UNIT 1: OVERVIEW OF OPERATING SYSTEM

DEFINITION

- An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer program

- An Operating System (OS) is an interface between a computer user and computer hardware.

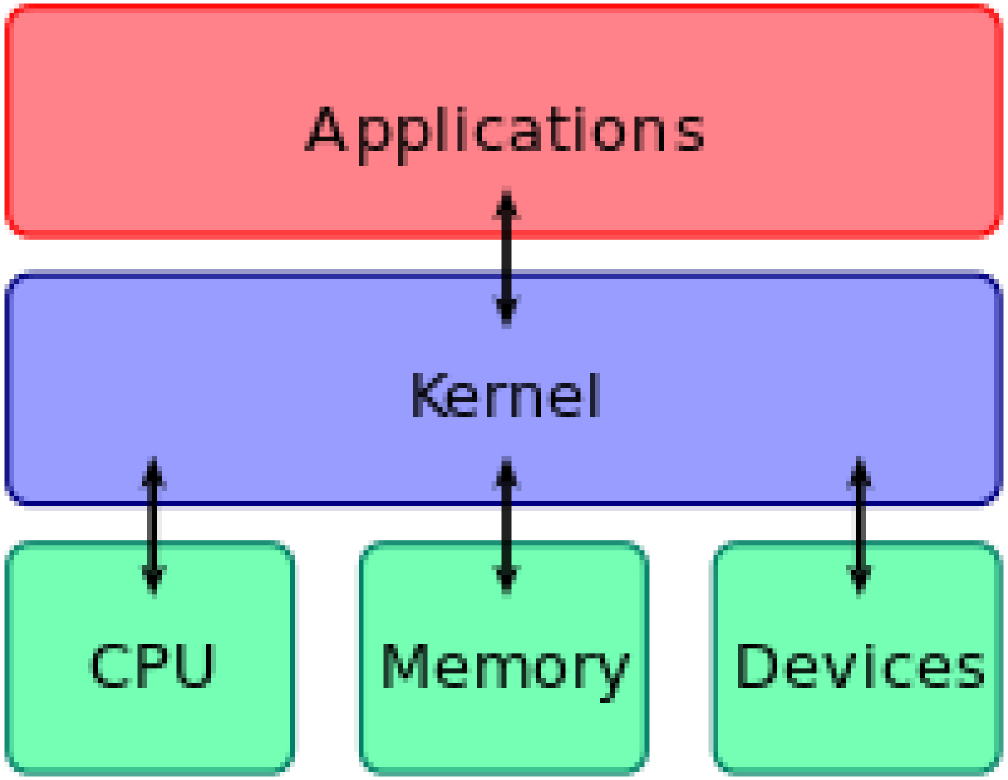
Functions the same way as ordinary computer software

- It is a program that is executed, but with extra privileges

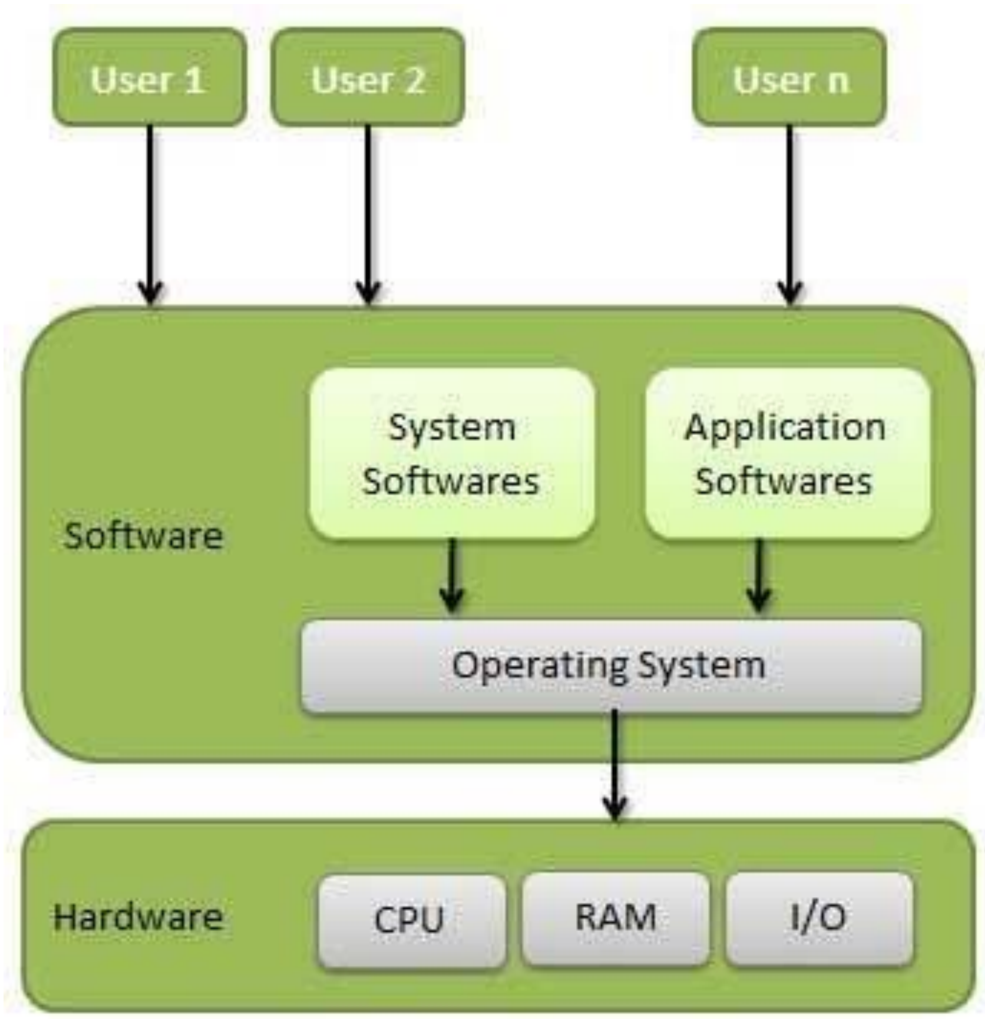
Kernel: Portion of operating system that is in main memory

- Contains most frequently used functions
- Also called the nucleus

- The kernel is the central module of an operating system (OS).
- It is the part of the operating system that loads first, and it remains in main memory.
- Because it stays in memory, it is important for the kernel to be as small as possible while still providing all the essential services required by other parts of the operating system and applications.
- The kernel code is usually loaded into a protected area of memory to prevent it from being overwritten by programs or other parts of the operating system.
- Typically, the kernel is responsible for memory management, process and task management, and disk management.
- The kernel connects the system hardware to the application software. Every operating system has a kernel.


Applications
Kernel
CPU
Memory
Devices

OPERATING SYSTEM STRUCTURE



OPERATING SYSTEM OPERATIONS

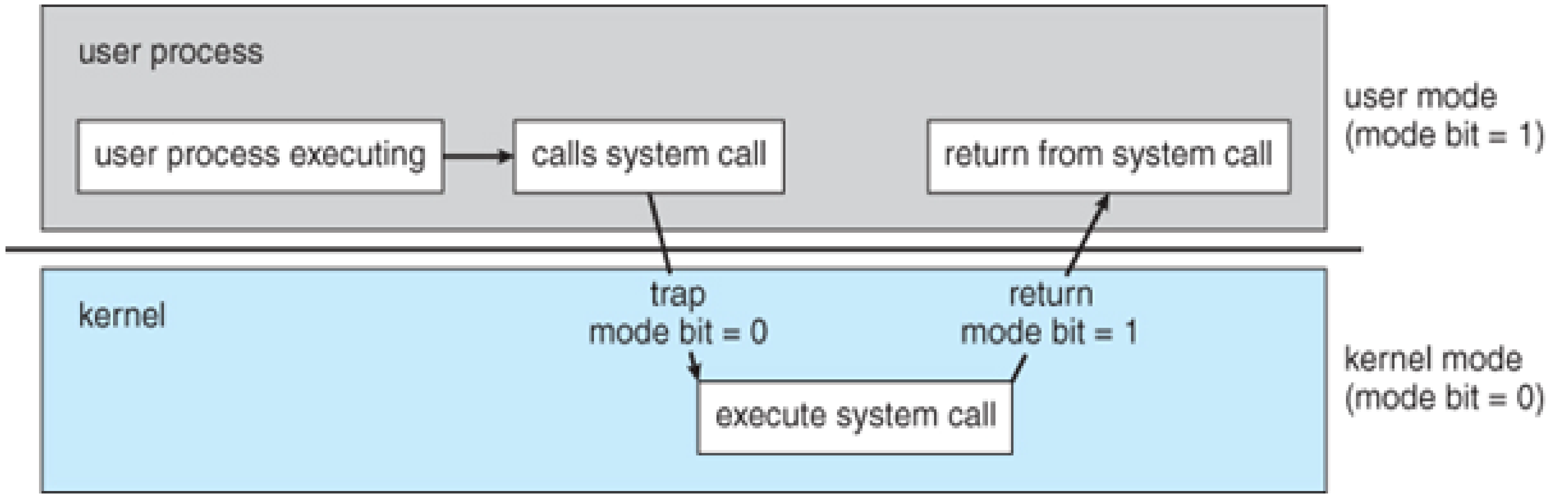
- Modern operating systems are interrupt driven.
- If there are no processes to execute, OS will sit idle and wait for some event to happen.
- Interrupts could be hardware interrupts or software interrupts. The OS is designed to handle both.
- A trap (or an exception) is a software generated interrupt caused either by an error (e.g. divide by zero) or by a specific request from a user program.
- A separate code segment is written in the OS to handle different types of interrupts. These codes are known as interrupt handlers/ interrupt service routine.
- A properly designed OS ensures that an illegal program should not harm the execution of other programs.
- To ensure this, the OS operates in dual mode.

DUAL MODE OF OPERATION

1. The OS is design in such a way that it is capable of differentiating between the execution of OS code and user defined code.
2. To achieve this OS need two different modes of operations(**User and Kernel/supervisor/system/privileged)**
3. This is thereby controlled by mode bit added to hardware of computer system

TRANSITION FROM USER TO KERNEL MODE

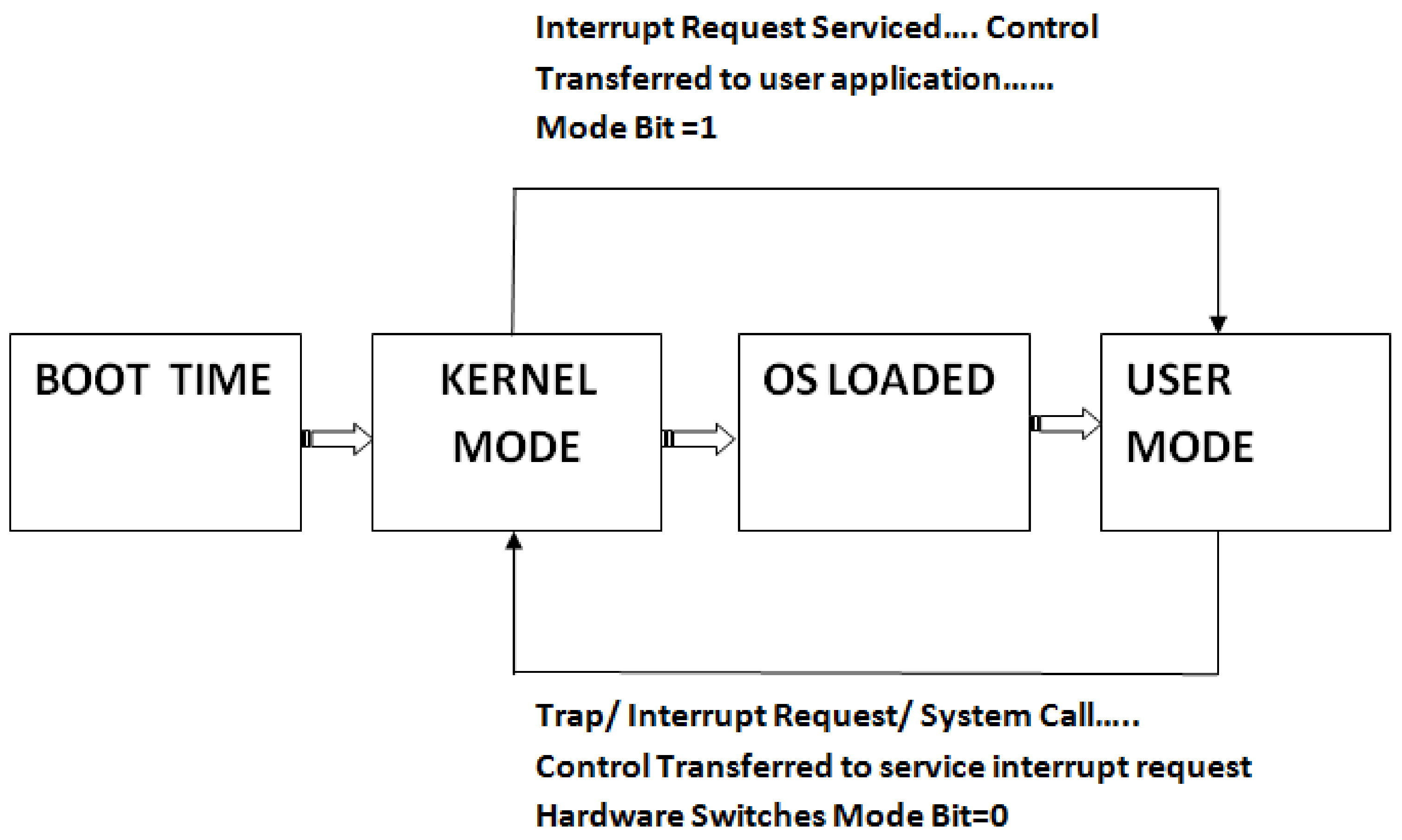
1. When a user application is executing on the computer system OS is working in user mode.
2. When a user application requests a service from OS (via a system call), the computer system transits from user mode to kernel mode to service that request


user process
user process executing
calls system call
return from system call
user mode
(mode bit = 1)
kernel
trap
mode bit = 0
return
mode bit = 1
execute system call
kernel mode
(mode bit = 0)

USER AND KERNEL MODE OF OPERATING SYSTEM

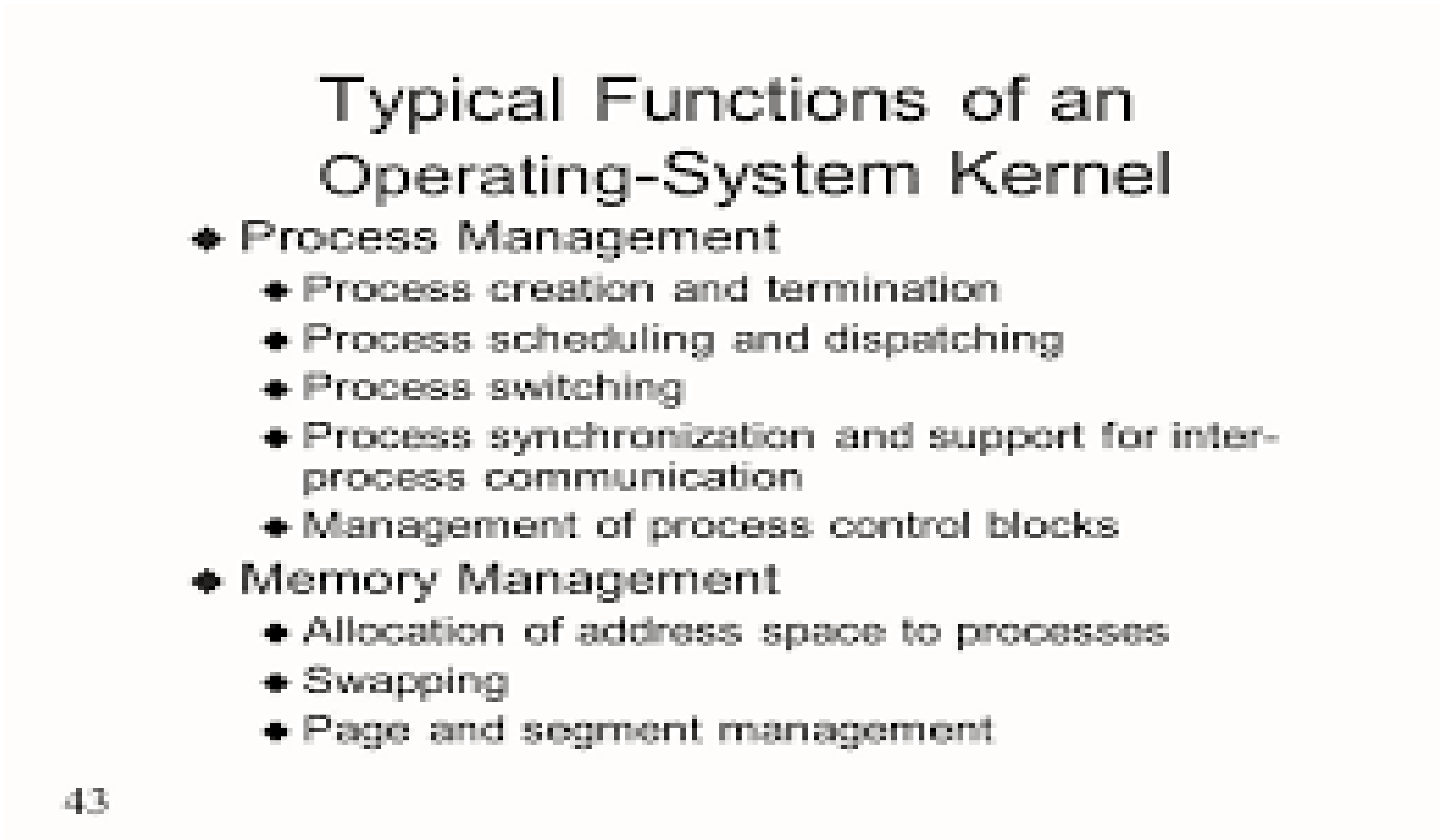
Mode Type	Definition	Mode Bit	Examples
User Mode	User Defined codes are executed	Mode Bit=1	Creation of word document or in general user using any application program
Kernel Mode	OS system codes are executed (also known as supervisor, system, or privileged mode)	Mode Bit=0	Handling interrupts-Transferring control of a process from CPU to I/O on request

- When the user starts the system the hardware starts in monitor/ kernel mode and loads the operating system.
- OS has the initial control over the entire system, when instructions are executed in kernel mode.
- OS then starts the user processes in user mode and on occurrence of trap, interrupt or system call again switch to kernel mode and gains control of the system.
- System calls are designed for the user programs through which user can ask OS to perform tasks reserved for operating system.
- System calls usually take the form of the trap. Once the OS service the interrupt it transfers control back to user program hence user mode by setting mode bit=1


Interrupt Request Serviced…. Control
Transferred to user application……
Mode Bit =1
BOOT TIME
KERNEL MODE
OS LOADED
USER MODE
Trap/ Interrupt Request/ System Call…..
Control Transferred to service interrupt request
Hardware Switches Mode Bit=0

BENEFITS OF DUAL MODE

- The dual mode of operation protects the operating system by designating some of the machine instructions that may cause harm as privileged instructions. These instructions can execute only in kernel mode.
- If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction, but rather treats the instruction as illegal and traps to the operating system.
- Examples of privileged instructions:
  - Switching to kernel mode
  - Managing I/O control
  - Timer Management
  - Interrupt Management



Typical Functions of an Operating-System Kernel

- Process Management
  - Process creation and termination
  - Process scheduling and dispatching
  - Process switching
  - Process synchronization and support for inter-process communication
  - Management of process control blocks
- Memory Management
  - Allocation of address space to processes
  - Swapping
  - Page and segment management

MAJOR OS CONCEPTS

- **Process management**
- **Memory management**
- **Storage management**
- **Information protection and security**
- **Distributed and special purpose Systems**

PROCESS MANAGEMENT

- A process is a program in execution.
- It is a unit of work within the system.
- Program is a passive entity, process is an active entity.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data (input)
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute. Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads

PROCESS MANAGEMENT ACTIVITIES

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

MEMORY MANAGEMENT

To execute a program all (or part) of the instructions must be in memory

All (or part) of the data that is needed by the program must be in memory.

- Data is brought to main memory by CPU generated I/O calls

Memory management determines what is in memory and when

- Optimizing CPU utilization and computer response to users

Memory management activities

- Keeping track of which parts of memory are currently being used and by whom
- Deciding which processes (or parts thereof) and data to move into and out of memory
- Allocating and de-allocating memory space as needed

STORAGE MANAGEMENT

OS provides uniform, logical view of information storage

- Abstracts physical properties to logical storage unit - **file**

1. File-System management
   - Most visible component of OS
   - Computers store information on several types of physical media
     - Magnetic disk-disk drive
     - Optical disk-CD
     - Magnetic tape-tape drive
   - Files usually organized into directories
   - Access control on most systems to determine who can access what
   - OS activities include
     - Creating and deleting files and directories
     - Primitives to manipulate files and directories
     - Mapping files onto secondary storage
     - Backup files onto stable (non-volatile) storage media

2. MASS-STORAGE MANAGEMENT

Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time

Most programs-compilers, assemblers, word processors, editors and formatters are stored on disk until loaded into memory hence proper management of disk is of central importance

Entire speed of computer operation hinges on disk subsystem and its algorithms

OS activities

- Free-space management
- Storage allocation
- Disk scheduling

PROTECTION AND SECURITY

Protection – any mechanism for controlling access of processes or users to resources defined by the OS (files, memory segments, CPU)

Security – defense of the system against internal and external attacks

- Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

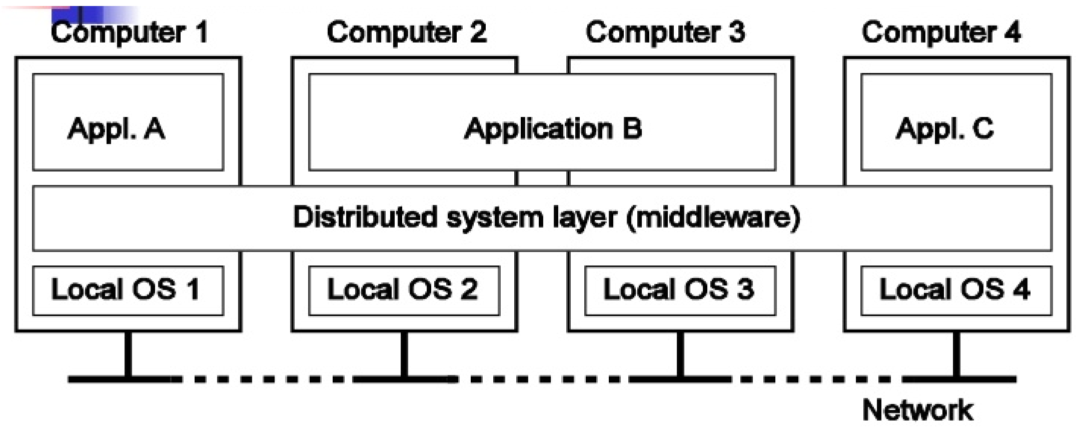
Systems generally first distinguish among users, to determine who can do what

- User identities (**user IDs**, security IDs) include name and associated number, one per user
- User ID then associated with all files, processes of that user to determine access control
- Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
- **Privilege escalation** allows user to change to effective ID with more right

DISTRIBUTED SYSTEMS

Collection of physically separate, possibly heterogeneous computer systems, networked to provide users with access to various resources amongst them (files, computing devices)

Middleware is the bridge that connects distributed applications across dissimilar physical locations, hardware platform, n/w technologies, OS and programming languages



A distributed system organized as middleware.

SPECIAL-PURPOSE SYSTEMS

Real-time embedded systems

- Found on embedded computers
  - VCRs, cars, microwaves
- Very specific tasks, little or no user interface
- Embedded systems almost always have **real-time OS (RTOS)**
  - **Home automation, ACs, refrigerator etc**

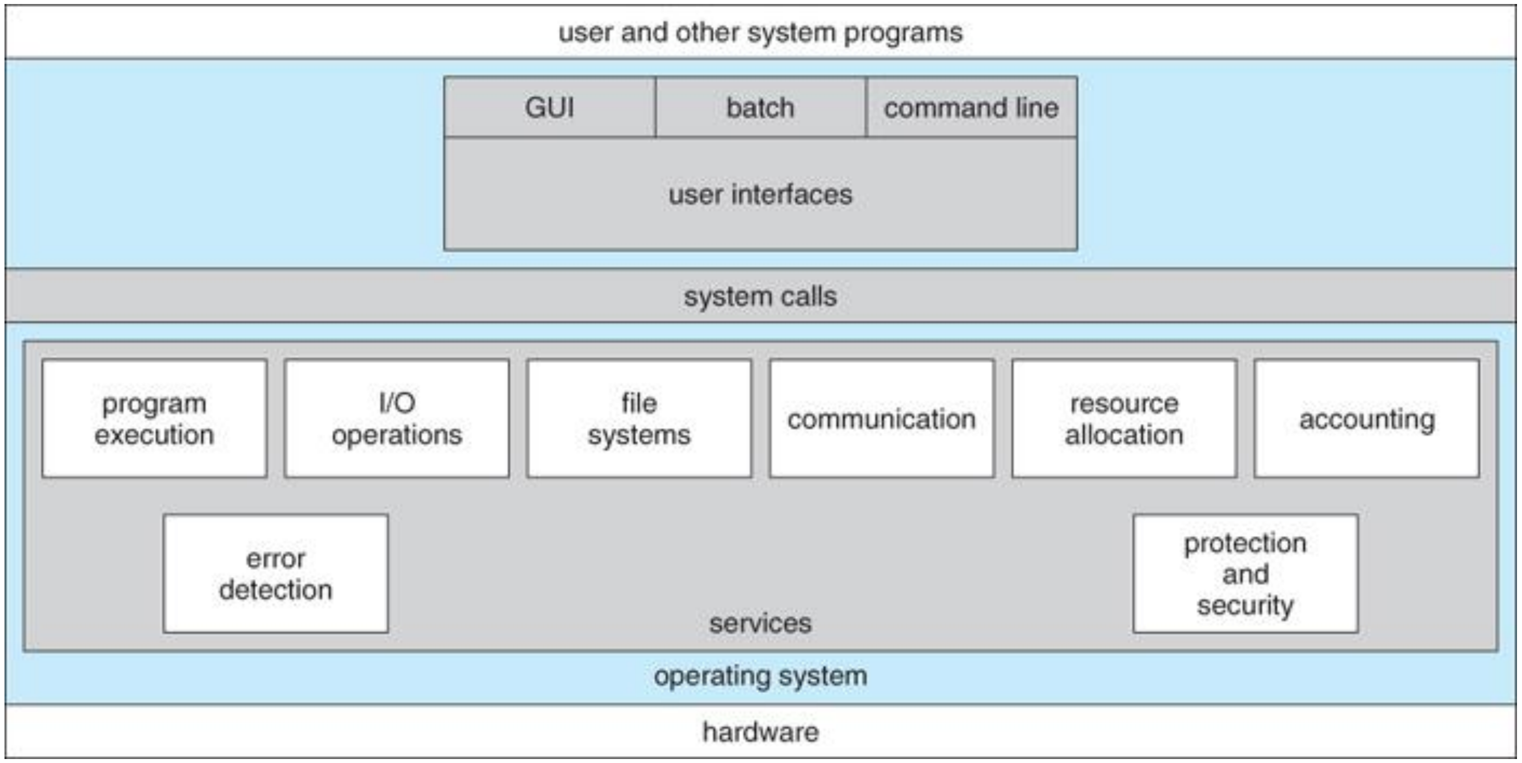
SPECIAL-PURPOSE SYSTEMS (2)

Handheld Systems

- PDAs and cell phones
- Use special-purpose embedded operating systems
- Many physical device limitations (user interface, storage, performance)

SERVICES PROVIDED BY THE OS

An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.



SERVICES PROVIDED BY THE OS

User Interface

- CLI, GUI

Program Execution

- load a program into memory and execute, end its execution, either normally or abnormally / forcefully.

File system manipulation

- create and delete, allowing or denying access to files or directories based on file ownership

SERVICES PROVIDED BY THE OS

Input / output Operations

- OS provide a means to do Input / Output(read or write) operation with any file

Communication

- Processes can communicate using shared memory or via message passing.

Resource Allocation

- Facilitate resource sharing (CPU, main memory storage, file storage and I/O devices)

SERVICES PROVIDED BY THE OS

- Error Detection
  - internal and external hardware errors
    - memory error
    - device failure
  - software errors
    - arithmetic overflow
    - access forbidden memory locations
  - Inability of OS to grant request of application

- Error Response
  - simply report error to the application
  - Retry the operation
  - Abort the application

SERVICES PROVIDED BY THE OS

Accounting

- collect statistics on resource usage
- monitor performance (eg: response time)
- keeps track of which users are using how much and what kinds of computer resources
- useful for anticipating future enhancements
- used for billing users (on multiuser systems)

Security : the user needs to authenticate him or her to the system before using

Protection: access to system resources in a controlled manner.

SYSTEM CALLS

Programming interface to the services provided by the OS

Routines/ functions/calls used to perform OS functions are called system calls.

Available in the form of commands

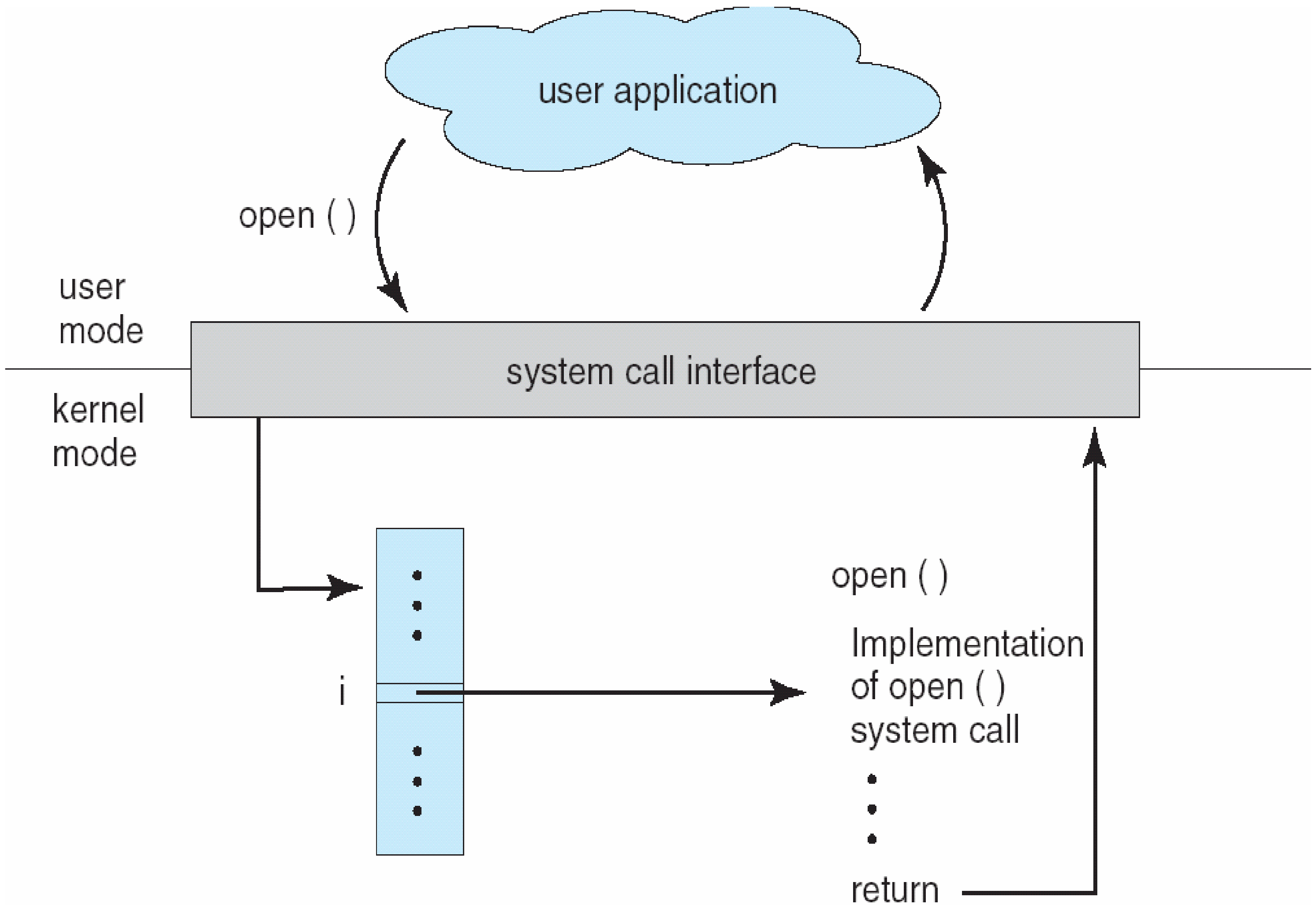
System call instructions are normally available in assembly language

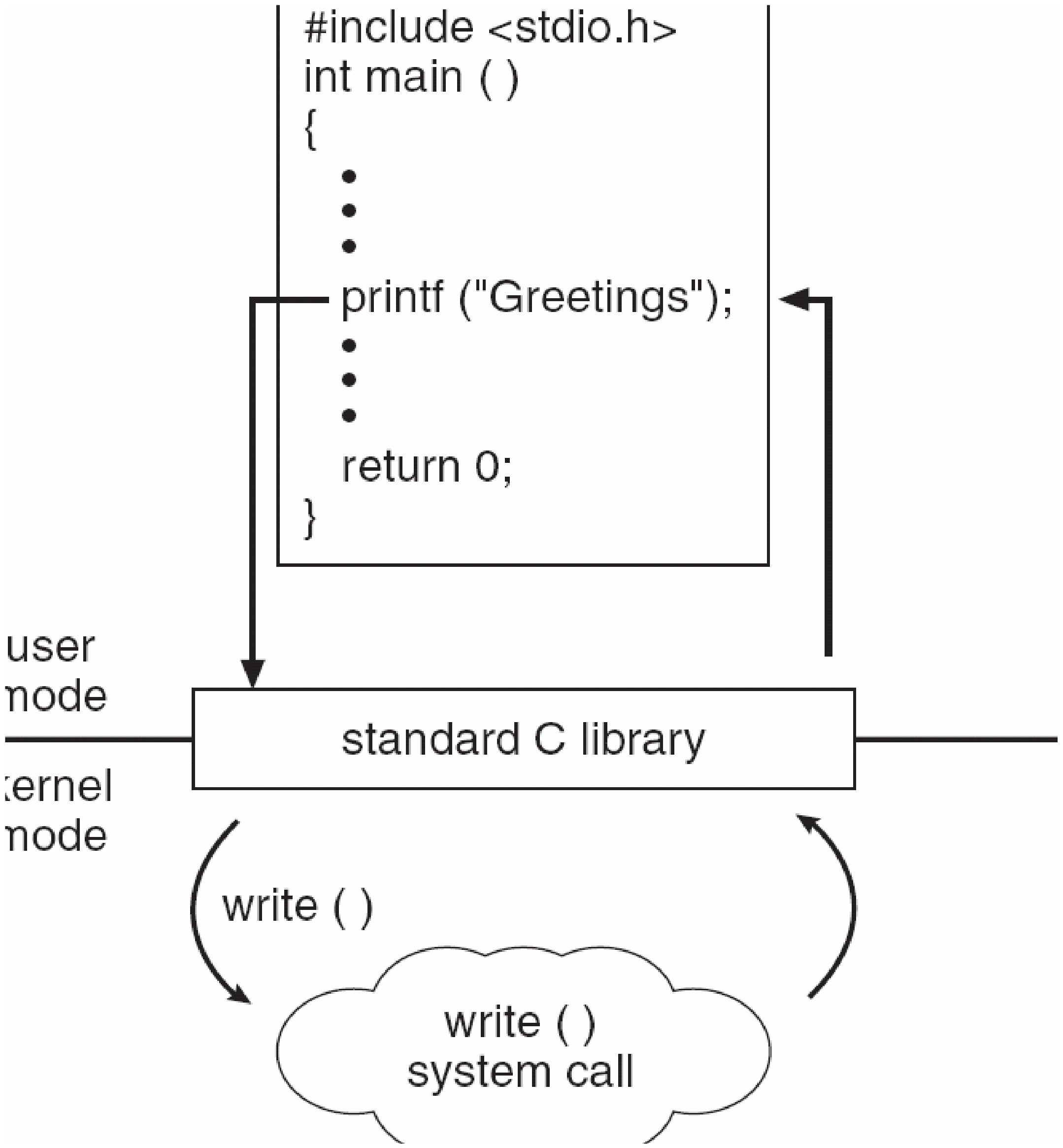
When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a system call.

When a program makes a system call, the mode is switched from user mode to kernel mode.

Generally, system calls are made by the user level programs in the following situations:

- Creating, opening, closing and deleting files in the file system.
- Creating and managing new processes.
- Creating a connection in the network, sending and receiving packets.
- Requesting access to a hardware device, like a mouse or a printer.


user application
open ( )
user mode
kernel mode
system call interface
i
open ( )
Implementation of open ( ) system call
return



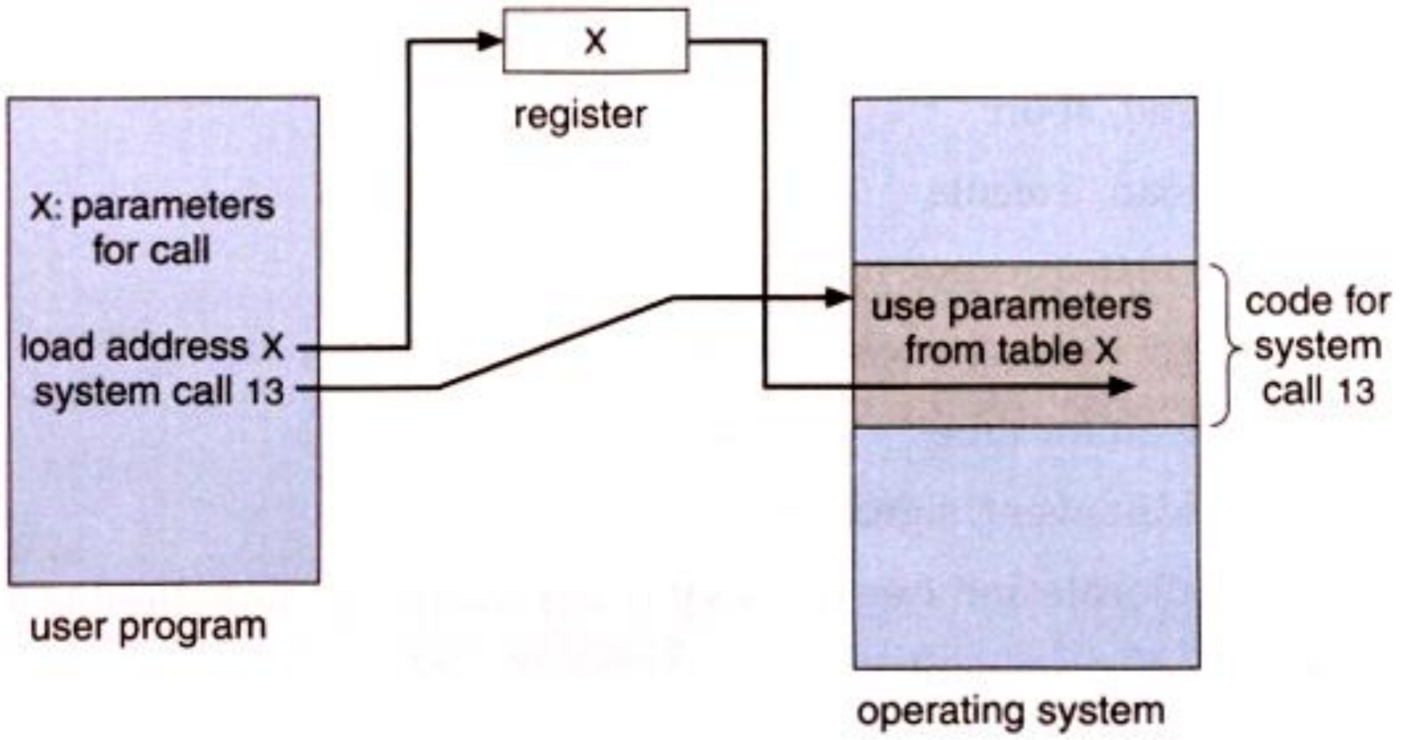
```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

SYSTEM CALL PARAMETERS

Three general methods used to pass parameters to the OS

- Simplest: pass the parameters in *registers*
- In some cases, may be more parameters than registers
- Parameters stored in a *block*, or table, in memory, and address of block is passed as a parameter in a register This approach taken by Linux and Solaris
- Parameters can be 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


X
register
X: parameters for call
load address X
system call 13
user program
use parameters from table X
code for system call 13
operating system

TYPES OF SYSTEM CALLS

There are 5 different categories of system calls:

Process control

- A running program needs to be able to stop execution either normally or abnormally.
- When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger.

file management

- Some common system calls are *create*, *delete*, *read*, *write*, *reposition*, or *close*.
- Also, there is a need to determine the file attributes – *get* and *set* file attribute.
- Many times the OS provides an API to make these system calls.

device management

- Process usually require several resources to execute, if these resources are available, they will be granted and control returned to the user process.
- These resources are also thought of as devices.
- Some are physical, such as a video card, and others are abstract, such as a file.
- User programs request the device, and when finished they release the device.
- Similar to files, we can *read*, *write*, and *reposition* the device.

information maintenance

- Some system calls exist purely for transferring information between the user program and the operating system. An example of this is time, or date, number of current users, version of OS, amount of free space etc
- The OS also keeps information about all its processes and provides system calls to report this information.

Communication

- There are two models of inter process communication, the message-passing model and the shared memory model.
- Message-passing uses a common mailbox to pass messages between processes.
- Shared memory use certain system calls to create and gain access to regions of memory owned by other processes.
- The two processes exchange information by reading and writing in the shared data.

- Process control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- File management
  - create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get process, file, or device attributes
  - set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices

Figure 2.8 Types of system calls.

SYSTEM PROGRAMS

System programs (system utilities) provide a convenient environment for program development and execution.

Some of them are simply user interfaces to system calls.

The can be divided into:

File management

- Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

Status information

- Some ask the system for info -date, time, amount of available memory, disk space, number of users
- Others provide detailed performance, logging, and debugging information
- Typically, these programs format and print the output to the terminal or other output devices
- Some systems implement a registry -used to store and retrieve configuration information

- File modification:
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- Programming-language support
  - Compilers, assemblers, debuggers and interpreters are sometimes provided
- Program loading and execution
  - Absolute loaders, relocatable loaders, linkage editors, overlay loaders, debugging systems for higher-level and machine language
- Communications
  - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

Background Services:

- Some processes terminate after completion of their task while others continue until system is halted
- Constantly running system program processes are known as services, subsystems, or daemons
  - n/w daemon, process schedulers

OS DESIGN AND IMPLEMENTATION

- Start by defining goals and specifications

- Design will be affected by choice of hardware, type of system: batch, time shared, single user, multi user, network, distributed, real time, general purpose.

- *Requirements can be divided into 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

Important principle is to separate

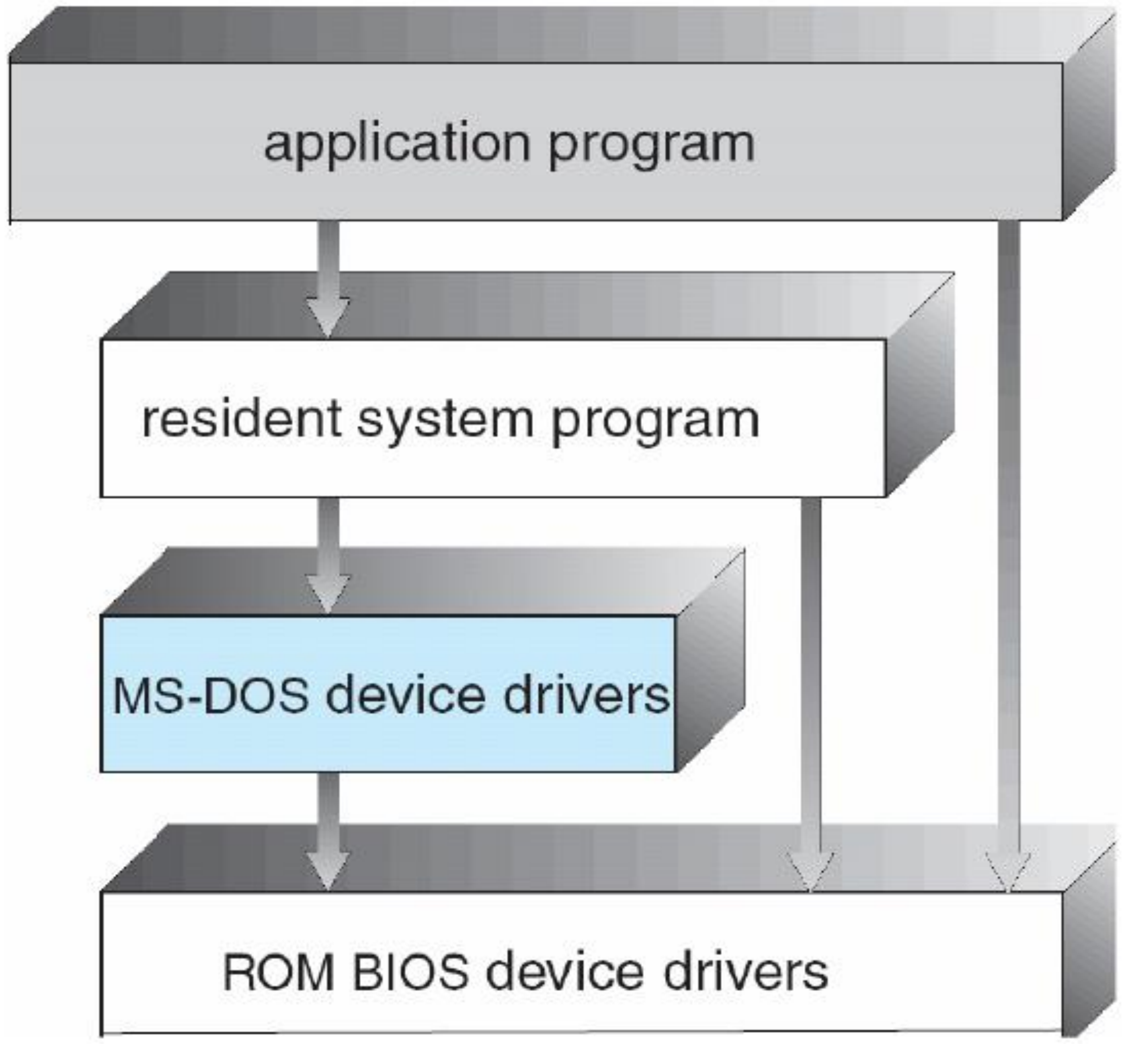
- **Policy:** What will be done?
- **Mechanism:** How to do it?
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

SYSTEM STRUCTURE

Simple Structure

- No well defined structure
- Started as small, simple and limited systems
- MS-DOS: written to provide the most functionality in the least space
- Not divided into modules
- Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

MS-DOS LAYER STRUCTURE



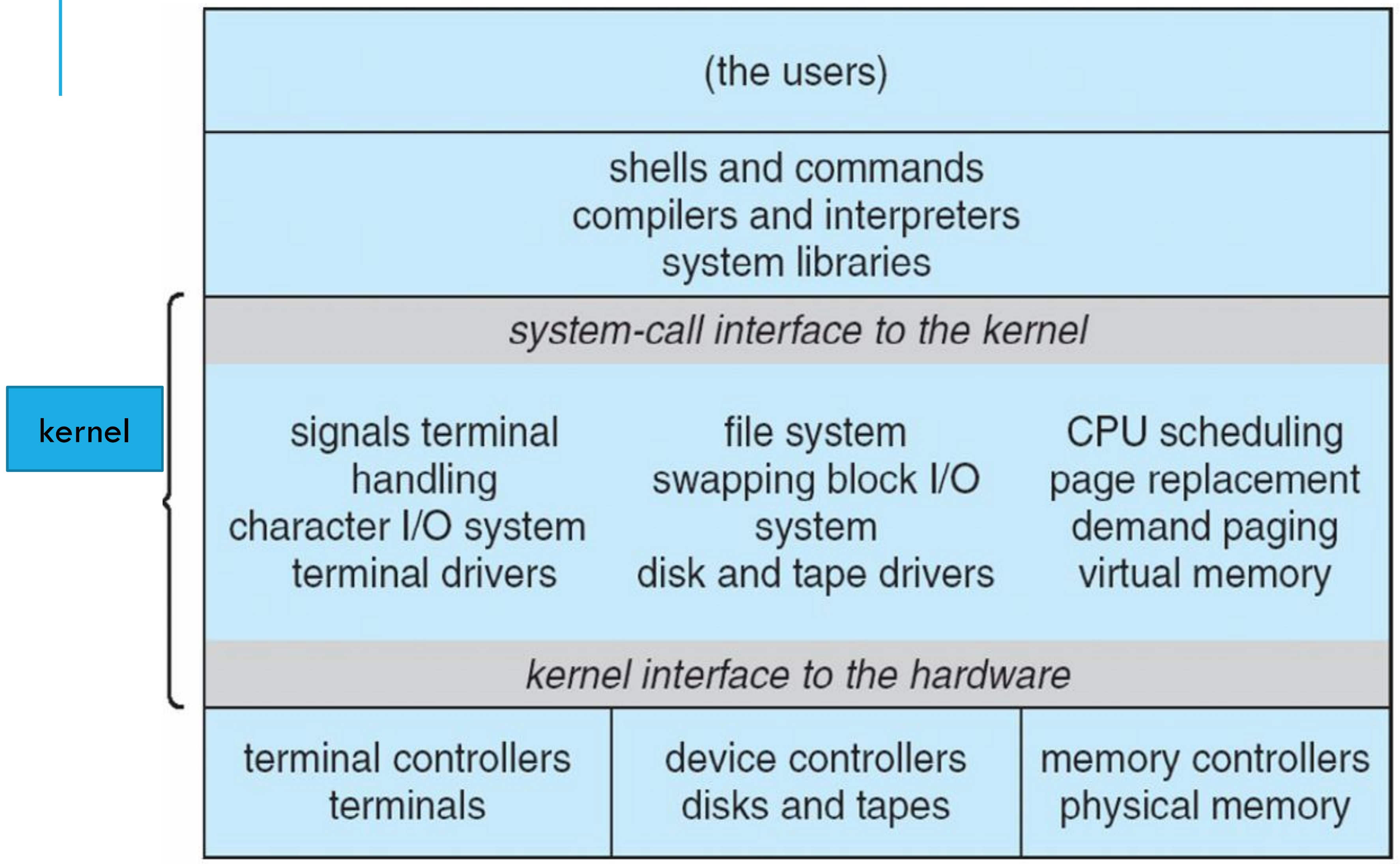
LAYERED APPROACH

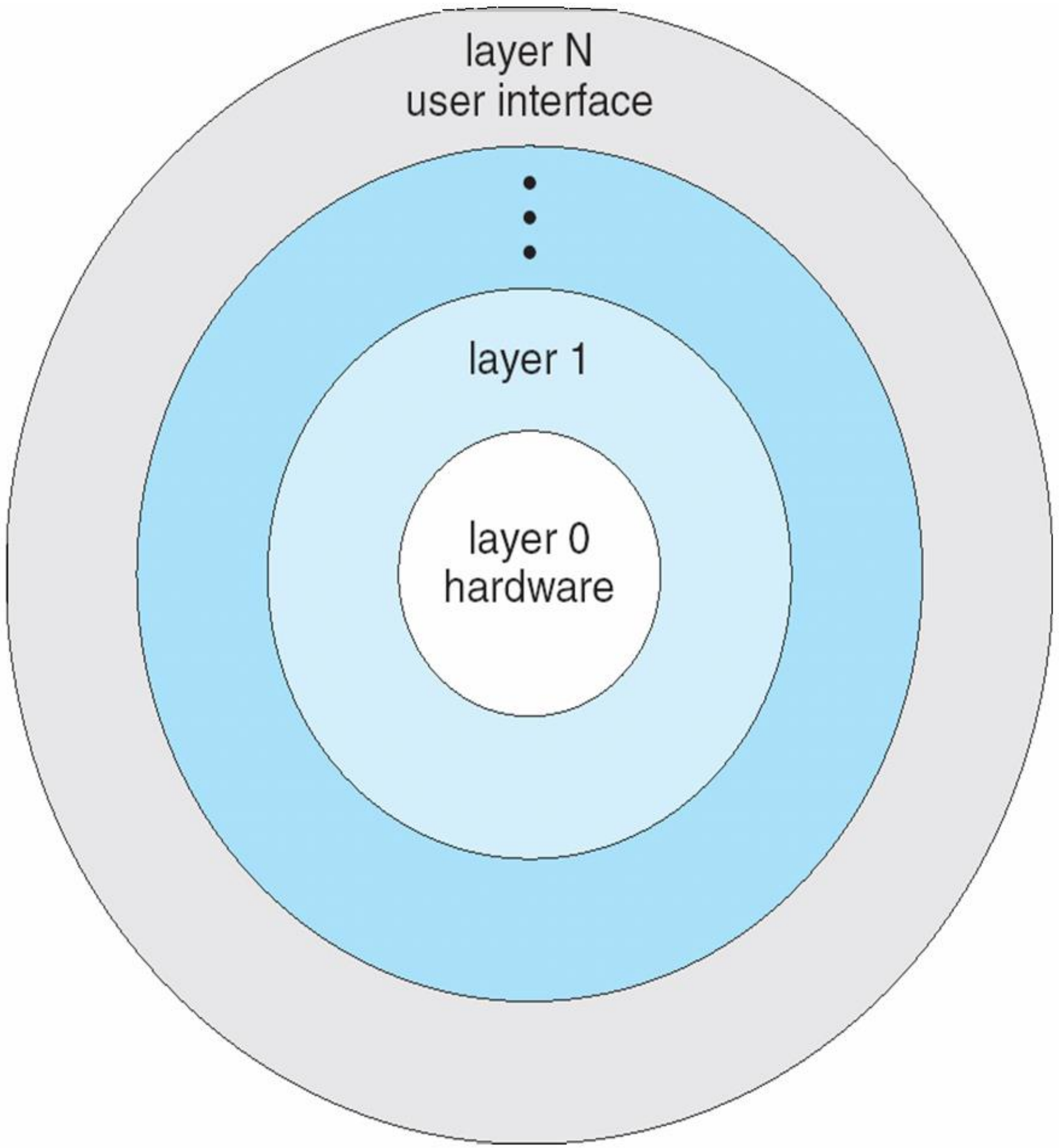
- The operating system is divided into a number of layers (levels), each built on top of lower layers.

- The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.

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

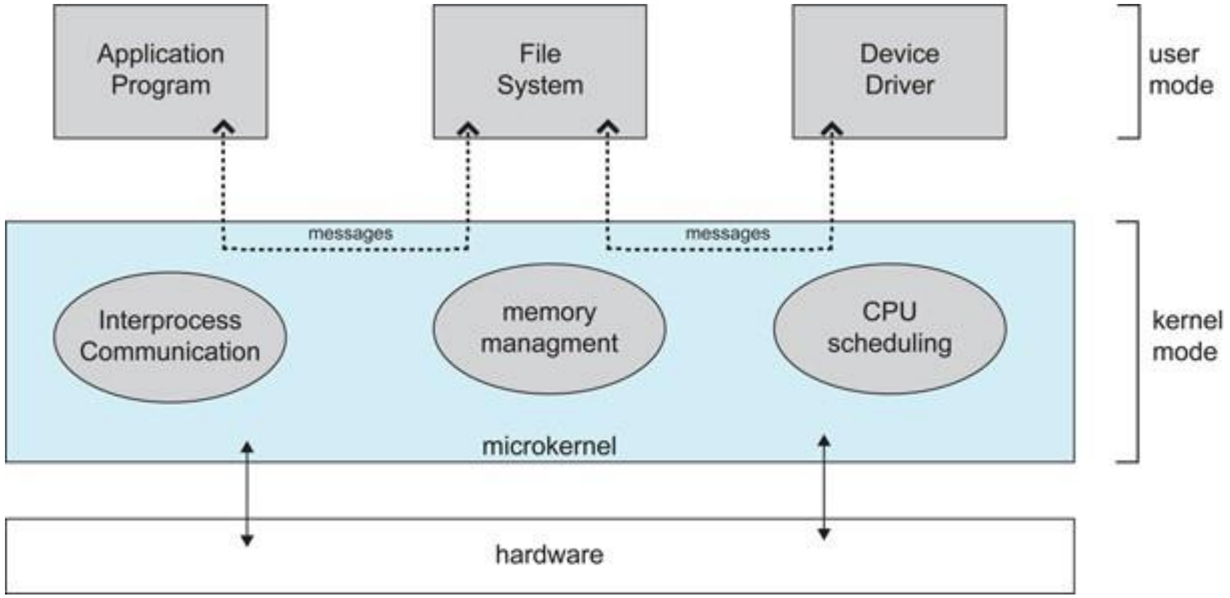
UNIX SYSTEM STRUCTURE




layer N
user interface
•
•
•
layer 1
layer 0
hardware

MICROKERNEL SYSTEM STRUCTURE

- Structures the OS by removing all non essential components from kernel and implementing them as system and user level programs.
- The result is smaller kernel.
- Communication takes place using message passing.
- Program and service never interact directly, indirectly by message passing with microkernel


Application Program
File System
Device Driver
user mode
messages
messages
Interprocess Communication
memory managment
CPU scheduling
kernel mode
microkernel
hardware

BENEFITS OF MICROKERNEL

Ease of extending OS: new services are added to user space, no modification reqd. for kernel.

Easier to port from 1 h/w design to another

Provides more security and reliability: since most services are running as user rather than kernel processes

If a service fails, rest of the OS remains untouched.

OTHER STRUCTURES

Modules

- Kernel has a set of core components and links in additional services via modules at boot or run time
  - Eg: Solaris

Hybrid Systems

- Mac OS X
- iOS
- Android

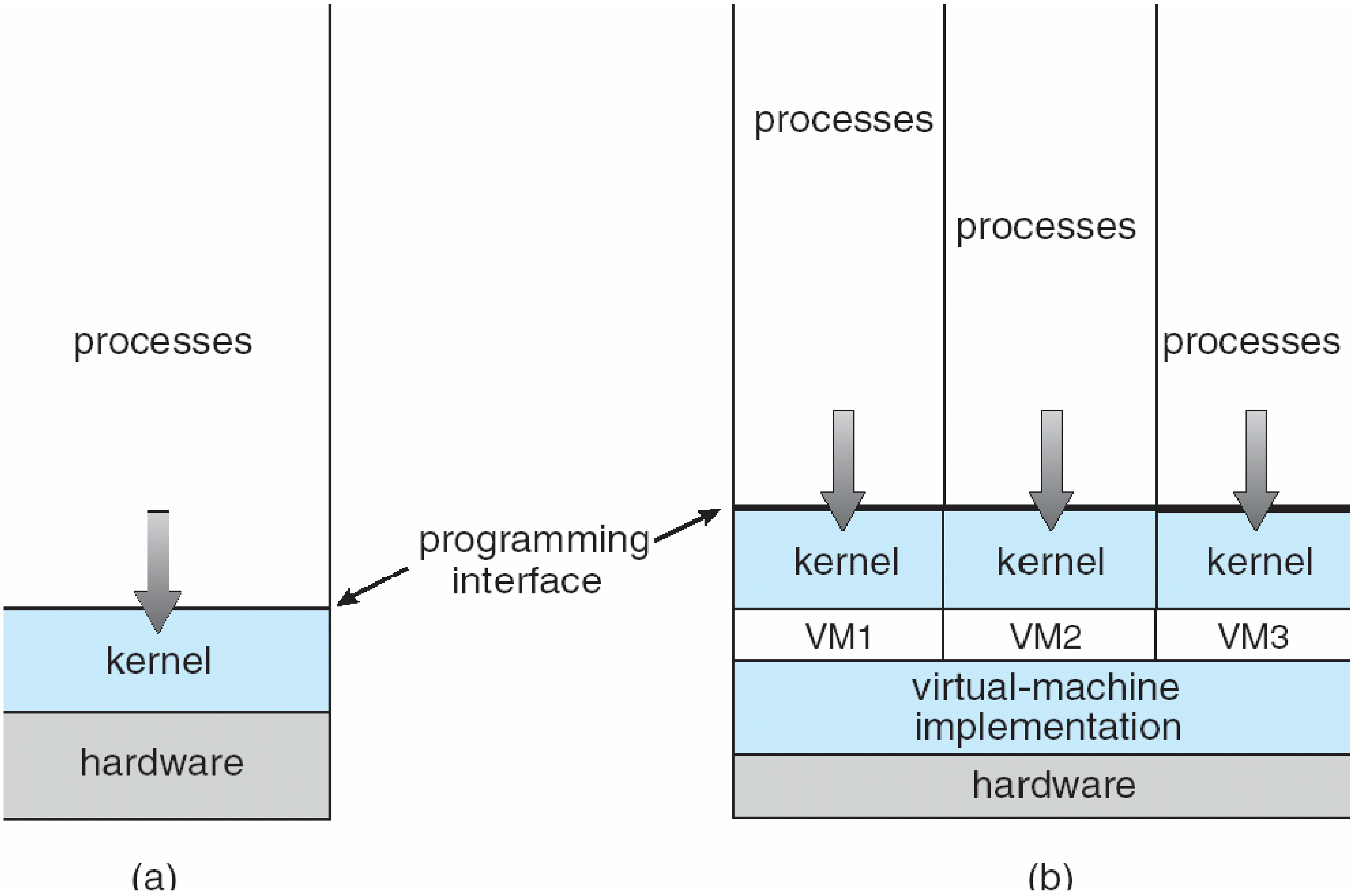
VIRTUAL MACHINES

A virtual machine takes the layered approach to its logical conclusion.

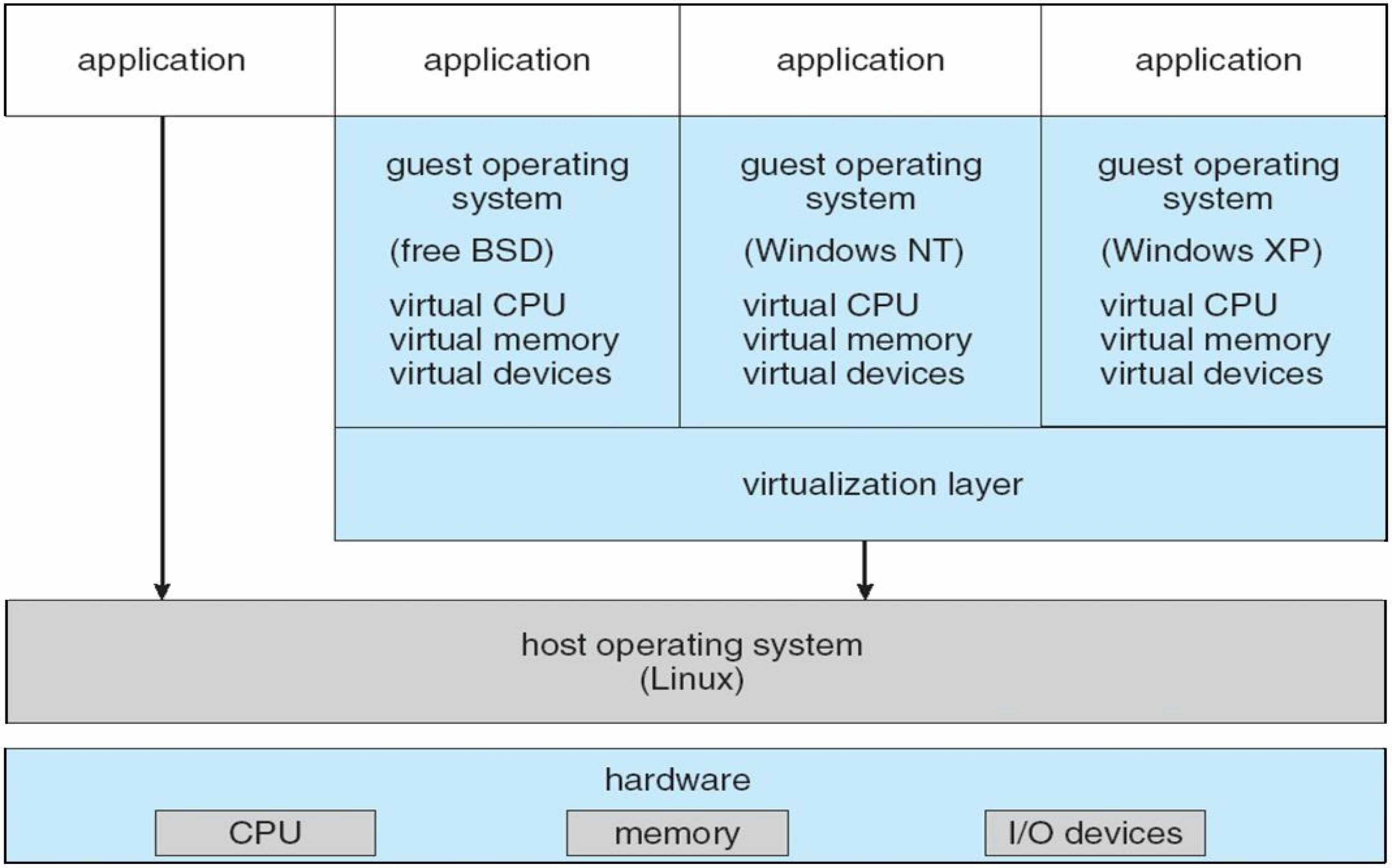
The operating system host creates the illusion that a process has its own processor and (virtual) memory

A virtual machine provides an interface *identical* to the underlying bare hardware

Each guest provided with a (virtual) copy of underlying computer


processes
kernel
hardware
(a)
programming interface
processes
processes
processes
kernel
kernel
kernel
VM1
VM2
VM3
virtual-machine implementation
hardware
(b)

VMWARE ARCHITECTURE



OPERATING-SYSTEM DEBUGGING

- Debugging is finding and fixing errors, or **bugs**
- Oses generate **log files** containing error information
- Failure of an application can generate **core dump** file capturing memory of the process
- Operating system failure can generate **crash dump** file containing kernel memory
- Beyond crashes, performance tuning can optimize system performance
- Kernighan's Law: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

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

BIOS --------executes MBR
MBR-----------executes GRUB
GRUB----------executes kernel
Kernel-----executes /sbin/init
Init---------executes run level programs
Runlevel-----runlevel programs are executed from /etc/rc.d/rc*.d/

BIOS	Kernel
Lives in motherboard	Lives in boot drive
OS independent	OS dependent
Should be up and running before kernel	
Used during booting process	1st thing that is loaded and remains in main memory for entire session