UNIT 5 – Device Management

Device Controller

It is a hardware program mainly utilized to connect a computer's operating system and functions in the phase by connecting the device driver. It is an electronic component that handles the link between incoming and outgoing signals in a processor by using chips.

It serves as a link between a device and any program that can receive commands from the operating system. These functions include buttons like reading, writing, etc. Every button and controller of various types of controllers differs from one another, with differences based on how they are utilized.

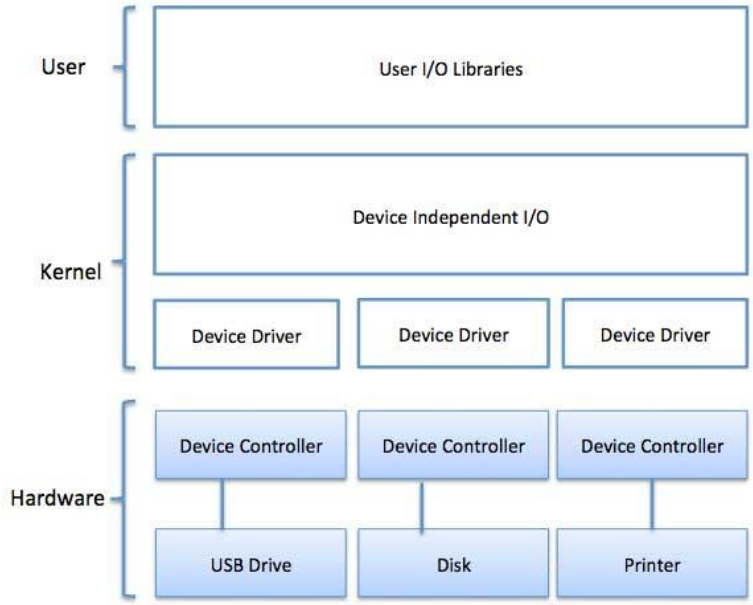
The device controller gets data from a connected system device and temporarily saves such data in a special purpose register inside the controller known as a local buffer. There is a device driver for every device controller. The memory is linked with the memory controller. The monitor is linked with the video controller, and the keyword is linked with the keyboard controller. The disk drive and USB drive are each attached to their respective disk controllers. These controllers are linked to the processor through the common bus.

Principles of I/O software:

I/O software is often organized in the following layers −

- **User Level Libraries** − This provides simple interface to the user program to perform input and output. For example, **stdio** is a library provided by C and C++ programming languages.
- **Kernel Level Modules** − This provides device driver to interact with the device controller and device independent I/O modules used by the device drivers.
- **Hardware** − This layer includes actual hardware and hardware controller which interact with the device drivers and makes hardware alive.

A key concept in the design of I/O software is that it should be device independent where it should be possible to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on a floppy disk, on a hard disk, or on a CD-ROM, without having to modify the program for each different device.



Device Drivers

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. Device drivers encapsulate device-dependent code and implement a standard interface in such a way that code contains device-specific register reads/writes. Device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM.

A device driver performs the following jobs −

- To accept request from the device independent software above to it.
- Interact with the device controller to take and give I/O and perform required error handling
- Making sure that the request is executed successfully

How a device driver handles a request is as follows: Suppose a request comes to read a block N. If the driver is idle at the time a request arrives, it starts carrying out the request immediately. Otherwise, if the driver is already busy with some other request, it places the new request in the queue of pending requests.

Interrupt handlers

An interrupt handler, also known as an interrupt service routine or ISR, is a piece of software or more specifically a callback function in an operating system or more specifically in a device driver, whose execution is triggered by the reception of an interrupt.

When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt, updates data structures and wakes up process that was waiting for an interrupt to happen.

The interrupt mechanism accepts an address ─ a number that selects a specific interrupt handling routine/function from a small set. In most architectures, this address is an offset stored in a table called the interrupt vector table. This vector contains the memory addresses of specialized interrupt handlers.

Device-Independent I/O Software

The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software. Though it is difficult to write completely device independent software but we can write some modules which are common among all the devices. Following is a list of functions of device-independent I/O Software −

- Uniform interfacing for device drivers
- Device naming - Mnemonic names mapped to Major and Minor device numbers
- Device protection
- Providing a device-independent block size
- Buffering because data coming off a device cannot be stored in final destination.
- Storage allocation on block devices
- Allocation and releasing dedicated devices
- Error Reporting

User-Space I/O Software

These are the libraries which provide richer and simplified interface to access the functionality of the kernel or ultimately interactive with the device drivers. Most of the user-level I/O software consists of library procedures with some exception like spooling system which is a way of dealing with dedicated I/O devices in a multiprogramming system.

I/O Libraries (e.g., stdio) are in user-space to provide an interface to the OS resident device-independent I/O SW. For example putchar(), getchar(), printf() and scanf() are example of user level I/O library stdio available in C programming.

Kernel I/O Subsystem

Kernel I/O Subsystem is responsible to provide many services related to I/O. Following are some of the services provided.

- **Scheduling** − Kernel schedules a set of I/O requests to determine a good order in which to execute them. When an application issues a blocking I/O system call, the request is placed on the queue for that device. The Kernel I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by the applications.
- **Buffering** − Kernel I/O Subsystem maintains a memory area known as **buffer** that stores data while they are transferred between two devices or between a device with an application operation. Buffering is done to cope with a speed mismatch between the producer and consumer of a data stream or to adapt between devices that have different data transfer sizes.
- **Caching** − Kernel maintains cache memory which is region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original.
- **Spooling and Device Reservation** − A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. The spooling system copies the queued spool files to the printer one at a time. In some operating systems, spooling is managed by a system daemon process. In other operating systems, it is handled by an in kernel thread.
- **Error Handling** − An operating system that uses protected memory can guard against many kinds of hardware and application errors.

Input/Output Software Layers in Operating System

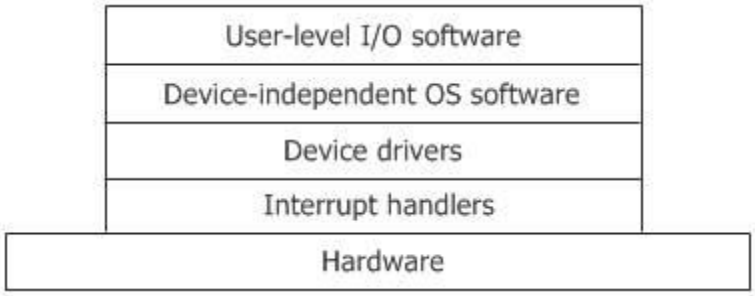
This post was written and made public with the intention of educating readers about the input/output software layers that are present in operating systems.

In its most fundamental form, input/output software can be broken down into the following four layers:

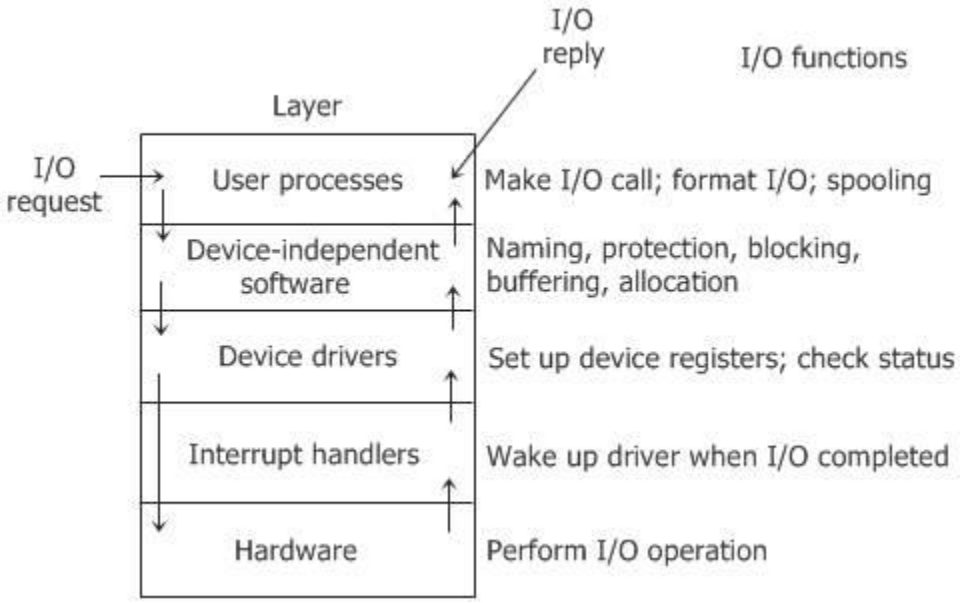
- Interrupt handlers
- Device drivers
- Device-independent input/output software
- User-space input/output software

In every input/output software system, each of the four layers has a well-defined function to perform and a well-defined interface to the adjacent layers.

The figure given below shows all the layers, along with the hardware, of the input/output software system.



Here is another figure that shows all the layers of the input/output software system along with their principal functions.



Now let's briefly describe all four input/output software layers that are listed above.

Interrupt Handlers

The interrupt procedure will perform whatever actions are necessary in order to deal with an interrupt whenever one of those events takes place.

Device Drivers

Device drivers, in their most basic form, are device-specific codes that are used solely for the purpose of controlling the input/output devices that are connected to a computer system.

The term "device drivers" is likely the most familiar to you out of the four layers described in this post; in fact, you have probably come across it at some point. It is not possible to use a device that is connected to a computer without the appropriate device driver. To give you an example, let's say that in order to use our computer to browse the internet, we need to connect a MODEM to it first. As a result, in order for our MODEM to work, the device driver that

corresponds to our MODEM needs to be installed on our computer. It is possible that the MODEM will not work if its device driver is not installed.

Device-Independent Input/Output Software

Some of the input/output software is device-specific, and other parts of that input/output software are device-independent.

The exact boundary between the device-independent software and drivers is device-dependent, and just because of that, some functions that could be done in a device-independent way sometimes are done in the drivers, for efficiency or any other reason.

Here is a list of some functions that are done in the device-independent software:

- Uniform interfacing for device drivers
- Buffering
- Error reporting
- Allocating and releasing dedicated devices
- Providing a device-independent block size

User-Space Input/Output Software

Generally, most of the input/output software is within the operating system (OS), and some small part of that input/output software consists of libraries that are linked with the user programs and even whole programs running outside the kernel.

What is RAID (Redundant Arrays of Independent Disks)?

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

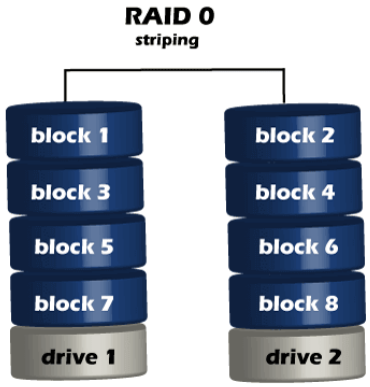
Many different ways of distributing data have been standardized into various RAID levels. Each RAID level is offering a trade-off of data protection, system performance, and storage space. The number of levels has been broken into three categories, standard, nested, and non-standard RAID levels.

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.

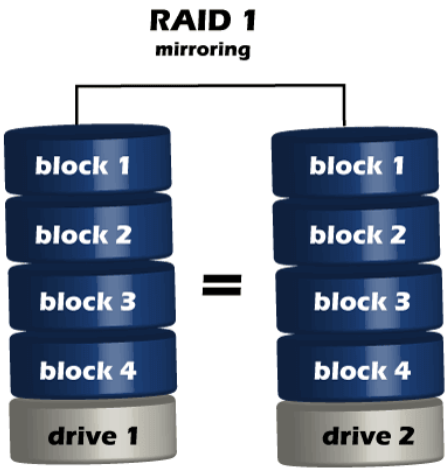


There is rarely a situation where you should use RAID 0 in a server environment. You can use it for cache or other purposes where speed is essential, and reliability or data loss does not matter at all.

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.

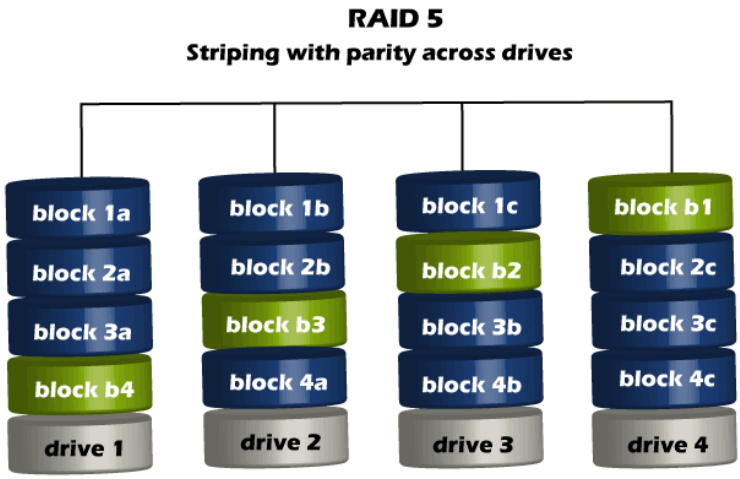
RAID 1 is capable of a much more complicated configuration. The point of RAID 1 is primarily for redundancy. If you completely lose a drive, you can still stay up and running off the other drive.



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.

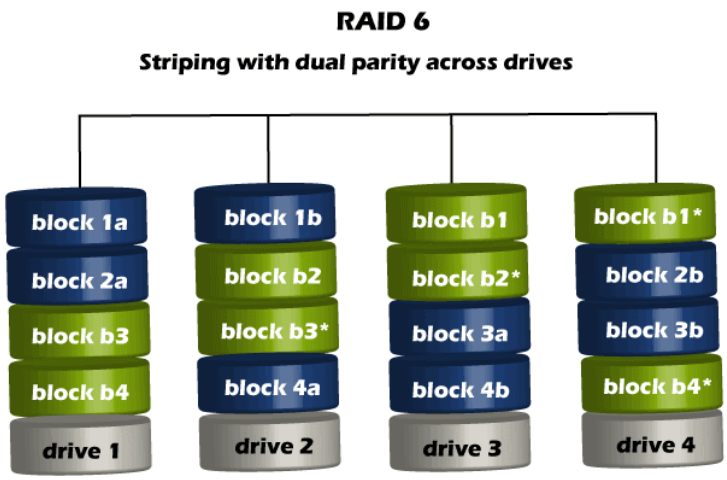
3. RAID 5(striped disks with single parity)

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.

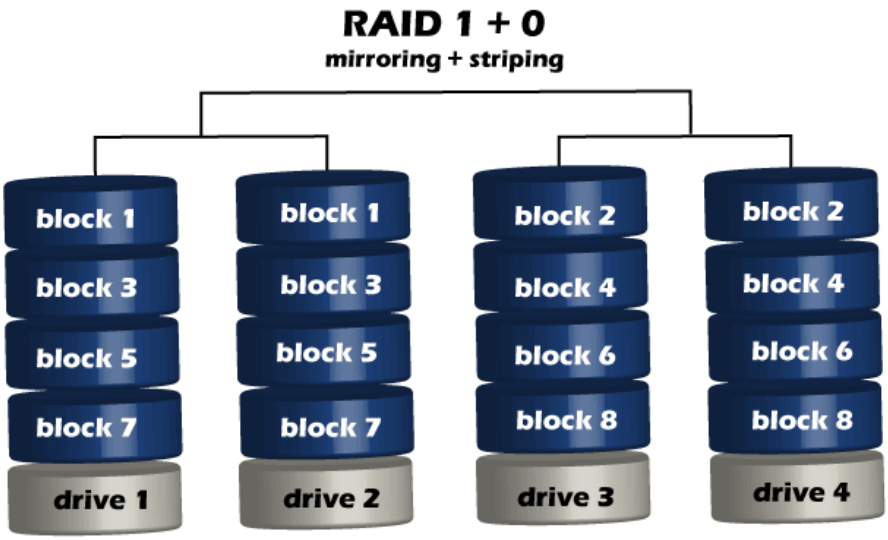
Nested RAID levels

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.

Implementation of RAID

The distribution of data across multiple drives can manage either by computer hardware or by software.

1. Hardware-based

Hardware-based RAID requires a dedicated controller installed in the server. Hardware RAID controllers can be configured through card BIOS or Option ROM before an operating system is booted. And after the operating system is booted, proprietary configuration utilities are available from each controller's manufacturer.

Hardware RAID is created using separate hardware. There are two options:

- AN inexpensive ***RAID chip***possibly built into the motherboard.
- More expensive option with a complex ***stand-alone RAID controller***. These controllers can be equipped with their CPU, battery-backed up cache memory, and typically hot-swapping.

A hardware-based RAID card does all the RAID array(s) management, providing logical disks to the system with no overhead on the part of the system itself. Additionally, hardware RAID can offer many different types of RAID configurations simultaneously to the system. This includes providing a RAID 1 array for the boot and application drive and a RAID-5 array for the large storage array.

Some other operating systems have implemented their own generic frameworks for interfacing with any RAID controller and provide tools for monitoring RAID volume status. A hardware RAID has some advantages over a software RAID, such as:

- It doesn't use the CPU of the host computer.
- It allows users to create boot partitions.
- It handles errors better since it communicates with the devices directly.
- It supports hot-swapping.

2. Software RAID

Software RAID is an included option in all of Steadfast's dedicated servers. This means there is NO cost for software RAID 1 and is highly recommended if you're using local storage on a system. It is highly recommended that drives in a RAID array be of the same type and size. Software RAID is one of the **cheapest RAID solutions**.

Software-based RAID will control some of the system's computing power to manage the RAID configuration. If you're looking to maximize a system's performance, such with a RAID 5 or 6 configurations, it's best to use a hardware-based RAID card when you're using standard HDDs.

Many modern operating systems provide software RAID implementations. Software RAID can be implemented as:

- A layer that abstracts multiple devices, thereby providing a single virtual machine.
- A layer that sits above any file system and provides parity protection to user data.

If a boot drive fails, the system must be sophisticated enough to boot from the remaining drive or drives.

There are certain limitations on the use of the software RAID to boot the system. **Only RAID 1 can contain boot partition**, while system boot is impossible with software RAID 5 and RAID 0.

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.

Disk Scheduling Algorithms

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

However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution.

The technique that operating system uses to determine the request which is to be satisfied next is called disk scheduling.

Let's discuss 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

It is the average of time spent by each request waiting for the IO operation.

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.

Goal of Disk Scheduling Algorithm

- Fairness
- High throughout
- Minimal traveling head time

Disk Scheduling Algorithms

The list of various disks scheduling algorithm is given below. Each algorithm is carrying some advantages and disadvantages. The limitation of each algorithm leads to the evolution of a new algorithm.

- FCFS scheduling algorithm
- SSTF (shortest seek time first) algorithm
- SCAN scheduling

- C-SCAN scheduling
- LOOK Scheduling
- C-LOOK scheduling