Chapter 5
Input/Output
I/O Devices

Figure 5-1. Some typical device, network, and bus data 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>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>
Memory-Mapped I/O (1)

Figure 5-2. (a) Separate I/O and memory space. (b) Memory-mapped I/O. (c) Hybrid.
Memory-Mapped I/O (2)

Figure 5-3. (a) A single-bus architecture. (b) A dual-bus memory architecture.
Direct Memory Access (DMA)

Figure 5-4. Operation of a DMA transfer.

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

CPU → DMA controller: Address, Count, Control

Buffer

Drive

Main memory

Interrupt when done

Bus
Interrupts Revisited

Figure 5-5. How an interrupt happens. The connections between the devices and the interrupt controller actually use interrupt lines on the bus rather than dedicated wires.
Precise and Imprecise Interrupts (1)

Properties of a precise interrupt

1. PC (Program Counter) is saved in a known place.
2. All instructions before the one pointed to by the PC have fully executed.
3. No instruction beyond the one pointed to by the PC has been executed.
4. Execution state of the instruction pointed to by the PC is known.
Figure 5-6. (a) A precise interrupt. (b) An imprecise interrupt.
Figure 5-7. Steps in printing a string.
Figure 5-8. 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); /* loop until ready */
    *printer_data_register = p[i]; /* output one character */
}
return_to_user();

Programmed I/O (2)
Figure 5-9. Writing a string to the printer using interrupt-driven I/O.
(a) Code executed at the time the print system call is made.
(b) Interrupt service procedure for the printer.
I/O Using DMA

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

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

(b)

Figure 5-10. Printing a string using DMA. (a) Code executed when the print system call is made. (b) Interrupt service procedure.
I/O Software Layers

Figure 5-11. Layers of the I/O software system.
Interrupt Handlers (1)

1. Save registers not already been saved by interrupt hardware.
2. Set up a context for the interrupt service procedure.
3. Set up a stack for the interrupt service procedure.
4. Acknowledge the interrupt controller. If there is no centralized interrupt controller, reenable interrupts.
5. Copy the registers from where they were saved to the process table.
Interrupt Handlers (2)

6. Run the interrupt service procedure.
7. Choose which process to run next.
8. Set up the MMU context for the process to run next.
9. Load the new process’ registers, including its PSW.
10. Start running the new process.
Device Drivers

Figure 5-12. Logical positioning of device drivers. In reality all communication between drivers and device controllers goes over the bus.
Figure 5-13. Functions of the device-independent I/O software.

<table>
<thead>
<tr>
<th>Function Description</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 dedicated devices</td>
</tr>
<tr>
<td>Providing a device-independent block size</td>
</tr>
</tbody>
</table>
Uniform Interfacing for Device Drivers

Figure 5-14. (a) Without a standard driver interface. (b) With a standard driver interface.
Buffering (1)

Figure 5-15. (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.
Figure 5-16. Networking may involve many copies of a packet.
User-Space I/O Software

Figure 5-17. Layers of the I/O system and the main functions of each layer.

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

**Figure 5-18.** Disk parameters for the original IBM PC 360-KB floppy disk and a Western Digital WD 18300 hard disk.

Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639
Figure 5-19. (a) Physical geometry of a disk with two zones. (b) A possible virtual geometry for this disk.
Figure 5-20. RAID levels 0 through 5.
Backup and parity drives are shown shaded.

Tanenbaum, Modern Operating Systems 3 e, (c) 2008 Prentice-Hall, Inc. All rights reserved. 0-13-6006639
Figure 5-20. RAID levels 0 through 5. Backup and parity drives are shown shaded.
CD-ROMs (1)

Figure 5-21. Recording structure of a compact disc or CD-ROM.
Figure 5-22. Logical data layout on a CD-ROM.
Figure 5-23. Cross section of a CD-R disk and laser. A silver CD-ROM has similar structure, except without dye layer and with pitted aluminum layer instead of gold layer.
DVD Improvements on CDs

1. Smaller pits
   (0.4 microns versus 0.8 microns for CDs).

2. A tighter spiral
   (0.74 microns between tracks versus 1.6 microns for CDs).

3. A red laser
   (at 0.65 microns versus 0.78 microns for CDs).
DVD (2)

DVD Formats

2. Single-sided, dual-layer (8.5 GB).
3. Double-sided, single-layer (9.4 GB).
4. Double-sided, dual-layer (17 GB).
Figure 5-24. A double-sided, dual-layer DVD disk.
Disk Formatting (1)

Figure 5-25. A disk sector.
Disk Formatting (2)

Figure 5-26. An illustration of cylinder skew.
Figure 5-27. (a) No interleaving. (b) Single interleaving. (c) Double interleaving.
Disk Arm Scheduling Algorithms (1)

Read/write time factors

1. Seek time (the time to move the arm to the proper cylinder).
2. Rotational delay (the time for the proper sector to rotate under the head).
3. Actual data transfer time.
Disk Arm Scheduling Algorithms (2)

Figure 5-28. Shortest Seek First (SSF) disk scheduling algorithm.
Disk Arm Scheduling Algorithms (3)

Figure 5-29. The elevator algorithm for scheduling disk requests.
Error Handling

Figure 5-30. (a) A disk track with a bad sector. (b) Substituting a spare for the bad sector. (c) Shifting all the sectors to bypass the bad one.
Stable Storage (1)

Operations for stable storage using identical disks:

1. Stable writes
2. Stable reads
3. Crash recovery
Figure 5-31. Analysis of the influence of crashes on stable writes.
Clock Hardware

Figure 5-32. A programmable clock.
Clock Software (1)

Typical duties of a clock driver

1. Maintaining the time of day.
2. Preventing processes from running longer than they are allowed to.
3. Accounting for CPU usage.
4. Handling alarm system call made by user processes.
5. Providing watchdog timers for parts of the system itself.
Figure 5-33. Three ways to maintain the time of day.
Figure 5-34. Simulating multiple timers with a single clock.
Soft Timers

Soft timers succeed according to rate at which kernel entries are made because of:

1. System calls.
2. TLB misses.
4. I/O interrupts.
5. The CPU going idle.
<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>

Figure 5-35. Characters that are handled specially in canonical mode.
The X Window System (1)

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESC [n A</td>
<td>Move up n lines</td>
</tr>
<tr>
<td>ESC [n B</td>
<td>Move down n lines</td>
</tr>
<tr>
<td>ESC [n C</td>
<td>Move right n spaces</td>
</tr>
<tr>
<td>ESC [n D</td>
<td>Move left n spaces</td>
</tr>
<tr>
<td>ESC [m; n H</td>
<td>Move cursor to (m, n)</td>
</tr>
<tr>
<td>ESC [s J</td>
<td>Clear screen from cursor (0 to end, 1 from start, 2 all)</td>
</tr>
<tr>
<td>ESC [s K</td>
<td>Clear line from cursor (0 to end, 1 from start, 2 all)</td>
</tr>
<tr>
<td>ESC [n L</td>
<td>Insert n lines at cursor</td>
</tr>
<tr>
<td>ESC [n M</td>
<td>Delete n lines at cursor</td>
</tr>
<tr>
<td>ESC [n P</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 [n m</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>

Figure 5-36. The ANSI escape sequences accepted by the terminal driver on output. ESC denotes the ASCII escape character (0x1B), and n, m, and s are optional numeric parameters.
Figure 5-37. Clients and servers in the M.I.T. X Window System.
The X Window System (3)

Types of messages between client and server:
1. Drawing commands from the program to the workstation.
2. Replies by the workstation to program queries.
3. Keyboard, mouse, and other event announcements.
4. Error messages.
Graphical User Interfaces (1)

```c
#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 */

    ...}
```

Figure 5-38. A skeleton of an X Window application program.
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 */
}
Figure 5-39. A sample window located at (200, 100) on an XGA display.
Graphical User Interfaces (4)

```c
#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.hlIcon = 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 */

    ... 

Figure 5-40. A skeleton of a Windows main program.
```
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 */
}

Figure 5-40. A skeleton of a Windows main program.
Figure 5-41. An example rectangle drawn using Rectangle. Each box represents one pixel.
Figure 5-42. Copying bitmaps using *BitBlt*. (a) Before. (b) After.
Figure 5-43. Some examples of character outlines at different point sizes.

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

![Table of THINC protocol display commands](image)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw</td>
<td>Display raw pixel data at a given location</td>
</tr>
<tr>
<td>Copy</td>
<td>Copy frame buffer area to specified coordinates</td>
</tr>
<tr>
<td>Sfill</td>
<td>Fill an area with a given pixel color value</td>
</tr>
<tr>
<td>Pfill</td>
<td>Fill an area with a given pixel pattern</td>
</tr>
<tr>
<td>Bitmap</td>
<td>Fill a region using a bitmap image</td>
</tr>
</tbody>
</table>

**Figure 5-44.** The THINC protocol display commands.
Power Management

Hardware Issues

<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>

Figure 5-45. Power consumption of various parts of a notebook computer.
Figure 5-46. The use of zones for backlighting the display.
(a) When window 2 is selected it is not moved.
(b) When window 1 is selected, it moves to reduce the number of zones illuminated.
Power Management

The CPU

Figure 5-47. (a) Running at full clock speed. (b) Cutting voltage by two cuts clock speed by two and power consumption by four.