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

<table>
<thead>
<tr>
<th>I/O Controller</th>
<th>I/O address</th>
<th>Hardware IRQ</th>
<th>DOS Interrupt Vector</th>
<th>MINIX Interrupt Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>040-043</td>
<td>0</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>Keyboard</td>
<td>060-063</td>
<td>1</td>
<td>9</td>
<td>41</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>1F0-1F7</td>
<td>14</td>
<td>118</td>
<td>54</td>
</tr>
<tr>
<td>Printer</td>
<td>378-37F</td>
<td>7</td>
<td>15</td>
<td>47</td>
</tr>
</tbody>
</table>
Direct Memory Access (DMA)

- Used mainly for block devices

Anatomy of a Disk
Basic components

- Track
- Block/Sector
- Cylinder
- Surface
- Spindle
- Head
- Platter
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

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”

Random access devices with non-uniform access times
**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.

![Interleaving Diagrams](image)

(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

<table>
<thead>
<tr>
<th>User-level Software</th>
<th>Device-Independent OS SW</th>
<th>Device Drivers</th>
<th>Interrupt Handlers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Layered Architecture Diagram:

- Layer 4: User processes
- Layer 3: System server, File system, Memory manager
- Layer 2: Network server, Clock task, Disk task
- Layer 1: Process management, I/O tasks
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

```
Layer:  
  User processes
  Device-independent software
  Device drivers
  Interrupt handlers
  Hardware

I/O request

I/O reply

I/O functions
Make I/O call; format I/O; spooling
Naming, protection, blocking, buffering, allocation
Set up device registers; check status
Wake up driver when I/O completed
Perform I/O operation
```