Chapter 7

DEVICE MANAGEMENT

In this Chapter

A Big Challenge Ahead

Here we talk about the greatest challenge faced by the OS, the management of hardware devices. Have you ever thought that all the devices connected to your PC are not the same. They are different in every aspect. They use different hardware. They serve different purposes. So if a user interacts with any such device, how can we bridge all such differences? The answer is simple: the OS helps us. But still, how?

Devices are different because of the following factors:

1. Data processing speed
2. Data volume
3. Their purpose
4. Direction of flow of information (some are input, some are output, still others are both input and output)
5. Protocols

The challenge is to deal with all this large number of devices in parallel, at the same time, and safeguard the user from the complex hardware details. Each device needs different type of support from the OS, so OS has to satisfy its needs. The hardware devices have their own timings and they do not depend upon the CPU for their operations. The OS runs on (mostly single) CPU, and it must deal with all such devices and listen for their parallel and shared use.

The OS not only deals with all these issues, it also presents an abstraction to the users and the applications. By abstraction we mean a generalized interface used by applications for interacting with devices so that the applications need not worry about the type and nature of any device. For example, it is a common practice that we use same copy/paste commands on Flash memory or on the hard disk, to do our intended task. Both devices are different, as they rely upon different data saving technologies, but it is the OS which translates the commands to reasonable device operation, according to the device in use. That's abstraction: the physical characteristics of devices are different, but they are managed and interfaced in a common way, thus a convenient environment for applications and users is maintained. It is also called device independence.

Many services provided by the OS, other than application programs, also rely upon device management. For example, the complete file management is built upon abstraction of I/O system. This I/O system management provides great ease in implementing a powerful file system. Swapping is dependent upon I/O and file management on disk.

Hardware I/O Organization

All small or big computers work primarily according to a single scheme. Only the minor details of this scheme are different from one computer to another.

Following devices are connected together by means of buses:

1. CPU
2. Memory
3. I/O devices

A bus is just like your back bone. It delivers data from one device to another. Physically it is a set of wires; a wire is called a line.

In such a scheme, the simplest application employs a single bus and the CPU, Memory and I/O devices are connected to it.

This scheme allows only any two devices to communicate at a time, otherwise there will be data collision. To ensure that there is no collision, a protocol is needed. This protocol restricts that no more than two devices would send data concurrently.

Working of the Collision Control Protocol

Such a protocol decides when two devices would communicate. The time is divided into small units, called clock cycles. *A cycle is the time needed to send one piece of information through the bus.* To control this mechanism, a device named **bus arbitor** is used. This device controls the allocation of devices and clock cycles to the bus. The device which acquires the clock cycle would communicate to any device if it is connected to the bus. To identify each device in the system, an addressing system must be used.

Normally the bus is allocated between memory and the CPU. They need the bus much frequently and their data transfer speed is also high. This increases the bus utilization and the bus goes less idle.

I/O devices need bus less frequently, but their need is much time sensitive and must be fulfilled immediately, so their requests get higher priority. *When a cycle is taken away from the CPU and is given to any other device, we call it* ***cycle stealing***.

Multi Bus Scheme

In other schemes, multiple buses are used. This increases parallel and concurrent access and improves performance, because it allows multiple communications to occur at the same time. E.g. CPU can talk to a device using a bus while the disk can send its data to memory via another bus.

But multiple buses are not much useful. Majority of the communications occurring in computer happen between CPU/memory and a device. If proper hardware to support multi-access is not used, concurrent communication with many devices cannot be achieved.

If proper multi-access hardware to support concurrent communication between many devices is not available, then multi buses alone cannot be used efficiently. Majority of the device communications involve either the memory or the CPU. They can talk with a single device at a time. If the memory or CPU is already in communication with a device, it will not entertain any new device.

Multiple buses are used for performance tuning. There are at least four buses in a modern PC.

a. Processor bus
b. PCI Bus (Peripheral component interconnect), it is the primary bus to connect devices.
c. Memory Bus, this bus is used between memory and CPU and its purpose is to improve data transmission between them.
d. ISA, it is connected to the PCI and provides backward compatibility for old ISA devices. When a new technology is able to work with older technologies, we call it backward compatible.

I/O Control

Here we see how the hardware is controlled by the CPU. There are there schemes for it.

1. This is the simplest scheme. In this scheme, only CPU and the communicating I/O device are involved. The CPU interacts with the I/O device directly. All device operations are controlled by the CPU. This scheme is very limited and is found in only embedded systems which are controlled by the CPU.
2. The second scheme is used in PCs. In this scheme, a device, called **device controller**, is introduced. *The device controller issues the needed detailed instructions on behalf of the CPU to carry out the required operation.* The CPU communicates with the device controller, which communicates with the requested device. In this way the complexity of the device is hidden from the CPU and it does not spend its precious time in handling minute details of every hardware operation. Now CPU can perform other processing tasks. Every device has its own type of controller, and the controller controls the operations of its device. Some specialized controllers are can service more than one types of devices.
3. In the more specialized systems, CPU, I/O channel, and I/O controller work together. The CPU works with I/O channel, I/O channel communicates with the I/O controller, on the behalf of the CPU. I/O channels are much more reliable and complex than I/O controllers and sometimes they have their own CPU.I/O channels load channel programs, which are loaded into main memory or into the channel memory. The communication between CPU and I/O channel occurs at an abstract and higher level . Thus the CPU remains free from managing I/O details. The I/O subsystem does this task.

The Concept of Memory Mapped I/O and Ports

There must be some communication between CPU and I/O module to perform an I/O. This communication can occur between CPU and a device, its Controller or I/O channel. An I/O module uses registers for its working. The number of registers varies from device to device. The communication between module and CPU is done by using these registers. When CPU needs to send a command to an I/O module, it writes in the registers, and module reads it. When the module needs to send a signal to the CPU, it writes it in the registers and CPU reads it. So actually they act as a mailbox. These registers contain some special values configurable by the CPU or the I/O module.

There are two methods for the CPU to access module registers:

a. Port I/O
b. Memory mapped I/O

Port I/O

In this scheme, very few instructions are needed to perform an I/O. *Port numbers are a mechanism to identify an attached device.* The instruction for a device contains the port number of the I/O device, and the I/O is applied to this device.

Memory Mapped I/O

In this method, a portion of memory is reserved for every I/O module. So, an instruction accessing a portion of memory can also access an I/O module. This scheme is not good for the systems having limited address space. But, some systems like VAX have much powerful memory referencing systems, so they can use it very effectively.

In many systems either port I/O or memory mapped I/O is used. In some systems, both are used. No matter which scheme is in use, the port number or memory addresses must be linked to the I/O device to make the scheme work. There are 3 methods to do this:

Permanent Assignment

a-This can be done by the manufacturer.

Disadvantage: it is not an efficient or practical scheme. There are not many addresses available to be assigned to every manufacturer so that it would assign it to its devices.

b-A second approach is that to base addressing on the type of I/O module.

Disadvantage: If there are more than two module of same type, an address conflict will occur. A system cannot have more than one module with same address.

c- Configurable/Dynamic assignment

It means that port numbers and memory address allocation is not permanent. It is resettable. It is done either using Jumpers/DIP (Dual in-line package) switch or by using programs.

Assignment by Hardware

It is done when plug and play is working. Hardware detects and assigns an address to the device.

For Your Information: from Wikipedia

VAX was an instruction set architecture (ISA) developed by Digital Equipment Corporation (DEC) in the mid-1970s. A 32-bit complex instruction set computer(CISC) ISA, it was designed to extend or replace DEC's various Programmed Data Processor (PDP) ISAs. The VAX name was also used by DEC for a family of computer systems based on this processor architecture.

The VAX architecture's primary features were virtual addressing (for exampledemand paged virtual memory) and its orthogonal instruction set. VAX has been perceived as the quintessential CISC ISA, with its very large number of programmer-friendly addressing modes and machine instructions, highly orthogonal architecture, and instructions for complex operations such as queue insertion or deletion and polynomial evaluation.

Read More from Wikipedia

http://en.wikipedia.org/wiki/VAX

Module registers

Many registers are used to manage hardware. But the number and function of each is set and dictated by the I/O module in use.

In simplest scheme for an input device, at least two registers are used. One register contains data (the data buffer), second contains control information. The data buffer register contains the data to be moved between input device and the CPU, while the control register contains such bits which control the operation to be carried by the I/O module. These bits are set by the CPU and control the different operations including:

- Start the device
- Restart/reset the device
- Start reading some data

Sometimes, the I/O module also writes some data into control register. These bits direct the following:

- Device ready status
- Operation completion status
- Error conditions

The modules controlling complex devices need more registers. For example, the module controlling a disk needs an area where it can store the address of the required location and the amount of data to retrieve.

Protocols for Communication between I/O Module and CPU

Four different protocols are employed for communication between an I/O module and the CPU.

1. Busy Wait I/O
2. Polled I/O
3. Interrupt I/O
4. Direct Memory Access

Busy Wait I/O

This protocol works in this way: upon need, the CPU issues a command to the I/O module and sets the bits in the control register to start an operation.

If the operation is an output in nature, then the data to be output is also given.

Then the CPU executes a loop and checks the status of the device to determine when the said operation is done. When the operation is completed, the status is checked to confirm if the operation was a success.

If it was an input operation, then the data buffer is checked and data is transferred from buffers to somewhere appropriate.

This scheme is not favored and it is not efficient. The reason is simple, while the CPU is in waiting state for a progressing I/O, no other process can execute.

Polled I/O

After starting an I/O, the CPU starts executing some other processes. An interrupt timer is set which periodically interrupts the CPU. And CPU returns to check the I/O operation, frequently interrupting the executing process.

It is relatively efficient than busy wait I/O but high cost of frequent process suspension, context switching between processes and I/O management routines makes it quite an overhead.

Every device is different. Every I/O module has different needs. CPU will need to check the completion status of every device at different intervals. It is not suited for the modules requiring fast CPU response.

Interrupt I/O

In this mechanism, the CPU begins I/O operation and starts execution of another process. It does not periodically check the status of the operation. Rather, an interrupt is sent to the CPU at the completion of the operation. This interrupt is also a binary code stored in a special register. The overhead is greatly reduced and only the processing of the interrupt is involved.

Advantage: This is quite efficient technique

Disadvantage: This is a complex technique. Much careful coding is needed, otherwise *Race Condition* will occur.

An interrupt can occur when other interrupts are being processed. In this situation, much care is needed.

Let us suppose that there is a input device which needs fast response to its interrupt. While its interrupt is in progress, some other interrupts also occur, and CPU switches to service them. The delay thus caused would delay the processing of the original interrupt much longer and the input would be lost. But on the other hand, if the interrupts are processed sequentially, then a new, time critical interrupt would be delayed, as older one would be executing.

A solution to this is using priorities. When CPU is processing an interrupt, it would be interrupted only when a high priority interrupt is generated. The routines which service the interrupts would run for a short time, to avoid failures of interrupts when they occur concurrently. The OS design which is fast and responsive enough and deals naturally with all interrupts is a challenging task.

Direct Memory Access

DMA lets an I/O module to access memory directly and write or read data to/from there. In this way, it is not compulsory for the data to travel through the CPU. It is used with an interrupt system, meaning that when the requested operation is completed, the CPU is explicitly notified.

It is especially useful when large volumes of data are exchanged, e.g. in an operation involving the disk.

Software I/O Organization

The running processes use SVCs to request services from OS. These services include performing I/O. For the sake of abstraction, many operations on different devices are performed using same functions/primitives. Since files are much like I/O devices in many aspects, so the primitives used for I/O are mostly those used with files. Such primitives promote device independent I/O.

Device independent I/O can be achieved to a limit. Many devices are much different from others. For such devices, device dependent I/O is needed.

On UNIX, a command *ioctl* (IO control) implements device independent I/O. This command can be applied to many devices, and depending upon the device in use, it can do device relevant actions. For example, it can eject the CD drive tray or it can set the data transfer rate (baud rate) of a serial line.

Many factors affect the type and nature of SVCs provided to the process to interface with the OS.

1. Device Speed
2. Mutual Exclusion
3. Unidirectional Devices

Device Speed

Device I/O is done in a very slow manner. The devices are of two types namely blocking and nonblocking.

On few OS, the non-blocking I/O is also supported. An OS provides two methods to handle nonblocking I/O.

1. First is to store/buffer data temporarily somewhere ion memory. And when the system call occurs, this data is immediately transferred. Such a buffer is called input buffer. The return value of such a primitive tells how much data has been transferred. If 0 is returned, then it means that no data was transferred.
2. Secondly, after completion of an I/O operation, the process which requested the I/O is notified. This notification can be done in three ways:
    i. A signal can be sent to tell the process about the completing I/O
    ii. A variable in the process can be set to a specific value to represent status of I/O operation.
    iii. A system call can also be provided which would check the status of the operation. This call will be executed by the process.

Mutual Exclusion

Majority of the devices connected to a computer are not sharable. What the OS will do when many requests to use such a device arrive concurrently. OS has four options to manage such a situation:

1. To implement a policy where only one process can use a device at a time.
2. The membership of the device can be assigned to a user who opened it and all other accesses to it can be denied. But the owner may allow other users to access the device, thus ignoring the default setting, a phenomena known as overriding.
3. At the application level, such a problem can be avoided using a spooling system. Some other alternatives are also available.
4. The OS may do nothing, and the users would resolve the conflict themselves.

Unidirectional Devices

Most devices are either input or output. An output device cannot do input and vice versa. It lies on the shoulders of OS to avoid issuing a command on a device which it cannot handle. Mostly a device is opened and then it can be used. The command to open a device specifies which type of access is desired and how the device will be used. So, if a system call specifies an inappropriate type of operation while opening a device, the attempt will fail.

Network I/O

Now the computers are increasingly connected together in a form of network. It allows them to communicate, but adds much more complexity. All the communications over a network are done by following complex protocols. If the compliance with such protocols fails, then the whole communication will fail. So the OS must support the needed protocols. The OS provides network applications with network I/O system calls.

Abstraction is also implemented here.

In networks, the most common abstract concept is socket.

At least two computers are involved in network I/O, one is local, meanings where user is situated, second is remote, the far away computer. Socket is two sided jack on the network, on one side of which is local computer plugged into the socket. On the other side of the socket, the remote computer is plugged. The network system calls are used to:

a. Create socket
b. Connect sockets on remote computers
c. To check if a remote computer has connected to a local socket
d. Send messages using sockets
e. Receive messages through sockets

Logical I/O

At this level of device management, the OS attempts to implement a scheme which is device independent. It is done by implementing device independent kernel I/O objects.

However, these objects are not totally device independent. Rather, they are semi device independent objects. The OS also provides few classes and libraries which are particularly written to handle specific devices. If some other module of the OS needs some capability of an I/O device, then at this level it will be provided.

There are two types of I/O devices: blocking and non-blocking.

Blocking: These are the devices which exchange data of a certain size. The data of such a specific size is termed as a block. Most commonly used block device is disk, which takes a sector as a block. Any I/O operations on a disk are applied in terms of blocks or sectors.

Non-Blocking Devices: They transfer data character by character. For example modems and printers.

The I/O objects also contain some information. This information is used to execute or control a device in use. The information about a device contained in a kernel I/O object includes:

1. Is the device in use is blocking or it is non-blocking (type of device).
2. Which operation is to be performed (Read or write operation)
3. What is the status of an operation sent to a device? There can be following status of a directed operation:
    a. Pending, the operation has not started yet.
    b. In progress, the operation has started but not completed yet
    c. Done, the operation has completed either successfully or it has failed.
4. If an error occurred, then what was its type? (Error status)
5. Which device is to be used for the directed operation
6. In case of an input operation, what is the logical location to store input data and from where it will be obtained?
7. In case of output operation, where the output will be transferred so that the output device will receive it.
8. Total number of bytes to be transferred

9. The actual number of bytes which have been transferred
10. Device scheduling information, in case it is to be shared among many processes.

If the device being used is a random device, then it is necessary to give the logical location for data transfer. The specified location is always a logical location/block address not the physical address on the device.

Buffering

While using logical I/O, it also gives us buffering capability needed by I/O data. Following are the uses of buffers:

- The computer has no information and control over user data input behavior. The user will start typing whenever he wants.
- Even if there is no process which requested inputs from any device, still the user can input data. In such a case the user input is stored in buffers and waits until a process requests input. Sometimes such an input is ignored by the operating system.
-
- Many I/O devices transfer data in form of blocks. Blocks are of fixed size data units, but application can request data of any size. The incoming data is placed in buffers and they are packed into blocks of appropriate sizes.
-
- While DMA is in use, all input and output is transferred to buffer. Input device put their data in input buffers and the CPU reads it from there. For output, the CPU places its data in output buffers and an output device reads. Such a scheme improves data transfer operation integrity and quality.

Concept of Caching the I/O Data

Cache is a storage device/location which stores data on behalf of a slow I/O device. It stores I/O data of an I/O device and I/O request for that device are satisfied using this cache. I/O caches are mostly created in memory. It is quite fast in itself. The I/O operations on the slower device are performed using cache, so the process speeds up.

If a device is using cache, then its performance is much improved and boosted up than a device not using the cache. Cache has been found to be useful in many cases.

The scheme works by creating buffer pools. The size of such buffer is kept larger than actually needed, and becomes a cache. Such caches also need more information to be used effectively. For example, a record of locations form device copied to cache must be kept. So all operations to a location are carried out using the cache.

When a write operation is in progress, one out of two methods can be used.

1. Write through
2. Delayed write

Write Through

When some information is available in buffers, it is saved on disk immediately. As the block devices operate in blocks, when a write through operation is initiated on such a device, the function does not return until the write is done for the complete block.

Delayed Write

In this scheme, the OS delays the saving of the data to the disk until some future time. If a blocking device is in use, the operation will not return until all the required data is cached.

The delayed write scheme is quite efficient, but there is a threat of data loss until the buffers has been saved. If the power is cut off, then the data in the buffers will be lost and the data already on disk would become inconsistent. That’s why there users are asked not to just power off the system, instead the system should be properly shut down using OS commands. It would transfer the data in the buffers to the disk, and the threat of data loss will be eliminated. The time until which the OS will wait is found using many criteria including:

1. To wait until a specific number of buffers are full and waiting to be saved.
2. There is no pending I/O operation waiting for the device in use.
3. The process which requested the write operation has issued a close command.
4. The system is being shut down.

When all of the buffers are full, and there is still a request for caching more data, then buffers must be recycled. An algorithm selects a buffer to recycle. These algorithms are similar to those used by demand paging systems.

In some cases, some devices, including disks, have their own special memory on their controllers for caching. In such a case, if the OS is also caching the data, then it will be not be useful, rather it will have a negative impact on system performance.

Device Drivers

Device drives are a type of system software and they are used to control a specific type of device. When an I/O object in logical I/O is created, it is passed to the correct device driver. A device driver can handle one or more related devices. Every device needs a specific set of functions/operations applied to it. These operations include:

1. **open:** It means to perform some necessary tasks which are needed before a device can be given to a process.
2. **close:** The task needed to shut down a device
3. **schedule:** a device may be requested by many processes simultaneously. To deal such requests, the devices are scheduled among different processes. The device scheduling is done using logical I/O software before the requests are transferred to the device drivers. In some cases, device drivers can also implement their own scheduling mechanism.
4. **startio:** The device is checked for its current activity, if it is not busy, then a requested operation is selected from scheduling queue and started.
5. **interrupt:** the function to be performed when the device issues an interrupt to CPU.
6. **ioctl:** This is used to implement any special function that a particular device can handle.

Also, the error checking and recovery from errors is done in these routines. The I/O objects are designed to report an error to the process which initiated the device using error codes.

A PC can be connected to any device, and at a time, many devices can be connected to a single system. Every device needs its own drivers. It is impossible and impractical that an OS of a PC to contain drivers for all possible devices to be connected. For connected devices, a proper configuration is maintained. The OS only needs to include drivers for those devices which are included in the configuration.

If the configuration changes, then it results in addition or removal of the devices of the system. As a result, the OS on disk is different from the OS image running. So, many OS require restarting the computer to use the changed image. Some OS do not need to restart. They are called dynamically changeable drivers or loadable. Linux can do this. Also, Linux can configure drivers either as loadable, present or not present.

Hardware Devices

Graphics Management/Monitors

For an OS designer, it is very difficult to design a system to handle the information needed by a monitor to display an image. To make a screen, many pixels (picture elements) are used. The glowing capacity of every pixel is controlled by the OS.

In early days, the monitors were black and white, and could display text only. They displayed 200 lines of text with 320 characters in each. Now the situation is very different.

Normally monitors use either of the following resolutions:

- 800X600
- 1024X768
- 1280X960

To control the illumination of each pixel, data in Video RAM is used. On basic monochrome monitors, only one bit was needed to control a single pixel. But for colored displays, the number of bits needed is high. The color system used is RGB. These are called primary colors, and all other colors can be produced by mixing appropriate intensities of these colors. In one scheme, every color in RGB requires one byte, i.e. 8 bit color scheme. Hence the complete pixel will need 24 bits. Every color is controlled by the values between 0 and 255. Thus the intensity can vary from 0 to 255 levels.

Using the scheme described above, if the monitor is operating at the resolution of 1024X768, then the total memory needed to make one screen is 2.3Mb. However, if the system introduces some indexing mechanism, then the memory needed would reduce. In indexing, the total displayable colors are restricted to 256 in number. The complete information needed to display a pixel is stored in a single byte. In this way, the memory required reduces and for a 1024X768 display resolution, the memory needed will be less than 1 Mb. The value in the byte of pixel is used to reference the color palette, a table containing 256 colors in 24 bit value. When a pixel byte references a palette, the value is found and pixel is lit according to it.

Remaking or refreshing of screen means to modify a screen. If resolution is high, and bits needed to store color information are also high, then the amount of data needed will increase. Given the high refresh rate, the needed data will increase even more. All this will affect the demand of Video RAM. For example, while operating at refresh rate of 25Hz, 1024X768 resolution, RGB colors, the same 2.3 Mb needed to paint a screen once will result in total 58 Mb needed per second.

The hardware has also evolved and can manage these increasing needs of data transfer.

PCI bus can transfer data at the speed of 132Mb per second.

AGP can transfer data at the speed of 528Mb per second.

Today's modern graphic controllers are intelligent and they can interact with the Video RAM, accept instructions and process them. As a result, they can change the data in Video RAM. As a result the performance of the system increase as CPU does not process graphic data. But to use such a graphic controller, the driver must be accordingly designed.

Text Based Displays

A text based display can display only textual characters. Serial terminals used with old mainframes were text based. Also many printers are considered as text based displays. *The complete set of characters which a text based display can show is called **Alphabet** of the display***.** Normally monitors support 256 characters in their alphabet. Such alphabet also contain 128 ASCII characters i.e. 7 bit ASCII.

A good aspect of text based displays is that they can be refreshed at a very high speed. Every character requires one byte for its representation. The display needs only 2000 bytes to paint the screen having 25 rows and 80 columns. This is very important reduction for those users which are at low network bandwidth or facing congestion in the network. The need for reduced bandwidth allows them to carry out their operations even at the remote computers. This can happen using even a regular modem. A good example of such a case is a terminal communicating with a mainframe.

Text based displays have following features:

Line Editing

When a user starts typing on the keyboard, text is not delivered to the application as it is typed. Rather it is stored temporarily in buffers until enter key is hit. Some editing facilities are given to help the user in entering correct commands, which include:

1. To erase an entered character, this function is provided by delete or backspace key.
2. Line erase function is sometimes provided which can erase a full line.

The editing is done before a line is passed to the application. So, the application never comes to know which character

To support text based displays, the OS logical I/O subsystem is used.

END Of File:

This happens when a user program is reading data from a file. When the user program comes to the end of the file, the OS returns an EOF character. The arrival of this character at the application informs the application that there is no more data available in the file. Same EOF character is input to inform an application from a text device to tell no more data exists.

UNIX uses Ctrl+D combination as EOF.

DOS uses Ctrl+Z combination as EOF

Signaling

It is a method to ask OS to abort (exit/terminate) all the processes which are receiving input from the text device.

Mostly, either Ctrl+Y or Ctrl+C are set as signal (abort) character.

Echo

Echo is used to display a character on screen. When a character is typed, it is not displayed by itself, rather it must be *echo*ed. Applications do not echo any typed characters, rather the text interface provides this functionality automatically. But sometimes, automatic echo is not desirable, especially when entering a password.

NL/CR translation

While typing on keyboard, a move to the next new line is done when a CarriageReturn (CR) character and a NewLine (NL) character is given. For a user, the new line is indicated by a stroke of enter key. This pressing of enter generates a CR and gives it to the computer. While with many applications, the NL is used to end a line. As the circumstances would allow, a single CR or a single NL or both can be used to represent End of Line. It is the responsibility of the text based interface to configure between the representations as needed.

Case Management

May application programs and devices cannot understand difference in cases of similar characters. We say they operate in limited case environment. So, the case management is done automatically and characters are translated from lowercase to uppercase and vice versa.

Serial Control

Mostly, a text device is connected to a serial interface. Many settings can be applied to a serial interface and they are controlled by text interface. Following configurable options are mostly supported by a text interface:

1. Flow control
2. Parity bit (error control)
3. Speed (Baud rate)
4. Control of modem

Storage Disks

It is strange but it is true that a disk is considered to be an I/O device. Almost every computer has it.

A computer may or may not have printers or scanners. The servers may not have even the keyboards or the monitors. In most environments, theses computers are accessed using network interfaces.

But there are few computers without a disk. For example, an X terminal or diskless workstations. But we do not consider them a computer. They are rather ***information appliances***.

The disks are different on many grounds including:

- speed
- capacity
- type
    - magnetic
    - optical

Despite of all this, all the disks have many things in common. For example the following properties and behaviors are common to HDD, FDD and CDD, DVDs;

1. For every disk, the smallest unit of data allocation, reading and/or writing is a sector.
    a. **Size:** size of the sector is 512 bytes. A sector contains
    b. **Components**
        i. Actual data
        ii. Administrative information, to identify the sector and to redundancy check information
2. The sectors are organized on a disk. This medium is a round physical flat surface.
3. The disk is mounted on to a motor. The motor spins it.
4. There are read/ write heads in the disk drive. When the disk revolves, they either read or write data.
5. The heads can move over the revolving disk. They move from inner center to the outer edge of the disk and vice versa.
6. When a sector is to be read, the heads are moved in the appropriate direction till the needed location/track is reached. Then the disk waits until the required sector comes under the head as the disk rotates.

Specific Features of Disks

DVDs and CDs

They are different because they arrange sectors in a different way than HDD or FDD.

In them, the sectors are continued on a single track. The track starts from the center, and forming a groove, it spirals from the center till the edge. As the distance from the center increases, the disk rotation speed must be slowed down to allow the heads read it accurately.

FDDs and HDDs

On these disks, the media is kept in rotation at a constant pace. First, the media is divided in many concentric circles. We call them tracks. The sectors are organized on these tracks.

As we move away from the center to outer edge, the tracks get longer. On floppy disks, all the sectors contain same number of sectors, although the outer ones can contain more sectors than the inner ones. On such diskettes, the bit density decreases from inside to outside.

Hard disks arrange more sectors on outer tracks. So, an outer track contains more information than an inner one.

The media of floppy disks is made of plastic and it is flexible. We call it a cookie. Data is stored on both sides of it.

On hard disks, there are one or more metal platters. The data is stored on both sides of the platters. But on some disks, the top most and bottom most surface is not used to store data for performance reasons.

Whatever the scheme is, a read and a write head is needed separately for each surface to record and retrieve the data. All heads are bundled together and move together.

Cylinder is a term which defines all the tracks which come under a particular head position. *We may say the collection of all equal radius tracks on every platter of the disk form a cylinder*. The number of cylinders on a disk equal to the number of tracks on any one surface of a platter.

To find the total capacity of the disk, use the following formula:

Total capacity= (number of cylinders) X (number of read/write heads) X (number of sectors per track) X (number of bytes per track)

Measuring the Performance of Hard Disk

The total access time can be used to aggregate the disk performance factors. Disk performance is measured using seek time, latency and transfer time.

So, total access time =Seek time +*Latency + Transfer Time*

i.e. ➔ *A=S+L+T*

Seek time *is the time that is needed to take the head to the appropriate track/cylinder. It is affected by*

1. ***Initialization time****, I, the time needed to startup and initiate head's movement.*
2. ***Head Speed,*** *H*
3. ***Distance to be covered by the head,*** *C*

We write it as

➔ *S=HC+I*

Latency

The time needed to bring the first location containing the required information under the head while spinning the disk. It is observed that it is the half time of one revolution of the disk. To obtain it, 30 is divided by the revolutions rate, measured as RPM (revolution per minute).

Latency=30/revolutions per minute (RPM)

➔ *L=30/R*

Transfer Time

It is the time to transfer the information to memory. It is calculated by considering

1. the number of bytes to read, B
2. the number of bytes per track

3. the revolutions per minute (RPM)

Transfer time=(60*bytes to transfer)/(revolutions per Minute*number of bytes per track)

➔T=60B/RN

Hard disk Scheduling Algorithms

When multiple requests come to read data from the hard disk, we need some decision making system to determine which request should be satisfied first. Also, the time to move the head to a specific area can affect the performance. For such a case, we also need a disk scheduling scheme.

Following is the list of algorithms used in disk scheduling:

1. FIFO
2. Priority
3. SSTF
4. SCAN
5. C-SCAN
6. LOOK
7. C-LOOK
8. N- Step SCAN
9. F-SCAN

FIFO

When a request for harddisk arrives, it is stored along with its arrival time. Then from the pool of these requests, one with the oldest arrival time is selected. In other words, requests are processed in the order they arrive.

Priority

Every process requesting disk is allocated a priority number. When the disk becomes available, the highest priority request is selected.

SSTF

It stands for shortest seek time first.

The controller finds the current location of the head and calculates the distance to the track from where to fetch the data. The request for the closest track is satisfied first.

We can say it is the priority scheduling with seek time taken as priority criteria.

SCAN

The head can move from innermost track to outer most and vice versa. While moving in one direction, the head fulfills all read/write requests for the track under it. It passes over each track and its requests are satisfied.

Sometimes starvation occurs, in case there are many requests for a single track.

LOOK

The head starts moving in one direction and satisfies all the data requests for the tracks in the way. When there are no more requests for tracks in that direction, the head goes back. It avoids useless movement of the head between innermost and outermost tracks.

C-SCAN

It is similar to SCAN. But when the innermost/outermost track is reached, it returns to start position without reading/writing any data. Doing so minimizes delays while fulfilling a request.

C-LOOK

It is similar to LOOK. But when all requests on tracks in one direction are satisfied, it returns to the start position without reading/writing any data. Doing so minimizes delays while fulfilling a request.

N-Step SCAN

This scheme does not allow starvation to occur. There are many sub-queues within requests queue. Every sub-queue can contain N requests. To select a sub-queue, FIFO is applied. After selecting a subqueue, selection of requests is made on the basis of SCAN.

If a request comes during a subqueue is being fulfilled, it will be placed in next non-filled queue.

FSCAN

It is similar to N- step SCAN. But there are only two subqueues. Every subqueue can contain indefinite number of requests. While a queue is being serviced and there is an incoming request, it will be placed in the next queue.

To improve performance, the algorithms are based on seek time. We do not use latency time as a basis because it is not always possible to determine as rotating location is hard to find. But if there are multiple requests about a single track, they can be served based upon their latency time.

Disk Geometry

The number of sectors per track, total cylinders, number of heads i.e. surfaces is called disk geometry.

Geometry also plays a very important role. Many algorithms designed to improve disk performance cannot be implemented if we do not know the geometry. In many cases, disk does not present its actual geometry due to OS restrictions. Rather it presents a different geometry. For example, using DOS, a disk should:

1. not have heads more than 256
2. not have sectors more than 64 per track
3. cylinders should be less than 1024

So the disk will present a compatible virtual geometry and will hide its real geometry from the OS.

Formatting the Disk

We cannot write data directly onto a disk in raw form. Rather, we must perform some processes to prepare the disk to hold data. This preparation is called formatting. In this process the sectors and tracks are defined and a file system s created.

Formatting is of two types:

- Low level formatting
- High level formatting

Low level formatting: it is done at the factory by the manufacturer. This type of formatting creates sectors on the disk surface.

The disks have some extra sectors available on them. In a case that there are some defective sectors, the disk does not use the defective ones. Rather a remapping is used to allocate extra sectors as a replacement for the defective ones.

At the level above the device drivers, *the record for failed sectors is maintained. It is called bad block mechanism.* This bad sector mechanism is implemented in blocks.

Disk performance is highly affected by the placement of sectors on a track.

In some cases, the disk heads read one sector at a time. So if there are more than one sectors to read, the first sector will be read and transferred, then the controller will issue a signal to read the next, it will be read and transferred, and so on. It means to read multiple sectors, several I/O operations are to be performed, thus increasing the total time needed to transfer complete requested information. Some disks do not stop and continuously revolve. If contiguous sequential sectors are to be read, then performance problems can arise. While transferring a sector the next moves away due to disk rotation. It will delay the transfer of all sectors. Now there must be a wait until the rotation of disk brings the needed sector again under the head. It can be a full rotation time.

InterLeaving

To handle such a situation, the sectors are interleaved. In this mechanism, the sectors are not continuously organized. Sector 2 is not placed after sector 1, rather it can be after sector 5.

The distance covered by the head when a sector has been read and request for the next sector is issued determines the distance between sectors. The interleaving should be organized so that when the next command comes from the controller, the next sector should be under the head as the disk rotates.

Modern hard disks controllers do not require interleaving. For performance reasons, they are built with enough memory. So when a request to perform a read operation is generated for a sector, the complete track containing the sector is read. Practically, interleaving is done on floppy disks.

On many OS, the concept of partitions is supported. DOS, UNIX, Windows have common scheme of portioning a disk. So it is possible for them to be present on the same disk.

High Level Formatting/Logical Format

In this formatting, an empty file system is created on disk. It creates many data structures needed to store files. Without this, a file cannot be stored. But it is not always needed, especially on some OS, applications are allowed to save data directly. Such a directly accessible disk device acts as a large collection of storage blocks in a sequence. When applications are allowed to work in this way, the responsibility of ordering and structuring of data files and other objects on the disk lies with the application.

Redundant Array of Inexpensive Disks

It is abbreviated as RAID. It is a technology to arrange many harddisks together in a group to get a much more reliable storage space. The RAID can be implemented by separate hardware or the OS.

There are six types of RAIDs defined so far. Every type is termed as a level, starting from level 0 to level 5

RAID Level 0

In this level, many individual disks are combined into one large virtual disk. The storage space is divided into storage units called ***Strips***. The size of a strip is a multiple of 512 bytes, i.e. it is the multiple of sector size. Sometimes one strip is equal to one sector size. *The large virtual disk is a large collection of strips organized on physical disks in a sequence*.

Advantages/Disadvantages

The advantages are limited. Its only advantage is the creation of one large virtual disk. This disk can be used as a single high space disk.

This scheme distributes a file over many physical disks. So, if a disk failure occurs, complete data cannot be recovered.

But, this scheme offers ***pipleining***. In pipelining, the sequential files are accessed faster than normal. *Since a sequential file is distributed on many strips, if the first strip of a file is being read then at the same time, the second strip can also be read and transferred*. If the RAID contains N disks, then N I/O operations can be done simultaneously in this way.

RAID Level 1

In this level, there are two copies of every strip. Every copy is maintained on different disk. The two disks contain exact same copies of data. Read requests are optimized, they are satisfied using the copy available on fastest disk. In many cases, these requests can also be pipelined.

The situation is different when performing write operations. A single bit of information is to be saved at two places. So, it is not as efficient as read operations. When write operation is initiated, the RAID system waits until the slowest disk has done updates.

RAID Level 2

Each sequential bit is placed on a different drive. This level maintains a single copy of each strip. If the disk fails then partial recovery is possible.

The strips are kept too small so many strips can be accessed in parallel.

This scheme uses an error detection and correction code. Hamming code is one such example. This code is used to detect and correct errors in the data. Its implementation involves the calculation of code against all corresponding bits from every disk.

RAID Level 3

It is much similar to the RAID level 2. But it does not use a complex code. Rather it uses parity bits. It is a single bit maintained against each set of corresponding bits of each strip. It needs only one extra disk to store all parity bits. If a disk suffers from failure, then its data can be recovered using other disks.

RAID Level 4

It also works in the way similar to the RAID level 3, but the strips are large. A single operation can be performed using a single disk. While performing write operations, parity is also calculated and saved as well as data. If the number of write requests is large, then the calculation and saving of parity bits present much difficulty for the systems, as all parity bits are found on a single disk and all of them must be updated.

RAID Level 5

This level also uses parity bits and the size of strips is also large. But the parity bits are distributed over many disks. So, if there are many write requested operations then the performance issues found in RAID level 4 do not occur.

The concept of RAM Disks

It is created from system RAM. It is a virtual block device. To work with read and write operations in blocks, the RAM disk device driver is used.

Advantages

The RAM disk is actually a portion of RAM. It is not a separate device. Disk like behavior is provided using its driver. Since RAM is direct access, and does not involve heads or platters, so there are no seek and rotational delays. The speed of read/write operations is fast on RAM disks.

They are much useful if we need to store:

1. Temporary files
2. Frequently accessed files
3. Very small files

Disadvantages

1. **Cost**: They are very expensive. The system RAM is utilized for RAM disk. So, the memory to be used by processes is cut short. Or a memory module must be purchased and added to increase the system RAM, to make RAM disks implementable.
2. **Volatility:** Since memory is volatile, if the power is lost then RAM disk’s contents are also lost.
3. **Remaking of File System at system startup:** When the system starts up, file system is created if OS needs to use it. It is done every time the system restarts.
4. **Potential loss of data:** If there is some unsaved data on RAM disk and power goes off, then it will be lost.

Implementation of RAM Disk

It is a simple one. The RAM disk is created by selecting a contiguous portion of RAM bytes. If the portion to be used is large enough, then its selection and allocation as RAM disk is quite simple at the system boot. Obviously, keeping the memory management schemes in mind, it is much difficult to create a large RAM disk when there are already running processes on system and they are occupying the RAM. So, some OS require that RAM disk should be created at startup time, while some can create it dynamically as needed during their operations.