CS 444 Operating Systems

Chapter 12 Operating System Design

J. Holly DeBlois

October 8, 2025

goals.png

- a

1. Define abstractions.
2. Provide primitive operations.
3. Ensure isolation.
4. Manage the hardware.

- b
- c

corbato.png

First, it is important to emphasize the value of simplicity and elegance, for complexity has a way of compounding difficulties and as we have seen, creating mistakes. My definition of elegance is the achievement of a given functionality with a minimum of mechanism and a maximum of clarity.

- a
- b
- c

paradigms.png

```
main( )
{
        int ... ;

        init( );
        do_something( );
        read(...);
        do_something_else( );
        write(...);
        keep_going( );
        exit(0);
}
```

(a)

```
main( )
{
        mess_t msg;

        init( );
        while (get_message(&msg)) {
                switch (msg.type) {
                        case 1: ... ;
                        case 2: ... ;
                        case 3: ... ;
                }
        }
}
```

(b)

Figure 12-1. (a) Algorithmic code. (b) Event-driven code.

- a
- b

system_call.png

- a
- b

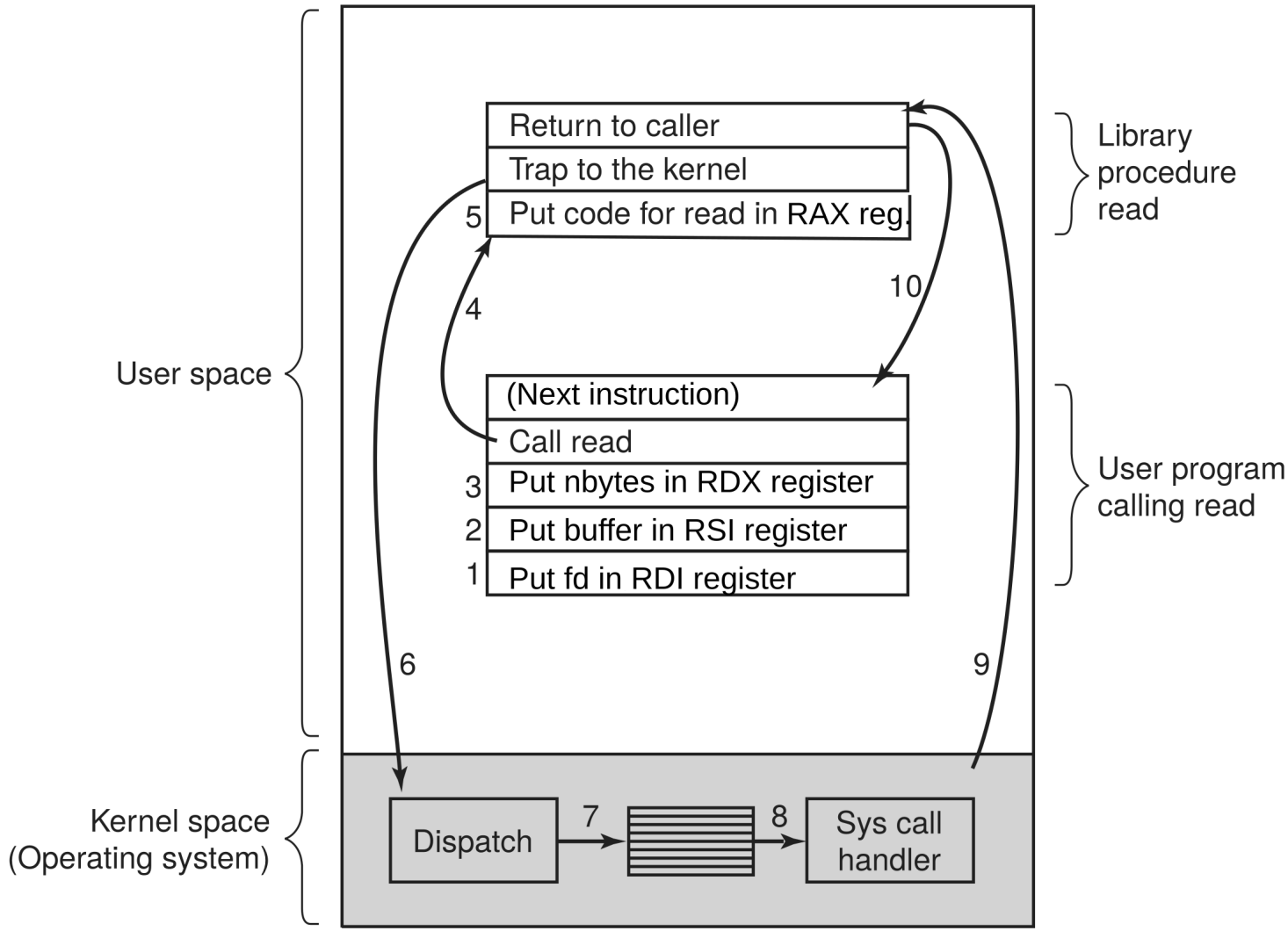


Figure 1-17. The 10 steps in making the system call read(fd, buffer, nbytes).

minimal_mechanism.png

exec(name, argp, envp);

which loads the executable file *name* and gives it arguments pointed to by *argp* and environment variables pointed to by *envp*. Sometimes it is convenient to list the arguments explicitly, so the library contains procedures that are called as follows:

execl(name, arg0, arg1, ..., argn, 0);
execle(name, arg0, arg1, ..., argn, envp);

- a
- b
- c

layered_system.png

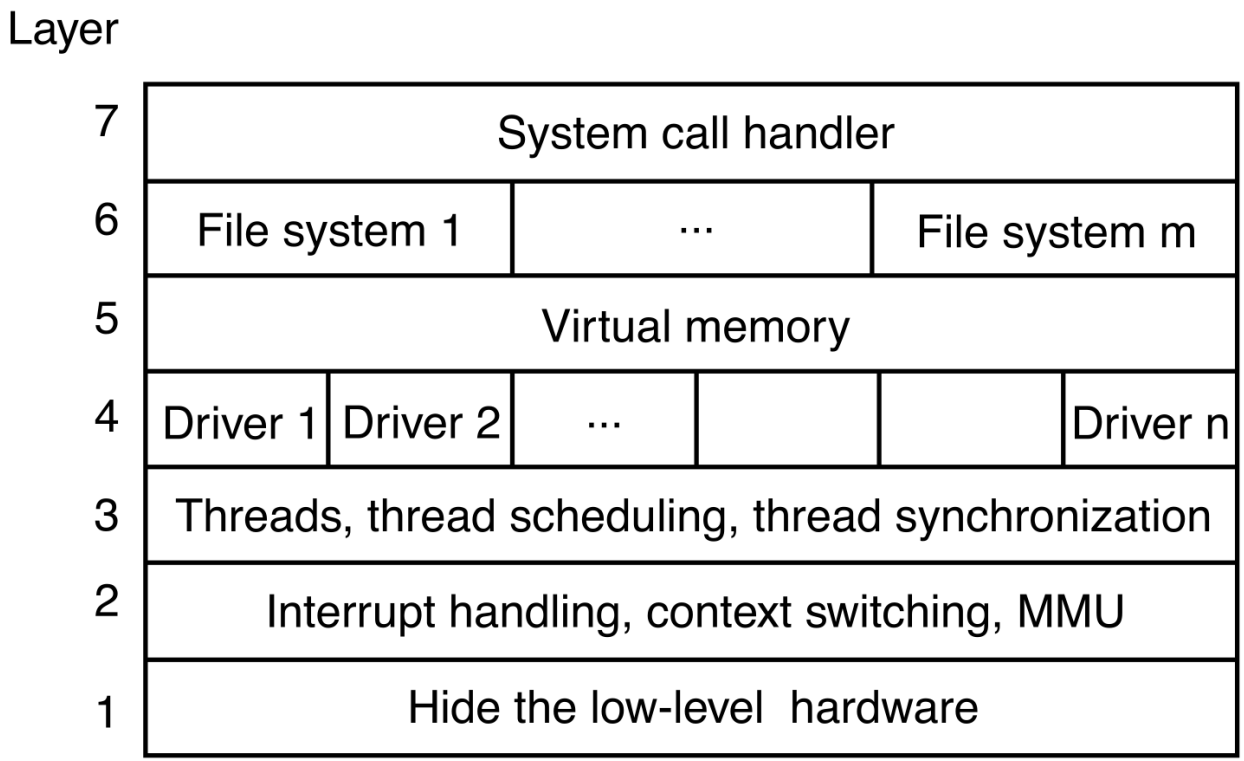


Figure 12-2. One possible design for a modern layered operating system.

- a
- b

windows.png

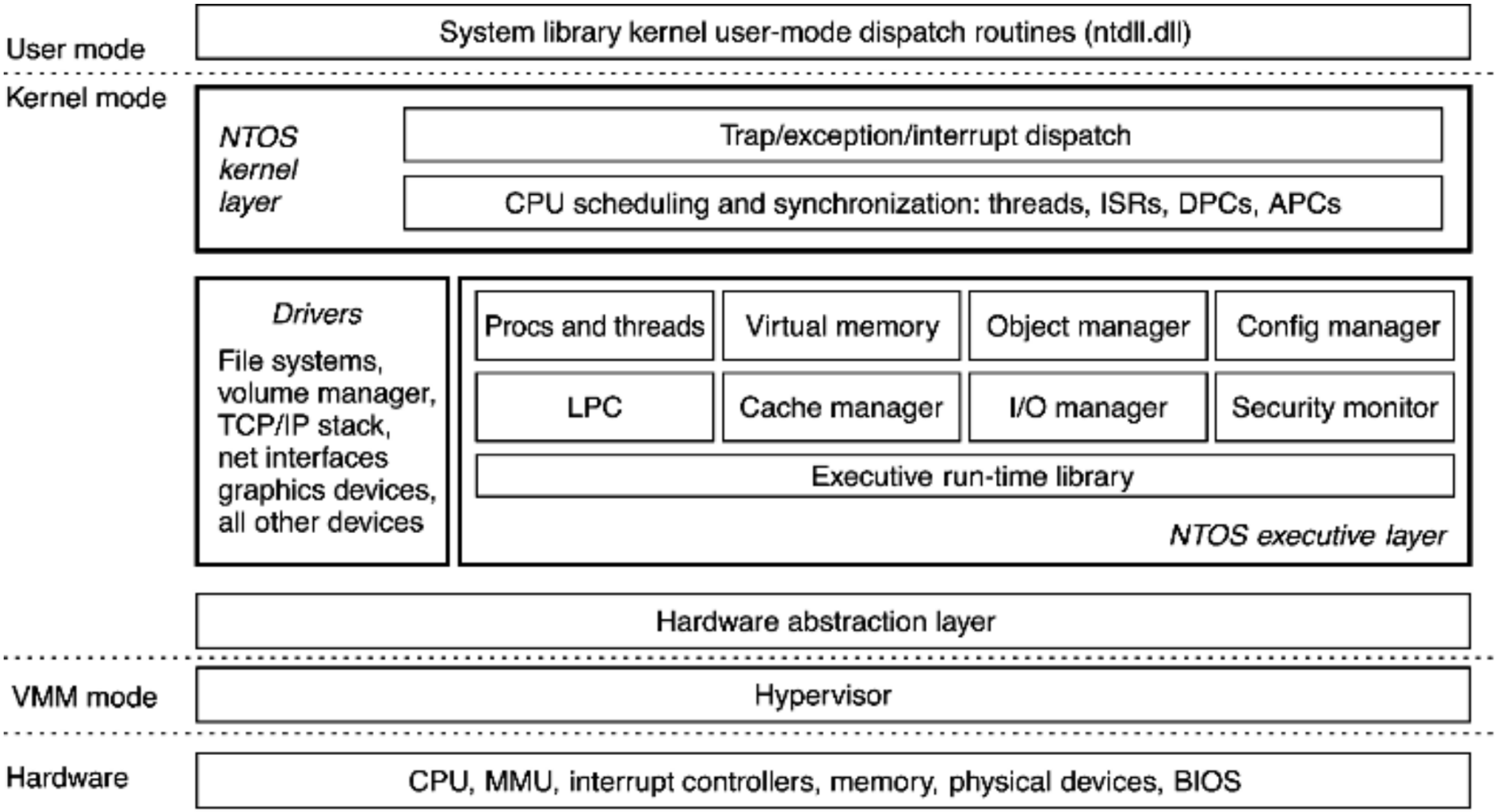


Figure 11-11. Windows kernel-mode organization.

microkernel.png

- a

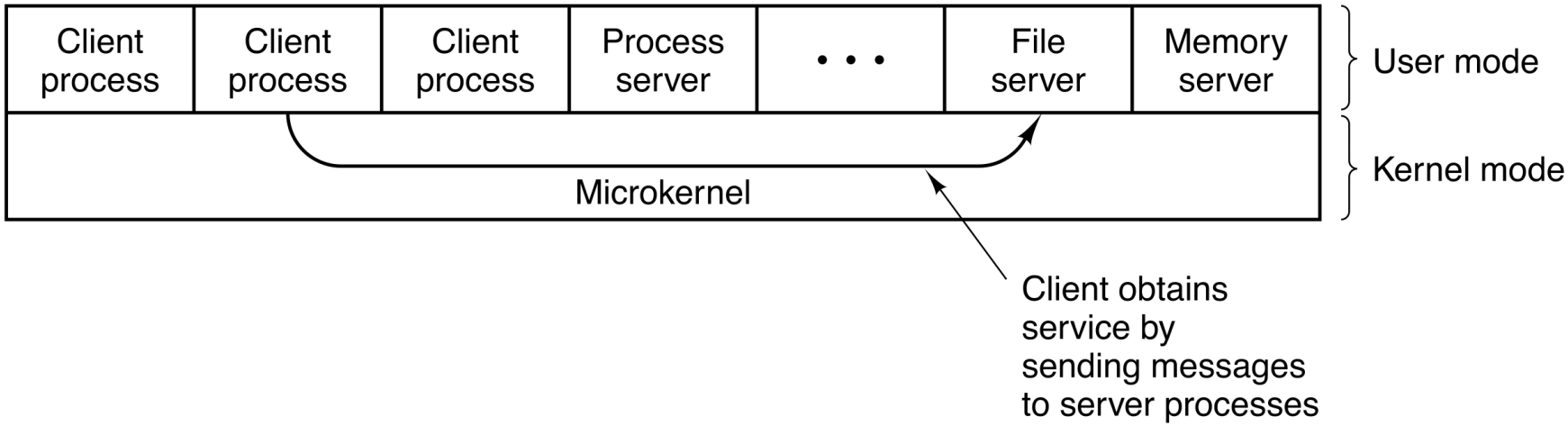


Figure 12-3. Client-server computing based on a microkernel.

- b
- c

naming.png

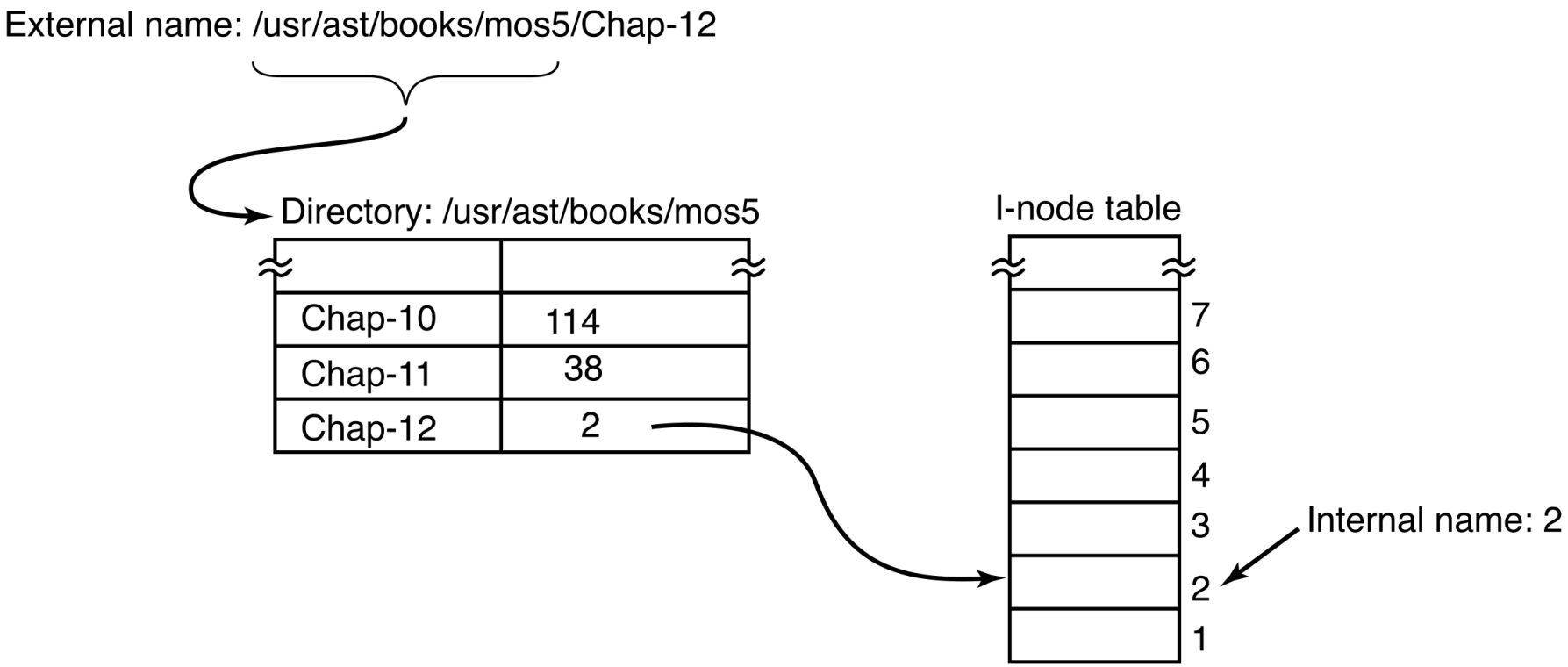


Figure 12-4. Directories are used to map external names onto internal names.

- a
- b

static_structure.png

```
found = 0;
for (p = &proc_table[0]; p < &proc_table[PROC_TABLE_SIZE]; p++) {
        if (p->proc_pid == pid) {
                found = 1;
                break;
        }
}
```

Figure 12-5. Code for searching the process table for a given PID.

- a
- b
- c

conditional_compilation.png

```
#include "config.h"

init( )
{
#if (CPU == IA32)
/* IA32 initialization here. */
#endif

#if (CPU == ULTRASPARC)
/* UltraSPARC initialization here. */
#endif
```

(a)

```
}
```

```
#include "config.h"

#if (WORD_LENGTH == 32)
typedef int Register;
#endif

#if (WORD_LENGTH == 64)
typedef long Register;
#endif

Register R0, R1, R2, R3;
```

(b)

Figure 12-6. (a) CPU-dependent conditional compilation. (b) Word-length-dependent conditional compilation.

- a
- b

space-time_trade-off.png

```
#define BYTE_SIZE 8                          /* A byte contains 8 bits */

int bit_count(int byte)
{                                            /* Count the bits in a byte. */
        int i, count = 0;

        for (i = 0; i < BYTE_SIZE; i++)      /* loop over the bits in a byte */
                if ((byte >> i) & 1) count++; /* if this bit is a 1, add to count */
        return(count);                       /* return sum */
}
```

(a)

```
/*Macro to add up the bits in a byte and return the sum. */
#define bit_count(b) ((b&1) + ((b>>1)&1) + ((b>>2)&1) + ((b>>3)&1) + \
                      ((b>>4)&1) + ((b>>5)&1) + ((b>>6)&1) + ((b>>7)&1))
```

(b)

```
/*Macro to look up the bit count in a table. */
char bits[256] = {0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4, 1, 2, 2, 3, 2, 3, 3, ...};
#define bit_count(b) (int) bits[b]
```

(c)

Figure 12-7. (a) A procedure for counting bits in a byte. (b) A macro to count the bits. (c) A macro that counts bits by table lookup.

GIF_example.png

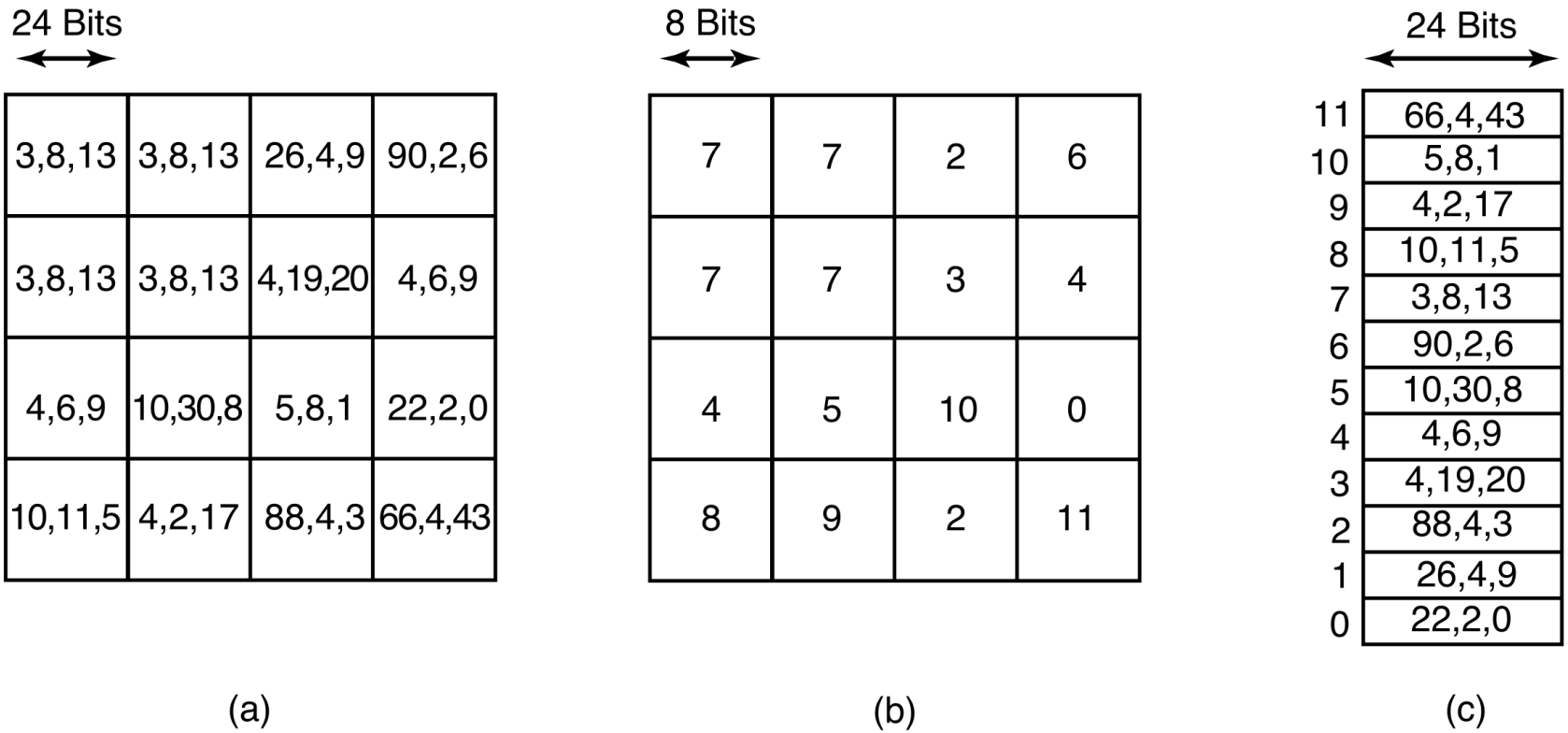


Figure 12-8. (a) Part of an uncompressed image with 24 bits per pixel. (b) The same part compressed with GIF, with 8 bits per pixel. (c) The color palette.

- a
- b

caching_example.png

1. Read the i-node for the root directory (i-node 1).
2. Read the root directory (block 1).
3. Read the i-node for */usr* (i-node 6).
4. Read the */usr* directory (block 132).
5. Read the i-node for */usr/ast* (i-node 26).
6. Read the */usr/ast* directory (block 406).

- a
- b
- c

example_continued.png

Path	I-node number
/usr	6
/usr/ast	26
/usr/ast/mbox	60
/usr/ast/books	92
/usr/bal	45
/usr/bal/paper.ps	85

Figure 12-9. Part of the i-node cache for Fig. 4-36.

- a
- b
- c

team_structure.png

Title	Duties
Chief programmer	Performs the architectural design and writes the code
Copilot	Helps the chief programmer and serves as a sounding board
Administrator	Manages the people, budget, space, equipment, reporting, etc.
Editor	Edits the documentation, which must be written by the chief programmer
Secretaries	The administrator and editor each need a secretary
Program clerk	Maintains the code and documentation archives
Toolsmith	Provides any tools the chief programmer needs
Tester	Tests the chief programmer's code
Language lawyer	Part timer who can advise the chief programmer on the language

Figure 12-10. Mills' proposal for populating a 10-person chief programmer team.

- a
- b

brooks_solution.png

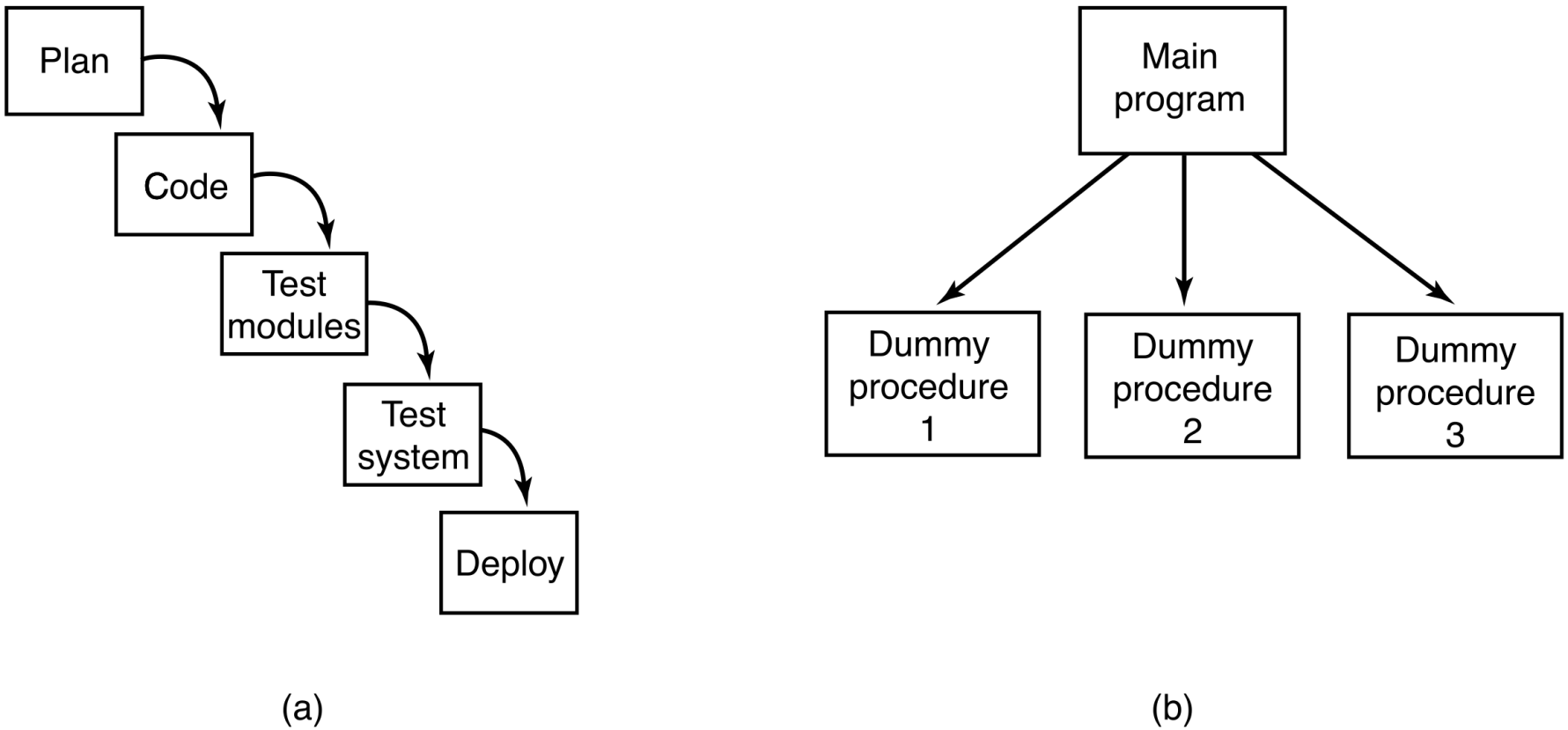


Figure 12-11. (a) Traditional software design progresses in stages. (b) Alternative design produces a working system (that does nothing) starting on day 1.

- a

```
#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 */
}
```

Figure 5-34. A skeleton of an X Window application 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 */

    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 5-36. A skeleton of a Windows main program.