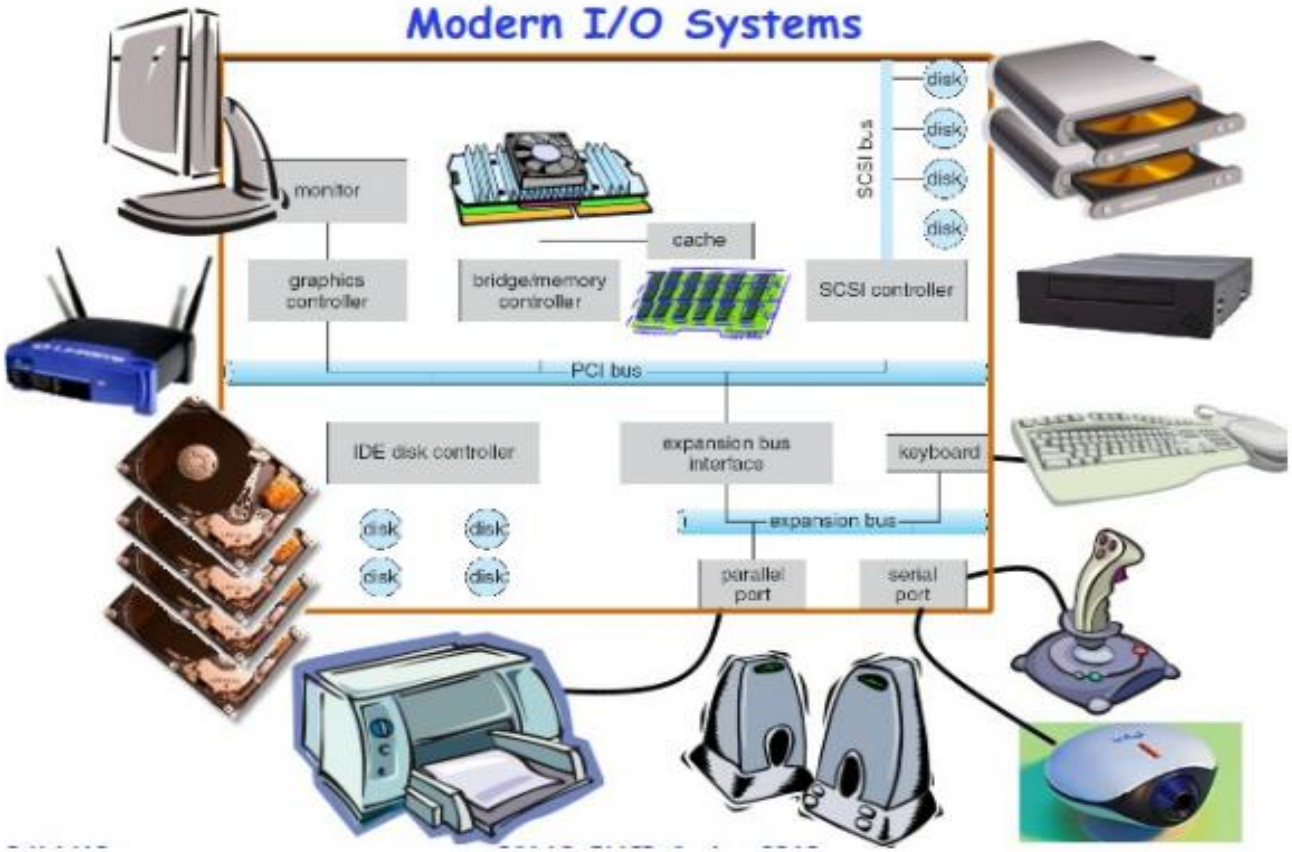
BCA
Fourth Semester
" Operating System"



What about I/O?

- Without I/O, computers are useless (disembodied brains?)
- But… thousands of devices, each slightly different
- How can we standardize the interfaces to these devices?
- Devices unreliable: media failures and transmission errors
- How can we make them reliable???
- Devices unpredictable and/or slow
- How can we manage them if we don't know what they will do or how they will perform?

Some operational parameters:

Byte/Block
Some devices provide single byte at a time (*e.g.* keyboard)
Others provide whole blocks (*e.g.* disks, networks, etc)
Sequential/Random
Some devices must be accessed sequentially (*e.g.* tape)
Others can be accessed randomly (*e.g.* disk, cd, etc.)

Polling/Interrupts
Some devices require continual monitoring
Others generate interrupts when they need service

I/O devices can be roughly divided into two categories: **block devices and character devices**. A block device is one that stores information in fixed-size blocks, each one with its own address. Common block sizes range from 512 bytes to 32,768 bytes. The essential property of a block device is that it is possible to read or write each block independently of all the other ones. Disks are the most common block devices.

The other type of I/O device is the **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. Printers, network interfaces, mice (for pointing), rats (for psychology lab experiments), and most other devices that are not disk-like can be seen as character devices.

Block Devices: e.g. disk drives, tape drives, DVD-ROM
Access blocks of data
Commands include open() , read() , write() , seek()
Raw I/O or file-system access
Memory-mapped file access possible

Character Devices: e.g. keyboards, mice, serial ports, some USB devices
Single characters at a time
Commands include get() , put()
Libraries layered on top allow line editing

Network Devices: e.g. Ethernet, Wireless, Bluetooth
Different enough from block/character to have own interface
Unix and Windows include socket interface
Separates network protocol from network operation
Includes select() functionality
Usage: pipes, FIFOs, streams, queues, mailboxes

Device Controllers:

A device controller is a hardware unit which is attached with the input/output bus of the computer and provides a hardware interface between the computer and the input/output devices. On one side it knows how to communicate with input/output devices and on the other side it knows how to communicate with the computer system though input/output bus. A device controller usually can control several input/output devices.

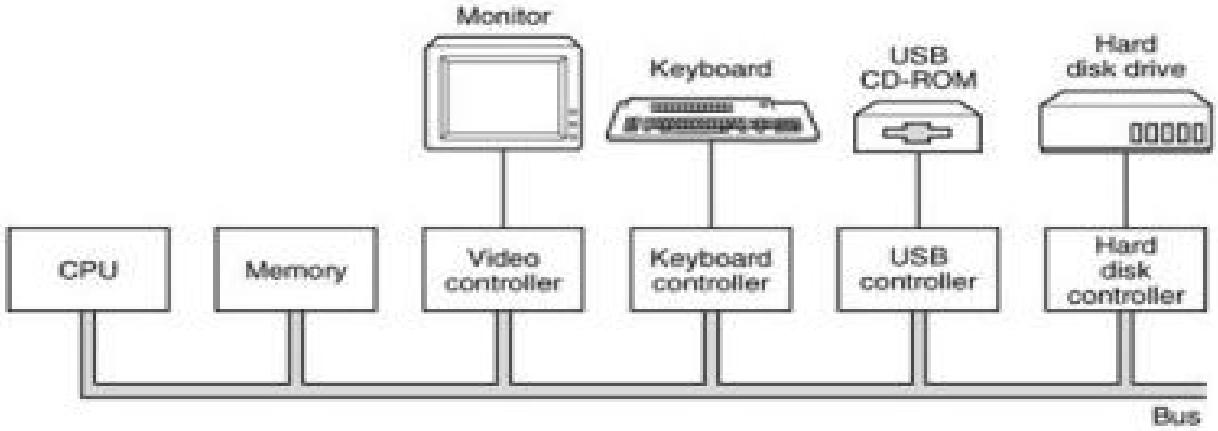


Fig: A model for connecting the CPU, memory, controllers, and I/O devices

Typically the controller is on a card (eg. LAN card, USB card etc). Device Controller play an important role in order to operate that device. It's just like a bridge between device and operating system.

Most controllers have DMA(Direct Memory Access) capability, that means they can directly read/write memory in the system. A controller without DMA capability provide or accept the data, one byte or word at a time; and the processor takes care of storing it, in memory or reading it from the memory. DMA controllers can transfer data much faster than non-DMA controllers. Presently all controllers have DMA capability.

DMA is a memory-to-device communication method that by passes the CPU.

Memory-mapped Input/Output:

Each controller has a few 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.
In addition to the control registers, many devices have a data buffer that the operating system can read and write. For example, a common way for computers to display pixels on the screen is to have a video RAM, which is basically just a data buffer, available for programs or the operating system to write into.

There are two alternatives that the CPU communicates with the control registers and the device data buffers.

Port-mapped I/O :

each control register is assigned an I/O port number, an 8- or 16-bit integer. Using a special I/O instruction such as

IN REG,PORT

the CPU can read in control register PORT and store the result in CPU register REG. Similarly, using

OUT PORT,REG

the CPU can write the contents of REG to a control register. Most early computers, including nearly all mainframes, such as the IBM 360 and all of its successors, worked this way.

In this scheme, the address spaces for memory and I/O are different, as shown in Fig. (a).Port-mapped I/O uses a special class of CPU instructions specifically for performing I/O.

In this scheme, the address spaces for memory and I/O are different, as shown in Fig. (a).Port-mapped I/O uses a special class of CPU instructions specifically for performing I/O.

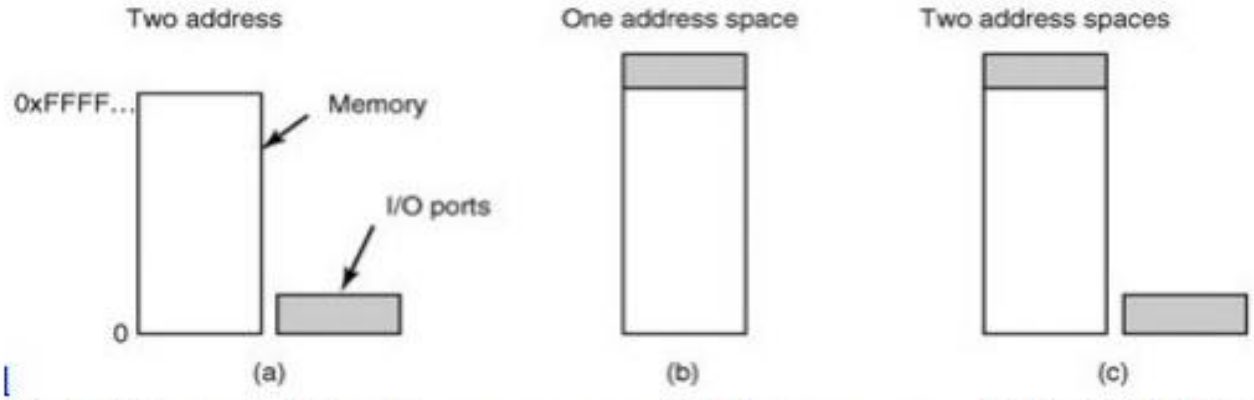


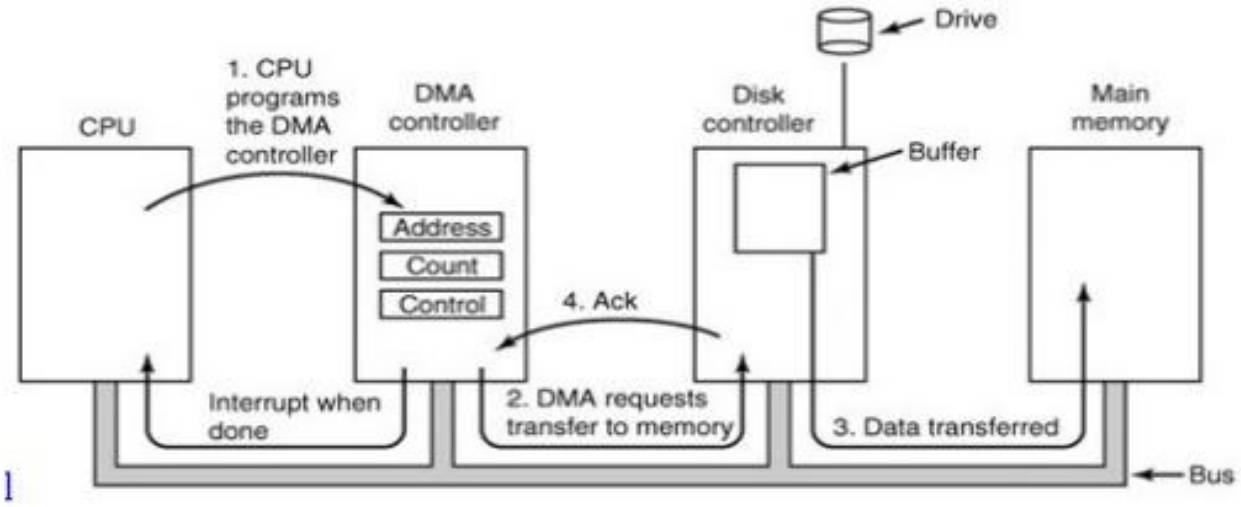
Fig:(a) Separate I/O and memory space. (b) Memory-mapped I/O. (c) Hybrid.

On other computers, I/O registers are part of the regular memory address space, as shown in Fig.(b). This scheme is called memory-mapped I/O, and was introduced with the PDP-11 minicomputer.Memory-mapped I/O (not to be confused with memory-mapped file I/O) uses the same address bus to address both memory and I/O devices, and the CPU instructions used to access the memory are also used for accessing devices. In order to accommodate the I/O devices, areas of the CPU's addressable space must be reserved for I/O.

DMA: (Direct Memory Access)

Short for direct memory access, a technique for transferring data from main memory to a device without passing it through the CPU. Computers that have DMA channels can transfer data to and from devices much more quickly than computers without a DMA channel can. This is useful for making quick backups and for real-time applications.

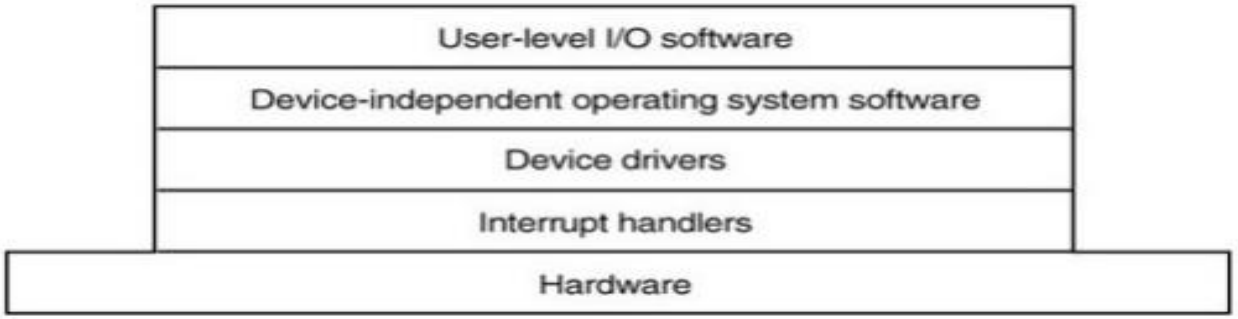
Direct Memory Access (DMA) is a method of allowing data to be moved from one location to another in a computer without intervention from the central processor (CPU).



First the CPU programs the DMA controller by setting its registers so it knows what to transfer where (step 1 in Fig.). It also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum. When valid data are in the disk controller's buffer, DMA can begin.

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, also 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 causes an interrupt. When the operating system starts up, it does not have to copy the block to memory; it is already there.

Layers of the I/O software system:



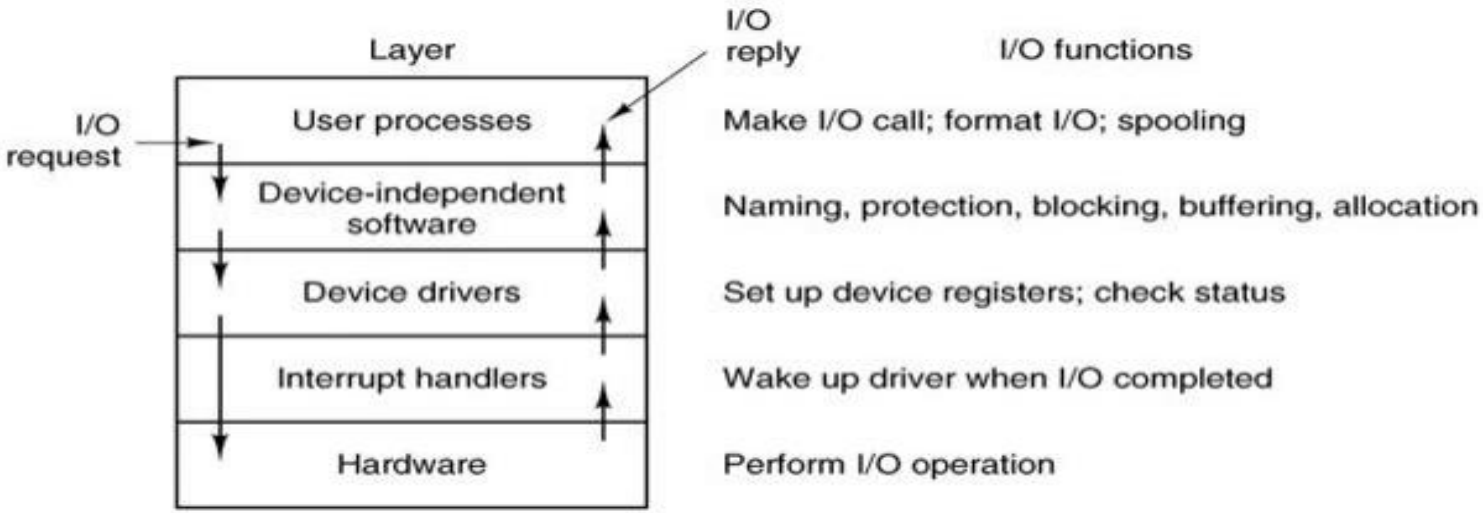


Fig. Layers of the I/O system and the main functions of each layer.

The arrows in fig above show the flow of control. When a user program tries to read a block from a file, for example, the operating system is invoked to carry out the call. The device-independent software looks for it in the buffer cache, for example. If the needed block is not there, it calls the device driver to issue the request to the hardware to go get it from the disk. The process is then blocked until the disk operation has been completed.

When the disk is finished, the hardware generates an interrupt. The interrupt handler is run to discover what has happened, that is, which device wants attention right now. It then extracts the status from the device and wakes up the sleeping process to finish off the I/O request and let the user process continue.

Device Driver:

In computing, a device driver or software driver is a computer program allowing higher-level computer programs to interact with a hardware device.

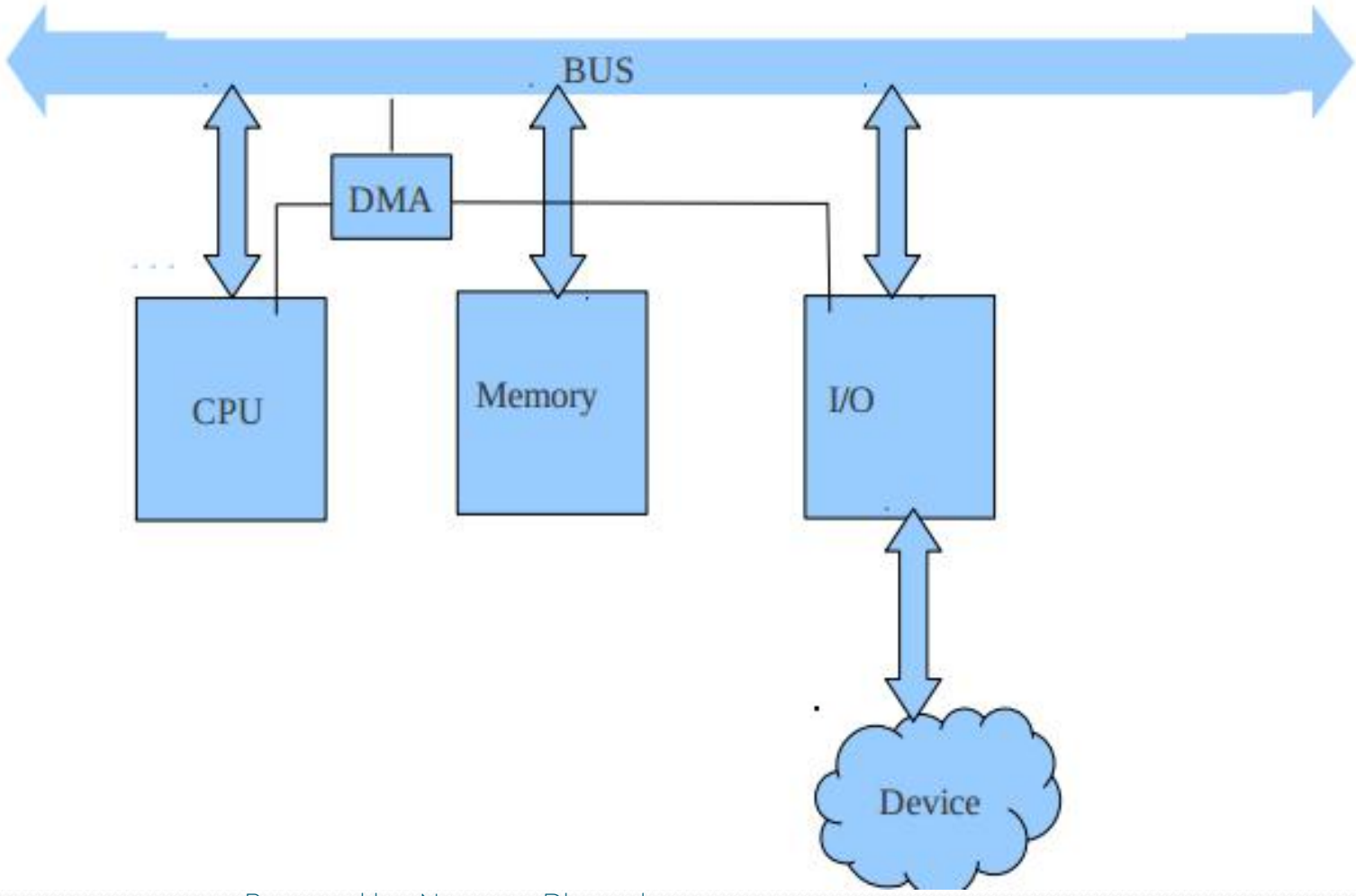
A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects. When a calling program invokes a routine in the driver, the driver issues commands to the device. Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating-system-specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface.

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. For example, a mouse driver has to accept information from the mouse telling how far it has moved and which buttons are currently depressed. In contrast, a disk driver has to know about sectors, tracks, cylinders, heads, arm motion, motor drives, head settling times, and all the other mechanics of making the disk work properly. Obviously, these drivers will be very different.

Ways to do INPUT/OUTPUT:

There are three fundamentally different ways to do I/O.

1. Programmed I/O
2. Interrupt-driven
3. Direct Memory access



Programmed I/O

The processor issues an I/O command, on behalf of a process, to an I/O module; that process then busy waits for the operation to be completed before proceeding.

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

Interrupt-driven I/O:

The problem with the programmed I/O is that the processor has to wait a long time for the input/output 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 Input/ Output module. As a result the level of performance of entire system is degraded.

An alternative approach for this is interrupt driven Input / Output. The processor issue an Input/Output command to a module and then go on to do some other useful work. The input/ Output 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 as before and then resumes its former processing. Interrupt-driven input/output still consumes a lot of time because every data has to pass with processor.

DMA:

The previous ways of I/O suffer from two inherent drawbacks.

1. The I/O transfer rate is limited by the speed with which the processor can test and service a device.
2. The processor is tied up in managing an I/O transfer;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:Direct memory access. 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 follow.

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

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

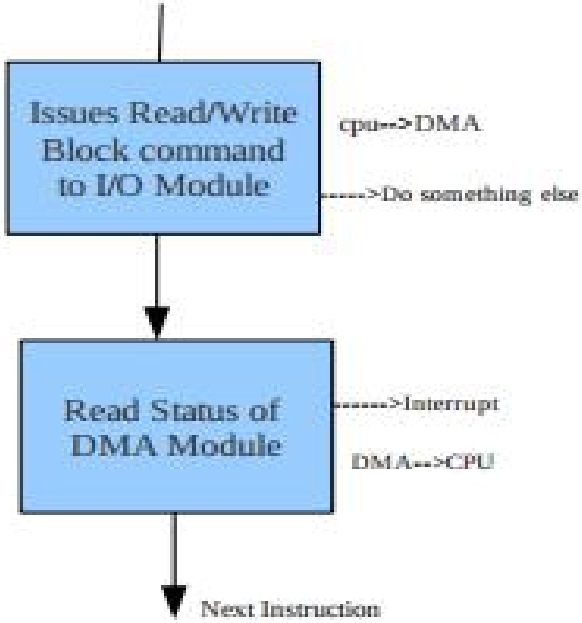


Fig:DMA

The processor then continues with other work. It has delegated this I/O operation to the DMA module, and that module will take care of it. The DMA module transfers the entire block of data, one word at at 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.

In programmed I/O cpu takes care of whether the device is ready or not. Data may be lost. Whereas in Interrupt-driven I/O, device itself inform the cpu by generating an interrupt signal. if the data rate of the i/o is too fast. Data may be lost. In this case cpu most be cut off, since cpu is too slow for the particular device. the initial state is too fast.

it is meaningful to allow the device to put the data directly to the memory. This is called DMA.

dma controller will take over the task of cpu. Cpu is general purpose but the dma controller is specific purpose.

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.

Disk Operations

Latency Time: The time taken to rotate from its current position to a position adjacent to the read-write head.

Seek: The processes of moving the arm assembly to new cylinder.

To access a particular record, first the arm assembly must be moved to the appropriate cylinder, and then rotate the disk until it is immediately under the read-write head.

The time taken to access the whole record is called *transmission time*.

Disk Scheduling

OS is responsible to use the hardware efficiently – for the disk drive this means fast seek, latency and transmission time.

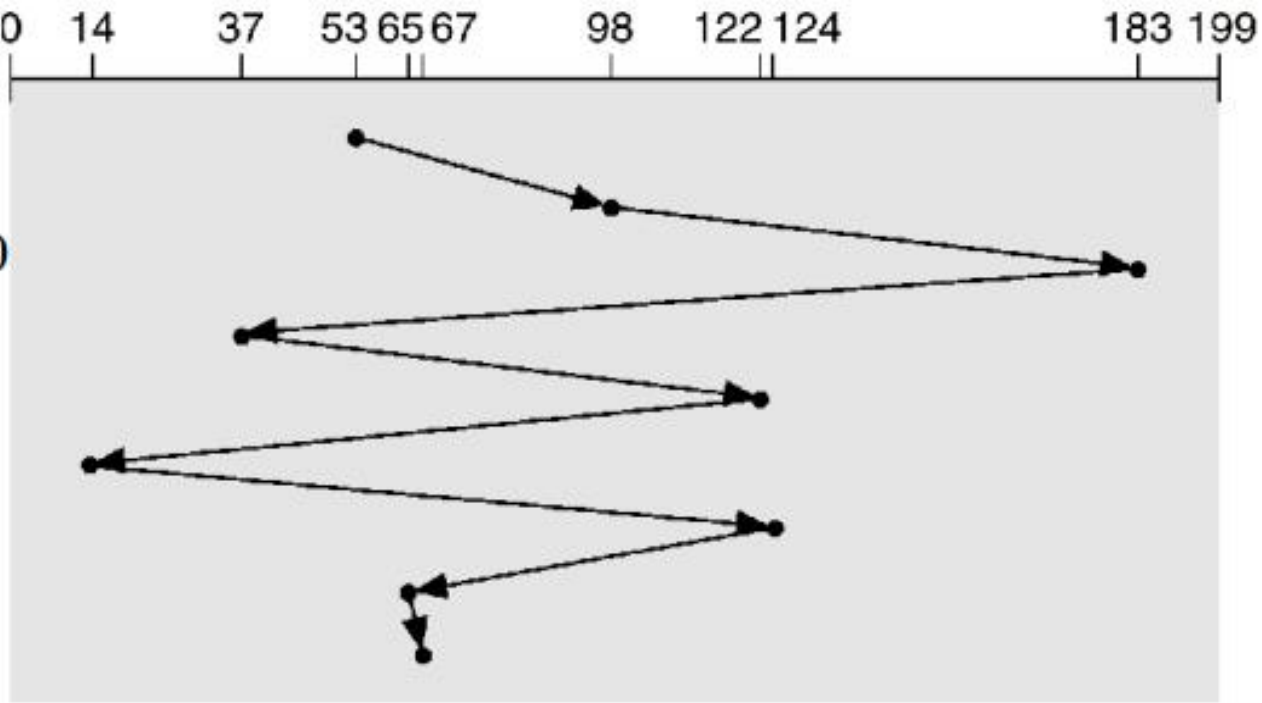
For most disks, the *seek time* dominates the other two times, so reducing the *mean seek time* can improve system performance substantially.

First-Come First-Served (FCFS)

The first request to arrive is the first one serviced.

Example:

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



Total Head Movement = 640 cylinders

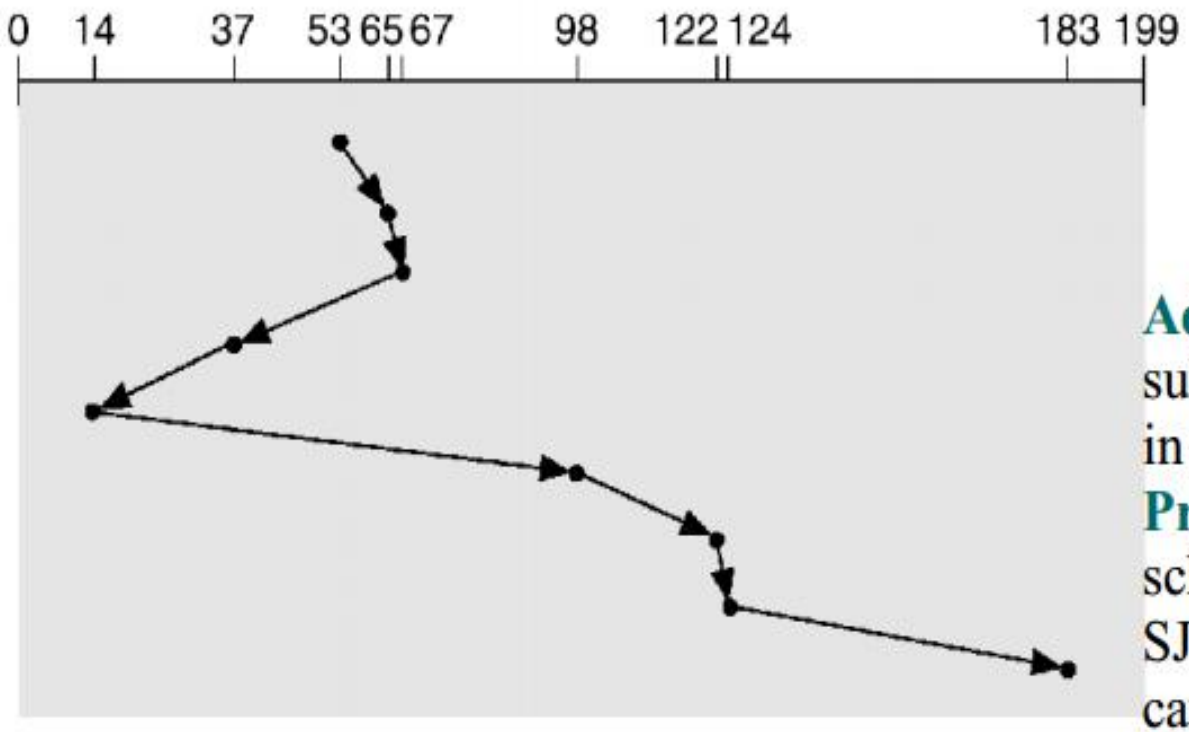
Advantages: Simple and Fair.

Problems: Does not provide fastest service.

Shortest-Seek-Time-First (SSTF)

Selects the request with the minimum seek time from the current head position.

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



Total Head Movement = 236 cylinders

Advantages: Gives a substantial improvement in performance.

Problems: SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests. Not optimal.

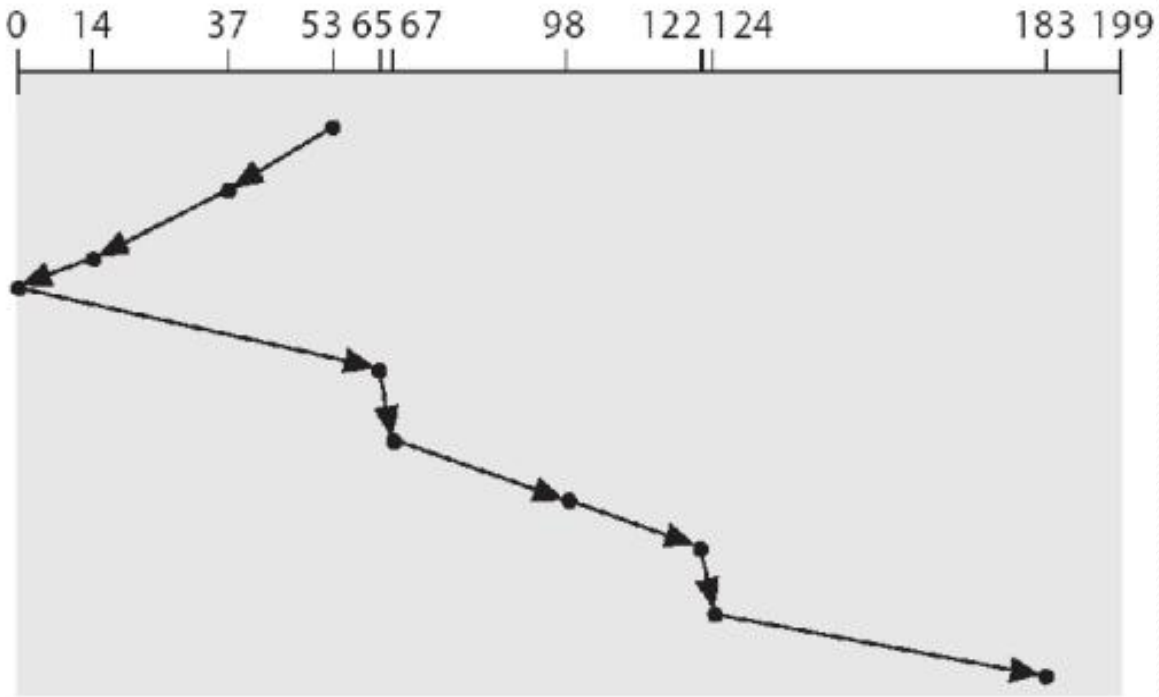
Used in batch system where throughput is the major consideration but unacceptable in interactive system.

SCAN

The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

Sometimes called the *elevator algorithm*.

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



Total Head Movement = 208 cylinders

Advantages: Decreases variances in seek and improve response time.
Problem: Starvation is possible if there are repeated request in current track.

C-SCAN

When a uniform distribution of request for cylinders, only the few request are in extreme cylinders, since these cylinders have recently been serviced.

Why not go to the next extreme?

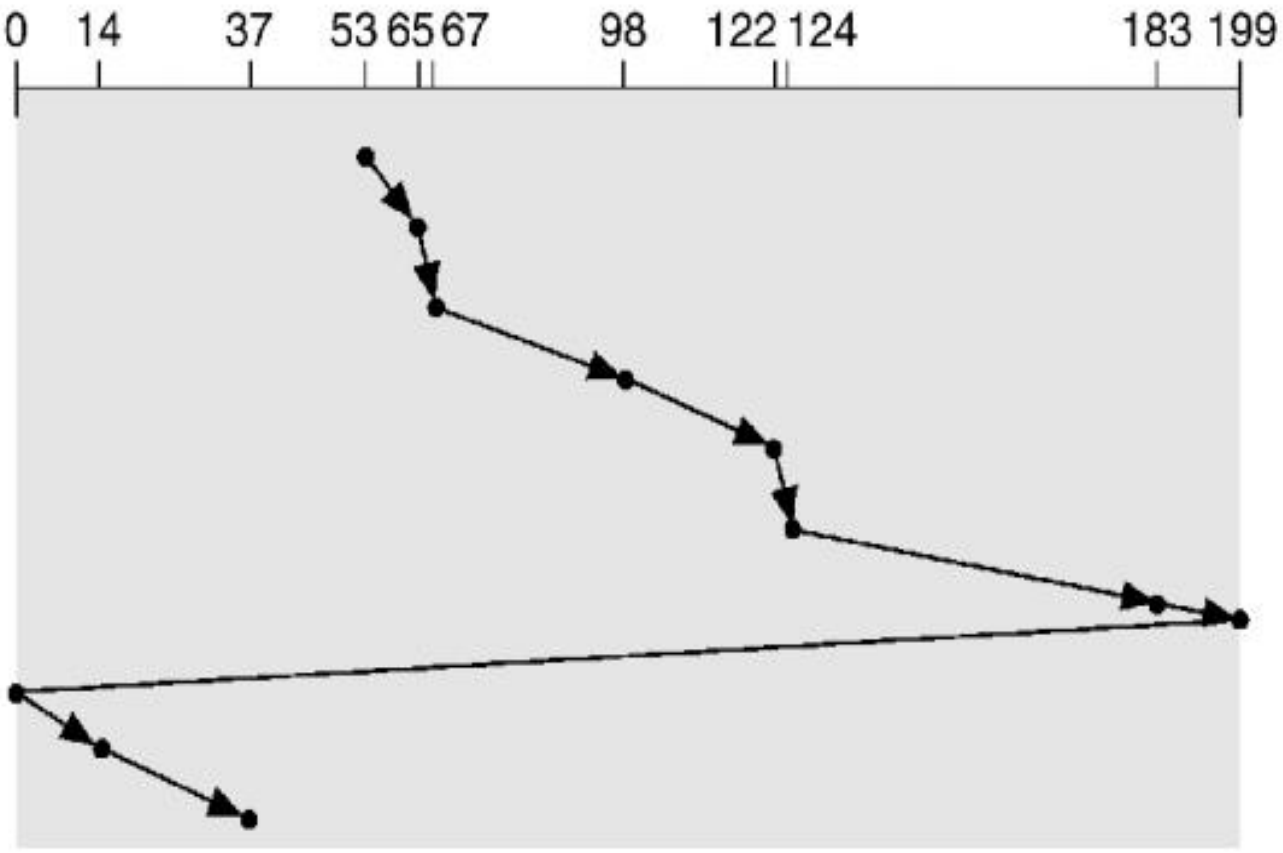
Circular SCAN is a variant of SCAN designed to provide a more uniform wait time.

The head moves from one end of the disk to the other. Servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.

Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

C-SCAN

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53

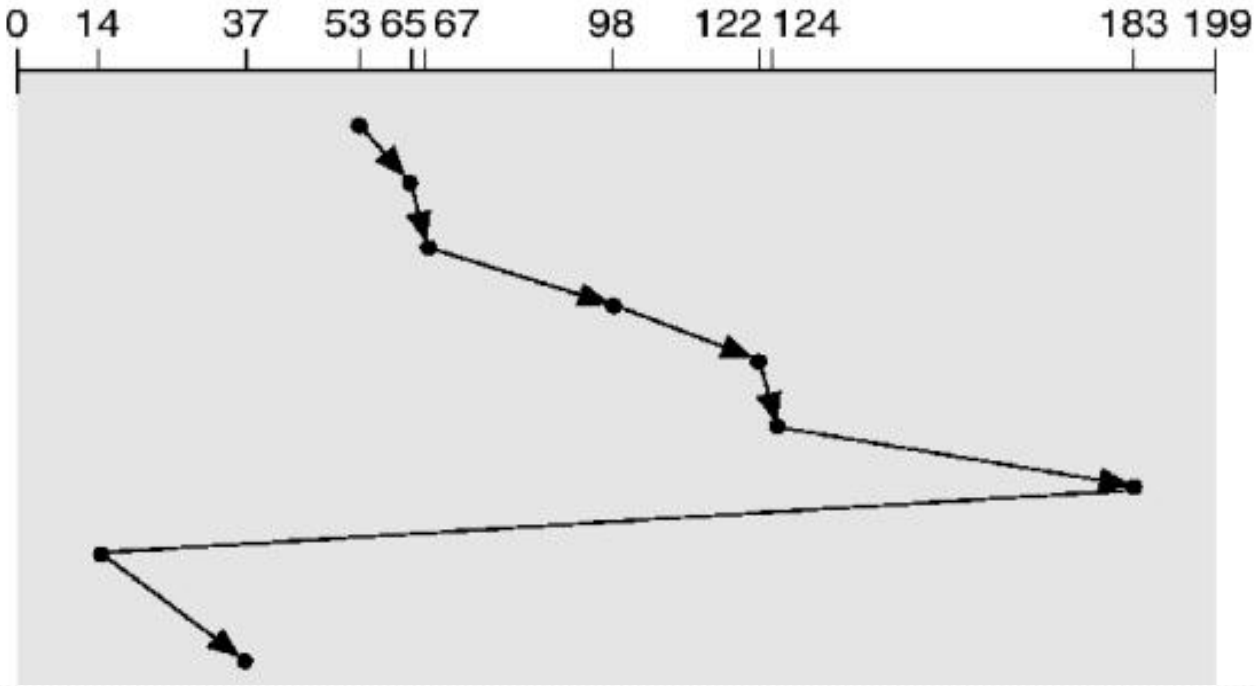


C-LOOK

Version of C-SCAN

Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



Class Work

- Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of the pending requests, in FIFO order, is 86, 1470, 913, 1774, 948, 1502, 1022, 1750, 130.Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the following disk-scheduling algorithms?

END