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.
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
• Logical position of device drivers is shown here
• Drivers (originally) compiled into the kernel
  – Including OS/161
  – Device installers were technicians
  – Number and types of devices rarely changed
• Nowadays they are dynamically loaded when needed
  – Linux modules
  – Typical users (device installers) can’t build kernels
  – Number and types vary greatly
    • Even while OS is running (e.g. hot-plug USB devices)
Device Drivers

• Drivers classified into similar categories
  – Block devices and character (stream of data) device

• OS defines a standard (internal) interface to the different classes of devices
  – Example: USB HID (human interface device) class specifications
    • human input devices follow a set of rules making it easier to design a standard interface.
## USB Device Classes

<table>
<thead>
<tr>
<th>Base Class</th>
<th>Descriptor Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00h</td>
<td>Device</td>
<td>Use class information in the Interface Descriptors</td>
</tr>
<tr>
<td>01h</td>
<td>Interface Audio</td>
<td></td>
</tr>
<tr>
<td>02h</td>
<td>Both</td>
<td>Communications and CDC Control</td>
</tr>
<tr>
<td>03h</td>
<td>Interface HID</td>
<td>HID (Human Interface Device)</td>
</tr>
<tr>
<td>05h</td>
<td>Interface Physical</td>
<td></td>
</tr>
<tr>
<td>06h</td>
<td>Interface Image</td>
<td></td>
</tr>
<tr>
<td>07h</td>
<td>Interface Printer</td>
<td></td>
</tr>
<tr>
<td>08h</td>
<td>Interface Mass Storage</td>
<td></td>
</tr>
<tr>
<td>09h</td>
<td>Device Hub</td>
<td></td>
</tr>
<tr>
<td>0Ah</td>
<td>Interface CDC-Data</td>
<td></td>
</tr>
<tr>
<td>0Bh</td>
<td>Interface Smart Card</td>
<td></td>
</tr>
<tr>
<td>0 Dh</td>
<td>Interface Content Security</td>
<td></td>
</tr>
<tr>
<td>0Eh</td>
<td>Interface Video</td>
<td></td>
</tr>
<tr>
<td>0Fh</td>
<td>Interface Personal Healthcare</td>
<td></td>
</tr>
<tr>
<td>10h</td>
<td>Interface Audio/Video Devices</td>
<td></td>
</tr>
<tr>
<td>DCh</td>
<td>Both</td>
<td>Diagnostic Device</td>
</tr>
<tr>
<td>E0h</td>
<td>Interface Wireless Controller</td>
<td></td>
</tr>
<tr>
<td>EFh</td>
<td>Both</td>
<td>Miscellaneous</td>
</tr>
<tr>
<td>FEh</td>
<td>Interface Application Specific</td>
<td></td>
</tr>
<tr>
<td>FFh</td>
<td>Both</td>
<td>Vendor Specific</td>
</tr>
</tbody>
</table>
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>

USB 3.0 625 MB/s (5 Gb/s)
Thunderbolt 2.5GB/sec (20 Gb/s)
PCIe v3.0 x16 16GB/s
Device Drivers

- **Device drivers job**
  - translate request through the device-independent standard interface (open, close, read, write) into appropriate sequence of commands (register manipulations) for the particular hardware
  - Initialise the hardware at boot time, and shut it down cleanly at shutdown
Device Driver

• **After issuing the command to the device, the device either**
  – Completes immediately and the driver simply returns to the caller
  – Or, device must process the request and the driver usually blocks waiting for an interrupt indicating I/O completion.

• **Drivers are thread-safe** as they can be called by another process while a process is already blocked in the driver.
  – Thread-safe: Synchronised....
Device-Independent I/O Software

- There is commonality between drivers of similar classes
- Divide I/O software into device-dependent and device-independent I/O software
- Device independent software includes
  - Buffer or Buffer-cache management
  - TCP/IP stack
  - Managing access to dedicated devices
  - Error reporting
Driver $\Leftrightarrow$ Kernel Interface

• Major Issue is uniform interfaces to devices and kernel
  
  – Uniform device interface for kernel code
    • Allows different devices to be used the same way
      – No need to rewrite file-system to switch between SCSI, IDE or RAM disk
    • Allows internal changes to device driver with fear of breaking kernel code
  
  – Uniform kernel interface for device code
    • Drivers use a defined interface to kernel services (e.g. kmalloc, install IRQ handler, etc.)
    • Allows kernel to evolve without breaking existing drivers
  
  – Together both uniform interfaces avoid a lot of programming implementing new interfaces
    • Retains compatibility as drivers and kernels change over time.
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

1. DMA Controller in Device

2. Separate DMA Controller
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. 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