CSCE 351
Operating System Kernels

Principles of I/O Hardware and Software

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE351

I/O Hardware

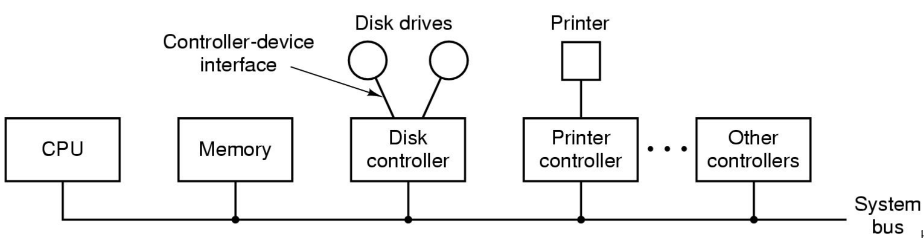
- I/O Devices
  - Block Devices
  - Character Devices
- Device Controllers
  - Memory-mapped I/O
  - I/O Ports
  - Interrupt Request Line (IRQ)
- Direct Memory Access (DMA)
- I/O Controllers and DMA
  - Disk Interleaving

I/O Devices

- Block Devices
  - » Information is stored and accessed in fixed-size blocks
  - » Block addressable, not byte addressable
  - » Common block sizes: 512 – 32,768 bytes
  - » Examples?
- Character Devices
  - » Send or receive streams of characters
  - » NOT byte or block addressable
  - » Examples?
- What about
  - » Tape drives?
  - » Clocks?
  - » Memory-mapped screens?
  - » Mice?

Device Controllers

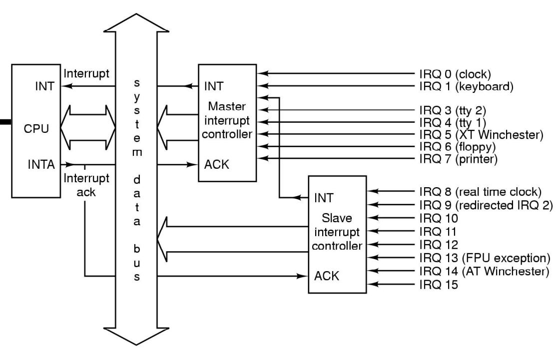
- Most I/O devices are electro-mechanical.
- The electrical component that interfaces with the CPU (actually the OS) is called the device controller or adapter.
- The Controller is the *go-between* for the OS and the device
- Controllers for PCs and embedded devices are implemented as daughter cards and inserted into the back-plane of the parentboard (or motherboard)



Interfacing with Controllers

- Memory-mapped I/O
  - » Controller Registers are part of regular address space
  - » Usually flags in special registers indicate that the memory access is for memory-mapped I/O
  - » Used by Motorola 680x0 processors
- I/O Ports
  - » Each controller is assigned a special address called a port.
- Interrupt Request Lines (IRQ)
  - » Physical input to the interrupt controller chip
  - » Signals when the device controller is ready to have its registers read/written via the I/O Ports (addresses) assigned that controller.

I/O Ports and IRQs



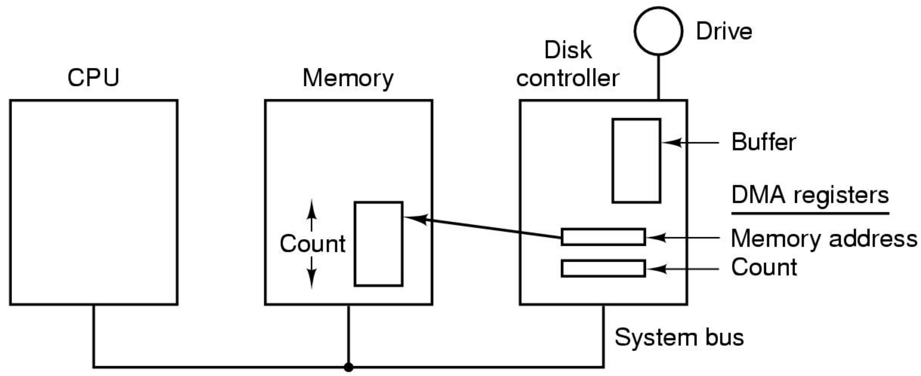
Example:

PC with MS-DOS

I/O Controller	I/O address	Hardware IRQ	DOS Interrupt Vector	MINIX Interrupt Vector
Clock	040-043	0	8	40
Keyboard	060-063	1	9	41
Hard Disk	1F0-1F7	14	118	54
Printer	378-37F	7	15	47

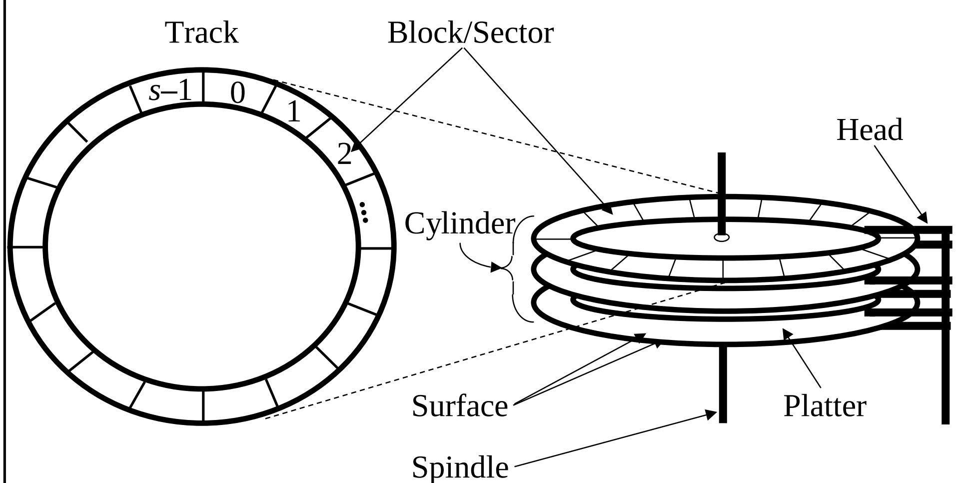
Direct Memory Access (DMA)

◆ Used mainly for block devices



Anatomy of a Disk

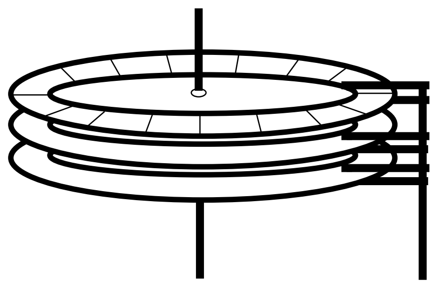
Basic components



Anatomy of a Disk

Example: Seagate 9GB Fast/Wide/Differential SCSI disk

- Specs:
  - » 12 platters
  - » 22 heads
  - » variable # of sectors/track
  - » 7,200 RPM
    - ❖ average latency: 4.2 ms.
  - » Seek times
    - ❖ track-to-track: 1 ms
    - ❖ average: 7.9 ms
  - » 40MB/s peak transfer rate
  - » 11 arms
  - » 4,735 tracks
  - » 512 bytes/sector

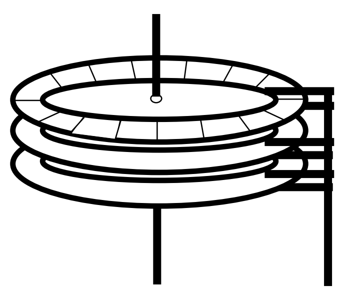


Disk Operations

Data transfer in units of sectors

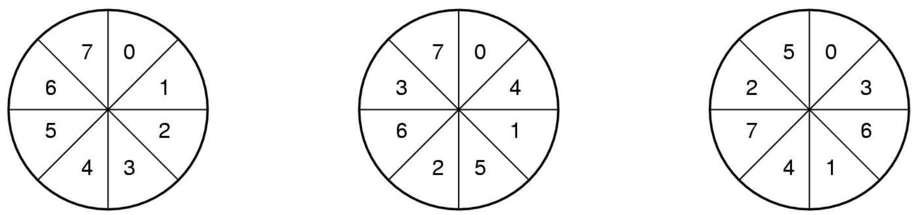
Random access devices with non-uniform access times

- Present disk with a sector address
  - » *DA* = (*drive*, *surface*, *track*, *sector*)
- Head moved to appropriate track
  - » "seek time"
- The appropriate head is enabled
- Wait for the sector to appear under the head
  - » "rotational latency"
- Read/write the sector
  - » "transfer time"



Disk Interleaving

- Blocks are often placed on the disk in non-sequential order to allow time for the DMA buffer to be transferred to main memory.



(a) No interleaving | (b) Single interleaving | (c) Double interleaving

I/O Software

- Goals of I/O Software
- Structured I/O Software
  - » Interrupt Handlers
  - » Device Drivers
  - » Device-Independent I/O SW
  - » User-Space I/O SW

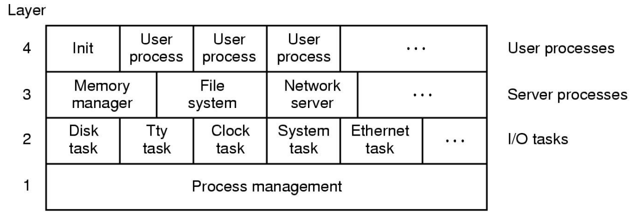
Goals of I/O Software

- Device independence
  - » Read and Write from Floppy, Disk, CD-ROM without modifying programs
- Uniform naming
  - » Names are not dependent on the specific device
    - Different devices of same type have similar name
  - » In UNIX, all I/O is integrated with the file system
- Error handling done as close to HW as possible
  - » Propagate errors up only when lower layer cannot handle it.
  - » Hide errors as much as possible—many HW errors are transient
- Synchronous read/write at application level
  - » Most I/O hardware operates asynchronously
  - » Synchronous (blocking) read/write easier to program

Structured I/O SW to meet Goals

- Structured I/O SW to meet these Goals

User-level Software
Device-Independent OS SW
Device Drivers
Interrupt Handlers



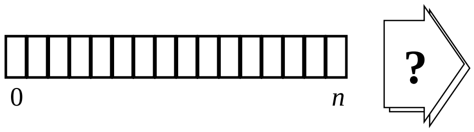
Interrupt Handlers

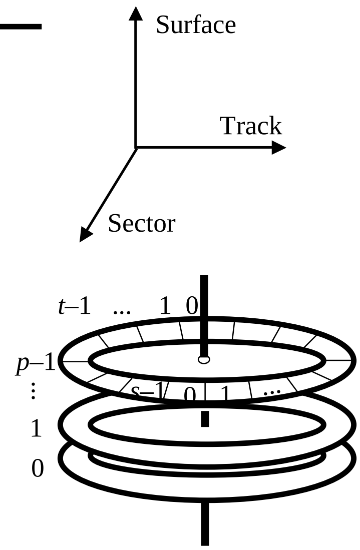
- Hidden from applications
- Used to bridge gap between asynchronous I/O hardware and synchronous read/write semantics
- Often implemented as top-half and bottom-half handlers
  - » Top-half
    - does as little as possible
    - not scheduled
  - » Bottom-half
    - closely related to (if not exactly) the device driver
    - scheduled

Device Drivers

- Device dependent code
- Each device driver handles (at most) one class of devices
- Device drivers communicate with the device controllers
  - » Only part that knows the details of the device.
  - » Hence, device dependent
- Translate device-independent (abstract) requests to device-specific commands

Device Driver
Block to Sector Mappings





- Device driver translates block requests into cylinder, track, and sector requests.

Device-Independent I/O Software

- Functions of device-independent I/O SW
  - » Uniform interfacing for device drivers
  - » Device naming
    - ❖ Mnemonic names mapped to Major and Minor device numbers
  - » Device protection
  - » Providing a device-independent block size
  - » Buffering
  - » Storage allocation on block devices
  - » Allocation and releasing dedicated devices
  - » Error Reporting
- In MINIX, most device-independent I/O SW is in the file system (layer 3).

User-Space I/O SW

- I/O Libraries (e.g., stdio) are in user-space to provide an interface to the OS resident device-independent I/O SW
  - » These routines do the formatting for the user that is such a pain to do, but everyone wants it
- Simultaneous Peripheral Operations On-Line (Spooling)
  - » A user-space print command puts a file in the spooling directory and then asks a daemon process to execute the I/O request
  - » Printing is one use of spooling.
  - » Others?
  - » Why spool I/O?

Putting It All Together