MCA Part-II

Paper-XIII: Operating System

Topic: DISK SCHEDULING

PREPARED BY: DR. KIRAN PANDEY

(School of Computer Science)

Email-id: kiranpandey.nou@gmail.com

INTRODUCTION

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

In the earlier unit, we had studied the memory management of primary memory. The physical memory, as we have already seen, is not large enough to accommodate all of the needs of a computer system. Also, it is not permanent. Secondary storage consists of disk units and tape drives onto which data can be moved for permanent storage.

The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file. Files are mapped by the operating system, onto physical devices.

Definition: A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user. Data files may be numeric, alphabetic or alphanumeric. Files may be free-form, such as text files, or may be rigidly formatted.

A File Structure should be according to a required format that the operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. UNIX, MS-DOS support minimum number of file structure.

File management is one of the most visible services of an operating system. Computers can store information in several different physical forms among which magnetic tape, disk, and drum are the most common forms. Each of these devices has their own characteristics and physical organization.

Normally files are organized into directories to ease their use. When multiple users have access to files, it may be desirable to control by whom and in what ways files may be accessed. The operating system is responsible for the following activities in connection with file management:

- The creation and deletion of files.
- The creation and deletion of directory.
- The support of primitives for manipulating files and directories.
- The mapping of files onto disk storage.
- Backup of files on stable (nonvolatile) storage.

The most significant problem in I/O system is the speed mismatch between I/O devices and the memory and also with the processor. This is because I/O system involves both H/W and S/W support and there is large variation in the nature of I/O devices, so they cannot compete with the speed of the processor and memory.

A well-designed file management structure makes the file access quick and easily movable to a new machine. Also it facilitates sharing of files and protection of non - public files. For security and privacy, file system may also provide encryption and decryption capabilities. This makes information accessible to the intended user only.

In this unit we will study the disk scheduling and the file management techniques used by the operating system in order to manage them efficiently.

THE I/O FUNCTION

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.

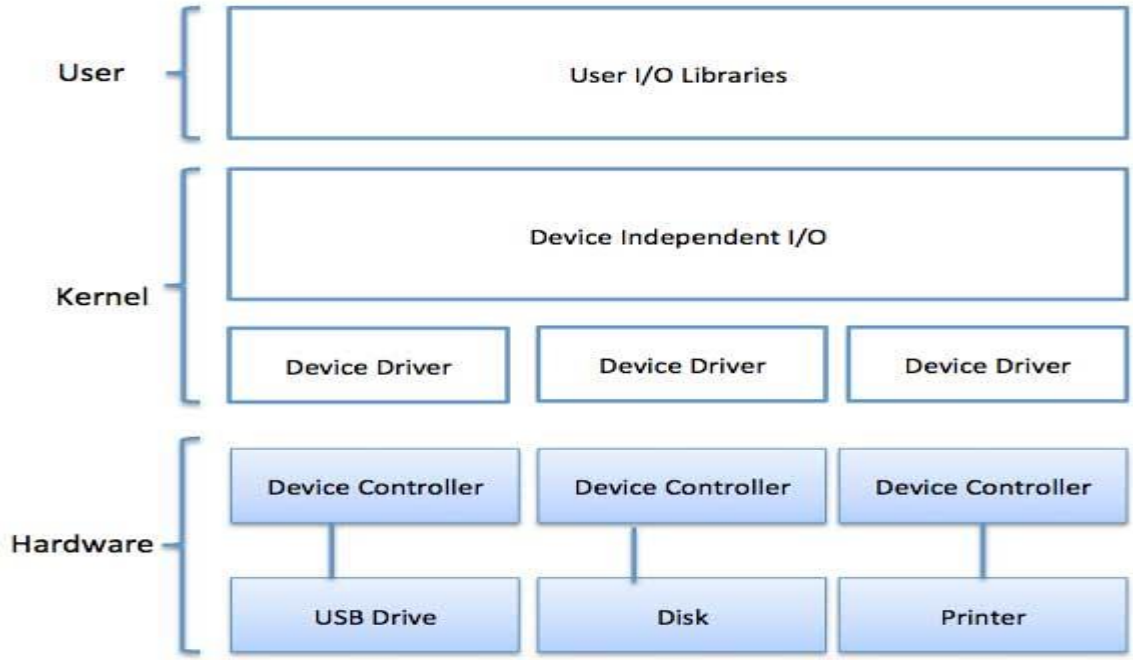

Figure 1: I/O software

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.

Based on this I/O software can be structured in the following four layers given below with brief descriptions:

(i) **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.

The CPU starts the transfer and goes off to do something else until the interrupt arrives. The I/O device performs its activity independently and simultaneously with CPU activity. This enables the I/O devices to run asynchronously with the processor. The device sends an interrupt to the processor when it has completed the task, enabling CPU to initiate a further task.

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. These interrupts can be hidden with the help of device drivers discussed below as the next I/O software layer.

(ii) **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.

(iii) **Device-independent Operating System Software:** The basic function of the device-independent operating 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

(iv) **User level software**: It consists of library procedures linked together with user programs. These libraries make system calls. 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. It makes I/O call, format I/O and also support spooling. Spooling is a way of dealing with dedicated I/O devices like printers in a multiprogramming environment.

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.

I/O BUFFERING

Input/output (I/O) buffering is a mechanism that improves the throughput of input and output operations. It is implemented directly in hardware and the corresponding drivers (hence the *block devices* found in UNIX-like systems), and is also ubiquitous among programming language standard libraries.

A buffer is an intermediate memory area under operating system control that stores data in transit between two devices or between user's work area and device. It allows computation to proceed in parallel with I/O.

In a typical unbuffered transfer situation the processor is idle for most of the time, waiting for data transfer to complete and total read-processing time is the sum of all the transfer/read time and processor time as shown in Figure 2(a).

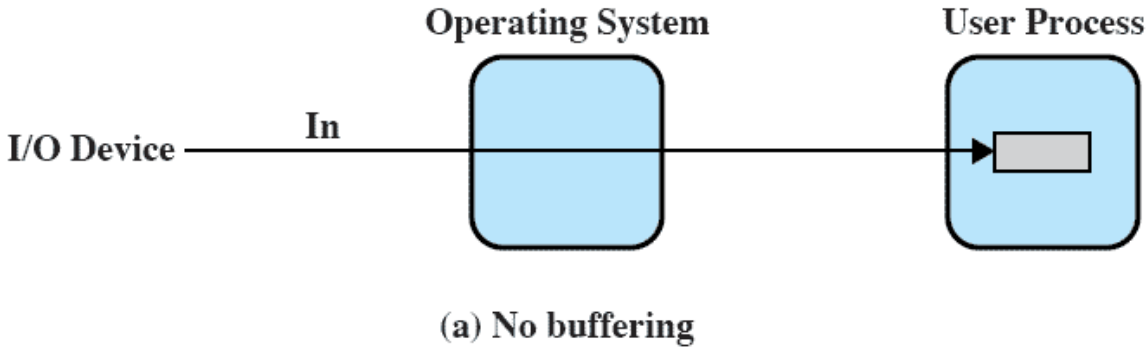

Figure 2(a) : Unbuffered Transfers

In case of single-buffered transfer, blocks are first read into a buffer and then moved to the user's work area. When the move is complete, the next block is read into the buffer and processed in parallel with the first block. This helps in minimizing speed mismatch between devices and the processor. Also, this allows process computation in parallel with input/output as shown in Figure 2(b).

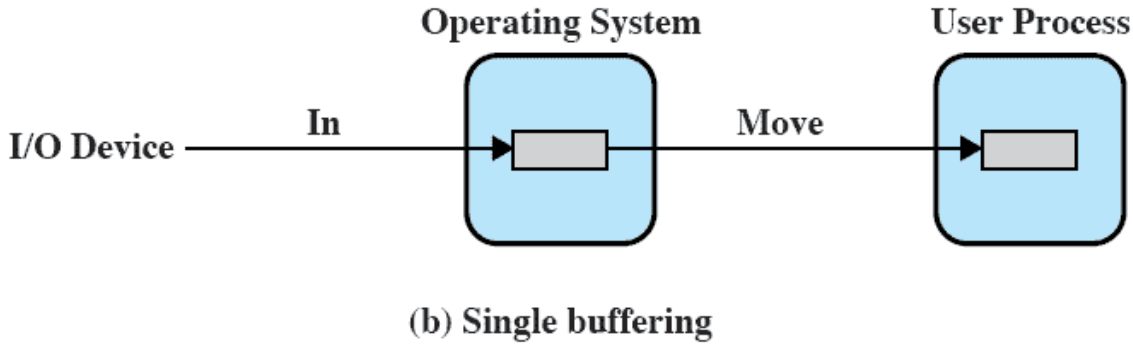

Figure 2(b): Single Buffering

Double buffering is an improvement over this. A pair of buffers is used; blocks/records generated by a running process are initially stored in the first buffer until it is full. Then from this buffer it is transferred to the secondary storage. During this transfer the other blocks generated are deposited in the second buffer and when this second buffer is also full and first buffer transfer is complete, then transfer from the second buffer is initiated. This process of alternation between buffers continues which allows I/O to

occur in parallel with a process's computation. This scheme increases the complexity but yields improved performance as shown in Figure 2©.

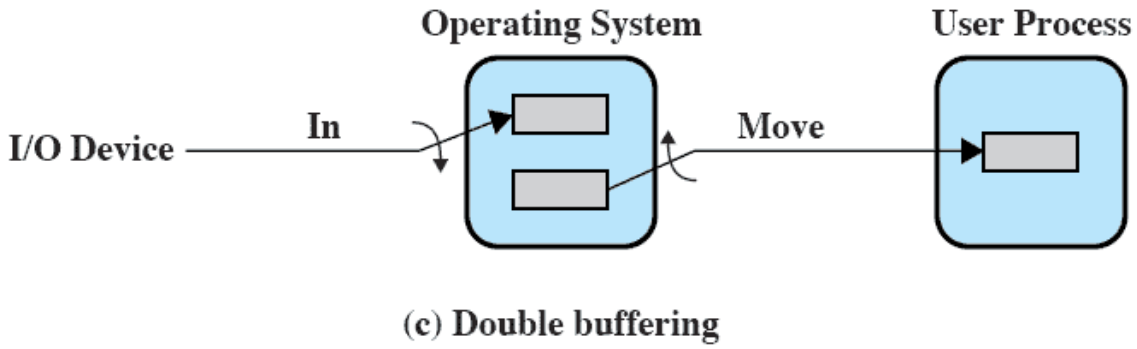

Figure 2©: Double Buffering

A circular buffer is a memory allocation scheme where memory is reused (reclaimed) when an index, incremented modulo the buffer size, writes over a previously used location. In Circular buffering more than two buffers are used. Each individual buffer is one unit in a circular buffer. It is used when I/O operation must keep up with process. Ts is most useful for bursty I/O. This scheme is shown in Figure 2(d).

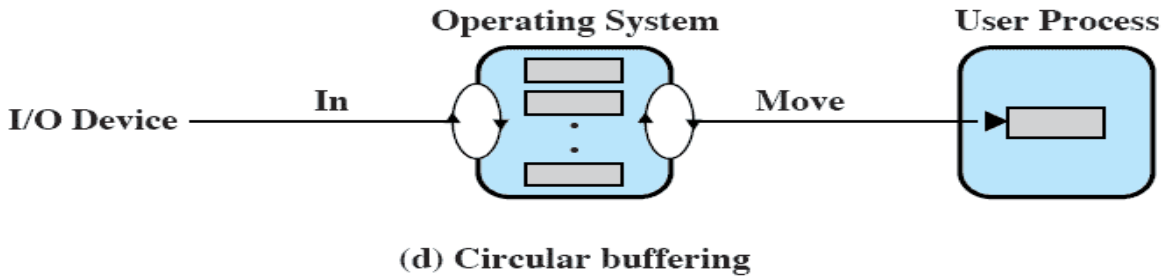

Figure 2(d): Circular Buffering

DISK ORGANISATION

A variety of Input/Output devices can be used for file storage. Magnetic disks provide the bulk of secondary storage for modern computer system. Today disks come in different shapes and sizes. The most obvious distinction between floppy disks, diskettes and hard disks is: floppy disks and diskettes consist, of a single disk of magnetic material, while hard -disks normally consist of several stacked on top of one another. Hard disks are totally enclosed devices which are much more finely engineered and therefore require protection from dust. A hard disk spins at a constant speed, while the

rotation of floppy drives is switched on and off. On the Macintosh machine, floppy drives have a variable speed operation, whereas most floppy drives have only a single speed of rotation. As hard drives and tape units become more efficient and cheaper to produce, the role of the floppy disk is diminishing. We look therefore mainly at hard drives.

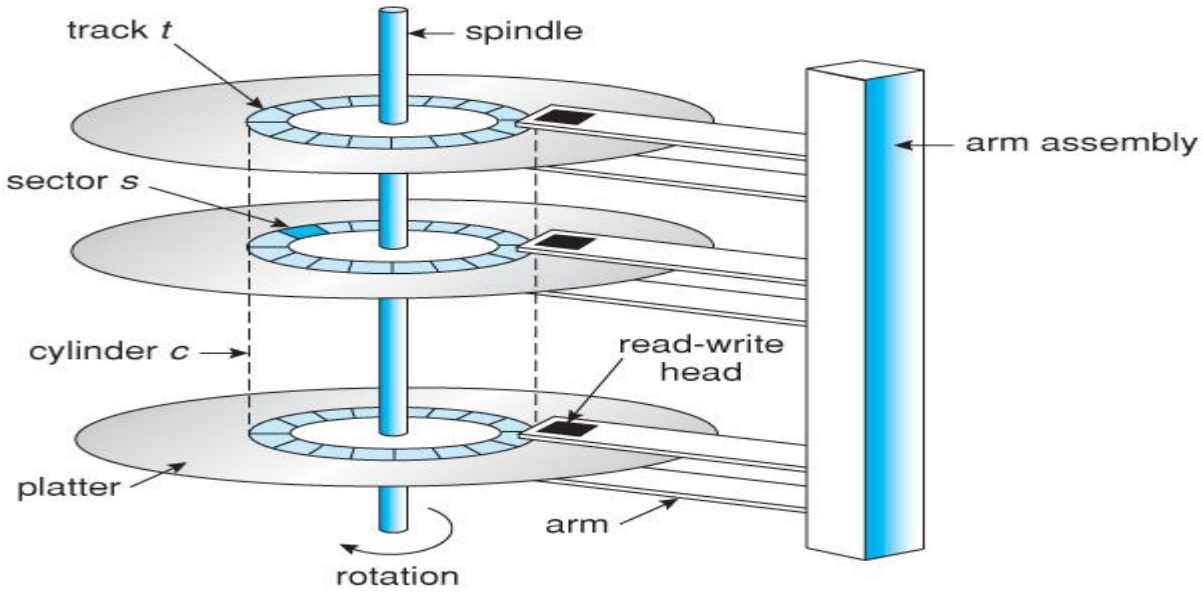

Figure 3: Hard Disk

Looking at the Figure 2, we see that a hard disk is composed of several physical disks stacked on top of each other. The disk shown in the Figure 2 has 3 platters and 6 recording surfaces (two on each platter). A separate read head is provided for each surface. Although the disks are made of continuous magnetic material, there is a limit to the density of information which can be stored on the disk. The heads are controlled by a stepper motor which moves them in fixed-distance intervals across each surface. i.e., there is a fixed number of tracks on each surface. The tracks on all the surfaces are aligned, and the sum of all the tracks at a fixed distance from the edge of the disk is called a cylinder. To make the disk access quicker, tracks are usually divided up into sectors- or fixed size regions which lie along tracks. When writing to a disk, data are written in units of a whole number of sectors. (In this respect, they are similar to pages or frames in physical memory). On some disks, the sizes of sectors are decided by the manufacturers in hardware. On other systems (often microcomputers) it might be chosen in software when the disk is prepared for use (formatting). Because the heads of the disk move together on all surfaces, we can increase read-write efficiency by allocating blocks in parallel across all surfaces. Thus, if a file is stored in consecutive blocks, on a disk with n surfaces and n heads, it could read n sectors per-track without

any head movement. When a disk is supplied by a manufacturer, the physical properties of the disk (number of tracks, number of heads, sectors per track, speed of revolution) are provided with the disk. An operating system must be able to adjust to different types of disk. Clearly sectors per track is not a constant, nor is the number of tracks. The numbers given are just a convention used to work out a consistent set of addresses on a disk and may not have anything to do with the hard and fast physical limits of the disk. To address any portion of a disk, we need a three component address consisting of (surface, track and sector).

The seek time is the time required for the disk arm to move the head to the cylinder with the desired sector. The rotational latency is the time required for the disk to rotate the desired sector until it is under the read-write head.

Device drivers

A **device driver** is a computer program that operates or controls a particular type of device that is attached to a computer. A driver provides a software interface to hardware devices, enabling operating systems and other computer programs to access hardware functions without needing to know precise details of the hardware being used.

A driver communicates with the device through the computer bus or communications subsystem to which the hardware connects. When a calling program invokes a routine in the driver, the driver issues commands to the device. Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware dependent and operating-system-specific.

The most popular type of drive for larger personal computers and workstations is the SCSI drive. SCSI (pronounced scuzzy) (Small Computer System Interface) is a protocol and now exists in four variants SCSI 1, SCSI 2, fast SCSI 2, and SCSI 7. SCSI disks live on a data bus which is a fast parallel data link to the CPU and memory, rather like a very short network. Each drive coupled to the bus identifies itself by a SCSI address and each SCSI controller can address up to seven units. If more disks are required, a second controller must be added. SCSI is more efficient at multiple accesses sharing than other disk types for microcomputers. In order to talk to a SCSI disk, an operating system must have a SCSI device driver. This is a layer of software which translates disk requests from the operating system's abstract command-layer into the language of signals which the SCSI controller understands.

They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface.

Checking Data Consistency and Formatting

Hard drives are not perfect: they develop defects due to magnetic dropout and imperfect manufacturing. On more primitive disks, this is checked when the disk is formatted and these damaged sectors are avoided. If the sector becomes damaged under operation, the structure of the disk must be patched up by some repair program. Usually the data are lost.

On more intelligent drives, like the SCSI drives, the disk itself keeps a defect list which contains a list of all bad sectors. A new disk from the manufacturer contains a starting list and this is updated as time goes by, if more defects occur. Formatting is a process by which the sectors of the disk are:

- (If necessary) created by setting out 'signposts' along the tracks,
- Labelled with an address, so that the disk controller knows when it has found the correct sector.

On simple disks used by microcomputers, formatting is done manually. On other types, like SCSI drives, there is a low-level formatting already on the disk when it comes from the manufacturer. This is part of the SCSI protocol, in a sense. High level formatting on top of this is not necessary, since an advanced enough filesystem will be able to manage the hardware sectors. Data consistency is checked by writing to disk and reading back the result. If there is disagreement, an error occurs. This procedure can best be implemented inside the hardware of the disk-modern disk drives are small computers in their own right. Another cheaper way of checking data consistency is to calculate a number for each sector, based on what data are in the sector and store it in the sector. When the data are read back, the number is recalculated and if there is disagreement then an error is signaled. This is called a cyclic redundancy check (CRC) or error correcting code. Some device controllers are intelligent enough to be able to detect bad sectors and move data to a spare 'good' sector if there is an error. Disk design is still a subject of considerable research and disks are improving both in speed and reliability by leaps and bounds.**1**

DISK SCHEDULING

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

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by disk controller. Thus other I/O requests need to wait in waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.

- Hard drives are one of the slowest parts of computer system and thus need to be accessed in an efficient manner.

There are many Disk Scheduling Algorithms but before discussing them let's have a quick look at some of the important terms:

- **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.
- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
- **Disk Access Time:** Disk Access Time is:

Disk Access Time= Seek Time + Rotational latency+ Transfer time

- **Disk Response Time:** Response Time is the average of time spent by a request waiting to perform its I/O operation. *Average Response time* is the response time of the all requests. *Variance Response Time* is measure of how individual request are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better. These scheduling algorithms are discussed below:

Example : Given the following queue -- 95, 180, 34, 119, 11, 123, 62, 64 with the read-write head initially at the track 50 and the tail track being at 199 let us now discuss the different algorithms.

1. **FCFS:** FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next. For this case it went from 50 to 95 to 180 and so on. From 50 to 95 it moved 45 tracks. If you tally up the total number of tracks you will find how many tracks it had to go through before finishing the entire request. In this example, it had a total head movement of 640 tracks.

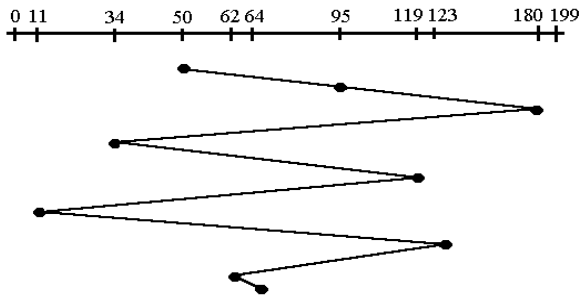

Figure 4: FCFS

Advantages:

- Every request gets a fair chance
- No indefinite postponement

Disadvantages:

- Does not try to optimize seek time
- May not provide the best possible service

2. **SSTF:** In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. *Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of. For example the next case would be to move from 62 to 64 instead of 34 since there are only 2 tracks between them and not 18 if it were to go the other way. Although this seems to be a better service being that it moved a total of 236 tracks, this is not an optimal one. There is a great chance that starvation would take place. The reason for this is if there were a lot of requests close to each other the other requests will never be handled since the distance will always be greater.* SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

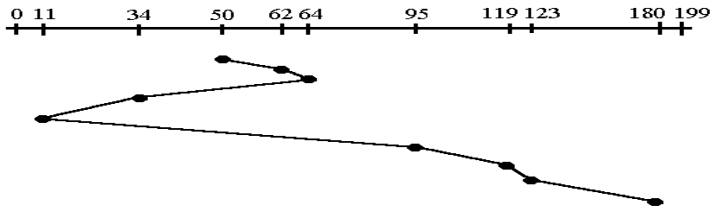

Figure 5: SSTF

Advantages:

- Average Response Time decreases
- Throughput increases

Disadvantages:

- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has higher seek time as compared to incoming requests
- High variance of response time as SSTF favours only some requests

SCAN: In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works like an elevator and hence also known as **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait. *This process moved a total of 230 tracks. Once again this is more optimal than the previous algorithm, but it is not the best.*

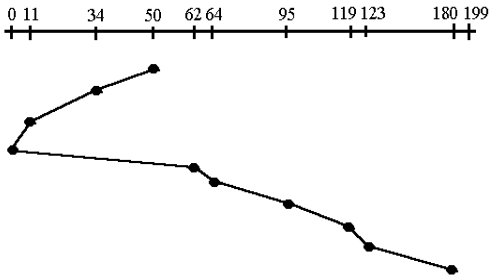

Figure 6: SCAN

Advantages:

- High throughput
- Low variance of response time
- Average response time

Disadvantages:

- Long waiting time for requests for locations just visited by disk arm

CSCAN(Circular Scan): In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area. These situations are avoided in *CSAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 track, but still this isn't the most sufficient.

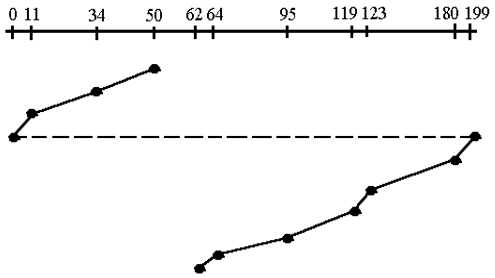

Figure 7: Circular SCAN

Advantages:

- Provides more uniform wait time compared to SCAN

LOOK: It is similar to the SCAN disk scheduling algorithm except the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

CLOOK: This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 157 tracks. From this you were able to see a scan change from 644 total head movements to just 157. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk. It is shown below in the graph.

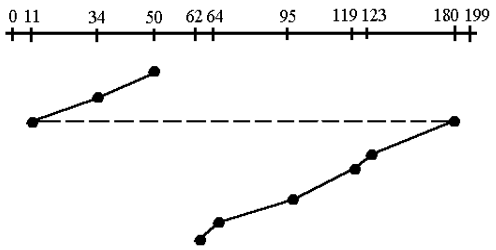

Figure 8: CLOOK

RAID

Disks have high failure rates and hence there is the risk of loss of data and lots of downtime for restoring and disk replacement. To improve disk usage many techniques have been implemented. One such technology is RAID (Redundant Array of Inexpensive Disks. In fact, RAID is the way of combining several independent and relatively small disks into a single storage of a large size. The disks included into the array are called array members. The disks can be combined into the array in different ways which are known as **RAID levels**. Each of RAID levels has its own characteristics of:

- **Fault-tolerance** which is the ability to survive of one or several disk failures.
- **Performance** which shows the change in the read and write speed of the entire array as compared to a single disk.
- **The capacity** of the array which is determined by the amount of user data that can be written to the array. The array capacity depends on the RAID level and does not always match the sum of the sizes of the RAID member disks.

How RAID is organized?

Two independent aspects are clearly distinguished in the RAID organization.

1. The **organization of data** in the array (RAID storage techniques: striping, mirroring, parity, combination of them).
2. **Implementation of each particular RAID installation** - hardware or software.

RAID storage techniques

The main methods of storing data in the array are:

- **Striping** - splitting the flow of data into blocks of a certain size (called "block size") then writing of these blocks across the RAID one by one. This way of data storage effects on the performance.
- **Mirroring** is a storage technique in which the identical copies of data are stored on the RAID members simultaneously. This type of data placement affects the fault tolerance as well as the performance.
- **Parity** is a storage technique which is utilized striping and checksum methods. In parity technique, a certain parity function is calculated for the data blocks. If a drive fails, the missing block are recalculated from the checksum, providing the RAID fault tolerance.

All the existing RAID types are based on striping, mirroring, parity, or combination of these storage techniques.

RAID levels

RAID 0: This configuration has striping, but no redundancy of data. It offers the best performance, but no fault tolerance.

RAID 1: Also known as *disk mirroring*, this configuration consists of at least two drives that duplicate the storage of data. There is no striping. Read performance is improved since either disk can be read at the same time. Write performance is the same as for single disk storage.

RAID 2: This configuration uses striping across disks, with some disks storing Error Checking and Correcting (ECC) information. It has no advantage over RAID 3 and is no longer used.

RAID 3: This technique uses striping and dedicates one drive to storing parity information. The embedded ECC information is used to detect errors. Data_recovery is accomplished by calculating the exclusive OR (XOR) of the information recorded on the other drives. Since an I/O operation addresses all the drives at the same time, RAID 3 cannot overlap I/O. For this reason, RAID 3 is best for single-user systems with long record applications.

RAID 4: This level uses large stripes, which means you can read records from any single drive. This allows you to use overlapped I/O for read operations. Since all write operations have to update the parity drive, no I/O overlapping is possible. RAID 4 offers no advantage over RAID 5.

RAID 5: This level is based on block-level striping with parity. The parity information is striped across each drive, allowing the array to function even if one drive were to fail. The array's architecture allows read and write operations to span multiple drives. This results in performance that is usually better than that of a single drive, but not as high as that of a RAID 0 array. RAID 5 requires at least three disks, but it is often recommended to use at least five disks for performance reasons.

RAID 5 arrays are generally considered to be a poor choice for use on write-intensive systems because of the performance impact associated with writing parity information. When a disk does fail, it can take a long time to rebuild a RAID 5 array. Performance is usually degraded during the rebuild time, and the array is vulnerable to an additional disk failure until the rebuild is complete.

RAID 6: This technique is similar to RAID 5, but includes a second parity scheme that is distributed across the drives in the array. The use of additional parity allows the array to continue to function even if two disks fail simultaneously. However, this extra protection comes at a cost. RAID 6 arrays have a higher cost per gigabyte (GB) and often have slower write performance than RAID 5 arrays.

Nested RAID levels

Some RAID levels are referred to as *nested RAID* because they are based on a combination of RAID levels. Here are some examples of nested RAID levels.

RAID 10 (RAID 1+0): Combining RAID 1 and RAID 0, this level is often referred to as RAID 10, which offers higher performance than RAID 1, but at a much higher cost. In RAID 1+0, the data is mirrored and the mirrors are striped.

RAID 01 (RAID 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.

RAID 03 (RAID 0+3, also known as RAID 53 or RAID 5+3): This level uses striping (in RAID 0 style) for RAID 3's virtual disk blocks. This offers higher performance than RAID 3, but at a much higher cost.

RAID 50 (RAID 5+0): This configuration combines RAID 5 distributed parity with RAID 0 striping to improve RAID 5 performance without reducing data protection.

RAID implementations

RAID can be created by two different ways:

- with the use of operating system drivers, so called **software RAID**;
- with the use of special hardware, so called **hardware RAID**.