Overview of Mass Storage Structure

Magnetic disks provide bulk of secondary storage of modern computers

Drives rotate at 60 to 250 times per second

Transfer rate is rate at which data flow between drive and computer

Positioning time (random-access time) is time to move disk arm to desired cylinder (seek time) and time for desired sector to rotate under the disk head (rotational latency)

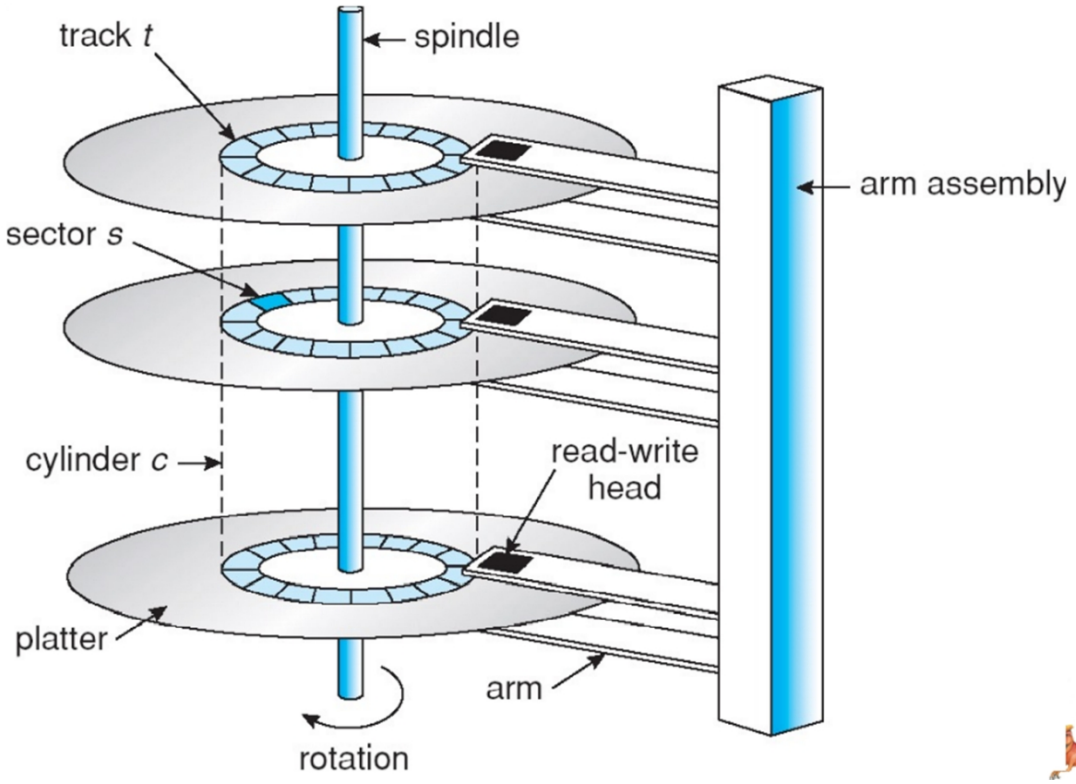
Head crash results from disk head making contact with the disk surface -- That's bad

Disks can be removable

Drive attached to computer via I/O bus

Busses vary, including EIDE, ATA, SATA, USB, Fibre Channel, SCSI, SAS, Firewire

Host controller in computer uses bus to talk to disk controller built into drive or storage array



Hard Disks

Platters range from .85” to 14” (historically)

Commonly 3.5”, 2.5”, and 1.8”

Range from 30GB to 3TB per drive

Performance

Transfer Rate – theoretical – 6 Gb/sec

Effective Transfer Rate – real – 1Gb/sec

Seek time from 3ms to 12ms – 9ms common for desktop drives

Average seek time measured or calculated based on 1/3 of tracks

Latency based on spindle speed

1 / (RPM / 60) = 60 / RPM

Average latency = ½ latency

Magnetic Tape

Was early secondary-storage medium

Evolved from open spools to cartridges

Relatively permanent and holds large quantities of data

Access time slow

Random access ~1000 times slower than disk

Mainly used for backup, storage of infrequently-used data, transfer medium between systems

Kept in spool and wound or rewound past read-write head

Once data under head, transfer rates comparable to disk

140MB/sec and greater

200GB to 1.5TB typical storage

Common technologies are LTO-{3,4,5} and T10000

Solid-State Disks

Nonvolatile memory used like a hard drive

Many technology variations

Can be more reliable than HDDs

More expensive per MB

Maybe have shorter life span

Less capacity

But much faster

Busses can be too slow -> connect directly to PCI for example

No moving parts, so no seek time or rotational latency

On Disk Data Structures

There are various on disk data structures that are used to implement a file system. This structure may vary depending upon the operating system.

Boot Control Block

Boot Control Block contains all the information which is needed to boot an operating system from that volume. It is called boot block in UNIX file system. In NTFS, it is called the partition boot sector.

Volume Control Block

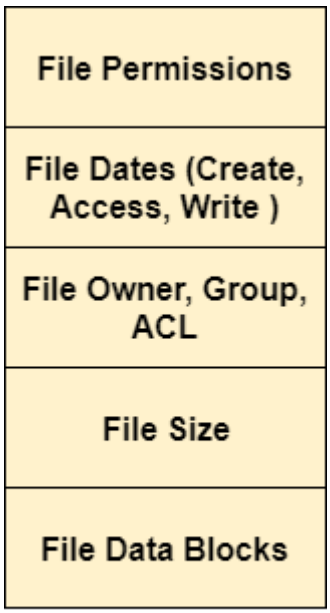
Volume control block all the information regarding that volume such as number of blocks, size of each block, partition table, pointers to free blocks and free FCB blocks. In UNIX file system, it is known as super block. In NTFS, this information is stored inside master file table.

Directory Structure (per file system)

A directory structure (per file system) contains file names and pointers to corresponding FCBs. In UNIX, it includes inode numbers associated to file names.

File Control Block

File Control block contains all the details about the file such as ownership details, permission details, file size,etc. In UFS, this detail is stored in inode. In NTFS, this information is stored inside master file table as a relational database structure. A typical file control block is shown in the image below.



File Control Block

Disk Attachment in Operating System

There are two ways for computers to access disc storage. One method is to use I/O ports, also known as host-attached storage, on small systems. Another method is through a remote host in a distributed file system, which is known as network-attached storage.

ost-Attached Storage

Host-attached storage (HAS) is storage accessed via local I/O ports. These ports make use of various technologies. IDE or ATA is the I/O bus architecture used by most desktop PCs. This architecture allows for up to two drivers per I/O bus. SATA is a new related standard that has simplified cabling. High-end

workstations and servers typically use more advanced I/O architectures such as Small Computer System Interface (SCSI) and fibre channel (FC).

SCSI (Small Computer System Interface) is bus architecture. Its physical medium is typically a ribbon wire with many conductors. A maximum of 16 devices may be attached to the bus using the SCSI protocol. The devices typically comprise of one controller card (SCSI initiator) on the host and up to 15 storage devices (SCSI targets). A SCSI disk is a typical SCSI target. Although, the protocol permits every SCSI target to address up to 8 logical units. Logical unit addressing is widely used to direct commands to the RATD array or portable media library components.

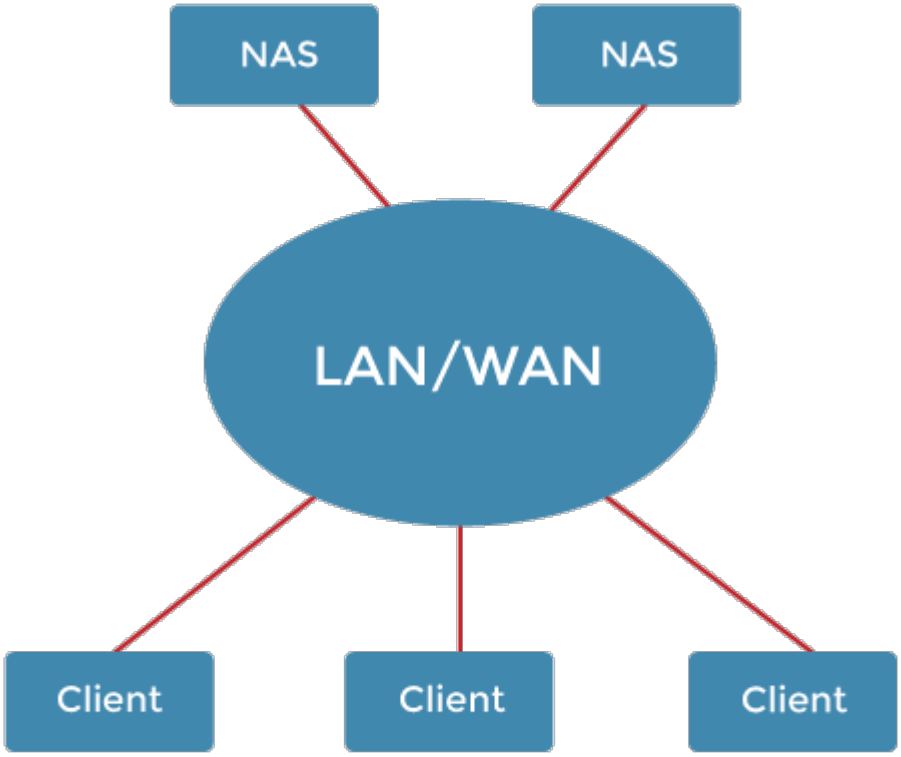
A Fibre Channel is a high-speed serial architecture that may use optical fibre or a four-conductor copper wire. It comes in two varieties. One type of fabric is a big switched fabric with a 24-bit address space. This variant is projected to take the lead in the future and will serve as the foundation for storage-area networks (SANs). Many hosts and storage devices may be connected to the fibre due to the large address space and switched nature of the communication that provides better flexibility in I/O communication. Another PC version is an arbitrated loop (FC-AL), which may address up to 126 devices, including drives and controllers.

As host-attached storage, a wide range of storage devices are suitable. Hard disc devices, RAID arrays, CDs, DVDs, and tape drives are among them. The I/O commands that start data transfer to a host-attached storage device can read and write logical data blocks routed to specially designated storage units.

Network-Attached Storage

A network-attached storage (NAS) device is a dedicated storage system that can be accessed via the data network. Clients access NAS through a remote procedure call interface, like a CIFS for Windows computers or a network file system for Unix. Remote procedure calls (RPCs) are transmitted through TCP or UDP via an IP network, which is often the same LAN that delivers all data traffic to the clients. Generally, the network-attached storage unit is constructed as a

RAID array with software that supports the RPC interface. It is easy to consider NAS as just another storage-access protocol. For example, a NAS system may access storage through RPC via TCP/IP rather than a SCSI device driver and SCSI protocols.



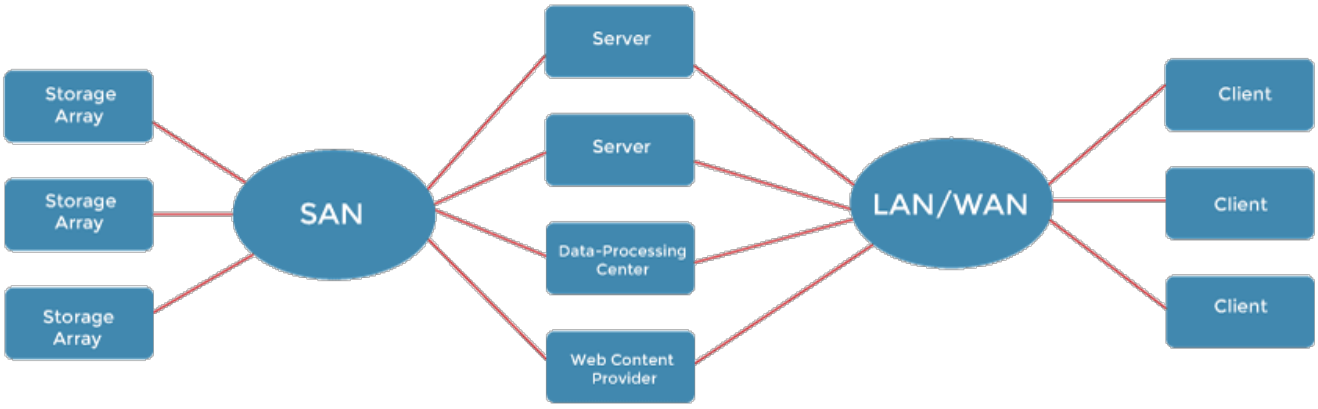
Network - Attached Storage

NAS allows all systems on a local area network to share a pool of storage with the same simplicity of naming and access as local host-attached storage provides. However, it is less efficient and performs worse than other direct-attached storage solutions.

ISCSI (Internet Small Computer Systems Interface) is the latest network-attached storage protocol. The SCSI protocol is carried across the IP network protocol. As a result, the SCSI protocol uses the networks to interconnect hosts and their storage than SCSI cables.

Storage-Area Network

The one downside of network-attached storage systems is bandwidth usage on the data network by storage I/O activities, which increases network latency. This problem is most acute in large client-server configurations because server-to-client communications compete for bandwidth with server-to-storage communications.



Storage - Area Network

SAN is a private network that uses storage protocols to connect servers and storage devices instead of networking standards. The flexibility of a SAN is its strength. Multiple hosts and storage arrays may connect to the same SAN, and storage may be allocated dynamically to hosts.

A SAN switch permits or prohibits access between hosts and storage. For example, if a host's disk space is running short, the SAN may be set to allocate more storage to that server. SANs enable server clusters to share the same storage and storage arrays to contain several direct host connections. SANs often have more ports at a lower cost than storage arrays. The most common is FC. InfiniBand is a new option for high-speed connectivity networks for servers and storage units. It is a special-purpose bus architecture that provides hardware and software support for servers and storage units high-speed connectivity networks.

Disk Scheduling

a process needs two type of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk.

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

some important terms related to disk scheduling

- **Seek Time**
- Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.
- **Rotational Latency**
- It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.
- **Transfer Time**
- It is the time taken to transfer the data.
-

Disk Access Time

Disk access time is given as,

Disk Access Time = Rotational Latency + Seek Time + Transfer Time

Disk Response Time

Purpose of Disk Scheduling

The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed

Disk Scheduling Algorithms

FCFS scheduling algorithm

SSTF (shortest seek time first) algorithm

SCAN scheduling

C-SCAN scheduling

LOOK Scheduling

C-LOOK scheduling

Disk Management in Operating System

The operating system is responsible for various operations of disk management. Modern operating systems are constantly growing their range of services and add-ons, and all operating systems implement four essential operating system administration functions. These functions are as follows:

Process Management

Memory Management

File and Disk Management

I/O System Management

Most systems include secondary storage devices (magnetic disks). It is a low-cost, non-volatile storage method for data and programs. The user data and programs are stored on different storage devices known as files. The OS is responsible for allocating space to files on secondary storage devices as required.

It doesn't ensure that files are saved on physical disk drives in contiguous locations. It is highly dependent on the provided space. New files are mostly stored in various locations if the disk drive is full. On the other hand, the OS example file hides that the file is divided into several parts.

The OS requires tracking the position of the disk drive for each section of every file on the disk. It may include tracking many files and file segments on a physical

disk drive in some circumstances. Furthermore, the OS must be able to identify each file and conduct read and writes operations on it according to the requirements. As a result, the OS is mainly responsible for setting the file system, assuring the security and reliability of reading and writing activities to secondary storage, and keeping access times consistent.

Disk Management of the OS includes the various aspects, such as:

1. Disk Formatting

A new magnetic disk is mainly a blank slate. It is platters of the magnetic recording material. Before a disk may hold data, it must be partitioned into sectors that may be read and written by the disk controller. It is known as physical formatting and low-level formatting.

Low-level formatting creates a unique data structure for every sector on the drive. A data structure for a sector is made up of a header, a data region, and a trailer. The disk controller uses the header and trailer to store information like an error-correcting code (ECC) and a sector number.

The OS must require recording its own data structures on the disk drive to utilize it as a storage medium for files. It accomplishes this in two phases. The initial step is to divide the disk drive into one or more cylinder groups. The OS may treat every partition as it were a separate disk. For example, one partition could contain a copy of the OS executable code, while another could contain user files. The second stage after partitioning is logical formatting. The operating store stores the initial file system data structure on the disk drive in this second stage.

2. Boot Block

When a system is turned on or restarted, it must execute an initial program. The start program of the system is called the bootstrap program. It starts the OS after initializing all components of the system. The bootstrap program works by looking for the OS kernel on disk, loading it into memory, and jumping to an initial address to start the OS execution.

The bootstrap is usually kept in read-only memory on most computer systems. It is useful since read-only memory does not require initialization and is at a fixed location where the CPU may begin executing whether powered on or reset. Furthermore, it may not be affected by a computer system virus because ROM is read-only. The issue is that updating this bootstrap code needs replacing the ROM hardware chips.

As a result, most computer systems include small bootstrap loader software in the boot ROM, whose primary function is to load a full bootstrap program from a disk drive. The entire bootstrap program can be modified easily, and the disk is rewritten with a fresh version. The bootstrap program is stored in a partition and is referred to as the boot block. A boot disk or system disk is a type of disk that contains a boot partition.

3. Bad Blocks

Disks are prone to failure due to their moving parts and tight tolerances. When a disk drive fails, it must be replaced and the contents transferred to the replacement disk using backup media. For some time, one or more sectors become faulty. Most disks also come from the company with bad blocks. These blocks are handled in various ways, depending on the use of disk and controller.

On the disk, the controller keeps a list of bad blocks. The list is initialized during the factory's low-level format and updated during the disk's life. Each bad sector may be replaced with one of the spare sectors by directing the controller. This process is referred to as sector sparing.

Swap-Space Management in Operating System

A computer has a sufficient amount of physical memory, but we need more, so we swap some memory on disk most of the time. Swap space is a space on a hard disk that is a substitute for physical memory. It is used as virtual memory, which contains process memory images. Whenever our computer runs short of physical memory, it uses its virtual memory and stores information in memory on a disk.

This interchange of data between virtual memory and real memory is called swapping and space on disk as swap space. Swap space helps the computer's operating system pretend that it has more RAM than it actually has. It is also called a swap file.

Virtual memory is a combination of RAM and disk space that running processes can use. Swap space is the portion of virtual memory on the hard disk, used when RAM is full. Swap space can be useful to computers in the following various ways, such as:

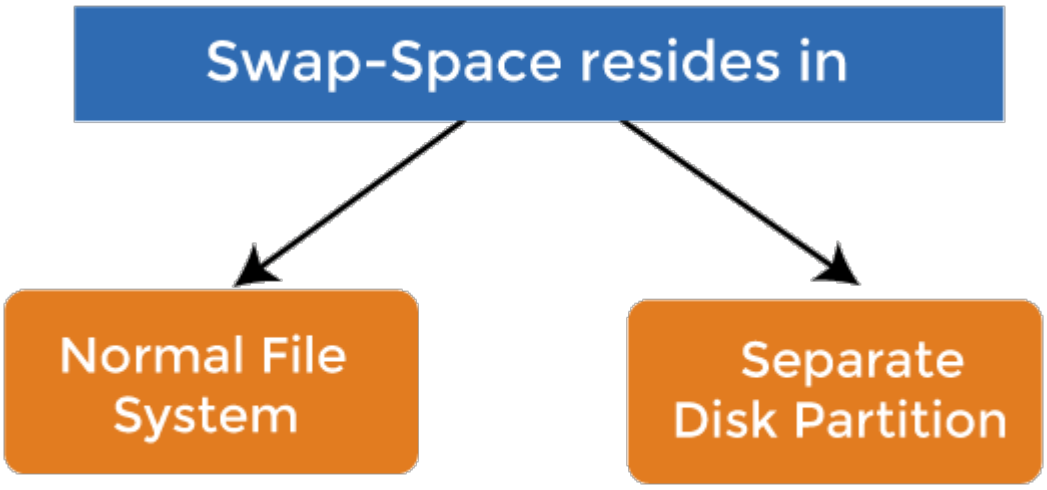
It can be used as a single contiguous memory which reduces I/O operations to read or write a file.

Applications that are not used or less used can be kept in a swap file.

Having sufficient swap files helps the system keep some physical memory free all the time.

The space in physical memory that has been freed due to swap space can be used by OS for other important tasks.

Operating systems such as Windows, Linux, etc. systems provide a certain amount of swap space by default which users can change according to their needs. If you don't want to use virtual memory, you can easily disable it together. Still, if you run out of memory, then the kernel will kill some of the processes to create a sufficient amount of space in physical memory so that it totally depends upon the user whether he wants to use swap space or not.



If the swap space is simply a large file within the file system, normal file-system routines can be used to create, name, and allocate its space. This approach, though easy to implement, is inefficient. Navigating the directory structure and a disk-allocation data structure takes extra disk access.

External fragmentation can greatly increase swapping times by forcing multiple seeks during the reading or writing of a process image. We can improve performance by caching the block location information in physical memory and by using special tools to allocate physically contiguous blocks for the swap file. However, the cost of traversing the file-system data structures still remains.

Alternatively, swap space can be created in a separate raw partition, as no file system or directory structure is placed in this space. Rather, a separate swap-space storage manager is used to allocate and deallocate the blocks from the raw partition. This manager uses algorithms optimized for speed rather than storage efficiency because swap space is accessed more frequently than file systems.

Internal fragmentation may increase, but this trade-off is acceptable because the life of data in the swap space generally is much shorter than that of files in the file system. Swap space is reinitialized at boot time, so any fragmentation is short-lived. This approach creates a fixed amount of swap space during disk partitioning. Adding more swap space requires repartitioning the disk or adding another swap space elsewhere.

Some operating systems are flexible and can swap both in raw partitions and file-system space, such as Linux. The policy and implementation are separate, allowing the machine's administrator to decide which type of swapping to use. The trade-off is between the convenience of allocation and management in the file system and swapping performance in raw partitions.

What is RAID?

RAID or redundant array of independent disks is a data storage virtualization technology that combines multiple physical disk drive components into one or more logical units for data redundancy, performance improvement, or both.

It is a way of storing the same data in different places on multiple hard disks or solid-state drives to protect data in the case of a drive failure. A RAID system consists of two or more drives working in parallel. These can be hard discs, but there is a trend to use SSD technology (Solid State Drives).

RAID combines several independent and relatively small disks into single storage of a large size. The disks included in the array are called array members. The disks can combine into the array in different ways, which are known as RAID levels. Each of RAID levels has its own characteristics of:

Fault-tolerance is the ability to survive one or several disk failures.

Performance shows the change in the read and writes speed of the entire array compared to a single disk.

The array's capacity is determined by the amount of user data written to the array. The array capacity depends on the RAID level and does not always match the sum of the RAID member disks' sizes. To calculate the particular RAID type's capacity and a set of member disks, you can use a free online RAID calculator.

RAID systems can use with several interfaces, including SATA, SCSI, IDE, or FC (fiber channel.) Some systems use SATA disks internally but that have a FireWire or SCSI interface for the host system.

Sometimes disks in a storage system are defined as JBOD, which stands for Just a Bunch of Disks. This means that those disks do not use a specific RAID level and acts as stand-alone disks. This is often done for drives that contain swap files or spooling data.

How RAID Works

RAID works by placing data on multiple disks and allowing input/output operations to overlap in a balanced way, improving performance. Because various disks increase the mean time between failures (MTBF), storing data redundantly also increases fault tolerance.

RAID arrays appear to the operating system as a single logical drive. RAID employs the techniques of disk mirroring or disk striping.

Disk Mirroring will copy identical data onto more than one drive.

Disk Striping partitions help spread data over multiple disk drives. Each drive's storage space is divided into units ranging from 512 bytes up to several megabytes. The stripes of all the disks are interleaved and addressed in order.

Disk mirroring and disk striping can also be combined in a RAID array.

In a single-user system where significant records are stored, the stripes are typically set up to be small (512 bytes) so that a single record spans all the disks and can be accessed quickly by reading all the disks at the same time.

In a multi-user system, better performance requires a stripe wide enough to hold the typical or maximum size record, allowing overlapped disk I/O across drives.

Levels of RAID

Many different ways of distributing data have been standardized into various RAID levels. Each RAID level is offering a trade-off of data protection, system

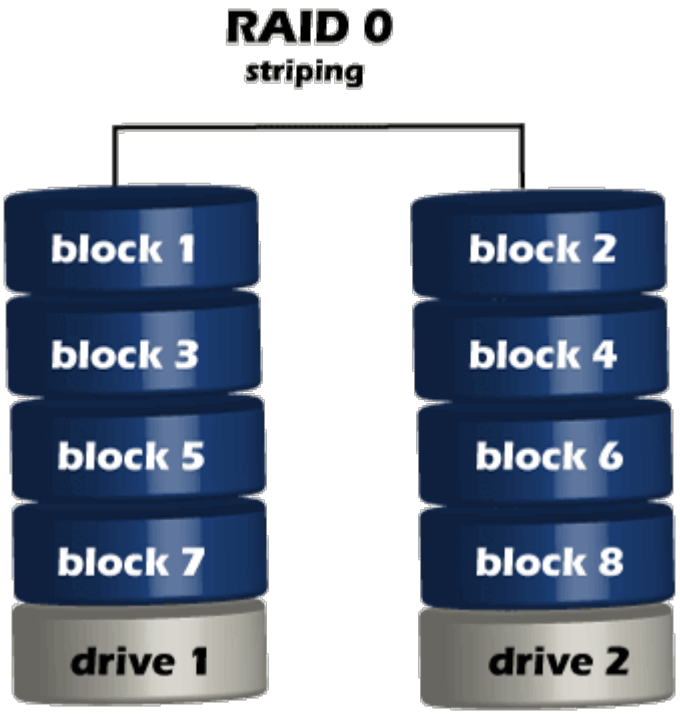
performance, and storage space. The number of levels has been broken into three categories, standard, nested, and non-standard RAID levels.

Standards RAID Levels

Below are the following most popular and standard RAID levels.

1. RAID 0 (striped disks)

RAID 0 is taking any number of disks and merging them into one large volume. It will increase speeds as you're reading and writing from multiple disks at a time. But all data on all disks is lost if any one disk fails. An individual file can then use the speed and capacity of all the drives of the array. The downside to RAID 0, though, is that it is NOT redundant. The loss of any individual disk will cause complete data loss. This RAID type is very much less reliable than having a single disk.

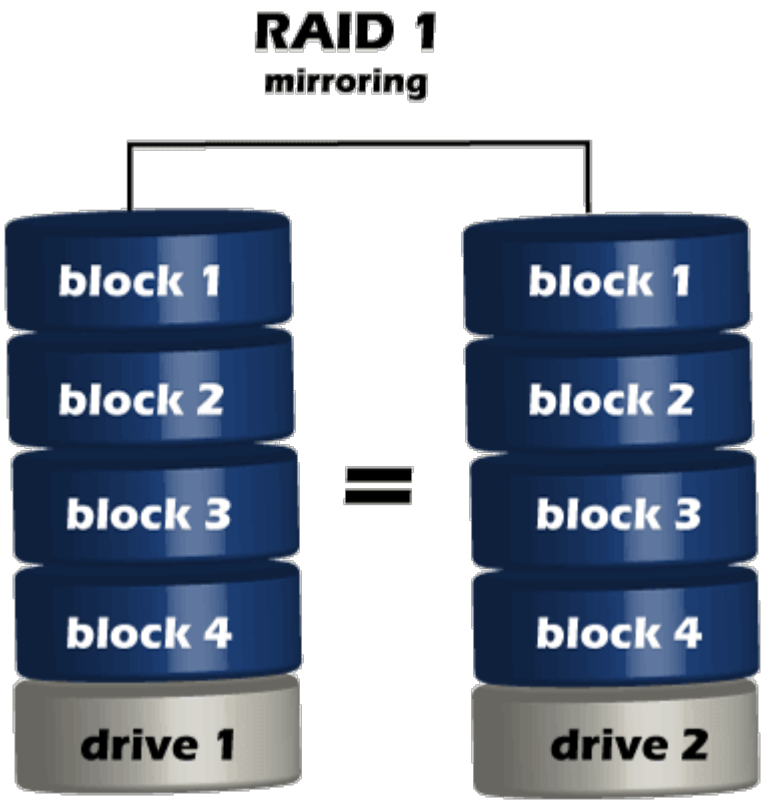


There is rarely a situation where you should use RAID 0 in a server environment. You can use it for cache or other purposes where speed is essential, and reliability or data loss does not matter at all.

2. RAID 1 (mirrored disks)

It duplicates data across two disks in the array, providing full redundancy. Both disks are store exactly the same data, at the same time, and at all times. Data is not lost as long as one disk survives. The total capacity of the array equals the capacity of the smallest disk in the array. At any given instant, the contents of both disks in the array are identical.

RAID 1 is capable of a much more complicated configuration. The point of RAID 1 is primarily for redundancy. If you completely lose a drive, you can still stay up and running off the other drive.

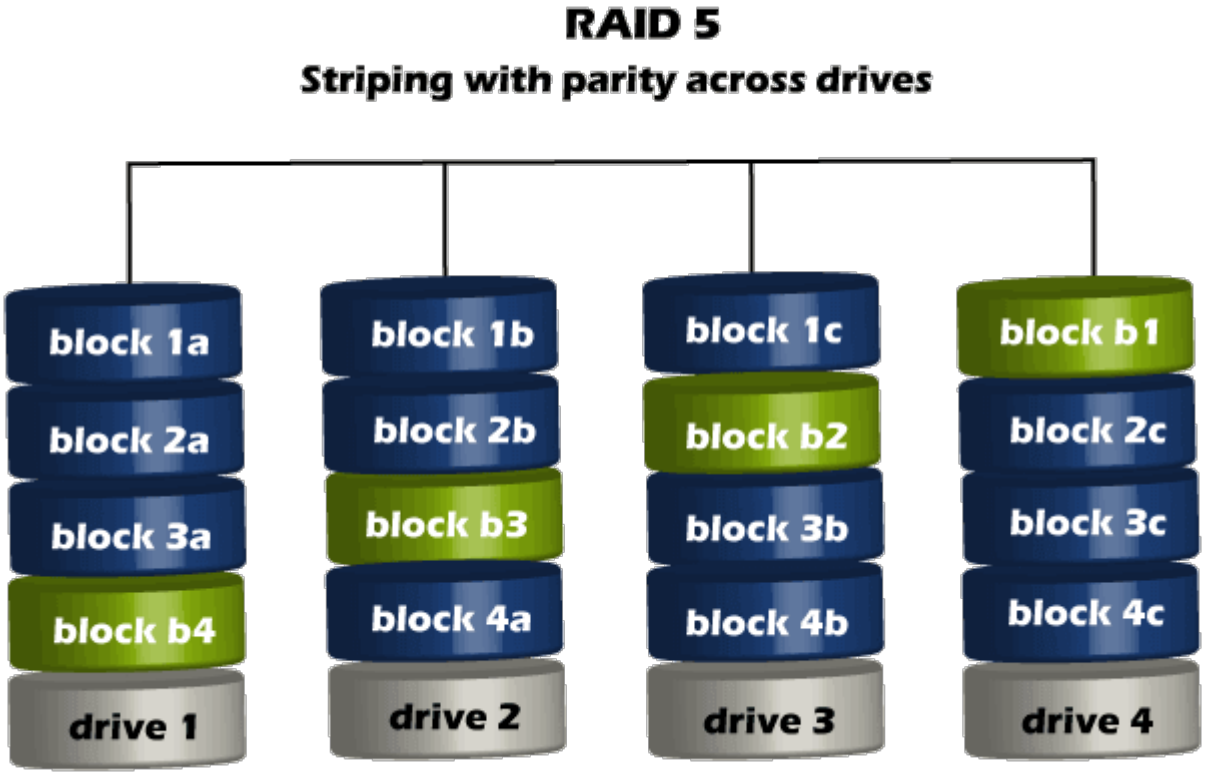


If either drive fails, you can then replace the broken drive with little to no downtime. RAID 1 also gives you the additional benefit of increased read

performance, as data can read off any of the drives in the array. The downsides are that you will have slightly higher write latency. Since the data needs to be written to both drives in the array, you'll only have a single drive's available capacity while needing two drives.

3. RAID 5(striped disks with single parity)

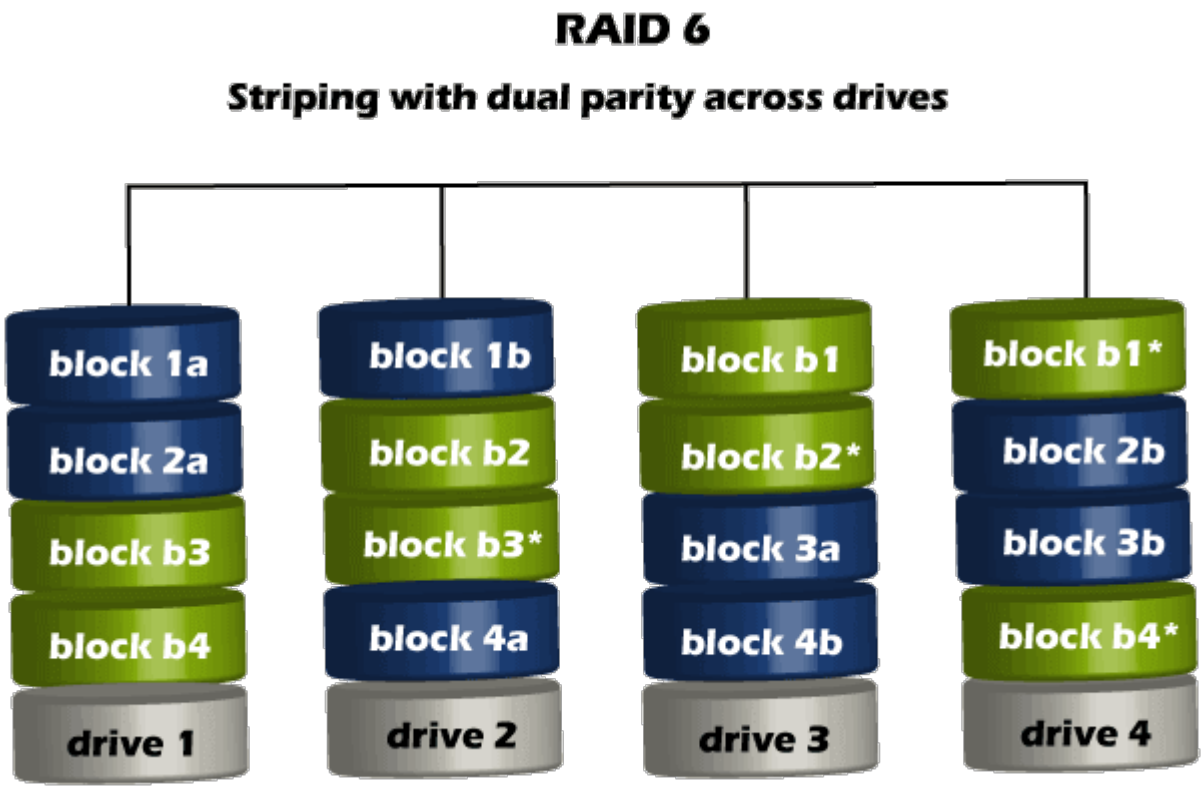
RAID 5 requires the use of at least three drives. It combines these disks to protect data against loss of any one disk; the array's storage capacity is reduced by one disk. It strips data across multiple drives to increase performance. But, it also adds the aspect of redundancy by distributing parity information across the disks.



4. RAID 6 (Striped disks with double parity)

RAID 6 is similar to RAID 5, but the parity data are written to two drives. The use of additional parity enables the array to continue to function even if two disks

fail simultaneously. However, this extra protection comes at a cost. RAID 6 has a slower write performance than RAID 5.



The chances that two drives break down at the same moment are minimal. However, if a drive in a RAID 5 system died and was replaced by a new drive, it takes a lot of time to rebuild the swapped drive. If another drive dies during that time, you still lose all of your data. With RAID 6, the RAID array will even survive that second failure also.

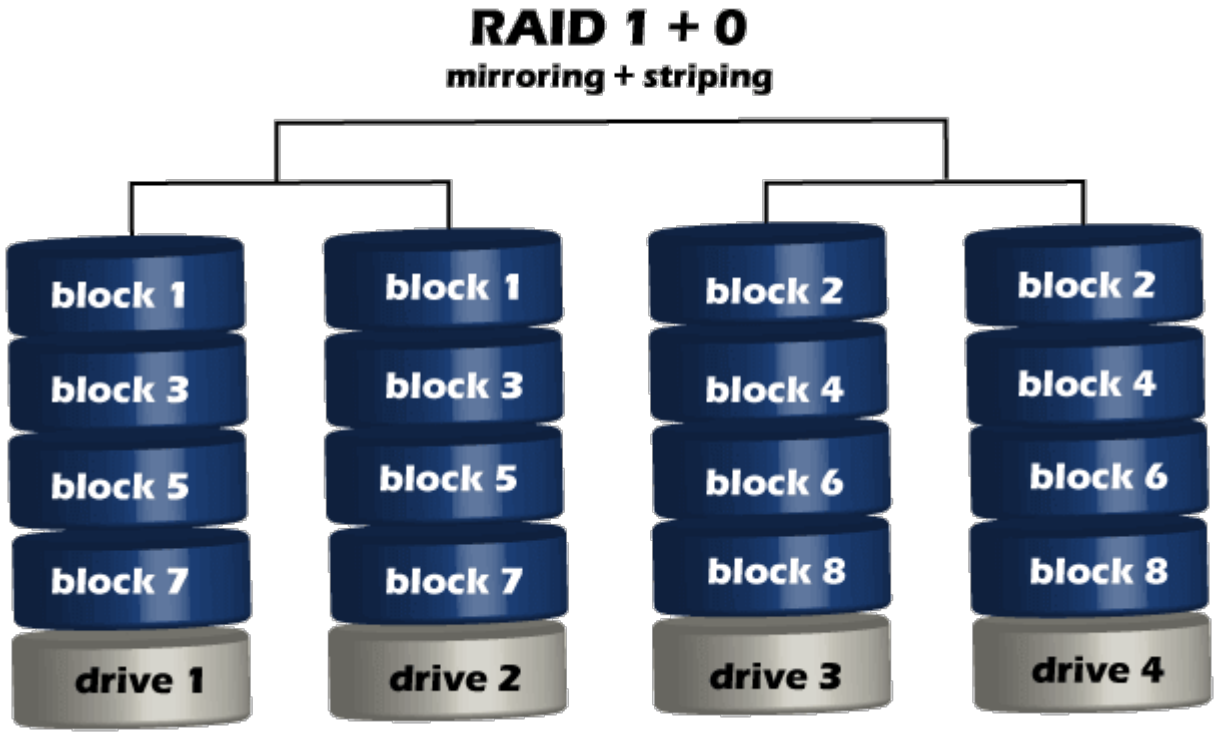
Nested RAID levels

Some RAID levels are referred to as nested RAID because they are based on a combination of RAID levels, such as:

1. RAID 10 (1+0)

This level Combines RAID 1 and RAID 0 in a single system, which offers higher performance than RAID 1, but at a much higher cost.

This is a nested or hybrid RAID configuration. It provides security by mirroring all data on secondary drives while using striping across each set of drives to speed up data transfers.



2. RAID 01 (0+1)

RAID 0+1 is similar to RAID 1+0, except the data organization method is slightly different. Rather than creating a mirror and then striping the mirror, RAID 0+1 creates a stripe set and then mirrors the stripe set.

3. RAID 03 (0+3, also known as RAID 53 or RAID 5+3)

This level uses striping similar to RAID 0 for RAID 3's virtual disk blocks. This offers higher performance than RAID 3 but at a higher cost.

4. RAID 50 (5+0)

This configuration combines RAID 5 distributed parity with RAID 0 striping to improve RAID 5 performance without reducing data protection.

Non-standard RAID levels

Non-standard RAID levels vary from standard RAID levels, and they are usually developed by companies or organizations for mainly proprietary use, such as:

1. RAID 7

A non-standard RAID level is based on RAID 3 and RAID 4 that adds caching. It includes a real-time embedded OS as a controller, caching via a high-speed bus, and other stand-alone computer characteristics.

2. Adaptive RAID

This level enables the RAID controller to decide how to store the parity on disks. It will choose between RAID 3 and RAID 5, depending on which RAID set type will perform better with the kind of data being written to the disks.

3. Linux MD RAID 10

The Linux kernel provides this level. It supports the creation of nested and non-standard RAID arrays. Linux software RAID can also support standard RAID 0, RAID 1, RAID 4, RAID 5, and RAID 6 configurations.

In most cases, software RAID doesn't implement the hot-swapping, and so it cannot be used where continuous availability is required.

Benefits of RAID

Benefits of RAID include the following.

An improvement in cost-effectiveness because lower-priced disks are used in large numbers.

The use of multiple hard drives enables RAID to improve the performance of a single hard drive.

Increased computer speed and reliability after a crash depending on the configuration.

There is increased availability and resiliency with RAID 5. With mirroring, RAID arrays can have two drives containing the same data. It ensures one will continue to work if the other fails.

Drawbacks of RAID

RAID has the following drawbacks or disadvantages:

Nested RAID levels are more expensive to implement than traditional RAID levels because they require many disks.

The cost per gigabyte of storage devices is higher for nested RAID because many of the drives are used for redundancy.

When a drive fails, the probability that another drive in the array will also soon fail rises, which would likely result in data loss. This is because all the drives in a RAID array are installed simultaneously. So all the drives are subject to the same amount of wear.

Some RAID levels, such as RAID 1 and 5, can only sustain a single drive failure.

RAID arrays are in a vulnerable state until a failed drive is replaced and the new disk is populated with data.

When RAID was implemented, it takes a lot longer to rebuild failed drives because drives have much greater capacity.

However, nested RAID levels address these problems by providing a greater degree of redundancy, significantly decreasing the chances of an array-level failure due to simultaneous disk failures.

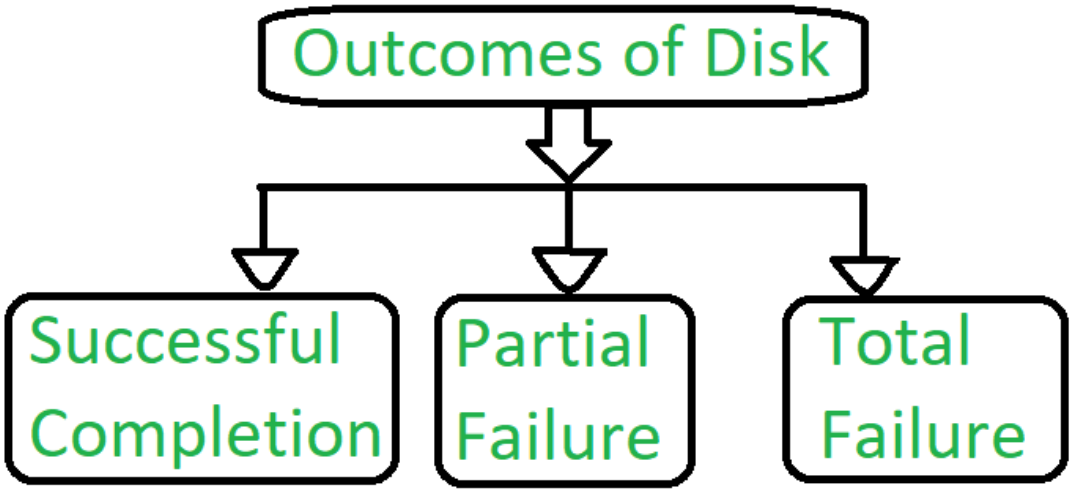
Stable-Storage Implementation in Operating system

By definition, information residing in the Stable-Storage is never lost. Even, if the disk and CPU have some errors, it will never lose any data.

Stable-Storage Implementation :

To achieve such storage, we need to replicate the required information on multiple storage devices with independent failure modes. The writing of an update should be coordinate in such a way that it would not delete all the copies of the state and that, when we are recovering from a failure, we can force all the copies to a consistent and correct value, even if another failure occurs during the recovery. In these, we discuss how to meet these needs.

The disk write operation results to one of the following outcome:



Successful completion –

The data will written correctly on disk.

Partial Failure –

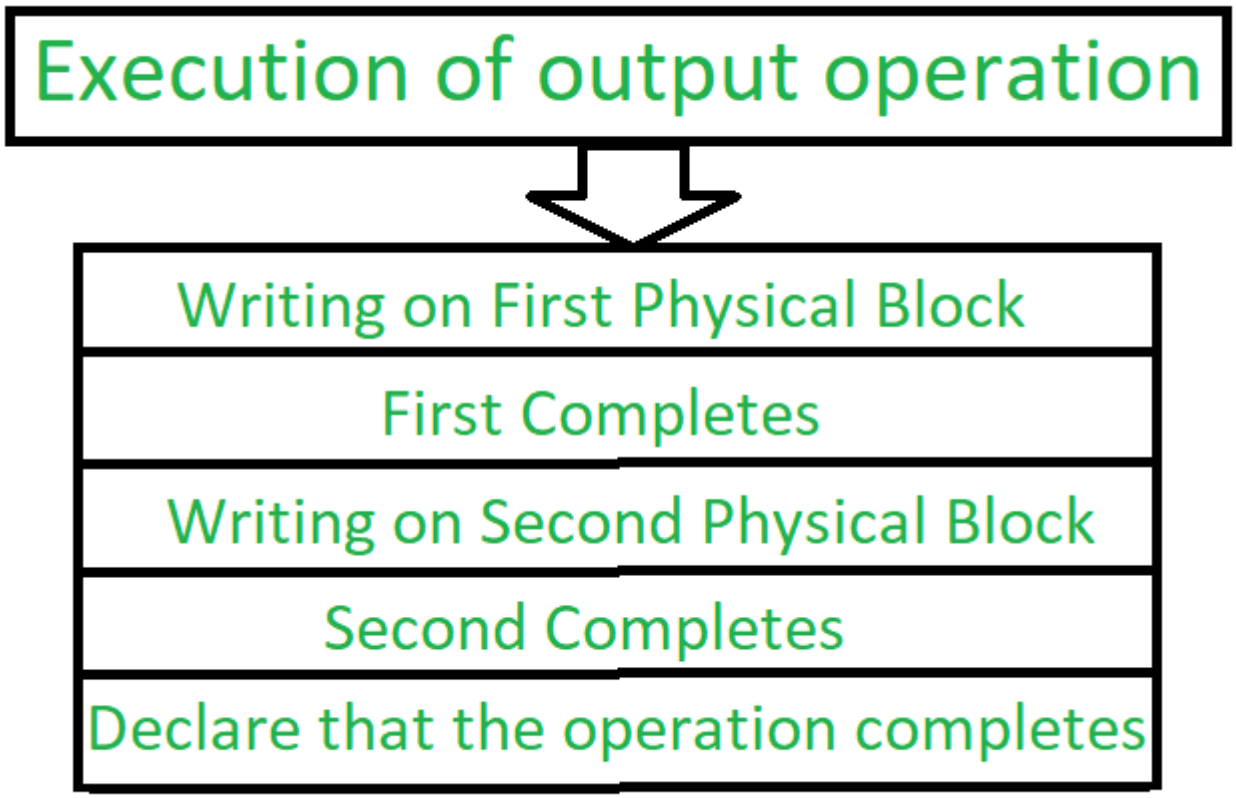
In this case, failure is occurred in the middle of the data transfer, so that only some sectors were written with the new data, and the sector which is written during the failure may have been corrupted.

Total Failure –

The failure occurred before the disk write started, so the previous data values on the disk remains intact.

During writing a block somehow if failure occurs, the system's first work is to detect the failure and then invoke a recovery process to restore the consistent state. To do that, the system must contain two physical block for each logical block.

An output operation is executed as follows:



Write the information onto the first physical block.

When the first write completes successfully, write the same operation onto the second physical block.

When both the operation declares successfully, declare the operation as complete.

During the recovery from a failure each of the physical block is examined. If both are the same and no detectable error exists, then no further action is necessary. If one block contains detectable errors then we replace its content with the value of the other block. If neither block contains the detectable error, but the block differ in content, then we replace the content of first block with the content of the second block.This procedure of the recovery give us an conclusion that either the write to stable content succeeds successfully or it results in no change.

This procedure will be extended if we want arbitrarily large number of copies of each block of the stable storage. With the usage of large number of copies, the chances of the failure reduces. Generally, it is usually reasonable to simulate

stable storage with only two copies. The data present in the stable storage is safe unless a failure destroys all the copies. The data that is present in the stable storage is guaranteed to be safe unless a failure destroys all the copies.

Because waiting for disk writes to complete is time consuming, many storage arrays add NVRAM as a cache. Since the memory is no-volatile it can be trusted to store the data en route to the disks. In this way it is considered as a part of the stable storage. Writing to the stable storage is much faster than to disk, so performance is greatly improved.

Tertiary Storage structure

Basically we want the programs and data to reside in main memory permanently.

This arrangement is usually not possible for the following two reasons:

Main memory is usually too small to store all needed programs and data permanently.

Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost.

There are two types of storage devices:-

Volatile Storage Device –

It loses its contents when the power of the device is removed.

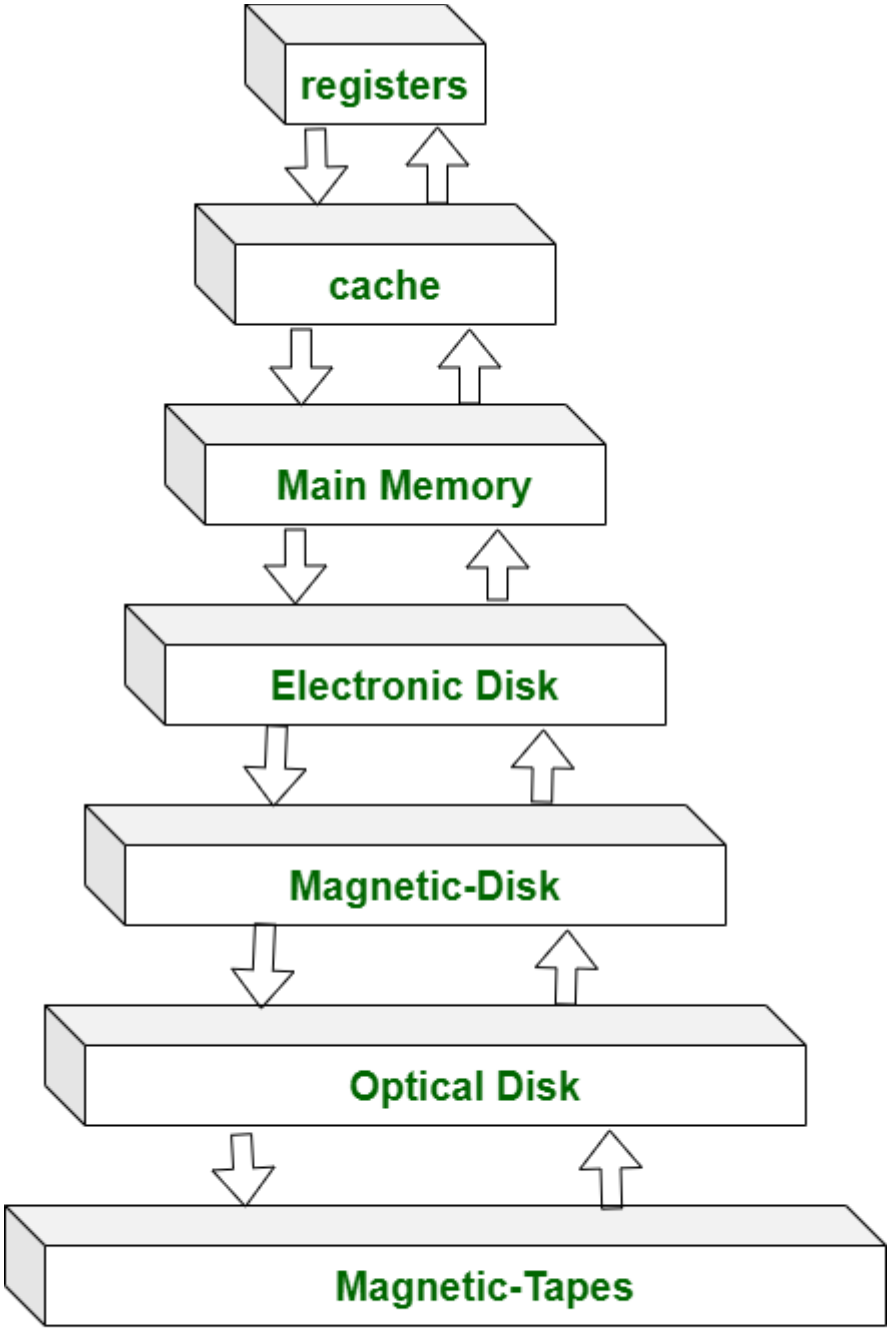
Non-Volatile Storage device –

It does not loses its contents when the power is removed. It holds all the data when the power is removed.

Secondary Storage is used as an extension of main memory. Secondary storage devices can hold the data permanently.

Storage devices consists of Registers, Cache, Main-Memory, Electronic-Disk, Magnetic-Disk, Optical-Disk, Magnetic-Tapes. Each storage system provides the basic system of storing a datum and of holding the datum until it is retrieved at a later time. All the storage devices differ in speed, cost, size and volatility. The most common Secondary-storage device is a Magnetic-disk, which provides storage for both programs and data.

In this fig Hierarchy of storage is shown –



Storage Device Hierarchy

In this hierarchy all the storage devices are arranged according to speed and cost. The higher levels are expensive, but they are fast. As we move down the hierarchy, the cost per bit generally decreases, where as the access time generally increases.

The storage systems above the Electronic disk are Volatile, where as those below are Non-Volatile.

An Electronic disk can be either designed to be either Volatile or Non-Volatile. During normal operation, the electronic disk stores data in a large DRAM array, which is Volatile. But many electronic disk devices contain a hidden magnetic hard disk and a battery for backup power. If external power is interrupted, the electronic disk controller copies the data from RAM to the magnetic disk. When external power is restored, the controller copies the data back into the RAM.

The design of a complete memory system must balance all the factors. It must use only as much expensive memory as necessary while providing as much inexpensive, Non-Volatile memory as possible. Caches can be installed to improve performance where a large access-time or transfer-rate disparity exists between two components.

Operating System Input Output I/O

The three main jobs of a computer are Input, Output, and Processing. In most of the cases, the most important job is Input / Output, and the processing is simply incidental. For an example, when we browse a web page or edit any file, our immediate attention is to read or enter some information, not for computing an answer. The fundamental role of the operating system in computer Input / Output is to manage and organize I/O operations and all I/O devices.

The various devices that are connected to the computer need to be controlled and it is a key concern of operating-system designers. This is as I/O devices vary so widely in their functionality and speed (for example a mouse, a hard disk and a CD-ROM), varied methods are required for controlling them. These methods form the I/O sub-system of the kernel of OS that separates the rest of the kernel from the complications of managing I/O devices.

I/O Hardware

Computers operate many huge kinds of devices. General categories of storage devices are like disks, tapes, transmission devices (like network interface cards, modems) and human interface devices (like screen, keyboard, etc.).

A device which communicates with the operating system of a computer by transferring signals over cable or even through the air. The Peripheral devices that communicate with the machine through a connection point also called ports- (one example is a serial port). Whenever devices use a set of wires or cables, that connecting cables are called a "bus". Simply, a bus is a collection of wires and a firmly defined protocol that describes a set of messages that can be sent on the wires.

An I/O port usually contains four different registers –

(1) control, (2) status, (3) data-in, and (4) data-out registers.

The data-in register is read by the host for getting input.

The data-out register is written by the host for sending output.

The status register holds bits which can be read by the host.

The control register is written by the host for starting a command or for changing the mode of any device.

The data registers are usually 1 to 4 bytes in size. Some of the controllers have FIFO chips which hold several bytes of input or output data for expanding the capacity of the controller beyond the size of the data register.

Operating System - I/O Hardware

One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.

An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories –

Block devices – A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.

Character devices – A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc

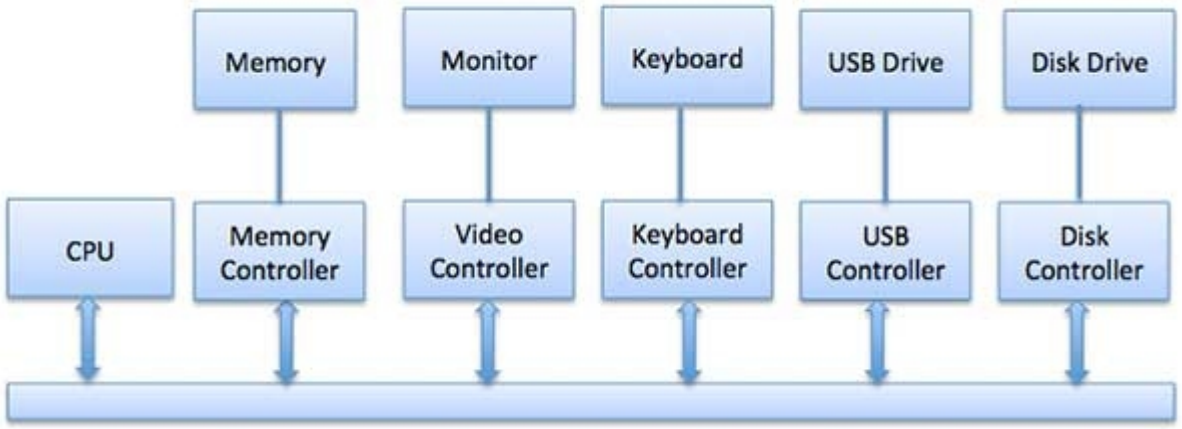
Device Controllers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices.

The Device Controller works like an interface between a device and a device driver. I/O units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.

There is always a device controller and a device driver for each device to communicate with the Operating Systems. A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary.

Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.



Synchronous vs asynchronous I/O

Synchronous I/O – In this scheme CPU execution waits while I/O proceeds

Asynchronous I/O – I/O proceeds concurrently with CPU execution

Communication to I/O Devices

The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.

Special Instruction I/O

Memory-mapped I/O

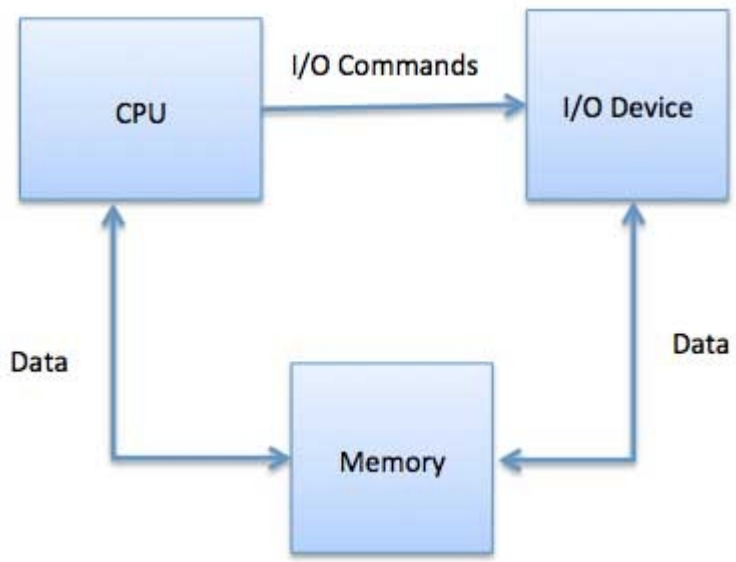
Direct memory access (DMA)

Special Instruction I/O

This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

Memory-mapped I/O

When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.



While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.

The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

Applications of Input/Output Interface

/O Interface :

There is need of surface whenever any CPU wants to communicate with I/O devices. The interface is used to interpret address which is generated by CPU. Thus, surface is used to communicate to I/O devices i.e. to share information between CPU and I/O devices interface is used which is called as I/O Interface.

Various applications of I/O Interface :

Application of I/O is that we can say interface have access to open any file without any kind of information about file i.e., even basic information of file is unknown. It also has feature that it can be used to also add new devices to computer system even it does not cause any kind of interrupt to operating system. It can also used to abstract differences in I/O devices by identifying general kinds. The access to each of general kind is through standardized set of function which is called as interface.

Each type of operating system has its own category for interface of device-drivers. The device which is given may ship with multiple device-drivers-for instance, drivers for Windows, Linux, AIX and Mac OS, devices may is varied by dimensions which is as illustrated in the following table :

Character-stream or Block :

A character stream or block both transfers data in form of bytes. The difference between both of them is that character-stream transfers bytes in linear way i.e., one after another whereas block transfers whole byte in single unit.

Sequential or Random Access :

To transfer data in fixed order determined by device, we use sequential device whereas user to instruct device to seek to any of data storage locations, random-access device is used.

Synchronous or Asynchronous :

Data transfers with predictable response times is performed by synchronous device, in coordination with others aspects of system. An irregular or unpredictable response times not coordinated with other computer events is exhibits by an asynchronous device.

Sharable or Dedicated :

Several processes or threads can be used concurrently by sharable device; whereas dedicated device cannot.

Speed of Operation :

The speed of device has range set which is of few bytes per second to few gega-bytes per second.

Read-write, read only, write-only :

Different devices perform different operations, some supports both input and output, but others supports only one data transfer direction either input or output.

Kernel I/O Subsystem in Operating System

Prerequisite – Microkernel

The kernel provides many services related to I/O. Several services such as scheduling, caching, spooling, device reservation, and error handling – are provided by the kernel, s I/O subsystem built on the hardware and device-driver infrastructure. The I/O subsystem is also responsible for protecting itself from errant processes and malicious users.

I/O Scheduling –

To schedule a set of I/O requests means to determine a good order in which to execute them. The order in which the application issues the system call is the best choice. Scheduling can improve the overall performance of the system, can share device access permission fairly to all the processes, reduce the average waiting time, response time, turnaround time for I/O to complete.

OS developers implement schedules by maintaining a wait queue of the request for each device. When an application issue a blocking I/O system call, The request is placed in the queue for that device. The I/O scheduler rearranges the order to improve the efficiency of the system.

Buffering –

A buffer is a memory area that stores data being transferred between two devices or between a device and an application. Buffering is done for three reasons.

The first is to cope with a speed mismatch between producer and consumer of a data stream.

The second use of buffering is to provide adaptation for data that have different data-transfer sizes.

he third use of buffering is to support copy semantics for the application I/O, “copy semantic ” means, suppose that an application wants to write data on a disk that is stored in its buffer. it calls the write() system’s call, providing a pointer to the buffer and the integer specifying the number of bytes to write.

Caching –

A cache is a region of fast memory that holds a copy of data. Access to the cached copy is much easier than the original file. For instance, the instruction of the currently running process is stored on the disk, cached in physical memory, and copied again in the CPU’s secondary and primary cache.

The main difference between a buffer and a cache is that a buffer may hold only the existing copy of a data item, while a cache, by definition, holds a copy on faster storage of an item that resides elsewhere.

Spooling and Device Reservation –

A spool is a buffer that holds the output of a device, such as a printer that cannot accept interleaved data streams. Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixes together.

The OS solves this problem by preventing all output from continuing to the printer. The output of all applications is spooled in a separate disk file. When an application finishes printing then the spooling system queues the corresponding spool file for output to the printer.

Error Handling –

An Os that uses protected memory can guard against many kinds of hardware and application errors so that a complete system failure is not the usual result of each minor mechanical glitch, Devices, and I/O transfers can fail in many ways, either for transient reasons, as when a network becomes overloaded or for permanent reasons, as when a disk controller becomes defective.

I/O Protection –

Errors and the issue of protection are closely related. A user process may attempt to issue illegal I/O instructions to disrupt the normal function of a system. We can use the various mechanisms to ensure that such disruption cannot take place in the system.

To prevent illegal I/O access, we define all I/O instructions to be privileged instructions. The user cannot issue I/O instruction directly.

Transforming of I/O Requests to Hardware Operations

We know that there is handshaking between device driver and device controller but here question is that how operating system connects application request or we can say I/O request to set of network wires or to specific disk sector or we can say to hardware -operations.

To understand concept let us consider example which is as follows.

Example –

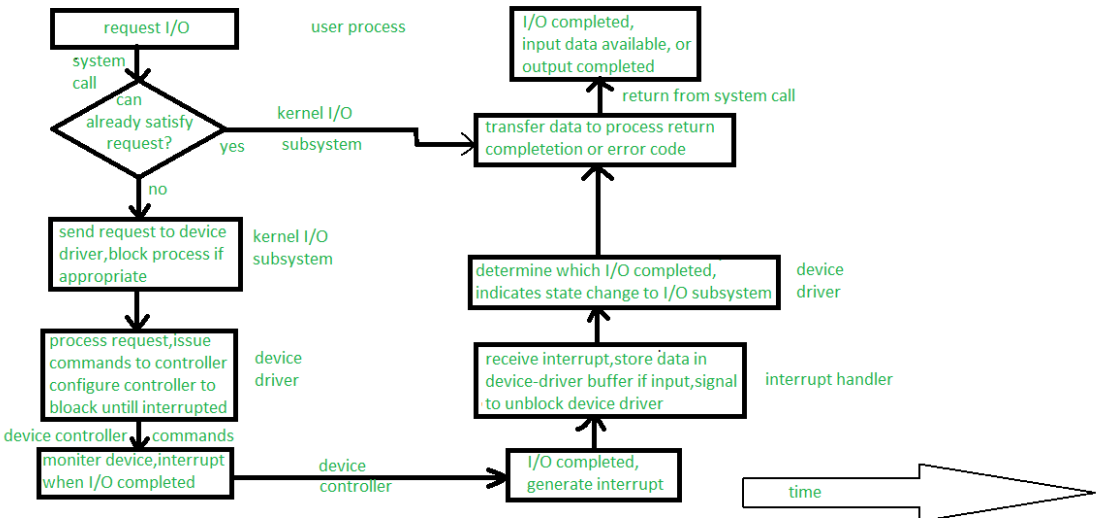
We are reading file from disk. The application we request for will refers to data by file name. Within disk, file system maps from file name through file-system directories to obtain space allocation for file. In MS-DOS, name of file maps to number that indicates as entry in file-access table, and that entry to table tells us that which disk blocks are allocated to file. In UNIX, name maps to inode number, and inode number contains information about space-allocation. But here question arises that how connection is made from file name to disk controller?

The method that is used by MS-DOS, is relatively simple operating system. The first part of MS-DOS file name, is preceding with colon, is string that identifies that there is specific hardware device.

The UNIX uses different method from MS-DOS. It represents device names in regular file-system name space. Unlike MS-DOS file name, which has colon separator, but UNIX path name has no clear separation of device portion. In fact, no part of path name is name of device. Unix has mount table that associates with prefixes of path names with specific hardware device names.

Modern operating systems gain significant flexibility from multiple stages of lookup tables in path between request and physical device stages controller. There is general mechanisms is which is used to pass request between application and drivers. Thus, without recompiling kernel, we can introduce new devices and drivers into computer. In fact, some operating system have the ability to load device drivers on demand. At the time of booting, system firstly probes hardware buses to determine what devices are present. It is then loaded to necessary drivers, accordingly I/O request.

The typical life cycle of blocking read request, is shown in the following figure. From figure, we can suggest that I/O operation requires many steps that together consume large number of CPU cycles.



System call –

Whenever, any I/O request comes, process issues blocking read() system call to previously opened file descriptor of file. Basically, role of system-call code is to check parameters for correctness in kernel. If data we put in form of input is already available in buffer cache, data is going to returned to process, and in that case I/O request is completed.

Alternative approach if input is not available –

If the data is not available in buffer cache then physical I/O must be performed. The process is removes from run queue and is placed on wait queue for device, and I/O request is scheduled. After scheduling, I/O subsystem sends request to device driver via subroutine call or in-kernel message but it depends upon operating system by which mode request will send.

Role of Device driver –

After receiving the request, device driver have to receive data and it will receive data by allocating kernel buffer space and after receiving data it will schedules I/O. After all this, command will be given to device controller by writing into device-control registers.

Role of Device Controller –

Now, device controller operates device hardware. Actually, data transfer is done by device hardware.

Role of DMA controller –

After data transfer, driver may poll for status and data, or it may have set up DMA transfer into kernel memory. The transfer is managed by DMA controller. At last when transfers complete, it will generates interrupt.

Role of interrupt handler –

The interrupt is send to correct interrupt handler through interrupt-vector table. It store any necessary data, signals device driver, and returns from interrupt.

Completion of I/O request –

When, device driver receives signal. This signal determines that I/O request has completed and also determines request's status, signals kernel I/O subsystem that request has been completed. After transferring data or return codes to address space kernel moves process from wait queue back to ready queue.

Completion of System call –

When process moves to ready queue it means process is unblocked. When the process is assigned to CPU, it means process resumes execution at completion of system call.