I/O SYSTEMS

PROCESSOR – I/O INTERFACES

- Connections exist between processors, memory, and IO devices. The OS must manage these connections.
  <<< FIGURE 12.1 >>> Shows a good overview of possible mechanisms.

- An I/O request has a number of steps that can be seen in
  <<< FIGURE 12.10>>>

PHYSICAL CHARACTERISTICS

- A disk can be viewed as an array of blocks. In fact, a file system will want to view it at that logical level.

- However, there's a mapping scheme from logical block address B, to physical address (represented by a track / sector pair.)

- The smallest storage allocation is a block - nothing smaller can be placed on the disk. This results in unused space (internal fragmentation) on the disk, since quite often the data being placed on the disk doesn't need a whole block.

DISK SCHEDULING

- The components making up disk service time include:

  time = setup + seek + rotation time + transfer + wrap-up

- The methods discussed below try to optimize seek time but make no attempt to account for the total time. The ideal method would optimize the total time and many controllers are now able to accomplish this.

FCFS: Do requests in the order they come in: makes no attempt to minimize head motion. This may not give the best (average) service. **<<< FIGURE 13.1 >>>**

SHORTEST SEEK TIME FIRST:
Select the request with cylinder closest to the current position. This is not the optimal or shortest path. Can cause starvation of some requests Substantial improvement over FCFS.
<<< FIGURE 13.2 >>>

SCAN: Move from one end of the disk to the other, servicing requests as encountered; then reverse at end. Analogy: like shoveling snow while it's snowing. **<<< FIGURE 13.3 >>>**

Upon reversing, it's those requests at the far end which have waited the longest, but will still need to wait until the arm gets to that end.

C-SCAN: Returns to lowest cylinder at end of each scan. This results in a more uniform wait time.

LOOK: (In practice, both SCAN and C-SCAN stop at the last request, rather than actually traveling to the end of the disk.) This is called LOOK and C-LOOK. **<<< FIGURE 13.5 >>>**

SELECTING A SCHEDULING ALGORITHM

- Evaluate the performance based on usage.
- How the files were allocated becomes important (sequential, indexed, etc.)
- Locality of reference also matters.
- If the request queue is short, algorithm doesn't matter.
- Today's disks do all this scheduling in their own firmware.

DISK MANAGEMENT:

Disk formatting Creates a logical disk from the raw disk. Includes setting aside chunks of the disk for booting, bad blocks, etc. Also provides information needed by the driver to understand its positioning.

Boot block That location on the disk that is accessed when trying to boot the operating system. It's a well-known location that contains the code that understands how to get at the operating system - generally this code has a rudimentary knowledge of the file system.

Bad blocks The driver knows how to compensate for a bad block on the disk. It does this by putting a pointer, at the location of the bad block, indicating where a good copy of the data can be found.

Swap Space Management The Operating System requires a contiguous space where it knows that disk blocks have been reserved for paging. This space is needed because a program can't be given unshared memory unless there's a backing store location for that memory.

PERFORMANCE AND RELIABILITY IMPROVEMENTS

MIRRORING One way to increase reliability is to "mirror" data on a disk. Every piece of data is maintained on two disks - disk drivers must be capable of getting data from either disk. Performance issues: a read is faster since data can be obtained from either disk - writes are slower since the data must be put on both disks.

RAID Redundant Array of Inexpensive Disks: Rather than maintain two copies of the data, maintain one copy plus parity. For example, four disks contain data, and a fifth disk holds the parity of the XOR of the four data disks. Reads slower than mirroring, writes much slower. But RAID is considerably CHEAPER than mirroring.

DISK STRIPING Disks tend to be accessed unevenly - programs ask for a number of blocks from the same file, for instance. Accesses can be distributed more evenly by spreading a file out over several disks. This works well with RAID. Thus block 0 is on disk 0, block 1 is on disk 1, block 4 is on disk 0.

Consider how to recover from a failure on these architectures.