This Unit: I/O

- I/O system structure
- Devices, controllers, and buses
- Device characteristics
  - Disks
- I/O control
  - Polling and interrupts
  - DMA
One Instance of I/O

- Have briefly seen one instance of I/O
  - **Disk**: bottom of memory hierarchy

A More General/Realistic I/O System

- A computer system
  - CPU/Memory: connected by memory bus
  - **I/O peripherals**: disks, input devices, displays, network cards, ...
    - With built-in or separate I/O (or DMA) controllers
    - All connected by a **system bus**
Disk Bandwidth: Sequential vs Random

- Disk is bandwidth-inefficient for page-sized transfers
  - Sequential vs random accesses

**Random accesses:**
- One read each disk access latency (~10ms)
- Randomly reading 4KB pages
  - 10ms is 0.01 seconds → 100 access per second
  - 4KB * 100 access/sec → 400KB/second bandwidth

**Sequential accesses:**
- Stream data from disk (no seeks)
- 128 sectors/track, 512 B/sector, 6000 RPM
  - 64KB per rotation, 100 rotation/per sec
  - 6400KB/sec → 6.4MB/sec
- Sequential access is ~10x or more bandwidth than random
  - Still no where near the 1GB/sec to 10GB/sec of memory

Increasing Disk Bandwidth

- Single disk:
  - Shorter access times (latency helps bandwidth)
  - Schedule access efficiently for multiple parallel requests
    - Reduce seek time by scheduling seeks
  - Higher RPMs
  - More sequential seeks (layout files on disk intelligently)

- More disks: **stripe data across multiple disks**
  - Increases both sequential and random access bandwidth
  - More later on these disk arrays
Disk Interfaces

- Disks talk a “language”, too
  - Much like an ISA for a processor

- **ATA/IDE**
  - Simple, one request at a time
  - Limited number of devices
  - Cheap, high volume

- **SCSI**
  - Many parallel requests
    - Split request from response
  - Many devices, high transfer rates
  - Expensive, high-end

- **Newcomers**: Serial-ATA (S-ATA) and iSCSI
  - S-ATA - single device, allows parallel requests
  - iSCSI - same SCSI commands, use ethernet for physical link

Two Buses

- **Buses**: connects system components
  - Insufficient bandwidth can bottleneck system
  - Performance factors
    - Physical length
    - Number and type of connected devices (taps)

- **Processor-memory bus**
  - Connects CPU and memory, no direct I/O interface
  + Short, few taps → fast, high-bandwidth
    - System specific

- **I/O bus**
  - Connects I/O devices, no direct processor interface
    - Longer, more taps → slower, lower-bandwidth
  + Industry standard

- Bridge connects these busses
Standard Bus Examples

<table>
<thead>
<tr>
<th>Type</th>
<th>PCI</th>
<th>SCSI</th>
<th>USB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplexed?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Clocking</td>
<td>33 (66) MHz</td>
<td>5 (10) MHz</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>Data rate</td>
<td>133 (266) MB/s</td>
<td>10 (20) MB/s</td>
<td>0.2, 1.5, 80 MB/s</td>
</tr>
<tr>
<td>Arbitration</td>
<td>Parallel</td>
<td>Self-selection</td>
<td>Daisy-chain</td>
</tr>
<tr>
<td>Maximum masters</td>
<td>1024</td>
<td>7–31</td>
<td>127</td>
</tr>
<tr>
<td>Maximum length</td>
<td>0.5 m</td>
<td>2.5 m</td>
<td>–</td>
</tr>
</tbody>
</table>

**USB (universal serial bus)**
- Popular for low-/moderate-bandwidth external peripherals
  - Packetized interface (like TCP) extremely flexible
  - Also supplies power to the peripheral

OS Plays a Big Role

- **I/O interface is typically under OS control**
  - User applications access I/O devices indirectly (e.g., SYSCALL)
  - Why?

- **Virtualization**: same argument as for memory
  - Physical devices shared among multiple apps
  - Direct access could lead to conflicts

- **Synchronization**
  - Most have asynchronous interfaces, require unbounded waiting
  - OS handles asynchrony internally, presents synchronous interface

- **Standardization**
  - Devices of a certain type (disks) can/will have different interfaces
  - OS handles differences (via drivers), presents uniform interface
Sending Commands to I/O Devices

- Usually only OS can do this. Why?
- **I/O instructions**
  - OS only? Instructions are privileged
  - E.g., IA32
- **Memory-mapped I/O**
  - Portion of physical address space reserved for I/O
  - BIOS/Boot code uses configuration registers to map I/O physical addresses to I/O device control registers
  - OS maps virtual addresses to I/O physical addresses
  - Stores/loads to these addresses are commands to I/O devices
    - Main memory ignores them, I/O devices recognize and respond
    - Address may specify both I/O device and command
    - Generally, these address are not cached. Why?
  - OS only? I/O physical addresses only mapped in OS address space
  - E.g., almost every architecture other than IA32

Direct Memory Access (DMA)

- Interrupts remove overhead of polling...
- But still requires OS to transfer data one word at a time
  - OK for low bandwidth I/O devices: mice, microphones, etc.
  - Bad for high bandwidth I/O devices: disks, monitors, etc.

- **Direct Memory Access (DMA)**
  - Block I/O memory transfers without processor control
  - Transfers entire blocks (e.g., pages, video frames) at a time
    - Can use bus "burst" transfer mode if available
    - Only interrupts processor when done (or if error occurs)
DMA Controllers

- To do DMA, I/O device attached to **DMA controller**
  - Multiple devices can be connected to one controller
  - Controller itself seen as a memory mapped I/O device
    - Processor initializes start memory address, transfer size, etc.
    - DMA controller takes care of bus arbitration and transfer details
  - That’s why buses support arbitration and multiple masters

I/O Processors

- A DMA controller is a very simple component
  - May be as simple as a FSM with some local memory
- Some I/O requires complicated sequences of transfers
  - **I/O processor**: heavier DMA controller that executes instruction
    - Can be programmed to do complex transfers
Example: 850 Chipset [2003]

- Memory Controller Hub
  - Memory Controller Hub
  - 400MHz RDRAM
  - AGP Graphics
- I/O Controller Hub
  - LAN
  - Audio
  - USB
  - PCI
  - ATA
    - For storage devices

Example: 946G Chipset [2005]

- Vs. 850 Chip Set
- RDRAM → DDR2 (~3x bandwidth)
- AGP4X → PCI Express (~8x bandwidth)
- 4 USB 1.0 → 8 USB 2.0
- Etc.
Example: VIA K8T800 Pro Chipset [2005]

- For AMD Opteron (or Athlon64)
- DDR memory directly (not via North Bridge)
- North Bridge via HyperTransport “Bus”
- Other Athlons via HyperTransport
  - (not shown)
  - Glueless multiprocessor!

Example: IBM 3090 I/O

- Mainframe computer
  - Processors
  - IOPs (channels)
### Reliability: RAID

- **Error correction:** more important for disk than for memory
  - Error correction/detection per block (handled by disk hardware)
  - Mechanical disk failures (entire disk lost) most common failure mode
    - Many disks means high failure rates
    - Entire file system can be lost if files striped across multiple disks

- **RAID (redundant array of inexpensive disks)**
  - Add redundancy
  - Similar to DRAM error correction, but...
  - Major difference: which disk failed is known
    - Even parity can be used to recover from single failures
    - Parity disk can be used to reconstruct data faulty disk
  - RAID design balances bandwidth and fault-tolerance
  - Implemented in hardware (fast, expensive) or software

### Levels of RAID - Summary

- **RAID-0 - no redundancy**
  - Multiplies read and write bandwidth

- **RAID-1 - mirroring**
  - Pair disks together (write both, read one)
  - 2x storage overhead
  - Multiples only read bandwidth (not write bandwidth)

- **RAID-3 - bit-level parity** (dedicated parity disk)
  - N+1 disks, calculate parity (write all, read all)
  - Good sequential read/write bandwidth, poor random accesses
  - If N=8, only 13% overhead

- **RAID-4/5 - block-level parity**
  - Reads only data you need
  - Writes require read, calculate parity, write data & parity
RAID-3: Bit-level parity

- **RAID-3 - bit-level parity**
  - dedicated parity disk
  - N+1 disks, calculate parity (write all, read all)
  - Good sequential read/write bandwidth, poor random accesses
  - If N=8, only 13% overhead

RAID 4/5 - Block-level Parity

- **RAID-4/5**
  - Reads only data you need
  - Writes require read, calculate parity, write data&parity

  - Naïve approach
    1. Read all disks
    2. Calculate parity
    3. Write data&parity

  - Better approach
    - Read data&parity
    - Calculate parity
    - Write data&parity

  - Still worse for **writes** than RAID-3
RAID-4 vs RAID-5

- RAID-5 rotates the parity disk, avoid single-disk bottleneck