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 (cycles/byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modem</td>
<td>100</td>
</tr>
<tr>
<td>56K modem</td>
<td>7.24 KHz</td>
</tr>
<tr>
<td>Printers, Video Display</td>
<td>10</td>
</tr>
<tr>
<td>Keyboard</td>
<td>1000</td>
</tr>
<tr>
<td>Disk drives (5400 rpm)</td>
<td>15</td>
</tr>
<tr>
<td>Tape drives (1500 rpm)</td>
<td>15</td>
</tr>
<tr>
<td>Sensor</td>
<td>1000</td>
</tr>
<tr>
<td>Controller</td>
<td>1000</td>
</tr>
<tr>
<td>Actuator</td>
<td>1000</td>
</tr>
<tr>
<td>Ethernet</td>
<td>1000</td>
</tr>
<tr>
<td>Modem</td>
<td>10</td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>1000</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
    - Printer on fire!
  - Expected error rate also differs

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

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

Bus Architectures

- Separate I/O and memory space
- Memory-mapped I/O
- Hybrid

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

# 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
- DMA controller transfers data to buffer in DMA transfer (I/O module)
- DMA module transfers data to buffer

# 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.
  - Its a computer in its own right
  - Example: Myrinet 10 gigabit NIC