I/O Management
Intro
Chapter 5
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

<table>
<thead>
<tr>
<th>Device</th>
<th>Data rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>10 bytes/sec</td>
</tr>
<tr>
<td>Mouse</td>
<td>100 bytes/sec</td>
</tr>
<tr>
<td>56K modem</td>
<td>7 KB/sec</td>
</tr>
<tr>
<td>Telephone channel</td>
<td>8 KB/sec</td>
</tr>
<tr>
<td>Dual ISDN lines</td>
<td>16 KB/sec</td>
</tr>
<tr>
<td>Laser printer</td>
<td>100 KB/sec</td>
</tr>
<tr>
<td>Scanner</td>
<td>400 KB/sec</td>
</tr>
<tr>
<td>Classic Ethernet</td>
<td>1.25 MB/sec</td>
</tr>
<tr>
<td>USB (Universal Serial Bus)</td>
<td>1.5 MB/sec</td>
</tr>
<tr>
<td>Digital camcorder</td>
<td>4 MB/sec</td>
</tr>
<tr>
<td>IDE disk</td>
<td>5 MB/sec</td>
</tr>
<tr>
<td>40x CD-ROM</td>
<td>6 MB/sec</td>
</tr>
<tr>
<td>Fast Ethernet</td>
<td>12.5 MB/sec</td>
</tr>
<tr>
<td>ISA bus</td>
<td>16.7 MB/sec</td>
</tr>
<tr>
<td>EIDE (ATA-2) disk</td>
<td>16.7 MB/sec</td>
</tr>
<tr>
<td>FireWire (IEEE 1394)</td>
<td>50 MB/sec</td>
</tr>
<tr>
<td>XGA Monitor</td>
<td>60 MB/sec</td>
</tr>
<tr>
<td>SONET OC-12 network</td>
<td>78 MB/sec</td>
</tr>
<tr>
<td>SCSI Ultra 2 disk</td>
<td>80 MB/sec</td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>125 MB/sec</td>
</tr>
<tr>
<td>Ultrium tape</td>
<td>320 MB/sec</td>
</tr>
<tr>
<td>PCI bus</td>
<td>528 MB/sec</td>
</tr>
<tr>
<td>Sun Gigaplane XB backplane</td>
<td>20 GB/sec</td>
</tr>
</tbody>
</table>
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
    - *lp0: printer on fire!*
  - Expected error rate also differs
I/O Device Handling Considerations

• 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

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

(a) A single-bus architecture
(b) A dual-bus memory architecture

CPU reads and writes of memory go over this high-bandwidth bus

This memory port is to allow I/O devices access to memory
Intel IXP420
Interrupts

- 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.
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.

(c) Direct memory access
DMA Considerations

✓ Reduces number of interrupts
  – Less (expensive) context switches or kernel entry-exits

✗ Requires contiguous regions
  – Copying
  – Scatter-gather

• Synchronous/Asynchronous
• Shared bus must be arbitrated
  – CPU cache reduces (but not eliminates) CPU need for bus
The Process to Perform DMA Transfer

1. Device driver is told to transfer disk data to buffer at address X
2. Device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. Disk controller initiates DMA transfer
4. Disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. When C = 0, DMA interrupts CPU to signal transfer completion
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