Chapter 13: I/O Systems

13.2 I/O Hardware

- Application I/O Interface
- Kernel I/O Subsystem
- Translating I/O Requests to Hardware Operations
- I/O Buffers
- Performance

13.3 Objectives

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

13.4 Overview

- I/O management is a major component of operating system design and operation
- Important aspect of computer operation
- I/O devices vary greatly
- Various methods to control them
- Performance management
- New types of devices frequent

- Ports, busses, device controllers connect to various devices
- Device drivers encapsulate device details
- Present uniform device-access interface to I/O subsystem

13.5 I/O Hardware

- Incredible variety of I/O devices
  - Storage
  - Transmission
  - Human interface
- Common concepts – signals from I/O devices interface with computer
  - Port – connection point for device
  - Bus – daisy chain or shared direct access
  - Controller (host adapter) – electronic that operate port, bus, device
    - Sometimes integrated
    - Sometimes separate circuit board (host adapter)
  - Contains processor, microcode, private memory, bus controller, etc.
- Some talk to per-device controller with bus controller, microcode, memory, etc.

13.6 A Typical PC Bus Structure
I/O Hardware (Cont.)

- I/O instructions control devices
- Devices usually have registers where device driver places commands, addresses, and data to write, or read data from
- Direct I/O instructions alter command register
- Devices have addresses, used by
  - Direct I/O instructions
  - Memory-mapped I/O
- Memory-mapped I/O
  - Device data and command registers mapped to processor address space
  - Especially for large address spaces (graphic)

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>020-02F</td>
<td>interrupt controller</td>
</tr>
<tr>
<td>040-043</td>
<td>timer</td>
</tr>
<tr>
<td>080-08F</td>
<td>game controller</td>
</tr>
<tr>
<td>1F0-1FF</td>
<td>serial port (secondary)</td>
</tr>
<tr>
<td>1F0-1FF</td>
<td>hard-disk controller</td>
</tr>
<tr>
<td>3C0-3FF</td>
<td>parallel port</td>
</tr>
<tr>
<td>3D0-3FF</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>

Polling

- For each type of I/O:
  1. Read busy bit from status register until 0
  2. Host sets read or write bit and if write copies data into data-out register
  3. Host sets command-ready bit
  4. Controller sets busy bit, executes transfer
  5. Controller clears busy bit, error bit, command-ready bit when transfer done

- Step 1 is busy-wait cycle to wait for I/O from device
- Reason able to be fast
- But inefficient if device slow
- CPU switches to other tasks
- But if miss a cycle data overwritten / lost

Interrupts

- Polling can happen in 3 instruction cycles
  - Read status, logical and to extract status bit branch if not zero
  - CPU interrupt-request bus triggered by I/O device
    - Checked by processor after each instruction
    - CPU interrupt requires bus triggred by I/O device
      - Check by processor after each instruction
      - CPU interrupt requires bus triggered by I/O device
    - Interrupt handler receives interrupts
      - Variable to ignore or delay some interrupts
      - Interrupt handler to dispatch interrupt to correct handler
      - Context switch at start and end
      - Based on priority
      - Some nonmaskable
      - Interrupt chaining if more than one device at same interrupt number

Interrupt-Driven I/O Cycle

Intel Pentium Processor Event-Vector Table
Interrupts (Cont.)

- Interrupt mechanism also used for exceptions
  - Terminate process, crash system due to hardware error
- Page fault executes when memory access error
- System call executes via trap to trigger kernel to execute request
- Multi-CPU systems can process interrupts concurrently
  - If operating system designed to handle it
- Used for time-sensitive processing; frequent, must be fast

Direct Memory Access

- Used to avoid programmed I/O (one byte at a time) for large data movement
- Requires DMA controller
  - Can bypass CPU to transfer data directly between I/O device and memory
- OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
  - Write location of command block to DMA controller
  - Bus mastering of DMA controller – grants bus from CPU
- When done, interrupts to signal completion

Six Step Process to Perform DMA Transfer

1. Device driver is told to transfer data from buffer at address Y to buffer at address X
2. Device driver tells DMA controller to transfer data from device to buffer at address X
3. DMA controller initiates DMA transfer
4. DMA controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C
6. When C = 0, DMA interrupts CPU for signal transfer completion

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
- New device bringing already-implemented protocol need no extra work
- Each OS has its own I/O subsystem structures and device driver frameworks
- Devices vary in many dimensions
  - Character-stream or block
  - Sequential or random-access
  - Synchronous or asynchronous (or both)
  - Sharable or dedicated
  - Speed of operation
  - Read/write, read only, or write only

Characteristics of I/O Devices

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Variation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data-transfer modes</td>
<td>character</td>
<td>block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>terabyte disk</td>
</tr>
<tr>
<td>Access method</td>
<td>sequential</td>
<td>random</td>
</tr>
<tr>
<td></td>
<td></td>
<td>modern CD-ROM</td>
</tr>
<tr>
<td>Transfer schedule</td>
<td>synchronous</td>
<td>asynchronous</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tape keyboard</td>
</tr>
<tr>
<td>Sharing</td>
<td>dedicated</td>
<td>sharable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tape keyboard</td>
</tr>
<tr>
<td>Device speed</td>
<td>idle</td>
<td>seek time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>transfer rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>disk between operations</td>
</tr>
<tr>
<td>I/O direction</td>
<td>read only</td>
<td>write only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CD-ROM, graphics controller, disk</td>
</tr>
</tbody>
</table>
Characteristics of I/O Devices (Cont.)

- Subtleties of devices handled by device drivers.
- Broadly, I/O devices can be grouped by the OS into:
  - Block I/O
  - Character I/O (Stream)
  - Memory-mapped file access
  - Network sockets
- For direct manipulation of I/O device specific characteristics, usually an escape / back door:
  - Unix `ioctl()` call to send arbitrary bits to a device control register and data to device data register

Block and Character Devices

- Block devices include disk drives:
  - Commands include read, write, seek
  - Block I/O, direct I/O, or file-based access
  - Memory-mapped file access possible
    - File mapped to virtual memory and blocks brought via demand paging
  - DMA
- 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
  - Separate network protocol from network operation
  - Includes `select()` functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, multiaxes)

Clocks and Timers

- Provide current time, elapsed time, timer
  - Normal resolution about 1/60 second
  - Some systems provide higher-resolution timers
- Programmable interval timer used for timing, periodic timespans.
  - `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
  - 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
  - `select()` to find if data ready then `read()` or `write()` to transfer
- Asynchronous - process runs while I/O executes
  - Difficult to use
  - I/O subsystems signal process when I/O completed

Two I/O Methods

- Synchronous
- Asynchronous
Kernel I/O Subsystem

- Scheduling
  - Some I/O request ordering via per-device queue
  - Some O/S by fairness
  - Some implement Quality Of Service (i.e. IPQOS)

- Buffering - store data in memory while transferring between devices
  - To align with device speed mismatch
  - To maintain “copy semantics”
  - Device-status Table

- Some I/O request ordering via per-device queue
- Some OSs try fairness
- Some implement Quality Of Service (i.e. IPQOS)

- Buffering - store data in memory while transferring between devices
  - To align with device speed mismatch
  - To maintain “copy semantics”
  - Device-status Table
  - Double buffering – two copies of the data
    - Kernel and user
    - Varying sizes
    - Full / being processed and not full / being used
    - Copy-on-write can be used for efficiency in some cases

Buffering
- To cope with device speed mismatch
- To cope with device transfer size mismatch
- To maintain “copy semantics”
- Double buffering - two copies of the data
  - Kernel and user
  - Varying sizes
  - Full / being processed and not full / being used
  - Copy-on-write can be used for efficiency in some cases

Kernel I/O Subsystem

- Caching – faster device holding copy of data
  - Always just a copy
  - Key to performance
  - Sometimes combined with buffering

- 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 de-allocation
  - Watch out for deadlock

Error Handling

- OS can recover from disk read, device unavailable, transient write failures
- Retry a read or write, for example
- Track error frequencies, stop using device with increasing frequency of retryable errors

- Most return an error number or code when I/O request fails
- System error log holds problem reports

I/O Protection

- User processes may accidentally or purposely 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 I/O Subsystem

- Device-status Table

<table>
<thead>
<tr>
<th>Device</th>
<th>Status</th>
<th>Request for</th>
<th>Operation</th>
<th>Address</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>Idle</td>
<td>way</td>
<td>read, write</td>
<td>35546</td>
<td>1372</td>
</tr>
<tr>
<td>Laser Printer</td>
<td>Busy</td>
<td>request</td>
<td>operation</td>
<td>37445</td>
<td>34512</td>
</tr>
<tr>
<td>Mouse</td>
<td>Idle</td>
<td>request</td>
<td>operation</td>
<td>37445</td>
<td>34512</td>
</tr>
<tr>
<td>Disk Unit 1</td>
<td>Idle</td>
<td>request</td>
<td>operation</td>
<td>37445</td>
<td>34512</td>
</tr>
<tr>
<td>Disk Unit 2</td>
<td>Busy</td>
<td>request</td>
<td>operation</td>
<td>37445</td>
<td>34512</td>
</tr>
</tbody>
</table>

Sun Enterprise 6000 Device-Transfer Rates

- Caching - faster device holding copy of data
  - Always just a copy
  - Key to performance
  - Sometimes combined with buffering

- 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 de-allocation
  - Watch out for deadlock

Error Handling

- OS can recover from disk read, device unavailable, transient write failures
- Retry a read or write, for example
- Track error frequencies, stop using device with increasing frequency of retryable errors

- Most return an error number or code when I/O request fails
- System error log holds problem reports

I/O Protection

- User processes may accidentally or purposely 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 I/O Subsystem

- Device-status Table

<table>
<thead>
<tr>
<th>Device</th>
<th>Status</th>
<th>Request for</th>
<th>Operation</th>
<th>Address</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>Idle</td>
<td>way</td>
<td>read, write</td>
<td>35546</td>
<td>1372</td>
</tr>
<tr>
<td>Laser Printer</td>
<td>Busy</td>
<td>request</td>
<td>operation</td>
<td>37445</td>
<td>34512</td>
</tr>
<tr>
<td>Mouse</td>
<td>Idle</td>
<td>request</td>
<td>operation</td>
<td>37445</td>
<td>34512</td>
</tr>
<tr>
<td>Disk Unit 1</td>
<td>Idle</td>
<td>request</td>
<td>operation</td>
<td>37445</td>
<td>34512</td>
</tr>
<tr>
<td>Disk Unit 2</td>
<td>Busy</td>
<td>request</td>
<td>operation</td>
<td>37445</td>
<td>34512</td>
</tr>
</tbody>
</table>
**Use of a System Call to Perform I/O**

1. Stop to monitor
2. kernel
3. perform I/O
4. return to user program

**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
- Windows uses message passing
- Message with I/O information passed from user mode into kernel
- Message modified as it flows through device driver and back to process
- Pros / cons?

**UNIX I/O Kernel Structure**

- System-wide open-file table
- File-system record
- Device record
- Per-process
- Read-write queues
- User process memory
- Kernel memory

**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

**STREAMS**

- STREAM - a full-duplex communication channel between a user-level process and a device in Unix System V and beyond
- A STREAM consists of:
  - STREAM head interfaces with the user process
  - Other modules between the device
- Each module contains a read queue and a write queue
- Message passing is used to communicate between queues
- Flow control option to indicate available or busy
- Asynchronous internally, synchronous where user process communicates with stream head
The STREAMS Structure

- In I/O, a major factor in system performance:
  - Device drivers execute device driver, kernel I/O code
  - Context switches due to interrupts
  - Data copying
  - Network traffic especially stressful

Intercomputer Communications

- Improve performance by:
  - Reducing number of context switches
  - Reducing data copying
  - Reducing interrupts by using large transfers, smart controllers, polling
  - Using DMA
  - Using smarter hardware devices
  - Balancing CPU, memory, bus, and I/O performance for highest throughput
  - Moving user-mode processes / daemons to kernel threads

Device-Functionality Progression

End of Chapter 12