Announcements

- Reading
  - Chapter 13
- Project 5 due Wednesday, 6pm
  - Late deadline extended to Friday 6pm
- Project 6 posted Wednesday
  - Due a week from Thursday, 6pm
  - On I/O (stdin, stdout, message passing)
A Typical PC Bus Structure

I/O Hardware

- I/O instructions control devices
- Devices have addresses, used by
  - Direct I/O instructions
    - `inb`, `outb` on Intel x86
  - Memory-mapped I/O
- Device registers to communicate with device
  - Status register, Command register, Data-in register, Data-out register, etc.
Device I/O Port Locations on PCs (partial)

<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>

Programmed I/O

- I/O between memory and device is controlled by the CPU
- Two forms
  - Polling/Handshaking I/O
  - Interrupt-driven I/O
Polling

• CPU checks device status repeatedly (the status bit)
  - If data is available, the CPU will read it (Data-in register).
  - If CPU has data to write, waits until the device is ready, then writes a byte (Sets Command register, writes to Data-out register)

Interrupts

• Device readiness signaled by an interrupt
  - Maskable to ignore or delay some interrupts
  - Interrupt vector to dispatch interrupt to correct handler
    • Based on priority
    • Some unmaskable
## Interrupt-Driven I/O Cycle

![Diagram of Interrupt-Driven I/O Cycle]

## Intel Pentium Processor Event-Vector Table

<table>
<thead>
<tr>
<th>Vector Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>divide error</td>
</tr>
<tr>
<td>1</td>
<td>debug exception</td>
</tr>
<tr>
<td>2</td>
<td>null interrupt</td>
</tr>
<tr>
<td>3</td>
<td>breakpoint</td>
</tr>
<tr>
<td>4</td>
<td>INTO-detected overflow</td>
</tr>
<tr>
<td>5</td>
<td>bound range exception</td>
</tr>
<tr>
<td>6</td>
<td>invalid opcode</td>
</tr>
<tr>
<td>7</td>
<td>device not available</td>
</tr>
<tr>
<td>8</td>
<td>double fault</td>
</tr>
<tr>
<td>9</td>
<td>coprocessor segment overrun (reserved)</td>
</tr>
<tr>
<td>10</td>
<td>invalid task state segment</td>
</tr>
<tr>
<td>11</td>
<td>segment not present</td>
</tr>
<tr>
<td>12</td>
<td>stack fault</td>
</tr>
<tr>
<td>13</td>
<td>general protection</td>
</tr>
<tr>
<td>14</td>
<td>page fault</td>
</tr>
<tr>
<td>15</td>
<td>(Intel reserved, do not use)</td>
</tr>
<tr>
<td>16</td>
<td>floating-point error</td>
</tr>
<tr>
<td>17</td>
<td>alignment check</td>
</tr>
<tr>
<td>18</td>
<td>machine check</td>
</tr>
<tr>
<td>19831</td>
<td>(Intel reserved, do not use)</td>
</tr>
<tr>
<td>329350</td>
<td>maskable interrupts</td>
</tr>
</tbody>
</table>
Direct Memory Access (DMA)

- Used to avoid programmed I/O for large data movement
- Requires DMA controller
  - Shepherds the data transfer rather than the CPU
  - Uses the memory bus, preventing the CPU from using it

Six Step Process to Perform DMA Transfer
Coping with Many Devices

- Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only
- May have multiple devices of the same type (e.g. two serial ports, two disks)

**Characteristics of I/O Devices**

<table>
<thead>
<tr>
<th>aspect</th>
<th>variation</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>data-transfer mode</td>
<td>character</td>
<td>terminal</td>
</tr>
<tr>
<td></td>
<td>block</td>
<td>disk</td>
</tr>
<tr>
<td>access method</td>
<td>sequential</td>
<td>modem</td>
</tr>
<tr>
<td></td>
<td>random</td>
<td>CD-ROM</td>
</tr>
<tr>
<td>transfer schedule</td>
<td>synchronous</td>
<td>tape</td>
</tr>
<tr>
<td></td>
<td>asynchronous</td>
<td>keyboard</td>
</tr>
<tr>
<td>sharing</td>
<td>dedicated</td>
<td>tape</td>
</tr>
<tr>
<td></td>
<td>sharable</td>
<td>keyboard</td>
</tr>
<tr>
<td>device speed</td>
<td>latency</td>
<td>CD-ROM</td>
</tr>
<tr>
<td></td>
<td>seek time</td>
<td>graphics controller</td>
</tr>
<tr>
<td></td>
<td>transfer rate</td>
<td>disk</td>
</tr>
<tr>
<td></td>
<td>delay between operations</td>
<td></td>
</tr>
</tbody>
</table>
Abstracting the I/O Interface

- Goal: hide complexity (differences) of different devices from different parts of the OS and applications
  - System call layer encapsulates device behaviors in generic classes
  - Device-driver layer hides differences among I/O controllers (of the same class) from kernel

A Kernel I/O Structure

- syscall layer
  - System call interface
  - Rest of kernel (scheduler, etc.)
- driver layer
  - Keyboard device driver
  - Floppy device driver
  - Serial device driver
  - ATA/PI device driver
  - Keyboard device controller
  - Floppy device controller
  - Serial port controller
  - ATA/PI device controller
Device Driver

- Device-specific code that implements I/O subsystem’s device-generic API
  - For example, there are many kinds of disks supporting the same operations. A driver for disk X implements those operations for disk X, while another driver does so for disk Y.
- How to determine this API? What if a new device supports additional operations?

I/O subsystem

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- A key technique is to treat I/O components as objects, each with their own “methods” for implementing I/O API operations.
  - Allows new devices to be added later with little change to I/O subsystem code.
UNIX I/O Kernel Structure

System Call Layer

- Another abstraction boundary
  - Hides differences in device APIs from user application.
- Example: read() system call
  - Can perform on a file, a network socket, a message queue (pipe), a keyboard, ...
  - Some of these are block-oriented, some are character-oriented. I/O subsystem hides that fact from user
Block and Character Devices

- Block devices include disk drives
  - Commands include `read`, `write`, `seek`
  - Raw I/O or file-system access
  - Memory-mapped file access possible

- Character devices include keyboards, mice, serial ports
  - Commands include `get`, `put`
  - Libraries layered on top allow line editing

Network Devices

- Varying enough from block and character to have own interface

- Unix and Windows NT/9i/2000 include socket interface
  - Separates network protocol from network operation

- Implementation approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)
Clocks and Timers

- Provide current time, elapsed time, timer
- If programmable interval time used for timings, periodic interrupts
- `ioctl` (on UNIX) covers odd aspects of I/O such as clocks and timers

Blocking and Nonblocking I/O

- **Blocking** - process suspended until I/O completed
  - Easy to use and understand
  - Inhibits application-level concurrency
- **Nonblocking** - I/O call returns as much data as is available, fails, etc.
  - Returns count of bytes read or written
  - `select()` system call to poll
    - Used to implement user-level multi-threading
Example

- Read a key from the keyboard in GeekOS (keyboard.c)
- Non-blocking: Read_Key
- Blocking: Wait_For_Key

Asynchronous I/O

- Process runs while I/O executes
  - Event-driven notification: I/O subsystem signals process when I/O completed. For example, OS invokes a “callback” routine registered by the application at the time of I/O dispatch.
I/O Subsystem Duties

• Scheduling
  - Some I/O request ordering via per-device queue
  - Some OSs try fairness

• Buffering - store data in memory while transferring between devices
  - To cope with device speed mismatch
  - To cope with device transfer size mismatch
  - To maintain “copy semantics”

• Caching - fast memory holding copy of data
  - Always just a copy
  - Key to performance

• Spooling - hold output for a device
  - If device can serve only one request at a time
  - i.e., Printing

• Device reservation - provides exclusive access to a device
  - System calls for allocation and deallocation
  - Possibility of deadlock
Error Handling

• OS can recover from disk read, device unavailable, transient write failures

• Most return an error number or code when I/O request fails

• System error logs hold problem reports

I/O Requests to Hardware Operations

• Consider reading a file from disk for a process:
  - Determine device holding file
  - Translate name to device representation
  - Physically read data from disk into buffer
  - Make data available to requesting process
  - Return control to process
Life Cycle of An I/O Request

Performance

- I/O a major factor in system performance
  - Demands CPU to execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful
Intercomputer Communications

Improving Performance

- Reduce number of context switches
- Reduce data copying
- Reduce interrupts by using large transfers, smart controllers, polling
- Use DMA
- Balance CPU, memory, bus, and I/O performance for highest throughput
New Device-Functionality Progression

Increased time (generations) → Increased efficiency → Increased development cost → Increased abstraction

- New algorithm
- Application code
- Kernel code
- Device-driver code
- Device-controller code (hardware)
- Device code (hardware)

Increased flexibility