I/O Systems
CSCI 444/544 Operating Systems
Fall 2008

Agenda
- I/O Hardware
  - Storage devices (disks, tapes)
  - Transmission devices (network cards, modems)
  - Human-interface devices (monitor, keyboard, mouse)
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- Performance

Objectives
- Explore the structure of an operating system's I/O subsystem
- Discuss the principles of I/O hardware and its complexity
- Provide the performance aspects of I/O hardware and software

I/O Hardware
- Incredible variety of I/O devices
- Common concepts
  - Port
  - Bus (daisy chain or shared direct access)
  - Controller (host adapter)
- I/O instructions control devices
- Devices have addresses, used by
  - Direct I/O instructions
  - Memory-mapped I/O (no special I/O instructions needed)
A Typical PC Bus Structure

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>2F8–3FF</td>
<td>serial port (secondary)</td>
</tr>
<tr>
<td>300–30F</td>
<td>hard disk controller</td>
</tr>
<tr>
<td>378–37F</td>
<td>parallel port</td>
</tr>
<tr>
<td>300–30F</td>
<td>graphics controller</td>
</tr>
<tr>
<td>3F0–3FF</td>
<td>diskette-drive controller</td>
</tr>
<tr>
<td>3F8–3FF</td>
<td>serial port (primary)</td>
</tr>
</tbody>
</table>

Registers in an I/O Port

- Data-in register: is read by the host to get input
- Data-out register: is written by the host to send output
- Status register: contains bits indicating the states of the I/O device
- Control (Command) register: can be written by the host to start a command or change the mode

Polling

- Determines state of device
  - command-ready
    - Host signals its wish through this bit in command register
  - Busy
    - Controller indicates its state through this bit in status register
  - Error
- Busy-wait cycle to wait for I/O from device
Interrupts

- CPU Interrupt-request line (a wire) triggered by I/O device
- Interrupt handler receives interrupts
- **Maskable** to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Based on priority
  - Some nonmaskable
- Interrupt mechanism also used to support exceptions

Interrupt-Driven I/O Cycle

1. Device driver initiates I/O
2. CPU executes checks for interrupts between instructions
3. CPU receiving interrupt, generates interrupt acknowledgment
4. Input read, output write, or error generates interrupt signal
5. Interrupt handler processes data, initiates interruption
6. CPU resumes processing of interrupted task

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>nmi interrupt</td>
</tr>
<tr>
<td>3</td>
<td>breakpoint</td>
</tr>
<tr>
<td>4</td>
<td>I/O overload 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>divide 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>(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>19–31</td>
<td>reserved</td>
</tr>
<tr>
<td>32–255</td>
<td>reserved interrupts</td>
</tr>
</tbody>
</table>

Direct Memory Access

- Used to avoid **programmed I/O** for large data movement
- Requires **DMA** controller
- Bypasses CPU to transfer data directly between I/O device and memory
Six Step 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.

Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes.
- Device-driver layer hides differences among I/O controllers from kernel.
- 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

A Kernel I/O Structure

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/9x/2000 include socket interface
  - Separates network protocol from network operation
  - Includes `select` functionality
  - Manage a set of sockets
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)

Blocking and Nonblocking I/O

- Blocking (synchronous) - process suspended (waits) until I/O completed
  - Easy to use and understand
  - Insufficient for some needs
- Nonblocking - I/O call returns as much as available
  - User interface, data copy (buffered I/O)
  - Implemented via multi-threading
  - Returns quickly with count of bytes read or written
- Asynchronous - process runs while I/O executes
  - An asynchronous call returns immediately, without waiting for I/O
  - I/O subsystem signals process when I/O completed
    - Data transfer will be performed in its entirety but will complete at some future time

Two I/O Methods

- Synchronous
- Asynchronous

Kernel I/O Subsystem

- 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"
Kernel I/O Subsystem

- **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
  - Watch out for 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 Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
  - I/O must be performed via system calls
    - Memory-mapped and I/O port memory locations must be protected too

Kernel Data Structures

- Kernel keeps state info for I/O components, including open file tables, network connections, character device state
- Many, many complex data structures to track buffers, memory allocation, “dirty” blocks
- Some use object-oriented methods and message passing to implement I/O
UNIX I/O Kernel Structure

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
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

Device-Functionality Progression

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

Increased flexibility

Increased efficiency

Increased development cost

Increased duration