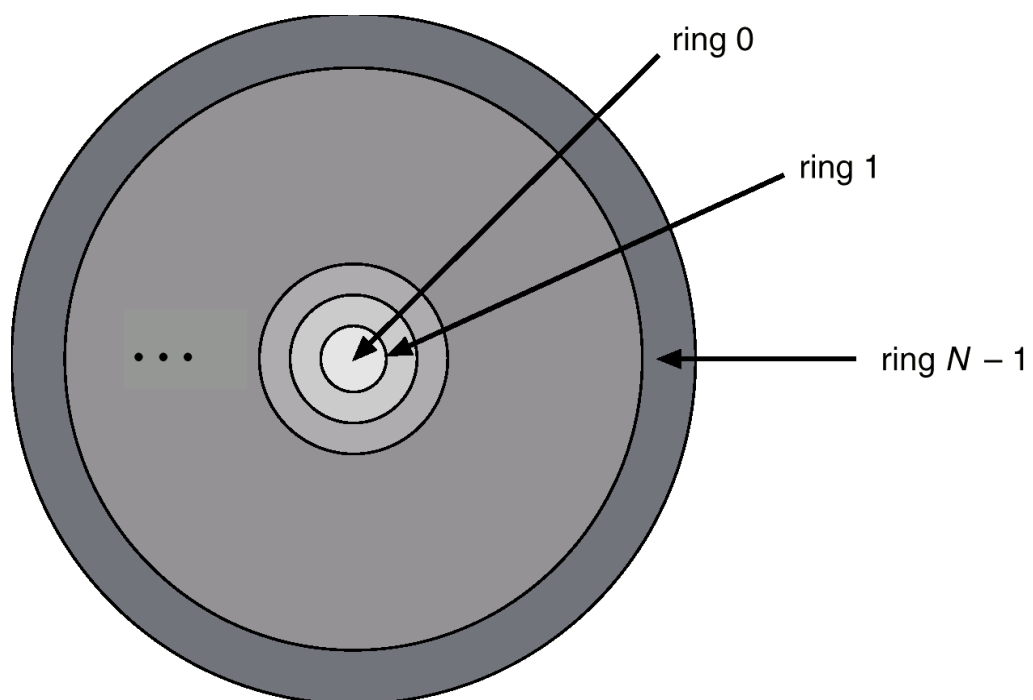
```
-rwxr--r--  1 robert-s staff  21 Oct  8 15:36 program
bash-2.05$ chmod u+s program
-rwsr--r--  1 robert-s staff  21 Oct  8 15:36 program
bash-2.05$ chmod o+x program
-rwsr--r-x  1 robert-s staff  21 Oct  8 15:36 program
```

# Multics ring structure

Let  $D_i$  and  $D_j$  be any two domain rings.

If  $j < i \Rightarrow D_i \subseteq D_j$

A process executing in  $D_j$  has more privileges than one executing in  $D_i$ .





## Multics segments

Each file is loaded as a segment. It has associated permissions – read, write, execute – and a ring number (the ring it runs in or is loaded into).

Access to other segments depends on both the current ring number, the ring number of the other segment and the type of access required.

The current ring number is maintained when a lower permission ring is entered by a process.

Sometimes a lower permission segment needs to access a segment in an higher permission ring.

There are specified entry points which allow this – more access is allowed under controlled conditions.

## Other approaches to domain switches

Special directories – programs in these directories run with the access privileges associated with the directory.

This is slightly safer than setuid programs because all privileged processes must be in these directories rather than scattered all around the system.

Have server processes running with the necessary privileges – the normal user processes send messages to the server process when they need the privileged access.

Of course a system call is a change of domain and the hardware guarantees that when the call returns the domain reverts to its previous status.

All such techniques require great care.

# 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  $Object_j$

<div>object</div> <div>domain</div>	$F_1$	$F_2$	$F_3$	printer
$D_1$	read		read	
$D_2$				print
$D_3$		read	execute	
$D_4$	read write		read write	

# Changing permissions

When an object is created a new column is added and the permissions are set (usually by the creator/owner).

The domains are objects as well.

This way we can control access to the domains.

Transfer to another domain – switch e.g. a process executing in  $D_2$  can switch to  $D_3$  or  $D_4$

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			

## Changing in a column

Copy right “\*” signifies the permission can be copied. Can only work on the same object/column.

Owner right means any values on the object/column can be changed.

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)			

## Changing in a row

The control right allows one domain to remove rights from another domain.

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					
$D_4$	read write		read write		switch			

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			

# Before next time

## Read from the textbook

18.1 – Goals of Protection

18.2 – Domain of Protection

18.3 – Access Matrix