Input/Output

Chapter 5
I/O Devices (1)

• Block devices
  – Stores information in fixed-size blocks
  – Transfers are in units of entire blocks

• Character devices
  – Delivers or accepts stream of characters, without regard to block structure
  – Not addressable, does not have any seek operation
## I/O Devices (2)

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 at 300 dpi</td>
<td>1 MB/sec</td>
</tr>
<tr>
<td>Digital camcorder</td>
<td>3.5 MB/sec</td>
</tr>
<tr>
<td>4x Blu-ray disc</td>
<td>18 MB/sec</td>
</tr>
<tr>
<td>802.11n Wireless</td>
<td>37.5 MB/sec</td>
</tr>
<tr>
<td>USB 2.0</td>
<td>60 MB/sec</td>
</tr>
<tr>
<td>FireWire 800</td>
<td>100 MB/sec</td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>125 MB/sec</td>
</tr>
<tr>
<td>SATA 3 disk drive</td>
<td>600 MB/sec</td>
</tr>
<tr>
<td>USB 3.0</td>
<td>625 MB/sec</td>
</tr>
<tr>
<td>SCSI Ultra 5 bus</td>
<td>640 MB/sec</td>
</tr>
<tr>
<td>Single-lane PCIe 3.0 bus</td>
<td>985 MB/sec</td>
</tr>
<tr>
<td>Thunderbolt 2 bus</td>
<td>2.5 GB/sec</td>
</tr>
<tr>
<td>SONET OC-768 network</td>
<td>5 GB/sec</td>
</tr>
</tbody>
</table>

Tanenbaum & Bo, Modern Operating Systems: 4th ed., (c) 2013 Prentice-Hall, Inc. All rights reserved.
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

Figure 5-4. Operation of a DMA transfer.
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 Interrupt

Four properties of a *precise interrupt*:  
1. The PC saved in a known place.  
2. All instructions before that pointed to by PC have fully executed.  
3. No instruction beyond that pointed to by PC has been executed.  
4. Execution state of instruction pointed to by PC is known.
Precise vs. Imprecise

Figure 5-6. (a) A precise interrupt. (b) An imprecise interrupt.
Goals of the I/O Software

Issues:
• Device independence
• Uniform naming
• Error handling
• Synchronous versus asynchronous
• Buffering.
Programmed I/O (1)

Figure 5-7. Steps in printing a string.
Programmed I/O (2)

```c

void 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 */
```

Figure 5-8. Writing a string to the printer using programmed I/O.
Interrupt-Driven I/O

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

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)

Typical steps after hardware interrupt completes:

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

Typical steps after hardware interrupt completes:

6. Run interrupt service procedure. Extract information from interrupting device controller’s registers.

7. Choose which process to run next.

8. Set up the MMU context for process to run next.

9. Load 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.
Device-Independent I/O Software

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

Figure 5-13. Functions of the device-independent I/O software.
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.
Buffering (2)

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.
## Magnetic Disks (1)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IBM 360-KB floppy disk</th>
<th>WD 3000 HLFS hard disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of cylinders</td>
<td>40</td>
<td>36481</td>
</tr>
<tr>
<td>Tracks per cylinder</td>
<td>2</td>
<td>255</td>
</tr>
<tr>
<td>Sectors per track</td>
<td>9</td>
<td>63 (avg)</td>
</tr>
<tr>
<td>Sectors per disk</td>
<td>720</td>
<td>586,072,368</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>300 GB</td>
</tr>
<tr>
<td>Seek time (adjacent cylinders)</td>
<td>6 msec</td>
<td>0.7 msec</td>
</tr>
<tr>
<td>Seek time (average case)</td>
<td>77 msec</td>
<td>4.2 msec</td>
</tr>
<tr>
<td>Rotation time</td>
<td>200 msec</td>
<td>6 msec</td>
</tr>
<tr>
<td>Time to transfer 1 sector</td>
<td>22 msec</td>
<td>1.4 μsec</td>
</tr>
</tbody>
</table>

Figure 5-18. Disk parameters for the original IBM PC 360-KB floppy disk and a Western Digital WD 3000 HLFS ("Velociraptor") hard disk.
Magnetic Disks (2)

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 3. Backup and parity drives are shown shaded.
Figure 5-20. RAID levels 4 through 6. Backup and parity drives are shown shaded.
Disk Formatting (1)

Figure 5-21. A disk sector.
Disk Formatting (2)

Figure 5-22. An illustration of cylinder skew.
Disk Formatting (3)

Figure 5-23. (a) No interleaving. (b) Single interleaving. (c) Double interleaving.
Disk Arm Scheduling Algorithms (1)

Factors of a disk block read/write:
1. Seek time (the time to move the arm to the proper cylinder).
2. Rotational delay (how long for the proper sector to come under the head).
3. Actual data transfer time.
Disk Arm Scheduling Algorithms (2)

Figure 5-24. Shortest Seek First (SSF) disk scheduling algorithm.
Disk Arm Scheduling Algorithms (3)

Figure 5-25. The elevator algorithm for scheduling disk requests.
Figure 5-26. (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)

• Uses pair of identical disks
• Either can be read to get same results
• Operations defined to accomplish this:
  1. Stable Writes
  2. Stable Reads
  3. Crash recovery
Figure 5-27. Analysis of the influence of crashes on stable writes.
Clock Hardware

Figure 5-28. A programmable clock.
Typical duties of a clock driver:
1. Maintaining the time of day.
2. Preventing processes from running longer than allowed.
3. Accounting for CPU usage.
4. Handling alarm system call from user processes.
5. Providing watchdog timers for parts of system itself.
Clock Software (2)

Figure 5-29. Three ways to maintain the time of day.
Clock Software (3)

Figure 5-30. Simulating multiple timers with a single clock.
Soft Timers

Soft timers stand or fall with the rate at which kernel entries are made for other reasons. These reasons include:

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

<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 5-31. Characters that are handled specially in canonical mode.
Output Software - Text Windows

<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;m,H</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>

Figure 5-32. 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.
The X Window System (1)

Figure 5-33. Clients and servers in the M.I.T. X Window System.

Tanenbaum & Bo, Modern Operating Systems: 4th ed., (c) 2013 Prentice-Hall, Inc. All rights reserved.
The X Window System (2)

Types of messages between client and server:
1. Drawing commands from program to workstation.
2. Replies by workstation to program queries.
3. Keyboard, mouse, and other event announcements.
4. Error messages.
The X Window System (3)

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

main(int argc, char *argv[]) {
    Display disp;
    Window win;
    GC gc;
    XEvent 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) {
            ...  
            break; /* interrupt window */
        }
    }
}
```

Figure 5-34. A skeleton of an X Window application program.
The X Window System (4)

```c
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 */
```

Figure 5-34. A skeleton of an X Window application program.
Graphical User Interfaces (1)

Figure 5-35. A sample window located at (200, 100) on an XGA display.
Graphical User Interfaces (2)

```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.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 */
```

Figure 5-36. A skeleton of a Windows main program.
Graphical User Interfaces (3)

```
UpdateWindow(hwnd); /* tell the window to paint itself */

while (GetMessage(&msg, NULL, 0, 0)) {
    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 5-36. A skeleton of a Windows main program.
Figure 5-37. An example rectangle drawn using *Rectangle*. Each box represents one pixel.
Bitmaps

Figure 5-38. Copying bitmaps using BitBlt. (a) Before. (b) After.
Figure 5-39. Some examples of character outlines at different point sizes.
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 5-40. Power consumption of various parts of a notebook computer.
Figure 5-41. 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.
Operating System Issues
The CPU

Figure 5-42. (a) Running at full clock speed. (b) Cutting voltage by two cuts clock speed by two and power consumption by four
End

Chapter 5