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)

- Devices have **addresses** (logical bus address), used by
  - **Direct I/O instructions** (to registers on device/controller)
    - Typically: status, control, data-in, data-out registers
  - **Memory-mapped I/O**
  - Some systems / architectures / devices use **both**
    - E.g., a graphics controller may use ports for basic operations, but a memory-mapped region for screen data
## 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

- One method of performing I/O
  - Typically: status, control (command), data-in, data-out registers

- Poll to determine state of device (check status bits)
  - command-ready
  - busy
  - error

- Busy-wait cycle to wait for I/O from device

- When device is ready
  - Host sets write in command register
  - Host writes a byte into the data-out buffer
  - Host sets the command-ready bit
  - When controller notices command-ready, sets busy bit
  - Controller reads command register, interprets command, reads data-out register, writes data to the device
  - Controller clears command-ready bit, clears error bit, clears busy bit
Interrupts

- **Alternative** I/O technique, eliminates busy wait and polling

- General **process** of handling interrupts for I/O
  - CPU **Interrupt request** line triggered by I/O device
    - This is a hardware signal
  - **Interrupt handler** (software) receives interrupts
    - Somewhat like an exception handler
    - Interrupts can be unmaskable or maskable; have priority
  - **Interrupt vector** used to dispatch interrupt to correct handler
    - Prioritized, so handler execution can be interrupted
    - Some unmaskable

- Interrupt mechanism **also used for exceptions and traps**
  (software interrupts or system calls)
Interrupt-Driven I/O Cycle

1. CPU
2. device driver initiates I/O
3. I/O controller
4. CPU executing checks for interrupts between instructions
5. initiates I/O
6. CPU receiving interrupt, transfers control to interrupt handler
7. input ready, output complete, or error generates interrupt signal
8. interrupt handler processes data, returns from interrupt
9. 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>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
- Increases throughput for CPU and system
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

- Additional level of abstraction
  - Insert device driver between I/O system (in kernel) and devices
  - I/O system calls encapsulate device behaviors in generic classes

- Kernel I/O subsystem uses standard APIs (for a particular OS) to communicate with device drivers
  - Device-driver layer hides differences among I/O controllers from kernel – handles communication between kernel and controllers
### A Kernel I/O Structure

<table>
<thead>
<tr>
<th>hardware</th>
<th>kernel I/O subsystem</th>
<th>software</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSI device controller</td>
<td></td>
<td>kernel</td>
</tr>
<tr>
<td>keyboard</td>
<td>PCI bus device driver</td>
<td></td>
</tr>
<tr>
<td>mouse</td>
<td>floppy device driver</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATAPI device driver</td>
<td></td>
</tr>
<tr>
<td>SCSI devices</td>
<td></td>
<td></td>
</tr>
<tr>
<td>keyboard</td>
<td>PCI bus controller</td>
<td></td>
</tr>
<tr>
<td>mouse</td>
<td>floppy-disk drives</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATAPI devices</td>
<td></td>
</tr>
<tr>
<td>keyboard</td>
<td>PCI bus controller</td>
<td></td>
</tr>
<tr>
<td>mouse</td>
<td>floppy-disk drives</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATAPI devices</td>
<td></td>
</tr>
</tbody>
</table>
Device Characteristics

- 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

- Must be **considered** when
  - Writing device drivers
  - Using devices
### 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</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></td>
</tr>
<tr>
<td></td>
<td>seek time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>transfer rate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>delay between operations</td>
<td></td>
</tr>
<tr>
<td>I/O direction</td>
<td>read only</td>
<td>CD-ROM</td>
</tr>
<tr>
<td></td>
<td>write only</td>
<td>graphics controller</td>
</tr>
<tr>
<td></td>
<td>read–write</td>
<td>disk</td>
</tr>
</tbody>
</table>
Block and Character Devices

- **Block devices** include disk drives
  - Commands include `read()`, `write()`, `seek()`
  - Raw I/O or **file-system access**
    - Most use the file system, except for some OS code / functions
  - Memory-mapped file access possible

- **Character (stream) devices** include keyboards, mice, serial ports
  - Commands include `get()`, `put()`
  - Libraries layered on top allow line editing
Network Devices

■ **Varies** enough from block and character to have own interface

■ Many operating systems include a **socket** interface
  - Separates network protocol from network operation
  - Includes `select()` functionality
    - Returns info indicating which sockets have packets waiting and which sockets have room to accept a new packet
    - Eliminates polling and busy waits
    - Can block while waiting

■ **Approaches** vary widely (pipes, FIFOs, streams, queues, mailboxes)
  - Can look and function quite similar to these familiar structures
Clocks and Timers

- **Hardware** functions
  - Provide *current time, elapsed time, timer*
  - Heavily used by OS and some applications

- **Programmable interval timer** used for timings, periodic interrupts
  - Used by *scheduler* with time slices
  - Flushing *caches, timeouts, wait/sleep*, etc.

- Timer is often rather **coarse resolution**, compared to actual system clock speed
  - E.g., only 18-60 ticks per second
    - Must take into consideration
  - Often an instruction also exists which will retrieve the actual system time, if need greater resolution
Blocking and Nonblocking I/O

- **Blocking/Synchronous** - process suspended until I/O completed
  - Easy to use and understand
  - Insufficient for some needs, required by others
  - Reduces performance and throughput, unless required
  - Can use multi-threading to improve or simulate asynchronous I/O

- **Nonblocking** - I/O call returns as much as available (e.g., kbd, mouse, video, audio – think of a browser window)
  - 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
  - Sometimes can be difficult to use correctly and well
  - I/O subsystem signals process when I/O completed
Two I/O Methods

(a) Synchronous

(b) Asynchronous
Kernel I/O Subsystem Services

- **Scheduling**
  - Way to improve throughput and I/O performance
  - Some I/O request ordering via per-device queue
  - Some OSs try for 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”
  - *Double buffering* can be used to decouple producer from consumer
    - See following pages for different device speed characteristics
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

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
Sun Enterprise 6000 Device-Transfer Rates

- Gigaplane bus
- SBUS
- SCSI bus
- Fast ethernet
- Hard disk
- Ethernet
- Laser printer
- Modem
- Mouse
- Keyboard
Kernel I/O Subsystem

- **Caching** - fast memory holding copy of data
  - Different from buffering – to reduce unnecessary I/Os
  - Always just a copy – must manage refreshing and writing out (who manages depends on where located)
  - Important impact on performance

- **Spooling** - hold output for a device
  - Special kind of buffering – to serialize I/O
  - If device can serve only one request at a time
    - i.e., printer, tape

- **Device reservation** - provides exclusive access to device
  - Used when device can service only one request at a time
    - e.g., lock tape drive to particular process
  - System calls for allocation and deallocation
  - Watch out for deadlock
Error Handling

- OS **must be able to handle** read, device unavailable, transient write failures

- Initially **try to recover** – retries, timeouts

- If **unrecoverable** error, must return an error number or code (errno) when I/O request fails (or throw an exception)

- **Log** error and related information in system error logs

- System must handle error **gracefully** (if can)

- Good OSs will provide enough **information** to debug and correct the problem easily
I/O Protection

- User process may accidentally or intentionally 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
  - This means that all I/O must be done by OS

- Discussed this at beginning of course
Use of a System Call to Perform I/O

1. Trap to monitor
2. Perform I/O
3. Return to user

System call $n$
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

- File descriptor
- Per-process open-file table
- User-process memory
- System-wide open-file table
  - File-system record
    - Inode pointer
    - Pointer to read and write functions
    - Pointer to select function
    - Pointer to ioctl function
    - Pointer to close function
  - Networking (socket) record
    - Pointer to network info
    - Pointer to read and write functions
    - Pointer to select function
    - Pointer to ioctl function
    - Pointer to close function
- Kernel memory
- Active-inode table
- Network-information table
What Happens During an I/O Request?

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

1. User process requests I/O.

2. System call.

3. Kernel I/O subsystem is called.

4. Can already satisfy request? If yes, transfer data (if appropriate) to process, return completion or error code. If no, send request to device driver, block process if appropriate.

5. Process request, issue commands to controller, configure controller to block until interrupted.

6. Device controller commands.

7. Monitor device, interrupt when I/O completed.

8. Receive interrupt, store data in device-driver buffer if input, signal to unblock device driver.

9. Determine which I/O completed, indicate state change to I/O subsystem.

10. I/O completed, generate interrupt.

11. Time.
STREAMS

- **STREAM** – a full-duplex communication channel between a user-level process and a device (actually, a device driver)

- A **STREAM** consists of:
  - **STREAM head**, which interfaces with the user process
  - **Driver end**, which 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

- Stream I/O is **asynchronous**, except when the user process communicates with the stream head
Performance

- I/O a major factor in *system performance*:
  - Requires **CPU cycles** to execute device driver, kernel I/O code
  - Results in **context switches** due to interrupts
  - Involves much **data copying**
  - Network traffic especially stressful
Intercomputer Communications

![Diagram showing intercomputer communications process](image-url)
Improving Performance

- Reduce number of **context switches**
- Reduce **data copying**
- Reduce **interrupts** by using large transfers, smart controllers, **polling**
- Use **DMA** to increase concurrency
- **Balance** CPU, memory, bus, and I/O performance for highest throughput
- **Reduce** number of opens / closes, setup / teardown, etc.
  - Threading / thread pools
End of Chapter 13