12

OPERATING SYSTEM DESIGN

12.1 THE NATURE OF THE DESIGN PROBLEM

12.2 INTERFACE DESIGN

12.3 IMPLEMENTATION

12.4 PERFORMANCE

12.5 PROJECT MANAGEMENT

12.6 TRENDS IN OPERATING SYSTEM DESIGN

12.7 SUMMARY

```
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)

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

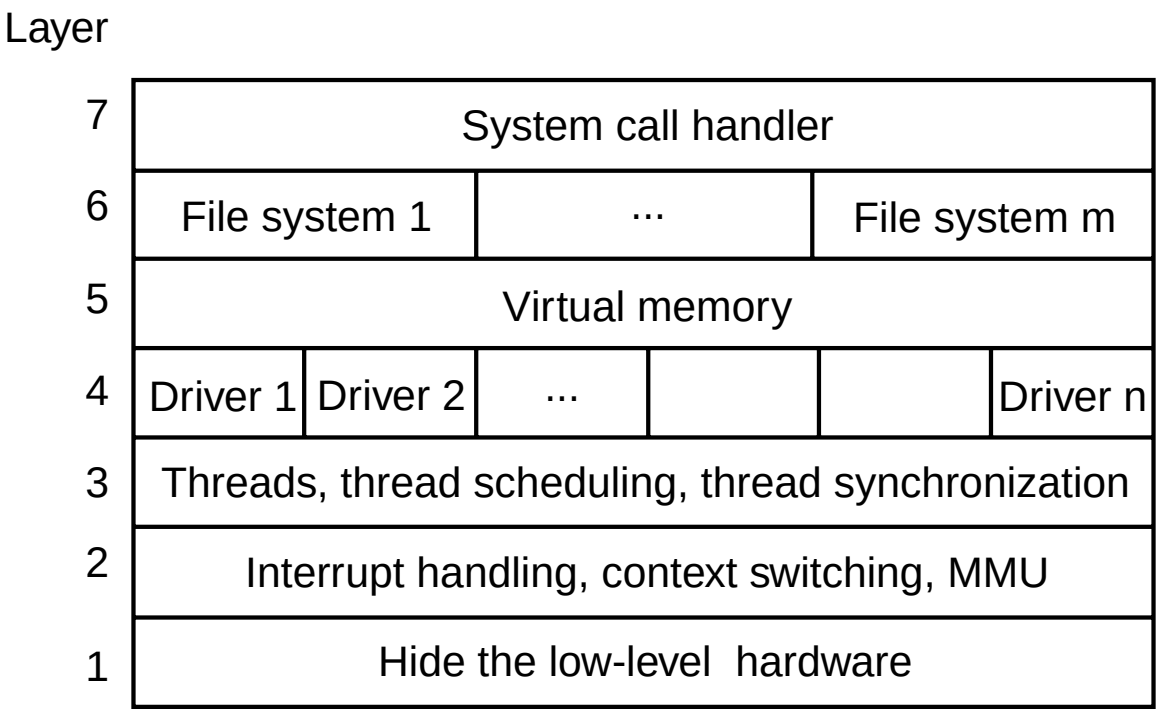

Fig. 12-2. One possible design for a modern layered operating system.

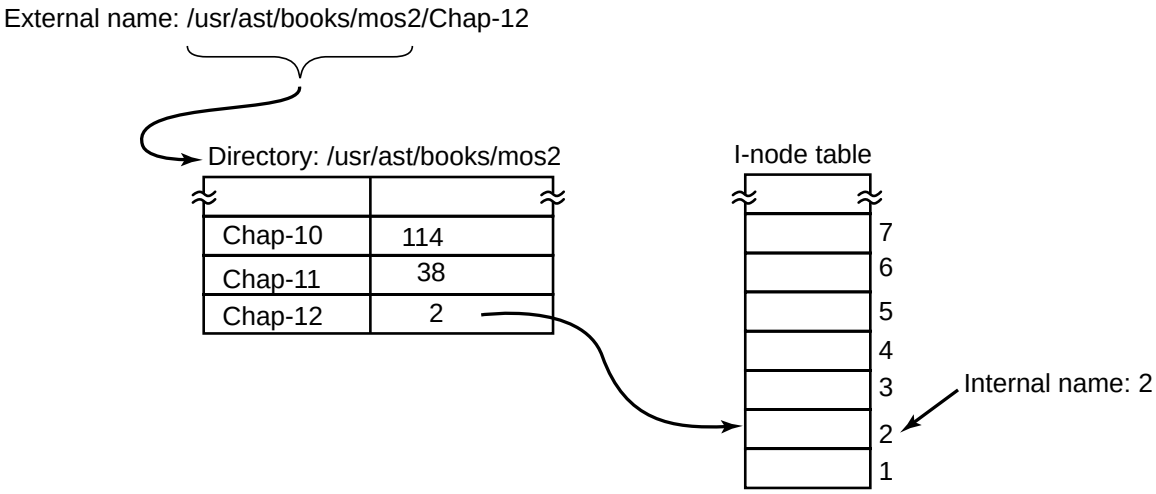

Fig. 12-3. Directories are used to map external names onto internal names.

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

Fig. 12-4. Code for searching the process table for a given PID.

```
#include "config.h"

init( )
{
#if (CPU == PENTIUM)
/* Pentium 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)

Fig. 12-5. (a) CPU-dependent conditional compilation. (b) Word-length dependent conditional compilation.

```
#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)

Fig. 12-6. (a) A procedure for counting bits in a byte. (b) A macro to count the bits.

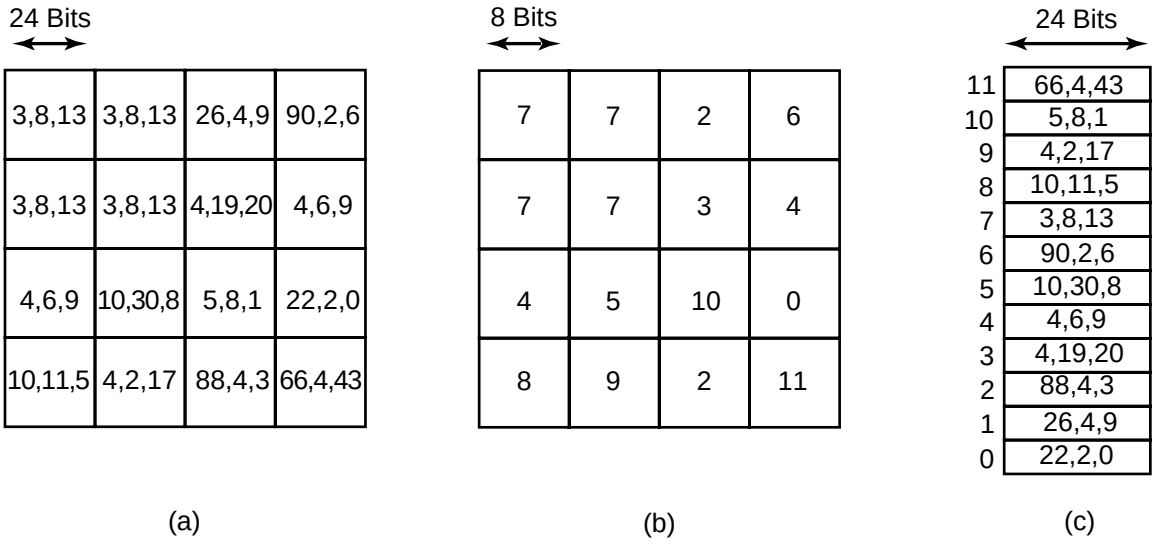

Fig. 12-7. (a) Part of an uncompressed image with 24 bits per pixel. (b) The same part compressed with GIF, with 8 bits per pixel. (c) The color palette.

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

Fig. 12-8. Part of the i-node cache for Fig. 6-39.

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

Fig. 12-9. Mills' proposal for populating a 10-person chief programmer team.

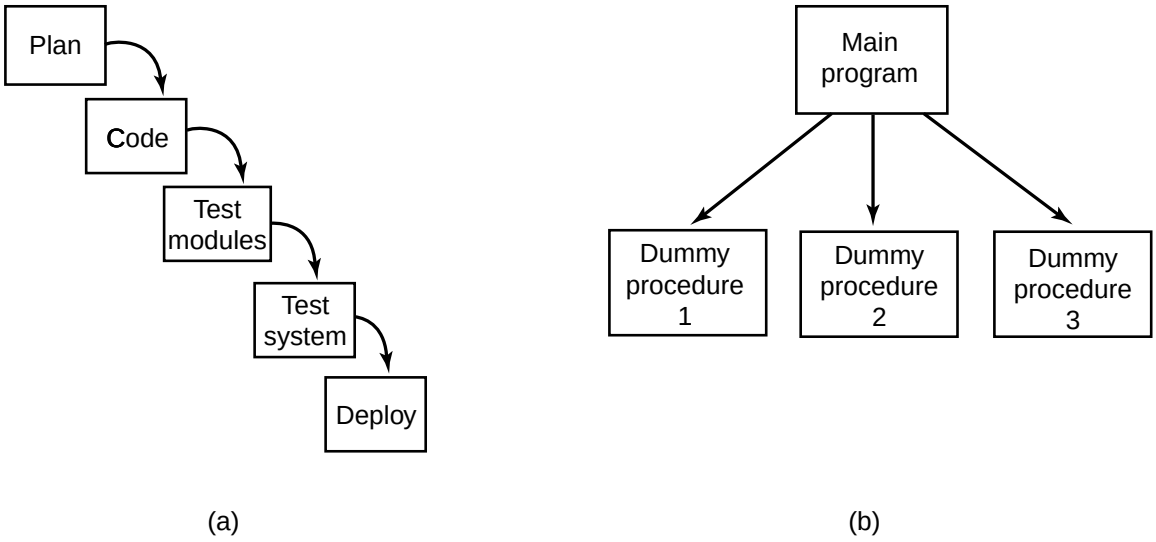

Fig. 12-10. (a) Traditional software design progresses in stages. (b) Alternative design produces a working system (that does nothing) starting on day 1.