CHAPTER 13: I/O SYSTEMS
Chapter 13: I/O Systems

- I/O Hardware
- Application I/O Interface
- Kernel I/O Subsystem
- Transforming I/O Requests to Hardware Operations
- Streams
- Performance
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
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
A Typical PC Bus Structure
I/O Transfer

- **I/O instruction**
  - Specify a byte or word to I/O port address.
  - Trigger bus lines to move bits into/out of device register.

- **Memory map I/O**
  - Map the device control registers as address space of the processor.
  - E.g. in graphics controller, memory-mapped methods can reduce millions of I/O instructions.
  - Have to take care about access of invalid memory addresses.
Registers of I/O ports

- **Data-in register**
  - Read by the host.

- **Data-out register**
  - Written by the host.

- **Status register (bits)**
  - Read by the host.
  - Completeness, error, availability, etc.

- **Control register**
  - Written by the host.
  - Issue commands or change the device mode.
## 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>
Polling

- Determines state of device
  - command-ready, busy, error.

- **Busy-wait** cycle to wait for I/O from device
  1. The host repeatedly reads the *busy bit* until it is clear.
  2. The host set the *write* bit in the *command register* and data to *data-out register*.
  3. The host sets the *command-ready* bits.
  4. The controller notices that the *command-ready bits* is set; it sets the *busy bits*.
  5. The controller reads the *command* and *data-out* registers and does the I/O.
  6. The controller clears the *command-ready bit* and the *error* bits to indicate the I/O success and clears the *busy* bit to indicate that it’s finished.
Interrupts

- CPU Interrupt-request line triggered by I/O device

- System save the status and jump to interrupt handler at a fixed address in memory.
  - Interrupt vector table: reduce the need to search all possible source.
  - Interrupt chaining: for systems with more devices.

- **Maskable** to ignore or delay some interrupts

- Interrupt vector to dispatch interrupt to correct handler
  - Based on priority
  - Some nonmaskable

- Interrupt mechanism also used for exceptions
Interrupt and exceptions

- Privileged kernel routines can be implemented by software interrupts (trap)

- Page faults
  - The interrupt suspends the current process and the page-fault handler fetches the page and resume the process.

- System calls
  - Save the user process status, switches to supervisor modes and dispatches to kernel routines for the requested service.
Interrupt-Driven I/O Cycle

1. CPU
   - device driver initiates I/O

2. I/O controller
   - initiates I/O

3. input ready, output complete, or error generates interrupt signal

4. CPU receiving interrupt, transfers control to interrupt handler

5. interrupt handler processes data, returns from interrupt

6. CPU resumes processing of interrupted task

7. CPU executing checks for interrupts between instructions
# 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>19–31</td>
<td>(Intel reserved, do not use)</td>
</tr>
<tr>
<td>32–255</td>
<td>maskable 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
- DMA through **DMA-request** and **DMA-acknowledge** wires.
  - CPU writes the address of source and destinations and byte numbers to DMA controller.
  - Device controller signals **DMA-request** when a word is available.
  - DMA controller seize the memory bus and place the desired address on the memory address wires and signals **DMA-acknowledge**.
  - Device controller than transfers the word to memory and clear **DMA-request** signal.
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
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
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

- **Kernel I/O Subsystem**:
  - **Software**:
    - SCSI device driver
    - Keyboard device driver
    - Mouse device driver
    - …
    - PCI bus device driver
    - Floppy device driver
    - ATAPI device driver
  - **Hardware**:
    - SCSI device controller
    - Keyboard device controller
    - Mouse device controller
    - …
    - PCI bus device controller
    - Floppy device controller
    - ATAPI device controller
    - SCSI devices
    - Keyboard
    - Mouse
    - …
    - PCI bus
    - Floppy-disk drives
    - ATAPI devices (disks, tapes, drives)
# 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 block</td>
<td>terminal disk</td>
</tr>
<tr>
<td>access method</td>
<td>sequential random</td>
<td>modem CD-ROM</td>
</tr>
<tr>
<td>transfer schedule</td>
<td>synchronous asynchronous</td>
<td>tape keyboard</td>
</tr>
<tr>
<td>sharing</td>
<td>dedicated sharable</td>
<td>tape keyboard</td>
</tr>
<tr>
<td>device speed</td>
<td>latency, seek time, transfer rate, delay between operations</td>
<td></td>
</tr>
<tr>
<td>I/O direction</td>
<td>read only, write only, read–write</td>
<td>CD-ROM graphics controller disk</td>
</tr>
</tbody>
</table>
Block and Character Devices

- **Block devices include disk drives**
  - Commands include read, write, seek
    - Raw I/O: handled by applications
    - File-system access: handled by OS.
  - Memory-mapped file access possible

- **Character devices include keyboards, mice, serial ports**
  - Commands include `get`, `put`
  - Libraries layered on top allow buffering and 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

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

- Provide current time, elapsed time, timer

- **Programmable interval timer** used for timings, periodic interrupts

- Support more timer requests than the number of timer by “virtual clocks”.
  - Sort the request in the earliest-time-first order
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 quick 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
  - Difficult to use
  - I/O subsystem signals process when I/O completed
Two I/O Methods

(a) Synchronous and (b) asynchronous I/O
Kernel I/O Subsystem

- Kernels provides: scheduling, buffering, caching, spooling, error handling, etc.

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

---

Device-status Table

- device: keyboard
  - status: idle
- device: laser printer
  - status: busy
- device: mouse
  - status: idle
- device: disk unit 1
  - status: idle
- device: disk unit 2
  - status: busy
  - : : 

Requests:
- request for laser printer
  - address: 38546
  - length: 1372
- request for disk unit 2
  - file: xxx
  - operation: read
  - address: 43046
  - length: 20000
- request for disk unit 2
  - file: yyy
  - operation: write
  - address: 03458
  - length: 500
Kernel I/O Subsystem (cont.)

- 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” : e.g. guarantee write() to buffer instead of the disk
Kernel I/O Subsystem (cont.)

- **Caching** - fast memory holding copy of data
  - Always just a copy
  - Key to performance
  - Different policy for update: write-through, delayed write, etc.

- **Spooling** - hold output for devices
  - if a device cannot accept interleaved data streams, e.g. printer, tape drives.
  - Each application’s output is spooled to a separate disk file.

- **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
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

- Efficiency vs. protection: memory-mapped graphics controller memory.
Use of a System Call to Perform I/O
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
Consider reading a file from disk for a process:

- Check the parameters’ correctness.
- Check whether the data are available in the buffer cache.
- If it is not available, place the user process to the device’s waiting queue.
- I/O subsystem sends request to the device driver.
- The device driver allocates kernel buffer space and send commands to the device controller’s registers.
- The device controller operates the data transfer.
- Polling or interrupts for checking data completion. (interrupt handler signal the corresponding device driver.)
- The device driver signals the I/O subsystem for later data transferring to the requesting process.
- Moving the requesting process to the ready queue.
Life Cycle of An I/O Request

1. request I/O
   - system call
   - user process
   - I/O completed, input data available, or output completed

2. can already satisfy request?
   - yes: transfer data (if appropriate) to process, return completion or error code
   - no: send request to device driver, block process if appropriate

3. process request, issue commands to controller, configure controller to block until interrupted
   - device-controller commands
   - monitor device, interrupt when I/O completed

4. receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
   - interrupt handler
   - interrupt

5. determine which I/O completed, indicate state change to I/O subsystem
   - I/O completed, generate interrupt
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
  - driver end interfaces with the device
  - zero or more STREAM modules between them.

- Each module contains a **read queue** and a **write queue**

- Message passing is used to communicate between queues
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
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