Napredni operativni sistemi

Dizajn operativnih sistema

Prof. dr Dragan Stojanović

Katedra za računarstvo
Elektronski fakultet u Nišu

Operating System Design

Chapter 12

Goals

Main goals of general purpose operating systems:

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

Why Is It Hard to Design an Operating System?

- OSs have become extremely large programs
  - Subsystems often interact
- OSs have to deal with concurrency
- OSs have to deal with potentially hostile users
- Many users do want to share their information and resources with selected other users
- OSs live for a very long time.

Why Is It Hard to Design an Operating System? (cont'd)

- OS designers really do not have a good idea of how their systems will be used
  - Need to provide for considerable generality
- Need to be backward compatible with some previous operating system

Interface Design
Guiding Principles

1. Simplicity
2. Completeness
3. Efficiency

Execution Paradigms

```
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.

Implementation

- System Structure
  - Monolitic
  - Layered Systems
  - Exokernels
  - Microkernel-Based Client-Server Systems
  - Extensible Systems
  - Kernel Threads

System Structure
Layered Systems

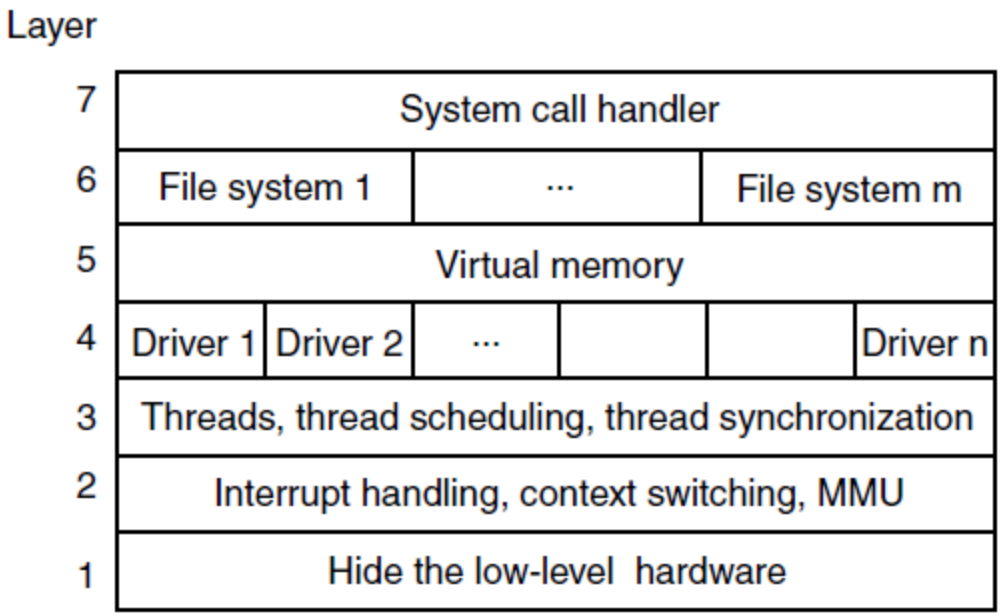

Figure 12-2. One possible design for a modern layered operating system.

Microkernel-Based Client-Server Systems

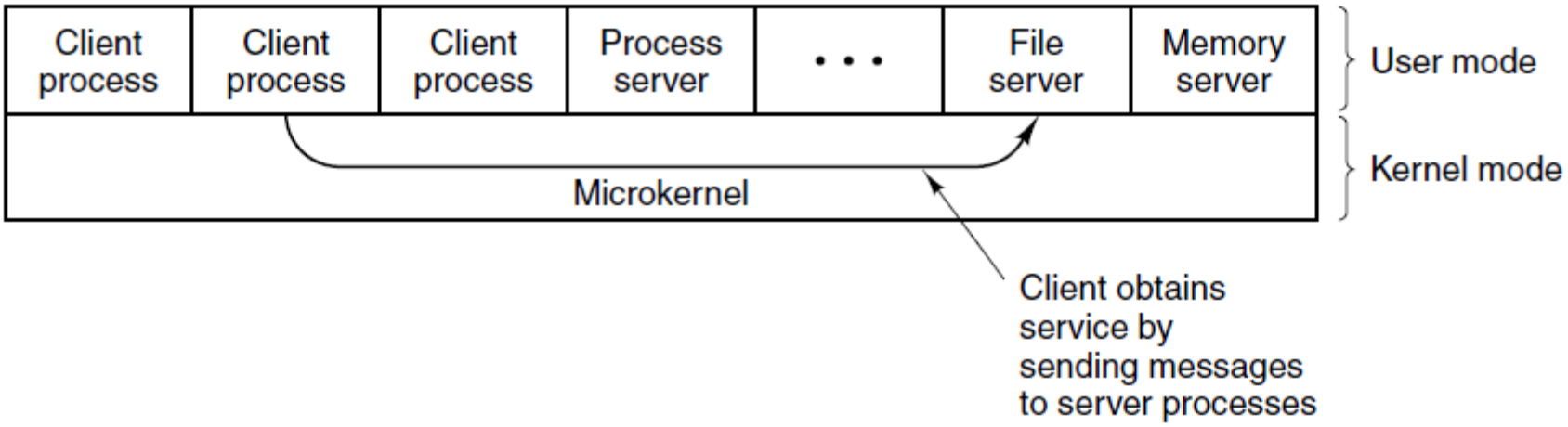

Figure 12-3. Client-server computing based on a microkernel.

Implementation (cont'd)

- Mechanism vs. Policy
  - Example: Multi level scheduling
- Orthogonality
  - Example: Processes and threads in Windows
- Naming
  - External and internal
  - Example: File name, i-node number, fd in Unix
- Binding Time
- Static vs. Dynamic Structures
  - User/kernel stack, process scheduling, kernel

Naming

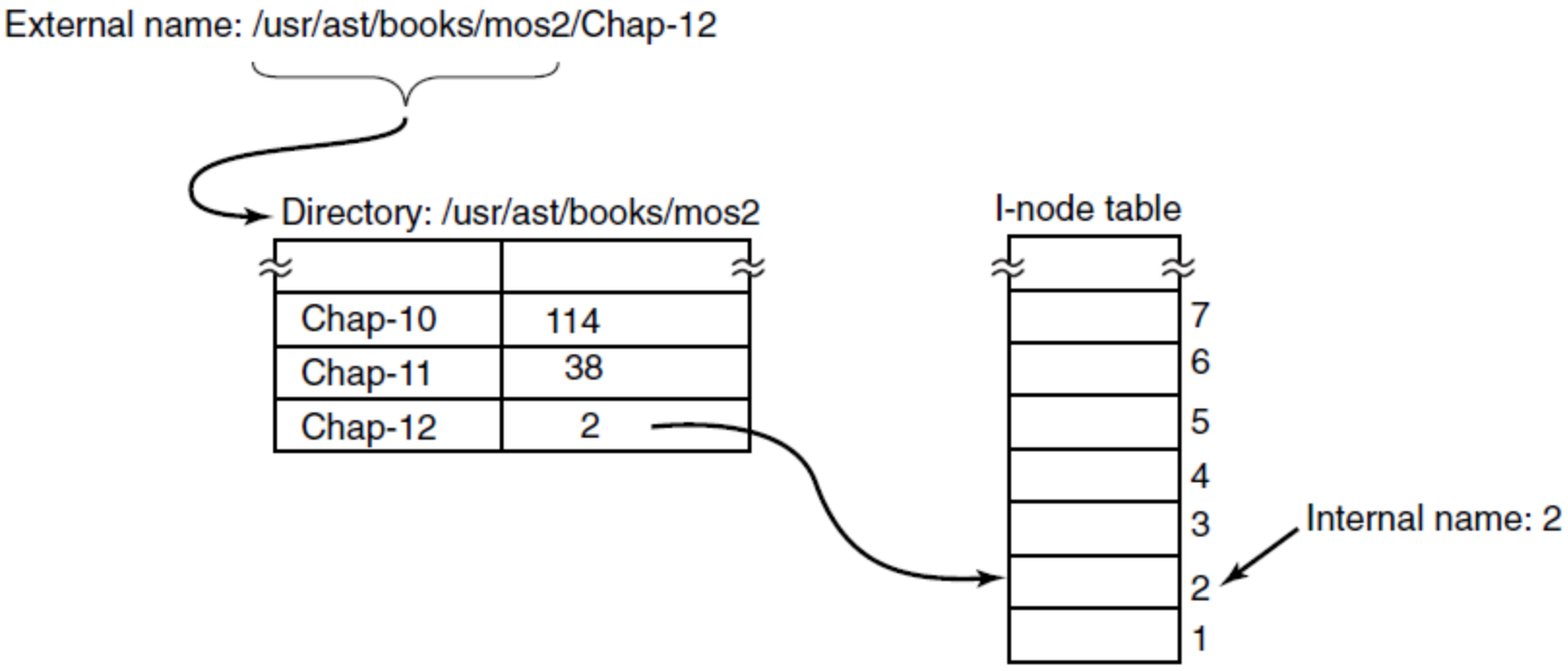

Figure 12-4. Directories are used to map external names onto internal names.

Static vs. Dynamic Structures

```
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.

Implementation (cont'd)

- Top-Down vs. Bottom-Up Implementation
  - From HAL to system calls or vice versa
- Synchronous vs. Asynchronous Communication
  - Blocking vs. non-blocking
- Useful Techniques
  - Hiding the Hardware
  - Interrupts → Threads, unlock operation, message
- Indirection
- Reusability
- …

Hiding the Hardware

```
#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.

Performance

- Why Are Operating Systems Slow?
- What Should Be Optimized?
- Space-Time Trade-offs
  - A general approach to improving performance is to trade off time vs. space
- Caching
  - Whenever it is likely the same result will be needed multiple times
- Hints
- Exploiting Locality

Space-Time Trade-offs (1)

```
#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)

Figure 12-7. (a) A procedure for counting bits in a byte.

Space-Time Trade-offs (2)

```
/*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. (b) A macro to count the bits. (c) A macro that counts bits by table lookup.

Space-Time Trade-offs (3)

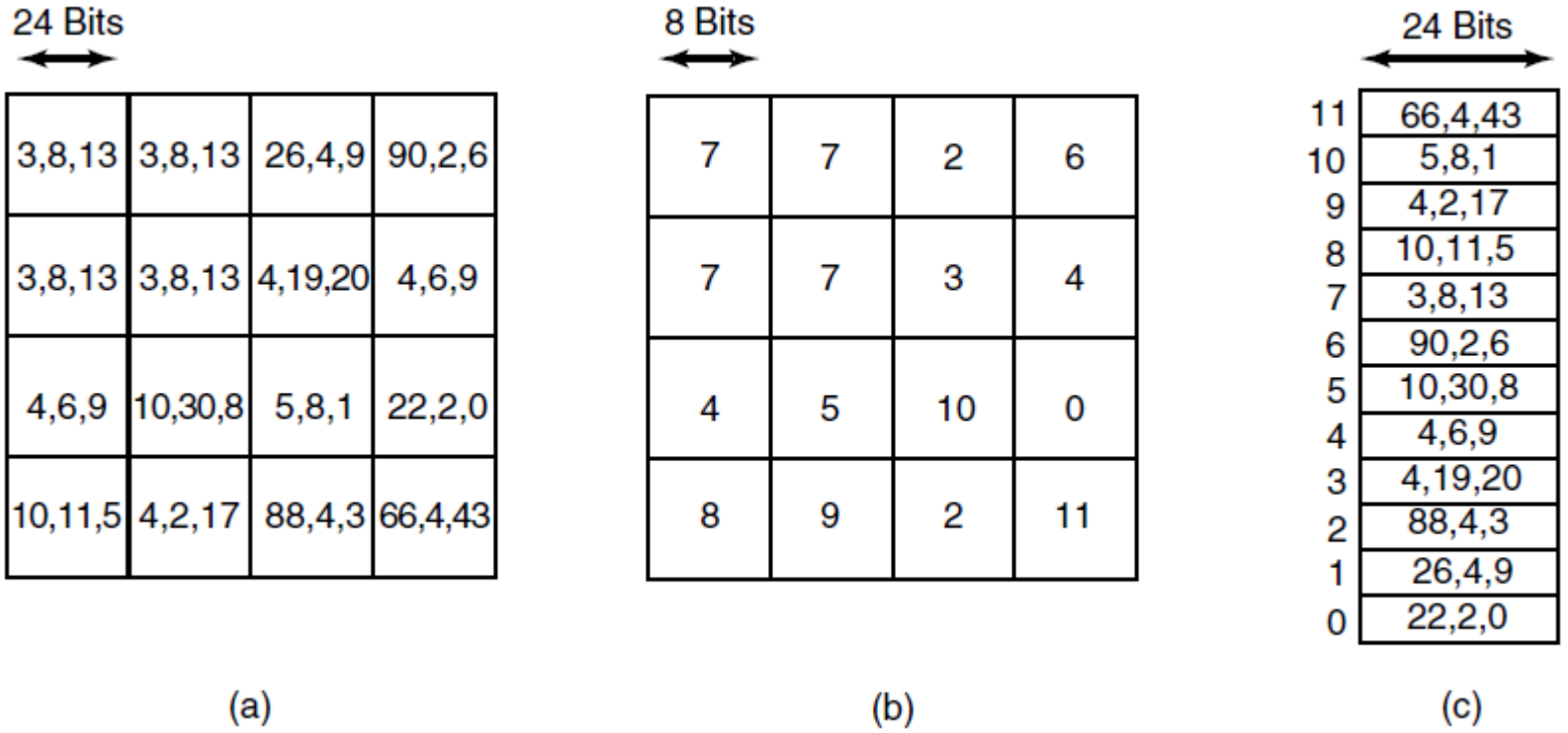

Figure 12-8. (a) Part of an uncompressed image with 24 bits per pixel. (b) The same part compressed with GIF, with 8 bits per pixel. (c) The color palette.

Caching (1)

To look up */usr/ast/mbox* requires the following disk accesses:

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).

Caching (2)

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. 4-34.

Project management

- The Mythical Man Month (1975, 1995)
  - Fred Brooks, one of the designers of OS/360
    - 1/3 Planning
    - 1/6 Coding
    - 1/4 Module testing
    - 1/4 System testing
- Team Structure
- The Role of Experience
  - CTSS – MULTICS – Unix
- No Silver Bullet

Project Management
Team Structure

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.

The Role of Experience

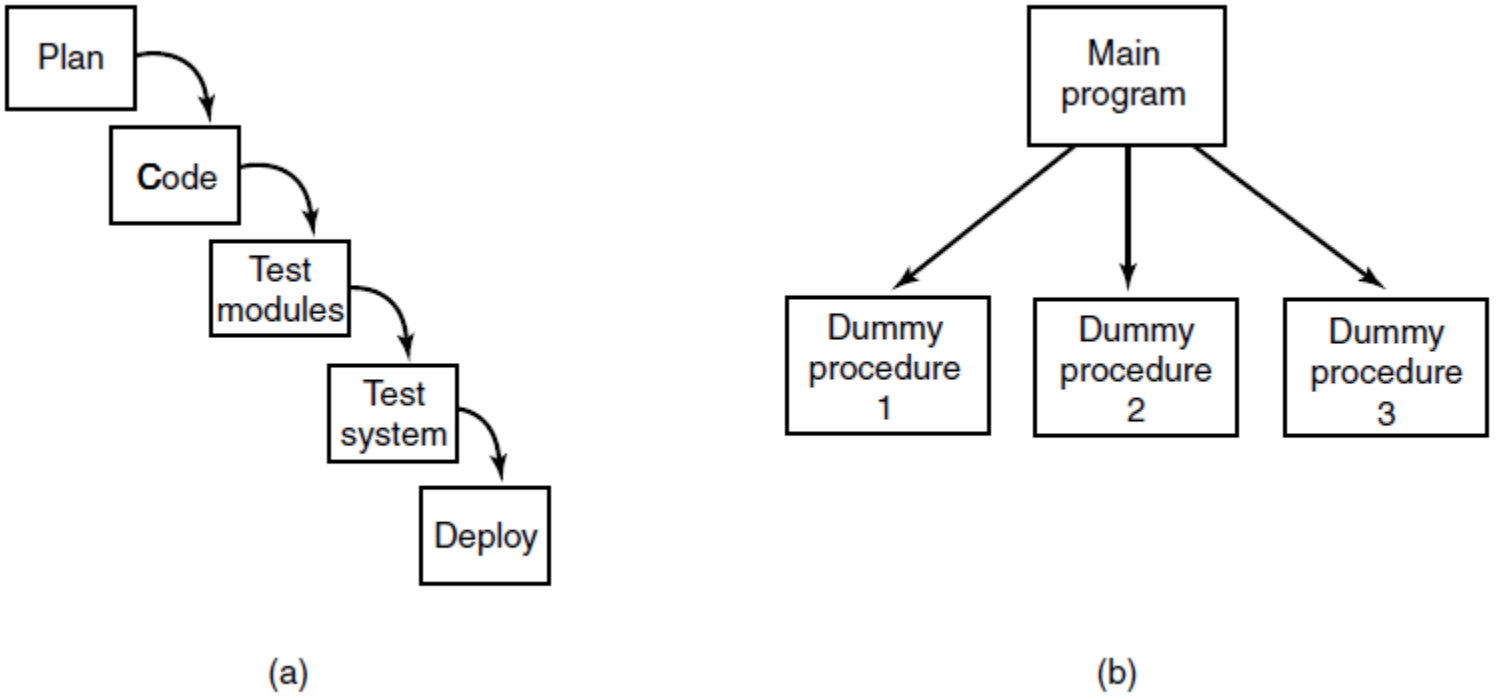

Figure 12-11. (a) Traditional software design progresses in stages. (b) Alternative design produces a working system (that does nothing) starting on day 1.

Trends In Operating System Design

- Virtualization and the Cloud
- Manycore Chips
- Large-Address-Space Operating Systems
- Seamless Data Access
- Battery-Powered Computers
- Embedded Systems

Trends In Operating System Design
Virtualization and the Cloud

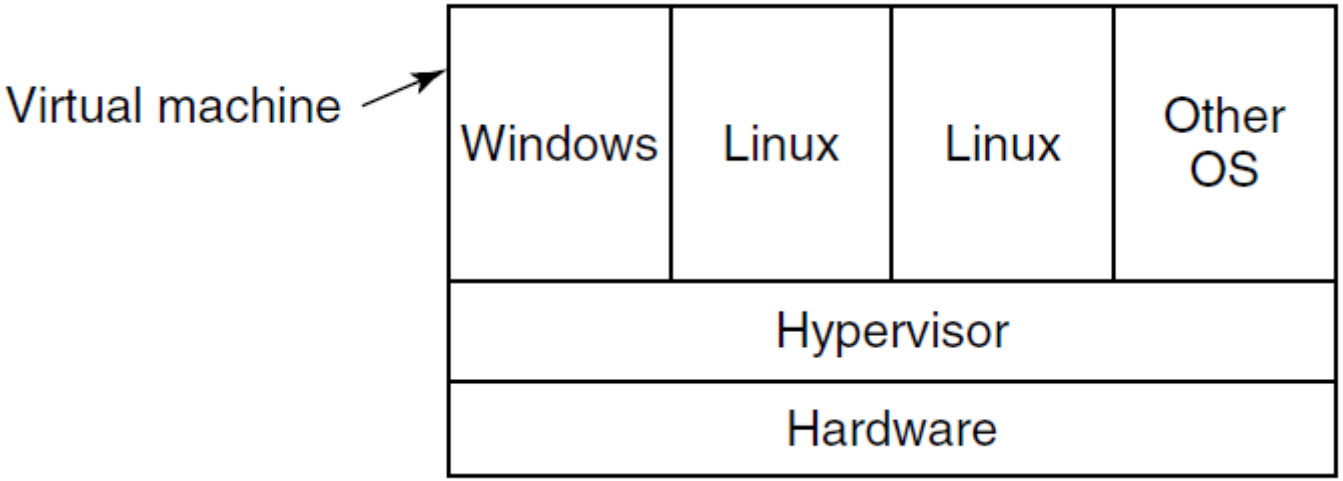

Figure 12-12. A hypervisor running four virtual machines.

End

Chapter 12