Chapter 5

Input/Output

5.1 Principles of I/O hardware
5.2 Principles of I/O software
5.3 I/O software layers
5.4 Disks
5.5 Clocks
5.6 Character-oriented terminals
5.7 Graphical user interfaces
5.8 Network terminals
5.9 Power management
## Principles of 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>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>

Some typical device, network, and data base rates
Device Controllers

- **I/O devices have components:**
  - mechanical component
  - electronic component
- **The electronic component is the device controller**
  - may be able to handle multiple devices
- **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 (1)

- Separate I/O and memory space
- Memory-mapped I/O
- Hybrid
Memory-Mapped I/O (2)

(a) A single-bus architecture
(b) A dual-bus memory architecture
Direct Memory Access (DMA)

Operation of a DMA transfer
Interrupts Revisited

How interrupts happens. Connections between devices and interrupt controller actually use interrupt lines on the bus rather than dedicated wires.
Principles of I/O Software

Goals of I/O Software (1)

• **Device independence**
  – programs can access any I/O device
  – without specifying device in advance
    · (floppy, hard drive, or CD-ROM)

• **Uniform naming**
  – name of a file or device a string or an integer
  – not depending on which machine

• **Error handling**
  – handle as close to the hardware as possible
Goals of I/O Software (2)

- **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**
  - disks are sharable
  - tape drives would not be
Programmed I/O (1)

Steps in printing a string
Programmed I/O (2)

```c
void copy_from_user(char *buffer, char *p, int 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
Interrupt-Driven I/O

copy_from_user(buffer, p, count);
enable_interruptions();
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)

• Writing a string to the printer using interrupt-driven I/O
  – Code executed when print system call is made
  – Interrupt service procedure
I/O Using DMA

- Printing a string using DMA
  - code executed when the print system call is made
  - interrupt service procedure

```c
void print_stringDMA(int buffer, int p, int count)
{
    copy_from_user(buffer, p, count);
    set_up_DMA_controller();
    scheduler();
    acknowledge_interrupt();
    unblock_user();
    return_from_interrupt();
}
```
## I/O Software Layers

<table>
<thead>
<tr>
<th>Layers of the I/O Software System</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User-level I/O software</strong></td>
</tr>
<tr>
<td><strong>Device-independent operating system software</strong></td>
</tr>
<tr>
<td><strong>Device drivers</strong></td>
</tr>
<tr>
<td><strong>Interrupt handlers</strong></td>
</tr>
<tr>
<td><strong>Hardware</strong></td>
</tr>
</tbody>
</table>
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

• **Steps must be performed in software after interrupt completed**
  1. Save regs not already saved by interrupt hardware
  2. Set up context for interrupt service procedure
Interrupt Handlers (2)

3. Set up stack for interrupt service procedure
4. Ack interrupt controller, reenable interrupts
5. Copy registers from where saved
6. Run service procedure
7. Set up MMU context for process to run next
8. Load new process' registers
9. Start running the new process
Device Drivers

- Logical position of device drivers is shown here
- Communications between drivers and device controllers goes over the bus
## Device-Independent I/O Software (1)

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform interfacing for device drivers</td>
</tr>
<tr>
<td>Buffering</td>
</tr>
<tr>
<td>Error reporting</td>
</tr>
<tr>
<td>Allocating and releasing dedicate devices</td>
</tr>
<tr>
<td>Providing a deice-independent block size</td>
</tr>
</tbody>
</table>

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

(a) Without a standard driver interface
(b) With a standard driver interface
Device-Independent I/O Software (3)

(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
Device-Independent I/O Software (4)

Networking may involve many copies
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
# Disks

## Disk Hardware (1)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IBM 360-KB floppy disk</th>
<th>WD 18300 hard disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of 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 (avg)</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>Disk capacity</td>
<td>360 KB</td>
<td>18.3 GB</td>
</tr>
<tr>
<td>Seek time (adjacent cylinders)</td>
<td>6 msec</td>
<td>0.8 msec</td>
</tr>
<tr>
<td>Seek time (average case)</td>
<td>77 msec</td>
<td>6.9 msec</td>
</tr>
<tr>
<td>Rotation time</td>
<td>200 msec</td>
<td>8.33 msec</td>
</tr>
<tr>
<td>Motor stop/start time</td>
<td>250 msec</td>
<td>20 sec</td>
</tr>
<tr>
<td>Time to transfer 1 sector</td>
<td>22 msec</td>
<td>17 μsec</td>
</tr>
</tbody>
</table>

Disk parameters for the original IBM PC floppy disk and a Western Digital WD 18300 hard disk
Disk Hardware (2)

- Physical geometry of a disk with two zones
- A possible virtual geometry for this disk
Disk Hardware (3)

- Raid levels 0 through 2
- Backup and parity drives are shaded
Disk Hardware (4)

- Raid levels 3 through 5
- Backup and parity drives are shaded
Disk Hardware (5)

Recording structure of a CD or CD-ROM
Logical data layout on a CD-ROM

Symbols of 14 bits each

42 Symbols make 1 frame

98 Frames make 1 sector

Frames of 588 bits, each containing 24 data bytes

Mode 1 sector (2352 bytes)
Disk Hardware (7)

- Cross section of a CD-R disk and laser
  - not to scale
- Silver CD-ROM has similar structure
  - without dye layer
  - with pitted aluminum layer instead of gold

CuuDuongThanCong.com
https://fb.com/tailieudientucntt
Disk Hardware (8)

A double sided, dual layer DVD disk
Disk Formatting (1)

A disk sector

Preamble | Data | ECC
An illustration of cylinder skew
Disk Formatting (3)

- No interleaving
- Single interleaving
- Double interleaving
Disk Arm Scheduling Algorithms (1)

- Time required to read or write a disk block determined by 3 factors
  1. Seek time
  2. Rotational delay
  3. Actual transfer time
- Seek time dominates
- Error checking is done by controllers
Disk Arm Scheduling Algorithms (2)

Shortest Seek First (SSF) disk scheduling algorithm
Disk Arm Scheduling Algorithms (3)

The elevator algorithm for scheduling disk requests

![Diagram of elevator algorithm for disk arm scheduling](image-url)
Error Handling

- A disk track with a bad sector
- Substituting a spare for the bad sector
- Shifting all the sectors to bypass the bad one
Stable Storage

Analysis of the influence of crashes on stable writes
Clocks
Clock Hardware

Crystal oscillator

Counter is decremented at each pulse

Holding register is used to load the counter

A programmable clock
Clock Software (1)

Three ways to maintain the 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
Clock Software (2)

Simulating multiple timers with a single clock
Soft Timers

• A second clock 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
RS-232 Terminal Hardware

- 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
• **Central buffer pool**
• **Dedicated buffer for each terminal**
## Input Software (2)

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

Characters handled specially in canonical mode
Output Software

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

The ANSI escape sequences
- accepted by terminal driver on output
- ESC is ASCII character (0x1B)
- n,m, and s are optional numeric parameters
Display Hardware (1)

Memory-mapped displays
• driver writes directly into display's video RAM
Display Hardware (2)

- 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
Output Software for Windows (1)

Sample window located at (200,100) on XGA display
Output Software for Windows (2)

#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.hlcon = 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 */
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 */
}
Output Software for Windows (4)

An example rectangle drawn using *Rectangle*
Output Software for Windows (5)

- Copying bitmaps using *BitBlt*.
  - before
  - after
Output Software for Windows (6)

Examples of character outlines at different point sizes

20 pt:  \( \text{abcdefgh} \)

53 pt:  \( \text{abcdefgh} \)

81 pt:  \( \text{abcdefgh} \)
Network Terminals
X Windows (1)

Clients and servers in the M.I.T. X Window System
X Windows (2)

#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(int argc, char *argv[])
{
    Display disp;        /* server identifier */
    Window win;          /* window identifier */
    GC gc;               /* graphic context identifier */
    XEvent event;        /* storage for one event */
    int running = 1;

    disp = XOpenDisplay("display_name"); /* connect to the X server */
    win = XCreateSimpleWindow(disp, ... ); /* allocate memory for new window */
    XSetStandardProperties(disp,...);     /* announces window to window mgr */
    gc = XCreateGC(disp, win, 0, 0);       /* create graphic context */
    XSelectInput(disp, win, ButtonPressMask | KeyPressMask | ExposureMask);
    XMapRaised(disp, win);                 /* display window; send Expose event */

    while (running) {
        XNextEvent(disp, &event);          /* get next event */
        switch (event.type) {
            case Expose: ...; break;       /* repaint window */
            case ButtonPress: ...; break;   /* process mouse click */
            case Keypress: ...; break;      /* process keyboard input */
        }
    }

    XFreeGC(disp, gc);                    /* release graphic context */
    XDestroyWindow(disp, win);            /* deallocate window’s memory space */
    XCloseDisplay(disp);                  /* tear down network connection */
}
The SLIM Network Terminal (1)

The architecture of the SLIM terminal system
The SLIM Network Terminal (2)

<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>

Messages used in the SLIM protocol from the server to the terminals
### 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