Today’s Topics

▪ Device characteristics
  • Block device vs. Character device
  • Direct I/O vs. Memory-mapped I/O
  • Polling vs. Interrupts
  • Programmed I/O vs. DMA
  • Blocking vs. Non-blocking I/O

▪ I/O software layers
A Typical PC Bus Structure

- **monitor**
- **processor**
- **cache**
- **memory**
- **SCSI controller**
- **SCSI bus**
- **IDE disk controller**
- **expansion bus interface**
- **keyboard**
- **parallel port**
- **serial port**

The diagram shows the typical bus structure of a PC, including the main components such as the monitor, processor, cache, memory, and peripherals like the IDE disk controller, keyboard, parallel port, and serial port. The PCI bus is central, connecting various components.
Modern PC Architecture

- Intel® Core™ processors
  - Processor Graphics
  - DDR3/3L up to 1600 MHz
  - DDR3 up to 1600 MHz

- Intel® High Definition Audio

- Intel® Express Chipset
  - 1x16 lanes or 2x8 - PCI Express® 3.0 Graphics
  - 1x8 & 1x4 lanes - PCI Express® 3.0 + 1x4 lanes for Thunderbolt

- 3 Independent Display Support

- 4xUSB 3.0 Ports
  - 10xUSB 2.0 Ports
  - Dual EHCI; USB Port Disable

- Intel® Integrated 10/100/1000 MAC
  - PCIe® x1
  - SM Bus

- Intel® Gigabit LAN Connect

- Intel® ME 8.x Firmware and BIOS Support

- Intel® AMT 8.0

- Intel® Small Business Advantage

- Optional

SWE3004: Operating Systems | Spring 2013 | Jin-Soo Kim (jinsookim@skku.edu)
I/O Devices (1)

- **Block device**
  - Stores information in fixed-size blocks, each one with its own address.
  - 512B – 32KB per block
  - It is possible to read or write each block independently of all the other ones.
  - Disks, tapes, etc.

- **Character device**
  - Delivers or accepts a stream of characters.
  - Not addressable and no seek operation.
  - Printers, networks, mice, keyboards, etc.
## I/O Devices (2)

<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 2.0**: 480Mbit/s  
**USB 3.0**: 5Gbit/s  
**SATA2**: 3Gbit/s  
**SATA3**: 6Gbit/s  
**Thunderbolt**: 10Gbit/s/ch x 2 ch
I/O Devices (3)

- **Device controller (or host adapter)**
  - I/O devices have components:
    - Mechanical component
    - Electronic component
  - The electronic component is the device controller.
    - May be able to handle multiple devices.
  - Controller’s tasks
    - Convert serial bit stream to block of bytes.
    - Perform error correction as necessary.
    - Make available to main memory.
Accessing I/O Devices (1)

- Direct I/O
  - Use special I/O instructions to an I/O port address.

<table>
<thead>
<tr>
<th>I/O address range (hexadecimal)</th>
<th>device</th>
</tr>
</thead>
<tbody>
<tr>
<td>000–00F</td>
<td>DMA controller</td>
</tr>
<tr>
<td>020–021</td>
<td>interrupt controller</td>
</tr>
<tr>
<td>040–043</td>
<td>timer</td>
</tr>
<tr>
<td>200–20F</td>
<td>game controller</td>
</tr>
<tr>
<td>2F8–2FF</td>
<td>serial port (secondary)</td>
</tr>
<tr>
<td>320–32F</td>
<td>hard-disk controller</td>
</tr>
<tr>
<td>378–37F</td>
<td>parallel port</td>
</tr>
<tr>
<td>3D0–3DF</td>
<td>graphics controller</td>
</tr>
<tr>
<td>3F0–3F7</td>
<td>diskette-drive controller</td>
</tr>
<tr>
<td>3F8–3FF</td>
<td>serial port (primary)</td>
</tr>
</tbody>
</table>
Accessing I/O Devices (2)

- Memory-mapped I/O
  - The device control registers are mapped into the address space of the processor.
    - The CPU executes I/O requests using the standard data transfer instructions.
  - I/O device drivers can be written entirely in C.
  - No special protection mechanism is needed to keep user processes from performing I/O
    - Can give a user control over specific devices but not others by simply including the desired pages in its page table.
  - Reading a device register and testing its value is done with a single instruction.
Polling vs. Interrupts (1)

- **Polled I/O**
  - CPU asks ("polls") devices if need attention.
    - ready to receive a command
    - command status, etc.
  - **Advantages**
    - Simple
    - Software is in control.
    - Efficient if CPU finds a device to be ready soon.
  - **Disadvantages**
    - Inefficient in non-trivial system (high CPU utilization).
    - Low priority devices may never be serviced.
Polling vs. Interrupts (2)

- **Interrupt-driven I/O**
  - I/O devices request interrupt when need attention.
  - Interrupt service routines specific to each device are invoked.
  - Interrupts can be shared between multiple devices.
  - **Advantages**
    - CPU only attends to device when necessary.
    - More efficient than polling in general.
  - **Disadvantages**
    - Excess interrupts slow (or prevent) program execution.
    - Overheads (may need 1 interrupt per byte transferred)
Polling vs. Interrupts (3)

1. CPU
   - device driver initiates I/O
   - CPU executing checks for interrupts between instructions
   - CPU receiving interrupt, transfers control to interrupt handler
   - interrupt handler processes data, returns from interrupt
   - CPU resumes processing of interrupted task

2. I/O controller
   - initiates I/O
   - input ready, output complete, or error generates interrupt signal

3. I/O controller
   - input ready, output complete, or error generates interrupt signal

4. I/O controller
   - input ready, output complete, or error generates interrupt signal

5. I/O controller
   - input ready, output complete, or error generates interrupt signal

6. I/O controller
   - input ready, output complete, or error generates interrupt signal

7. I/O controller
   - input ready, output complete, or error generates interrupt signal
Polling vs. Interrupts (4)
Programmed I/O vs. DMA

- **DMA (Direct Memory Access)**
  - Bypasses CPU to transfer data directly between I/O device and memory.
  - Used to avoid programmed I/O for large data movement.
Blocking vs. Non-Blocking I/O

- **Blocking I/O**
  - Process is suspended until I/O completed.
  - Easy to use and understand.

- **Nonblocking I/O**
  - I/O call returns quickly, with a return value that indicates how many bytes were transferred.
  - A nonblocking read() returns immediately with whatever data available – the full number of bytes requested, fewer, or none at all.
Goals of I/O Software

- **Goals**
  - Device independence
  - Uniform naming
  - Error handling
  - Synchronous vs. asynchronous
  - Buffering
  - Sharable vs. dedicated devices
I/O Software Layers

- User-level I/O Software
  - Device-independent I/O Software
    - Device Drivers
    - Interrupt Handlers
  - Hardware
    - Network
Interrupt Handlers

Handling interrupts

- Critical actions:
  - Acknowledge an interrupt to the PIC.
  - Reprogram the PIC or the device controller.
  - Update data structures accessed by both the device and the processor.

- Noncritical actions:
  - Update data structures that are accessed only by the processor.
    (e.g., reading the scan code from the keyboard)

- Noncritical deferred actions:
  - Actions may be delayed.
  - Copy buffer contents into the address space of some process (e.g., sending the keyboard line buffer to the terminal handler process).
  - Bottom half (Linux)

Reenable interrupts
Device Drivers (1)

- **Device drivers**
  - Device-specific code to control each I/O device interacting with device-independent I/O software and interrupt handlers.
  - Requires to define a well-defined model and a standard interface of how they interact with the rest of the OS.
  - Implementing device drivers:
    - Statically linked with the kernel.
    - Selectively loaded into the system during boot time.
    - Dynamically loaded into the system during execution. (especially for hot pluggable devices).
Device Drivers (2)
Device Drivers (3)

- The problem
  - Reliability remains a crucial, but unresolved problem
    - 5% of Windows systems crash every day
    - Huge cost of failures: stock exchange, e-commerce, ...
    - Growing “unmanaged systems”: digital appliances, consumer electronics devices
  - OS extensions are increasingly prevalent
    - 70% of Linux kernel code
    - Over 35,000 drivers with over 120,000 versions on Windows XP
    - Written by less experienced programmer
  - Extensions are a leading cause of OS failure
    - Drivers cause 85% of Windows XP crashes
    - Drivers are 7 times buggier than the kernel in Linux
Device-Independent I/O SW (1)

- **Uniform interfacing for device drivers**
  - In Unix, devices are modeled as special files.
    - They are accessed through the use of system calls such as open(), read(), write(), close(), ioctl(), etc.
    - A file name is associated with each device.
  - Major device number locates the appropriate driver.
    - Minor device number (stored in i-node) is passed as a parameter to the driver in order to specify the unit to be read or written.
  - The usual protection rules for files also apply to I/O devices.
### Device-Independent I/O SW (2)

**Buffering**

- (a) Unbuffered
- (b) Buffered in user space
- (c) Buffered in the kernel space
- (d) Double buffering in the kernel
### Error reporting

- Many errors are device-specific and must be handled by the appropriate driver, but the framework for error handling is device independent.
- Programming errors vs. actual I/O errors
- Handling errors
  - Returning the system call with an error code.
  - Retrying a certain number of times.
  - Ignoring the error.
  - Killing the calling process.
  - Terminating the system.
Device-Independent I/O SW (4)

- Allocating and releasing dedicated devices
  - Some devices cannot be shared.
    (1) Require processes to perform open()’s on the special files for devices directly.
      - The process retries if open() fails.
    (2) Have special mechanisms for requesting and releasing dedicated devices.
      - An attempt to acquire a device that is not available blocks the caller.

- Device-independent block size
  - Treat several sectors as a single logical block.
  - The higher layers only deal with abstract devices that all use the same block size.
User-Space I/O Software

- Provided as a library
  - Standard I/O library in C
    - fopen() vs. open()?  
    - Buffering for fgetc()? 

- Spooling
  - A way of dealing with dedicated I/O devices in a multiprogramming system.
  - Implemented by a daemon and a spooling directory.
  - Printers, network file transfers, USENET news, mails, etc.
I/O Systems Layers

I/O request → User processes

Device-independent software

Device drivers

Interrupt handlers

Hardware

Layer

I/O reply → I/O functions

Make I/O call; format I/O; spooling

Naming, protection, blocking, buffering, allocation

Set up device registers; check status

Wake up driver when I/O completed

Perform I/O operation