Chapter 5

Input/Output

Principles of I/O Hardware

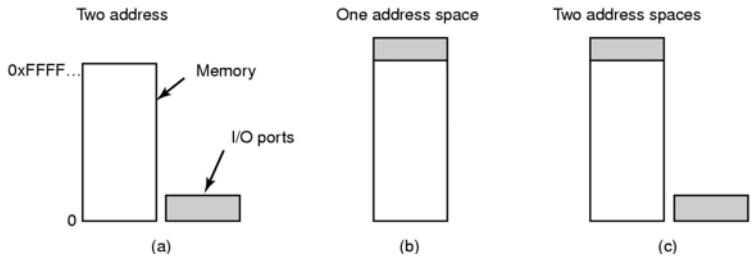
Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Telephone channel	8 KB/sec
Dual ISDN lines	16 KB/sec
Laser printer	100 KB/sec
Scanner	400 KB/sec
Classic Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Digital camcorder	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

Some typical device, network, and data base rates

Device Controllers

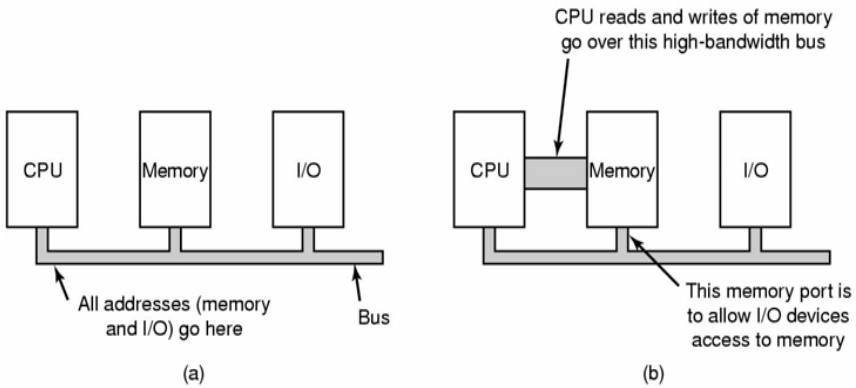
- I/O devices have components:
  - mechanical component
  - electronic component
- The electronic component is the device controller
  - may be able to handle multiple devices
- Controller's tasks
  - convert serial bit stream to block of bytes
  - perform error correction as necessary
  - make available to main memory

Memory-Mapped I/O (1)



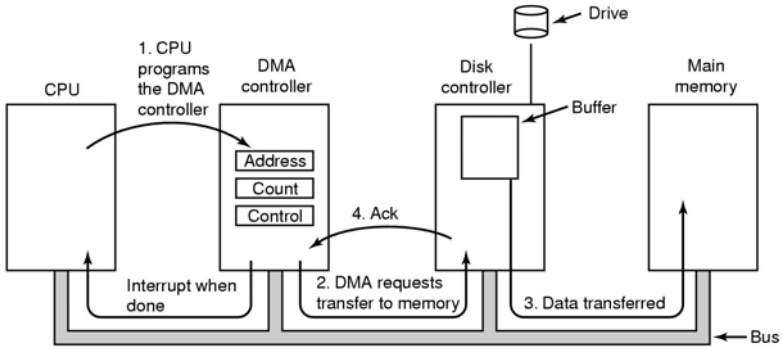
- Separate I/O and memory space
- Memory-mapped I/O
- Hybrid

Memory-Mapped I/O (2)



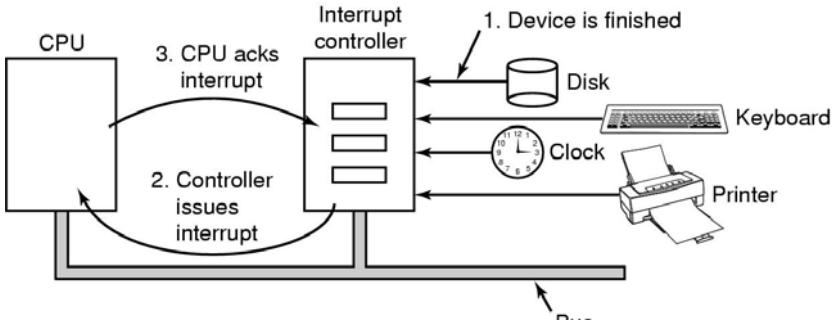
(a) A single-bus architecture
(b) A dual-bus memory architecture

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

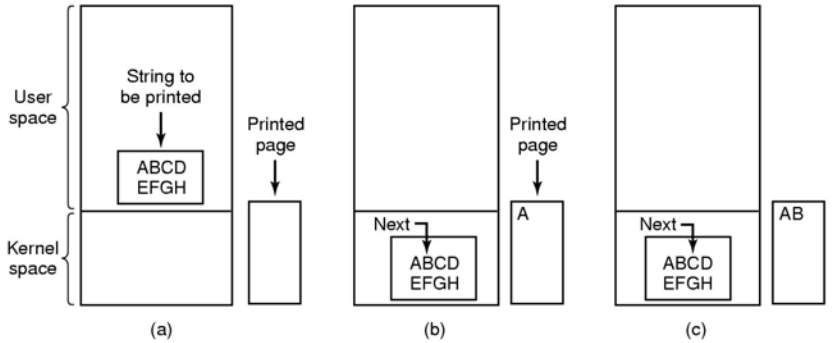
Principles of I/O Software
Goals of I/O Software (1)

- 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)

```
copy_from_user(buffer, p, count);          /* p is the kernel bufer */
for (i = 0; i < count; i++) {              /* loop on every character */
      while (*printer_status_reg != READY) ;   /* loop until ready */
      *printer_data_register = p[i];           /* output one character */
}
return_to_user( );
```

Writing a string to the printer using programmed I/O

Interrupt-Driven I/O

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

(a)

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

(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

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

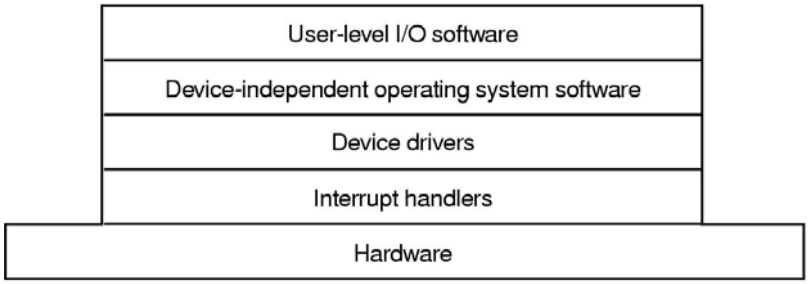
(a)

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

(b)

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

I/O Software Layers



Layers of the I/O Software System

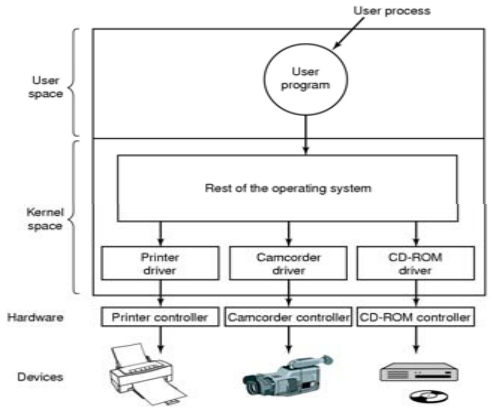
Interrupt Handlers (1)

- 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
- 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



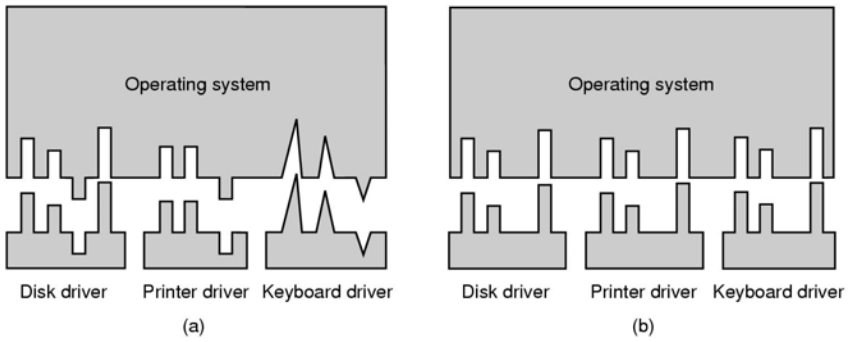
- Logical position of device drivers is shown here
- Communications between drivers and device controllers goes over the bus

Device-Independent I/O Software (1)

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

Functions of the device-independent I/O software

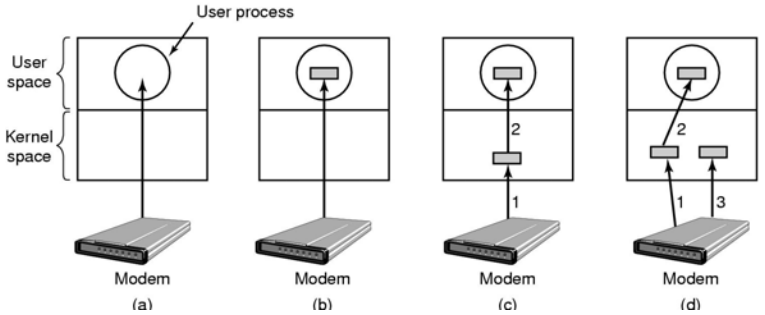
Device-Independent I/O Software (2)



(a) Without a standard driver interface

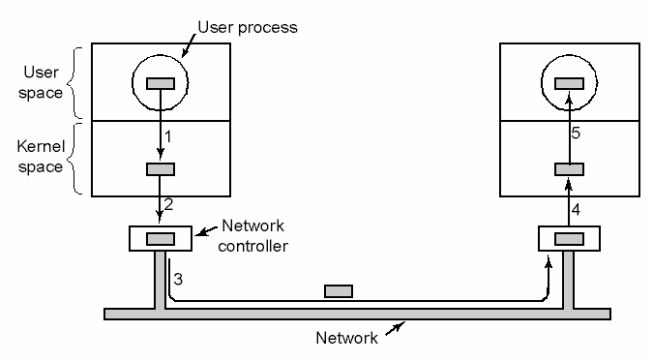
(b) With a standard driver interface

Device-Independent I/O Software (3)



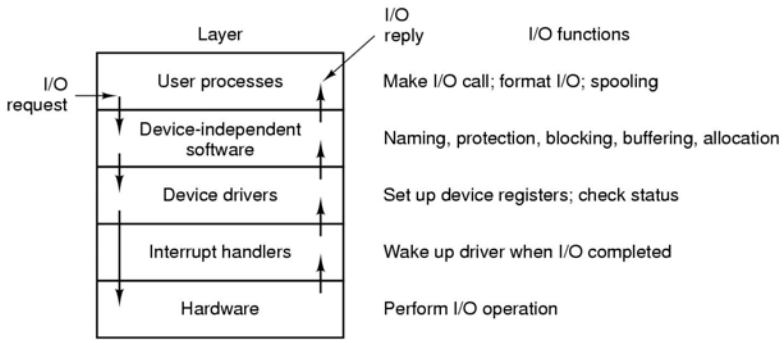
(a) Unbuffered input
(b) Buffering in user space
(c) Buffering in the kernel followed by copying to user space
(d) Double buffering in the kernel

Device-Independent I/O Software (4)



Networking may involve many copies

User-Space I/O Software



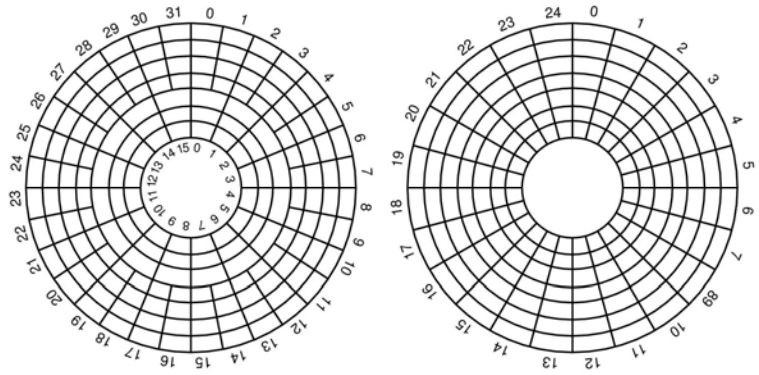
Layers of the I/O system and the main functions of each layer

Disks
Disk Hardware (1)

Parameter	IBM 360-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10601
Tracks per cylinder	2	12
Sectors per track	9	281 (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	360 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.8 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	8.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 μsec

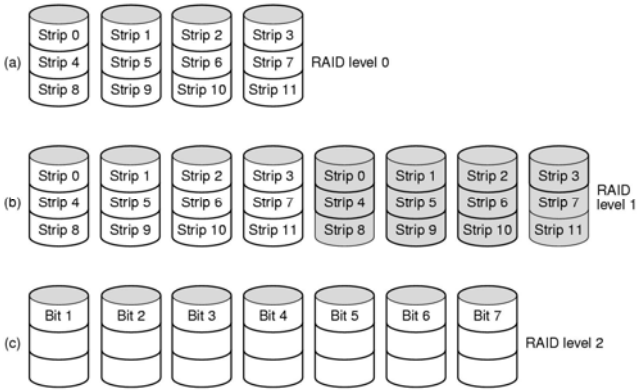
Disk parameters for the original IBM PC floppy disk and a Western Digital WD 18300 hard disk

Disk Hardware (2)



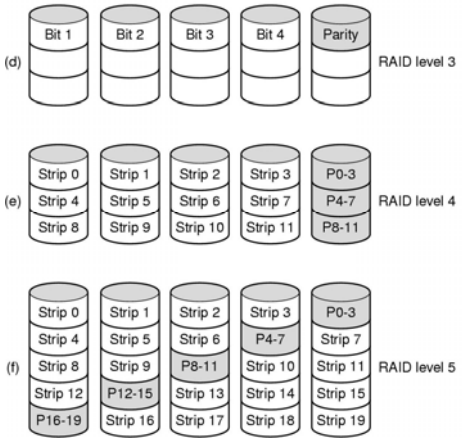
- Physical geometry of a disk with two zones
- A possible virtual geometry for this disk

Disk Hardware (3)



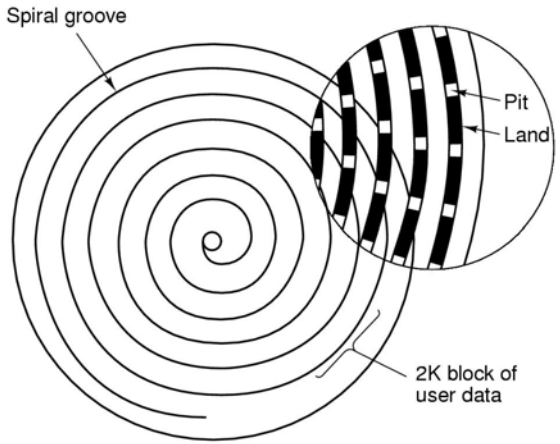
- Raid levels 0 through 2
- Backup and parity drives are shaded

Disk Hardware (4)



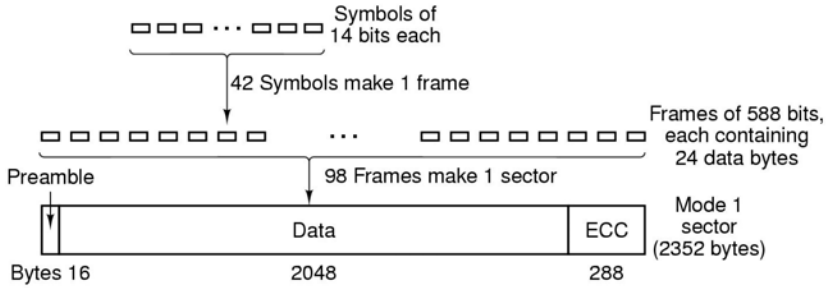
- Raid levels 3 through 5
- Backup and parity drives are shaded

Disk Hardware (5)



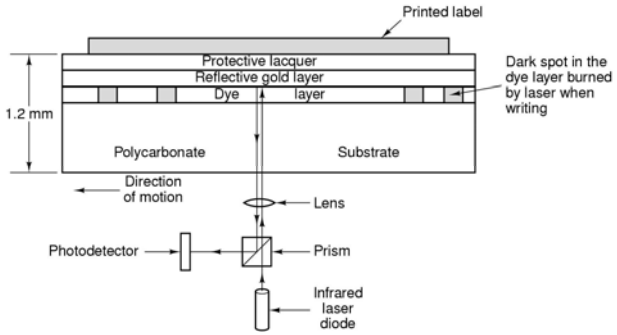
Recording structure of a CD or CD-ROM

Disk Hardware (6)



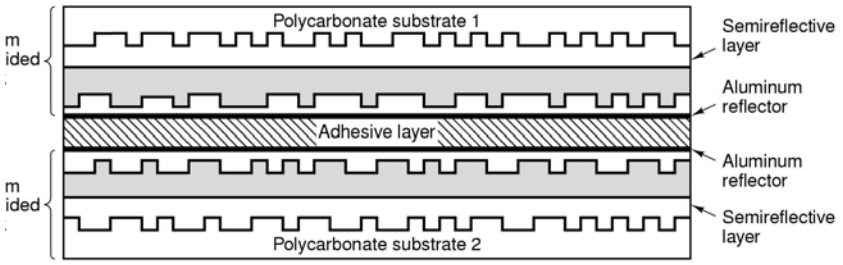
Logical data layout on a CD-ROM

Disk Hardware (7)



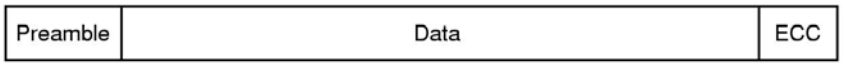
- Cross section of a CD-R disk and laser
  - not to scale
- Silver CD-ROM has similar structure
  - without dye layer
  - with pitted aluminum layer instead of gold

Disk Hardware (8)



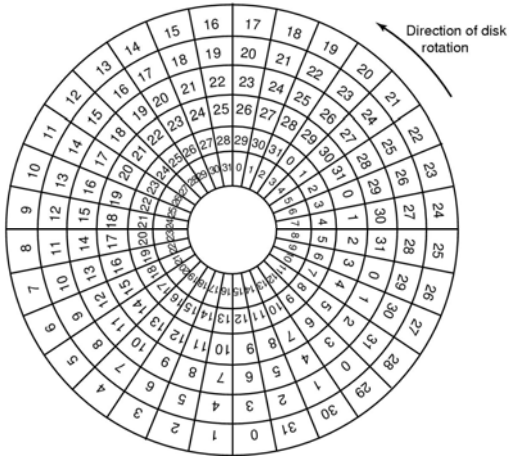
A double sided, dual layer DVD disk

Disk Formatting (1)



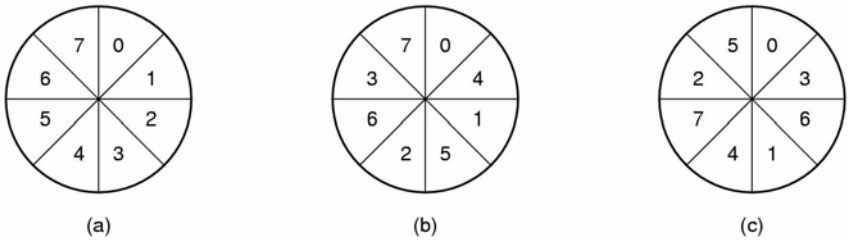
A disk sector

Disk Formatting (2)



An illustration of cylinder skew

Disk Formatting (3)

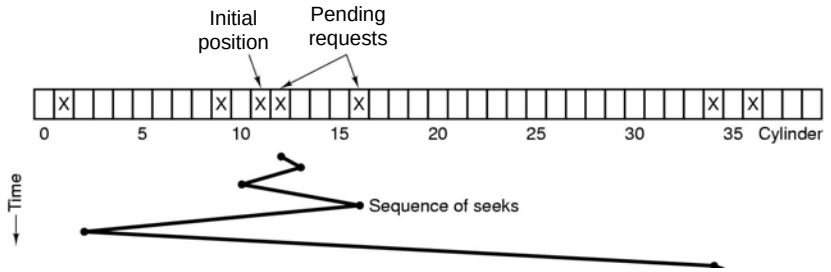


- No interleaving
- Single interleaving
- Double interleaving

Disk Arm Scheduling Algorithms (1)

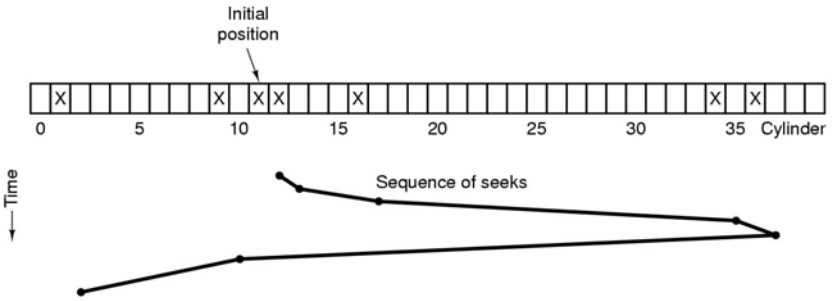
- Time required to read or write a disk block determined by 3 factors
  1. Seek time
  2. Rotational delay
  3. Actual transfer time
- Seek time dominates
- Error checking is done by controllers

Disk Arm Scheduling Algorithms (2)



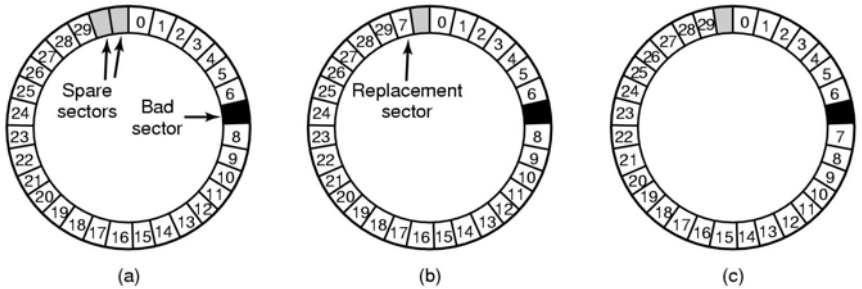
Shortest Seek First (SSF) disk scheduling algorithm

Disk Arm Scheduling Algorithms (3)



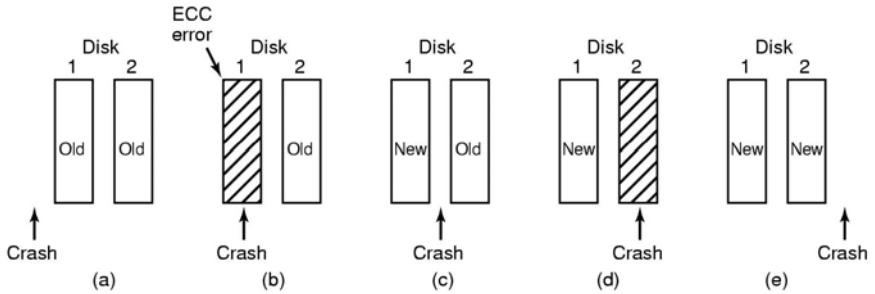
The elevator algorithm for scheduling disk requests

Error Handling



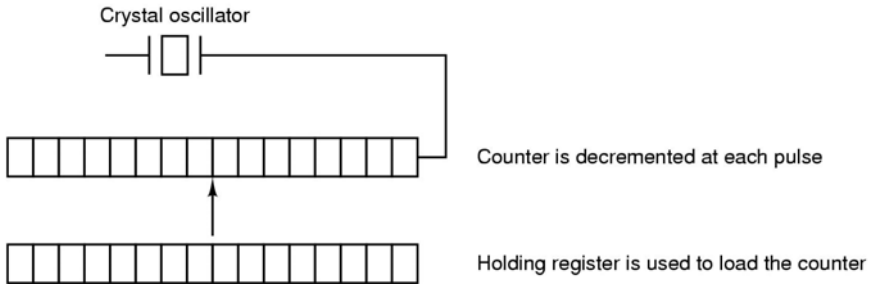
- A disk track with a bad sector
- Substituting a spare for the bad sector
- Shifting all the sectors to bypass the bad one

Stable Storage



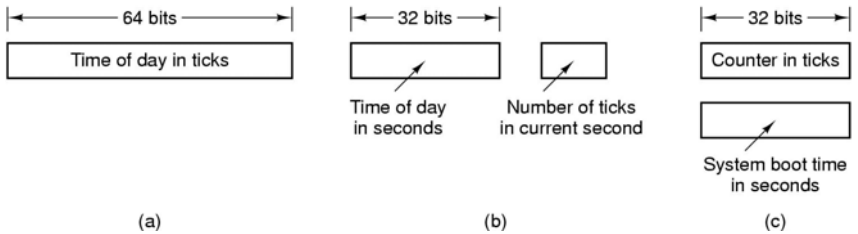
Analysis of the influence of crashes on stable writes

Clocks
Clock Hardware



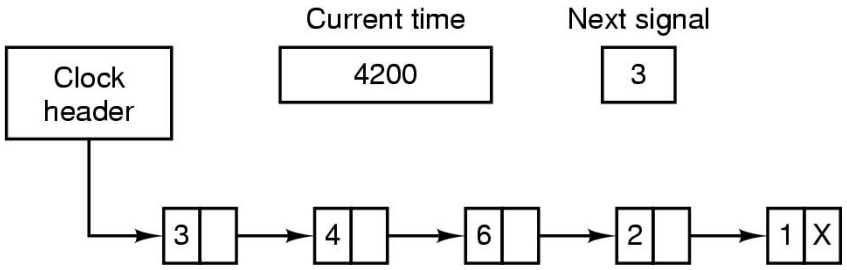
A programmable clock

Clock Software (1)



Three ways to maintain the time of day

Clock Software (2)



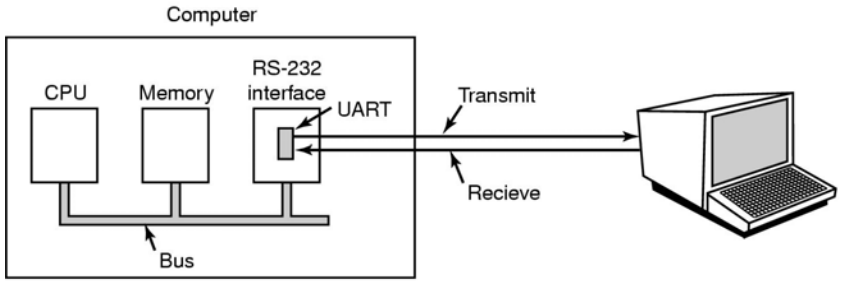
Simulating multiple timers with a single clock

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

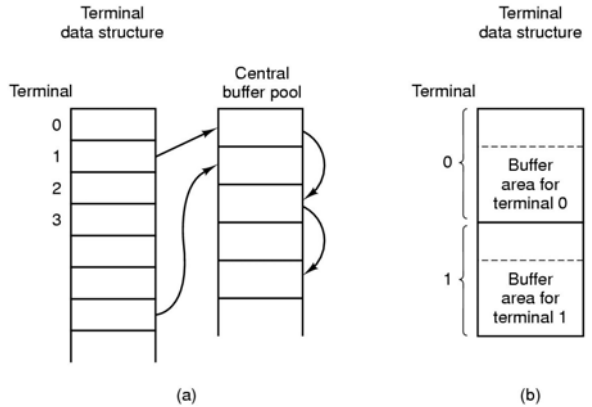
Character Oriented Terminals

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

Input Software (1)



- Central buffer pool
- Dedicated buffer for each terminal

Input Software (2)

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

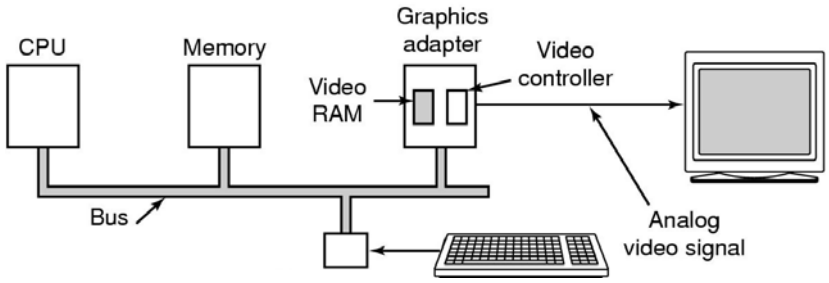
Characters handled specially in canonical mode

Output Software

Escape sequence	Meaning
ESC [*n* A	Move up *n* lines
ESC [*n* B	Move down *n* lines
ESC [*n* C	Move right *n* spaces
ESC [*n* D	Move left *n* spaces
ESC [*m* ; *n* H	Move cursor to (*m*,*n*)
ESC [*s* J	Clear screen from cursor (0 to end, 1 from start, 2 all)
ESC [*s* K	Clear line from cursor (0 to end, 1 from start, 2 all)
ESC [*n* L	Insert *n* lines at cursor
ESC [*n* M	Delete *n* lines at cursor
ESC [*n* P	Delete *n* chars at cursor
ESC [*n* @	Insert *n* chars at cursor
ESC [*n* m	Enable rendition *n* (0=normal, 4=bold, 5=blinking, 7=reverse)
ESC M	Scroll the screen backward if the cursor is on the top line

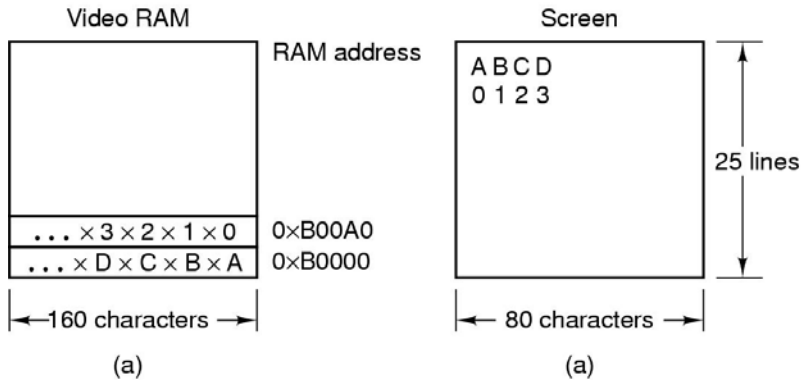
The ANSI escape sequences
- accepted by terminal driver on output
- ESC is ASCII character (0x1B)
- n,m, and s are optional numeric parameters

Display Hardware (1)



Memory-mapped displays
- driver writes directly into display's video RAM

Display Hardware (2)

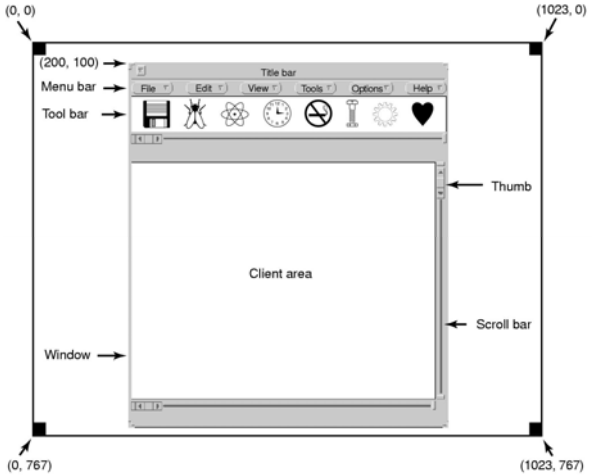


  - simple monochrome display
  - character mode
- Corresponding screen
  - the Xs are attribute bytes

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.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 */
```

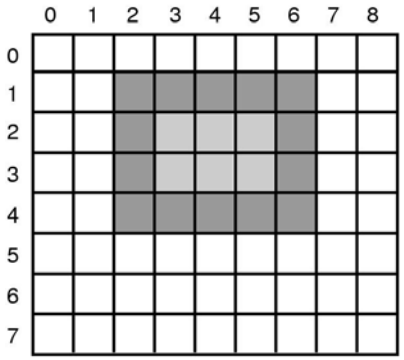
Skeleton of a Windows main program (part 1)

Output Software for Windows (3)

```
      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 */
}
```

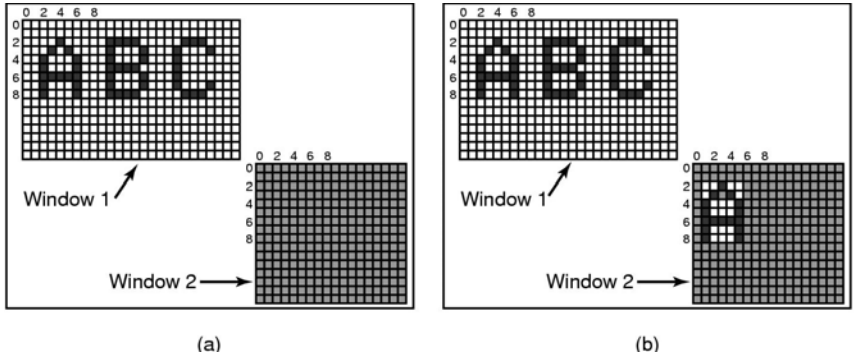
Skeleton of a Windows main program (part 2)

Output Software for Windows (4)



An example rectangle drawn using *Rectangle*

Output Software for Windows (5)



(a) (b)

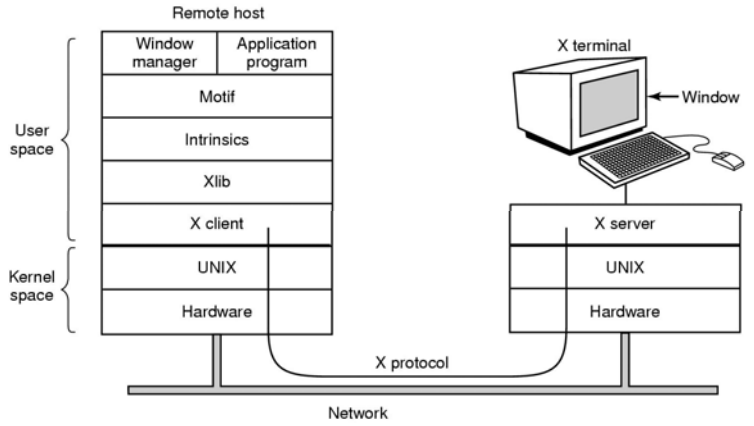
- Copying bitmaps using *BitBlt*.
  - before
  - after

Output Software for Windows (6)

Examples of character outlines at different point sizes

Network Terminals

X Windows (1)



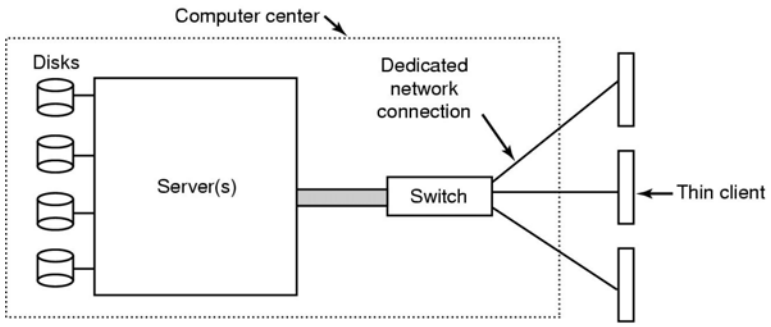
Clients and servers in the M.I.T. X Window System

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 */
}
```

Skeleton of an X Windows application program

The SLIM Network Terminal (1)



The architecture of the SLIM terminal system

The SLIM Network Terminal (2)

Message	Meaning
SET	Update a rectangle with new pixels
FILL	Fill a rectangle with one pixel value
BITMAP	Expand a bitmap to fill a rectangle
COPY	Copy a rectangle from one part of the frame buffer to another
CSCS	Convert a rectangle from television color (YUV) to RGB

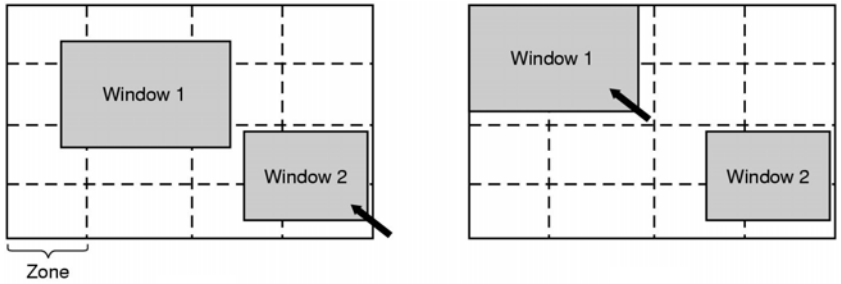
Messages used in the SLIM protocol from the server to the terminals

Power Management (1)

Device	Li et al. (1994)	Lorch and Smith (1998)
Display	68%	39%
CPU	12%	18%
Hard disk	20%	12%
Modem		6%
Sound		2%
Memory	0.5%	1%
Other		22%

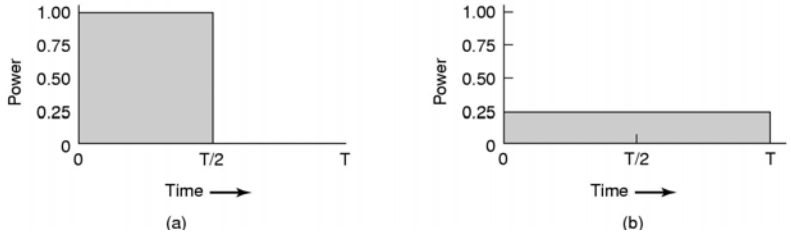
Power consumption of various parts of a laptop computer

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