I/O Subsystem

COMS W4118
Prof. Kaustubh R. Joshi
krj@cs.columbia.edu

http://www.cs.columbia.edu/~krj/os

References: Operating Systems Concepts (9e), Linux Kernel Development, previous W4118s
Copyright notice: care has been taken to use only those web images deemed by the instructor to be in the public domain. If you see a copyrighted image on any slide and are the copyright owner, please contact the instructor. It will be removed.
I/O Subsystem

• Goals
• Architecture
• Device Characteristics
• OS Mechanisms
  – Transferring data
  – Notification
  – Buffering
The Requirements of I/O

• Without I/O, computers are not very useful

• But... thousands of devices, each slightly different
  – How to standardize the interface to all devices?

• Devices unpredictable and/or slow
  – How can we utilize them if we dont know what they will do or how they will perform?

• Devices unreliable: media failures, transmission errors
  – How to make them reliable?
Varied I/O Speeds

Some typical device, network, and bus data rates.

- Device Rates vary over many orders of magnitude
  - System better be able to handle this wide range
  - Better not have high overhead/byte for fast devices!
  - Better not waste time waiting for slow devices

<table>
<thead>
<tr>
<th>Device</th>
<th>Data rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>10 bytes/sec</td>
</tr>
<tr>
<td>Mouse</td>
<td>100 bytes/sec</td>
</tr>
<tr>
<td>56K modem</td>
<td>7 KB/sec</td>
</tr>
<tr>
<td>Scanner</td>
<td>400 KB/sec</td>
</tr>
<tr>
<td>Digital camcorder</td>
<td>3.5 MB/sec</td>
</tr>
<tr>
<td>802.11g Wireless</td>
<td>6.75 MB/sec</td>
</tr>
<tr>
<td>52x CD-ROM</td>
<td>7.8 MB/sec</td>
</tr>
<tr>
<td>Fast Ethernet</td>
<td>12.5 MB/sec</td>
</tr>
<tr>
<td>Compact flash card</td>
<td>40 MB/sec</td>
</tr>
<tr>
<td>FireWire (IEEE 1394)</td>
<td>50 MB/sec</td>
</tr>
<tr>
<td>USB 2.0</td>
<td>60 MB/sec</td>
</tr>
<tr>
<td>SONET OC-12 network</td>
<td>78 MB/sec</td>
</tr>
<tr>
<td>SCSI Ultra 2 disk</td>
<td>80 MB/sec</td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>125 MB/sec</td>
</tr>
<tr>
<td>SATA disk drive</td>
<td>300 MB/sec</td>
</tr>
<tr>
<td>Ultrium tape</td>
<td>320 MB/sec</td>
</tr>
<tr>
<td>PCI bus</td>
<td>528 MB/sec</td>
</tr>
</tbody>
</table>

Table from Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639
Varied Device Characteristics

• Some operational parameters:
  – Byte/Block
    • Some devices provide single byte at a time (e.g. keyboard)
    • Others provide whole blocks (e.g. disks, tapes, etc)
  – Sequential/Random
    • Some devices must be accessed sequentially (e.g. tape)
    • Others can be accessed randomly (e.g. disk, cd, etc.)
  – Polling/Interrupts
    • Some devices require continual monitoring
    • Others generate interrupts when they need service
The Goal of the I/O Subsystem

• Provide uniform Interface for wide range of devices
  – This code works on many different devices:
    ```c
    int fd = open("/dev/something");
    for (int i = 0; i < 10; i++) {
        fprintf(fd, "Count %d\n",i);
    }
    close(fd);
    – Why? Because code that controls devices ("device driver") implements standard interface.
I/O Subsystem

• Goals

• Architecture

• Device Characteristics

• OS Mechanisms
  – Transferring data
  – Notification
  – Buffering
Review: I/O Architecture
- **Northbridge:**
  - Handles memory
  - Graphics

- **Southbridge:**
  - PCI bus
  - Disk controllers
  - USB controllers
  - Audio
  - Serial I/O
  - Interrupt controller
  - Timers
• **CPU interacts with device Controller**
  – Contains read/write **control registers**
  – May contain memory for request queues or bit-mapped images

• **Accessed by processor in two ways:**
  – **I/O instructions**: explicit in/out instructions
    • E.g., x86: out 0x21,AB
  – **Memory mapped I/O**: load/store instructions
    • Registers/memory appear in physical address space
    • I/O accomplished with load and store instructions
Memory-Mapped vs. Explicit I/O

• Explicit I/O Instructions:
  – Must use assembly language
  – Prevents user-mode I/O

• Memory-Mapped I/O:
  – No need for special instructions (can use in C)
  – Allows user-based I/O
  – Caching addresses must be prevented
## 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>
(a) Separate I/O and memory space.
(b) Memory-mapped I/O.
(c) Hybrid: control in I/O address, data in memory
Memory-Mapped I/O

(a) A single-bus architecture.
(b) A dual-bus memory architecture.
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 Drivers

Device-specific code in the kernel that interacts directly with the device hardware

- Supports a standard, internal interface
- Same kernel I/O system can interact easily with different device drivers
- Special device-specific configuration supported with the ioctl() system call
A Kernel I/O Structure

Increased specificity

Increased flexibility, reusability

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

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)
Kernel keeps state for I/O components, including open file tables, network connections, character device state.
Device Driver Structure

- Device Drivers typically divided into two pieces:
  - Top half: accessed in call path from system calls
    - Implements a set of standard, cross-device calls like open(), close(), read(), write(), ioctl()
    - Implement special VMAs to support mmap() based I/O
    - This is the kernel’s interface to the device driver
    - Top half will start I/O to device, may put thread to sleep until finished
  - Bottom half: run as interrupt routine
    - Gets input or transfers next block of output
    - May wake sleeping threads if I/O now complete
    - May use deferred processing (softirq, tasklet, kernel threads)
Life Cycle of An I/O Request

User Program

Kernel I/O or memory Subsystem

Device Driver Top Half

Device Driver Bottom Half

Device Hardware
I/O Subsystem

• Goals
• Architecture

• Device Characteristics
• OS Mechanisms
  – Transferring data
  – Notification
  – Buffering
# 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></td>
<td></td>
</tr>
<tr>
<td>access method</td>
<td>sequential random</td>
<td>modem</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CD-ROM</td>
</tr>
<tr>
<td>transfer schedule</td>
<td>synchronous asynchronous</td>
<td>tape</td>
</tr>
<tr>
<td></td>
<td></td>
<td>keyboard</td>
</tr>
<tr>
<td>sharing</td>
<td>dedicated sharable</td>
<td>tape</td>
</tr>
<tr>
<td></td>
<td></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
  - Memory-mapped file access possible

- **Character devices** include keyboards, mice, serial ports
  - Single character at a time
  - Commands include get, put
  - Libraries layered on top allow line editing
Network Devices

• Different 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)
Clock Devices and Timers

• Maintaining the time of day
• Accounting for CPU usage
• Preventing processes from running longer than they are allowed to
• Handling alarm system call made by user processes
• Providing watchdog timers for parts of the system itself.
• Doing profiling, monitoring, statistics gathering
A programmable clock.

Crystal oscillator

Counter is decremented at each pulse

Holding register is used to load the counter
I/O Subsystem

• Goals
• Architecture
• Device Characteristics

• OS Mechanisms
  – Transferring data
  – Notification
  – Buffering
Transferring Data to/from Controller

• **Programmed I/O:**
  – Each byte transferred via processor in/out or load/store
  – Pro: Simple hardware, easy to program
  – Con: Consumes processor cycles proportional to data size

• **Direct Memory Access:**
  – Give controller access to memory bus
  – Ask it to transfer data to/from memory directly
Programmed I/O

Writing a string to the printer using programmed I/O.

copy_from_user(buffer, p, count);
for (i = 0; i < count; i++) {
    while (*printer_status_reg != READY);
    *printer_data_register = p[i];
}
return_to_user();

/* p is the kernel buffer */
/* loop on every character */
/* loop until ready */
/* output one character */
Direct Memory Access (DMA)

- 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
Notifying the OS: Polling

• The OS needs to know when:
  – The I/O device has completed an operation
  – The I/O operation has encountered an error

• One way is to **Poll**:
  – OS periodically checks a device-specific status register
  – I/O device puts completion information in status register
  – **Busy-wait** cycle to wait for I/O from device
  – Pro: simple, potentially low overhead
  – Con: may waste many cycles on polling if infrequent, expensive, or unpredictable I/O operations
Notifying the OS: Interrupts

• I/O Interrupt:
  – Device generates an interrupt whenever it needs service
  – Pro: handles unpredictable events well
  – Con: interrupts relatively high overhead

• Some devices combine both polling and interrupts
  – Example: Intel E1000 Gigabit Ethernet Adapter:
    • Interrupt for first incoming packet
    • Poll for subsequent packets until hardware packet arrival queue is empty
I/O 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

• Improving performance:
  – Use DMA
  – Reduce/eliminate data copying
  – Reduce number of context switches
  – Reduce interrupts by using large transfers, smart controllers, polling
Blocking and Nonblocking I/O

• **Blocking Interface:** “Wait”
  – Put process to sleep until data is ready (for read) or written (for write)

• **Non-blocking Interface:** “Don’t Wait”
  – Returns quickly from read or write request with count of bytes successfully transferred
  – Read may return nothing, write may write nothing

• **Asynchronous Interface:** “Tell Me Later”
  – When request data, take pointer to user’s buffer, return immediately; later kernel fills buffer and notifies user
  – When send data, take pointer to user’s buffer, return immediately; later kernel takes data and notifies user
Asynchronous I/O

Synchronous

Asynchronous
Kernel I/O Subsystem

• Common Interfaces for
  – **Device reservation** - exclusive access to a device
    • System calls for allocation and deallocation
    • Watch out for deadlock
  – **Caching** - fast memory holding copy of data
    • Always just a copy
    • Key to performance
  – **Scheduling** – I/O request reordering
    • Via per-device queue, goal: fairness
  – **Spooling** - hold a copy of output for a device
    • If device can serve one request at a time, e.g., printing
• 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”
Buffering

- Unbuffered input
- Buffering in user space
- Buffering in the kernel followed by copying to user space
- Double buffering in the kernel.

Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639
Buffering

Networking may involve many copies of a packet. May reduce performance.
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
Interaction of I/O and VM

• The kernel deals with (kernel) virtual addresses
• These do not necessarily correspond to physical addresses
• Contiguous virtual addresses are probably not contiguous in physical memory
• Some systems have an I/O map — the I/O bus manager has a (version of) the virtual memory map
• More often, the kernel has to translate the virtual addresses itself
Other I/O Issues

- **Scatter/Gather I/O**
  - Suppose we’re reading a single packet or disk block into two or more non-contiguous pages
  - The I/O transfer has to use more than one (address, length) pair for that transfer to scatter the data around memory
  - The same applies on output, where it has to be gathered from different physical pages

- **Direct I/O**
  - For efficiency, we may want to avoid copying data to/from user space
  - Sometimes possible to do direct I/O
  - Must consult user virtual memory map for translation
  - Must lock pages in physical memory during I/O
I/O Protection

• 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