



# Operating System Concepts

Che-Wei Chang

[chewei@mail.cgu.edu.tw](mailto:chewei@mail.cgu.edu.tw)

Department of Computer Science and Information Engineering, Chang Gung University

# Contents

1. Introduction
- ➔ 2. System Structures
3. Process Concept
4. Multithreaded Programming
5. Process Scheduling
6. Synchronization
7. Deadlocks
8. Memory-Management Strategies
9. Virtual-Memory Management
10. File System
11. Implementing File Systems
12. Secondary-Storage Systems

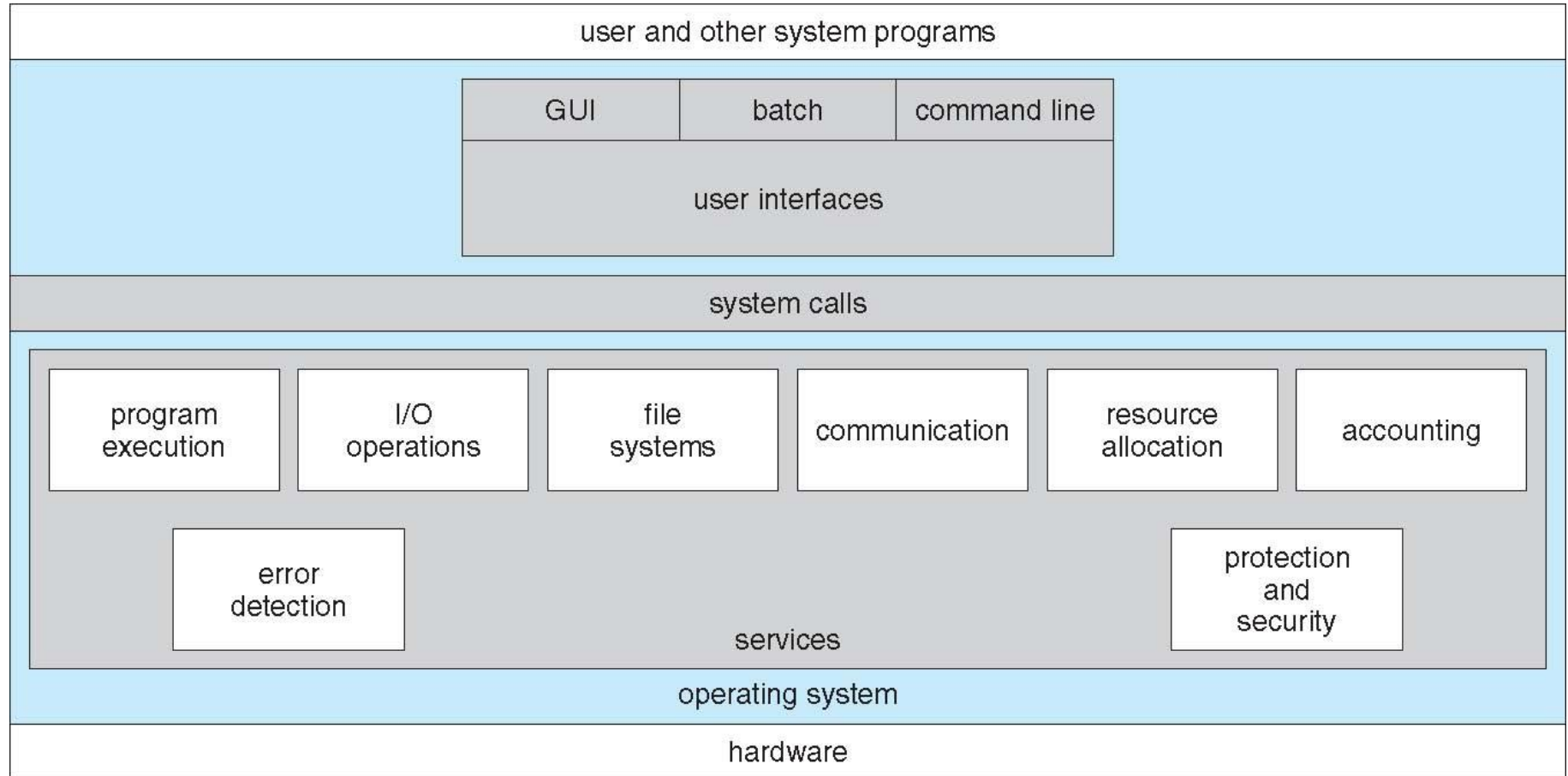




# Chapter 2.

## System Structures

# A View of Operating System Services



# Operation-System Services (1 / 3)

## ▶ User Interface (UI)

- Command line Interface, batch interface, graphical user interface (GUI), etc.
- Interface between the user and the operating system
- Friendly UI's
  - Command-line-based interfaces or mused-based window-and-menu interface
- For example, UNIX shell and command.com in MS-DOS

## ▶ Program Execution

- Loading, running, terminating, etc.



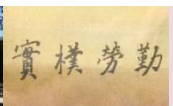
# Operation-System Services (2 / 3)

- ▶ I/O Operations
  - General/special operations for devices
    - Efficiency & protection
- ▶ File-System Manipulation
  - Read, write, create, delete, etc.
  - File and Directory Management
  - Permission Management
- ▶ Communications
  - Intra-processor or inter-processor communication
    - Shared memory or message passing



# Operation–System Services (3 / 3)

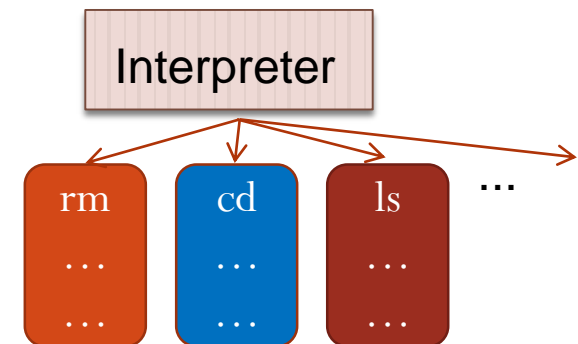
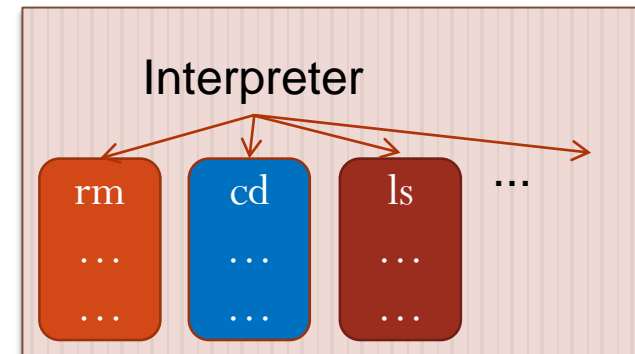
- ▶ Error Detection
  - Possible errors from CPU, memory, devices, user programs  
→ Ensure correct & consistent computing
- ▶ Resource Allocation
  - Multiple users might use some shared resources
  - Resource management has to be efficiency
- ▶ Accounting
  - Statistics or accounting
- ▶ Protection and Security
  - Ensure that all access to system resources is controlled
  - Enforce that all requests are authenticated





# User OS Interface— Command Interpreter

- ▶ Two Approaches to Implement a Command-Line Interpreter (CLI):
  - Contain codes to execute commands
    - Fast but the interpreter tends to be big
    - Painful in revision
  - Implement commands as system programs → Search programs which correspond to the commands (UNIX)
    - Using parameter passing
    - Being slow
    - Inconsistent interpretation of parameters





# User OS Interface— GUI

## ▶ Components

- Screen, icons, folders, pointer, etc.

## ▶ History

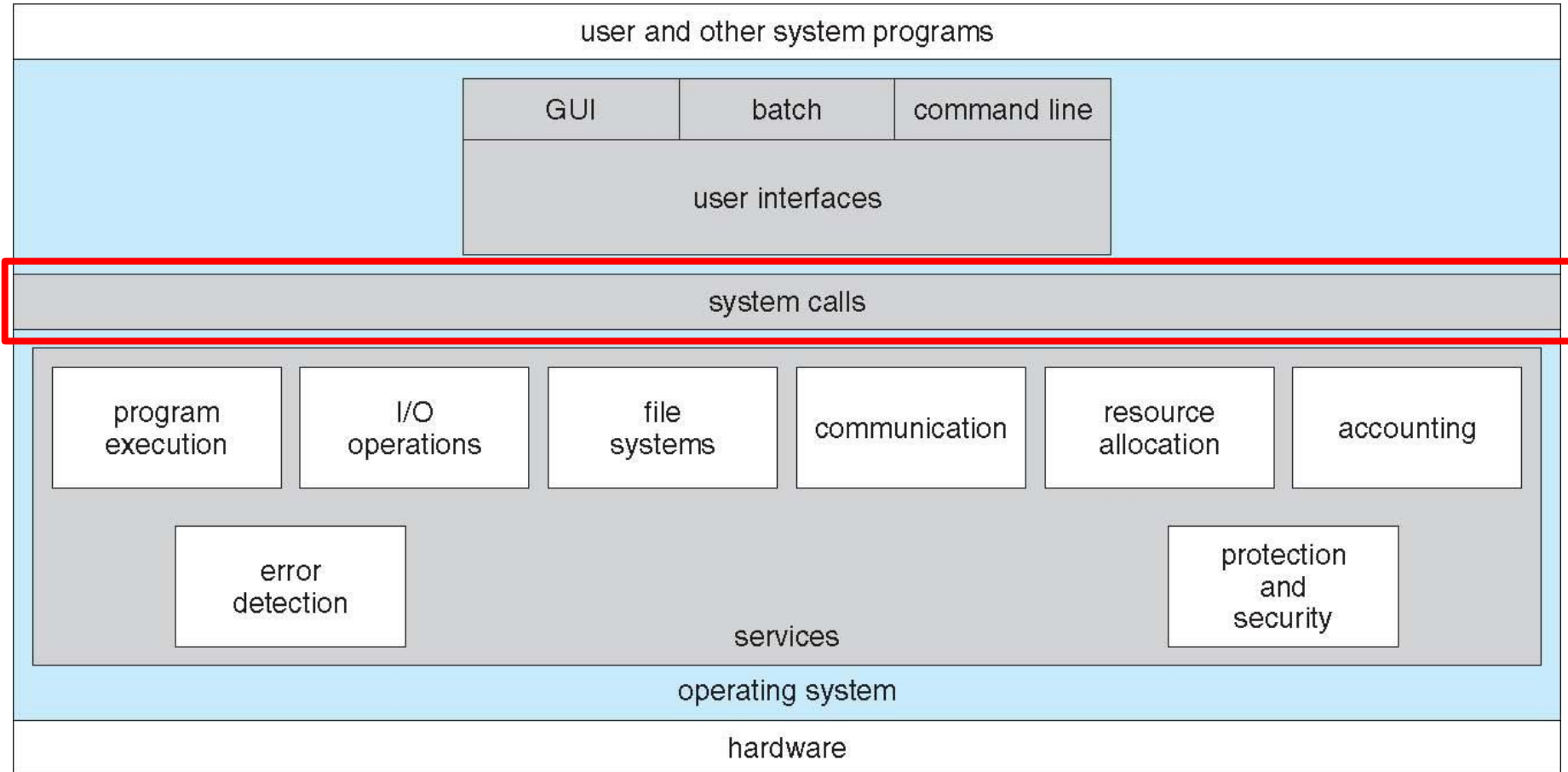
- Xerox PARC research facility (1970's)
- Mouse— 1968
- Mac OS— 1980's
- Windows 1.0~ 10

## ▶ Trends

- Mixture of GUI and command-line interfaces
- Multimedia, intelligence, etc.



# System Calls in an OS



# System Calls (1 / 2)

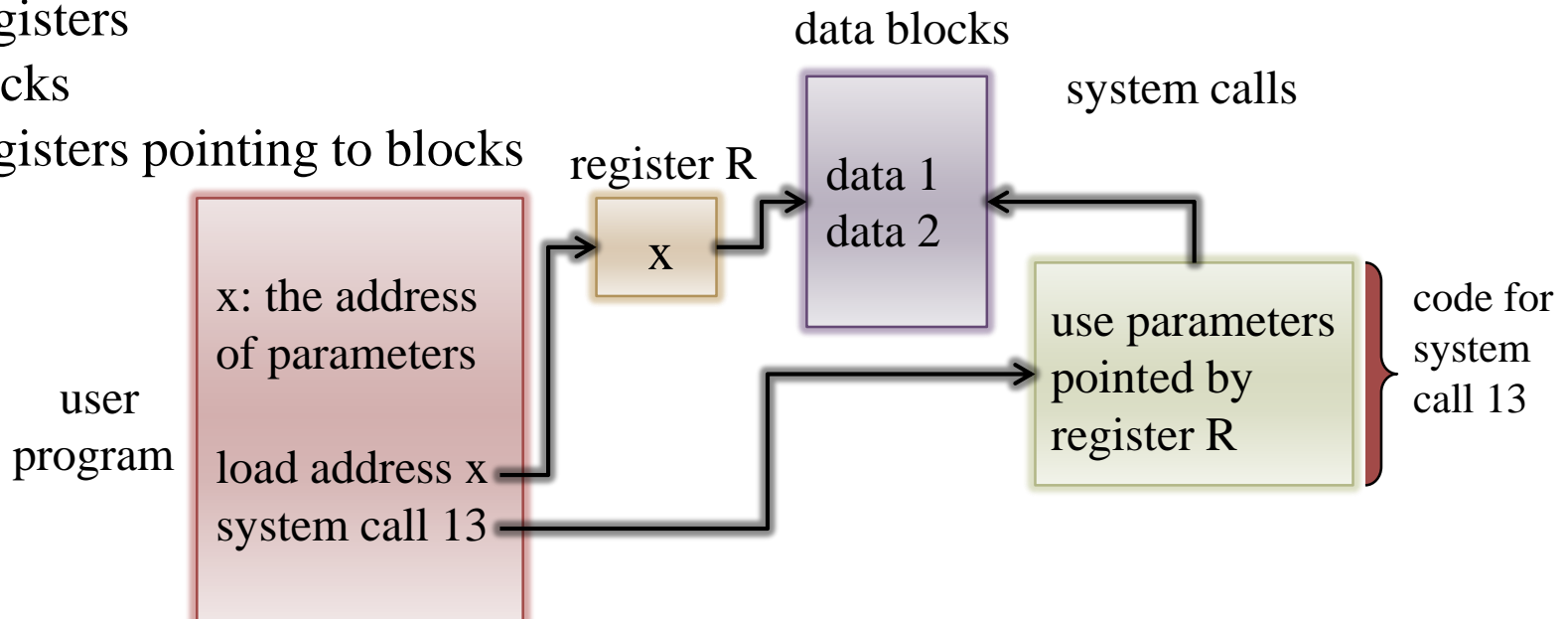
- ▶ System Calls
  - Interface between user processes and the OS
- ▶ Application Programming Interface (API)
  - Most details of OS interface hidden from programmer by API
  - Examples:
    - Win32 API for Windows
    - POSIX\* API for POSIX-based systems including UNIX, Linux, and Mac OS X
  - Benefits (API vs System Calls)
    - Good portability, Ease of use, and Better functionality

\*POSIX: Portable Operating System Interface

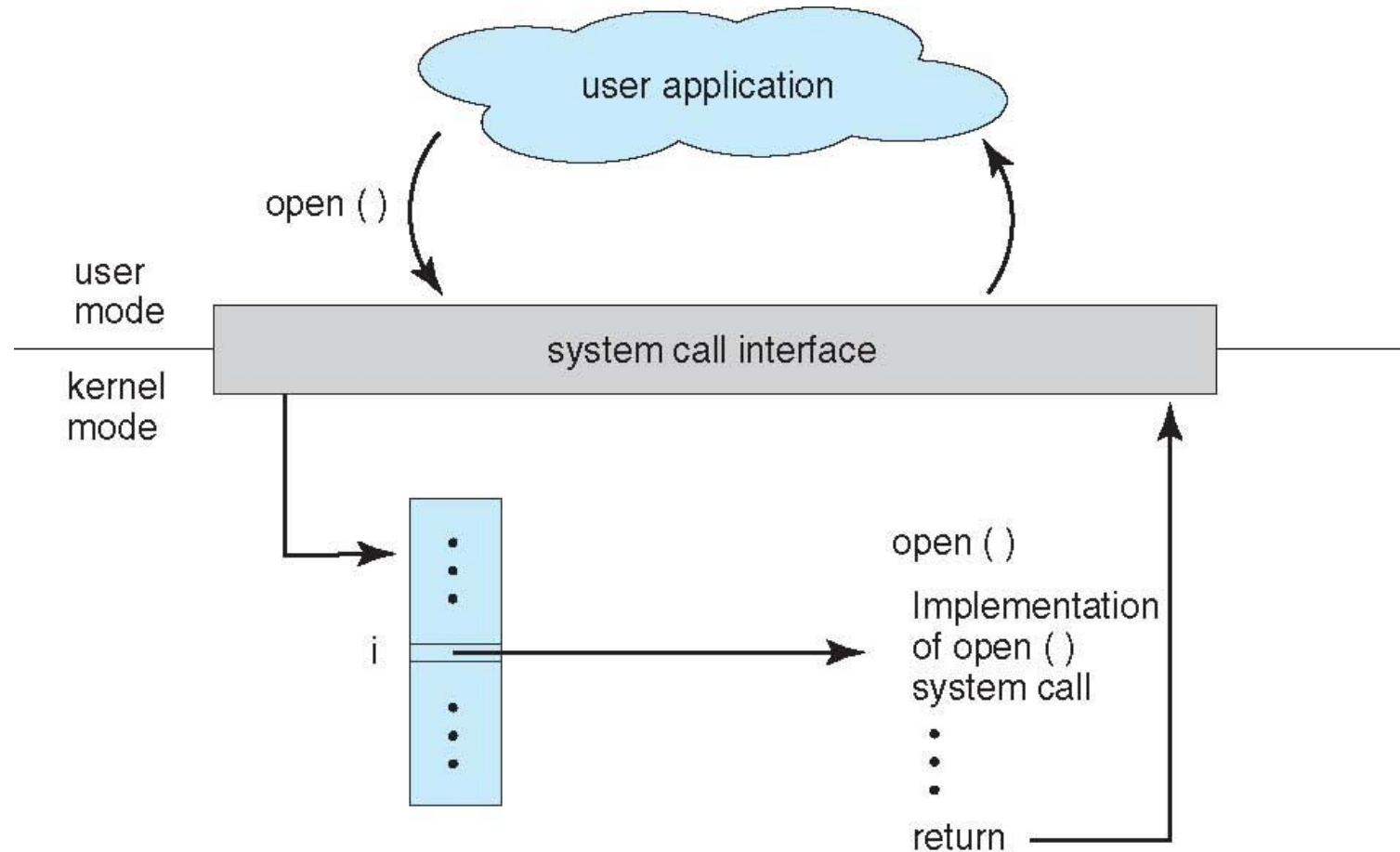


# System Calls (2 / 2)

- ▶ Triggering a System Call
  - Use a special instruction supported by the hardware
    - For Intel x86, it is “int 0x80”
  - Provide the type and parameters of the system call
- ▶ Parameter Passing
  - Registers
  - Stacks
  - Registers pointing to blocks

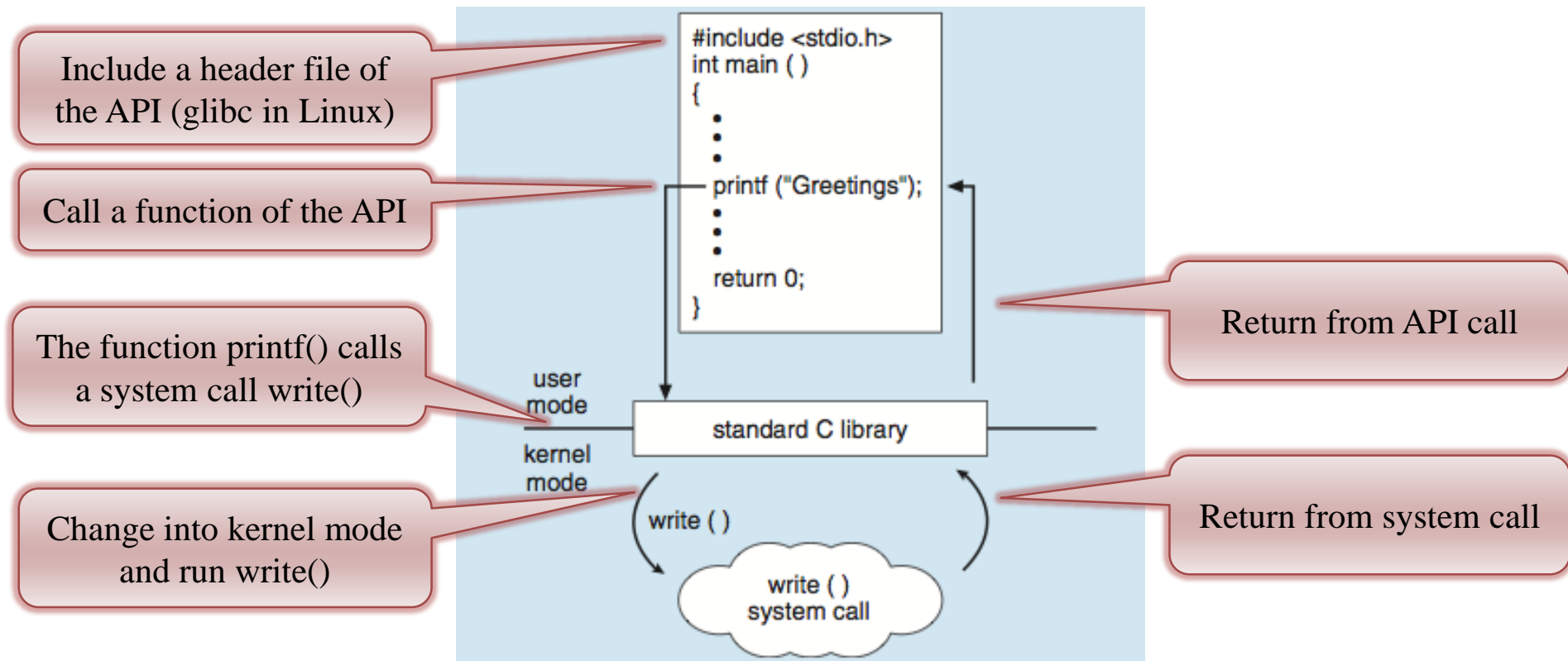


# Relationship of System Call and OS



# API, System Call and OS

- ▶ A C program can invoke printf() in the library (API)
- ▶ In the API implementation, printf() calls write() system call



# Types of System Calls

- ▶ Process Control
- ▶ File Management
- ▶ Device Management
- ▶ Information Maintenance
- ▶ Communications
- ▶ Protection





# System Calls— Process Control

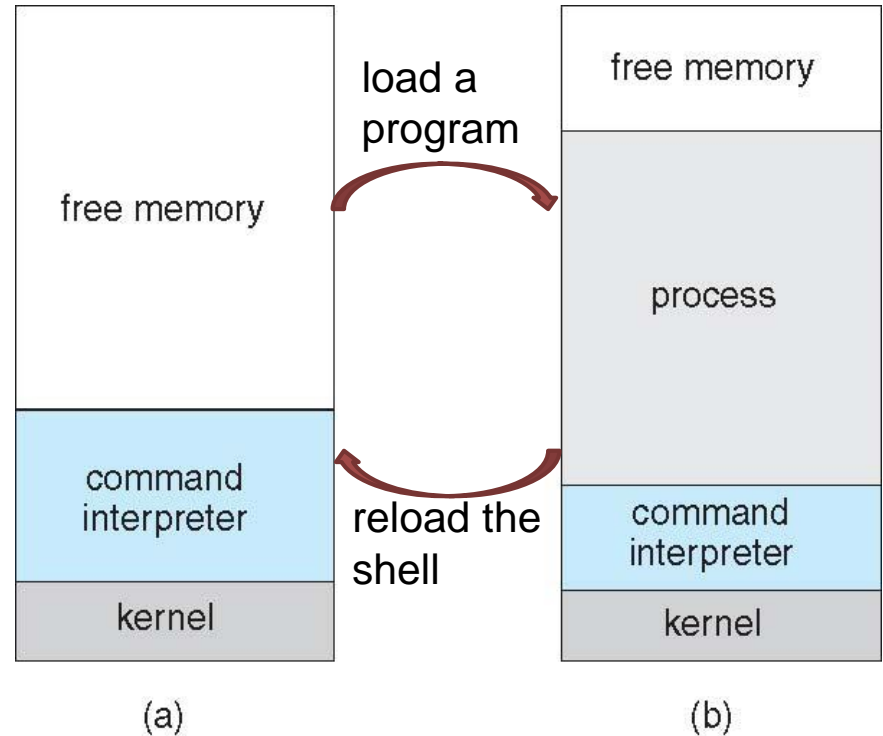
## (1 / 3)

- ▶ Load and execute
  - Have to return the control
- ▶ End (normal exit) or abort (abnormal)
  - Error level or no
  - Interactive, batch, GUI-supported systems
- ▶ Creation and/or termination of other processes
  - To support the techniques of multiprogramming and timesharing mentioned in Chapter 1
- ▶ Get process attributes, set process attributes
- ▶ Wait for time, wait event, signal event
- ▶ Allocate and free memory



# System Calls— Process Control (2/3)

- ▶ Example: MS-DOS
  - Single-tasking
  - Shell is invoked when system is booted
  - Single memory space
  - Loads program into memory, overwriting all but the kernel
  - Program exit → shell reloaded



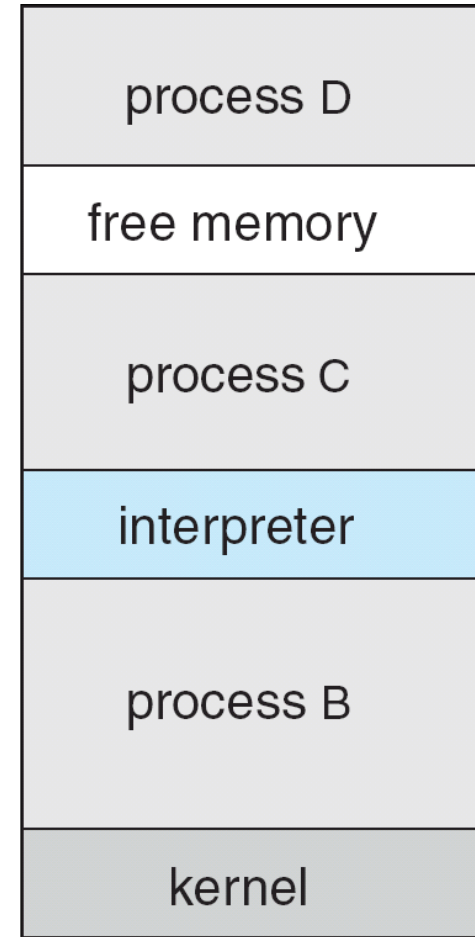
(a) At system startup (b) running a program

# System Calls— Process Control

## (3 / 3)

### ► Example: FreeBSD

- Multitasking
- OS invokes user's choice of shell
- Shell executes `fork()` system call to create process
- OS loads program into process
- Shell waits for process to terminate or continues with user commands
- Process exits with return code
  - with code of 0 → no error
  - with code > 0 → error code



# System Calls— File Management

- ▶ Create and delete
- ▶ Open and close
- ▶ Read, write, and reposition (e.g., rewinding)
- ▶ Get or set attributes of files
- ▶ Operations for directories



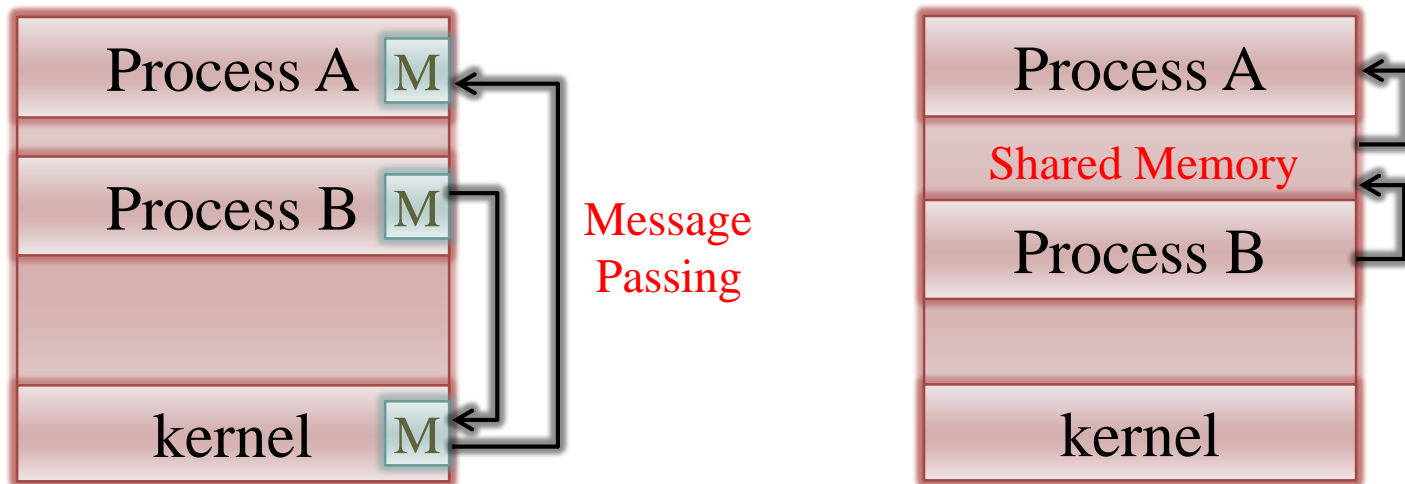
# System Calls— Device Management

- ▶ Request device, release device
- ▶ Read, write, reposition
- ▶ Get device attributes, set device attributes
- ▶ Logically attach or detach devices



# System Calls— Communications

- ▶ Message Passing
  - Open, close, accept connections
  - No access conflict and easy implementation
- ▶ Shared Memory
  - Memory mapping and process synchronization
  - Short latency and high throughput



# System Calls— Information Maintenance and Protection

- ▶ Information Maintenance
  - Get time or date, set time or date
  - Get system data, set system data
- ▶ Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access





# System Programs

## ▶ Goal:

- Provide a convenient environment for program development and execution

## ▶ Types

- File Management, e.g., rm
- Status information, e.g., date
- File Modifications, e.g., editors
- Program Loading and Executions, e.g., loader
- Programming Language Supports, e.g., compilers
- Communications, e.g., telnet





# Policies and Approaches of OS Implementation

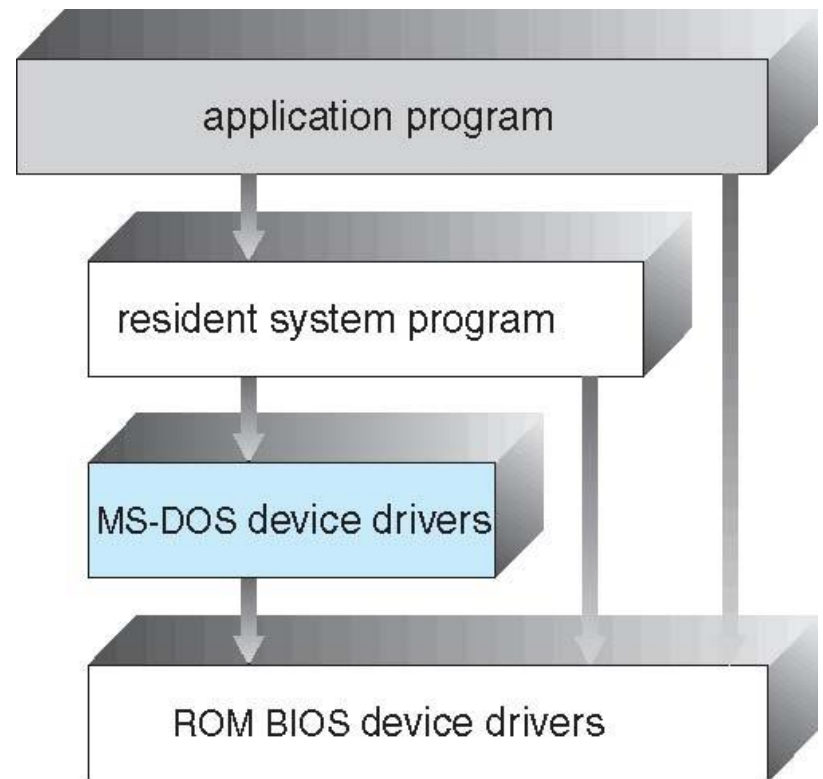
# Operating System Design and Implementation

- ▶ Design Goals and Specifications
  - User goals: ease of use, short latency
  - System goals: reliable, high utilization
- ▶ Separation of Policy and Mechanism
  - **Policy: What will be done**
  - **Mechanism : How to do things**
- ▶ OS Implementation in High-Level Languages
  - Advantages:
    - Being easy to understand and debug
    - Being written fast, more compact, and portable
  - Disadvantages:
    - Less efficient
    - Larger size



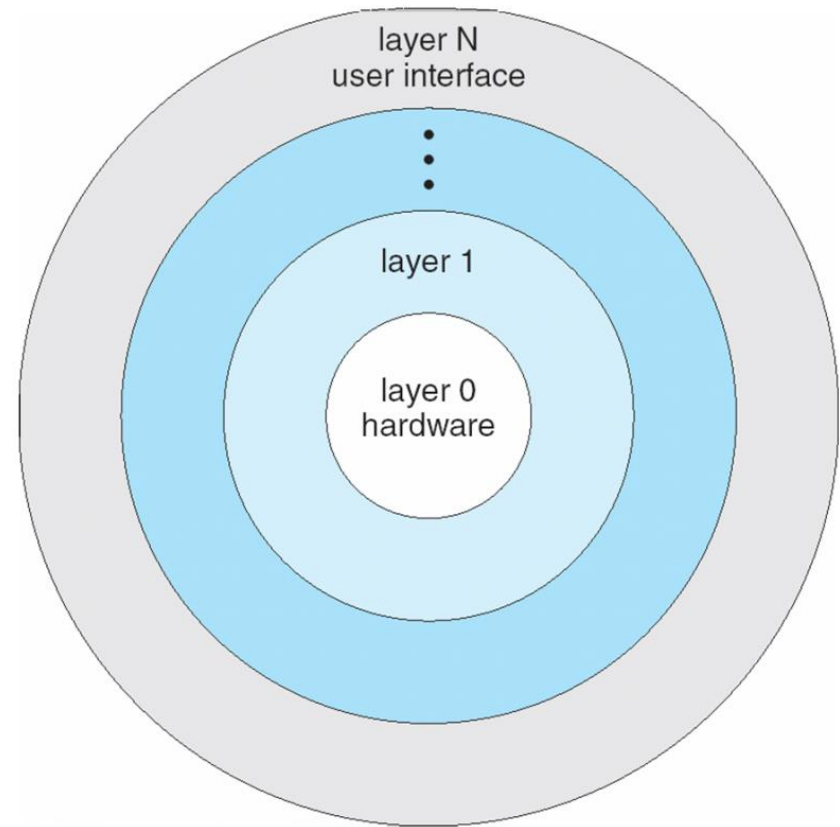
# Operating System Structure— MS-DOS

- ▶ Not divided into modules
- ▶ Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated



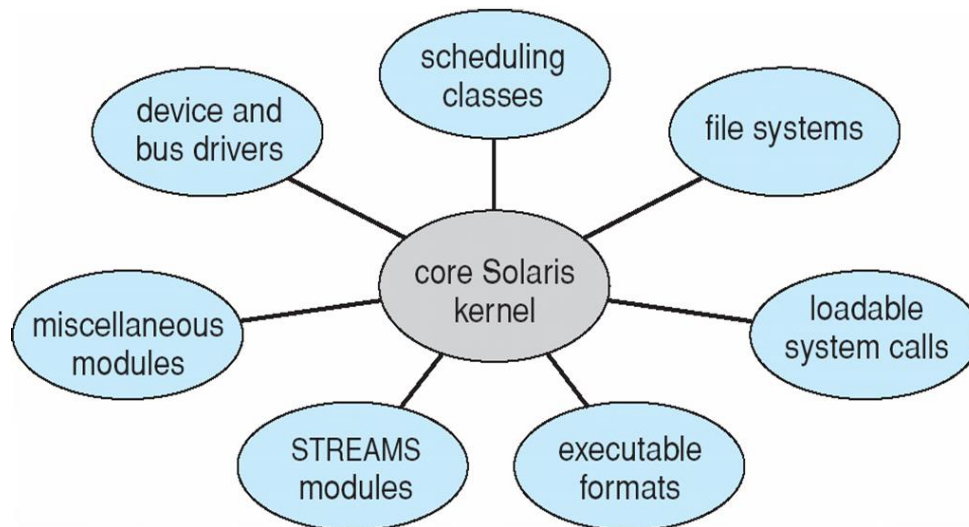
# Operating System Structure— Layered Approach

- ▶ Advantage: Modularity  
→ Debugging & Verification
- ▶ Difficulty: Appropriate layer definitions, less efficiency due to overheads
- ▶ A Layer Definition Example:
  - L5 User programs
  - L4 I/O buffering
  - L3 Operator-console device driver
  - L2 Memory management
  - L1 CPU scheduling
  - L0 Hardware



# OS Structure— Modules

- ▶ Most modern operating systems implement loadable kernel modules
  - Uses object-oriented approach
  - Each core component is separate
- ▶ Solaris Modular Approach



# OS Structure— Microkernels

- ▶ The concept of microkernels was proposed in CMU in mid 1980s (Mach)
  - Moving all nonessential components from the kernel to the user or system programs
- ▶ Benefits
  - Ease of OS service extensions → portability, reliability, security
- ▶ Examples
  - Tru64 UNIX (Mach kernel), MacOS X (Darwin kernel), L4 Microkernel

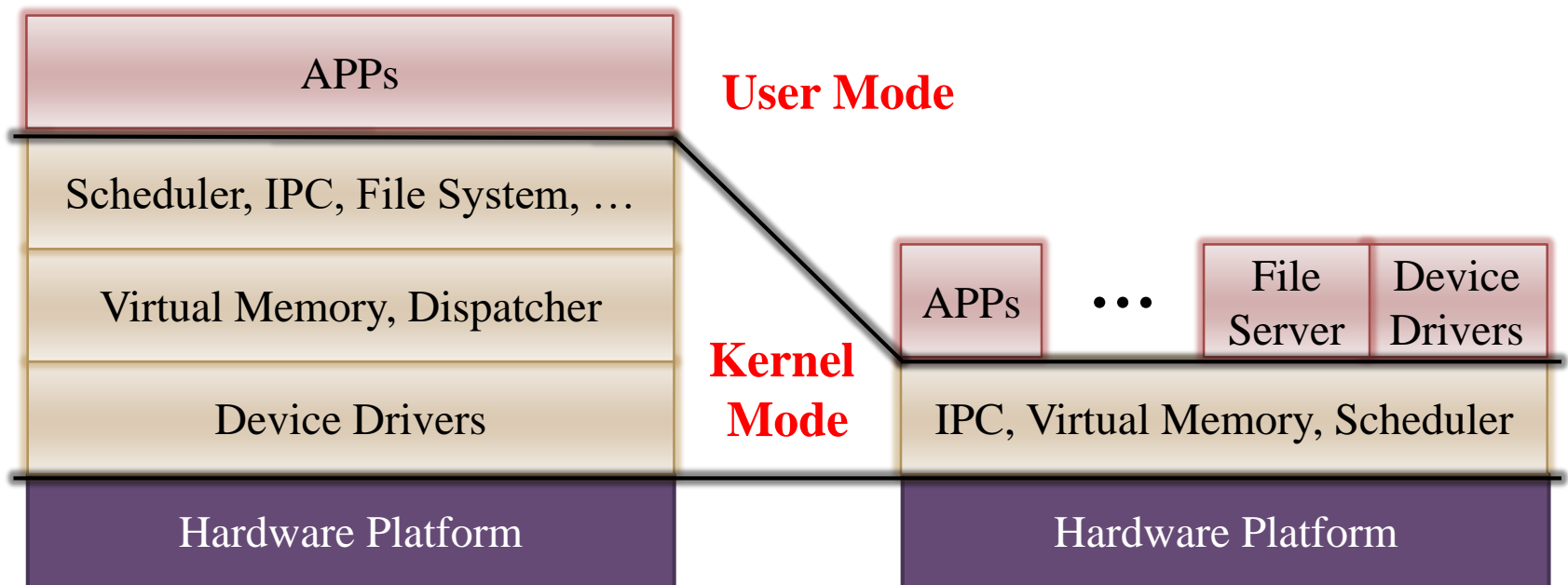




# Monolithic Kernel and Microkernel

## Monolithic Kernel

## Microkernel



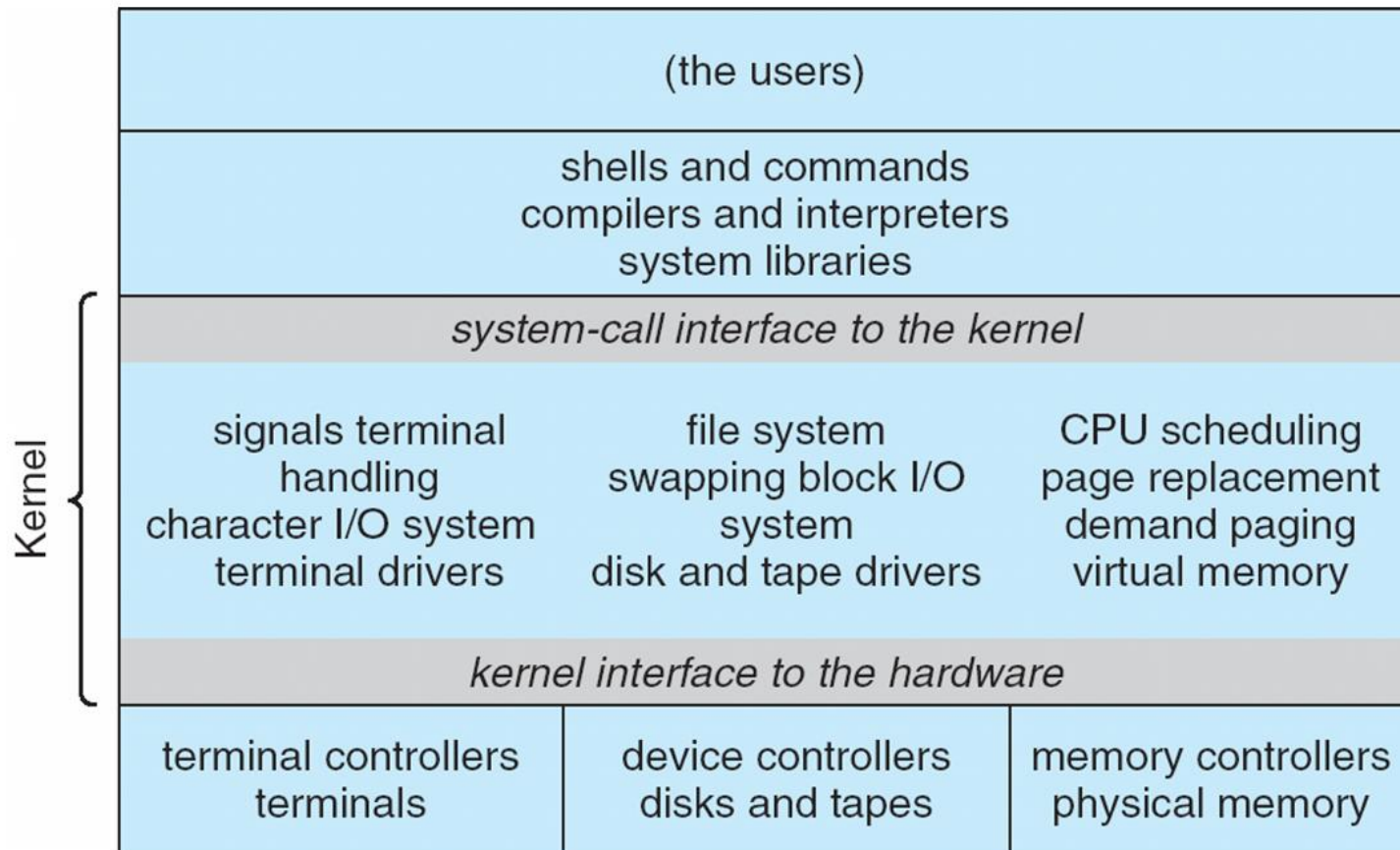
# Hybrid Systems

- ▶ Most modern operating systems actually use more than one model for their implementations
- ▶ Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem personalities
  - Apple Mac OS X is based on a microkernel and also hybrid, layered, Aqua UI plus Cocoa programming environment

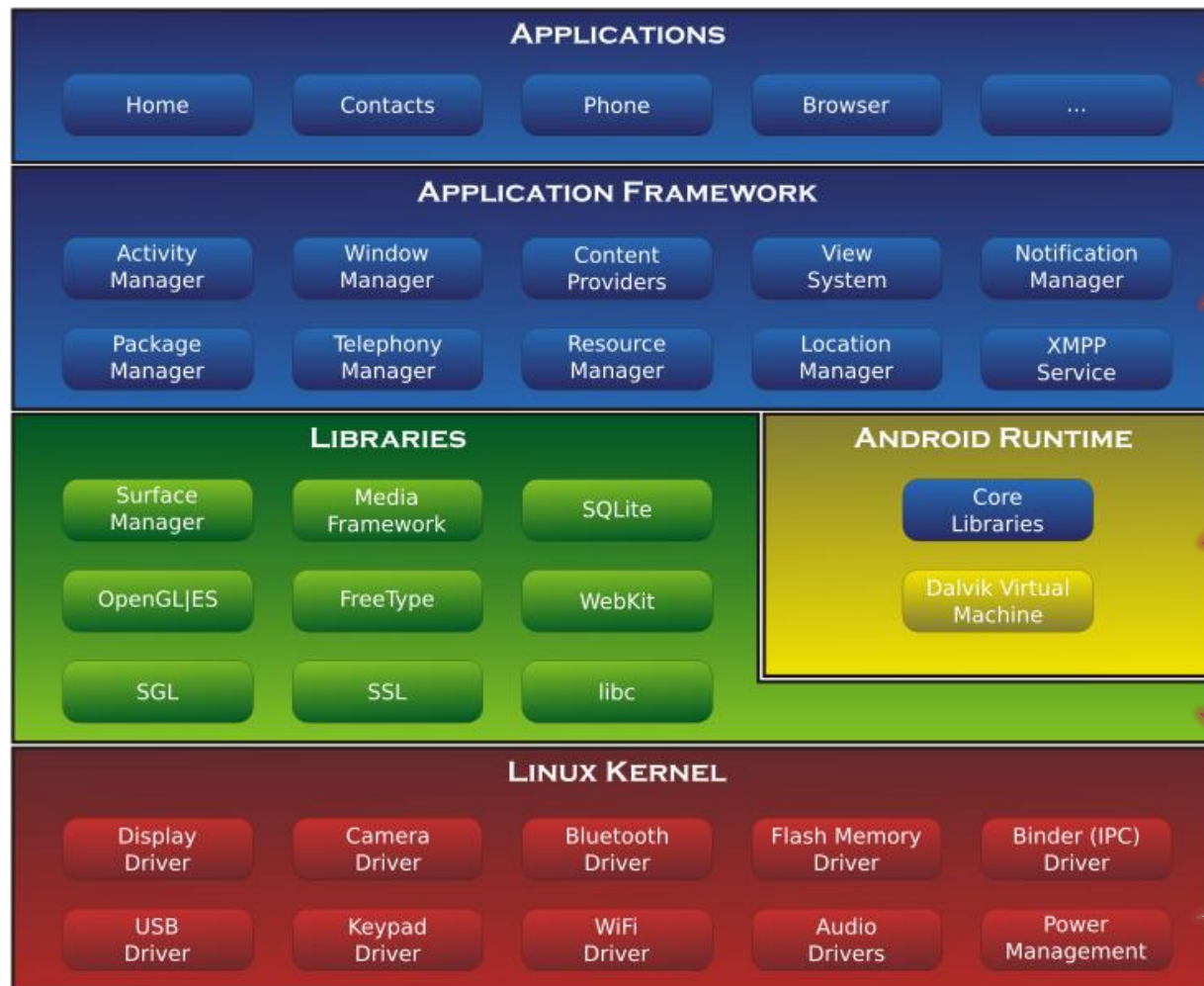


# Traditional UNIX System Structure

- Beyond simple but not fully layered



# OS Structure— Android



Include a Java-like API for user applications

Define frequently used function packages

Run each program on a Dalvik virtual machine

Provide useful libraries

Use the Linux kernel

Source: [http://en.wikipedia.org/wiki/Android\\_\(operating\\_system\)](http://en.wikipedia.org/wiki/Android_(operating_system))



# OS Structure— iOS

- ▶ Apple mobile OS for iPhone, iPad
  - Structured on Mac OS X, added functionality
  - Also runs on different CPU architecture (ARM vs. Intel)
  - Media services layer for graphics, audio, video
  - Cocoa Touch Objective-C and Swift APIs for developing apps



**Figure 2.17** Architecture of Apple's iOS.

# Operating System Debugging

## ▶ Debugging

- An activity in finding and fixing errors or bugs, including performance problem, that exist in hardware or software

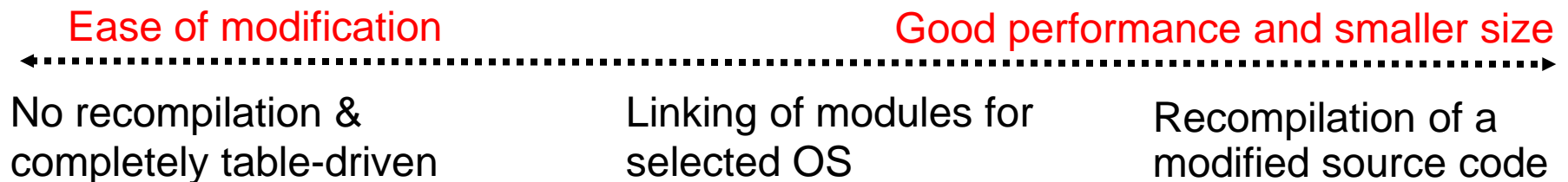
## ▶ Terminologies

- Profiling— A procedure to understand the statistical trends
- Performance tuning— A procedure that seeks to improve performance by removing bottlenecks
- Crash— A kernel failure
- Core dump— A capture of the memory of a process or OS



# Operating System Generation

- ▶ Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site
- ▶ SYSGEN program obtains information concerning the specific configuration of the hardware system
  - Used to build system-specific compiled kernel
  - Can generate more efficient code than one general kernel





# System Boot

- ▶ When power is initialized on a system, execution starts at a fixed memory location
  - Firmware ROM is used to hold initial boot code
- ▶ Operating systems must be made available to hardware so hardware can start it
  - Small piece of code— bootstrap loader, stored in ROM or EEPROM locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where boot block at fixed location loaded by ROM code, which loads bootstrap loader from disk
- ▶ Common bootstrap loader, GRUB, allows selection of kernel from multiple disks, versions, kernel options

