I/O Management

Intro

Chapter 5

Learning Outcomes

• A high-level understanding of the properties of a variety of I/O devices.
• An understanding of methods of interacting with I/O devices.
• An appreciation of the trend towards offloading more I/O handling to devices themselves.

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 KHz</td>
</tr>
<tr>
<td>Mouse</td>
<td>1000 cycles/byte</td>
</tr>
<tr>
<td>Hard disk</td>
<td>10 GB/s</td>
</tr>
<tr>
<td>Hard drive</td>
<td>7 MB/s</td>
</tr>
<tr>
<td>USB 2.0</td>
<td>480 Mbps</td>
</tr>
<tr>
<td>USB 3.0</td>
<td>5 GB/s</td>
</tr>
<tr>
<td>Firewire</td>
<td>5 GB/s</td>
</tr>
<tr>
<td>Firewire Thunderbolt</td>
<td>10 GB/s</td>
</tr>
<tr>
<td>Firewire Thunderbolt 2.5 GB/s</td>
<td>20 GB/s</td>
</tr>
<tr>
<td>Firewire Thunderbolt 4 GB/s</td>
<td>30 GB/s</td>
</tr>
<tr>
<td>Firewire Thunderbolt 8 GB/s</td>
<td>40 GB/s</td>
</tr>
<tr>
<td>Firewire Thunderbolt 15 GB/s</td>
<td>50 GB/s</td>
</tr>
<tr>
<td>Firewire Thunderbolt 30 GB/s</td>
<td>60 GB/s</td>
</tr>
<tr>
<td>Firewire Thunderbolt 60 GB/s</td>
<td>70 GB/s</td>
</tr>
<tr>
<td>Firewire Thunderbolt 120 GB/s</td>
<td>80 GB/s</td>
</tr>
<tr>
<td>Firewire Thunderbolt 240 GB/s</td>
<td>90 GB/s</td>
</tr>
<tr>
<td>Firewire Thunderbolt 480 GB/s</td>
<td>100 GB/s</td>
</tr>
<tr>
<td>Firewire Thunderbolt 960 GB/s</td>
<td>110 GB/s</td>
</tr>
<tr>
<td>Firewire Thunderbolt 1920 GB/s</td>
<td>120 GB/s</td>
</tr>
</tbody>
</table>

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

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

Bus Architectures

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

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.
I/O Interaction

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 data directly between Memory and Device
- CPU not needed for copying

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 or kernel entry-exits
✗ Requires contiguous regions (buffers)
  - Copying
  - Some hardware supports "Scatter-gather"
- Synchronous/Asynchronous
- Shared bus must be arbitrated (hardware)
  - CPU cache reduces (but not eliminates) CPU need for bus
**The Process to Perform DMA Transfer**

1. DMA controller transfers data to buffer in DMA controller
2. CPU sends start DMA transfer command
3. DMA controller transfers data to buffer and increments address
4. DMA controller transfers data to memory
5. DMA controller updates transfer completion

**Device Evolution - Complexity and Performance**

**Evolution of the I/O Function**

- Processor directly controls a peripheral device
  - Example: CPU controls a flip-flop to implement a serial line

```
<table>
<thead>
<tr>
<th>CPU</th>
<th>Memory</th>
<th>Flip Flop</th>
<th>Serial Line</th>
</tr>
</thead>
</table>
```

- '1' = 5V
- '0' = 0V

**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

```
<table>
<thead>
<tr>
<th>CPU</th>
<th>Memory</th>
<th>UART</th>
</tr>
</thead>
</table>
```

**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

```
<table>
<thead>
<tr>
<th>CPU</th>
<th>Memory</th>
<th>UART</th>
</tr>
</thead>
</table>
```

**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

```
<table>
<thead>
<tr>
<th>CPU</th>
<th>Memory</th>
<th>UART</th>
</tr>
</thead>
</table>
```
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.
  – Its a computer in its own right
  – Example: Myrinet 10 gigabit NIC

General Trend

• More specialised hardware
• Offloading more functionality into hardware
  – Reduced load on CPU
• Improved performance