Disk Management

Disk Management is used to manage the drives installed in a computer - like hard disk drives (internal and external), optical disk drives, and flash drives. Management and ordering of disk access requests is important. The major issues of Disk management are:

- Huge speed gap between memory and disk.
- Disk throughput is extremely sensitive to disk Request order disk Scheduling algorithm

Also some other issues are disk formatting and reliability of disk.

1. Magnetic disk

Magnetic disks provide the bulk of secondary storage for modern computer systems. Conceptually, disks are relatively simple (Figure 1.1). Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 5.25 inches. The two surfaces of a platter are covered with a magnetic material. We store information by recording it magnetically on the platters.

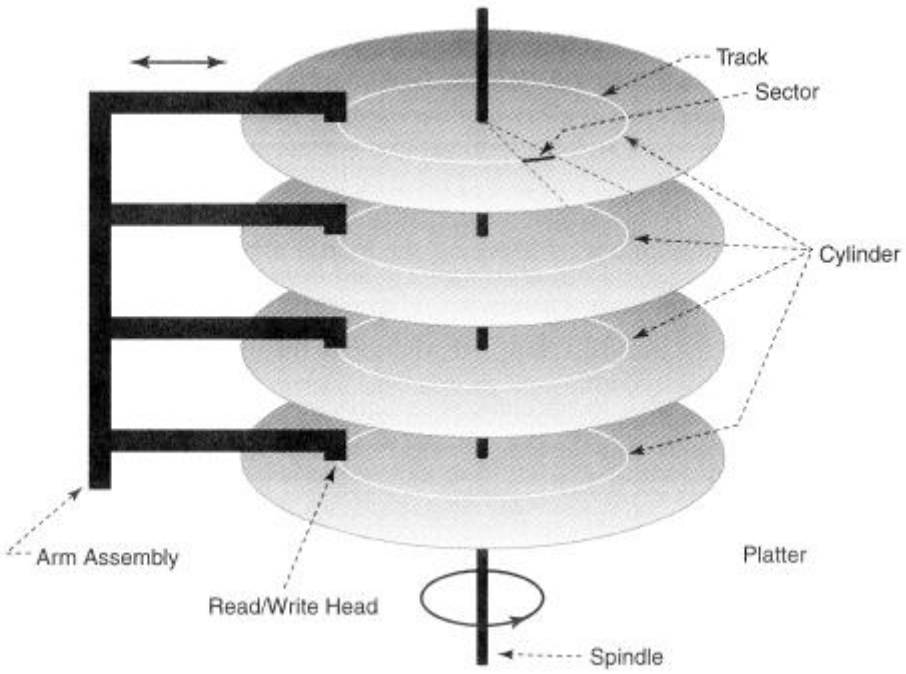


Fig. 1.1

A read-write head "flies" 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 concentric cylinders in a disk drive, and each track may contain hundreds of sectors. The storage capacity of common disk drives is measured in gigabytes.

Disk drives are addressed as large 1-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 be low-level formatted to have a different logical block size, such as 1,024 bytes. 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 (top platter) 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. By using this mapping, we can convert a logical block number into an old-style disk address that consists of a cylinder number, a track number within that cylinder, and a sector number within that track.
Modern disks pack many more sectors into outer cylinders than inner ones, using one of two approaches:

- With ***Constant Linear Velocity, CLV,*** the density of bits is uniform from cylinder to cylinder. Because there are more sectors in outer cylinders, the disk spins slower when reading those cylinders, causing the rate of bits passing under the read-write head to remain constant. This is the approach used by modern CDs and DVDs.
- With ***Constant Angular Velocity, CAV,*** the disk rotates at a constant angular speed, with the bit density decreasing on outer cylinders. (These disks would have a constant number of sectors per track on all cylinders.)

1.1 Some terminology

Disk Access Time: Two major components of disk access time.
Seek time (T_s) : It is the time for the disk to move the heads to the cylinder containing the desired sector. Typically 5-10 milliseconds
Rotational latency: is the additional time waiting for the disk to rotate the desired sector to the disk head. Typically, 2-4 milliseconds
Read/write time or transfer time: it is the actual time to transfer a block from disk to computer which is less than a millisecond
Head crash: Disk head flies on an extremely thin cushion of air (measured in microns), there is a danger that the head will make contact with the disk surface. Although the disk platters are coated with a thin protective layer, sometimes the head will damage the magnetic surface. This accident is called a **head crash.** A head crash normally cannot be repaired; the entire disk must be replaced.
Removable Disk: A disk can be **removable,** allowing different disks to be mounted as needed. Removable magnetic disks generally consist of one platter, held in a plastic case to prevent damage while not in the disk drive. **Floppy disks** are inexpensive removable magnetic disks.
Total average access time (T_a)=

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

Where, **T_s= seek time, r= rotational speed and b= no. bytes per block and N = no of byte per track.**

1.2 Disk Attachment

Disk drives are connected to the computer via a cable known as the ***I/O Bus.*** Some of the common interface formats include Enhanced Integrated Drive Electronics, EIDE; Advanced Technology Attachment, ATA; Serial ATA, SATA, Universal Serial Bus, USB; Fiber Channel, FC, and Small Computer Systems Interface, SCSI.
The ***host controller*** is at the computer end of the I/O bus, and the ***disk controller*** is built into the disk itself. The CPU issues commands to the host controller via I/O ports. Data is transferred between the magnetic surface and onboard ***cache*** by the disk controller, and then the data is transferred from that cache to the host controller and the motherboard memory at electronic speeds.
The most common interfaces are IDE or ATA, each of which allow up to two drives per host controller. SATA is similar with simpler cabling. High end workstations or other

systems in need of larger number of disks typically use SCSI disks. The SCSI standard supports up to 16 ***targets*** on each SCSI bus, one of which is generally the host adapter and the other 15 of which can be disk or tape drives. A SCSI target is usually a single drive, but the standard also supports up to 8 ***units*** within each target. These would generally be used for accessing individual disks within a RAID (Redundant Array of Inexpensive Disks) array. The SCSI standard also supports multiple host adapters in a single computer, i.e. multiple SCSI busses. Modern advancements in SCSI include "fast" and "wide" versions, as well as SCSI-2. SCSI cables may be either 50 or 68 conductors. SCSI devices may be external as well as internal.

FC is a high-speed serial architecture that can operate over optical fiber or four-conductor copper wires, and has two variants:

- A large switched fabric having a 24-bit address space. This variant allows for multiple devices and multiple hosts to interconnect, forming the basis for the ***storage-area networks, SANs.***.
- The ***arbitrated loop, FC-AL,*** which can address up to 126 devices (drives and controllers.)

A wide variety of storage devices are suitable for use as host-attached storage. Among these are hard disk drives, RAID arrays, and CD, DVD, and tape drives. The I/O commands that initiate data transfers to a host-attached storage device are reads and writes of logical data blocks directed to specifically identified storage units (such as bus ID, SCSI ID, and target logical unit).

1.2.1 Network-Attached Storage

Network attached storage connects storage devices to computers using a remote procedure call, RPC, interface, typically with something like NFS file system mounts. This is convenient for allowing several computers in a group common access and naming conventions for shared storage. NAS can be implemented using SCSI cabling, or ***ISCSI*** uses Internet protocols and standard network connections, allowing long-distance remote access to shared files. NAS allows computers to easily share data storage, but tends to be less efficient than standard host-attached storage.

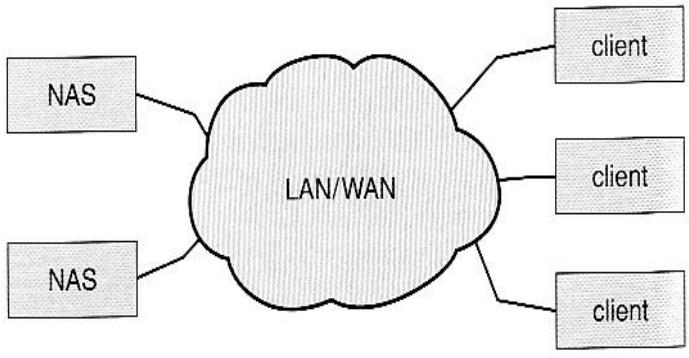


Figure Network-attached storage.

The remote procedure calls (RPCs) are carried via TCP or UDP over an IP network - usually the same local-area network (LAN) that carries all data traffic to the clients. The network attached storage unit is usually implemented as a RAID array with software that implements the RPC interface. It is easiest to think of NAS as simply another storage-access protocol. For example, rather than using a SCSI device driver and SCSI protocols to access storage, a system using NAS would use RPC over TCP/IP.

1.2.2 Storage-Area Network

A ***Storage-Area Network, SAN,*** connects computers and storage devices in a network, using storage protocols instead of network protocols. One advantage of this is that storage access does not tie up regular networking bandwidth. SAN is very flexible and dynamic, allowing hosts and devices to attach and detach on the fly. SAN is also controllable, allowing restricted access to certain hosts and devices. A storage-area network (SAN) is a private network (using storage protocols rather than networking protocols) connecting servers and storage units, as shown in Figure.

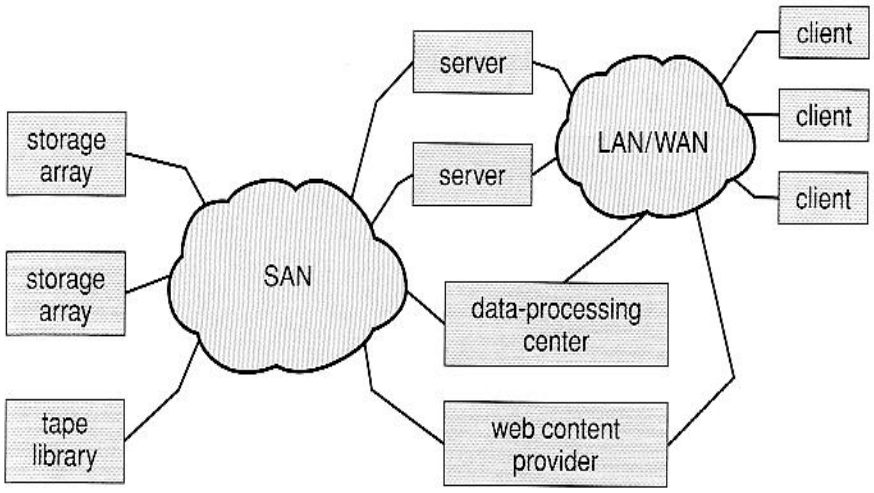


Figure Storage-area network.

The power of a SAN lies in its flexibility. Multiple hosts and multiple storage arrays can attach to the same SAN, and storage can be dynamically allocated to hosts. A SAN switch allows or prohibits access between the hosts and the storage. SANs typically have more ports, and less expensive ports, than storage arrays. FC is the most common to SAN interconnect.

2. Disk Scheduling

One of the responsibilities of the OS is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth.

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

We can improve both the access time and the bandwidth by scheduling the servicing of disk I/O requests in a good order.

For a multiprogramming system with many processes, the disk queue may often have several pending requests. Disk-scheduling algorithms decide which pending request to service next.

There different disk scheduling algorithm is available. Like as follows

- FCFS Scheduling
- SSTF Scheduling
- SCAN Scheduling
- C-SCAN Scheduling
- LOOK Scheduling

2.1 *First-Come First-Serve*

FCFS is simple and intrinsically fair, but not very efficient.

Consider, for example, a disk queue with requests for I/O to blocks on cylinders in that order; 98,183,37,122,14,124,65,67.

If the disk head is initially at cylinder 53.It will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67. A total head movement would be of 640 cylinders.

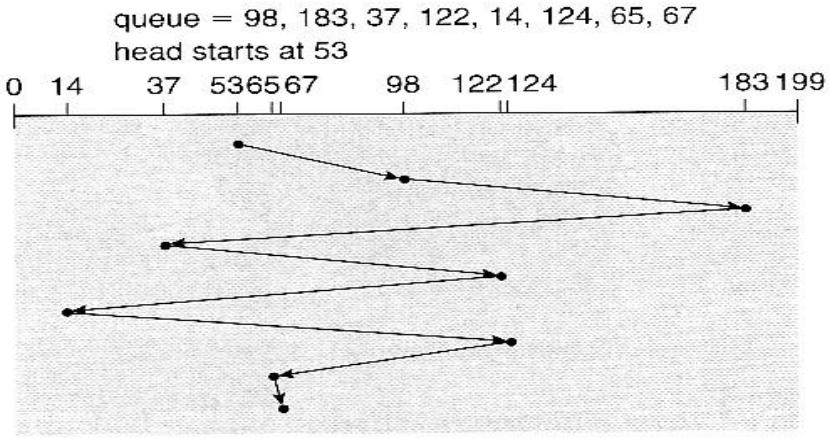


Figure FCFS disk scheduling.

Consider in the following sequence the wild swing from cylinder 122 to 14 and then back to 124. If the requests for cylinders 37 and 14 could be serviced together, before or after the requests at 122 and 124, the total head movement could be decreased substantially, and performance could be thereby improved**.**

2.2 SSTF Scheduling

It seems reasonable to service all the requests close to the current head position before moving the head far away to service other requests. This assumption is the basis for the shortest-seek-time-first (SSTF) algorithm.

The SSTF algorithm selects the request with the minimum seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.

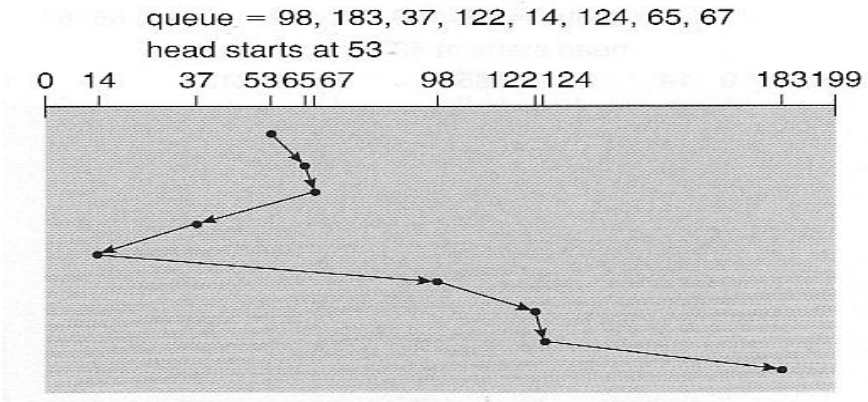


Figure SSTF disk scheduling.

For example: 53, 65, 67, 37, 14, 98, 122, 124 183 is the request queue. The closest request to the initial head position (53) is at cylinder 65. Once we are at cylinder 65, the next closest

request is at cylinder 67. From there, the request at cylinder 37 is closer than the one at 98, so 37 is served next. Continuing, we service the request at cylinder 14, then 98,122, 124, and finally 183 as shown in figure. This scheduling method results in a total head movement of only 236 cylinders-little more than one-third of the distance needed for FCFS scheduling of this request queue. This algorithm gives a substantial improvement in performance.

SSTF scheduling is essentially a form of shortest-job-first (SJF) scheduling; and like SJF scheduling, it may cause starvation of some requests (steady supply of shorter seek time requests). Although the SSTF algorithm is a substantial improvement over the FCFS algorithm, it is not optimal.

2.3 SCAN Scheduling

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

The SCAN algorithm is sometimes called the elevator algorithm, since the disk arms behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way. For our example request queue 37, 14, 65, 67, 98, 122, 124, 183

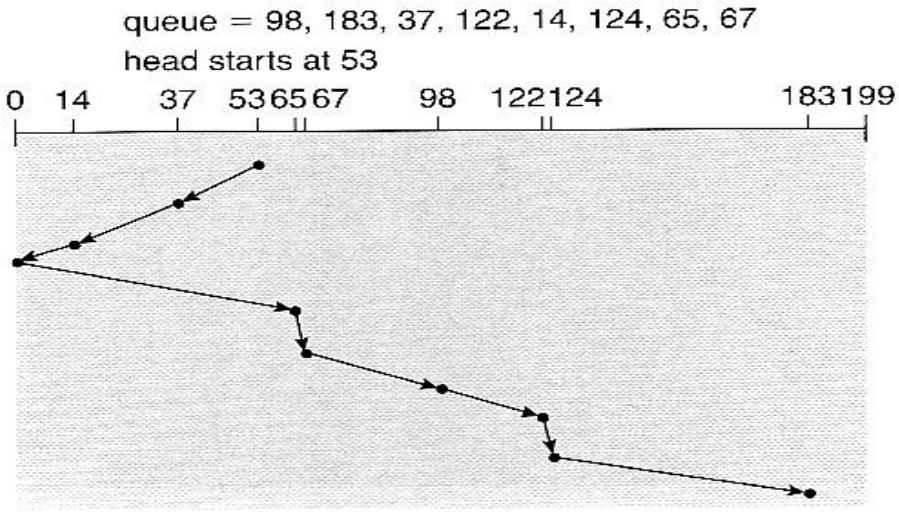


Figure SCAN disk scheduling.

If a request arrives in the queue just in front of the head, it will be serviced almost immediately. If a request arriving just behind the head will have to wait until the arm moves to the end of the disk, reverses direction, and comes back.

2.4 C-SCAN Scheduling

Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, CSCAN moves the head from one end of the disk to the other, servicing requests along the way.

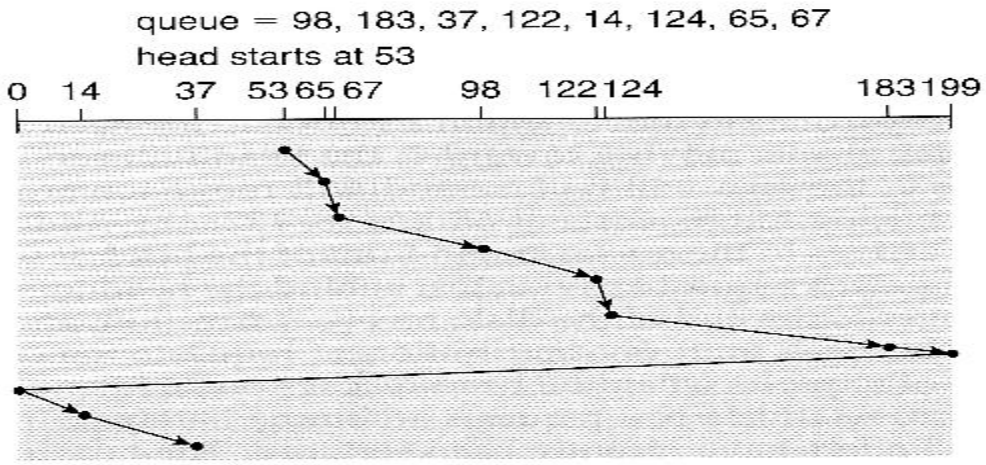


Figure C-SCAN disk scheduling.

2.5 LOOK Scheduling

As we described them, both SCAN and C-SCAN move the disk arm across the full width of the disk. In practice, neither algorithm is often implemented this way. More commonly, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk. Versions of SCAN and C-SCAN that follow this pattern are called LOOK and C-LOOK scheduling, because they look for a request before continuing to move in a given direction

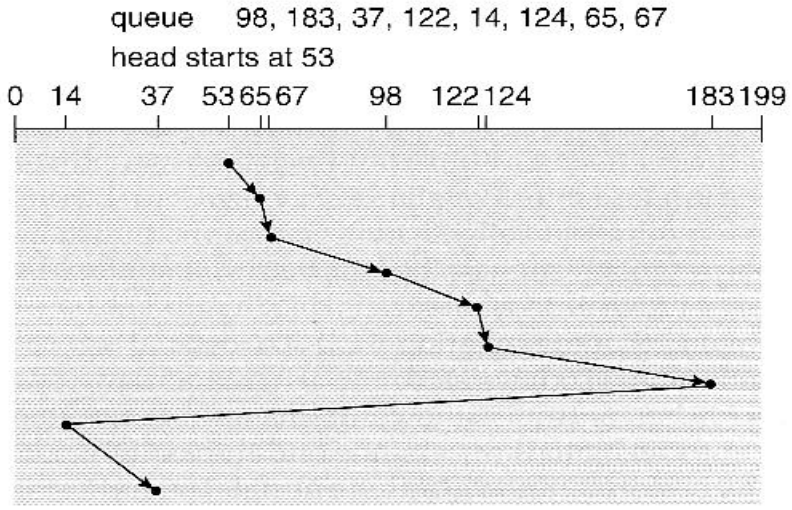


Figure C-LOOK disk scheduling.

Selecting a Disk-Scheduling Algorithm

SSTF is common and has a natural appeal. SCAN and C-SCAN perform better for systems that place a heavy load on the disk. Performance depends on the number and types of requests. Requests for disk service can be influenced by the file allocation method. The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary. Either SSTF or C-LOOK is a reasonable choice for the default algorithm (depending on load)

3. Disk Formatting

A new magnetic disk is a blank slate: It is just a platter of a magnetic recording material. Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called **low-level formatting**, or **physical formatting**.

Low-level formatting fills the disk with a special data structure for each sector. The data structure for a sector typically consists of a header, a data area (usually 512 bytes in size), and a trailer. The header and trailer contain information used by the disk controller, such as a sector number and an **error-correcting code** (ECC). When the controller **writes** a sector of data during normal I/O, the ECC is updated with a value calculated from all the bytes in the data area. When the sector is **read**, the ECC is recalculated and is compared with the stored value. If the stored and calculated numbers are different, this mismatch indicates that the data area of the sector has become corrupted and that the disk sector may be bad. The controller automatically does the ECC processing whenever a sector is read or written. To use a disk to hold files, the OS still needs to record its own data structures on the disk. It does so in two steps. The first step is to partition the disk into one or more groups of cylinders. The OS can treat each partition as though it were a separate disk.

For instance, one partition can hold a copy of the OS's executable code, while another holds user files. After partitioning, the second step is **logical formatting** (or creation of a file system). In this step, the OS stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space (a FAT or inodes) and an initial empty directory.

To increase efficiency, most file systems group blocks together into larger chunks, frequently called **clusters.** Disk I/O is done via blocks, but file system I /O is done via clusters, effectively assuring that I/O has more sequential-access and fewer random-access characteristics.

3.1 Boot Block

Computer ROM contains a ***bootstrap*** program (OS independent) with just enough code to find the first sector on the first hard drive on the first controller, load that sector into memory, and transfer control over to it. (The ROM bootstrap program may look in floppy and/or CD drives before accessing the hard drive, and is smart enough to recognize whether it has found valid boot code or not.)

The first sector on the hard drive is known as the ***Master Boot Record, MBR,*** and contains a very small amount of code in addition to the ***partition table.*** The partition table documents how the disk is partitioned into logical disks, and indicates specifically which partition is the ***active*** or ***boot*** partition. The boot program then looks to the active partition to find an operating system, possibly loading up a slightly larger / more advanced boot program along the way. In a ***dual-boot*** (or larger multi-boot) system, the user may be given a choice of which operating system to boot, with a default action to be taken in the event of no response within some time frame. Once the kernel is found by the boot program, it is loaded into memory and then control is transferred over to the OS. The kernel will normally continue the boot process by initializing all important kernel data structures, launching important system services (e.g. network daemons, sched, init, etc.), and finally providing one or more login prompts. Boot options at this stage may include ***single-user*** a.k.a. ***maintenance*** or ***safe*** modes, in which very few system services are started - These modes are designed for system administrators to repair problems or otherwise maintain the system.

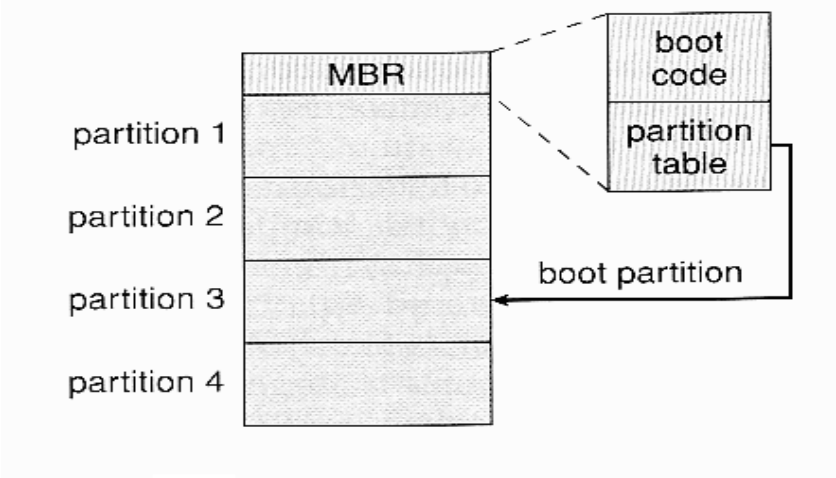


Figure Booting from disk in Windows 2000.

3.2 Bad Blocks

No disk can be manufactured to 100% perfection, and all physical objects wear out over time. For these reasons all disks are shipped with a few bad blocks, and additional blocks can be expected to go bad slowly over time. If a large number of blocks go bad then the entire disk will need to be replaced, but a few here and there can be handled through other means.

In the old days, bad blocks had to be checked for manually. Formatting of the disk or running certain disk-analysis tools would identify bad blocks, and attempt to read the data off of them one last time through repeated tries. Then the bad blocks would be mapped out and taken out of future service. Sometimes the data could be recovered, and sometimes it was lost forever. (Disk analysis tools could be either destructive or non-destructive.)

Modern disk controllers make much better use of the error-correcting codes, so that bad blocks can be detected earlier and the data usually recovered. (Recall that blocks are tested with every write as well as with every read, so often errors can be detected before the write operation is complete, and the data simply written to a different sector instead.)

Note that re-mapping of sectors from their normal linear progression can throw off the disk scheduling optimization of the OS, especially if the replacement sector is physically far away from the sector it is replacing. For this reason most disks normally keep a few spare sectors on each cylinder, as well as at least one spare cylinder. Whenever possible a bad sector will be mapped to another sector on the same cylinder, or at least a cylinder as close as possible. ***Sector slipping*** may also be performed, in which all sectors between the bad sector and the replacement sector are moved down by one, so that the linear progression of sector numbers can be maintained. If the data on a bad block cannot be recovered, then a ***hard error*** has occurred, which requires replacing the file(s) from backups, or rebuilding them from scratch.

4. RAID Structure

The general idea behind RAID is to employ a group of hard drives together with some form of duplication, either to increase reliability or to speed up operations, (or sometimes both.)

RAID originally stood for ***Redundant Array of Inexpensive Disks,*** and was designed to use a bunch of cheap small disks in place of one or two larger more expensive ones.

Today RAID systems employ large possibly expensive disks as their components, switching the definition to ***Independent*** disks.

4.1 Benefits of RAID

Improvement of Reliability via Redundancy

- Increasing disks on a system actually ***decreases*** the **Mean Time To Failure, MTTF** of the system.
- If, however, the same data was copied onto multiple disks, then the data would not be lost unless **both** (or all) copies of the data were damaged simultaneously, which a much lower probability than for a single disk is going bad. More specifically, the second disk would have to go bad before the first disk was repaired, which brings the **Mean Time To Repair** into play. This is the basic idea behind disk ***mirroring***, in which a system contains identical data on two or more disks.

Improvement in Performance via Parallelism

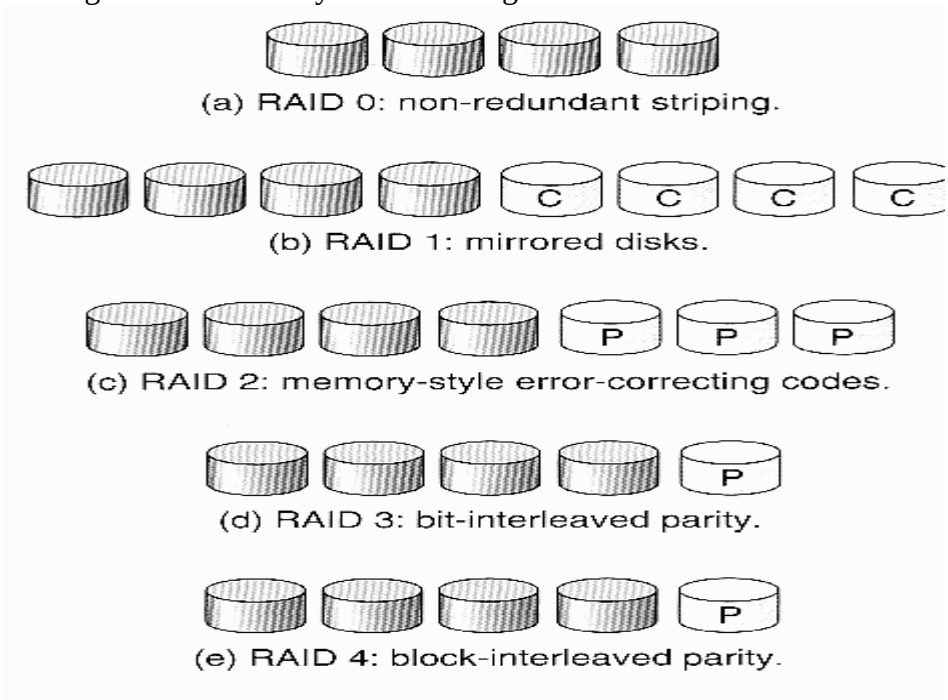
- There is also a performance benefit to mirroring, particularly with respect to reads. Since every block of data is duplicated on multiple disks, read operations can be satisfied from any available copy, and multiple disks can be reading different data blocks simultaneously in parallel.
- Another way of improving disk access time is with ***striping***, which basically means spreading data out across multiple disks that can be accessed simultaneously.
    - With ***bit-level striping*** the bits of each byte are striped across multiple disks. For example if 8 disks were involved, then each 8-bit byte would be read in parallel by 8 heads on separate disks. A single disk read would access 8 * 512 bytes = 4K worth of data in the time normally required to read 512 bytes. Similarly if 4 disks were involved, then two bits of each byte could be stored on each disk, for 2K worth of disk access per read or write operation.
    - ***Block-level striping*** spreads a file system across multiple disks on a block-by-block basis, so if block N were located on disk 0, then block N + 1 would be on disk 1, and so on. This is particularly useful when file systems are accessed in ***clusters*** of physical blocks. Other striping possibilities exist, with block-level striping being the most common.

4.2 RAID Levels

Mirroring provides reliability but is expensive; Striping improves performance, but does not improve reliability. Accordingly there are a number of different schemes that combine the principals of mirroring and striping in different ways, in order to balance reliability versus performance versus cost. These are described by different ***RAID levels***, as follows: (In the diagram that follows, "C" indicates a copy, and "P" indicates parity, i.e. checksum bits.)

1. ***Raid Level 0*** **-** This level includes striping only, with no mirroring.
2. ***Raid Level 1*** **-** This level includes mirroring only, no striping.
3. ***Raid Level 2*** **-** This level stores error-correcting codes on additional disks, allowing for any damaged data to be reconstructed by subtraction from the remaining undamaged data. Note that this scheme requires only three extra disks to protect 4 disks worth of data, as opposed to full mirroring. (The number of disks required is a function of the error-correcting algorithms, and the means by which the particular bad bit(s) is(are) identified.)

4. ***Raid Level 3 -*** This level is similar to level 2, except that it takes advantage of the fact that each disk is still doing its own error-detection, so that when an error occurs, there is no question about which disk in the array has the bad data. As a result a single parity bit is all that is needed to recover the lost data from an array of disks. Level 3 also includes striping, which improves performance. The downside with the parity approach is that every disk must take part in every disk access, and the parity bits must be constantly calculated and checked, reducing performance. Hardware-level parity calculations and NVRAM cache can help with both of those issues. In practice level 3 is greatly preferred over level 2.
5. ***Raid Level 4 -*** This level is similar to level 3, employing block-level striping instead of bit-level striping. The benefits are that multiple blocks can be read independently, and changes to a block only require writing two blocks (data and parity) rather than involving all disks. Note that new disks can be added seamlessly to the system provided they are initialized to all zeros, as this does not affect the parity results.
6. ***Raid Level 5 -*** This level is similar to level 4, except the parity blocks are distributed over all disks, thereby more evenly balancing the load on the system. For any given block on the disk(s), one of the disks will hold the parity information for that block and the other N-1 disks will hold the data. Note that the same disk cannot hold both data and parity for the same block, as both would be lost in the event of a disk crash.
7. ***Raid Level 6 -*** This level extends raid level 5 by storing multiple bits of error-recovery codes, for each bit position of data, rather than a single parity bit. In the example shown below 2 bits of ECC are stored for every 4 bits of data, allowing data recovery in the face of up to two simultaneous disk failures. Note that this still involves only 50% increase in storage needs, as opposed to 100% for simple mirroring which could only tolerate a single disk failure.



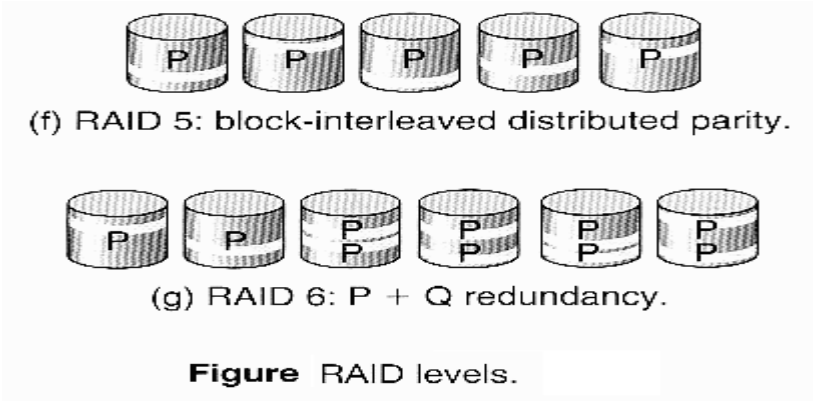


Figure RAID levels.

There are also two RAID levels which combine RAID levels 0 and 1 (striping and mirroring) in different combinations, designed to provide both performance and reliability at the expense of increased cost.

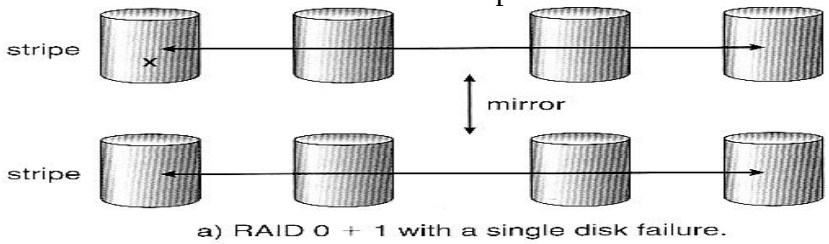
- **RAID level 0 + 1** disks are first striped, and then the striped disks mirrored to another set. This level generally provides better performance than RAID level 5.
- **RAID level 1 + 0** mirrors disks in pairs, and then stripes the mirrored pairs. The storage capacity, performance, etc. are all the same, but there is an advantage to this approach in the event of multiple disk failures, as illustrated below:.

In diagram (a) below, the 8 disks have been divided into two sets of four, each of which is striped, and then one stripe set is used to mirror the other set.

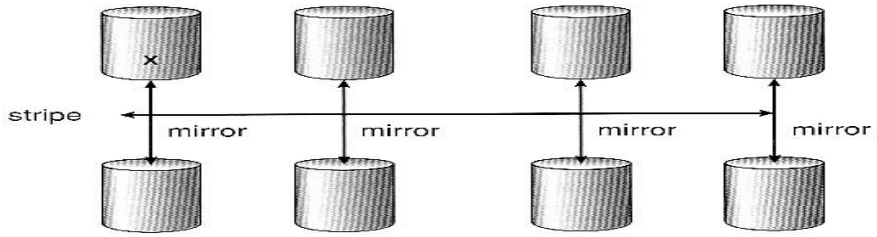
- If a single disk fails, it wipes out the entire stripe set, but the system can keep on functioning using the remaining set. However if a second disk from the other stripe set now fails, then the entire system is lost, as a result of two disk failures.

In diagram (b), the same 8 disks are divided into four sets of two, each of which is mirrored, and then the file system is striped across the four sets of mirrored disks.

- If a single disk fails, then that mirror set is reduced to a single disk, but the system rolls on, and the other three mirror sets continue mirroring. Now if a second disk fails, (that is not the mirror of the already failed disk), then another one of the mirror sets is reduced to a single disk, but the system can continue without data loss. In fact the second arrangement could handle as many as four simultaneously failed disks, as long as no two of them were from the same mirror pair.



a) RAID 0 + 1 with a single disk failure.



b) RAID 1 + 0 with a single disk failure.