Operating System

CS3-2015

Unit 5

Part -1 [Page 1 – Page 24]

Contents: I/O Management Introduction, I/O Device, I/O Subsystem: Mode of Input Output Operation { Programmed Mode, Polling Mode, Interrupt Mode, DMA Mode}, I/O Buffering – Different Buffering Techniques , Disk Storage and Structure, Disk Scheduling Algorithms, RAID.

1. I/O Management or I/O System

Introduction : One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.

Also, much of whatever a computer system provides as on-line services is essentially made available through specialized devices such as screen displays, printers, keyboards, mouse, etc. Clearly, management of all these devices can affect the throughput of a system. For this reason, input output management also becomes one of the primary responsibilities of an operating system

An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application.

2. I/O Devices and their types from point view of communication

(I) I/O Devices

Input devices attached with a computer system are used for providing the input to the computer. Different types of input devices are keyboard, mouse, scanner, etc.

Output devices are used to take the result from the computer and provide it to the user. Output devices are monitor, printer, speaker etc.

From point view of Communication I/O devices are of following types.

(a) Human Readable

- Used to communicate with the user
- Printers
- Video display terminals
    - Display
    - Keyboard

      Mouse

(b) Machine or Device readable

- Used to communicate with electronic equipment
- Disk and tape drives
- Sensors
- Controllers

(c) Communication Over Network

- Used to communicate with remote devices
- Digital line drivers
- Modems
- Router

Other Classification of I/O Devices

I/O devices can be divided into two categories –

- **Block devices** – A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.
- **Character devices** – A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards et

3. Mode for I/O Operation

Computers employ the following four basic modes of IO operation:

a. Programmed mode
b. Polling mode
c. Interrupt mode
d. Direct memory access mode.

(a)Programmed Mode

In this mode of communication, execution of an IO instruction ensures that a program shall not advance till it is completed. To that extent one is assured that IO happens before anything else happens. As depicted in Figure1, in this mode an IO instruction is issued to an IO device and the program executes in "busy-waiting" (idling) mode till the IO is completed. During the busy-wait period the processor is continually interrogating to check if the device has completed IO. Invariably the data transfer is accomplished through an identified register and a flag in a processor. For example, in Figure.1 depicts

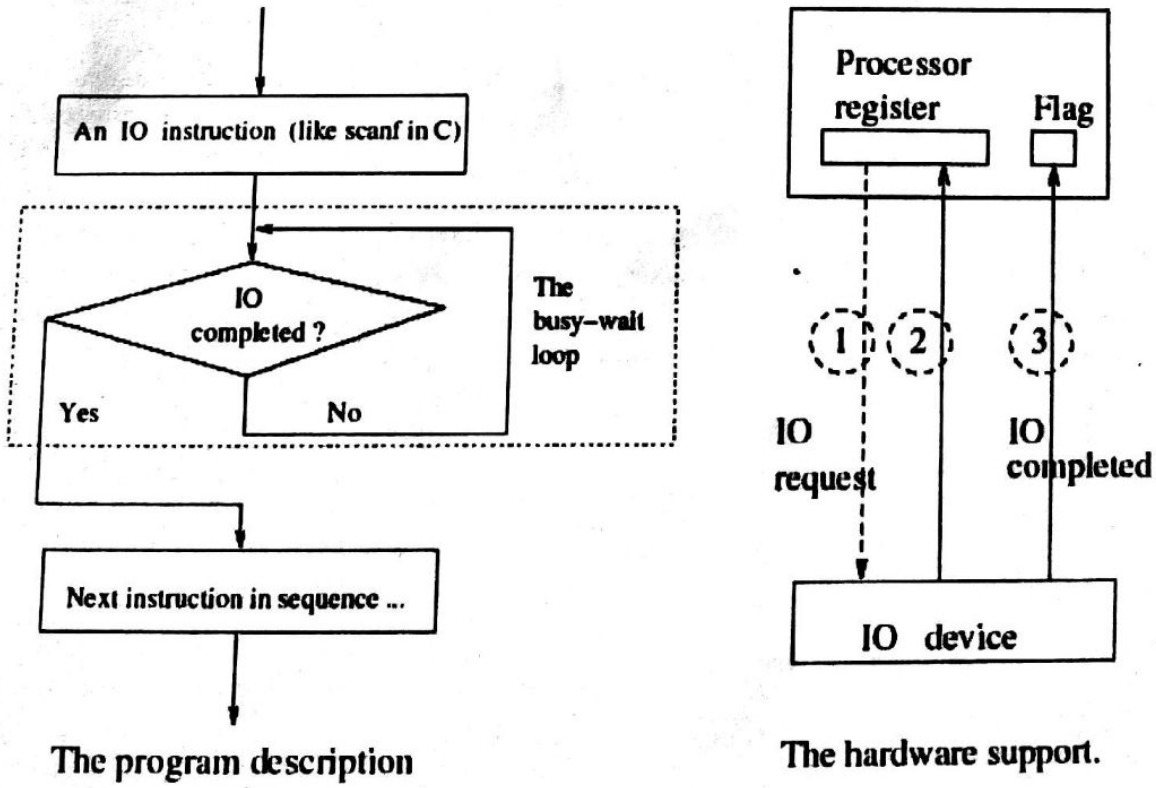


Figure 1: Programmed I/O.

how an input can happen using the programmed data mode. First the processor issues an IO request (shown as 1), followed by device putting a data in a register (shown as 2) and finally the flag (which is being interrogated) is set (shown as 3). The device either puts a data in the register (as in case of input) or it picks up data from the register (in case of output). When the IO is accomplished it signals the processor through the flag.

During the busy-wait period the processor is busy checking the flag. However, the processor is idling from the point of view of doing anything useful. This situation is similar to a car engine which is running when the car is not in motion – essentially "idling".

(b)Polling

In this mode of data transfer, shown in Figure 2, the system interrogates each device in turn to determine if it is ready to communicate. If it is ready, communication is initiated and subsequently the process continues again to interrogate in the same sequence. This is just like a round-robin strategy. Each IO device gets an opportunity to establish Communication in turn. No device has a particular advantage (like say a priority) over other devices.

Polling is quite commonly used by systems to interrogate ports on a network. Polling may also be scheduled to interrogate at some pre-assigned time intervals. It should be remarked here that most daemon software operate in polling mode. Essentially, they use a while true loop as shown in Figure.2.

In hardware, this may typically translate to the following protocol:
1. Assign a distinct address to each device connected to a bus.
2. The bus controller scans through the addresses in sequence to find which device wishes to establish a communication.
3. Allow the device that is ready to communicate to leave its data on the register.
4. The IO is accomplished. In case of an input the processor picks up the data. In case of an output the device picks up the data.
5. Move to interrogate the next device address in sequence to check if it is ready to communicate.

```
While true
    if device–1 ready to communicate
        then perform IO with device 1

    if device–2 ready to communicate
        then perform IO with device 2
                    .
                    .
                    .
    if device–k ready to communicate
        then perform IO with device k
end while
```

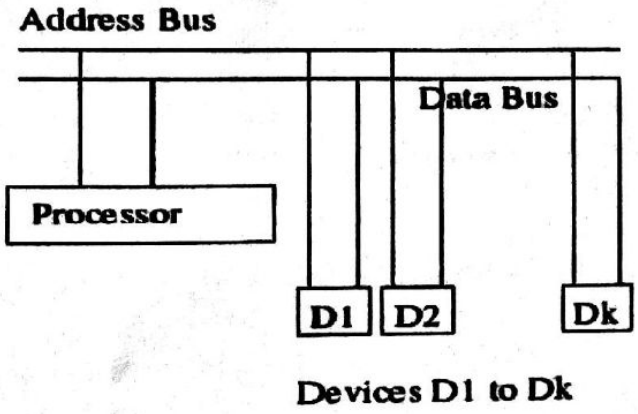


Figure 2 : Polling Mode.

(C) Interrupt Mode

Let us begin with a simple illustration to explain the basic rationale behind interrupt mode of data transfer. Suppose a program needs input from a device which communicates using interrupt. Even with the present-day technology the devices are one thousand or more times slower than the processor. **So if the program waits on the input device it would cycle through many processor cycles just waiting for the input device to be ready to communicate. This is where the interrupt mode of communication scores. To begin with, a program may initiate IO request and advance without suspending its operation. At the time when the device is actually ready to establish an IO, the device raises an interrupt to seek communication.**

Immediately the program execution is suspended temporarily and current state of the process is stored. The control is passed on to an interrupt service routine (which may be specific to the device) to perform the desired input. Subsequently, the suspended process context is restored to resume the program from the point of its suspension.

Interrupt processing may happen in the following contexts:

- **Internal Interrupt**: The source of interrupt may be a memory resident process or a function from within the processor. We regard such an interrupt as an internal interrupt. A processor malfunction results in an internal interrupt. An attempt to divide by zero or execute an illegal or non-existent instruction code results in an internal interrupt as well. A malfunction arising from a division by zero is called a trap.

Internal interrupt may be caused by a timer as well. This may be because either the allocated processor time slice to a process has elapsed or for some reason the process needs to be pre-empted. Note that an RTOS may pre-empt a running process by using an interrupt to ensure that the stipulated response time required is met. This would also be a case of internal interrupt.

- **External Interrupt:** If the source of interrupt in not internal, i.e. it is other than a process or processor related event then it is an external interrupt. This may be caused by a device which is seeking attention of a processor. As indicated earlier, a program may seek an IO and issue an IO command but proceed. After a while, the device from which IO was sought is ready to communicate. In that case the device may raise an interrupt. This would be a case of an external interrupt.

- **Software Interrupt:** Most OSs offer two modes of operation, the user mode and the system mode. Whenever a user program makes a system call, be it for IO or a special service, the operation must have a transition from user mode to system mode. An interrupt is raised to effect this transition from user to system mode of operation. Such an interrupt is called a software interrupt. We shall next examine how an interrupt is serviced. Suppose we are executing an

instruction at *i* in program P when interrupt signal has been raised. Let us also assume that we have an interrupt service routine which is to be initiated to service the interrupt.

The following steps describe how a typical interrupt service may happen.

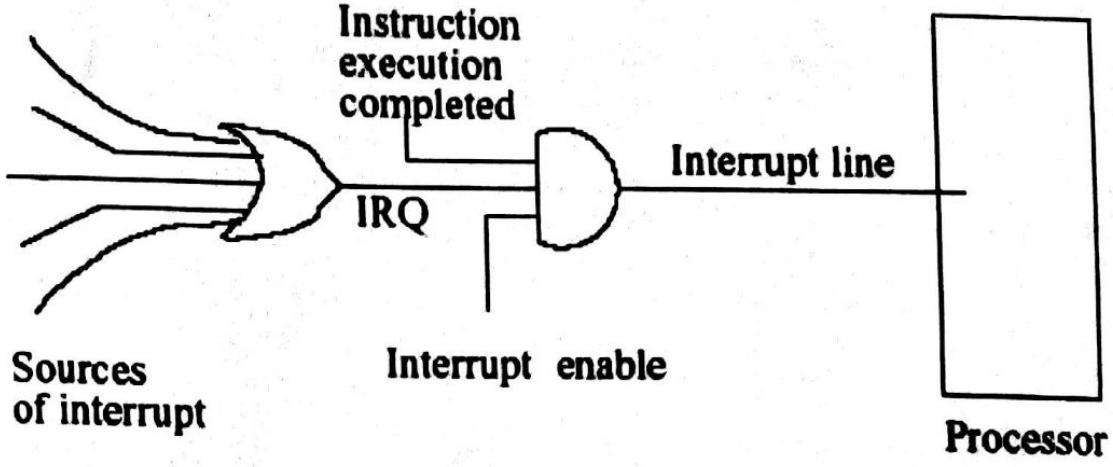


Figure 3: How an interrupt is detected ?

Detecting an interrupt: In Figure 5.3 we depict how a processor may detect an interrupt request. When a processor has an interrupt enable signal up, then at the end of an instruction cycle we shall recognize an interrupt if the interrupt request (IRQ) line is up. Once an interrupt is recognized, interrupt service shall be required. Two minor points now arise. One is when is interrupt enabled. The other is how a device which raises the interrupt is identified.

(D) Direct Memory Access

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.

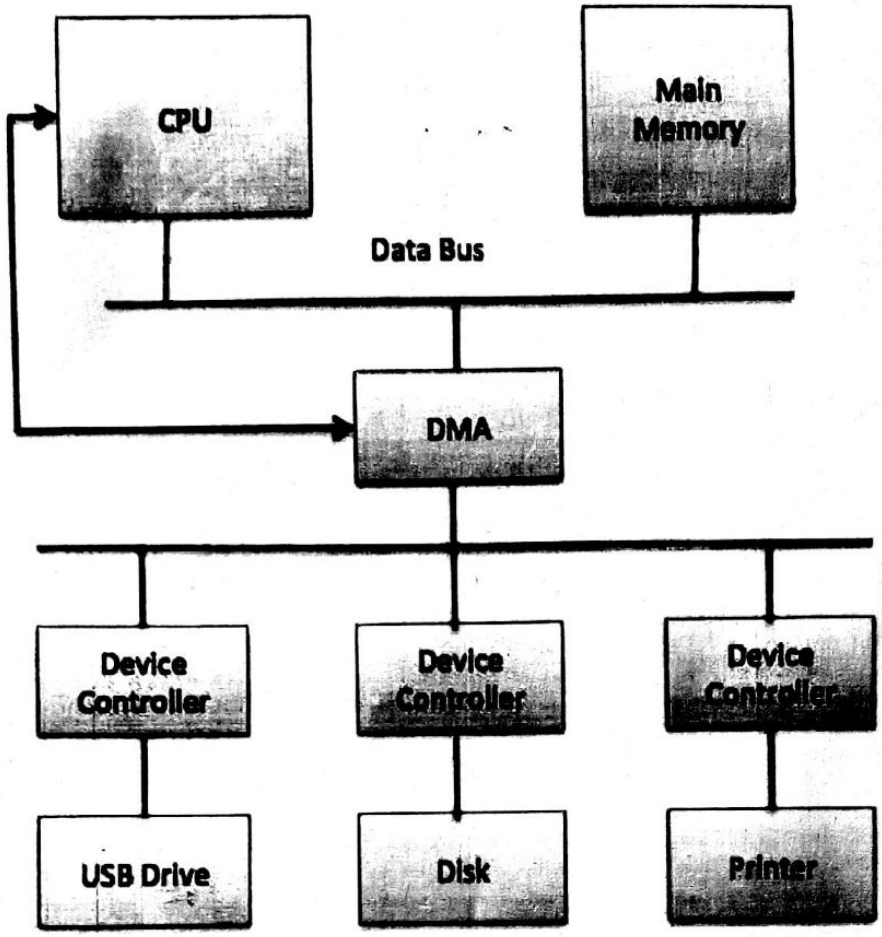


Figure 4 : Use of Direct Memory Access.

The operating system uses the DMA hardware as follows –

Step	Description
1	Device driver is instructed to transfer disk data to a buffer address X.
2	Device driver then instruct disk controller to transfer data to buffer.
3	Disk controller starts DMA transfer.
4	Disk controller sends each byte to DMA controller.

5 DMA controller transfers bytes to buffer, increases the memory address, decreases the counter C until C becomes zero.

6 When C becomes zero, DMA interrupts CPU to signal transfer completion

4.I/O Buffering Important

A Buffer is a memory area that stores data being transferred between two devices or between a device and an application.

Processes must wait for I/O to complete before proceeding. To avoid deadlock certain pages must remain in main memory during I/O

It may be more efficient to perform input transfers in advance of requests being made and to perform output transfers some time after the request is made

Buffering is done for following reasons **(or use of Buffering)**

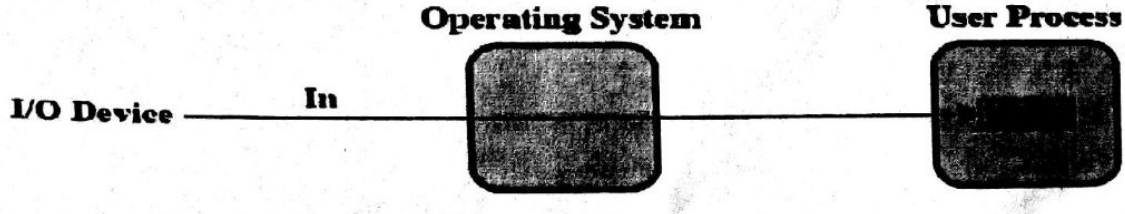
One reason is to cope with a speed mismatch between the producer and consumer of a data stream.

A second use of buffering is to provide adaptations for devices that have different data-transfer sizes. Such disparities are especially common in computer networking, where buffers are used widely for fragmentation and reassembly of messages. At the sending side, a large message is fragmented into small network packets. The packets are sent over the network, and the receiving side places them in a reassembly buffer to form an image of the source data.

4.1. Different Buffering Techniques

(a)NO Buffer Concept

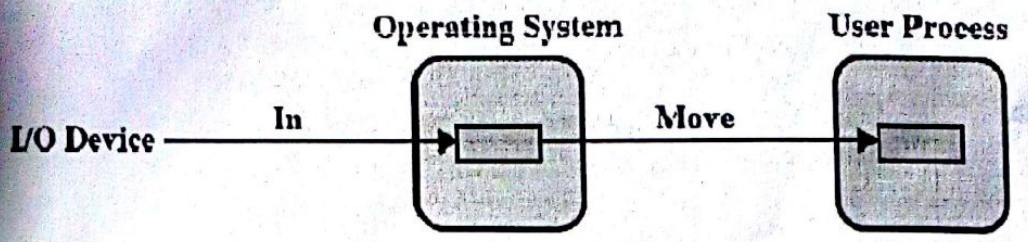
Without a buffer, the OS directly access the device as and when it needs.



(a) No buffering

(b)Single Buffer Concept

Operating system assigns a buffer in main memory for an I/O request.



(b) Single buffering

Block Oriented Single Buffer

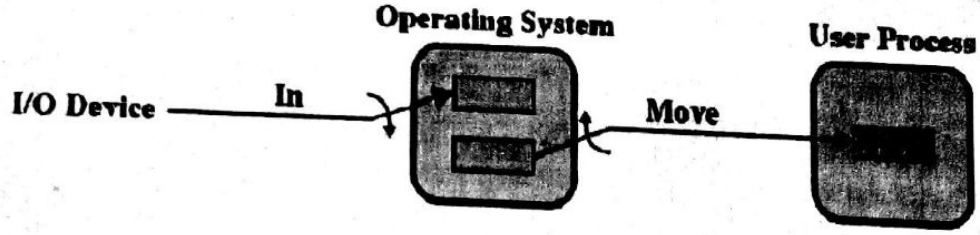
- Input transfers made to buffer
- Block moved to user space when needed
- The next block is moved into the buffer
  - *Read ahead* or *Anticipated Input*
- Often a reasonable assumption as data is usually accessed sequentially.

Stream-oriented Single Buffer

- Line-at-time or Byte-at-a-time.
- Terminals often deal with one line at a time with carriage return signaling the end of the line.
- Byte-at-a-time suites devices where a single keystroke may be significant.
  - Also sensors and controllers.

(c)Double Buffer

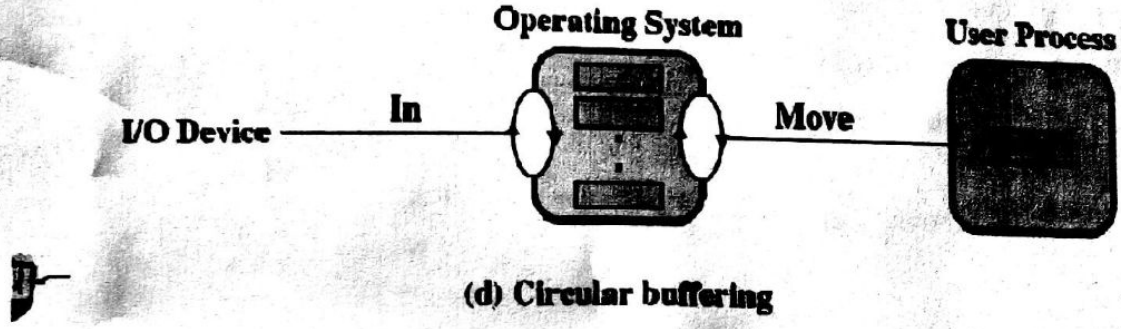
- Use two system buffers instead of one
- A process can transfer data to or from one buffer while the operating system empties or fills the other buffer



(c) Double buffering

(d)Circular Buffer

- More than two buffers are used.
- Each individual buffer is one unit in a circular buffer.
- Used when I/O operation must keep up with process.



(d) Circular buffering

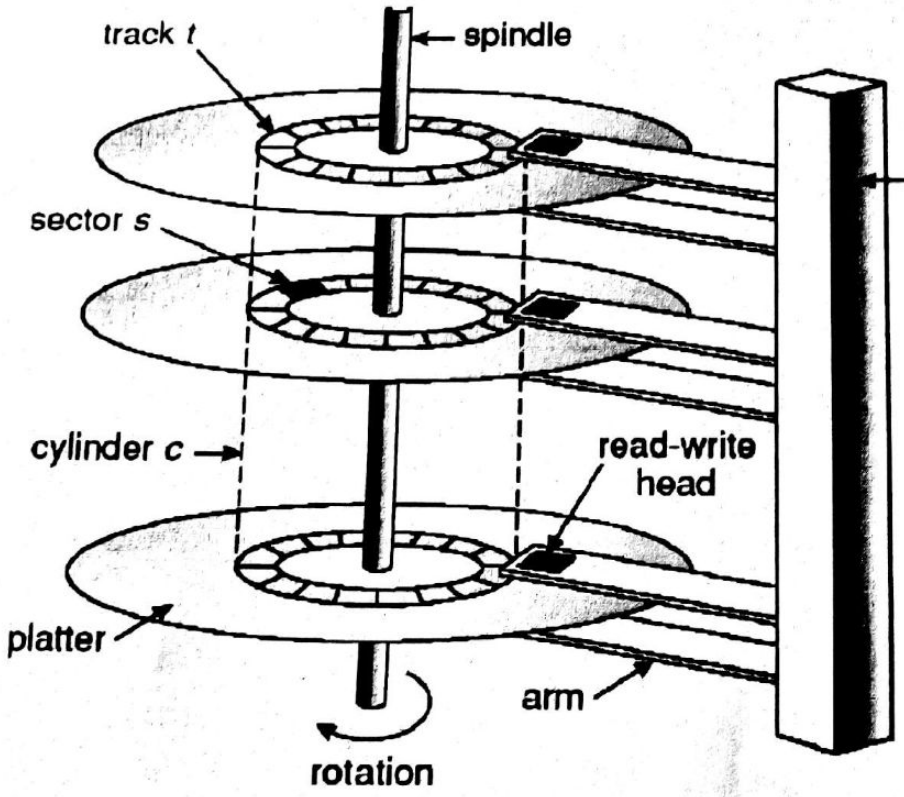
4.2 Buffer Limitations

Buffering smoothes out peaks in I/O demand. But with enough demand eventually all buffers become full and their advantage is lost. However, when there is a variety of I/O and process activities to service, buffering can increase the efficiency of the OS and the performance of individual processes

5.Disk Structure and Storage or Disk Organization

- Disk drives are addressed as large 1-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer.
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.
- Sector 0 is the first sector of the first track on the outermost cylinder.
- Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

Physical Structure of the Disk : As shown in following figure



- Range from 30GB to 3TB per drive
- Aluminum platters with magnetic coating
- Commonly, 2-5 platters per drive
- Common platter sizes: 3.5", 2.5", and 1.8"
- Magnetic heads