Module 5

File-System Implementation and Protection

Syllabus

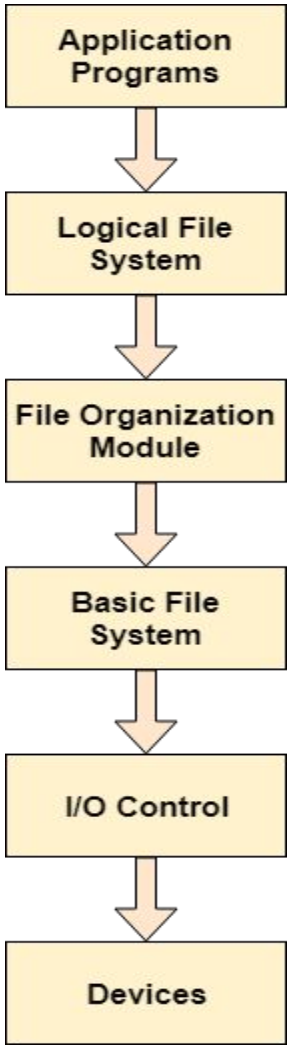
File-System structure, File-System Implementations, Directory Implementation, Allocation Methods, Free-space Management, Efficiency and Performance, Recovery

Disk Management: Disk Structure, Disk Scheduling, Disk Management, Swap-Space management, Disk Attachment

Protection and Security: Goals of Protection, Domain of Protection, Access Matrix, Implementation of Access Matrix, Revocation of Access Rights.

Filesystem Structure

- The filesystem is ***layered approach***.
- The lowest level is made up of device drivers along with interrupt handlers to transfer the information in between the device driver and the main memory.
- The human readable high-level commands are translated into a hardware specific instruction.
- The hardware specific instructions are interpreted by the I/O control level. It also controls the device location on which the command acts.
- Next is the ***basic file system*** makes use of generic commands to read or write into the physical blocks of device drivers.



Filesystem Structure

- Next is the ***file organization module*** translates the logical addresses into physical addresses with the knowledge of the physical blocks.
- This level also includes the ***free space manager*** which keeps track of free blocks. This information is used while allocating physical blocks.
- The **logical file system** includes the file system structure as metadata and uses the ***file control block*** to maintain the structure. This level is responsible for the ***secure access and the protection of files.***
- ***Application programs are the top most layer of the file system*** which accesses the files.

Filesystem Implementation

- A collection of structures is used for the implementation of the filesystem based on the OS and the filesystem.

Some of the structures used are as follows:

- ***Boot control block***: This structure contains booting information pertaining to the OS and is empty in the absence of the OS.
- ***Volume control block:*** Number of blocks, free block count, blocks in the partition and other volume related information is stored in the volume control block.
- ***Master file table***: This is a directory structure per file system which includes file names in the system along with the inode numbers.
- ***Per file FCB***: It contains the details regarding the file such as the size of data blocks, the location of data blocks, ownership, file permissions, etc.

Filesystem Implementation

- **In-memory mount table**: It holds the information about each mounted volume.
- **In-memory directory structure cache**: It contains recently accessed directories and its related information.
- **The system-wide open file table** holds every open file's FCB copy.
- **Per process open file table** has pointers to the entries in the system-wide open file table.

Filesystem Implementation

- When a new file is created, the application system looks into the logical file system for the format of the directory structure and a new or free **FCB (file control block**) is allocated and is updated in the system directory.
- While opening a file using open(), a pointer is returned to the appropriate entry in the per process file system table. Before the open() is applied, the open file table is checked to check whether the process is already open.
- When a file is closed by a process, per process table entry is deleted and the global open table count is decreased by one.

Filesystem Implementation

- The root partition contains the OS kernel and the other system files which are mounted during the boot time.
- When mounting takes place, the validity of the file system is checked. Mount table is updated as a file system is mounted along with the type of files held.

The two main functions of the Virtual File System are:

- *Separation of file system generic operations from their respective implementation by means of an interface*
- *VFS designates a network-wide unique code for the file which works throughout the network with the help of file representation structure called vnode .*

Four object types used in Linux are:

- **inode** object (for an individual file)
- **file object**(representing open file)
- **superblock object**(for the entire file system) and
- **dentry object**(for individual directory entry) to implement VFS.

Directory Implementation

Different forms of directory implementation are:

- **Linear list**: Though this method is time-consuming, it is considered as the simplest form of directory implementation.
- When a new file is to be created, all the entries in the linked list are searched to ensure that no such entry exists and the new file is appended at the end of the list.

Entries in the directory can be reused by:

- *allotting a special name*
- *maintaining a list of free entries*
- copying the last entry to the freed location and freeing the last entry

Directory Implementation

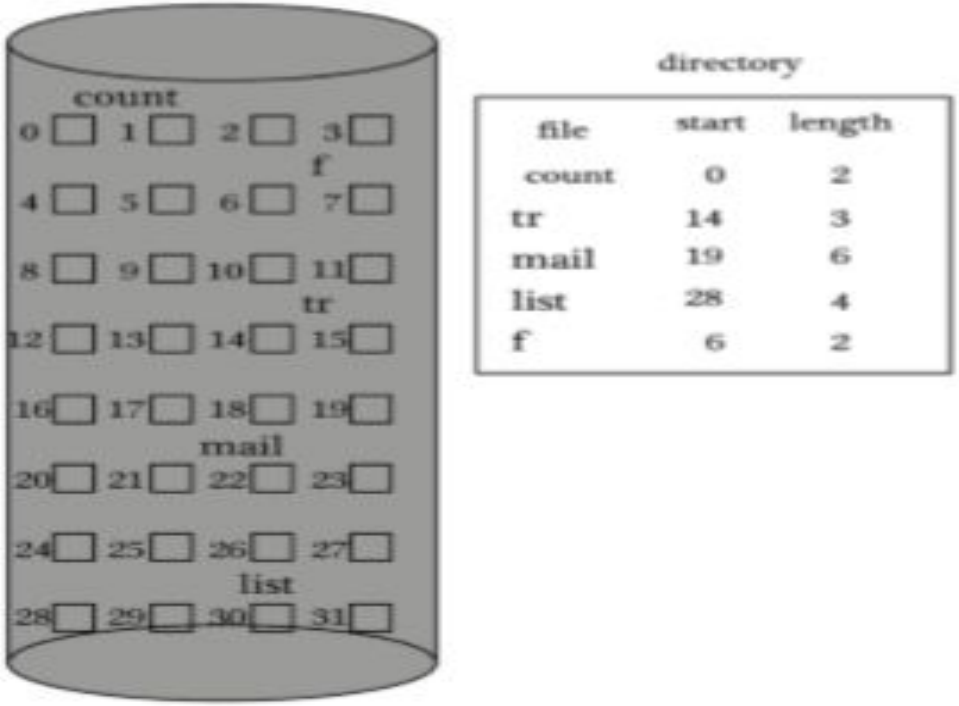
- **Hash table**: In this method, a linear list of the entries in the directory and a hash data structure is deployed.
- Hash table computes a value for the file name and returns a pointer to the file entry.

Allocation Methods

Allocation of disk space can be done in the following ways:

- ***Contiguous allocation***: Contiguous disk allocation involves the allocation of a set of contiguous blocks in a linear order on the disk. Accessing the next contiguous location in the block needs no head movement or in certain cases minimal head movement.
- Contiguous allocation of disk space to a file of n blocks, starting at block b occupies b, b+1, b+2,…, b+n-1.

Contiguous allocation

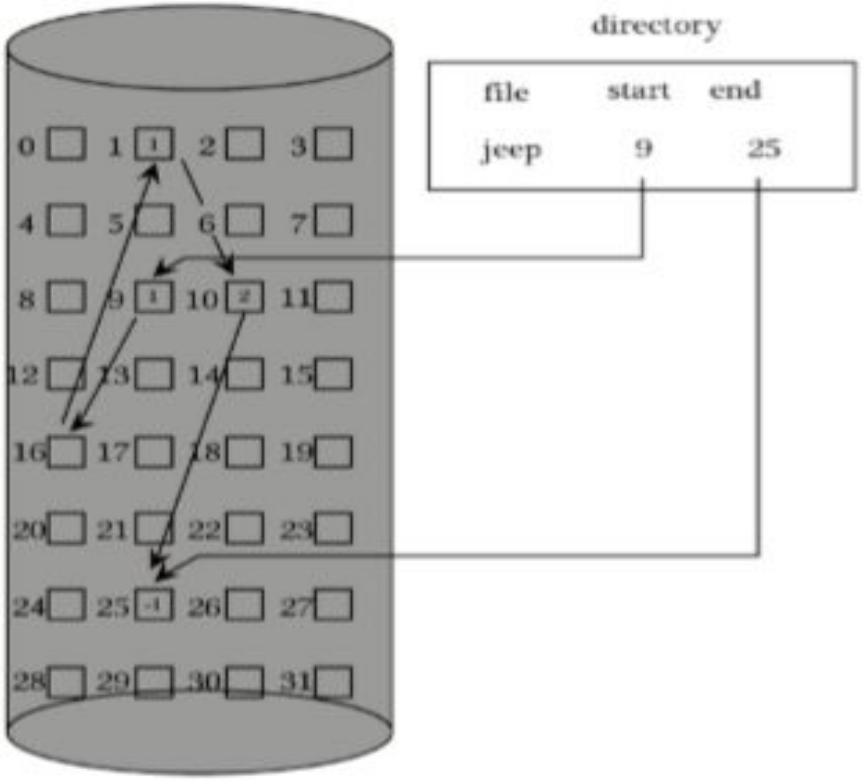


directory

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

Linked Allocation

- Linked allocation of disk space involves a linked list of disk blocks whose location can be scattered anywhere on the disk with the directory storing pointers to the first and last block of the file.
- External fragmentation does not occur here as only components of the file are linked together.



Indexed Allocation

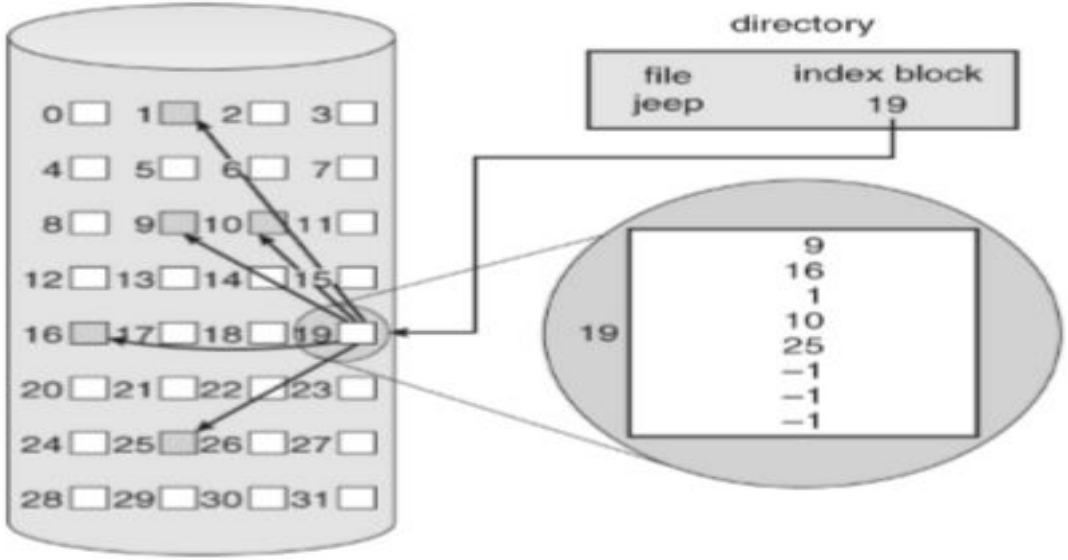
- An index block is used which stores the pointers to various blocks. When a file is created all the pointers in the index block are set to nil and the index block is updated whenever contents are written to any block.

The variations of indexed allocation are:

- ***Linked scheme***: Index block here is a disk block which can be read and written to, directly. Several index blocks are linked together for large files.
- ***Multilevel Index***: Here first level index blocks point to a collection of second level index blocks. As the file size increases the level of indexing increases.

Indexed Allocation

- ***Combined scheme***: 15 pointers are made use of, with 12 pointers pointing directly to blocks with the last three being indirect blocks with 13th pointing to single indirect block, 14th pointing to double indirect block and 15th pointing to triple indirect block.



Free space Management

- When files are deleted, free space should be kept track of with the help of free space list which can be implemented in the following ways.

Bit Vector: Bit vector or bitmap is used to keep track of free space. This approach is pretty simple and efficient in finding the ***first occurrence of free block or free space in n consecutive free blocks***. 1 is allotted if the block is free, else zero is assigned.

Linked List

- In this approach, ***all the free disk blocks are linked together such that each free block has the pointer to the next free block*** with the pointer to the first occurrence in a special pointer.

Free space Management

- ***Grouping*** :The addresses of ***n free blocks are stored in the first free block. N-1 blocks are free and*** the nth block contains the addresses of next set of n blocks.
- ***Counting*** :When the free space is contiguous rather than storing the address of all the free space, the first free block and the number of free spaces n that follow are kept track. The free space list stores the address of the first block and the number of free spaces that follow making the implementation smaller and easier.

Efficiency and Performance

- The efficiency of disk space relies on the usage of the disk allocation and directory algorithms.
- While dealing with inodes, file system performance can be increased by spreading them across the volume.
- Efficiency in terms of pointers is based on the space required by the pointer (16/32/64 bits). The size of the pointer limits the size of the file.
- Local memory is included in disk controllers to form an onboard cache, stores entire tracks at a time.
- When a seek is requested, the heads are relocated into places.
- An entire track is read, starting from whatever sector is currently under the heads.

Recovery

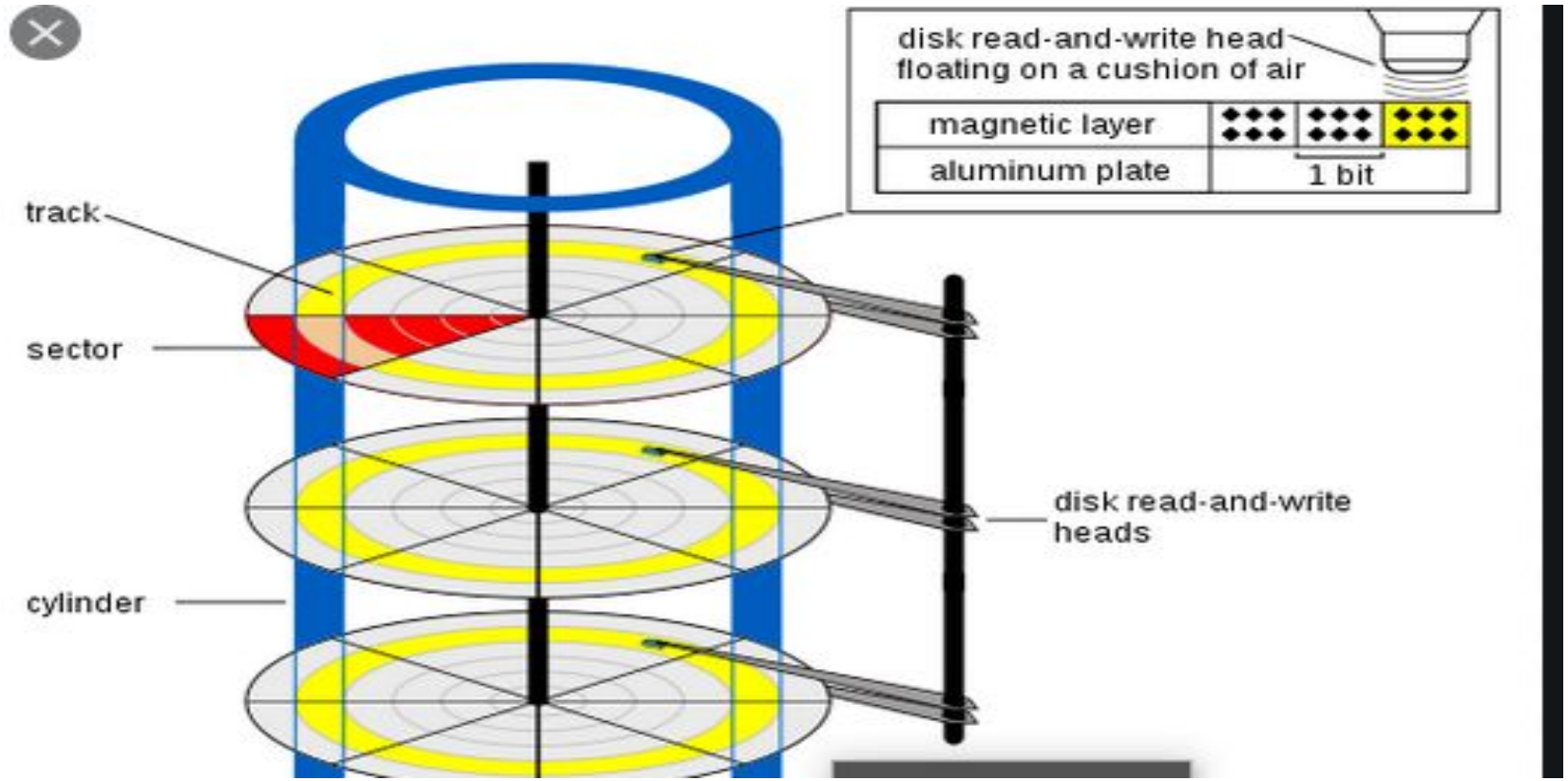
Consistency Checking:

- If there are two copies of directory information, then the version in main memory is up to date. When the computer crashes the buffer and cache contents are lost leaving the file in an inconsistent state, Consistency checker program checks for and fix inconsistencies. In this case, linked allocation is used then with the help of the link to next block, the entire structure can be recovered.

Backup

- Backup of data to a secondary storage device ensures that the data is not lost even if an entire system crashes. ***Incremental backup*** is taking backup of entities which are not already backed up.

Disk Management



Disk Management

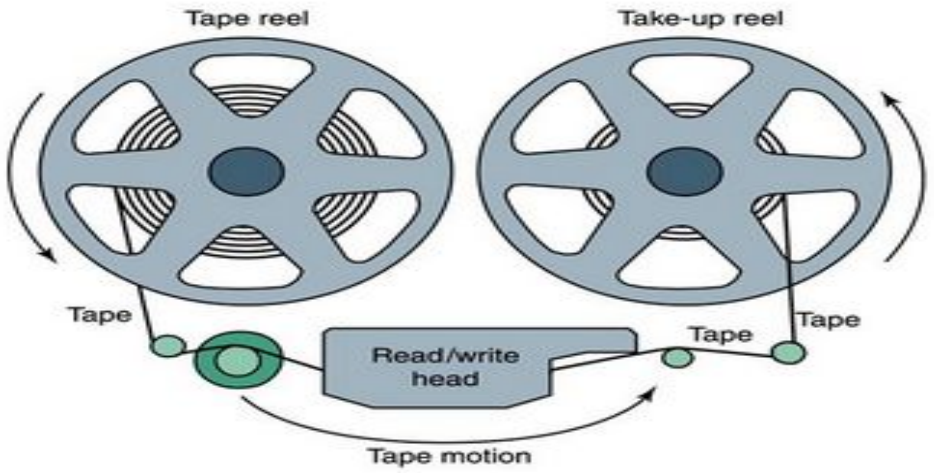
Overview of Mass Storage Structure :

Apart from main memory, the following secondary and tertiary storage devices are used:

- ***Magnetic Disk**: Magnetic disks provide for the secondary storage of the system commonly with the diameter ranging between 1.8 to 5.25 inches. Platters in the magnetic disk hold the information with a read/write head above the surface of the platter and disk arms for moving the heads simultaneously. The surface of the platter is divided into tracks which are further divided into sectors. The cylinder is the set of arms with the same arm position.*
- ***Transfer rate**: Speed of data flow between computer and drive*
- ***Random access time/Seek time**: Time taken to shift the disk arm to the desired cylinder*
- ***Rotational latency:** Time taken for the target sector to reach the disk head.*

Disk Management

- ***Magnetic Tapes***: Magnetic tapes are another secondary storage medium which can hold quite a large amount of data and is a permanent type of storage. However, the speed of access in magnetic tape is 1/100th of access speed using magnetic disks. The tape is usually wound or rewound to reach the target data, and once the head is set, the speed of access is equal to that of magnetic disks .



A magnetic tape storage mechanism

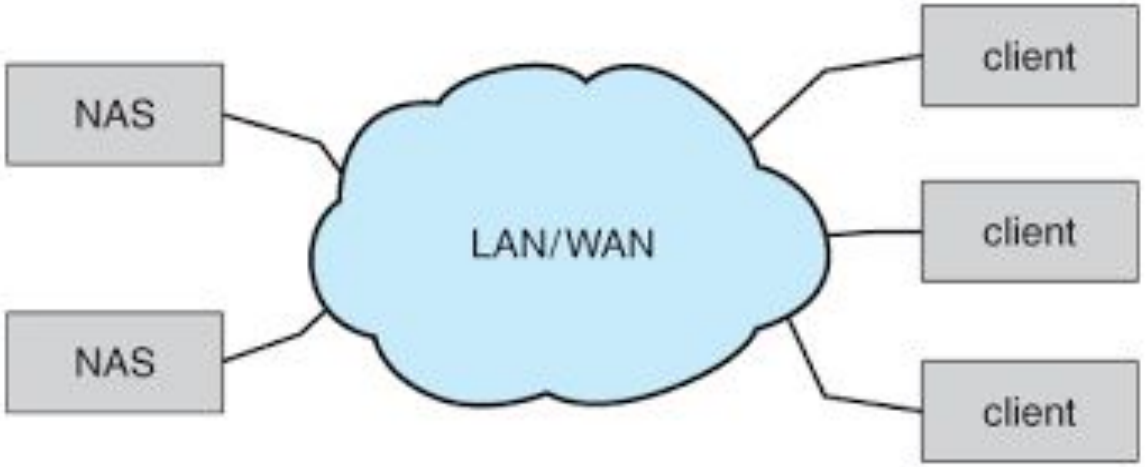
Disk Management

- Disks are divided into ***logical blocks*** and the disk is viewed and addressed as a large ***one dimensional array of blocks***.
- Block is mostly 512 bytes or in rare cases 1024 bytes. Sector 0 is the first track which is located in the outermost cylinder and the order proceeds that way.
- Logical blocks can be converted to disk address (with cylinder number, track number, sector number to reach the sector) on need.

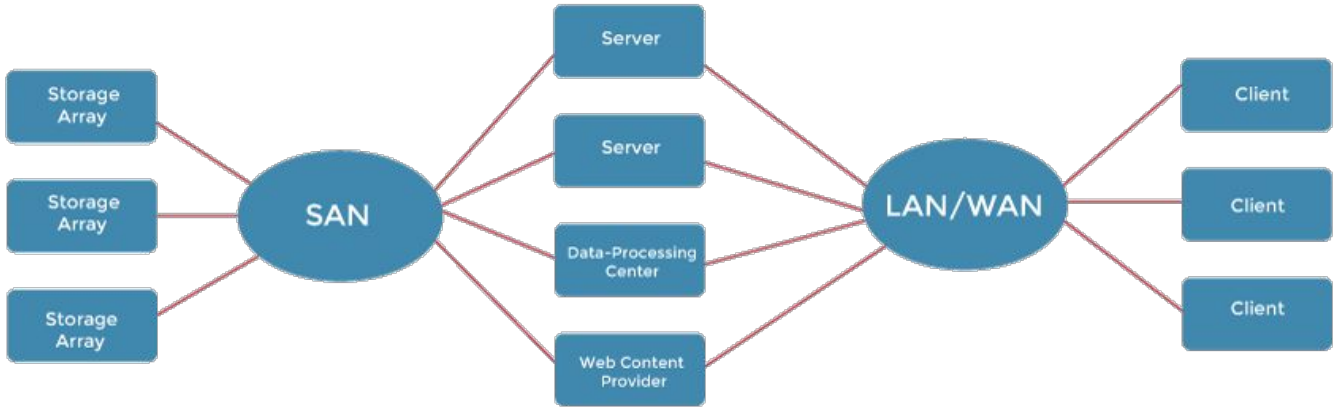
Disk Management

Disk Attachment :*Disk storage can be accessed in two ways, either by I/O ports or through a remote host in a distributed file system.*

- ***Host attached storage:*** *Disk storage is accessed via local I/O ports. I/O bus architecture is made use by a typical desktop named IDE or ATA.*
- **Network attached storage (NAS)** *is a special purpose storage system accessed by remote host storage over a data connection. Network attached storage is accessed by means of remote procedure calls which maybe TCP or UDP over an IP network. The RAID array is used to implement the attached storage unit.*



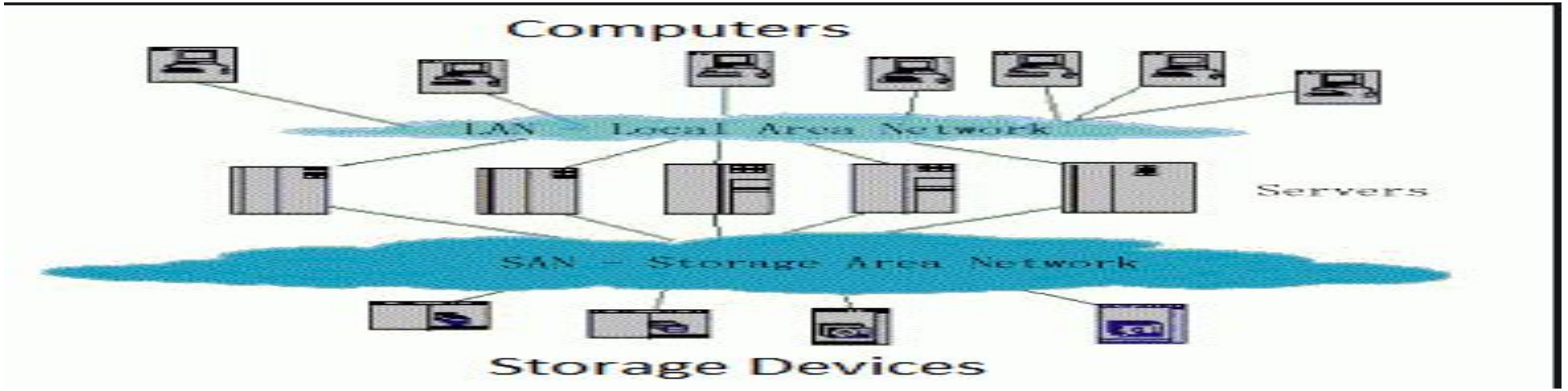
Network Attached Storage



Storage - Area Network

Disk Management

- ***Storage area network :***Bandwidth on the data network is required for network attached storage which in turn increases the latency of the network. Storage area network is a network private to the storage units and servers.



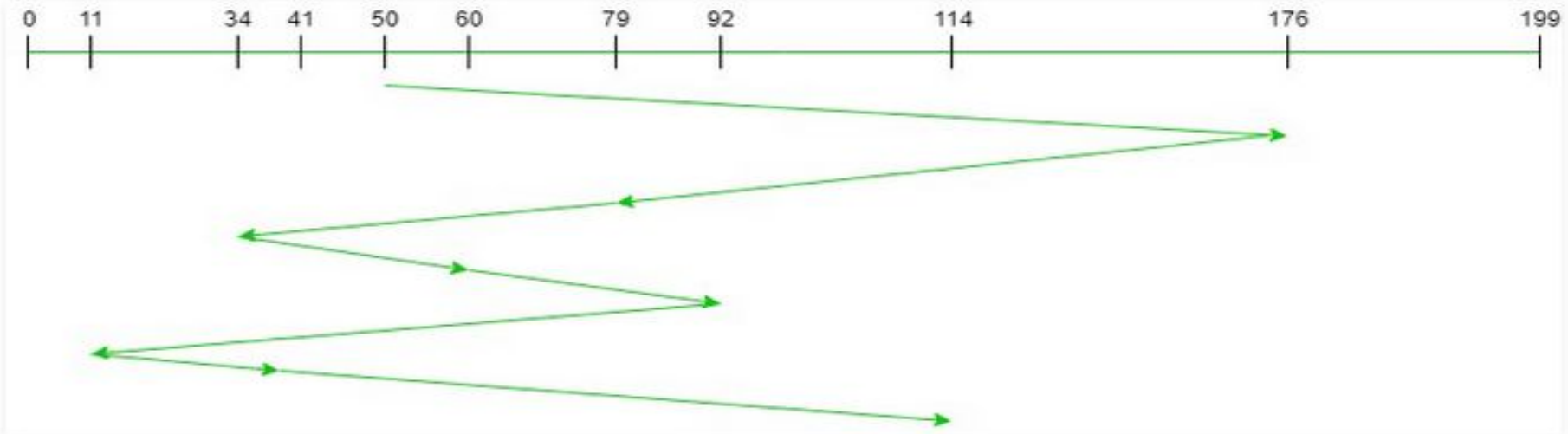
Disk Scheduling

- Disk scheduling uses an algorithm which works on reducing the ***seek time*** where seek time is the time required by the disk arm to make the cylinder head reach the target sector.
- The additional time required to rotate the disk till the target sector is reached is known as the ***rotational latency***.
- ***Bandwidth*** is the rate of transfer of bytes.

Disk Scheduling :FCFS

- The simplest form of scheduling which follows the first come first serve rule.

```
Input:
Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position = 50
```



Disk Scheduling :FCFS

Therefore, the total seek count is calculated as:

```
= (176-50)+(176-79)+(79-34)+(60-34)+(92-60)+(92-11)+(41-11)+(114-41)
= 510
```

SSTF (Shortest Seek Time First)Scheduling

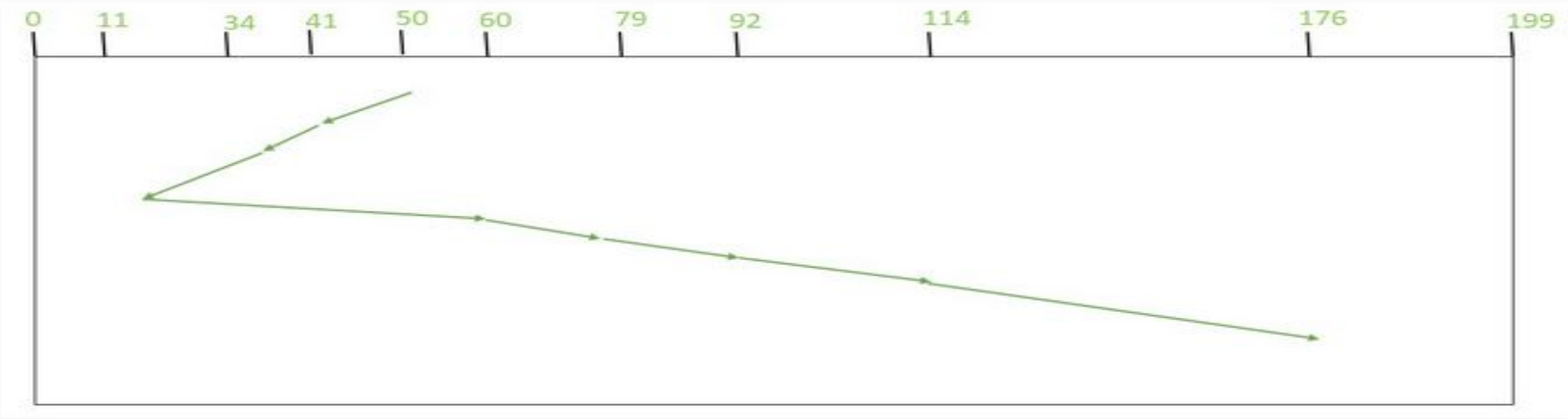
- Shortest seek time first technique selects or processes the requests with least seek time with respect to the current head position.

Example –

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}

Initial head position = 50

The following chart shows the sequence in which requested tracks are serviced using SSTF.



SSTF (Shortest Seek Time First)Scheduling

Therefore, total seek count is calculates as:

```
= (50-41)+(41-34)+(34-11)+(60-11)+(79-60)+(92-79)+(114-92)+(176-114)
= 204
```

SCAN Scheduling

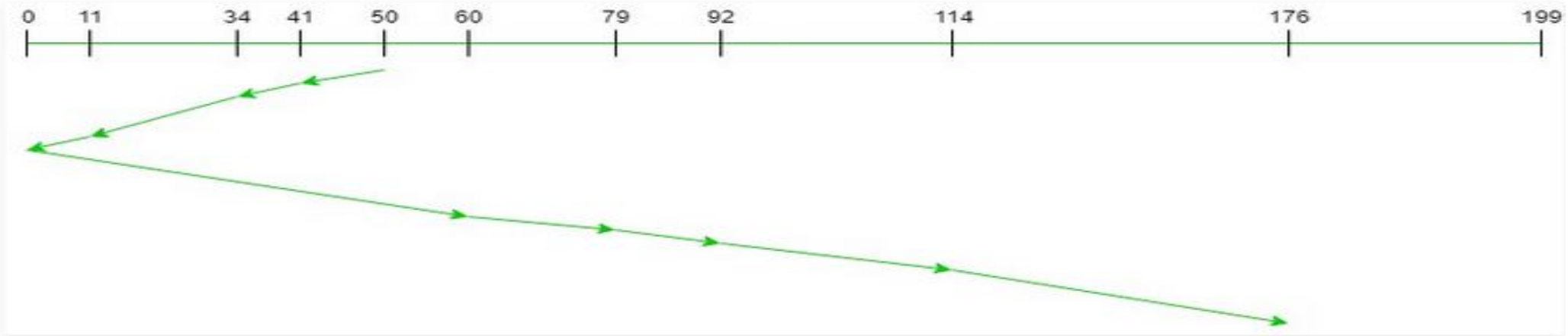
- In scan scheduling, disk arm scans the cylinder from one end to another servicing the requests in the direction and the rest of requests are serviced during the reverse scan in the opposite direction.

Input:

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}

Initial head position = 50

Direction = left (We are moving from right to left)



SCAN Scheduling

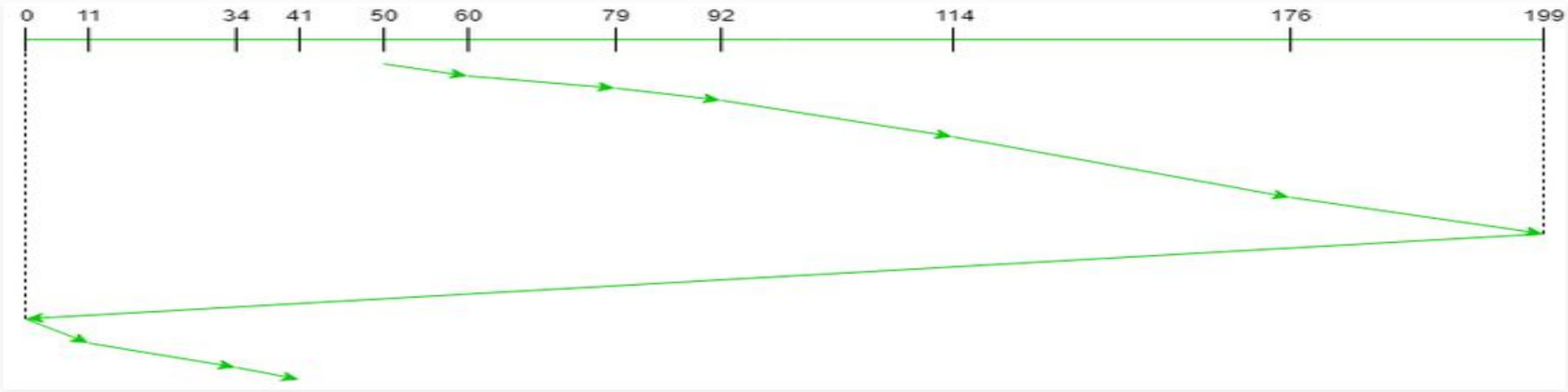
Therefore, the total seek count is calculated as:

```
= (50-41)+(41-34)+(34-11)
 +(11-0)+(60-0)+(79-60)
 +(92-79)+(114-92)+(176-114)
= 226
```

C-Scan Scheduling:

- C-Scan is similar to Scan scheduling, but it does not service requests during reverse scan and cylinders are treated as circular list redirected to the starting of the cylinder once it reaches the end rather than reverse scanning.

```
Input:
Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position = 50
```



C-Scan Scheduling:

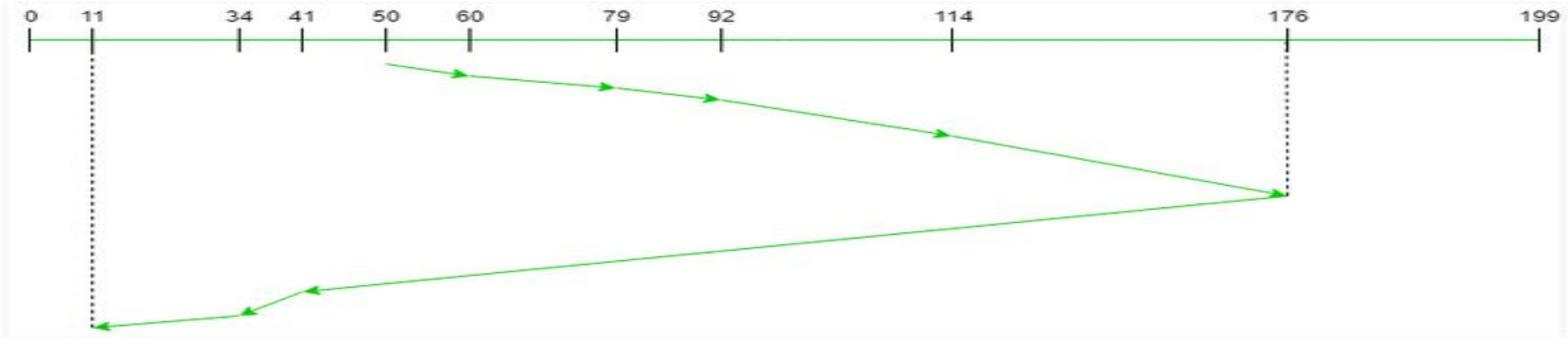
Therefore, the total seek count is calculated as:

```
= (60-50)+(79-60)+(92-79)
        +(114-92)+(176-114)+(199-176)+(199-0)
        +(11-0)+(34-11)+(41-34)
```

LOOK Scheduling:

- In look scheduling once the last request in the queue is serviced the end is not reached, the direction is reversed once the last request in queue is requested.

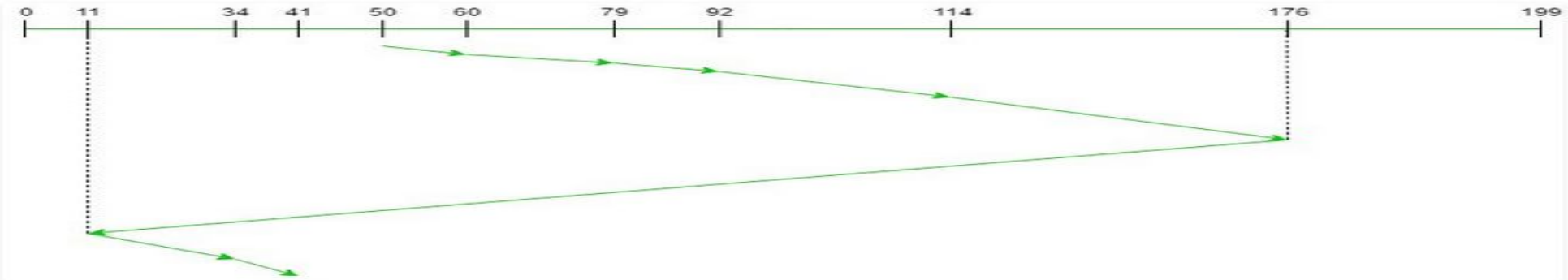
```
Input:
Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position = 50
Direction = right (We are moving from left to right)
```



C-Look Scheduling

> *Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}*
>
> *Initial head position = 50*
>
> *Direction = right (Moving from left to right)*

The head services requests only in one direction(either left or right) until all the requests in this direction are not serviced and then jumps back to the farthest request on the other direction and service the remaining requests .



Therefore, the total seek count = (60 − 50) + (79 − 60) + (92 − 79) + (114 − 92) + (176 − 114) + (176 − 11) + (34 − 11) + (41 − 34) = 321

Disk Management

- An empty magnetic disk is blank and must be formatted into sectors readable and writable by the disk controller via a process called low-level formatting.
- ***Low-level formatting*** fills the disk with a data structure (containing the header, data and trailer). Low-level formatting is mostly done during manufacture itself.
- ***ECC (error correcting code)*** is included in the header and trailer and is computed during I/O and stored. Whenever a read request comes in, ECC is calculated and compared with the stored value. If they do not match, corruption has occurred.

For a disk to hold files,

- First step: Partition the disk as cylinder(s) (one or more cylinders can be obtained after partition)
- Second step: Logical formatting where OS stores initial file system and data structures onto the disk

Swap-space Management

- Swap space management is a low-level OS task providing virtual memory.
- ***Swap space can hold any data or code or an entire process image or pages which are forced to be temporarily pushed out of memory***.
- The amount of swap space required varies from system to system based on the amount of virtual and physical memory.

Location of swap space may be in the

- ***File system***: Here, the swap space is considered as one large file and normal procedure is used to create, name or allocate space via the file system routines.
- ***Disk partition :***separately dedicated to swap space managed by a swap space storage manager, optimizing the speed with which the swap space is accessed

Goals of 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* – to ensure that each object is accessed correctly and only by those processes that are allowed to do so.

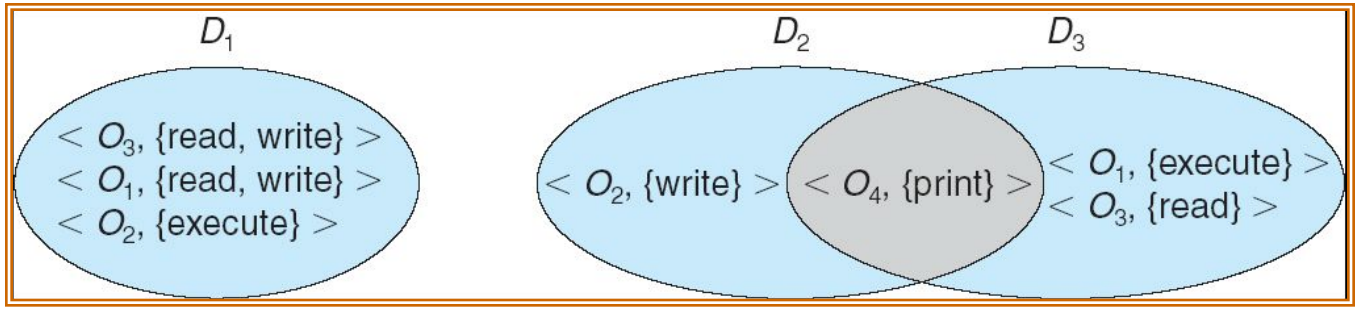
Guiding Principles of Protection

- *Principle of least privilege*
  - Programs, users and systems should be given just enough privileges to perform their tasks

- Separate *policy* from *mechanism*
  - *Mechanism:* the stuff built into the OS to make protection work
  - *Policy:* the data that says who can do what to whom

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



Conceptual Representation – Access Matrix

- View protection as a matrix (*access matrix*)
- Rows represent domains
- Columns represent objects
- *Access(i, j)* is set of operations that process executing in $Domain_i$ can invoke on $Object_j$

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	

- Columns are *access control lists* (ACLs)
  - Associated with each object
- Rows are *capabilities*
  - Associated with each user, group, or domain

Unix/Linux Matrix

	file1	*file 2*	*file 3*	*device*	*domain*
User/Domain 1	*r*	*rx*	*rwx*	–	*enter*
User/Domain 2	*r*	*x*	*rx*	*rwx*	–
User/Domain 3	*rw*	–	–	–	–
...					

- Columns are *access control lists* (ACLs)
  - Associated with each object
- Rows are *capabilities*
  - Associated with each user or each domain

- Domains as objects added to Access Matrix

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			

Revocation of Access Rights

- Role-Based Access Control, RBAC, assigns privileges to users, programs, or roles as appropriate, where "privileges" refer to the right to call certain system calls, or to use certain parameters with those calls.
- In dynamic protection system, we may need to revoke access rights to objects shared by different users, this revocation of 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?
- With an access list scheme revocation is easy, immediate, and can be selective, general, partial, total, temporary, or permanent, as desired.

Revocation of Access Rights

With capabilities lists the problem is more complicated, because access rights are distributed throughout the system. A few schemes that have been developed include:

- Reacquisition - Capabilities are periodically revoked from each domain, which must then re-acquire them.
- Back-pointers - A list of pointers is maintained from each object to each capability which is held for that object.
- Indirection - Capabilities point to an entry in a global table rather than to the object. Access rights can be revoked by changing or invalidating the table entry, which may affect multiple processes, which must then re-acquire access rights to continue.
- Keys - A unique bit pattern is associated with each capability when created, which can be neither inspected nor modified by the process. A master key is associated with each object. When a capability is created, its key is set to the object's master key. As long as the capability's key matches the object's key, then the capabilities remain valid. The object master key can be changed with the set-key command, thereby invalidating all current capabilities. More flexibility can be added to this scheme by implementing a list of keys for each object, possibly in a global table.