I/O Systems (1): General I/O

CSE 2431: Introduction to Operating Systems
Reading: Ch. 12, [OSC]
Outline

• Overview
• I/O Data Transfer
• Memory-Mapped I/O
• I/O Software Layers
I/O Overview

• I/O Software
  – Interrupt Handlers, Device Driver, Device-Independent Software, User-Space I/O Software

• Basic I/O hardware
  – Ports, buses, devices and controllers

• Real I/O devices
  – Disks, Character-Oriented Terminals, Graphical User Interfaces, Network Terminals
Device-Computer/Device-Device Communication

• *Physically:* Via signals over cable or through air

• *Logically:* Via a connection point: port (e.g., serial port)

• Multiple devices connected via a *bus*
  – A common set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires
## Typical Data Transfer Rates

<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>Telephone channel</td>
<td>8 KB/sec</td>
</tr>
<tr>
<td>Dual ISDN lines</td>
<td>16 KB/sec</td>
</tr>
<tr>
<td>Laser printer</td>
<td>100 KB/sec</td>
</tr>
<tr>
<td>Scanner</td>
<td>400 KB/sec</td>
</tr>
<tr>
<td>Classic Ethernet</td>
<td>1.25 MB/sec</td>
</tr>
<tr>
<td>USB (Universal Serial Bus)</td>
<td>1.5 MB/sec</td>
</tr>
<tr>
<td>Digital camcorder</td>
<td>4 MB/sec</td>
</tr>
<tr>
<td>IDE disk</td>
<td>5 MB/sec</td>
</tr>
<tr>
<td>40x CD-ROM</td>
<td>6 MB/sec</td>
</tr>
<tr>
<td>Fast Ethernet</td>
<td>12.5 MB/sec</td>
</tr>
<tr>
<td>ISA bus</td>
<td>16.7 MB/sec</td>
</tr>
<tr>
<td>EIDE (ATA-2) disk</td>
<td>16.7 MB/sec</td>
</tr>
<tr>
<td>FireWire (IEEE 1394)</td>
<td>50 MB/sec</td>
</tr>
<tr>
<td>XGA Monitor</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>Ultrium tape</td>
<td>320 MB/sec</td>
</tr>
<tr>
<td>PCI bus</td>
<td>528 MB/sec</td>
</tr>
<tr>
<td>Sun Gigaplane XB backplane</td>
<td>20 GB/sec</td>
</tr>
</tbody>
</table>
Device Controller (1)

• I/O units typically consist of
  – A mechanical component: the device itself
  – An electronic component: called the device controller or adapter.

• Interface between controller and device: very low-level

• Example:
  – Disk controller converts serial bit stream, coming off drive, into a block of bytes, and performs error correction. The block of bytes is first assembled in a buffer inside the controller. After its checksum has been verified, the error-free block is copied to main memory.
  – Built-in controllers
Device Controller (2)

- Example: Disk controller
  - implements the disk side of the protocol that does: bad error mapping, prefetching, buffering, caching
- Controller has registers for data and control
- CPU and controllers communicate via I/O instructions and registers

- What is the difference between
  - Disk controller
  - Disk driver?
Outline

• Overview
• I/O Data Transfer
• Memory-Mapped I/O
• I/O Software Layers
How to Perform I/O Operations?

• Programmed I/O
• Interrupt-driven I/O
• I/O using DMA
Programmed I/O (1)

• Every I/O operation is programmed and controlled.
• Example: Printing file from user program to printer means that data is first copied to the kernel, then OS enters a tight loop outputting the characters one at a time.
• Essential aspect: CPU continuously polls device to see if it is ready to accept another character. It uses polling (or busy waiting).
• Pros & Cons?
  – Pros: simple
  – Cons: requires CPU full time until all the I/O is done.
Programmed I/O (2)

Steps in printing a string
Programmed I/O (3)

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 */

Writing a string to the printer using programmed I/O
Interrupts Revisited

How interrupts happen; connections between devices and interrupt controller actually use interrupt lines on the bus rather than dedicated wires.
Interrupt-Driven I/O (1)

• CPU hardware has the interrupt report line that the CPU senses after executing every instruction.
  – Device raises an interrupt
  – CPU catches the interrupt and saves the state (e.g., instruction pointer)
  – CPU dispatches the interrupt handler
  – Interrupt handler determines the cause of the interrupt, services the device and clears the interrupt.
**Interrupt-Driven I/O (2)**

- Writing string to printer using interrupt-driven I/O
  - Code executed when print system call is made
  - Interrupt service procedure

```c
copy_from_user(buffer, p, count);
enable_interrupts();
while (*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler();

if (count == 0) {
    unblock_user();
} else {
    *printer_data_register = p[i];
    count = count - 1;
    i = i + 1;
}
acknowledge_interrupt();
return_from_interrupt();
```

(a) (b)
Direct Memory Access (DMA) (1)

• Use a special purpose processor: DMA controller
• Host writes a DMA command block into memory
  – Pointer to source, pointer to destination, count of bytes to be transferred
• CPU writes the address of this command block to the DMA controller and goes on with other work
• DMA controller proceeds to operate the memory bus directly without help of main CPU
• Pros and Cons?
Direct Memory Access (DMA) (2)

DMA controller feeds characters to printer one at the time, without bothering CPU. DMA is actually programmed I/O, only with DMA controller doing the work.
I/O Using DMA

- Printing a string using DMA
  - Code executed when `print()` system call is made
  - Interrupt service procedure

```c
copy_from_user(buffer, p, count);
set_up_DMA_controller();
scheduler();

acknowledge_interrupt();
unblock_user();
return_from_interrupt();
```

(a)                                                                                     (b)
DMA Issues

• Handshaking between DMA controller and the device controller

• *Cycle stealing:* DMA controller takes away CPU cycles when it uses CPU memory bus, hence blocks CPU from accessing memory

• In general: DMA controller improves overall system performance
Tradeoffs

• Programmed I/O
  – Pro: require no interrupt or DMA support
  – Con: waste CPU time

• Interrupt-drive IO:
  – Pro: save CPU time for busy polling
  – Con: triggering interrupt takes time, too

• I/O using DMA:
  – Pro: relieve CPU from I/O operation
  – Con: DMA is slower than CPU
Outline

• Overview
• I/O Data Transfer
• Memory-Mapped I/O
• I/O Software Layers
Memory-Mapped I/O (1)

(a) Separate I/O and memory space
(b) Memory-mapped I/O
(c) Hybrid

Advantages of memory-mapped I/O?
Memory-Mapped vs. Port I/O

• Memory-Mapped:
  – Pros:
    • No need for assembly languages to access I/O
    • No special protection mechanisms
    • Every instruction is able to access I/O
  – Cons:
    • Need to selectively disable caching
    • Both memory modules and all I/O devices need to check on memory bus
      – Harder with multiple buses (See next slide)
Different Bus Architectures

(a) A single-bus architecture
(b) A dual-bus memory architecture
Outline

• Overview
• I/O Data Transfer
• Memory-Mapped I/O
• I/O Software Layers
I/O Software Layers

Layers of the I/O Software System
Interrupt Handlers (1)

• Interrupt handlers are best hidden
  – Have driver starting an I/O operation block until interrupt notifies of completion

• Interrupt procedure does its task
  – Then unblocks driver that started it
Interrupt Handlers (2)

Steps must be performed in software after hardware interrupt completed:

1. Save regs not already saved by interrupt hardware
2. Set up context for interrupt service procedure
3. Set up stack for interrupt service procedure
4. Ack interrupt controller, reenable interrupts
5. Copy registers from where saved (to process table)
6. Run service procedure
7. Set up MMU context for process to run next
8. Load new process’s registers
9. Start running the new process
Device Drivers (1)

- Logical position of device drivers is shown here
- Communications between drivers and device controllers goes over the bus
Device Drivers (2)

- Device-specific code controls I/O device (manufacturer-written)
  - Each controller has device registers used to give it commands. Number of registers, commands’ nature vary (e.g., mouse driver takes info about movement, disk driver knows about sectors, tracks, heads, etc.)

- A device driver is usually part of the OS kernel
  - Compiled with the OS
  - Dynamically loaded into the OS during execution

- Each device driver handles
  - One device type (e.g., mouse)
  - One class of closely related devices (e.g., SCSI disk driver handles multiple disks of different sizes and speeds).

- Categories:
  - Block devices
  - Character devices
Functions in Device Drivers

- Accept abstract read and write requests from the device-independent layer above;
- Initialize the device;
- Manage power requirements and log events;
- Check whether input parameters are valid;
- Translate valid input from abstract to concrete terms – e.g., convert linear block number into the head, track, sector, and cylinder number for disk access;
- Check the device if it is in use;
- Control the device by issuing a sequence of commands; driver determines what commands are issued.
Device-Independent I/O Software (1)

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicate devices
Providing a device-independent block size

Functions of device-independent I/O software
Device-Independent I/O Software (2)

(a) Without a standard driver interface
(b) With a standard driver interface
Different Buffering Schemes

(a) Unbuffered input
(b) Buffering in user space
(c) Buffering in the kernel followed by copying to user space
(d) Double buffering in the kernel
Downside of Buffering

Networking may involve many copies (bufferbloat)
User-Space I/O Software

Layers of the I/O system and the main functions of each layer

- User processes: Make I/O call; format I/O; spooling
- Device-independent software: Naming, protection, blocking, buffering, allocation
- Device drivers: Set up device registers; check status
- Interrupt handlers: Wake up driver when I/O completed
- Hardware: Perform I/O operation
Summary

• Overview
• I/O Data Transfer
• Memory-Mapped I/O
• I/O Software Layers