Operating-System Structures

Chapter 2

Operating System Services

- One set provides functions that are helpful to the user:
  - User interface
  - Program execution
  - I/O operations
  - File-system manipulation
  - Communications
  - Error detection

Operating System Services (Cont.)

- Another set exists for ensuring the efficient operation of the system itself via resource sharing
  - **Resource allocation**
  - **Accounting**
  - **Protection and security**

System Programs

- Provide a convenient environment for program development and execution
- Most users' view of the operation system is defined by system programs, not the actual system calls
- File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- Status information

System Programs (cont.)

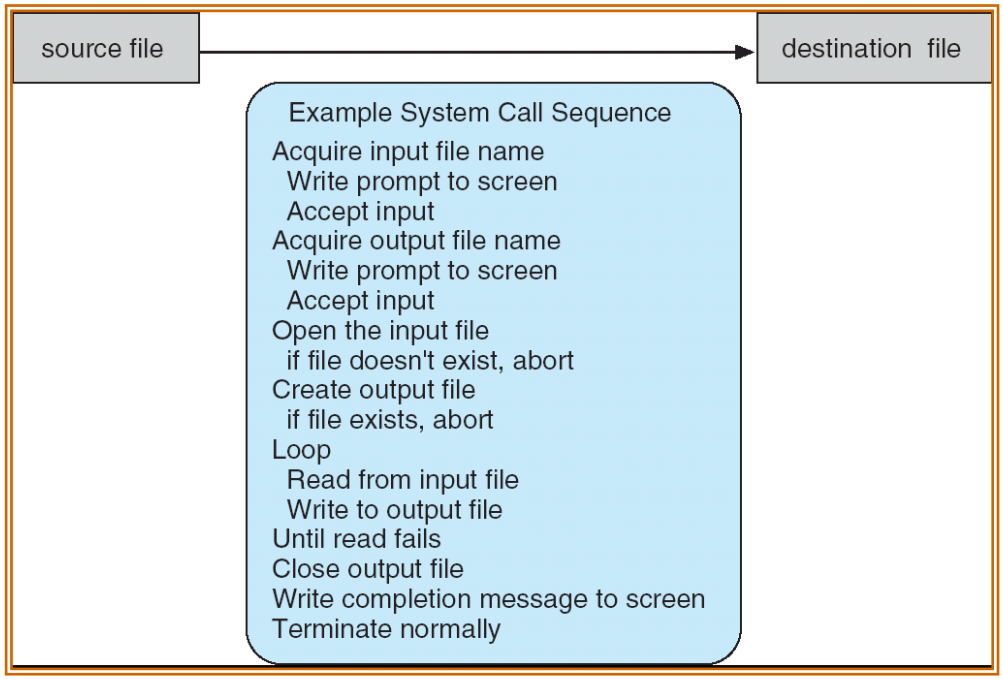
- File modification
- Programming-language support
- Program loading and Communications

System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
- Win32, POSIX, and Java API
- Why use APIs rather than system calls?

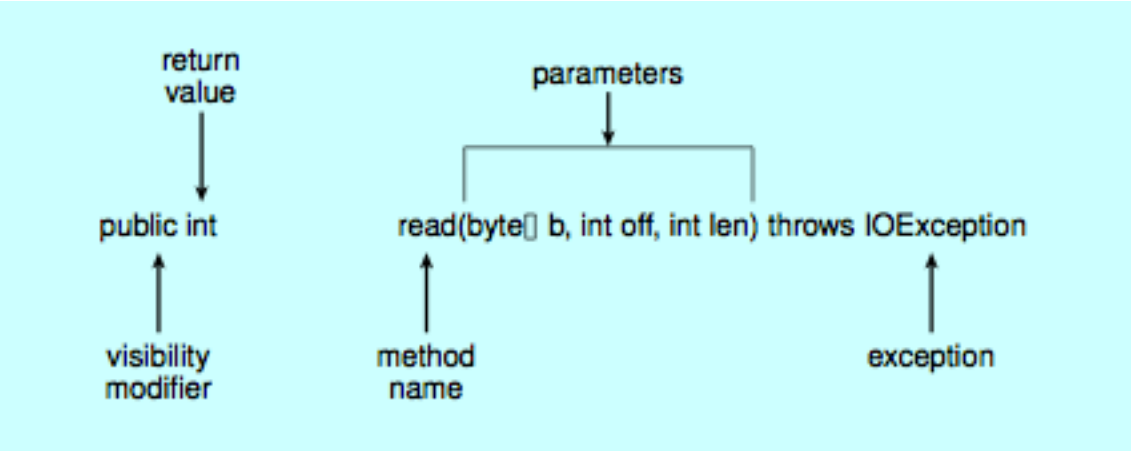
Example of System Calls

- System call sequence to copy the contents of one file to another file



Example of Standard API

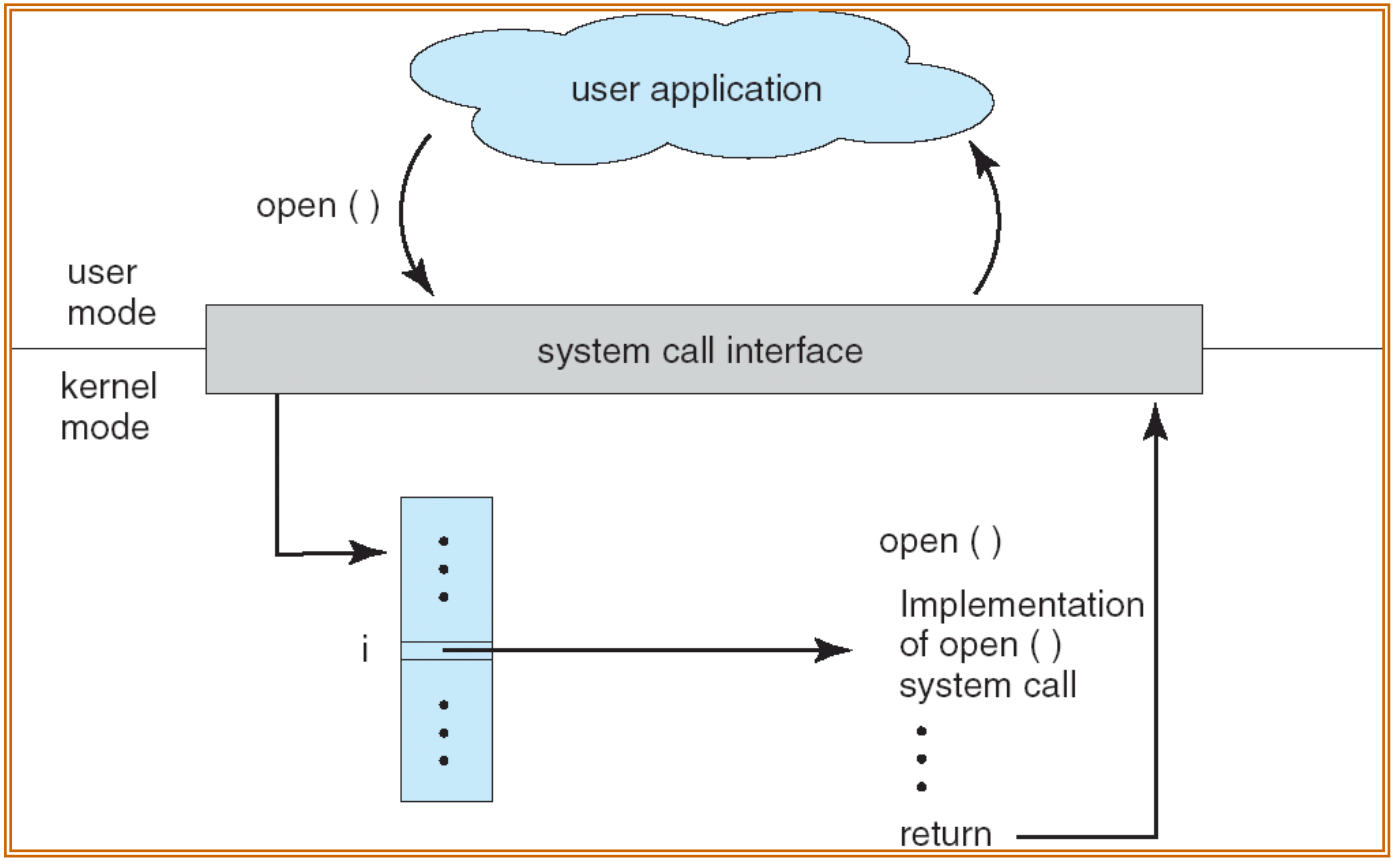
- Consider the the Java read()



System Call Implementation

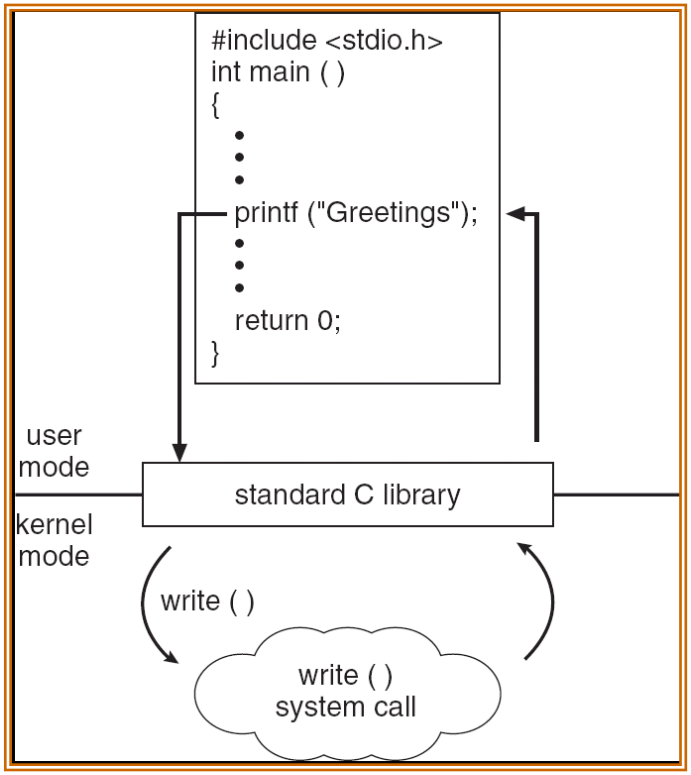
- Typically, a number associated with each system call
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented

API – System Call – OS Relationship



Standard C Library Example

- C program invoking printf() library call, which calls write() system call



System Call Parameter Passing

- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in ***registers***
  - Parameters stored in a ***block***, or **table**, in memory, and address of block passed as a parameter in a register (Linux & Solaris)
  - Parameters placed, or *pushed,* onto the ***stack*** by the program and *popped* off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications

Operating System Design and Implementation

- some approaches have proven successful
- Internal structure can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware, type of system

Operating System Design and Implementation (Cont.)

- *User* goals and *System* goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement and maintain, as well as flexible, reliable, error-free, and efficient

Operating System Design and Implementation (Cont.)

- Important principles to separate

  Policy: What will be done?
  Mechanism: How to do it?

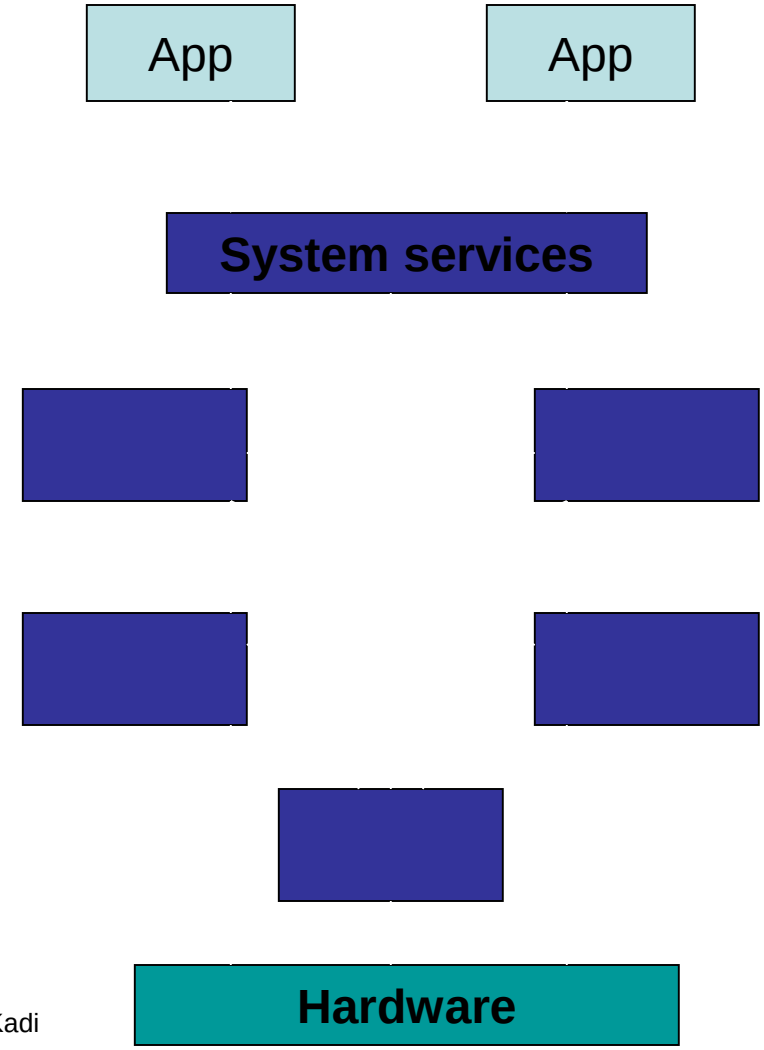
- Mechanisms determine how to do something, policies decide what will be done
  - The separation of policy from mechanism allows maximum flexibility if policy decisions are to be changed later

Simple Structure (no structure)

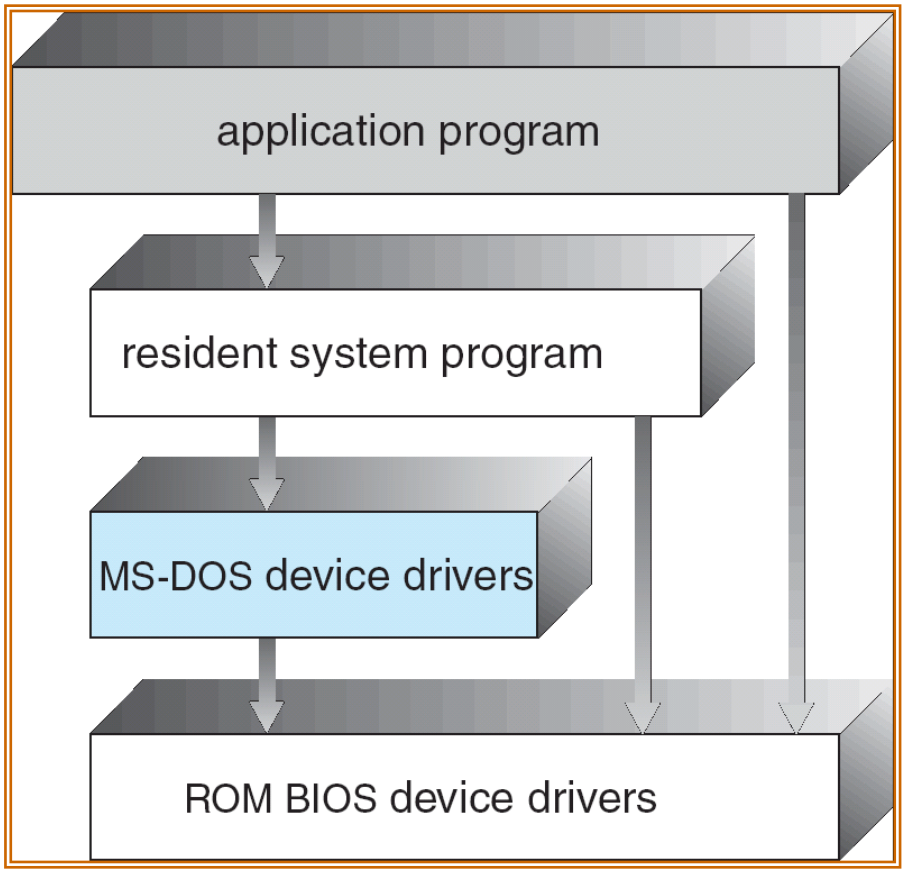
- **Monolithical** systems
- Unstructured
- Supervisor call changes

  from user mode into

  kernel mode

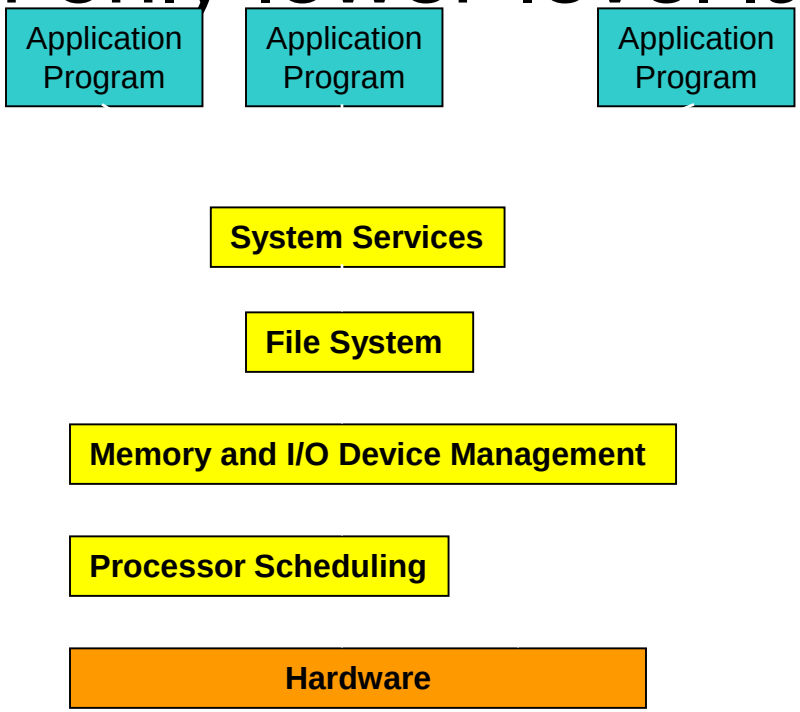


MS-DOS Structure

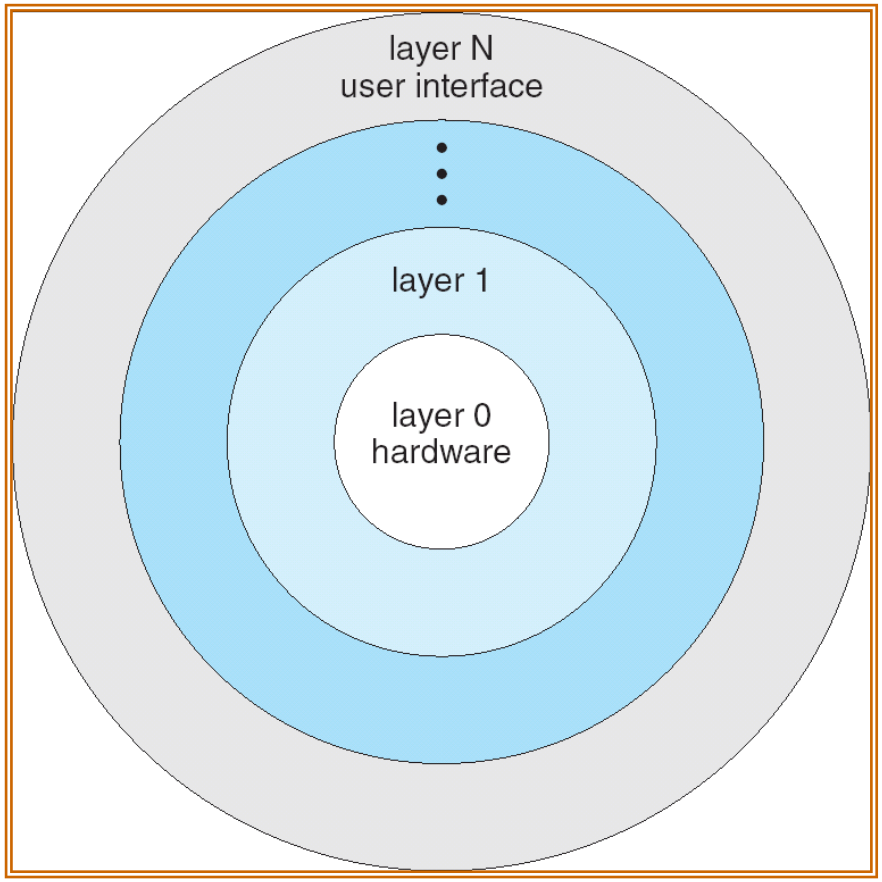


Layered Approach

- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



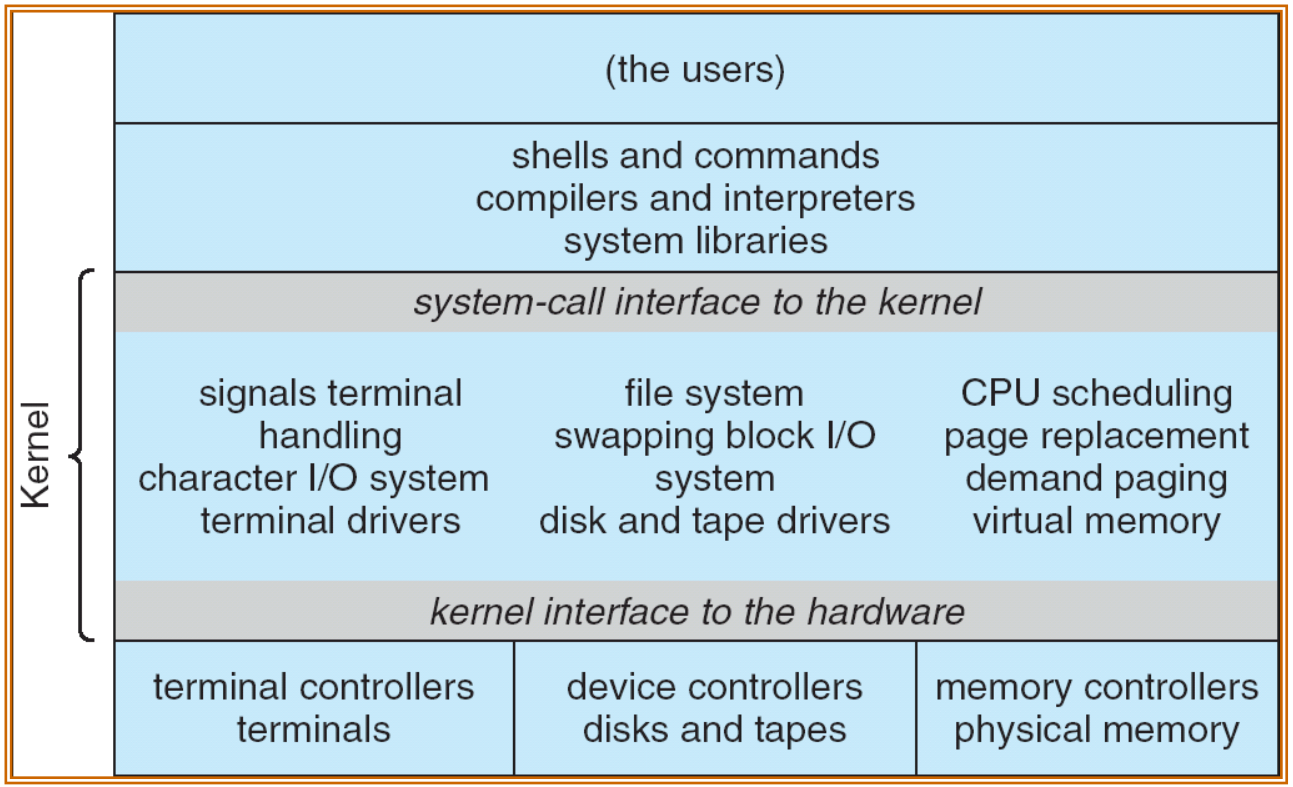
Layered Operating System



Structure of the THE operating system

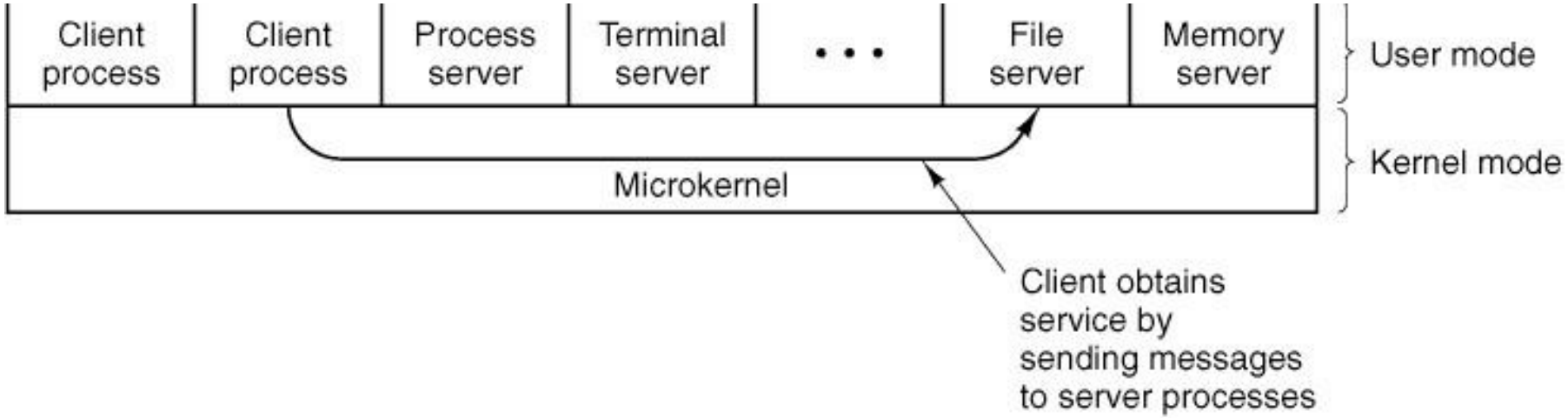
Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

UNIX System Structure



Microkernel System Structure

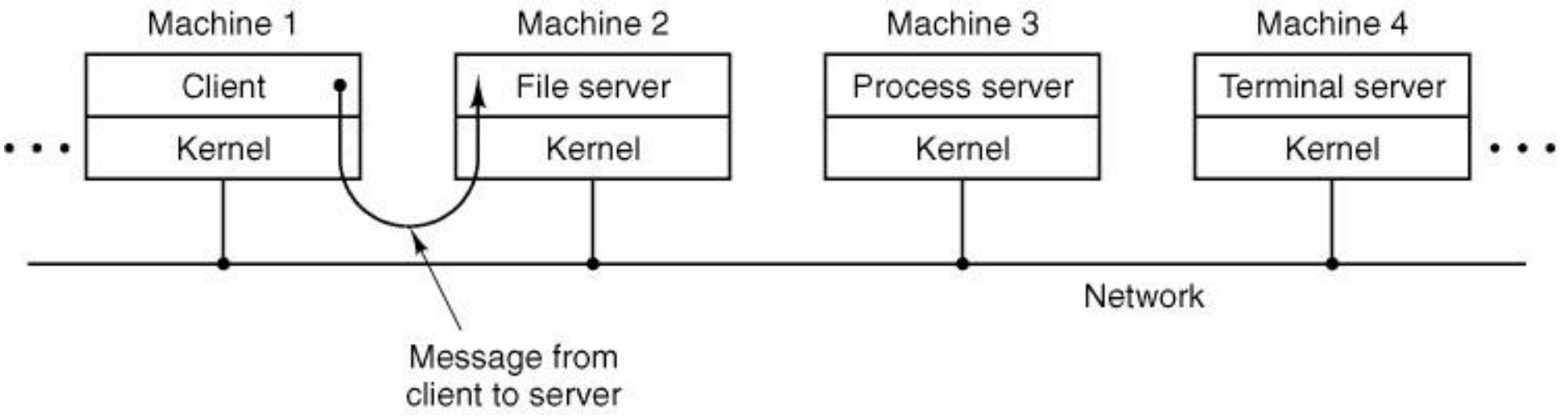
- Moves as much from the kernel into "*user*" space
- Communication takes place between user modules using message passing



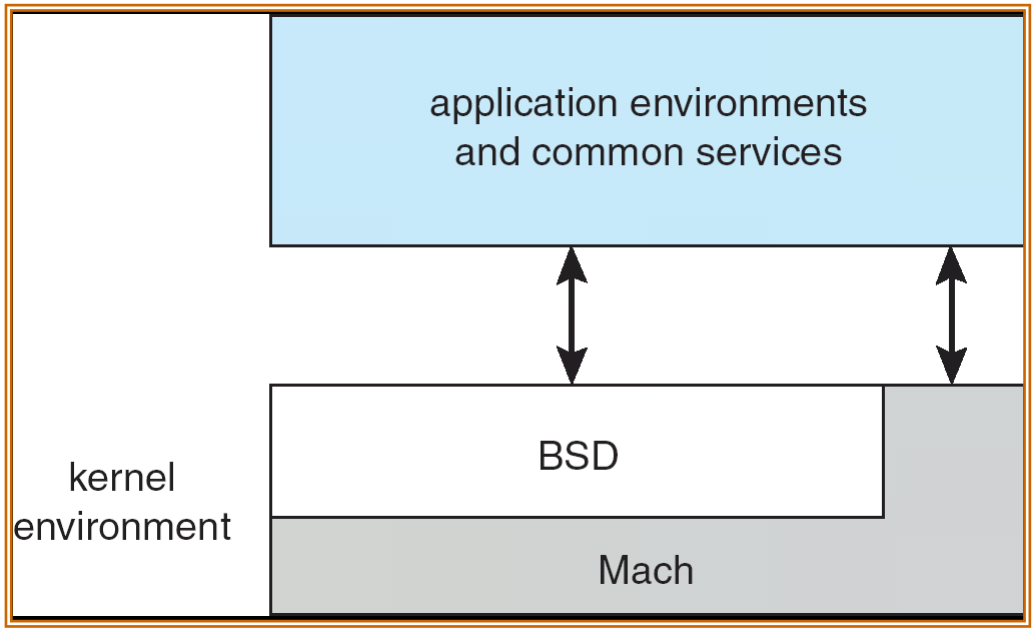
Microkernel System Structure (cont.)

- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Determents:
  - Performance overhead of user space to kernel space communication

Microkernel System Structure (cont.)



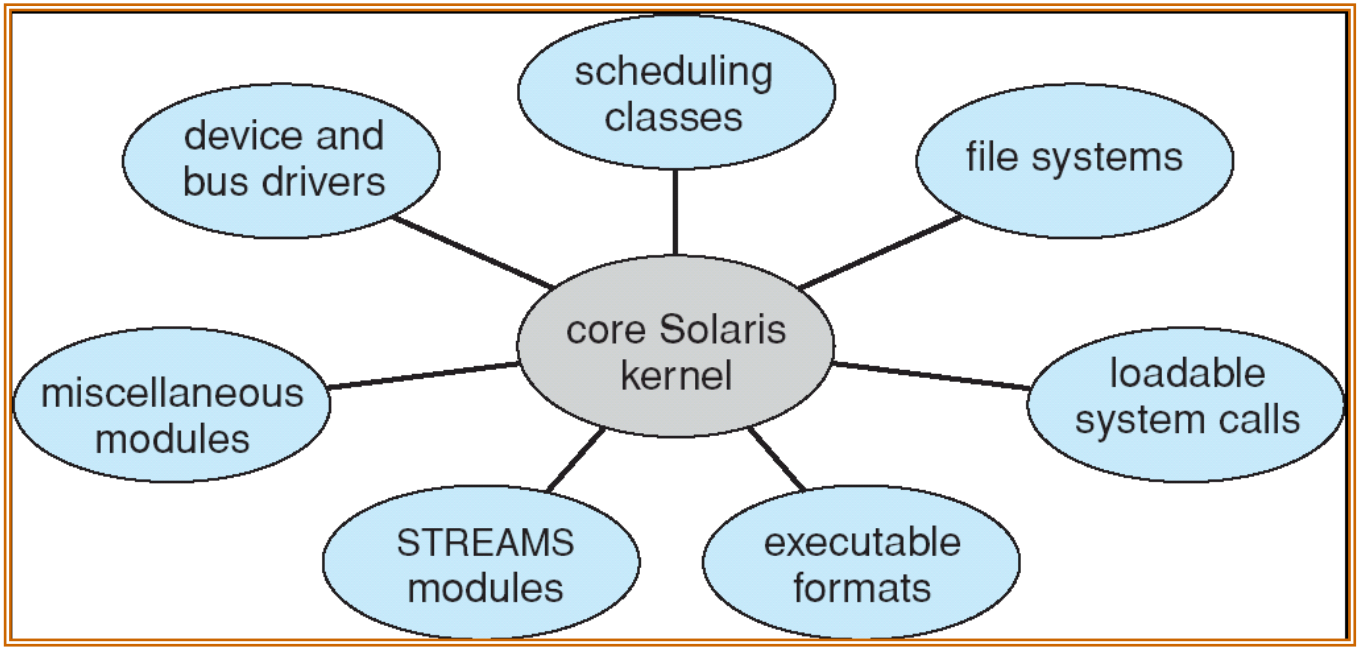
Mac OS X Structure



Modules

- Most modern operating systems implement kernel modules
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible

Solaris Modular Approach

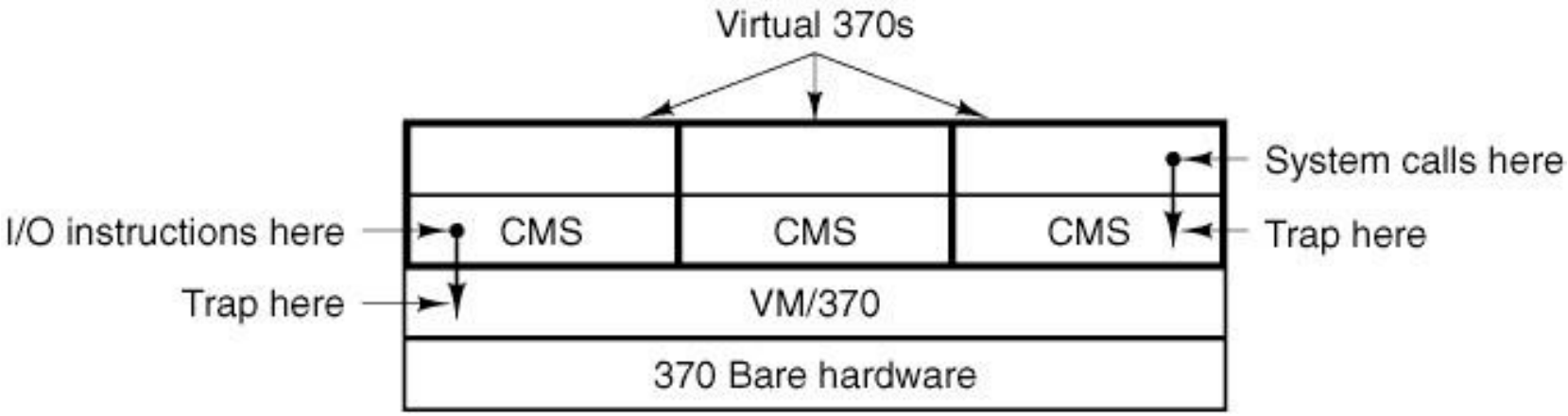


Virtual Machines

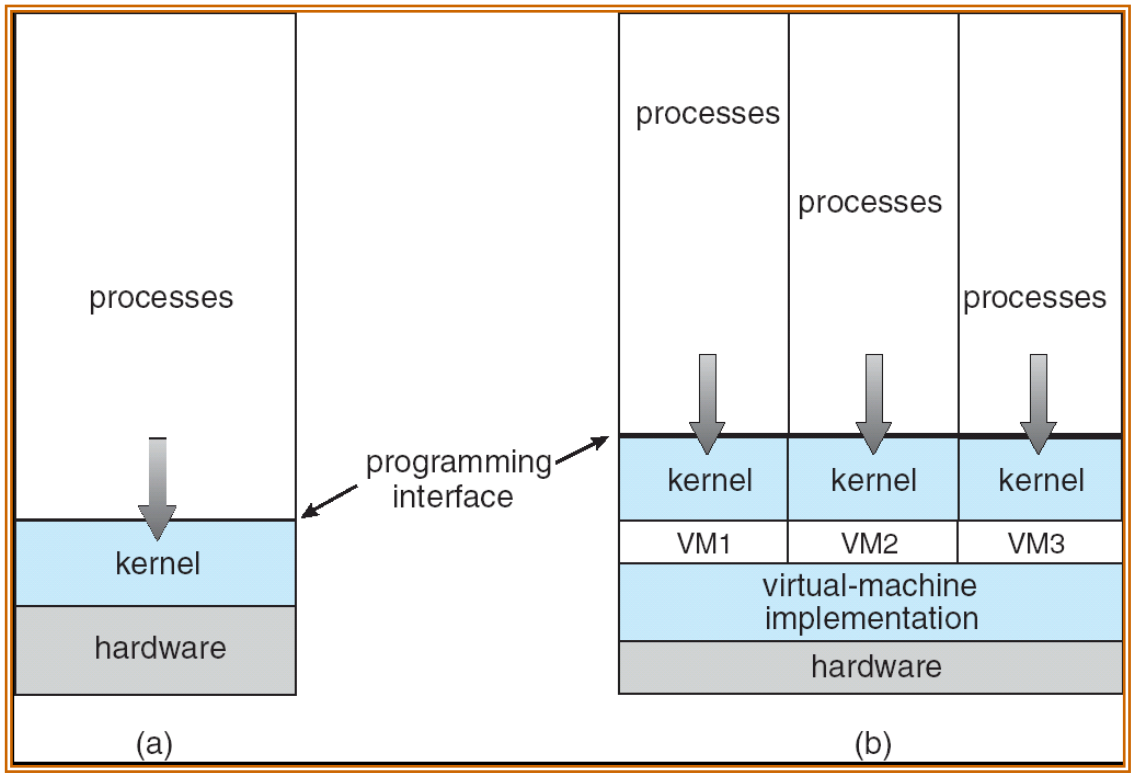
- A *virtual machine* treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface *identical* to the underlying bare hardware

Virtual Machines (Cont.)

- The operating system creates the illusion of multiple environments, each executing on its own processor with its own (virtual) memory



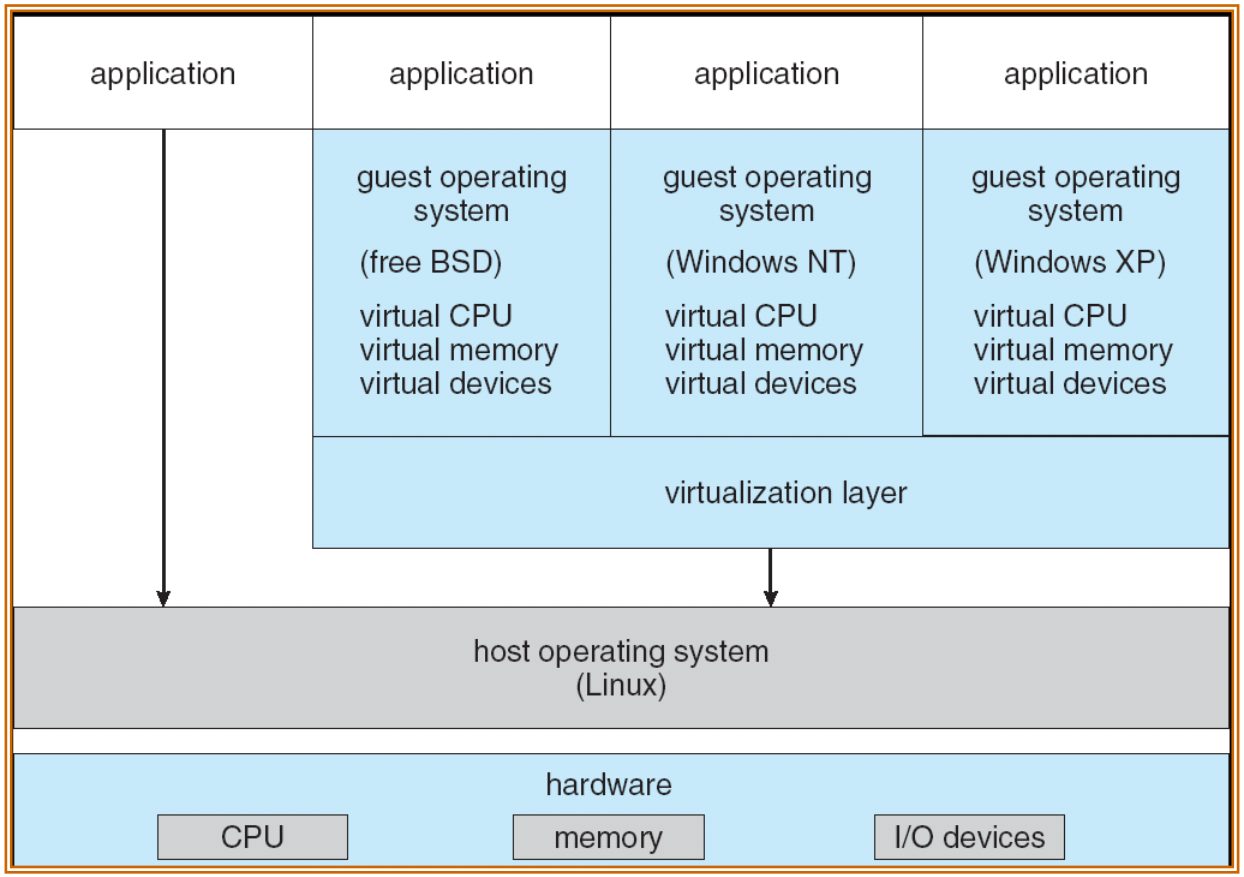
Virtual Machines (Cont.)



Virtual Machines (Cont.)

- provides complete protection of system resources. This isolation, however, permits no direct sharing of resources
- perfect vehicle for operating-systems research and development
- difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine

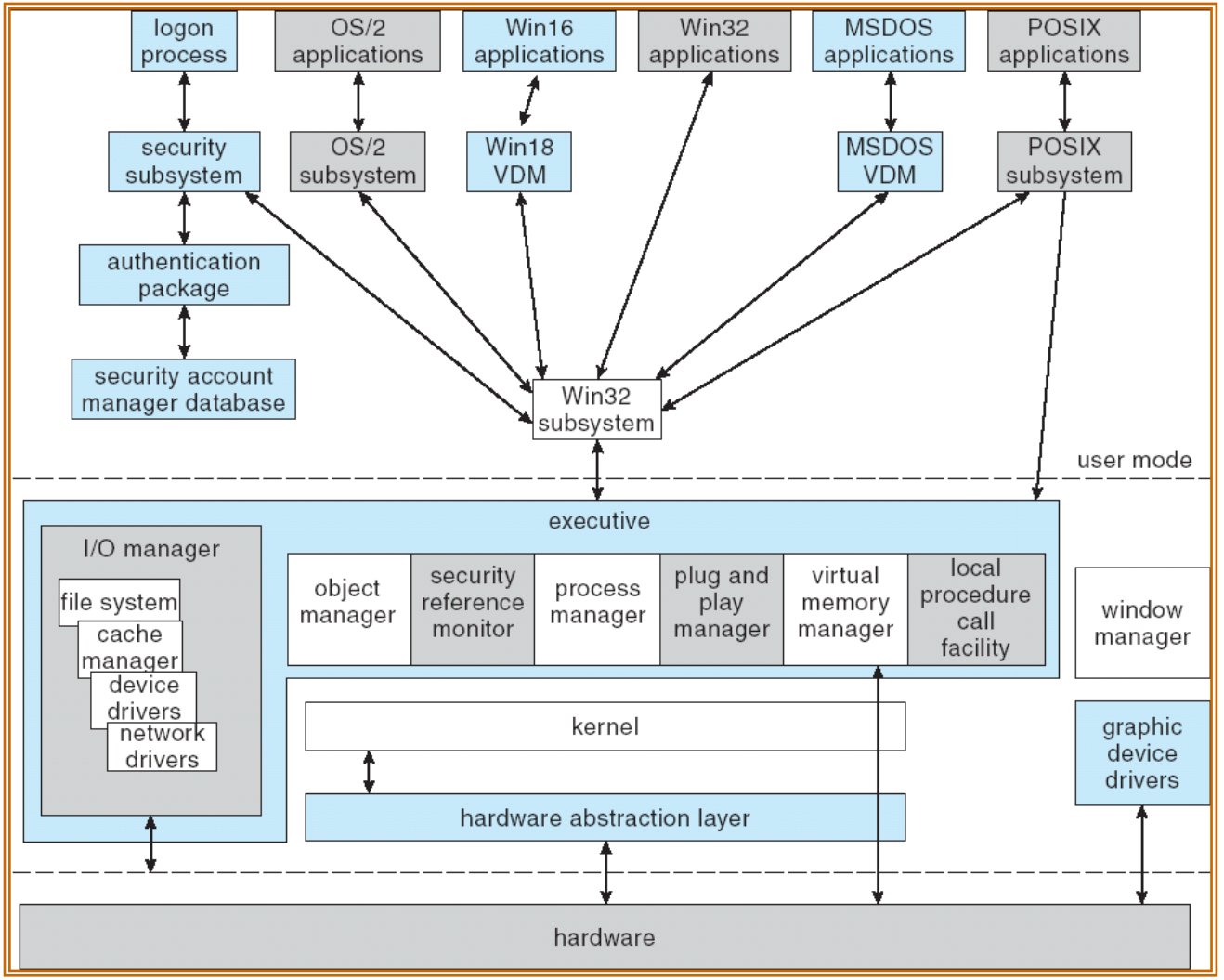
VMware Architecture



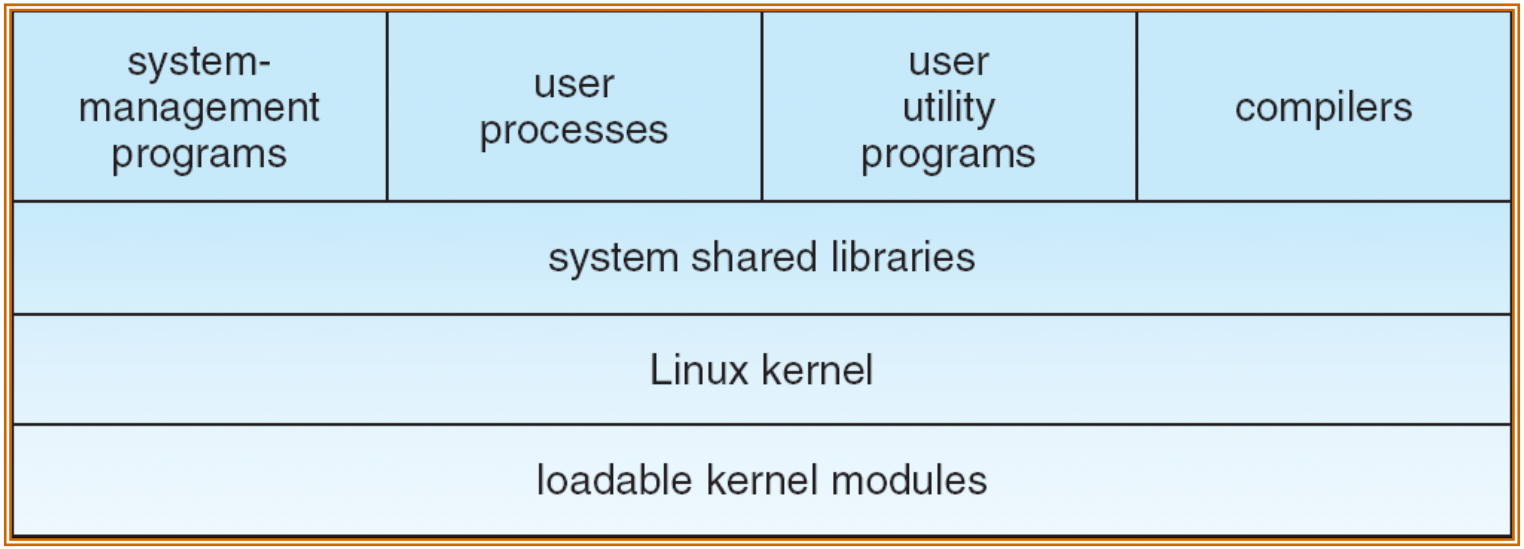
Hybrid approaches

- Windows xp
- linux

Windows xp



Linux



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
- ***Booting*** – starting a computer by loading the kernel
- ***Bootstrap program*** – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution

System Boot

- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loads bootstrap loader
  - When power initialized on system, execution starts at a fixed memory location
    - Firmware used to hold initial boot code