I/O AND DEVICE HANDLING
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
- Graphics Controller
- Processor
  - Bridge/Memory Controller
  - Cache
  - Memory
- IDE Disk Controller
  - Disk
  - Disk
  - Disk
- SCSI Controller
  - Disk
  - Disk
  - Disk
- Expansion Bus Interface
- Keyboard
- Parallel Port
- Serial Port

PCI Bus

SCSI Bus
## Device I/O Port Locations on PCs

<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>
Port-Mapped VS Memory-Mapped

- Each device connected to I/O bus has its own I/O addresses called ports
- X86 provides I/O address space of 64KB I/O ports
  - 1, 2 and 4 bytes can be grouped
  - Accessing with in, out, ins and outs instructions
- Memory mapped I/O is favored
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
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
A Kernel I/O Structure

- **Kernel I/O Subsystem**
  - Kernel
  - **Software**
    - SCSI device driver
    - Keyboard device driver
    - Mouse device driver
    - ... (ellipsis)
    - PCI bus device driver
    - Floppy device driver
    - ATAPI device driver

- **Hardware**
  - SCSI device controller
  - Keyboard device controller
  - Mouse device controller
  - ... (ellipsis)
  - PCI bus device controller
  - Floppy device controller
  - ATAPI device controller

- **Devices**
  - SCSI devices
  - Keyboard
  - Mouse
  - ... (ellipsis)
  - PCI bus
  - Floppy-disk drives
  - ATAPI devices (disks, tapes, drives)
Use of a System Call to Perform I/O
Life Cycle of An I/O Request

1. User process requests I/O.
2. System call is made.
3. Kernel I/O subsystem checks if request can be satisfied.
   - If yes, transfer data and return.
   - If no, send request to device driver.
4. Device driver processes request, issues commands to controller, configures controller to block until interrupted.
5. Device controller generates interrupt when I/O completed.
6. Device driver receives interrupt, stores data in device-driver buffer if input, signals to unblock device driver.
7. I/O completed, generate interrupt.
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

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

active-inode table

network-information table
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
/sys Filesystem

- A special filesystem similar to /proc
- Provides information on kernel internal data structures
- /sys/block, /sys/devices/, /sys/bus, /sys/drivers, /sys/class, /sys/power and so on
- Example
  - /sys/block/sda/device -> /sys/devices/pci0000:00
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

- 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
UNIX-like operating systems are based on the notion of a file, which is just an information container structured as a sequence of bytes.

I/O devices are treated as special files called device files.

Device files can be of two types: block or character.
- Major number identifies device type (12 bits)
- Minor number identifies specific device (20 bits)
- `Documentation/devices.txt` illustrates mappings between major/minor numbers and device types
- `/sys/class` are used to pass dynamic mapping information

Network cards are a notable exception to this.
- They are not directly associated with device files.
VFS Handling of Device Files

- Virtual file system layer differentiates accesses to the device files from that to the regular files.
- But standard file-related system calls do not always give applications full control of underlying hardware devices.
  - `ioctl`
Device Drivers

- A set of kernel routines that makes a hardware device respond to programming interface
- Programming interface is defined by canonical set of VFS functions
- Actual implementation of all these functions is delegated to device driver
- Register itself during initialization and discover corresponding devices
Polling

- Determines state of device
  - command-ready
  - busy
  - Error

- Busy-wait cycle to wait for I/O from device
  - Timeout with counter
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

2. I/O controller
   - Initiates I/O

3. CPU executing checks for interrupts between instructions

4. CPU receiving interrupt, transfers control to interrupt handler
   - Interrupt handler processes data, returns from interrupt

5. CPU resumes processing of interrupted task
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
  - Cache coherency?
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.
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
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