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

- Monitor
- Processor
- Graphics controller
- Bridge/memory controller
- Cache
- Memory
- SCSI controller
- IDE disk controller
- Disk
- Expansion bus interface
- Expansion bus
- Parallel port
- Serial port
- Keyboard

PCI bus

SCSI bus
## 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
Interrupts

- CPU **Interrupt-request line** 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 for exceptions
Interrupt-Driven I/O Cycle

1. CPU
   - device driver initiates I/O
   - CPU executing checks for interrupts between instructions
   - CPU receiving interrupt, transfers control to interrupt handler
   - interrupt handler processes data, returns from interrupt
   - CPU resumes processing of interrupted task

2. I/O controller
   - initiates I/O

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

4. 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
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
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

The kernel I/O structure is organized as follows:

- **Software**
  - **Kernel I/O Subsystem**
    - SCSI device driver
    - keyboard device driver
    - mouse device driver
    - ... (multiple entries)
    - PCI bus device driver
    - floppy device driver
    - ATAPI device driver

- **Hardware**
  - SCSI device controller
  - keyboard device controller
  - mouse device controller
  - ... (multiple entries)
  - PCI bus device controller
  - floppy device controller
  - ATAPI device controller

- Devices:
  - SCSI devices
  - keyboard
  - mouse
  - ... (multiple entries)
  - 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></td>
<td>block</td>
<td></td>
</tr>
<tr>
<td>access method</td>
<td>sequential random</td>
<td>modem</td>
</tr>
<tr>
<td></td>
<td>random</td>
<td>CD-ROM</td>
</tr>
<tr>
<td>transfer schedule</td>
<td>synchronous asynchronous</td>
<td>tape</td>
</tr>
<tr>
<td></td>
<td>asynchronous</td>
<td>keyboard</td>
</tr>
<tr>
<td>sharing</td>
<td>dedicated sharable</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>
<tr>
<td>I/O direction</td>
<td>read only</td>
<td></td>
</tr>
<tr>
<td></td>
<td>write only</td>
<td></td>
</tr>
<tr>
<td></td>
<td>read–write</td>
<td></td>
</tr>
</tbody>
</table>

*Operating System Concepts*
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 \texttt{get}, \texttt{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

- 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

- 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

- **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

(b) 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”
### Device-status Table

<table>
<thead>
<tr>
<th>Device</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>idle</td>
</tr>
<tr>
<td>Laser Printer</td>
<td>busy</td>
</tr>
<tr>
<td>Mouse</td>
<td>idle</td>
</tr>
<tr>
<td>Disk Unit 1</td>
<td>idle</td>
</tr>
<tr>
<td>Disk Unit 2</td>
<td>busy</td>
</tr>
</tbody>
</table>

- **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

Transfer Rates:

0 0.01 0.1 1 10 100 1,000 10,000 100,000 1,000,000
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
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
Use of a System Call to Perform I/O

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

Case $n$

System call $n$

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
UNIX I/O Kernel Structure

- **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
    - ...

- **Active-inode table**

- **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**

- **User-process memory**
  - Per-process open-file table
  - File descriptor

Operating System Concepts
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

- request I/O
- system call
- can already satisfy request?
  yes
  - I/O completed, input data available, or output completed
  - return from system call
  - transfer data (if appropriate) to process, return completion or error code
- no
  - send request to device driver, block process if appropriate
  - kernel I/O subsystem

- process request, issue commands to controller, configure controller to block until interrupted
  - device controller commands
  - device driver
  - device controller
  - receive interrupt, store data in device-driver buffer if input, signal to unblock device driver

- monitor device, interrupt when I/O completed
  - device controller
  - I/O completed, generate interrupt

- time
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
The STREAMS Structure

user process

stream head

read queue | write queue

read queue | write queue

read queue | write queue

read queue | write queue

driver end

device
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

- Character typed
  - Hardware
    - Interrupt generated
      - User process
        - Context switch
          - Sending system
            - Kernel
              - Context switch
                - System call completes
                  - Interrupt handled
                    - Device driver
                      - State save
                        - Kernel
                          - Context switch
                            - Network daemon
                              - Context switch
                                - Network subdaemon
                                  - Kernel
                                    - Network packet received
                                      - Network adapter
                                        - Device driver
                                          - State save
                                            - Kernel
                                              - Context switch
                                                - Receiving system
                                                  - Network adapter
                                                    - Device driver
                                                      - State save
                                                        - Kernel
                                                          - Context switch
                                                            - Network subdaemon
                                                              - Kernel
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
End of Chapter 13