I/O AND DEVICE HANDLING

2016 Operating Systems Design
Euiseong Seo (euiseong@skku.edu)
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
- Expansion bus interface
- Keyboard
- Expansion bus
- Parallel port
- Serial port
- Disks
## 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>
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
  - Accessed 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)
- decreased 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

The diagram illustrates the kernel I/O structure in a computer system. The kernel is at the top, with the kernel I/O subsystem in the middle. Below the kernel I/O subsystem are various device drivers and controllers, including:

- SCSI device driver
- keyboard device driver
- mouse device driver
- PCI bus driver
- floppy device driver
- ATAPI device driver

On the hardware level, below the device drivers and controllers, there are:

- SCSI devices
- keyboard
- mouse
- 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. request I/O
   - user process
   - return from system call
2. system call
   - I/O completed, input data available, or output completed
   - kernel I/O subsystem
   - transfer data (if appropriate) to process, return completion or error code
3. can already satisfy request?
   - yes
   - kernel I/O subsystem
   - determine which I/O completed, indicate state change to I/O subsystem
   - receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
   - interrupt
   - device controller
   - I/O completed, generate interrupt
   - device-controller commands
   - device-driver
   - process request, issue commands to controller, configure controller to block until interrupted

4. no
   - send request to device driver, block process if appropriate
   - monitor device, interrupt when I/O completed
   - time
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:
- 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. Device driver initiates I/O
2. 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
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. The device driver is told to transfer disk data to buffer at address X.
2. The device driver tells the disk controller to transfer C bytes from disk to buffer at address X.
3. The disk controller initiates DMA transfer.
4. The disk controller sends each byte to DMA controller.
5. The DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0.
6. When C = 0, DMA interrupts the 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**

- **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
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