I/O Devices

- There exists a large variety of I/O devices:
  - Many of them with different properties
  - They seem to require different interfaces to manipulate and manage them
    - We don’t want a new interface for every device
    - Diverse, but similar interfaces leads to code duplication
  - Challenge:
    - Uniform and efficient approach to I/O

Categories of I/O Devices (by usage)

- Human interface
  - Used to communicate with the user
  - Printers, Video Display, Keyboard, Mouse
- Machine interface
  - Used to communicate with electronic equipment
  - Disk and tape drives, Sensors, Controllers, Actuators
- Communication
  - Used to communicate with remote devices
  - Ethernet, Modems, Wireless

I/O Device Handling

- Data rate
  - May be differences of several orders of magnitude between the data transfer rates
  - Example: Assume 1000 cycles/byte I/O
    - Keyboard needs 10 Khz processor to keep up
    - Gigabit Ethernet needs 100 Ghz processor….

Sample Data Rates
I/O Device Handling Considerations

- Complexity of control
- Unit of transfer
  - Data may be transferred as a stream of bytes for a terminal or in larger blocks for a disk
- Data representation
  - Encoding schemes
- Error conditions
  - Devices respond to errors differently
  - Expected error rate also differs

"Failure Trends in a Large Disk Drive Population"; Eduardo Pinheiro, Wolf-Dietrich Weber and Luiz André Barroso – USENIX FAST ’07

Layering
- Need to be both general and specific, e.g.
- Devices that are the same, but aren’t the same
  - Hard-disk, USB disk, RAM disk
- Interaction of layers
  - Swap partition and data on same disk
  - Two mice
- Priority
  - Keyboard, disk, network

Accessing I/O Controllers

- Layering
  - Need to be both general and specific, e.g.
  - Devices that are the same, but aren’t the same
    - Hard-disk, USB disk, RAM disk
  - Interaction of layers
    - Swap partition and data on same disk
    - Two mice
  - Priority
    - Keyboard, disk, network

(a) Separate I/O and memory space
  - I/O controller registers appear as I/O ports
  - Accessed with special I/O instructions
(b) Memory-mapped I/O
  - Controller registers appear as memory
  - Use normal load/store instructions to access
(c) Hybrid
  - x86 has both ports and memory mapped I/O
    - Linux Device Drivers; Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman

Bus Architectures

Interrupts Revisited

- Devices connected to an interrupt Controller via lines on an I/O bus (e.g. PCI)
- Interrupt Controller signals interrupt to CPU and is eventually acknowledged.
- Exact details are architecture specific (APIC is most common).
### Programmed I/O
- Also called *polling*, or *busy waiting*
- I/O module (controller) performs the action, not the processor
- Sets appropriate bits in the I/O status register
- No interrupts occur
- Processor checks status until operation is complete
  - Wastes CPU cycles

### Interrupt-Driven I/O
- Processor is interrupted when I/O module (controller) ready to exchange data
- Processor is free to do other work
- No needless waiting
- Consumes a lot of processor time because every word read or written passes through the processor

### Direct Memory Access
- Transfers a block of data directly to or from memory
- An interrupt is sent when the task is complete
- The processor is only involved at the beginning and end of the transfer

### DMA Considerations
- Reduces number of interrupts
  - Less (expensive) context switches
- Requires contiguous regions
  - Copying
  - Scatter-gather
- Synchronous/Asynchronous
- Shared bus must be arbitrated

### DMA
- Commonly *burst-mode* is used
  - CPU uses several consecutive cycles to load entire cache line
  - DMA writes (or reads) a similar sized burst
  - Reason: More efficient (less cycles overall) to transfer a sequence of words than a word at a time.
  - No bus arbitration, read/write setup, or addressing cycles required after first transfer.
- Number of required busy cycles can be cut by
  - Path between DMA module and I/O module that does not include the system bus
The Process to Perform DMA Transfer

Evolution of the I/O Function
• Processor directly controls a peripheral device
  – Example: CPU controls a flip-flop to implement a serial line

Evolution of the I/O Function
• Controller or I/O module is added
  – Processor uses programmed I/O without interrupts
  – Processor does not need to handle details of external devices
  – Example: A Universal Asynchronous Receiver Transmitter
    • CPU simply reads and writes bytes to I/O controller
    • I/O controller responsible for managing the signaling

Evolution of the I/O Function
• Controller or I/O module with interrupts
  – Processor does not spend time waiting for an I/O operation to be performed

Evolution of the I/O Function
• Direct Memory Access
  – Blocks of data are moved into memory without involving the processor
  – Processor involved at beginning and end only

Evolution of the I/O Function
• I/O module has a separate processor
  – Example: SCSI controller
    • Controller CPU executes SCSI program code out of main memory
Evolution of the I/O Function

- **I/O processor**
  - I/O module has its own local memory, internal bus, etc.
  - It's a computer in its own right
  - Example: Myrinet 10 gigabit NIC

![Diagram of CPU, Memory, Myrinet Controller, and Bus connections including Interrupt Line and Myrinet NIC.]