Chapter 5: I/O Systems
Input/Output

- Principles of I/O hardware
- Principles of I/O software
- I/O software layers
- Disks
- Clocks
- Character-oriented terminals
- Graphical user interfaces
- Network terminals
- Power management
How fast is I/O hardware?

<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>Printer / scanner</td>
<td>200 KB/sec</td>
</tr>
<tr>
<td>USB</td>
<td>1.5 MB/sec</td>
</tr>
<tr>
<td>Digital camcorder</td>
<td>4 MB/sec</td>
</tr>
<tr>
<td>Fast Ethernet</td>
<td>12.5 MB/sec</td>
</tr>
<tr>
<td>Hard drive</td>
<td>20 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>PCI bus</td>
<td>500 MB/sec</td>
</tr>
</tbody>
</table>
Device controllers

- I/O devices have components
  - Mechanical component
  - Electronic component

- Electronic component controls the device
  - May be able to handle multiple devices
  - May be more than one controller per mechanical component (example: hard drive)

- Controller's tasks
  - Convert serial bit stream to block of bytes
  - Perform error correction as necessary
  - Make available to main memory
Memory-Mapped I/O

Separate I/O & memory space

Memory-mapped I/O

Hybrid: both memory-mapped & separate spaces
How is memory-mapped I/O done?

- **Single-bus**
  - All memory accesses go over a shared bus
  - I/O and RAM accesses compete for bandwidth

- **Dual-bus**
  - RAM access over high-speed bus
  - I/O access over lower-speed bus
  - Less competition
  - More hardware (more expensive…)

This port allows I/O devices access into memory
Direct Memory Access (DMA) operation

1. CPU programs the DMA controller
2. DMA requests transfer to memory
3. Data transferred
4. Acknowledgment

CPU
DMA controller
Disk controller
Main memory

Interrupt when done
Address
Count
Control
Buffer
Drive
Bus
Hardware’s view of interrupts

1. Device is finished

2. Controller issues interrupt

3. CPU acks interrupt

Bus

CPU

Interrupt controller

Disk

Clock

Keyboard

Printer
I/O software: goals

- **Device independence**
  - Programs can access any I/O device
  - No need to specify device in advance

- **Uniform naming**
  - Name of a file or device is a string or an integer
  - Doesn’t depend on the machine (underlying hardware)

- **Error handling**
  - Done as close to the hardware as possible
  - Isolate higher-level software

- **Synchronous vs. asynchronous transfers**
  - Blocked transfers vs. interrupt-driven

- **Buffering**
  - Data coming off a device cannot be stored in final destination

- **Sharable vs. dedicated devices**
Programmed I/O: printing a page

User

Kernel

Printed page

Printed page

Printed page

ABCD
EFGH

ABCD
EFGH

ABCD
EFGH

ABCD
EFGH

ABCD
EFGH

A

AB
Code for programmed I/O

copy_from_user (buffer, p, count); // copy into kernel buffer
for (j = 0; j < count; j++) {  // loop for each char
    while (*printer_status_reg != READY)
        ;                      // wait for printer to be ready
    *printer_data_reg = p[j]; // output a single character
}
return_to_user();
Interrupt-driven I/O

copy_from_user (buffer, p, count);
  j = 0;
  enable_interruptions();
  while (*printer_status_reg != READY);
  *printer_data_reg = p[0];
  scheduler(); // and block user
  if (count == 0) {
      unblock_user();
  } else {  
      *printer_data_reg = p[j];
      count--;
      j++;
  }
  acknowledge_interrupt();
  return_from_interrupt();

Code run by system call

Code run at interrupt time
I/O using DMA

copy_from_user(buffer, p, count);
set_up_DMA_controller();
scheduler(); // and block user

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

Code run by system call

Code run at interrupt time
Layers of I/O software

User-level I/O software & libraries

Device-independent OS software

Device drivers

Interrupt handlers

Hardware

User

Operating system (kernel)
Interrupt handlers

- Interrupt handlers are best hidden
  - Driver starts an I/O operation and blocks
  - Interrupt notifies of completion

- Interrupt procedure does its task
  - Then unblocks driver that started it
  - Perform minimal actions at interrupt time
    - Some of the functionality can be done by the driver after it is unblocked

- Interrupt handler must
  - Save regs not already saved by interrupt hardware
  - Set up context for interrupt service procedure
  - DLXOS: intrhandler (in dlxos.s)
What happens on an interrupt

- Set up stack for interrupt service procedure
- Ack interrupt controller, reenable interrupts
- Copy registers from where saved
- Run service procedure
- (optional) Pick a new process to run next
- Set up MMU context for process to run next
- Load new process' registers
- Start running the new process
Device drivers

- Device drivers go between device controllers and rest of OS
  - Drivers standardize interface to widely varied devices

- Device drivers communicate with controllers over bus
  - Controllers communicate with devices themselves
Device-independent I/O software

- Device-independent I/O software provides common “library” routines for I/O software
- Helps drivers maintain a standard appearance to the rest of the OS
- Uniform interface for many device drivers for
  - Buffering
  - Error reporting
  - Allocating and releasing dedicated devices
  - Suspending and resuming processes
- Common resource pool
  - Device-independent block size (keep track of blocks)
  - Other device driver resources
Why a standard driver interface?

(a) Non-standard driver interfaces

(b) Standard driver interfaces
Buffering device input

Unbuffered input

Buffering in user space

Buffer in kernel Copy to user space

Double buffer in kernel
Anatomy of an I/O request

I/O request

Layer

User processes

Device-independent software

Device drivers

Interrupt handlers

Hardware

I/O reply

I/O functions

Make I/O call; format I/O; spooling

Naming, protection, blocking, buffering, allocation

Set up device registers; check status

Wake up driver when I/O completed

Perform I/O operation
Disk drive structure

- **Data stored on surfaces**
  - Up to two surfaces per platter
  - One or more platters per disk

- **Data in concentric tracks**
  - Tracks broken into sectors
    - 256B-1KB per sector
  - Cylinder: corresponding tracks on all surfaces

- **Data read and written by heads**
  - Actuator moves heads
  - Heads move in unison
## Disk drive specifics

<table>
<thead>
<tr>
<th></th>
<th>IBM 360KB floppy</th>
<th>WD 18GB HD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cylinders</td>
<td>40</td>
<td>10601</td>
</tr>
<tr>
<td>Tracks per cylinder</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>Sectors per track</td>
<td>9</td>
<td>281 (average)</td>
</tr>
<tr>
<td>Sectors per disk</td>
<td>720</td>
<td>35742000</td>
</tr>
<tr>
<td>Bytes per sector</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>Capacity</td>
<td>360 KB</td>
<td>18.3 GB</td>
</tr>
<tr>
<td>Seek time (minimum)</td>
<td>6 ms</td>
<td>0.8 ms</td>
</tr>
<tr>
<td>Seek time (average)</td>
<td>77 ms</td>
<td>6.9 ms</td>
</tr>
<tr>
<td>Rotation time</td>
<td>200 ms</td>
<td>8.33 ms</td>
</tr>
<tr>
<td>Spinup time</td>
<td>250 ms</td>
<td>20 sec</td>
</tr>
<tr>
<td>Sector transfer time</td>
<td>22 ms</td>
<td>17 μsec</td>
</tr>
</tbody>
</table>
Disk “zones”

- Outside tracks are longer than inside tracks
- Two options
  - Bits are “bigger”
  - More bits (transfer faster)
- Modern hard drives use second option
  - More data on outer tracks
- Disk divided into “zones”
  - Constant sectors per track in each zone
  - 8–20 (or more) zones on a disk
Millions of sectors on the disk must be labeled

Two possibilities
  - Cylinder/track/sector
  - Sequential numbering

Modern drives use sequential numbers
  - Disks map sequential numbers into specific location
  - Mapping may be modified by the disk
    - Remap bad sectors
    - Optimize performance
  - Hide the exact geometry, making life simpler for the OS
Building a better “disk”

- Problem: CPU performance has been increasing exponentially, but disk performance hasn’t
  - Disks are limited by mechanics
- Problem: disks aren’t all that reliable
- Solution: distribute data across disks, and use some of the space to improve reliability
  - Data transferred in parallel
  - Data stored across drives (*striping*)
  - Parity on an “extra” drive for reliability
RAIDs, RAIDs, and more RAIDs

RAID 0
(Redundant Array of Inexpensive Disks)

RAID 1
(Mirrored copies)

RAID 4
(Striped with parity)

RAID 5
(Parity rotates through disks)
CD-ROM recording

- CD-ROM has data in a spiral
  - Hard drives have concentric circles of data
- One continuous track: just like vinyl records!
- Pits & lands “simulated” with heat-sensitive material on CD-Rs and CD-RWs
Structure of a disk sector

- Preamble contains information about the sector
  - Sector number & location information
- Data is usually 256, 512, or 1024 bytes
- ECC (Error Correcting Code) is used to detect & correct minor errors in the data
Sector layout on disk

- Sectors numbered sequentially on each track
- Numbering starts in different place on each track: *sector skew*
  - Allows time for switching head from track to track
- All done to minimize delay in sequential transfers
Sector interleaving

- On older systems, the CPU was slow \( \Rightarrow \) time elapsed between reading consecutive sectors
- Solution: leave space between consecutively numbered sectors
- This isn’t done much these days…

![Interleaving Diagrams](image)

No interleaving  Skipping 1 sector  Skipping 2 sectors
What’s in a disk request?

- Time required to read or write a disk block determined by 3 factors
  - Seek time
  - Rotational delay
    - Average delay = 1/2 rotation time
    - Example: rotate in 10ms, average rotation delay = 5ms
  - Actual transfer time
    - Transfer time = time to rotate over sector
    - Example: rotate in 10ms, 200 sectors/track => 10/200 ms = 0.05ms transfer time per sector

- Seek time dominates, with rotation time close
- Error checking is done by controllers
Disk request scheduling

- **Goal:** use disk hardware efficiently
  - Bandwidth as high as possible
  - Disk transferring as often as possible (and not seeking)

- **We want to**
  - Minimize disk seek time (moving from track to track)
  - Minimize rotational latency (waiting for disk to rotate the desired sector under the read/write head)

- **Calculate disk bandwidth by**
  - Total bytes transferred / time to service request
  - Seek time & rotational latency are overhead (no data is transferred), and reduce disk bandwidth

- **Minimize seek time & rotational latency by**
  - Using algorithms to find a good sequence for servicing requests
  - Placing blocks of a given file “near” each other
Disk scheduling algorithms

- Schedule disk requests to minimize disk seek time
  - Seek time increases as distance increases (though not linearly)
  - Minimize seek distance -> minimize seek time
- Disk seek algorithm examples assume a request queue & head position (disk has 200 cylinders)
  - Queue = 100, 175, 51, 133, 8, 140, 73, 77
  - Head position = 63

![Disk scheduling algorithm example](image)
First-Come-First Served (FCFS)

- Requests serviced in the order in which they arrived
  - Easy to implement!
  - May involve lots of unnecessary seek distance
- Seek order = 100, 175, 51, 133, 8, 140, 73, 77
- Seek distance = \((100-63) + (175-100) + (175-51) + (133-51) + (133-8) + (140-8) + (140-73) + (77-73)\) = 646 cylinders
Shortest Seek Time First (SSTF)

- Service the request with the shortest seek time from the current head position
  - Form of SJF scheduling
  - May starve some requests
- Seek order = 73, 77, 51, 8, 100, 133, 140, 175
- Seek distance = 10 + 4 + 26 + 43 + 92 + 33 + 7 + 35 = 250 cylinders
SCAN (elevator algorithm)

- Disk arm starts at one end of the disk and moves towards the other end, servicing requests as it goes
  - Reverses direction when it gets to end of the disk
  - Also known as elevator algorithm
- Seek order = 51, 8, 0, 73, 77, 100, 133, 140, 175
- Seek distance = 12 + 43 + 8 + 73 + 4 + 23 + 33 + 7 + 35 = 238 cyls
C-SCAN

- Identical to SCAN, except head returns to cylinder 0 when it reaches the end of the disk
  - Treats cylinder list as a circular list that wraps around the disk
  - Waiting time is more uniform for cylinders near the edge of the disk
- Seek order = 73, 77, 100, 133, 140, 175, 199, 0, 8, 51
- Distance = 10 + 4 + 23 + 33 + 7 + 35 + 24 + 199 + 8 + 43 = 386 cyls
C-LOOK

- Identical to C-SCAN, except head only travels as far as the last request in each direction
  - Saves seek time from last sector to end of disk
- Seek order = 73, 77, 100, 133, 140, 175, 8, 51
- Distance = $10 + 4 + 23 + 33 + 7 + 35 + 167 + 43 = 322$ cylinders
How to pick a disk scheduling algorithm

- SSTF is easy to implement and works OK if there aren’t too many disk requests in the queue
- SCAN-type algorithms perform better for systems under heavy load
  - More fair than SSTF
  - Use LOOK rather than SCAN algorithms to save time
- Long seeks aren’t too expensive, so choose C-LOOK over LOOK to make response time more even
- Disk request scheduling interacts with algorithms for allocating blocks to files
  - Make scheduling algorithm modular: allow it to be changed without changing the file system
  ⇒ Use SSTF for lightly loaded systems
  ⇒ Use C-LOOK for heavily loaded systems
When good disks go bad…

- Disks have defects
  - In 3M+ sectors, this isn’t surprising!
- ECC helps with errors, but sometimes this isn’t enough
- Disks keep spare sectors (normally unused) and remap bad sectors into these spares
  - If there’s time, the whole track could be reordered…
Clock hardware

- Crystal oscillator
  - Counter is decremented at each pulse
- Holding register is used to load the counter
Maintaining time of day

(a) 64 bits
Time of day in ticks

(b) 32 bits
Time of day in seconds
Number of ticks in current second

(c) 32 bits
Counter in ticks
System boot time in seconds
Doing multiple timers with a single clock

Clock header

Current time
4200

Next signal
3

3 → 4 → 6 → 2 → 1 X
Soft timers

- A second clock may be available for timer interrupts
  - Specified by applications
  - No problems if interrupt frequency is low
- Soft timers avoid interrupts
  - Kernel checks for soft timer expiration before it exits to user mode
  - How well this works depends on rate of kernel entries
Character-oriented terminals

- An RS-232 terminal communicates with computer 1 bit at a time
- Called a serial line – bits go out in series, 1 bit at a time
- Windows uses COM1 and COM2 ports, first to serial lines
- Computer and terminal are completely independent
Buffering for input

(a)

Terminal data structure

Terminal

0
1
2
3

Central buffer pool

(b)

Terminal data structure

Terminal

0

Buffer area for terminal 0

1

Buffer area for terminal 1
### Special terminal characters

<table>
<thead>
<tr>
<th>Character</th>
<th>POSIX name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL-H</td>
<td>ERASE</td>
<td>Backspace one character</td>
</tr>
<tr>
<td>CTRL-U</td>
<td>KILL</td>
<td>Erase entire line being typed</td>
</tr>
<tr>
<td>CTRL-V</td>
<td>LNEXT</td>
<td>Interpret next character literally</td>
</tr>
<tr>
<td>CTRL-S</td>
<td>STOP</td>
<td>Stop output</td>
</tr>
<tr>
<td>CTRL-Q</td>
<td>START</td>
<td>Start output</td>
</tr>
<tr>
<td>DEL</td>
<td>INTR</td>
<td>Interrupt process (SIGINT)</td>
</tr>
<tr>
<td>CTRL-\</td>
<td>QUIT</td>
<td>Force core dump (SIGQUIT)</td>
</tr>
<tr>
<td>CTRL-D</td>
<td>EOF</td>
<td>End of file</td>
</tr>
<tr>
<td>CTRL-M</td>
<td>CR</td>
<td>Carriage return (unchangeable)</td>
</tr>
<tr>
<td>CTRL-J</td>
<td>NL</td>
<td>Linefeed (unchangeable)</td>
</tr>
</tbody>
</table>
## Special output characters

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ESC [ nA</td>
<td>Move up $n$ lines</td>
<td></td>
</tr>
<tr>
<td>ESC [ nB</td>
<td>Move down $n$ lines</td>
<td></td>
</tr>
<tr>
<td>ESC [ nC</td>
<td>Move right $n$ spaces</td>
<td></td>
</tr>
<tr>
<td>ESC [ nD</td>
<td>Move left $n$ spaces</td>
<td></td>
</tr>
<tr>
<td>ESC [ m; nH</td>
<td>Move cursor to $(m,n)$</td>
<td></td>
</tr>
<tr>
<td>ESC [ sJ</td>
<td>Clear screen from cursor (0 to end, 1 from start, 2 all)</td>
<td></td>
</tr>
<tr>
<td>ESC [ sK</td>
<td>Clear line from cursor (0 to end, 1 from start, 2 all)</td>
<td></td>
</tr>
<tr>
<td>ESC [ nL</td>
<td>Insert $n$ lines at cursor</td>
<td></td>
</tr>
<tr>
<td>ESC [ nM</td>
<td>Delete $n$ lines at cursor</td>
<td></td>
</tr>
<tr>
<td>ESC [ nP</td>
<td>Delete $n$ chars at cursor</td>
<td></td>
</tr>
<tr>
<td>ESC [ n@</td>
<td>Insert $n$ chars at cursor</td>
<td></td>
</tr>
<tr>
<td>ESC [ nm</td>
<td>Enable rendition $n$ (0=normal, 4=bold, 5=blinking, 7=reverse)</td>
<td></td>
</tr>
<tr>
<td>ESC M</td>
<td>Scroll the screen backward if the cursor is on the top line</td>
<td></td>
</tr>
</tbody>
</table>
Memory-mapped display

Driver writes directly into display's video RAM
How characters are displayed

- A video RAM image
  - simple monochrome display
  - character mode
- Corresponding screen
  - the Xs are attribute bytes
Input software

- Keyboard driver delivers a number
  - Driver converts to characters
  - Uses a ASCII table
- Exceptions, adaptations needed for other languages
  - Many OS provide for loadable keymaps or code pages
  - Example: characters such as ç
Output software for Windows

- Sample window located at (200,100) on XGA display
Skeleton of a Windows program

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE h, HINSTANCE hprev, char *szCmd, int iCmdShow)
{
    WNDCLASS wndclass;  /* class object for this window */
    MSG msg;           /* incoming messages are stored here */
    HWND hwnd;         /* handle (pointer) to the window object */

    /* Initialize wndclass */
    wndclass.lpfnWndProc = WndProc;    /* tells which procedure to call */
    wndclass.lpszClassName = "Program name"; /* Text for title bar */
    wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION); /* load program icon */
    wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);  /* load mouse cursor */

    RegisterClass(&wndclass); /* tell Windows about wndclass */
    hwnd = CreateWindow ( ... ) /* allocate storage for the window */
    ShowWindow(hwnd, iCmdShow); /* display the window on the screen */
    UpdateWindow(hwnd); /* tell the window to paint itself */
```
Skeleton of a Windows program (cont’d)

```c
while (GetMessage(&msg, NULL, 0, 0)) {  /* get message from queue */
    TranslateMessage(&msg);   /* translate the message */
    DispatchMessage(&msg);   /* send msg to the appropriate procedure */
}
return(msg.wParam);
}

long CALLBACK WndProc(HWND hwnd, UINT message, UINT wParam, long lParam)
{
    /* Declarations go here. */

    switch (message) {
        case WM_CREATE:  ... ;  return ... ;   /* create window */
        case WM_PAINT:    ... ;  return ... ;   /* repaint contents of window */
        case WM_DESTROY:  ... ;  return ... ;   /* destroy window */
    }
    return(DefWindowProc(hwnd, message, wParam, lParam));  /* default */
}
```
Character outlines at different point sizes

20 pt:  abcdefgh

53 pt:  abcdefgh

81 pt:  abcdefgh
X Windows

Remote host

<table>
<thead>
<tr>
<th>Window manager</th>
<th>Application program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motif</td>
<td></td>
</tr>
<tr>
<td>Intrinsics</td>
<td></td>
</tr>
<tr>
<td>Xlib</td>
<td></td>
</tr>
<tr>
<td>X client</td>
<td></td>
</tr>
<tr>
<td>UNIX</td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td></td>
</tr>
</tbody>
</table>

User space

Kernel space

X terminal

X server

UNIX

Hardware

Network

X protocol
Architecture of the SLIM terminal system
## The SLIM Network Terminal

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET</td>
<td>Update a rectangle with new pixels</td>
</tr>
<tr>
<td>FILL</td>
<td>Fill a rectangle with one pixel value</td>
</tr>
<tr>
<td>BITMAP</td>
<td>Expand a bitmap to fill a rectangle</td>
</tr>
<tr>
<td>COPY</td>
<td>Copy a rectangle from one part of the frame buffer to another</td>
</tr>
<tr>
<td>CSCS</td>
<td>Convert a rectangle from television color (YUV) to RGB</td>
</tr>
</tbody>
</table>
### Power Management (1)

<table>
<thead>
<tr>
<th>Device</th>
<th>Li et al. (1994)</th>
<th>Lorch and Smith (1998)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display</td>
<td>68%</td>
<td>39%</td>
</tr>
<tr>
<td>CPU</td>
<td>12%</td>
<td>18%</td>
</tr>
<tr>
<td>Hard disk</td>
<td>20%</td>
<td>12%</td>
</tr>
<tr>
<td>Modem</td>
<td></td>
<td>6%</td>
</tr>
<tr>
<td>Sound</td>
<td></td>
<td>2%</td>
</tr>
<tr>
<td>Memory</td>
<td>0.5%</td>
<td>1%</td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td>22%</td>
</tr>
</tbody>
</table>

Power consumption of various parts of a laptop computer
Power management (2)

The use of zones for backlighting the display
Power Management (3)

- Running at full clock speed
- Cutting voltage by two
  - cuts clock speed by two,
  - cuts power by four
Power Management (4)

- Telling the programs to use less energy
  - may mean poorer user experience

- Examples
  - change from color output to black and white
  - speech recognition reduces vocabulary
  - less resolution or detail in an image