UNIT 5: INPUT/OUTPUT MANAGEMENT

1. Introduction

In a computer system there are varieties of input and output device with different speed and function. An input/output management module is responsible to manage and control such types of devices. I/O refers to the communication between information processing system, such as a computer, and the outside world possibly a human, or another information processing system. Inputs are the signals or data received by the system, and outputs are the signals or data sent from it. Input output management is also one of the primary responsibilities of an operating system.

2. Principle of I/O Hardware

Different people look input/output hardware in different ways. Electrical engineers look it in term of chips, wires, power supplies and all other physical component that make up the hardware. Programmers look it as the interface presented to the software. The hardware accepts commands, carry out functions and report the result. The role of operating system in computer system is to manage and control I/O operation and I/O devices.

2.1. I/O Devices

These are the devices that communicate with a computer system by sending signals over a cable. The devices use a common set of wires, called bus, for communication.

An input device is a device by which the computer takes input from the outside world. An input device:

- Accept data from outside world
- Convert data in computer readable form
- Passes the data to the CPU for further action.

An output device is a device by which the computer gives the information to the users. An output device:

- Receive data from the CPU in computer readable form.
- Convert these information into user readable form.
- Gives the result to the user.

I/O devices can be of two types:

- **Block Device:** A device which stores information in fixed size blocks, each one with its own address is termed as block device. Common block size would be range from 512 bytes to 32,768 bytes. The main feature of this device is that it reads or writes each block independently.
  Example: Disks are the most common block devices.
- **Character Device:** A character device delivers or accepts a stream of characters, without regard to any block structure. It is not addressable and does not have any seek operation.
  Example: printer, mouse, etc

2.2. <u>Device Controllers</u>

- An electronic device in the form of chip or circuit board that controls functioning of the I/O device is called the device controller or I/O Controller or adopter. The operating system directly deals with the device controller.
- It is the hardware that controls the communication between the system and the peripheral drive unit.
- Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller.
- Each device controller has a local buffer and a command register. It communicates with the CPU by interrupts.
- A device's controller plays an important role in the operation of that device; it functions as a bridge between the device and the operating system.
- I-O devices generally contain two parts: mechanical and electrical part. This electrical part is known as a device controller and can take the form of a chip on personal computers and mechanical part is a device.
- It takes care of low level operations such as error checking, moving disk heads, data transfer, and location of data on the device.
- There are many device controllers in a computer system. Example: serial port controller, SCSI bus controller, disk controller

<u>Function of device controllers:</u>

- Stops and starts the activity of the peripheral device.
- Generate error checking code.
- Checks the error in the data received from the interface.
- Abort that command which have errors.
- Retry the command having an error
- Receives the control signals from the interface unit
- Convert the format of the data
- Check the status of the device.

2.3. <u>Memory Mapped I/O</u>

Each controller has a few I/O registers that are used for communicating with the CPU. By writing into these registers, the operating system can command the device to deliver data, accept data, switch itself on or off, or otherwise perform some action. By reading from these registers, the operating system can learn what the device's state is, whether it is prepared to accept a new command, and so on.

The system that contains I/O registers as the part of regular memory address space is known as memory mapped I/O. Each control register is assigned a unique memory address to which no memory is assigned. Usually, the assigned addresses are at the top of the address space.

In short, the scheme that assigns specific memory location to I/O devices is known as memory mapped I/O. Thus communication to and from I/O device can be same as reading and writing to memory address that responds to the I/O devices.

2.4. Direct Memory Access (DMA)

DMA Stands for "Direct Memory Access." DMA is a method of transferring data from the computer's RAM to another part of the computer without processing it using the CPU. Most of the data that is input to or output from the computer is processed by the CPU, but some data does not require processing, or can be processed by another device. In these situations, DMA can save processing time and is a more efficient way to move data from the computer's memory to other devices.

In other words, DMA is a method that allows an I/O device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations. DMA is an essential feature of all modern computers, as it allows devices to transfer data without subjecting the CPU to a heavy overhead. With DMA, the CPU gets freed from this overhead and can do useful tasks during data transfer. The process is managed by a chip known as a DMA controller (DMAC). A DMA Controller is a device, which takes over the system bus to directly transfer information from one part of the system to another.

For example, a sound card may need to access data stored in the computer's RAM, but since it can process the data itself, it may use DMA to bypass the CPU. Video cards that support DMA can also access the system memory and process graphics without needing the CPU

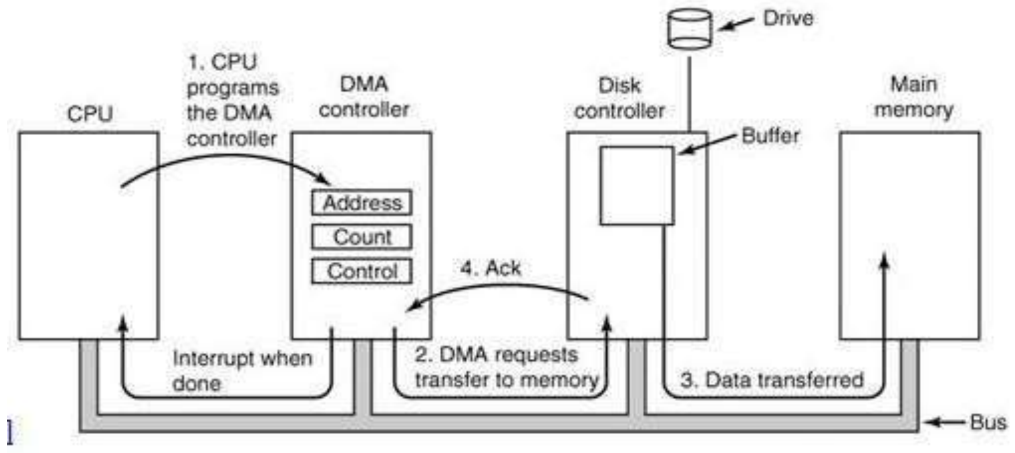


Fig: Operation of DMA transfer

First the CPU programs the DMA controller by setting its registers so it knows what to transfer and where (step 1 in Fig.). The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller (step 2). This read request looks like any other read request, and the disk controller does not know or care whether it came from the CPU or from a DMA controller. Typically, the memory address to write to is on the address lines of the bus so when the disk controller fetches the next word from its internal buffer, it knows where to write it. The write to memory is another standard bus cycle (step 3). When the write is complete, the disk controller sends an acknowledgement signal to the disk controller over the bus (step 4). The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0, steps 2 through 4 are repeated until the count reaches 0. At this point the controller interrupts the CPU to let it know that the transfer is now complete.

Working of DMA in short

- Transfers a block of data directly to or from memory
- An interrupt is sent when the task is complete.
- The processor is only involved at the beginning and end of the transfer

3. Principle Of I/O Software

3.1. Goals of I/O Software

The main goals of I/O software are:

- **Device Independence:** It means that 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.
- **Uniform Naming:** The name of a file or a device should simply be a string or an integer and not depend on the device in any way. The naming scheme for any device should be uniform or similar.
- **Error Handling:** Errors should be handled as close to the hardware as possible. If the controller discovers a read error, it should try to correct the error itself if it can. If it cannot, then the device driver should handle it.
- **Buffering:** Data that come off a device cannot be stored directly in its final destination. The data must be put into the buffer before storing in its final destination to increase the rate of data transfer.
- **Provide a Device Independent Block Size:** The sector size and block size vary from device to device. The I/O software must provide a uniform block size by treating many sectors as a single logical block.

3.2. Program I/O

When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module. In programmed I/O, the I/O module will perform the required action and then set the appropriate bits in the I/O status register. The I/O module takes no further action to alert the processor. In particular, it doesn't interrupt the processor. After the I/O instruction is invoked, the processor must periodically check the status of the I/O module until it finds that the operation is complete.

With this technique, the processor is responsible for extracting data from main memory for output and storing data in main memory for input. I/O software is written in such a way that the processor executes instructions that give it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data. Thus, the instruction set includes I/O instructions in the following categories:

Control: Used to activate an external device and tell it what to do. For example, a magnetic-tape unit may be instructed to rewind or to move forward one record.

Status: Used to test various status conditions associated with an I/O module and its peripherals.

Transfer: Used to read and/or write data between processor registers and external devices.

3.3. Interrupt Driven I/O

The problem with the programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of more data. The processor, while waiting, must repeatedly interrogate the status of the I/O module. As a result the performance level of entire system is degraded.

An alternative approach for this is interrupt driven I/O. in this approach, the processor issue an I/O command to a module and then go on to do some other useful work. The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor. The processor then executes the data transfer and then resumes its former processing. Interrupt-driven I/O still consumes a lot of time because every data has to pass with processor.

The interrupt driven I/O works as follow:

For input, the I/O module receives a READ command from the processor. The I/O module then proceeds to read data in from an associated peripheral. Once the data are in the module's data register, the module signals an interrupt to the processor over a control line. The module then waits until its data are requested by the processor. When the request is made, the module places its data on the data bus and is then ready for another I/O operation.

Sometimes there will be multiple I/O modules in a computer system, so mechanisms are needed to enable the processor to determine which device caused the interrupt and to decide, in the case of multiple interrupts, which one to handle first.

3.4. I/O Using DMA

Interrupt-driven I/O is more efficient than simple programmed I/O but it still requires the active intervention of the processor to transfer data between memory and an I/O module, and any data transfer must traverse a path through the processor. Thus both of these forms of I/O suffer from two inherent drawbacks:

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.
- The processor is tied up in managing an I/O transfer and a number of instructions must be executed for each I/O transfer.

When large volumes of data are to be moved, a more efficient technique is required and this technique is by using DMA. The DMA function can be performed by a separate module on the system bus, or it can be incorporated into an I/O module. In either case, the technique works as follows:

When the processor wishes to read or write a block of data, it issues a command to the DMA module, by sending to the DMA module the following information.

- Whether a read or write is requested.
- The address of the I/O devices involved.
- The starting location in memory to read data from or write data to.
- The number of words to be read or written.

The processor then continues with other work. It has delegated the I/O operation to the DMA module. The DMA module transfers the entire block of data, one word at a time, directly to or from memory, without going through the processor. When the transfer is complete, the

DMA module sends an interrupt signal to the processor. Thus the processor is involved only at the beginning and at the end of the transfer.

The DMA module needs to take control of the bus to transfer data to and from memory. Thus, a DMA module controls the exchange of data between main memory and an I/O module. The processor sends a request for the transfer of a block of data to the DMA module and is interrupted only after the entire block has been transferred. DMA is far more efficient than interrupt driven or programmed I/O.

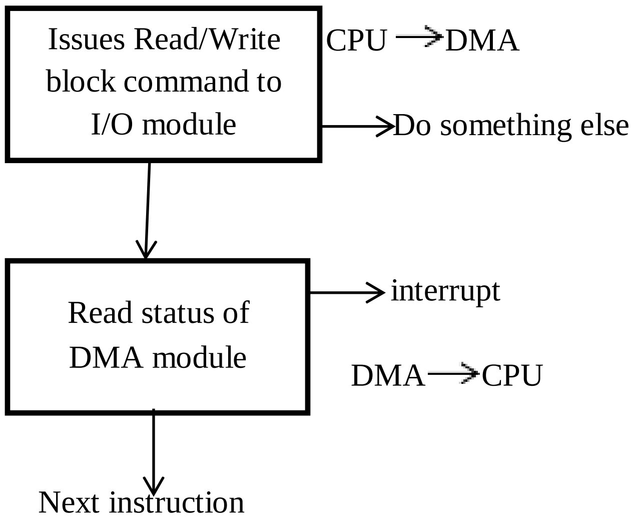

Issues Read/Write block command to I/O module
CPU → DMA
Do something else
Read status of DMA module
interrupt
DMA → CPU
Next instruction

Fig: DMA

4. I/O Software Layers

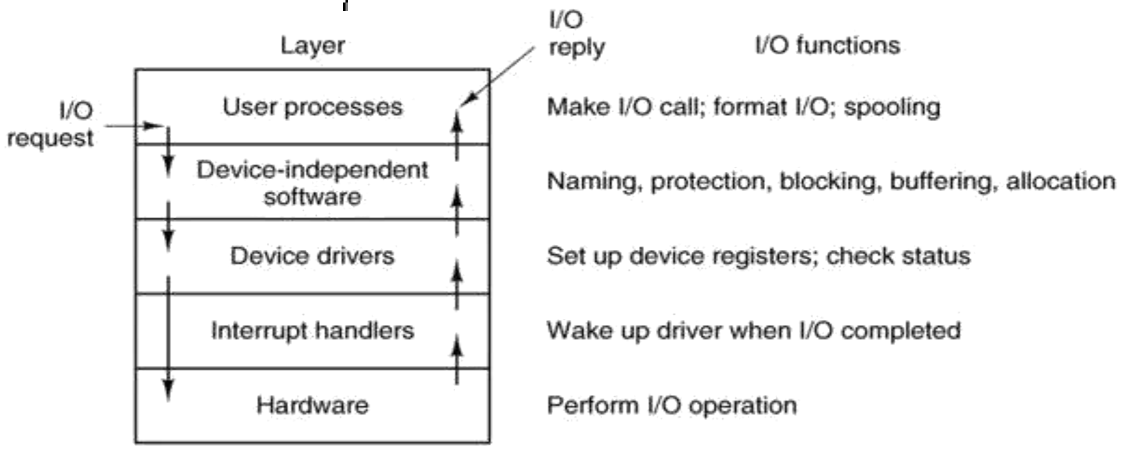

Layer
I/O reply
I/O functions
I/O request
User processes
Make I/O call; format I/O; spooling
Device-independent software
Naming, protection, blocking, buffering, allocation
Device drivers
Set up device registers; check status
Interrupt handlers
Wake up driver when I/O completed
Hardware
Perform I/O operation

Fig. Layers of the I/O software system and the main functions of each layer.

4.1. Interrupt Handler

It is the bottom layer of I/O software. When an event occurs the micro-controller generates a hardware interrupt. The interrupt forces the micro-controller's program counter to jump to a specific address in program memory. This special memory address is called the interrupt vector. At this memory location we install a special function known as an interrupt service routine (ISR) which is also known as an interrupt handler. So when a hardware interrupt is generated, program execution jumps to the interrupt handler and executes the code in that handler. When the handler is done, then program control returns the micro-controller to the original program it was executing. So a hardware interrupt allows a micro-controller to interrupt an existing program and react to some external hardware event.

4.2. <u>Device Drivers</u>

A driver is a small piece of software that tells the operating system and other software how to communicate with a piece of hardware. A **device driver** is a computer program that operates or controls a particular type of device that is attached to a computer. A device 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 device driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects. Device drivers are hardware-dependent and operating system specific.

Each device controller has registers used to give it commands or to read out its status or both. The number of registers and the nature of the commands vary radically from device to device. Each I/O device attached to a computer needs some device-specific code for controlling it. This code, called the device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM. Since each operating system needs its own drivers, device manufacturers commonly supply drivers for several popular operating systems. Each device driver normally handles one device type, or one class of closely related devices.

4.3. <u>Device Independent I/O Software</u>

Device independence is the process of making a software application to be able to function on a wide variety of devices i.e. the capability of a program, operating system or programming language to work on varieties of computers or peripherals, despite their electronic variation. 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. It also helps in buffering, error reporting, allocating and releasing dedicated devices and providing a device-independent block size.

4.4. <u>User Space I/O Software</u>

It is the upper most layer of I/O system. This layer mainly represents the outermost user interface that occurs between the operating system and end user. A user interface is that portion of an interactive computer system that communicates with the user. Proper design of a user interface can make a significant difference in training time, performance speed, error rates, user satisfaction, and the user's retention of knowledge of operations over time.

5. <u>Disk</u>

5.1. <u>Disk Hardware</u>

A hard disk is part of a computer system that stores and provides relatively quick access to large amounts of data i.e. a disk drive is a randomly addressable and rewritable storage device. A hard disk is really a set of stacked "disks," each of which has data recorded electromagnetically in concentric circles or "tracks" on the disk. A "head" records (writes) or reads the information on the tracks. Two heads, one on each side of a disk, read or write the data as the disk spins. Each read or write operation requires that data be located, which is an operation called a "seek."

Technical terms related to Disk Drive:

Platter: Hard drives are normally composed of multiple disks called platters. These platters are stacked on top of each other. It is the platter that actually stores the data.

Spindle/Motor: The platters are attached at the center to a rod or pin called a spindle that is directly attached to the shaft of the motor that controls the speed of rotation.

Head-Actuator Assembly: This assembly consists of an actuator, the arms, the sliders and the read/write heads. The actuator is the device that moves the arms containing read/write heads across the platter surface in order to store and retrieve information. The head arms move between the platters to access and store data. At the end of each arm is a head slider, which consists of a block of material that holds the head. The read/write heads convert the electronic 0s and 1s in the magnetic fields on the disks.

Logic Board: Logic boards consisting of chips, memory and other components control the disk speed and direct the actuator in all its movements. It also performs the process of transferring data from the computer to the magnetic fields on the disk.

Tracks: A track is a concentric ring on the disk where data is stored.

Cylinders: On drives that contain multiple platters, all the tracks on all the platters that are at the same distance from the center are referred to as a cylinder. The data from all the tracks in the cylinder can be read by simply switching between the different heads, which is much faster than physically moving the head between the different tracks on a single disk.

Sectors: Tracks are further broken down into sectors, which are the smallest units of storage on a disk. A larger number of sectors are recorded on the outer tracks, and progressively fewer toward the center. Data can also be read faster from the outer tracks and sectors than the inner ones.

Clusters: Sectors are grouped together into clusters. All the sectors in one cluster are taken as a single unit.

Extents: A set of contiguous clusters storing a single file or part of a file is called an extent. Reducing the number of extents in a file is achieved using a process known as defragmentation, which improves performance.

Bandwidth: The disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

Disk Management is used to manage the drives (hard disk drives, optical disk drives, flash drives) installed in a computer. Disk Management can be used to partition drives, format drives, assign drive letters, delete partition, and much more.

6. Disk Scheduling

When the disk drive is operating, the disk is rotating at constant speed. To read or write, the head must be positioned at the desired track and at the beginning of the desired sector on that track. Track selection involves moving the head in a moveable-head system. The time it takes to position the head at the proper track is known as **seek time.** In either case, once the track is selected, the disk controller waits until the appropriate sector rotates to line up with the head. The time it takes for the beginning of the sector to reach the head is known as **rotational delay** or **rotational**

latency. The sum of seek time if any and the rotational delay is the **access time**, the time it takes to get into position to read or write.
Once the head is in position, the read or write operation is performed as the sector moves under the head. The time required to transfer the required data from the sector to the memory is called **transfer time**.

7. Disk Scheduling Algorithms

The process of scheduling a request among several requests is called disk scheduling. The algorithm used to select which I/O request is going to be satisfied first is called disk scheduling algorithm. Some of the algorithms used for disk scheduling are:

7.1. First Come First Serve (FCFS) Scheduling

The simplest form of disk scheduling is the first-come, first-served (FCFS) algorithm. In this algorithm, the first request to arrive is the first one to be serviced. This algorithm implements a fair policy, but it generally does not provide the fastest service.
Example, a disk queue with requests for I/O to blocks on cylinders in that order;
98, 183, 37, 122, 14, 124, 65, 67
Working:
- If the disk head is initially at cylinder 53,
- It will first move from 53 to 98,
- then to 183, 37, 122, 14, 124, 65, and finally to 67,
- For a total head movement of 640 cylinders.

Total head movement= (98-53)+(183-98)+(183-37)+(122-37)+(122-14)+(124-14)+(124-65)+(67-65)
= 45+85+146+85+108+110+59+2
=640tracks

7.2. Shortest Seek Time First (SSTF) Scheduling

The SSTF service all the requests close to the current head position before moving the head far away to service other requests. This algorithm selects the request with the minimum seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.

SSTF scheduling is essentially a form of shortest-job-first (SJF) scheduling. Like SJF scheduling, it may cause starvation of some requests (steady supply of shorter seek time requests). Although the SSTF algorithm is a substantial improvement over the FCFS algorithm, it is not optimal
Example:
98, 183, 37, 122, 14, 124, 65, 67
Suppose initially the head is at cylinder 53. It will move as: **53, 65, 67, 37, 14, 98, 122, 124, 183**
This scheduling method results in a total head movement of only 236 cylinders. This algorithm gives a substantial improvement in performance.
Total head movement=(65-53)+(67-65)+(67-37)+(37-14)+(98-14)+(122-98)+(124-122)+(183-124)
=12+2+30+23+84+24+2+59
=236tracks

7.3. SCAN Scheduling

In the SCAN algorithm, the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. The direction of sweep must be chosen.

The SCAN algorithm is sometimes called the elevator algorithm, since the disk arms behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.

Example:

98, 183, 37, 122, 14, 124, 65, 67

Suppose initially the head is at cylinder 53 and there are cylinders from 0-190. Assume we are going down (i.e. towards 0). It will move as: **53, 37,14,0,65,67,98,122,124,183**

Total head movement=(53-37)+(37-14)+(14-0)+(65-0)+(67-65)+(98-67)+(122-98)+
(124-122)+(183-124)
=16+23+14+65+2+31+24+2+59
=233tracks

7.4. Circular Scan (C-SCAN) Scheduling

Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, CSCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.

Example:

98, 183, 37, 122, 14, 124, 65, 67

Suppose initially the head is at cylinder 53 and there are cylinders from 0-190. Assume we are going down (i.e. towards 0). It will move as: **53, 37,14,0,190,183,124,122,98,67,65**

Total head movement=(53-37)+(37-14)+(14-0)+(190-0)+(190-183)+(183-124)+(124-122)+
(122-98)+(98-67)+(67-65)
=16+23+14+190+7+59+2+24+31+2
=368tracks

7.5. Look Scheduling

It is a modified algorithm of SCAN scheduling. In this algorithm, the head goes only as far as the final request in each direction and then reverse its direction. It implies looking for a request before moving in that direction.

Example:

98, 183, 37, 122, 14, 124, 65, 67

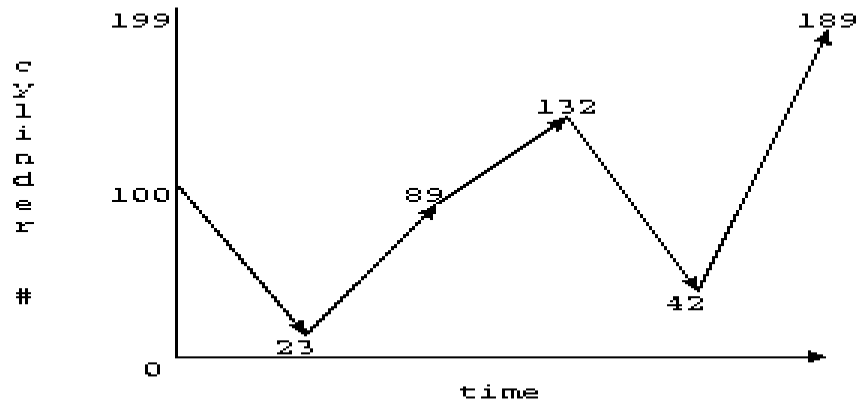
Suppose initially the head is at cylinder 53 and there are cylinders from 0-190. Assume we are going down (i.e. towards 0). It will move as: **53, 37, 14,65,67,98,122,124,183**

Total head movement=(53-37)+(37-14)+(65-14)+(67-65)+(98-67)+(122-98)+
(124-122)+(183-124)
=16+23+51+2+31+24+2+59
=208tracks

Examples of Disk Scheduling Algorithms

Work Queue: 23, 89, 132, 42, 187. There are 200 cylinders numbered from 0 - 199 and the disk head starts at number 100

FCFS



Total time is estimated by total arm motion

= |100 – 23| + |23 – 89| + |89 – 132| + |132 – 42| + |42 – 187|

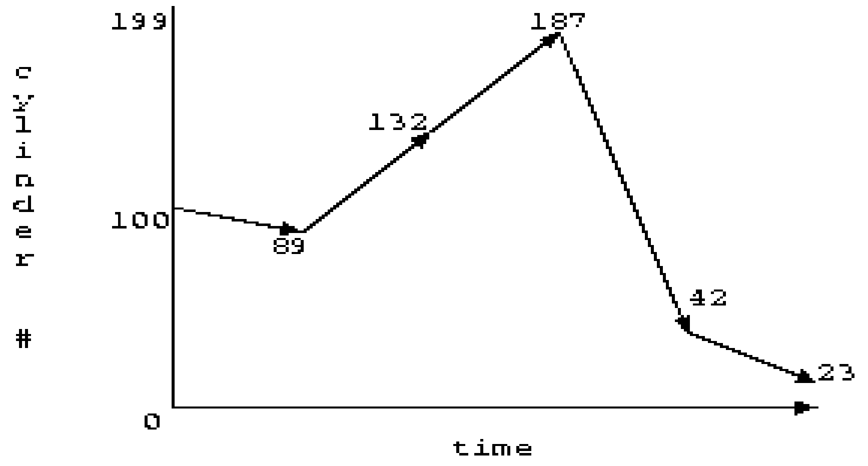
= 77 + 66 + 43 + 90 + 145

= 421

Average time = total time / number of seek

= 421 / 5 = 84.20

SSTF



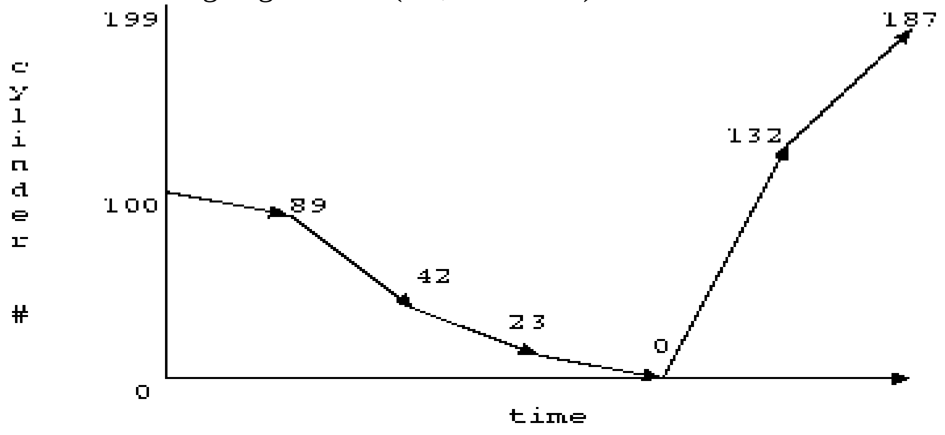
Total time is estimated by total arm motion

= |100 – 89| + |89 – 132| + |132 – 187| + |187 – 42| + |42 – 23|

= 11 + 43 + 55 + 145 + 19

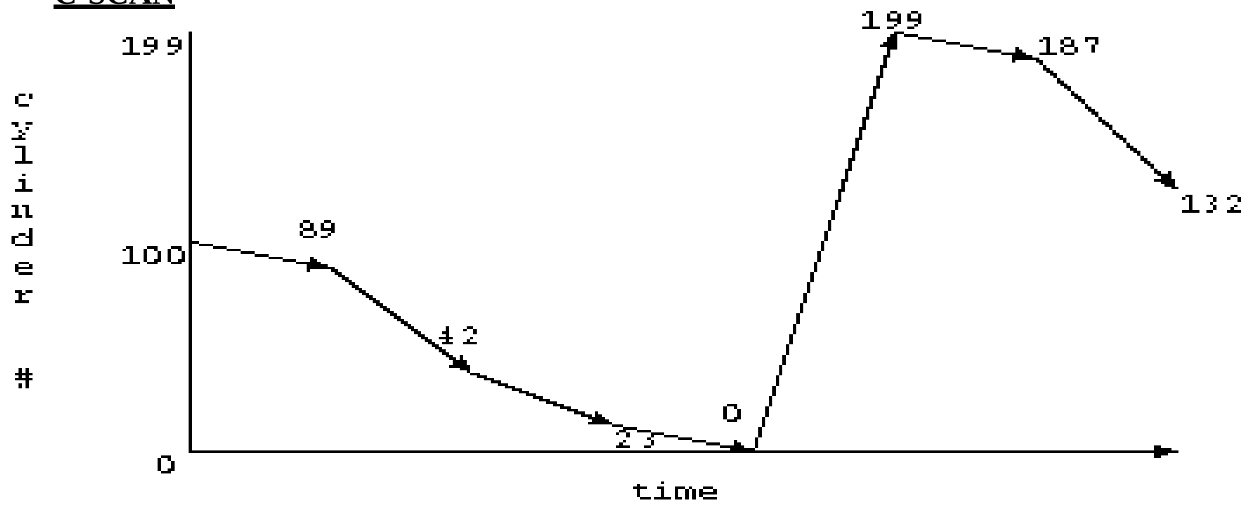
= 273

SCAN: assume we are going inwards (i.e., towards 0)



Total time is estimated by total arm motion
= |100 – 89|+|89 – 42| + |42 – 23| + |23 – 0| + |0 – 132 + |132 – 187|
= 11 + 47 + 19 + 23 + 132 + 55
= 287

C-SCAN



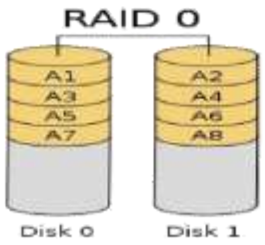
Total time is estimated by total arm motion
= |100 – 89|+|89 – 42| + |42 – 23| + |23 – 0| + |0 – 199 + |199 – 187| + |187 – 132|
= 11 + 47 + 19 + 23 + 199 + 12 + 55
= 366

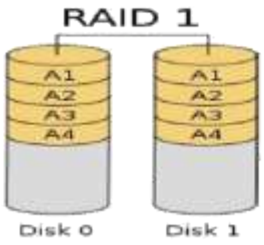
8. RAID

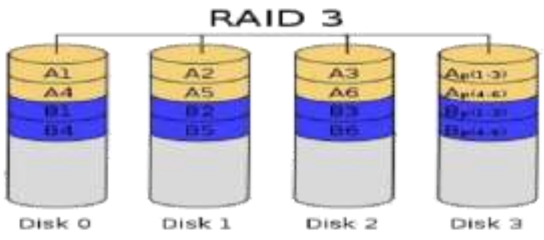
RAID (**redundant array of independent disks**, originally **redundant array of inexpensive disks)** is a storage technology that combines multiple disk drive components into a logical unit. Data is distributed across the drives in one of several ways called "RAID levels", depending on what level of redundancy and performance (via parallel communication) is required.

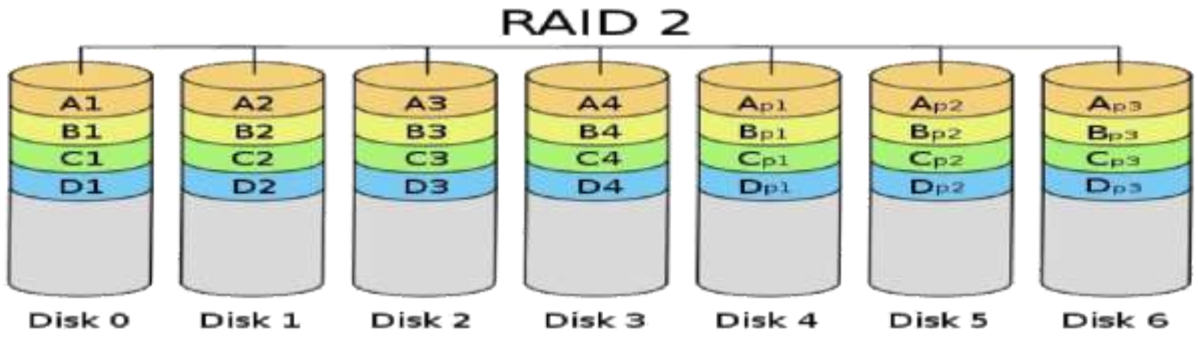
RAID Levels:

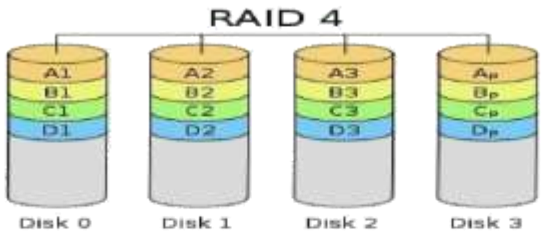
RAID 0 (block-level striping without parity or mirroring) has no (or zero) redundancy. It provides improved performance and additional storage but no fault tolerance. Hence simple stripe sets are normally referred to as RAID 0. Any drive failure destroys the array, and the likelihood of failure increases with more drives in the array (at a minimum, catastrophic data loss is almost twice as likely compared to single drives without RAID). A single drive failure destroys the entire array because when data is written to a RAID 0 volume, the data is broken into fragments called blocks. The number of blocks is dictated by the stripe size, which is a configuration parameter of the array. The blocks are written to their respective drives simultaneously on the same sector. This allows smaller sections of the entire chunk of data to be read off each drive in parallel, increasing bandwidth. RAID 0 does not implement error checking, so any error is uncorrectable. More drives in the array means higher bandwidth, but greater risk of data loss.

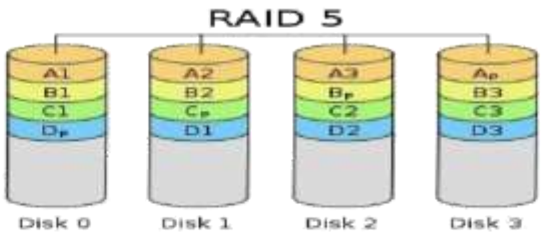












In **RAID 1** (mirroring without parity or striping), data is written identically to multiple drives, thereby producing a "mirrored set"; at least 2 drives are required to constitute such an array. While more constituent drives may be employed, many implementations deal with a maximum of only 2; of course, it might be possible to use such a limited level 1 RAID itself as a constituent of a level 1 RAID, effectively masking the limitation.The array continues to operate as long as at least one drive is functioning. With appropriate operating system support, there can be increased read performance, and only a minimal write performance reduction; implementing RAID 1 with a separate controller for each drive in order to perform simultaneous reads (and writes) is sometimes called multiplexing (or duplexing when there are only 2 drives).

In **RAID 2** (bit-level striping with dedicated Hamming- code parity), all disk spindle rotation is synchronized, and data is striped such that each sequential bit is on a different drive. Hamming-code parity is calculated across corresponding bits and stored on at least one parity drive.

In **RAID 3** (byte-level striping with dedicated parity), all disk spindle rotation is synchronized, and data is striped so each sequential byte is on a different drive. Parity is calculated across corresponding bytes and stored on a dedicated parity drive.

RAID 4 (block-level striping with dedicated parity) is identical to RAID 5 (see below), but confines all parity data to a single drive. In this setup, files may be distributed between multiple drives. Each drive operates independently, allowing I/O requests to be performed in parallel. However, the use of a dedicated parity drive could create a performance bottleneck; because the parity data must be written to a single, dedicated parity drive for each block of non-parity data, the overall write performance may depend a great deal on the performance of this parity drive.

RAID 5 (block-level striping with distributed parity) distributes parity along with the data and requires all drives but one to be present to operate; the array is not destroyed by a single drive failure. Upon drive failure, any subsequent reads can be calculated from the distributed parity such that the drive failure is masked from the end user. However, a single drive failure results in reduced performance of the entire array until the failed drive has been replaced and the associated data rebuilt. Additionally, there is the potentially disastrous RAID 5 write hole. RAID 5 requires at least 3 disks.

RAID 6 (block-level striping with double distributed parity) provides fault tolerance of two drive failures; the array continues to operate with up to two failed drives. This makes larger RAID groups more practical, especially for high-availability systems. This becomes increasingly important as large-capacity drives lengthen the time needed to recover from the failure of a single drive. Single-parity RAID levels are as vulnerable to data loss as a RAID 0 array until the failed drive is replaced and its data rebuilt; the larger the drive, the longer the rebuild takes. Double parity gives additional time to rebuild the array without the data being at risk if a single additional drive fails before the rebuild is complete.