OS Chapter 6

"I/O Management"

INTRODUCTION :

- One of the main functions of OS is to control all the IO devices of the computer system.
- OS hides internal complexity IO devices.

1) Principles of I/O Hardware:

1) I/O devices,
2) Device controllers ,
3) Direct memory access

1) I/O devices :

- The devices which are used for input and output purpose are known as I/O devices.
- I/O devices can be roughly divided into two categories: block devices and character devices.

Block Device

- A block device is one that stores information in fixed-size blocks, each one with its own address. Common block sizes range from 512 to 65,536 bytes. All transfers are in units of one or more entire (consecutive) blocks.
- It is possible to read or write each block independently of all the other ones.
- Hard disks, Blu-ray discs, and USB sticks are 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 :Printers, network interfaces

2) Device controllers

- I/O unit consist of a 2 components:
  1) Mechanical component: Device itself
  2) Electronic component: known as device controller
- "Electronic component of I/O devices is called the **Device Controller**."
- The controller card usually has a connector on it, into which a cable leading to the device itself can be plugged.
- Controller's tasks :
  1) convert serial bit stream to block of bytes
  2) perform error correction as necessary
  3) make available to main memory

4) Acts as an interface between device and OS
5) Each controller has few registers which are used for communicating with the CPU.
6) By writing into those registers, the OS can give commands to the device to do something, like turn on or turn off device, deliver data, accept data etc.

3) Direct memory access

- To explain how DMA works, let us first look at how disk reads occur when DMA is not used.

1) CPU requests the device controller to read one byte from the disk and transfer it into the RAM.
2) After receiving the request from the CPU, controller reads one byte from the disk and writes it in RAM.
3) After completing this for just one block, controller sends notification to the CPU that it has completed the transfer of one byte.
4) After receiving notification from the driver, CPU repeats steps 1) , 2) and 3) and completes the transfer of n number of blocks.

Disadvantages:

- Here, CPU utilization is low, because after transfer of each block is completed, CPU has to send request again and again.
- This concept is known as "Interrupt Driven I/O".

DMA:

- To overcome the problem occurring in the interrupt driven I/O, DMA can be used.
- DMA utilizes CPU more than interrupt driven I/O.
- In DMA CPU transfers the responsibility to DMA controller.
- When DMA is used, the procedure is different.

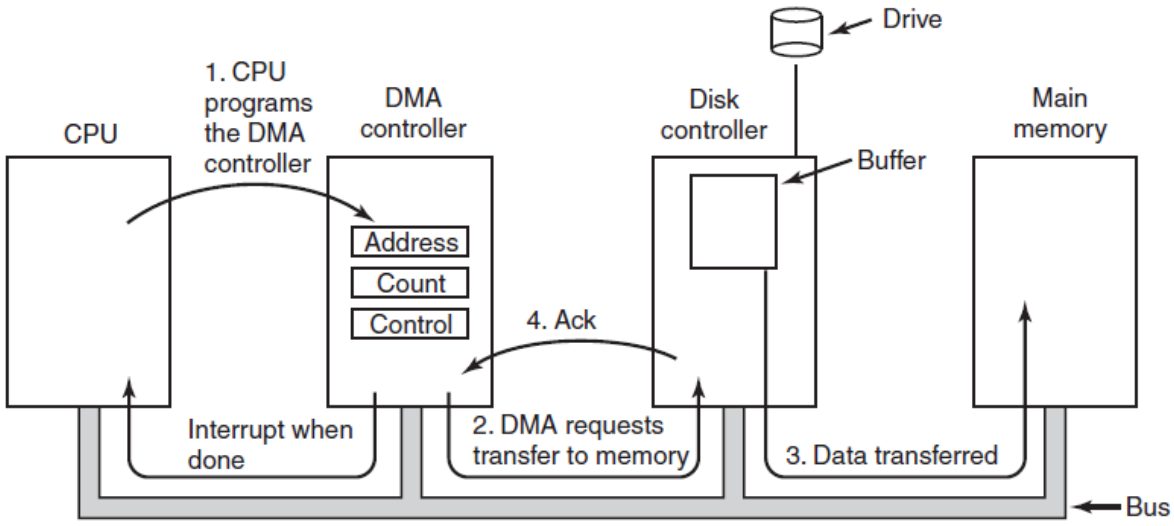


Figure: Operation of a DMA transfer

1) CPU initiates the transfer and requests DMA controller to begin the transfer by sending starting address and total no. of blocks to read (count).
2) DMA controller requests Disk Controller to transfer the data from the disk to main memory.
3) After completing the transfer the Disk Controller gives acknowledge to DMA controller.
4) DMA controller will decrement the count by one.

```
If(count > 0)
{
        Step 2) , 3) and 4) are repeated.
}
else
{
        DMA controller will acknowledge the CPU that reading task is
        completed.
}
```

Advantages:

- CPU doesn't receive acknowledgement after each block transfer like the case of interrupt driven I/O.
- So, CPU can be utilized in better way than interrupt driven I/O.

2) Principles of I/O Software:

1) Device independence
2) Uniform Naming
3) Error handling
4) Synchronous vs. Asynchronous transfers
5) Buffering

1) Device independence:

- A key concept in the design of I/O software is known as **device independence**.
- What it means is that we should be able 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 hard disk, a DVD, or on a USB stick without having to be modified for each different device.
- It is up to the operating system to take care of the problems caused by the fact that these devices really are different and require very different command sequences to read or write.

2) Uniform Naming:

- Closely related to device independence is the goal of **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 UNIX, all disks can be integrated in the file-system hierarchy, so the user need not be aware of which name corresponds to which device.
- All files and devices are addressed the same way: by a path name.

3) Error handling

- Another important issue for I/O software is **error handling**. In general, "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.
- Many errors are temporary, such as read errors caused by dust on the read head, and will frequently go away if the operation is repeated.
- Only if the lower layers are not able to deal with the problem, the upper layers are told about it.
- In many cases, error recovery can be done transparently at a low level without the upper levels even knowing about the error.

4) Synchronous vs. Asynchronous transfers

- Synchronous means blocking transfer
- Asynchronous means interrupt-driven transfers.
- Most physical I/O is asynchronous—the CPU starts the transfer and goes off to do something else until the interrupt arrives.
- User programs are much easier to write if the I/O operations are blocking—after a read system call the program is automatically suspended (blocked) until the data are available in the buffer.
- It is up to the operating system to make operations that are actually Interrupt-driven look blocking to the user programs.

5) Buffering

- Often data that come from a device cannot be stored directly in their final destination.

- For example, when a packet comes in from the network, the operating system does not know where to put it until it has stored the packet somewhere and examined it.
- Buffering involves copying and often has a major impact on I/O performance.

3) I/O Software Layers

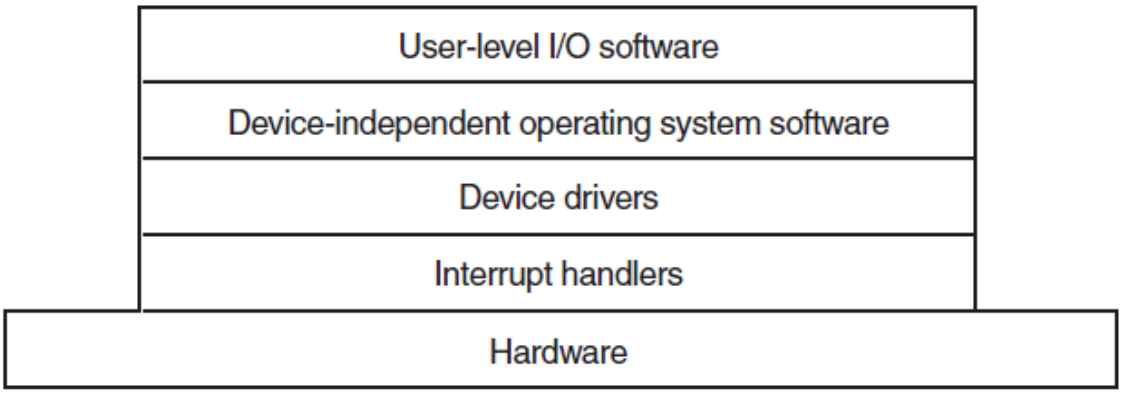


Figure: Layers of the I/O software system

Interrupt Handlers

- Interrupt: "A signal that gets the attention of the CPU and is usually generated when I/O is required".
- For example:
  - Hardware interrupts are generated when a key is pressed or when the mouse is moved.
  - Software interrupts are generated by a program requiring disk input or output.
  - An internal timer may continually interrupt the computer several times per second to keep the time of day current or for timesharing purposes.
- When an interrupt occurs, control is transferred to the operating system, which determines the action to be taken related to I/O device.

- Interrupts play a major role to communicate between Device and CPU.
- So we need to have a software layer which can handle the interrupts related to I/O device.

Steps:

- CPU saves the state of currently executing process.
- Stops current process.
- Runs the interrupt-service procedure. It will extract information from the interrupting device controller's registers. thus, it serves the new process by loading new data structure (Process table) of new process.
- After serving the interrupt, again old process is started from the point it left.

Device Drivers

- "Each I/O device attached to a computer needs some device-specific code for controlling it, This code is called as **device driver"**.
- It is generally written by the device's manufacturer and delivered along with the device.
- 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 at most, one class of closely related devices.
- In order to access the device's hardware, meaning the controller's registers, device driver should be a part of operation system kernel.
- Device drivers are normally positioned below the rest of Operating System.

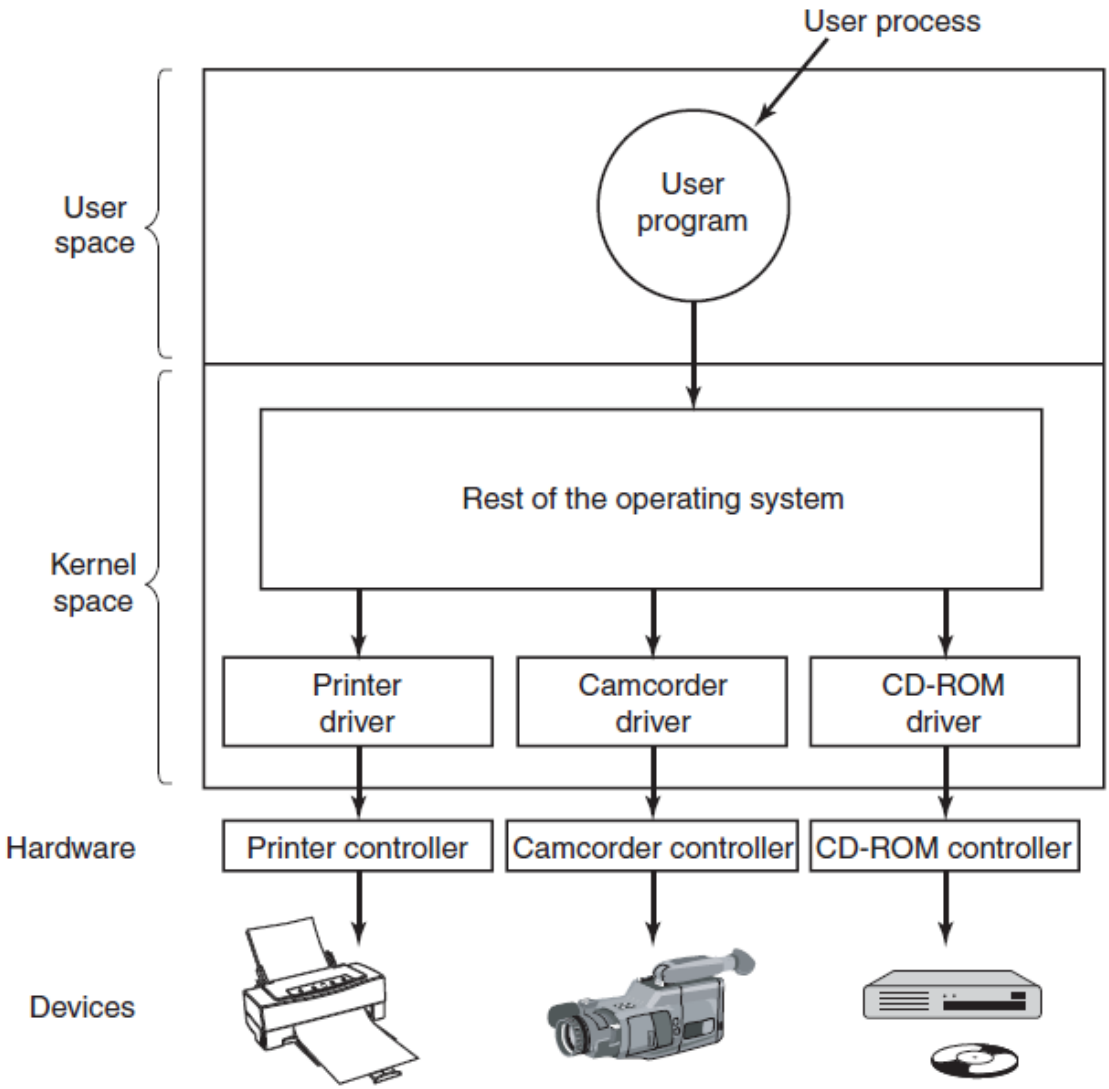


Figure : Logical positioning of device drivers.

Functions of device drivers:

1) Device driver accept read and write requests from device independent software.
2) Device driver must initialize the device if needed.
3) It also checks statues of devices. If it is currently in use then queue the request for latter processing. If device is in idle state then request can be handled now.
4) Controlling device means issuing a sequence of command to it. Device driver is a place where command sequence is determined, depending upon what has to be done.

5) Device driver writes the command sequence to the control register.
6) It provides error handling facility related to a particular I/O device.

Device-Independent I/O Software

- As we know OS interfaces with different device drivers.
- Each device has separate interface with OS and driver function of different device differ from driver to driver (and device to device). It means interfacing with each new driver requires lot of new programming effort.
- So we need a layer which can handle this issue and which is device independent.

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicated devices
Providing a device-independent block size

Figure : Functions of the device-independent I/O software

Responsibilities or functions of device independent I/O software:

1) Uniform interfacing for device drivers

- It provides uniform interface between OS and different device drivers.
- If this layer is not included, the programming efforts are large to write different interface code for different drivers.

2) Buffering

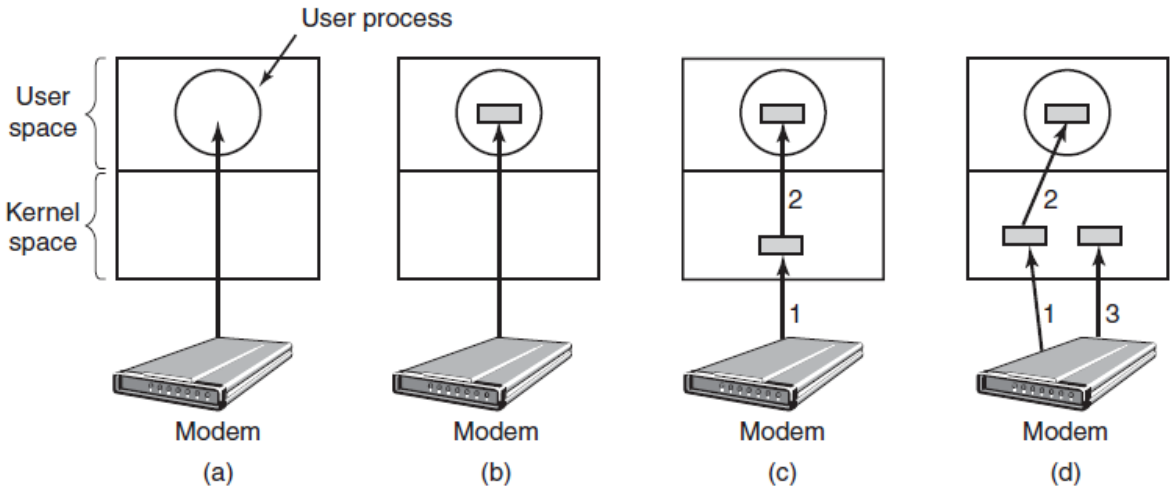


Figure:

(a) Unbuffered input.

(b) Buffering in user space.

(c) Buffering in the kernel followed by copying to user space.

(d) Double buffering in the kernel

In the last scheme, the first buffer is used to store characters. When it is full, it is being copied to user space. During that time the second buffer is used. In this way, two buffers take turns. This is called double buffering scheme.

- Here, temporary storage is provided to the I/O devices.
- I/O performance increases due to buffering capabilities.

3) Error reporting

- Errors which are specific to I/O device are handled by Device Controller/ Device Driver.
- General errors are handled by Device independent I/O Software.
- Here, general errors are handled like :
  - Writing to an input device (keyboard, mouse, scanner)
  - Reading from an output device (Printer, Plotter)
  - Specifying invalid device to use (Ex. trying to access Disk3, while system has only 2 Disks).

- Another class of the errors:
  - Trying to read/write a disk block that has been damaged.
  - Trying to read from a camcorder that has been switched off.
- In this case it is up to the device driver to decide what to do. If the driver doesn't know what to do, it may pass the problem to the device independent I/O software.

4) Allocating and releasing dedicated devices

- Some devices such as CD-ROM recorders can be used only by a single process at any given moment.
- A mechanism for requesting and releasing dedicated devices is required.
- An attempt to acquire a device that is not available blocks the caller instead of failing.
- Blocked processes are put on a queue, sooner or later the requested device becomes available and the first process on the queue is allowed to acquire it and continue execution.

5) Providing a device-independent block size

- Different disks may have different sector sizes.
- It is up to the device independent I/O software to hide this fact and provide a uniform block size to higher layers.
- Character devices deliver data character by character; others deliver them into large unit. This difference should be hidden.

4) Disk Structure

- Each disk platter has a flat circular shape, like a CD. The two surfaces of a platter are covered with a magnetic material. We store information by recording it magnetically on the platters.
- A read-write head "fly" just above each surface of every platter.
- The heads are attached to a **disk arm** that moves all the heads as a unit.
- The surface of a platter is logically divided into circular **tracks,** which are subdivided into **sectors.**
- The set of tracks that are at one arm position makes up a **cylinder.**
- There may be thousands of cylinders in a disk drive, and each track may contain hundreds of sectors.
- Modern disk drives are addressed as large one-dimensional arrays of **logical blocks,** where the logical block is the smallest unit of transfer.
- The size of a logical block is usually 512 bytes, although some disks can have a different logical block size, such as 1024 bytes.
- The one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially.
- The storage capacity of common disk drives is measured in gigabytes.

- When the disk is in use, a drive motor spins it at high speed. Disk speed has two parts. The **transfer rate** is the rate at which data flow between the drive and the computer. The **positioning time,** sometimes called the **random-access time,** consists of the time to move the disk arm to the desired cylinder, called the **seek time,** and the time for the desired sector to rotate to the disk head, called the **rotational latency.**

- Typical disks can transfer several megabytes of data per second, and they have seek times and rotational latencies of several milliseconds.

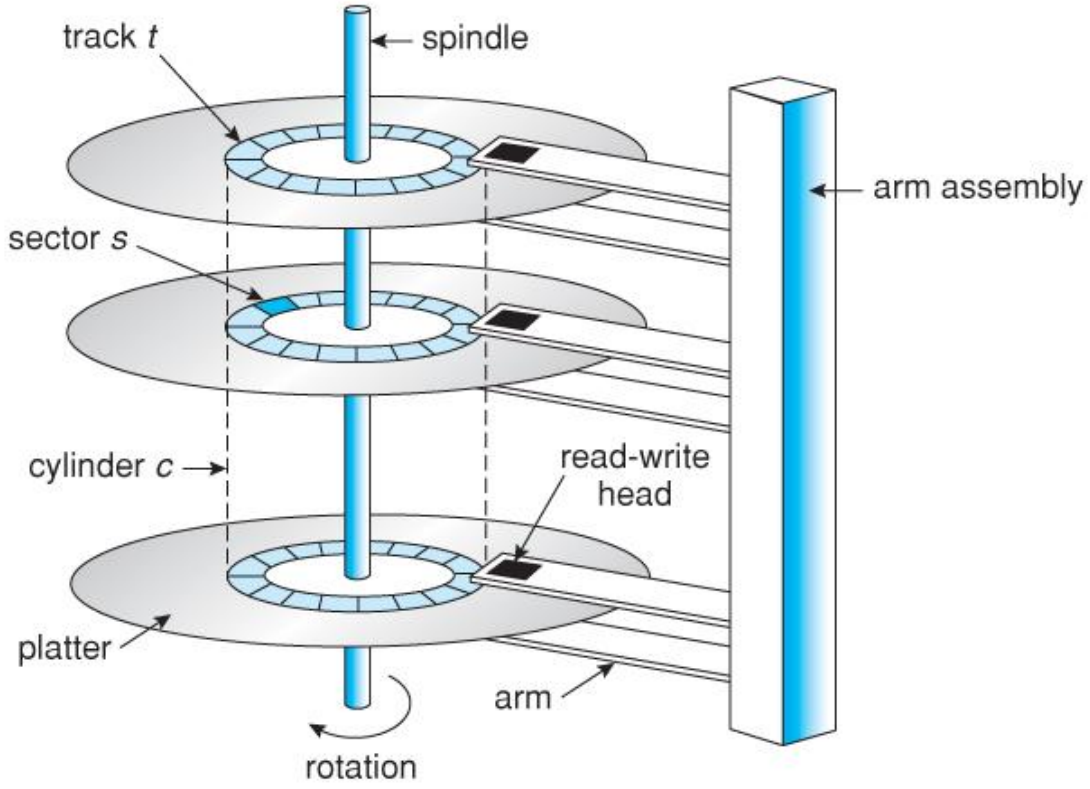


Figure: Disk Structure

5) Disk Arm Scheduling Algorithm

- The time required to read or write a disk block is determined by three factors:

  1. Seek time (the time to move the arm to the proper cylinder).
  2. Rotational delay (The Time required for the desired sector to rotate under the disk head).
  3. Actual data transfer time.

- For most disks, the seek time dominates the other two times, so reducing the mean seek time can improve system performance largely.
- Various types of disk arm scheduling algorithms are available to decrease mean seek time; which are as follows:

1. **FCFS**
2. **SSTF**
3. **SCAN**
4. **C-SCAN**
5. **LOOK**
6. **C-LOOK**

1. FCFS (First-Come, First-Served)

- "The request that comes first is served first".
- If the disk driver accepts requests one at a time and carries them out in that order, that is, FCFS.

Example:

Current Cylinder Position: **11**

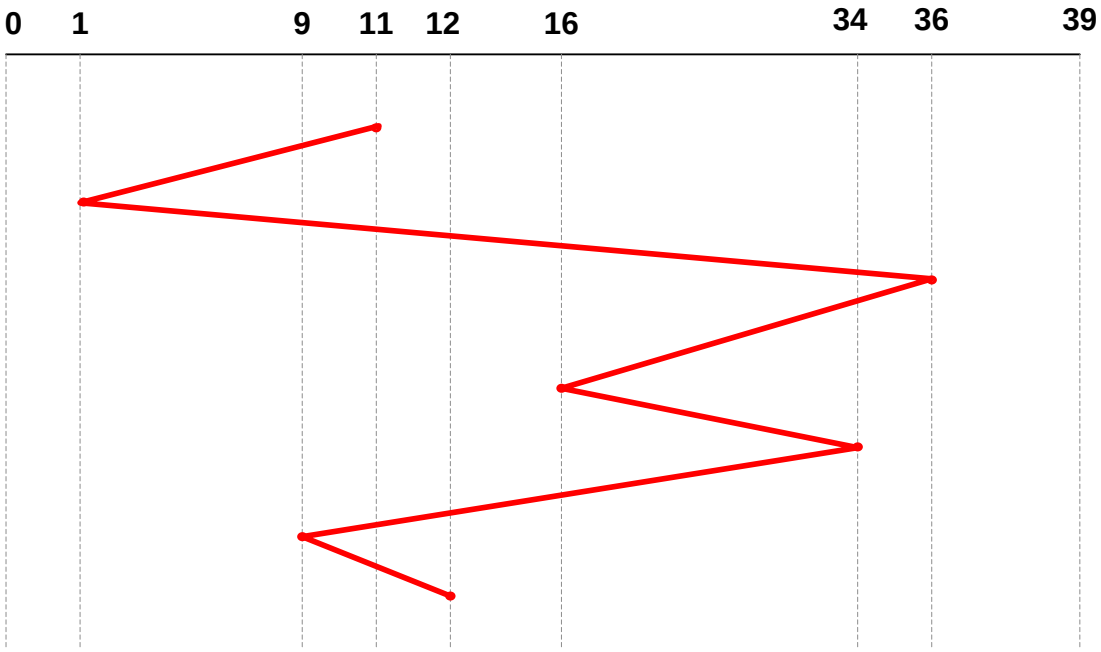
Cylinder Request: **1, 36, 16, 34, 9, 12**

Total Number of Cylinders: **40**

Find Total Arm Movement.

If 1 seek takes 5 ms then find total seek time for FCFS algorithm.

Note: Cylinder numbers start from 0. If Total number of cylinders is given as 40 then cylinder numbers are from 0 to 39.



Total Arm Movement = 10 + 35 + 20 + 18 + 25 + 3

= 111 Cylinders

Total Seek Time = 111 * 6

= 666 ms

2. SSF (Shortest Seek First) or SSTF (Shortest Seek Time First)

- The SSTF algorithm selects the request with the **minimum seek time** from the current head position.
- It always handles the **closest** request next, to minimize seek time.

Example:

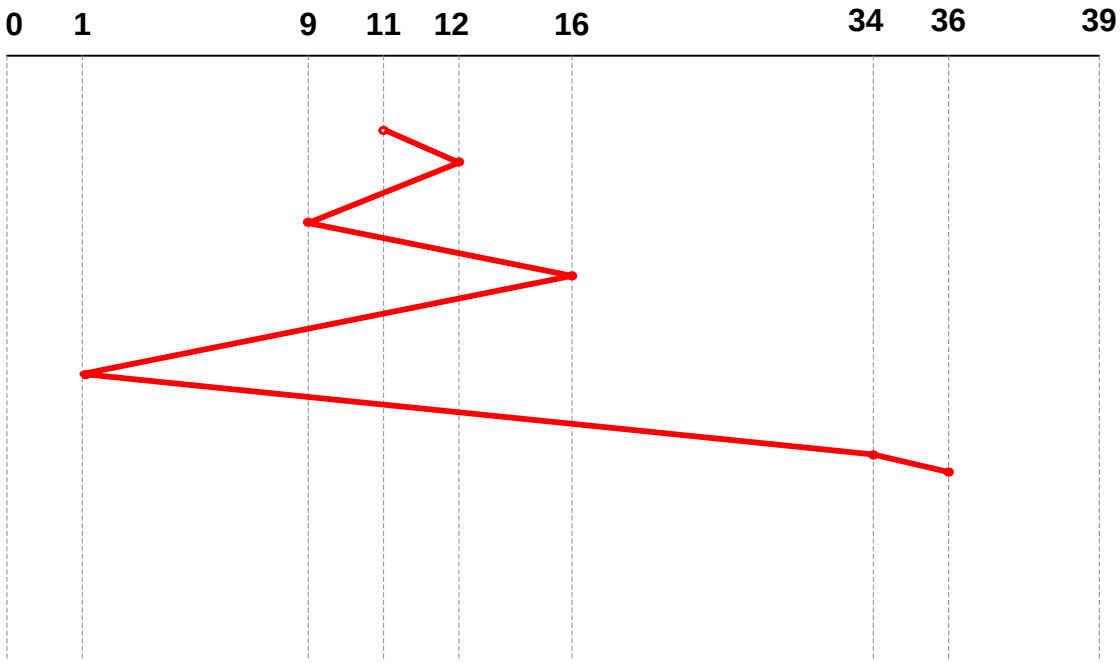
Current Cylinder Position: **11**

Cylinder Request: **1, 36, 16, 34, 9, 12**

Total Number of Cylinders: **40**

Find Total Arm Movement.

If 1 seek takes 5 ms then find total seek time for SSTF algorithm.



Total Arm Movement = 1 + 3 + 7 + 15 + 33 + 2

= 61 Cylinders

Total Seek Time = 61 * 6

= 366 ms

3. SCAN (or Elevator) *(According to Galvin and Greg, 1st reference book of the syllabus)*

- This algorithm works on the basic concept of elevator.
- **"**The arm starts moving in one direction (upward or downward) and serves all the requests coming in its path.
- After reaching to one end of disk, it reverses the direction and again serves all the remaining requests coming in its path**"**.

Example:

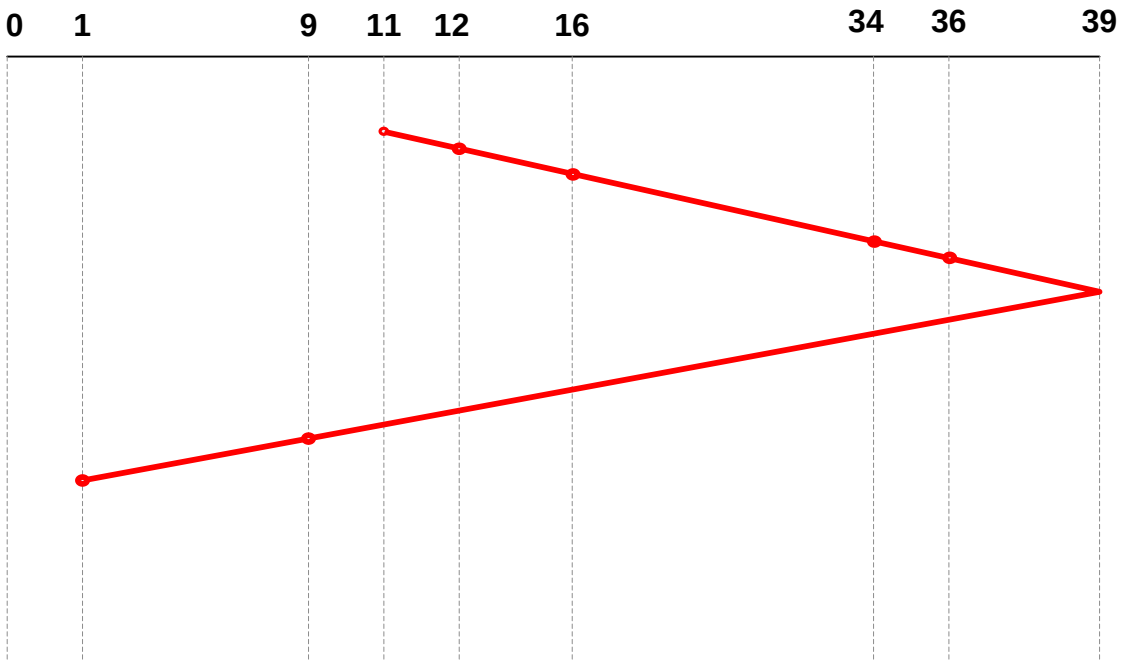
Current Cylinder Position: **11**

Cylinder Request: **1, 36, 16, 34, 9, 12**

Total Number of Cylinders: **40**

Find Total Arm Movement.

If 1 seek takes 5 ms then find total seek time for SCAN algorithm.



Total Arm Movement = 28 + 38

= 66 Cylinders

Total Seek Time = 66 * 6

= 396 ms

4. C-SCAN (Circular Scan)

- It works in the same way as SCAN with little changes as follows:
- "After reaching to One Disk End, it directly jumps to the Other End, without serving any request along that path".
- After reaching to that end again starts serving all the requests along that path.

Example:

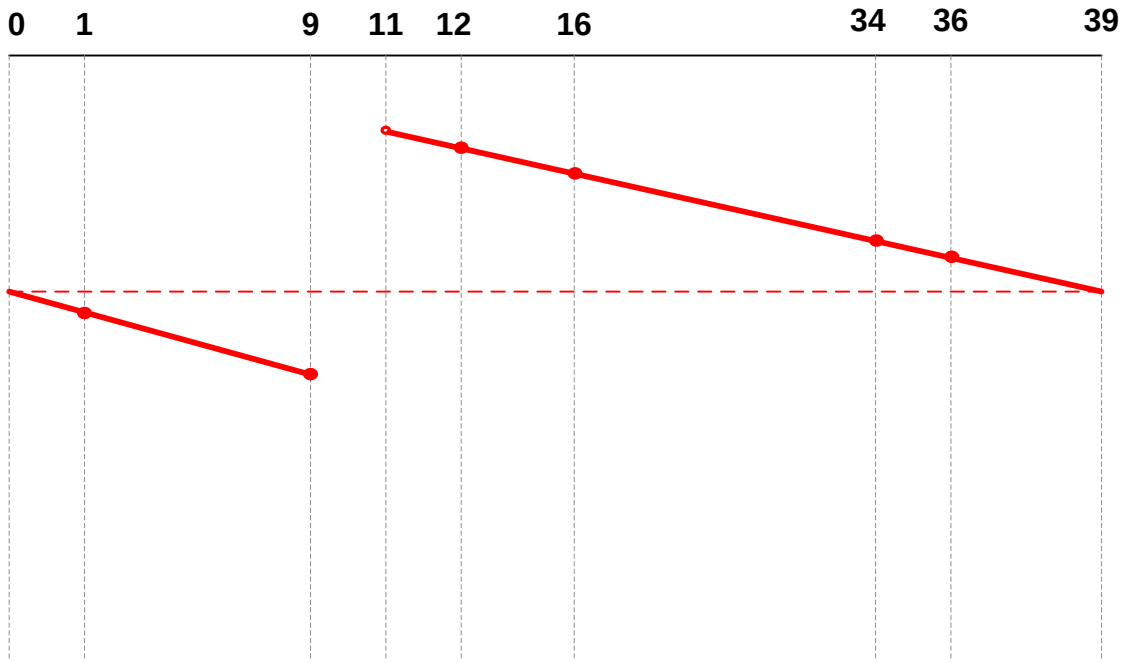
Current Cylinder Position: **11**

Cylinder Request: **1, 36, 16, 34, 9, 12**

Total Number of Cylinders: **40**

Find Total Arm Movement.

If 1 seek takes 5 ms then find total seek time for C-SCAN algorithm.



Total Arm Movement = 28 + 9

= 37 Cylinders

Total Seek Time = 37 * 6

= 222 ms

5. LOOK

- It is an enhanced version of SCAN algorithm.
- "The arm starts moving in one direction (upward or downward) and serves all the requests coming in its path.
- After reaching to last request in one direction of disk, it reverses the direction and again serves all the remaining requests coming in its path".

Example:

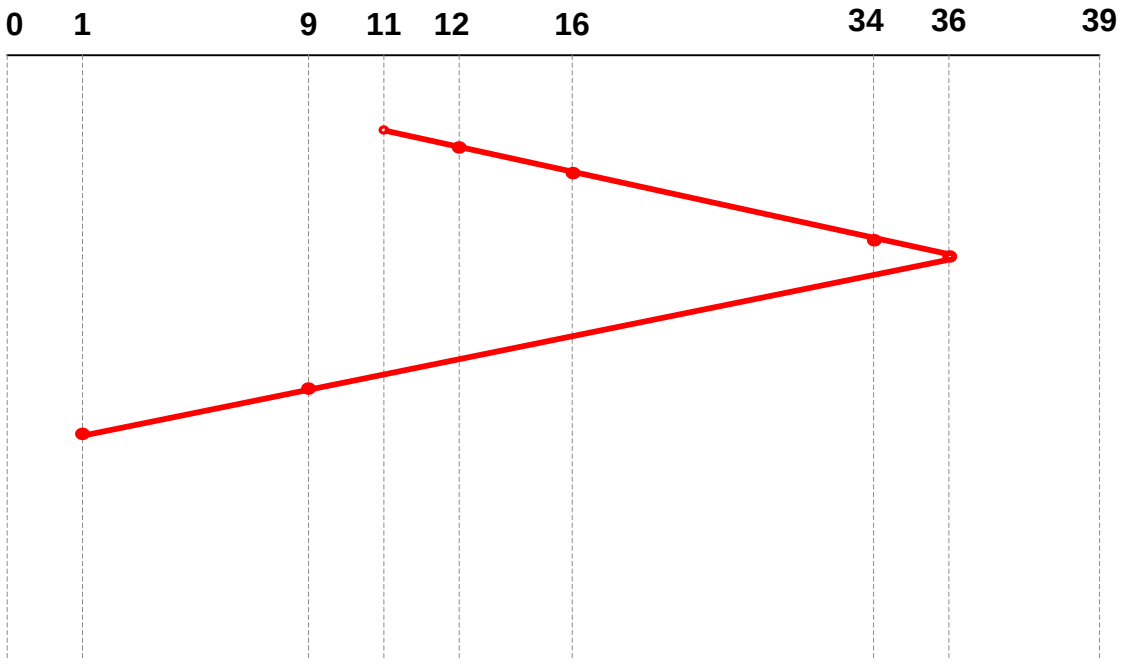
Current Cylinder Position: **11**

Cylinder Request: **1, 36, 16, 34, 9, 12**

Total Number of Cylinders: **40**

Find Total Arm Movement.

If 1 seek takes 5 ms then find total seek time for LOOK algorithm.



Total Arm Movement = 25 + 35

= 60 Cylinders

Total Seek Time = 60 * 6

= 360 ms

6. C-LOOK (Circular LOOK)

- It works in the same way as SCAN with little changes as follows:
- "After reaching to the last request in one direction, it directly jumps to the last request in the opposite direction, without serving any request along that path".

Example:

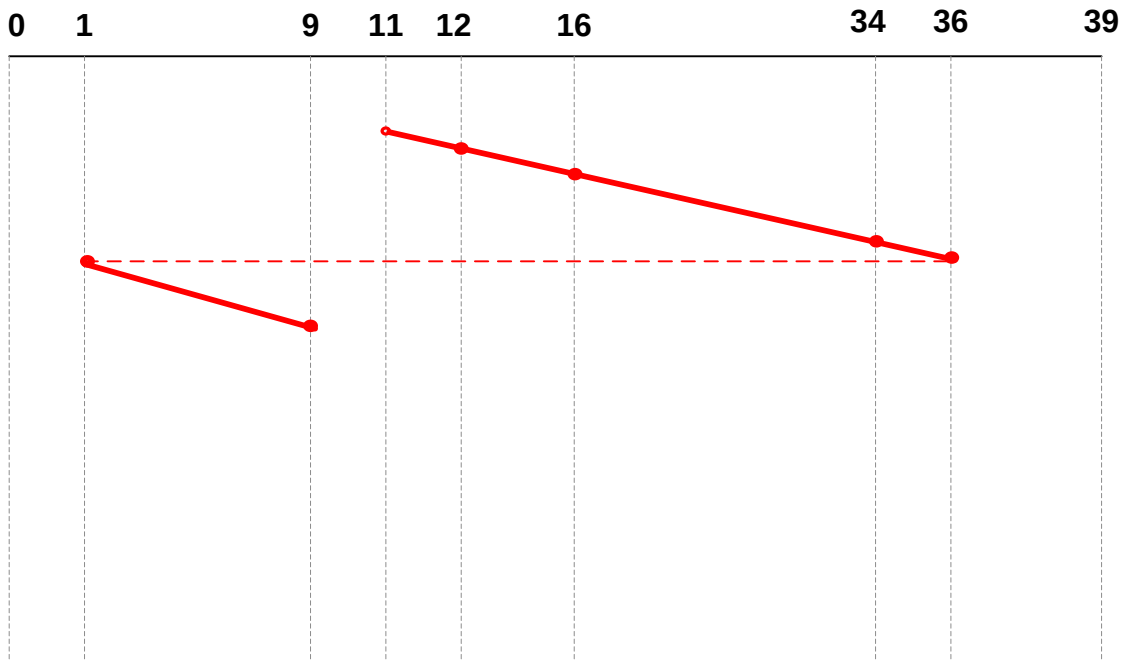
Current Cylinder Position: **11**

Cylinder Request: **1, 36, 16, 34, 9, 12**

Total Number of Cylinders: **40**

Find Total Arm Movement.

If 1 seek takes 5 ms then find total seek time for C-Look algorithm.



Total Arm Movement = 25 + 8

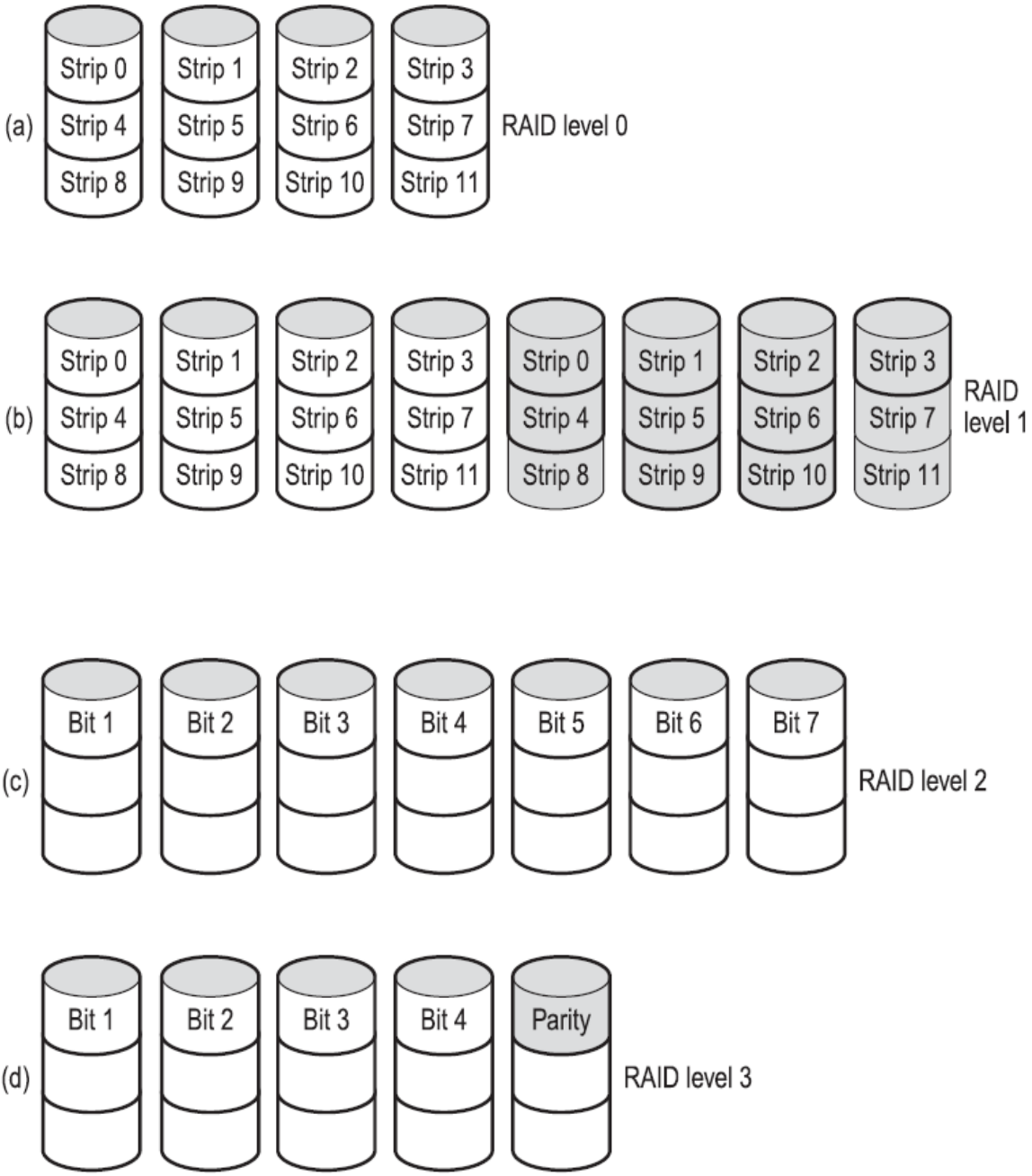
= 33 Cylinders

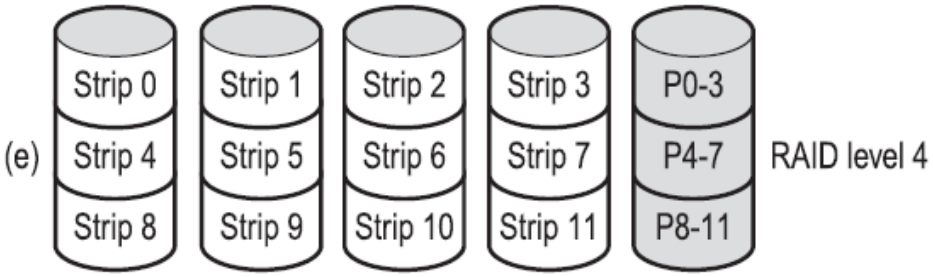
Total Seek Time = 33 * 6

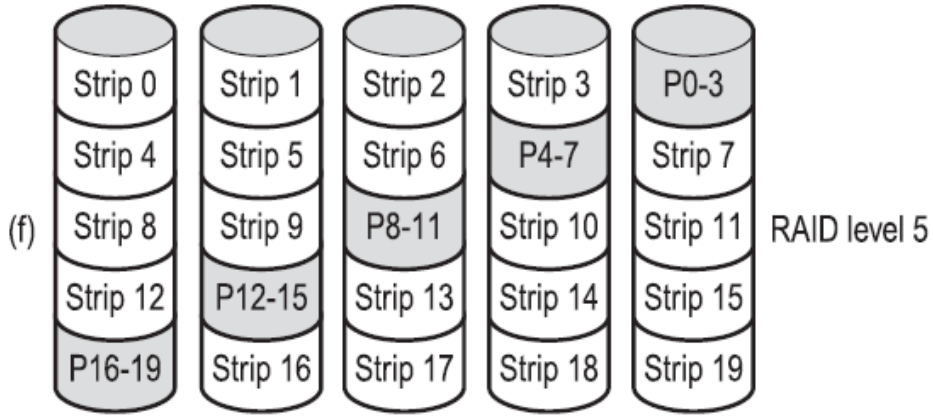
= 198 ms

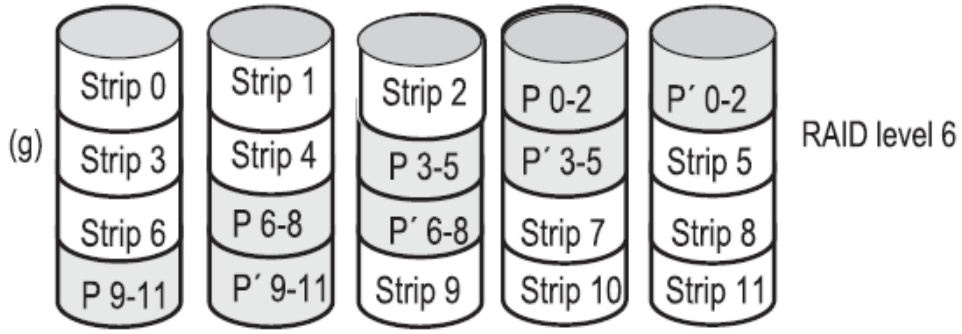
6) RAID Levels

RAID is **Redundant Array of Independent Disks**. Parallel processing is being used to speed up CPU performance. Nowadays, most manufacturers refer to the seven standard configurations as RAID level 0 through RAID level 6 to achieve parallel processing. Here actually there is no hierarchy but the term "level" simply indicates seven different organizations possible.









RAID Level 0

- It consists of viewing the virtual single disk (simulated by the RAID) as being divided up into strips of *k* sectors each, with
  sectors **0** to ***k* - 1** being strip 0,
  sectors ***k*** to **2*k* - 1** strip 1, and so on.
- The RAID level 0 organization writes consecutive strips over the drives in round-robin fashion, as shown in figure.
- Distributing data over multiple drives like this is called **striping**.

- For example, if the software issues a command to read a data block consisting of four consecutive strips starting at a strip boundary, the RAID controller will break this command up into four separate commands, one for each of the four disks, and have them operate in parallel.
- Thus we have parallel I/O. (without the software knowing about it).
- RAID level 0 works best with large requests, the bigger the better.
- Performance is excellent and the implementation is straightforward.
- Disadvantage of RAID level 0 is that the reliability is potentially worse than having a SLED (Single Large Expensive Disk).

RAID Level 1

- The next option, RAID level 1 is a true RAID.
- It duplicates all the disks, so there are four primary disks and four backup disks.
- On a write, every strip is written twice.
- On a read, either copy can be used.
- Consequently, write performance is no better than for a single drive, but read performance can be up to twice as good.
- Fault tolerance is excellent: if a drive crashes, the copy is simply used instead.
- Recovery consists of simply installing a new drive and copying the entire backup drive to it.

RAID Level 2

- RAID level 2 works on a word basis, possibly even a byte basis.
- Imagine splitting each byte into a pair of 4-bit nibbles, then adding a Hamming code to each one to form a 7-bit word (of which bits 1, 2, and 4 were parity bits).

- All seven drives need to be synchronized in terms of arm position and rotational position.

RAID Level 3

- RAID level 3 is a simplified version of RAID level 2.
- Here a single parity bit is computed for each data word and written to a parity drive.
- As in RAID level 2, the drives must be exactly synchronized, since individual data words are spread over multiple drives.
- In the case of a drive crashing, it provides full 1-bit error correction since the position of the bad bit is known.
- In the event that a drive crashes to recover all the word in the drive, parity disk can be checked. If a word has a parity error, the bit from the dead drive must have been a 1, so it is corrected.
- Although both RAID levels 2 and 3 offer very high data rates, the number of separate I/O requests per second they can handle is no better than for a single drive.

RAID Level 4

- RAID levels 4 and 5 work with strips again, not individual words with parity, and do not require synchronized drives.
- RAID level 4 is like RAID level 0, with a strip-for-strip parity written onto an extra drive.
- For example, if each strip is *k* bytes long, all the strips are EXCLUSIVE ORed together, resulting in a parity strip *k* bytes long.
- If a drive crashes, the lost bytes can be recomputed from the parity drive by reading the entire set of drives.
- This design protects against the loss of a drive but performs poorly for small updates. If one sector is changed, it is necessary to read

all the drives in order to recalculate the parity, which must then be rewritten.

- As a consequence of the heavy load on the parity drive, it may become a bottleneck.

RAID Level 5

- The bottleneck problem of level 4 is eliminated in RAID level 5 by distributing the parity bits uniformly over all the drives, round-robin fashion.
- However, in the event of a drive crash, reconstructing the contents of the failed drive is a complex process.

RAID Level 6

- Raid level 6 is similar to RAID level 5, except that an additional parity block is used.
- In other words, the data is striped across the disks with two parity blocks instead of one.
- As a result, writes are bit more expensive because of the parity calculations, but reading operation offers more reliability.

Reference Books:

1. "Operating System Concepts" by Silberschatz, Peter B. Galvin and Greg Gagne
2. "Modern Operating Systems" by Andrew S Tanenbaum