Directory Implementation

Two important directory implementation algorithms are

Linear List:

- The simplest method of implementing a directory is to use a linear list of file names with pointers to the data blocks.
- This method is simple to program but time-consuming to execute.
- To create a new file. we must first search the directory to be sure that no existing file has the same name. Then, we add a new entry at the end of the directory.
- To delete a file, we search the directory for the named file, then release the space allocated to it.

Hash Table:

- With this method, a linear list stores the directory entries, but a hash data structure is also used (In **hashing**, large keys are converted into small keys by using **hash** functions. The values are then stored in a data structure called **hash table)**.
- The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list. Therefore, it can greatly decrease the directory search time.
- Insertion and deletion are easy to implement, but hash table suffers from collisions—situations in which two file names hash to the same location.

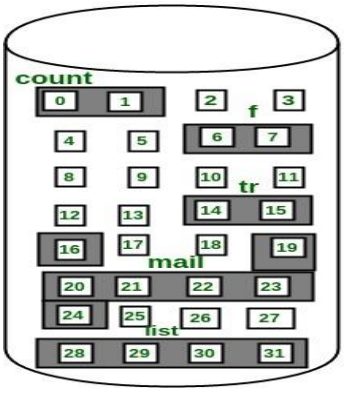
Space Allocation

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

Contiguous Allocation

- Each file occupies a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement. the blocks assigned to the file will be: *b, b+1, b+2,……b+n-1.*
- External fragmentation is a major issue with this type of allocation technique.

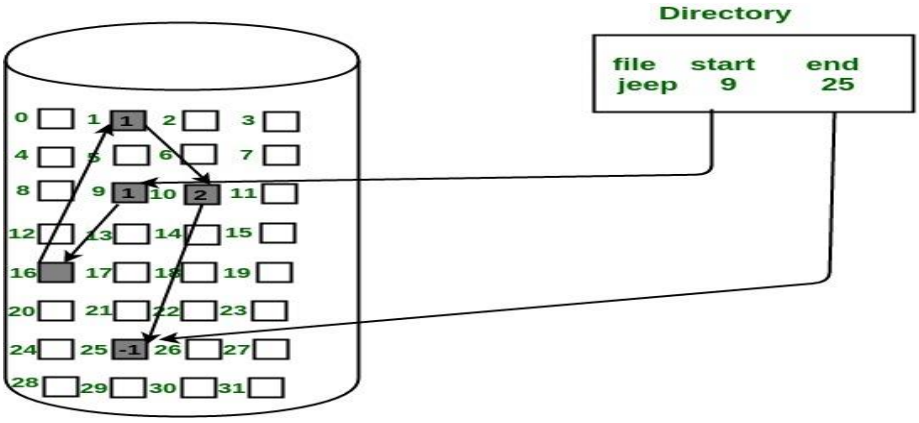


Directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

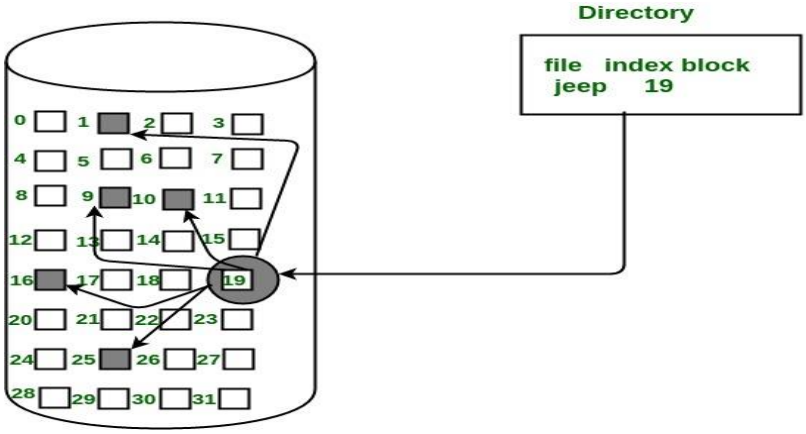
Linked Allocation

- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file.
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.



Indexed Allocation

- Provides solutions to problems of contiguous and linked allocation.
- A index block is created having all pointers to files.
- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.



FREE SPACE MANAGEMENT

✓ Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.

✓ To keep track of free disk space, the system maintains a free-space list.

✓ The free-space list records all free disk blocks – those not allocated to some file or directory.

✓ To create a file, we search the free-space list for the required amount of space, and allocate that space to the new file.

✓ this space is then removed from the free-space list.

✓ When a file is deleted, its disk space is added to the free-space list.

1. Bit Vector

- The free-space list is implemented as a bit map or bit vector.
- Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.
- For example, consider a disk where block
- 2,3,4,5,8,9,10,11,12,13,17,18,25,26 and 27 are free, and the rest of the block are allocated. The free space bit map would be
- 001111001111110001100000011100000 ...
- The main advantage of this approach is its relatively simplicity and efficiency in finding the first free block, or n consecutive free blocks on the disk.

2. Linked List

❖ another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
❖ this first block contains a pointer to the next free disk block, and so on. In our example, we would keep a pointer to block 2, as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.
❖ However, this scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time.

3. Grouping

- ❖ A modification of the free-list approach is to store the addresses of n free blocks in the first free block.
- ❖ The first n-1 of these blocks are actually free.
- ❖ The last block contains the addresses of another n free blocks, and so on.
- ❖ The importance of this implementation is that the addresses of a large number of free blocks can be found quickly.

4. Counting

- We can keep the address of the first free block and the number n of free contiguous blocks that follow the first block.
- Each entry in the free-space list then consists of a disk address and a count.
- Although each entry requires more space than would a simple disk address, the overall list will be shorter.

Recovery

Backup and Restore

- Magnetic disks sometimes fail, and care must be taken to ensure that the data lost in such a failure are not lost forever.
- To this end, system programs can be used to back up data from disk to another storage device, such as a floppy disk, magnetic tape, optical disk, or other hard disk.
- Recovery from the loss of an individual file, or of an entire disk, may then be a matter of restoring the data from backup.

A typical backup schedule may then be as follows:

- Day 1: Copy to a backup medium all files from the disk. This is called a **full backup.**
- Day 2: Copy to another medium all files changed since day 1. This is an **incremental backup.**
- Day 3: Copy to another medium all files changed since day 2.
- Day 4: Copy to another medium all files changed since day N-1.then go back to day 1.

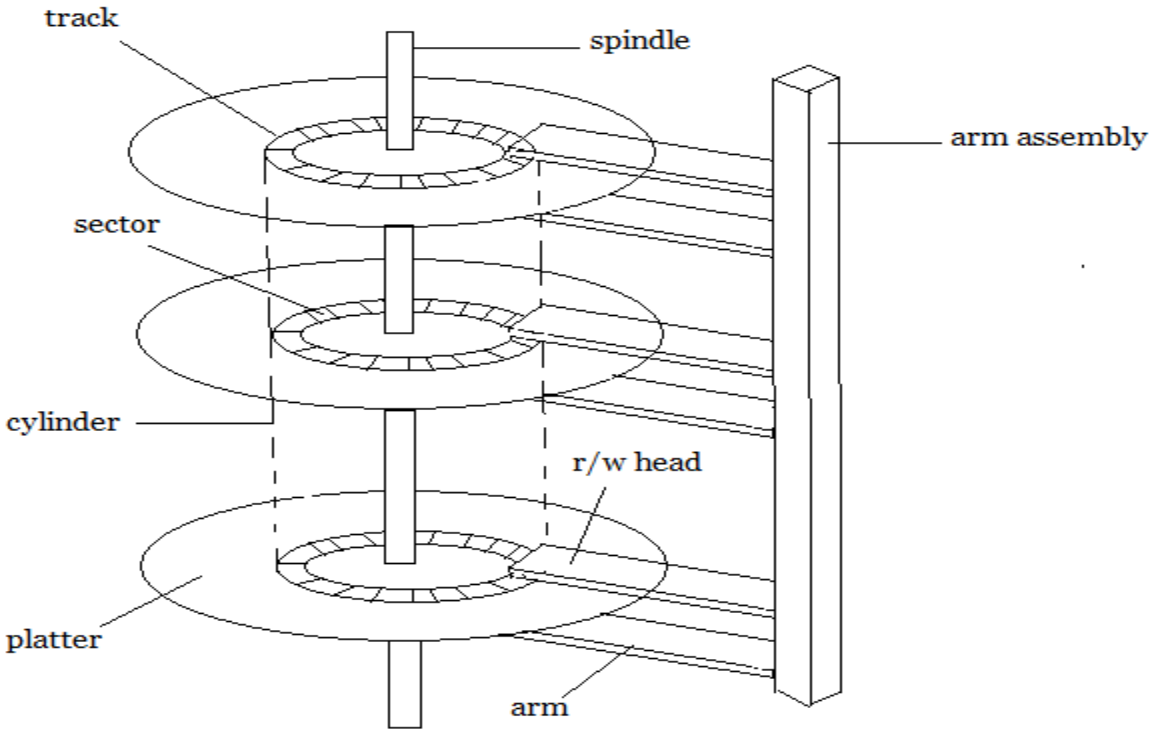
Secondary Storage Structure

Secondary storage devices are those devices whose memory is non volatile, meaning, the stored data will be intact even if the system is turned off. Here are a few things worth noting about secondary storage.

- Secondary storage is also called auxiliary storage.
- Secondary storage is less expensive when compared to primary memory like RAMs.
- The speed of the secondary storage is also lesser than that of primary storage.
- Hence, the data which is less frequently accessed is kept in the secondary storage.
- A few examples are magnetic disks, magnetic tapes, removable thumb drives etc.

Magnetic Disk Structure

In modern computers, most of the secondary storage is in the form of magnetic disks. Hence, knowing the structure of a magnetic disk is necessary to understand how the data in the disk is accessed by the computer.



Structure of a magnetic disk

A magnetic disk contains several **platters**.
Each platter is divided into circular shaped **tracks**.
The length of the tracks near the center is less than the length of the tracks farther from the center.
Each track is further divided into **sectors**.
Tracks of the same distance from center form a cylinder. A read-write head is used to read data from a sector of the magnetic disk.
The speed of the disk is measured as two parts:

- **Transfer rate:** This is the rate at which the data moves from disk to the computer.
- **Random access time:** It is the sum of the seek time and rotational latency.

Seek time is the time taken by the arm to move to the required track.

Rotational latency is defined as the time taken by the arm to reach the required sector in the track.

Even though the disk is arranged as sectors and tracks physically, the data is logically arranged and addressed as an array of blocks of fixed size. The size of a block can be **512** or **1024** bytes. Each logical block is mapped with a sector on the disk, sequentially. In this way, each sector in the disk will have a logical address.

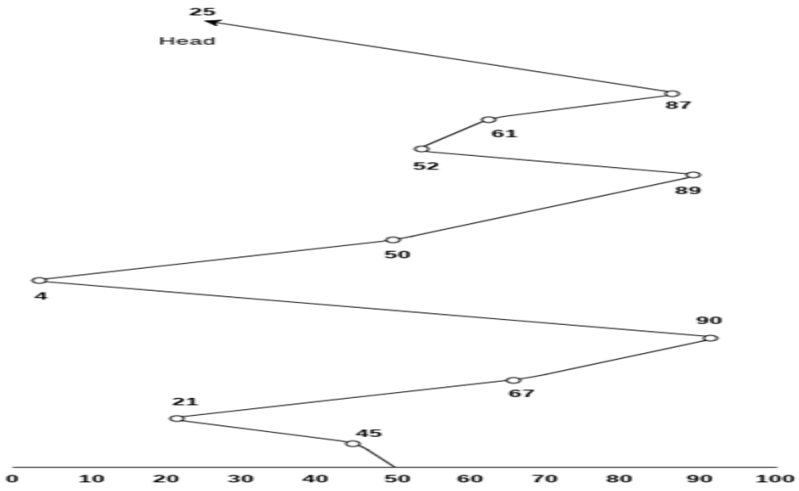
Disk Scheduling Algorithms

On a typical multiprogramming system, there will be multiple disk access requests at any point of time. So those requests must be scheduled to achieve good efficiency. Disk scheduling is similar to process scheduling.

First Come First Serve(FCFS):

This algorithm performs requests in the same order asked by the system. Example
Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25
Head pointer starting at 50 and moving in left direction.
Find the number of head movements in cylinders using FCFS scheduling.

Solution



Number of cylinders moved by the head

= (50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(61-52)+(87-61)+(87-25)

= 5 + 24 + 46 + 23 + 86 + 46 + 49 + 9 + 26 + 62

= 376

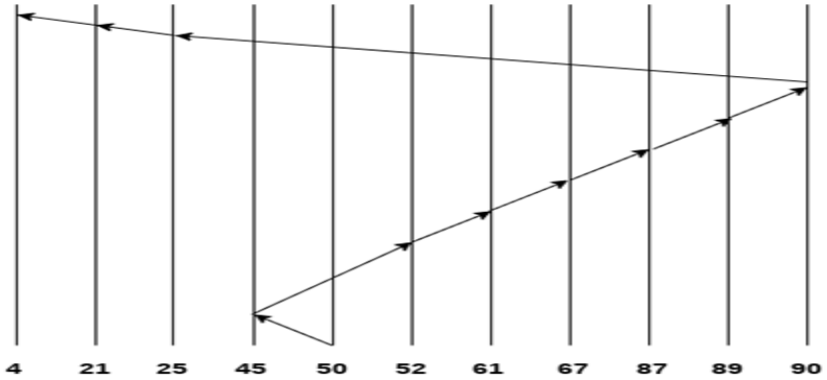
Shortest Seek Time First (SSTF):

Here the position which is closest to the current head position is chosen first.

Example

Consider the following disk request sequence for a disk with 100 tracks
45, 21, 67, 90, 4, 89, 52, 61, 87, 25
Head pointer starting at 50. Find the number of head movements in cylinders using SSTF scheduling.

Solution:



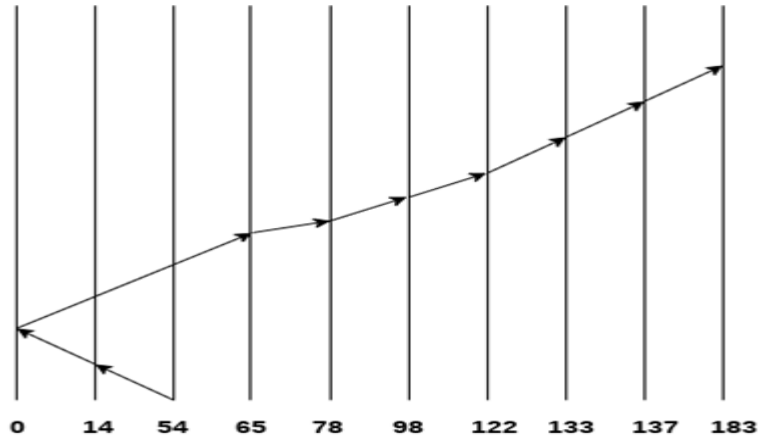
Number of cylinders = 5 + 7 + 9 + 6 + 20 + 2 + 1 + 65 + 4 + 17 = 136

SCAN algorithm:

This algorithm is also called the **elevator algorithm** because of its behavior.

Here, first the head moves in a direction (say backward) and covers all the requests in the path. Then it moves in the opposite direction and covers the remaining requests in the path. This behavior is similar to that of an elevator.

Example
Consider the following disk request sequence for a disk with 100 tracks
98, 137, 122, 183, 14, 133, 65, 78
Head pointer starting at 54 and moving in left direction.
Find the number of head movements in cylinders using SCAN scheduling.



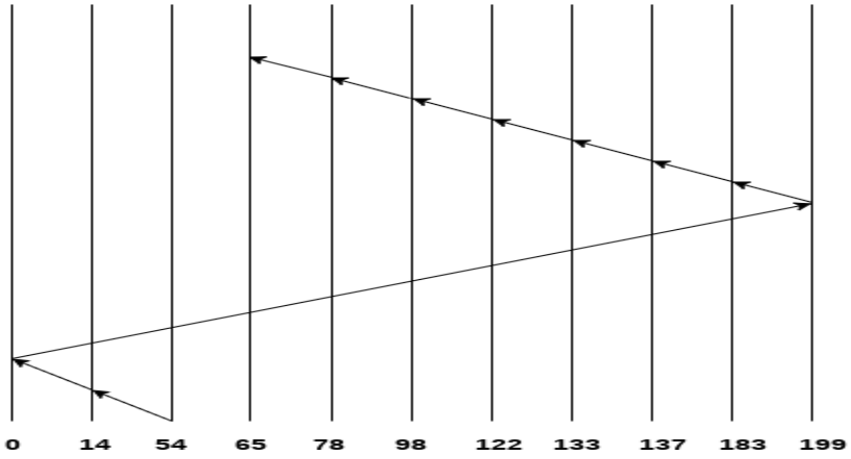
Number of Cylinders = 40 + 14 + 65 + 13 + 20 + 24 + 11 + 4 + 46 = 237

Circular Scan (C-SCAN)

In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction

without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

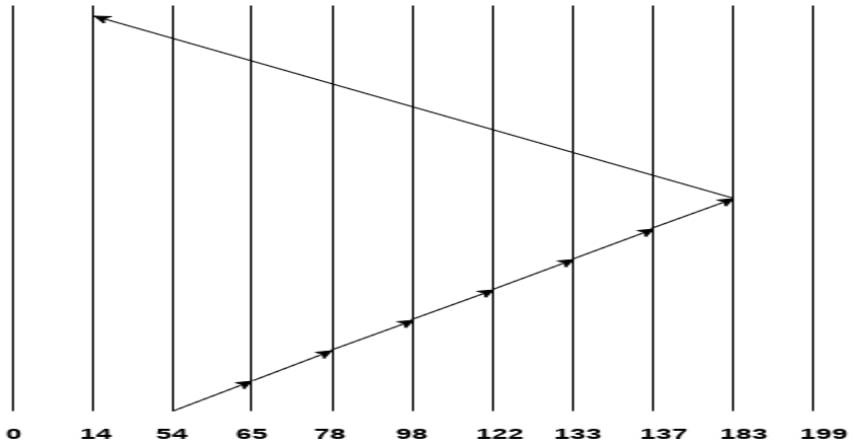
Example
Consider the following disk request sequence for a disk with 100 tracks
98, 137, 122, 183, 14, 133, 65, 78
Head pointer starting at 54 and moving in left direction.
Find the number of head movements in cylinders using C-SCAN scheduling.



No. of cylinders crossed = 40 + 14 + 199 + 16 + 46 + 4 + 11 + 24 + 20 + 13 = 387

C-LOOK

Example
Consider the following disk request sequence for a disk with 100 tracks
98, 137, 122, 183, 14, 133, 65, 78
Head pointer starting at 54 and moving in left direction.
Find the number of head movements in cylinders using C LOOK scheduling.



Number of cylinders crossed = 11 + 13 + 20 + 24 + 11 + 4 + 46 + 169 = 298

This is the enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request.

Disk Management

Disk Formatting

- Before a disk can be used, it has to be **low-level formatted**, which means laying down all of the headers and trailers demarking the beginning and ends of each sector. Included in the header and trailer are the linear sector numbers, and **error-correcting codes, ECC,** which allow damaged sectors to not only be detected, but in many cases for the damaged data to be recovered
- ECC calculation is performed with every disk read or write, and if damage is detected but the data is recoverable, then a **soft error** has occurred. Soft errors are generally handled by the on-board disk controller, and never seen by the OS.
- Once the disk is low-level formatted, the next step is to partition the drive into one or more separate partitions. This step must be completed even if the disk is to be used as a single large partition.
- After partitioning, then the file systems must be **logically formatted,** which involves laying down the master directory information (FAT table or inode structure), initializing free lists, and creating at least the root directory of the file system.

Boot Block

- Computer ROM contains a **bootstrap** program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller, load that sector into memory, and transfer control over to it.

- The first sector on the hard drive is known as the **Master Boot Record, MBR,** and contains a very small amount of code in addition to the **partition table.** The partition table documents how the disk is partitioned into logical disks, and indicates specifically which partition is the **active** or **boot** partition.
- The boot program then looks to the active partition to find an operating system, possibly loading up a slightly larger / more advanced boot program along the way.
- Once the kernel is found by the boot program, it is loaded into memory and then control is transferred over to the OS. The kernel will normally continue the boot process by initializing all important kernel data structures, launching important system services (e.g. network daemons, scheduler, init, etc.), and finally providing one or more login prompts.

Bad Blocks

- No disk can be manufactured to 100% perfection, and all physical objects wear out over time. For these reasons all disks are shipped with a few bad blocks, and additional blocks can be expected to go bad slowly over time. If a large number of blocks go bad then the entire disk will need to be replaced.
- Modern disk controllers make much better use of the error-correcting codes, so that bad blocks can be detected earlier and the data usually recovered.
- If the data on a bad block cannot be recovered, then a **hard error** has occurred. which requires replacing the file(s) from backups, or rebuilding them from scratch.

Swap-Space Management

- Modern systems typically swap out pages as needed, rather than swapping out entire processes. Hence the swapping system is part of the virtual memory management system.
- Managing swap space is obviously an important task for modern Operating systems.

Swap-Space Use

- The amount of swap space needed by an OS varies greatly according to how it is used. Some systems require an amount equal to physical RAM, some want a multiple of that, some want an amount equal to the amount by which virtual memory exceeds physical RAM, and some systems use little or none at all!
- Some systems support multiple swap spaces on separate disks in order to speed up the virtual memory system.

Swap-Space Location

Swap space can be physically located in one of two locations:

- As a large file which is part of the regular file system
- As a raw partition, possibly on a separate or little-used disk. This allows the OS more control over swap space management, which is usually faster and more efficient.

Disk Attachment

Disk drives can be attached either directly to a particular host (a local disk) or to a network.

Host-Attached Storage

- Local disks are accessed through I/O Ports.
- High end workstations or other systems in need of larger number of disks

Network-Attached Storage

- Network attached storage connects storage devices to computers using a remote procedure call, RPC, interface, typically with something like NFS filesystem mounts. This is convenient for allowing several computers in a group common access and naming conventions for shared storage.
- NAS can be implemented using Internet protocols and standard network connections, allowing long-distance remote access to shared files.
- NAS allows computers to easily share data storage, but tends to be less efficient than standard host-attached storage.

Protection

Protection dealt with protecting files and other resources from accidental misuse by cooperating users sharing a system, generally using the computer for normal purposes.

Goals of Protection

- To prevent malicious misuse of the system by users or programs.
- To ensure that each shared resource is used only in accordance with system *policies,* which may be set either by system designers or by system administrators.
- To ensure that errant programs cause the minimal amount of damage possible.
- Note that protection systems only provide the *mechanisms* for enforcing policies and ensuring reliable systems. It is up to administrators and users to implement those mechanisms effectively.

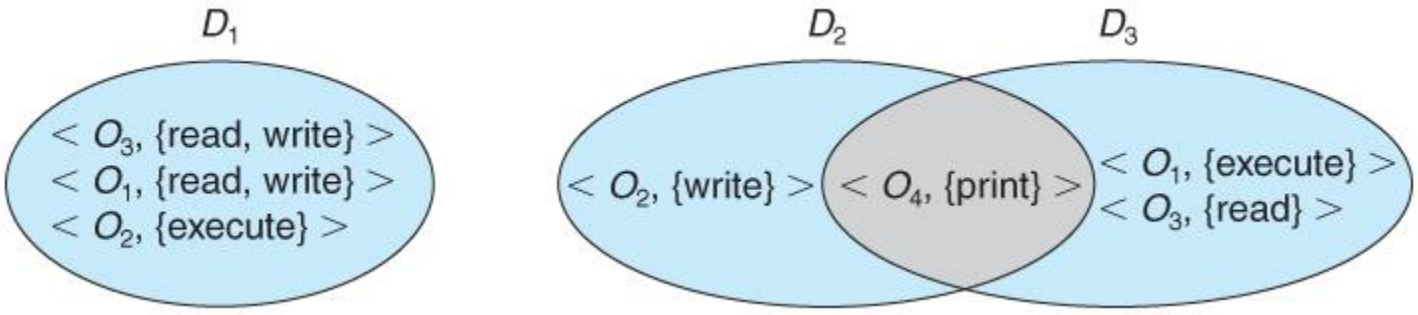
Mechanism determine how something will be done and
Policies decide what will be done.

Domain of Protection

- A computer can be viewed as a collection of *processes* and *objects* (both HW & SW).
- The **need to know principle** states that a process should only have access to those objects it needs to accomplish its task, and furthermore only in the modes for which it needs access and only during the time frame when it needs access.
- The modes available for a particular object may depend upon its type.

Domain Structure

- A **protection domain** specifies the resources that a process may access.
- Each domain defines a set of objects and the types of operations that may be invoked on each object.
- An **access right** is the ability to execute an operation on an object.
- A domain is defined as a set of < object, { access right set } > pairs, as shown below. Note that some domains may be disjoint while others overlap.



System with three protection domains.

- The association between a process and a domain may be *static* or *dynamic.*
    - If the association is static, then the set of resources available to the process are fixed throughout its lifetime.
    - If the association is dynamic, then the resources available to the process keep changing dynamically. There needs to be a mechanism for **domain switching.**

An Example: MULTICS
The MULTICS system uses a complex system of rings, each corresponding to a different protection domain, as shown below:

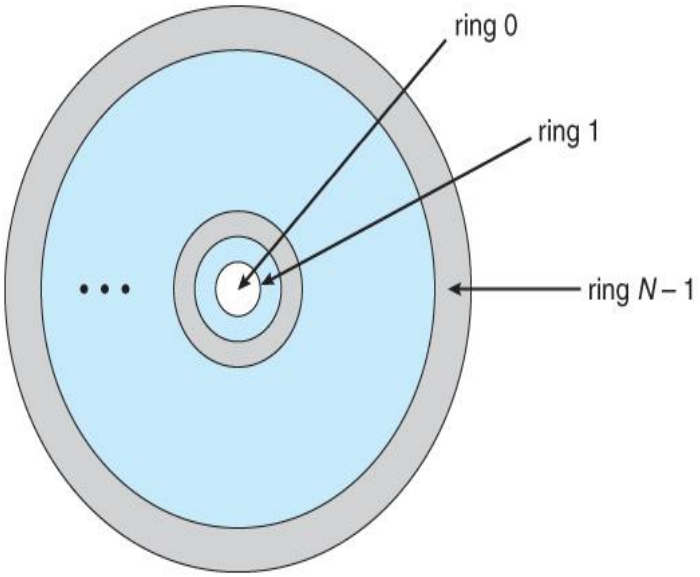


Fig-MULTICS ring structure.

- Rings are numbered from 0 to 7, with outer rings having a subset of the privileges of the inner rings.
- Each file is a memory segment, and each segment description includes an entry that indicates the ring number associated with that segment, as well as read, write, and execute privileges.
- Each process runs in a ring, according to the current-ring-number, a counter associated with each process.
- A process operating in one ring can only access segments associated with higher rings, and then only according to the access bits. Processes cannot access segments associated with lower rings.
- Domain switching is achieved by a process in one ring calling upon a process operating in a lower ring, which is controlled by several factors stored with each segment descriptor.

Access Matrix

- The model of protection that we have been discussing can be viewed as an ***access matrix,*** in which columns represent different system resources and rows represent different protection domains. Entries within the matrix indicate what access that domain has to that resource.

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	

Allow a process operating in one domain to affect the rights available in other domains. For example in the table below, a process operating in domain D2 has the right to control any of the rights in domain D4.

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			

Modified access matrix

Revocation of Access Rights

The need to revoke access rights dynamically raises several questions:

- Immediate versus delayed - If delayed, can we determine when the revocation will take place?
- Selective versus general - Does revocation of an access right to an object affect *all* users who have that right, or only some users?
- Partial versus total - Can a subset of rights for an object be revoked, or are all rights revoked at once?
- Temporary versus permanent - If rights are revoked, is there a mechanism for processes to re-acquire some or all of the revoked rights.