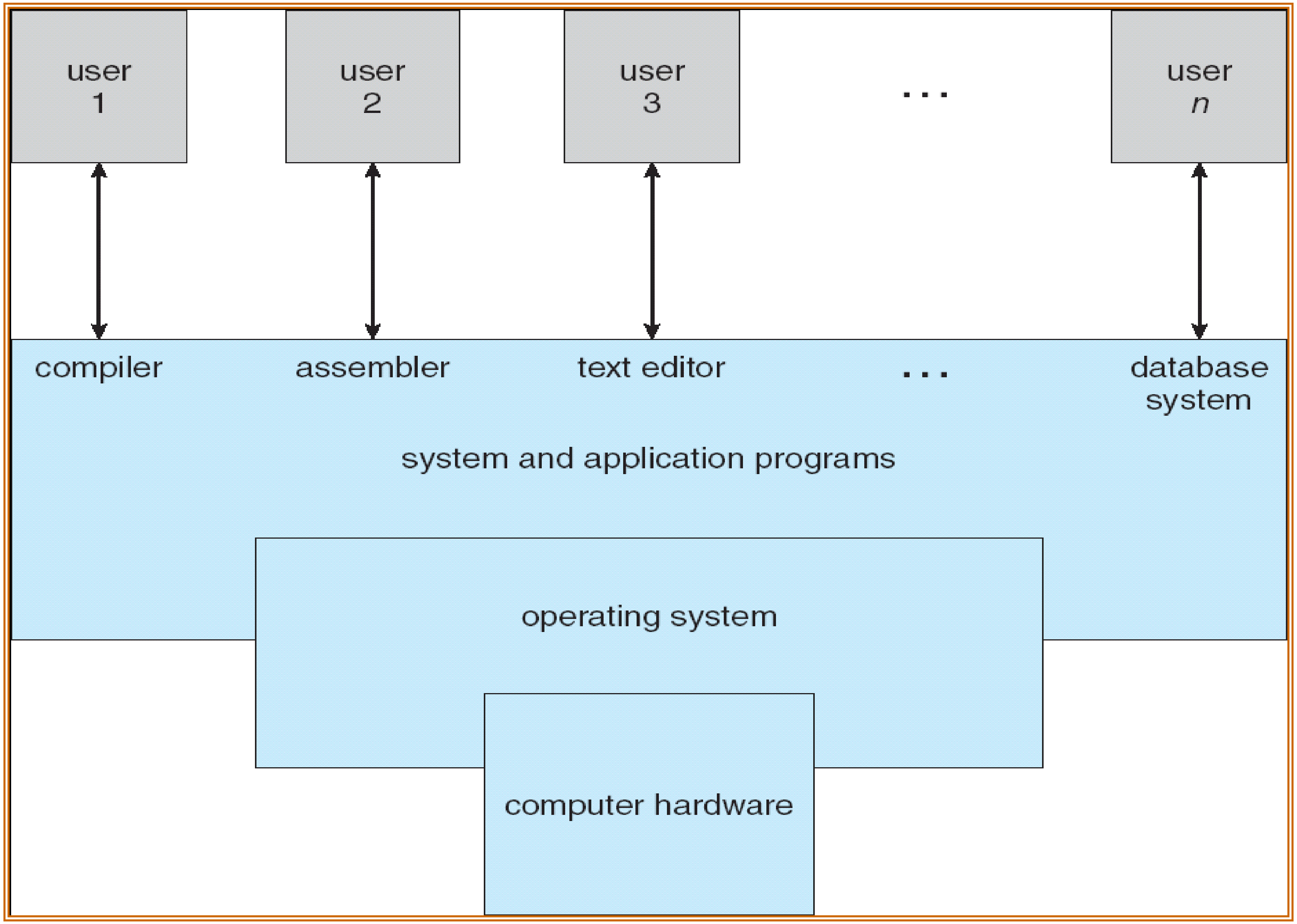
Operating systems:-

- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer
- “The one program running at all times on the computer” is the **kernel.** Everything else is either a system program (ships with the operating system) or an application program



Abstract view of the components of a Computer system

Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an *interrupt*

I/O Structure:-

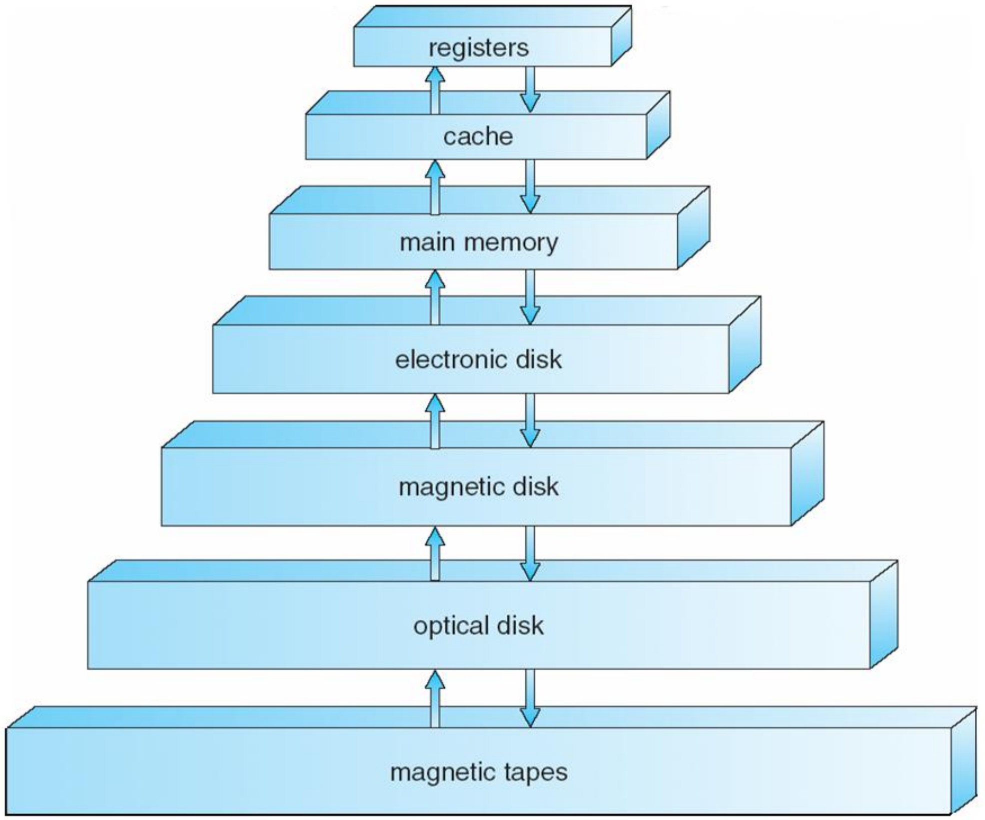
- After I/O starts, control returns to user program only upon I/O completion
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access)
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing
- After I/O starts, control returns to user program without waiting for I/O completion
  - **System call** – request to the operating system to allow user to wait for I/O completion
  - **Device-status table** contains entry for each I/O device indicating its type, address, and state
  - Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt

Storage Structure

- Main memory – only large storage media that the CPU can access directly
- Secondary storage – extension of main memory that provides large nonvolatile storage capacity
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
  - The **disk controller** determines the logical interaction between the device and the computer

Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility
- **Caching** – copying information into faster storage system; main memory can be viewed as a last *cache* for secondary storage



Operating systems functions

- Objective of **OS**:

- OS makes a computer more convenient to use.
- OS allows the computer system resources to be used in an efficient manner.
- It acts as a manager for all the system resources.

- Function: A number of statements grouped in to a single logic unit.
- OS provide the programmer with a convenient interface for using system.
- There are 7 functions in Operating Systems.

  1.Program development.

  2. Program execution

  3.Access to I/O devices / I/O devices

  4.Controlled access to files/File Management

  5.System access.

  6.Erroer detection and response.

  7.Accounting.

1. <u>Program development</u>:

OS provide the Editors and debuggers to assist the programmer in creating program.

These are not the part of the core OS and OS is referred/supported the software of program development tools.

2. <u>Program execution:</u>

A number of steps needed to performed for execution of a program. Instructions and data are loaded into memory, I/O devices and files must be initialized ,other resources must be prepared. These all duties scheduled by OS.

3. Access to I/O devices :

Each I/O device need its own set of instructions for operation. The OS provides uniform interface so the user can access such devices using simple read and writes.

4. Controlled access to files :

For file access the OS must understand the type of I/O device (CD, tape drive) and structure of data stored in the storage.

5. System access :

For shared or public systems the OS provide the control access to the system based on the specific system resources.

The OS provide the protection of resources and data from unauthorized users.

6. Error detection and response :

If any errors can occur while a computer system is running. Like H/w errors , memory errors and S/W program errors like divide by zero…then OS must provide the related response to clears the error condition with latest running application.

7. Accounting :

A good OS collect usage statistics for various resources and monitors performance parameters like response time. Due to monitoring future need and tuning then only improve the performance.

Protection and Security

- ***Protection***:

  When information is kept in a computer system, a major concern is its protection from both physical damages (reliability) and improper access (protection).

  Def: Any mechanism for controlling accesses of processes/user to resources defined by OS.

- ***Reliability*** is generally provided by duplicate copies of files.

  File system can be damaged by H/W problems, power failures, head crashes, temperature extremes.

Protection:

- **Protection** is nothing but provide controlled access by limiting the type of file access that can be made.
- Access permitted/denied depending on the several features, one of which is type of access requested.
  - Read: Read from a file.
  - Write :write/rewrite the file.
  - Append: write new information at the end of file.
  - Delete: delete the file and free its space for possible reuse.

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 is 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.

Distributed Systems

- *Distributed system* is nothing but to share computational tasks and provide a rich set of features to user.
- *Distributed system* consists of collection of processors that do not share memory/a clock.
- Each processor has its own local memory.
- The processors communicate one another through various communication lines such as high speed buses or telephone lines.
- Also referred to as loosely coupled system.

- **Network:**
- It is a communication path between two/more systems.

Networks are divided based on the distance between their nodes.

1. LAN: Local Area Network

   LAN exists within a room/a floor/ a building.

2. MAN: Metropolitan Area Network.

   MAN could link buildings within a city.

   Bluetooth devices communicate over a short distance of several feet's, in essence creating a small-area network

3. WAN: Wide area Network.

   Usually exists between buildings, cities, or countries.

   i.e. The global companies may have use a WAN to connect its offices world wide.

Special purpose Systems

- The Special purpose operating systems are nothing but Real-time Systems.
- ***A Real-time system*** is used when **rigid** time requirement have been placed on the operation of the processor /the flow of data.
- Example: 1. Scientific experiments, medical imaging system, industrial control system and display system, Some of automobile engines fuel-injecting system, homo-appliances controllers and weapon systems.
- Sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs.
- A real time system has well defined, fixed time constraints.
- Processing must be done within the defined constraints, otherwise system will fail.

Real-time systems are mainly 2 types

- 1. ***Hard real time system:*** The response must be **within the deadline**. (OR)
- A hard real-time system guarantees that **critical tasks** be completed **on time**.
- This goal requires that all delays in the system be bounded, from the retrieval of stored data to the time that it takes the Operating system to finish any request made of it.
- Example: Car breaks, Rocket. . .
- 2. ***Soft real-time system:*** A soft-real-time system response of the system must be within the average value (OR) output is does not meet dead line but within the average time.
- Example: Washing machine, OHP projector . . .

Operating-System Operations

- Interrupt driven by hardware
- Software error or request creates **exception** or **trap**
  - Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user

Operating System Structures

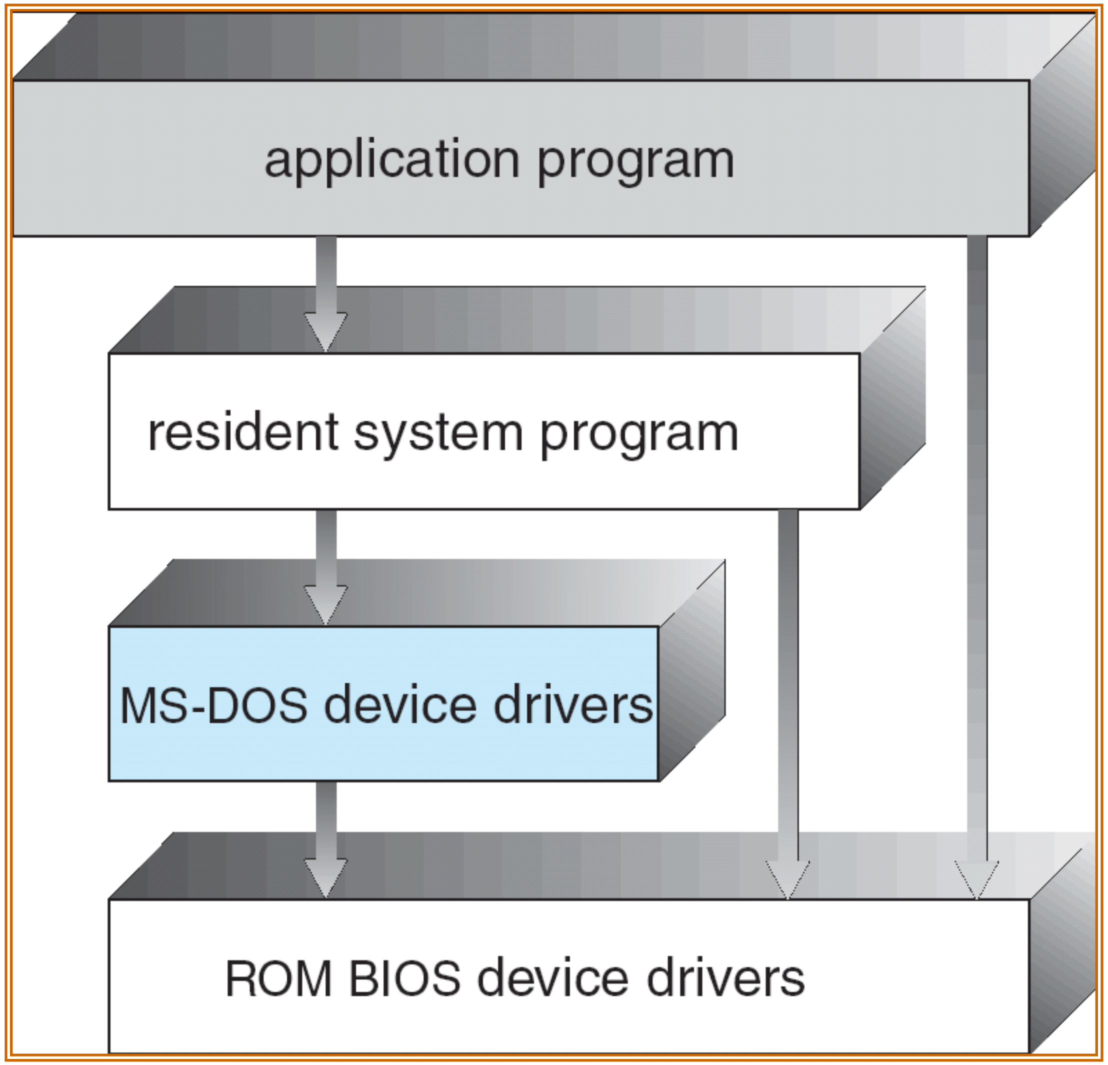
- Operating system provides environment within which programs are executed.

- **Multiprogramming** needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job
- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨**process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

Simple Structure

- 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

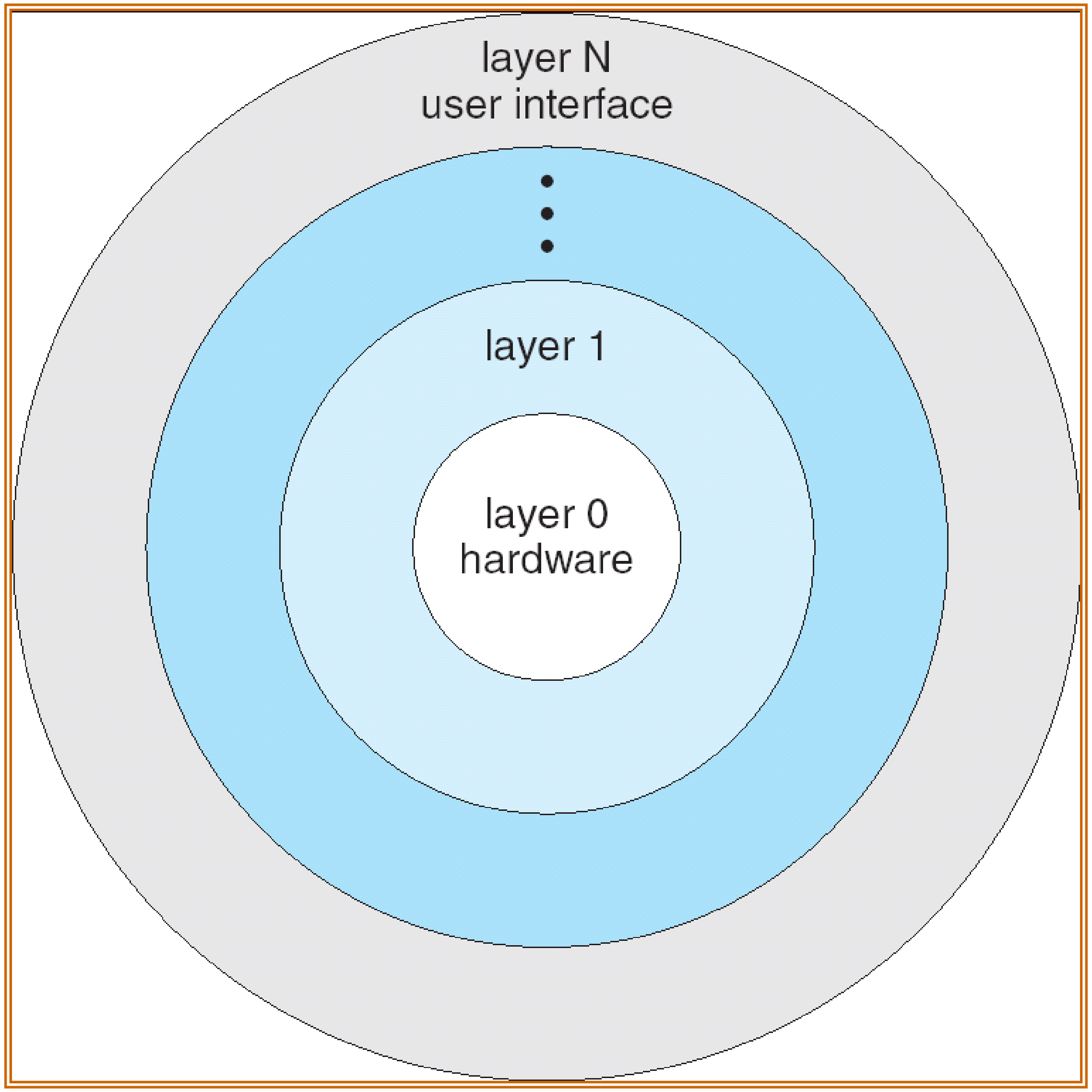
MS-DOS Layer Structure



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

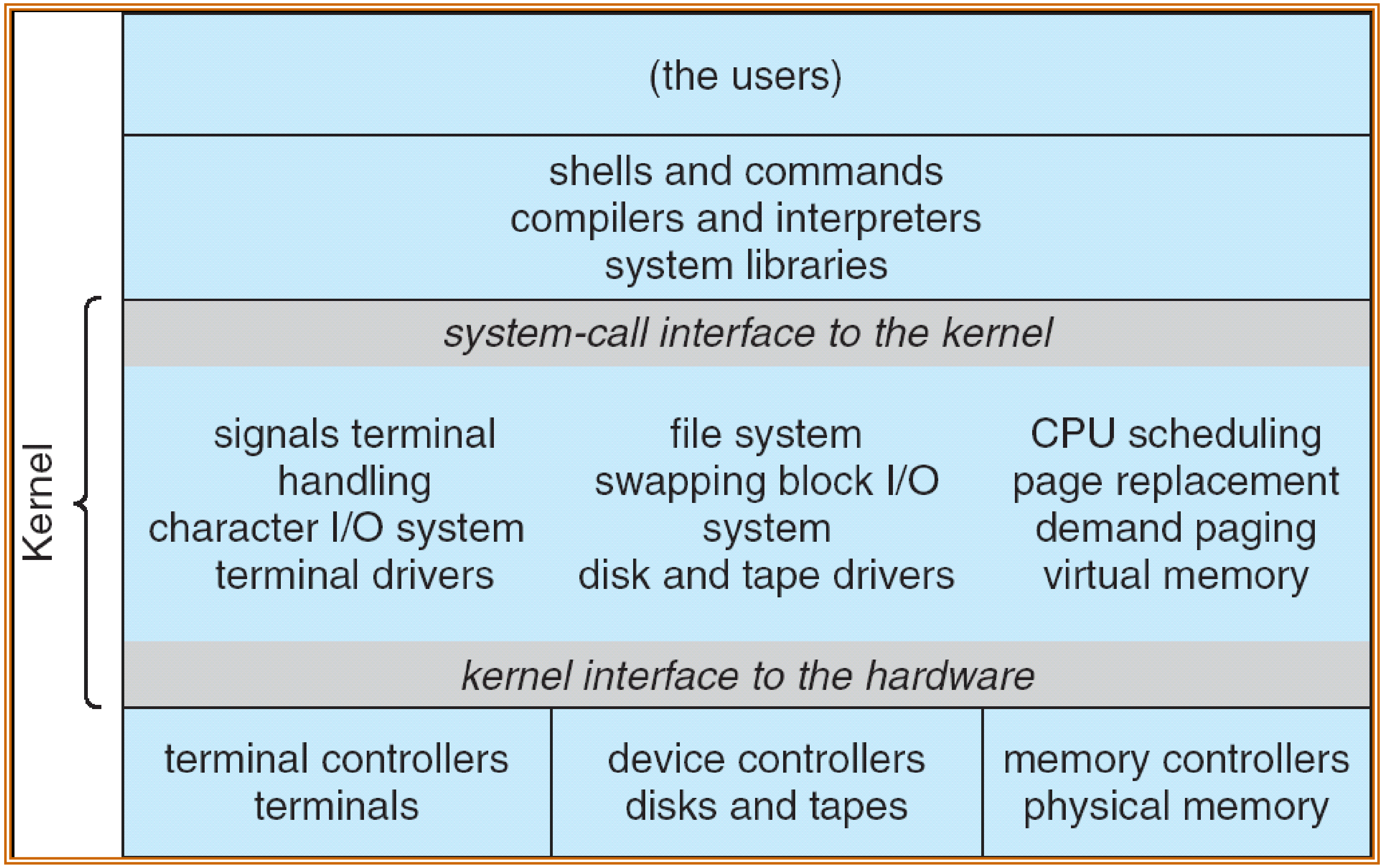
Layered Operating System



UNIX

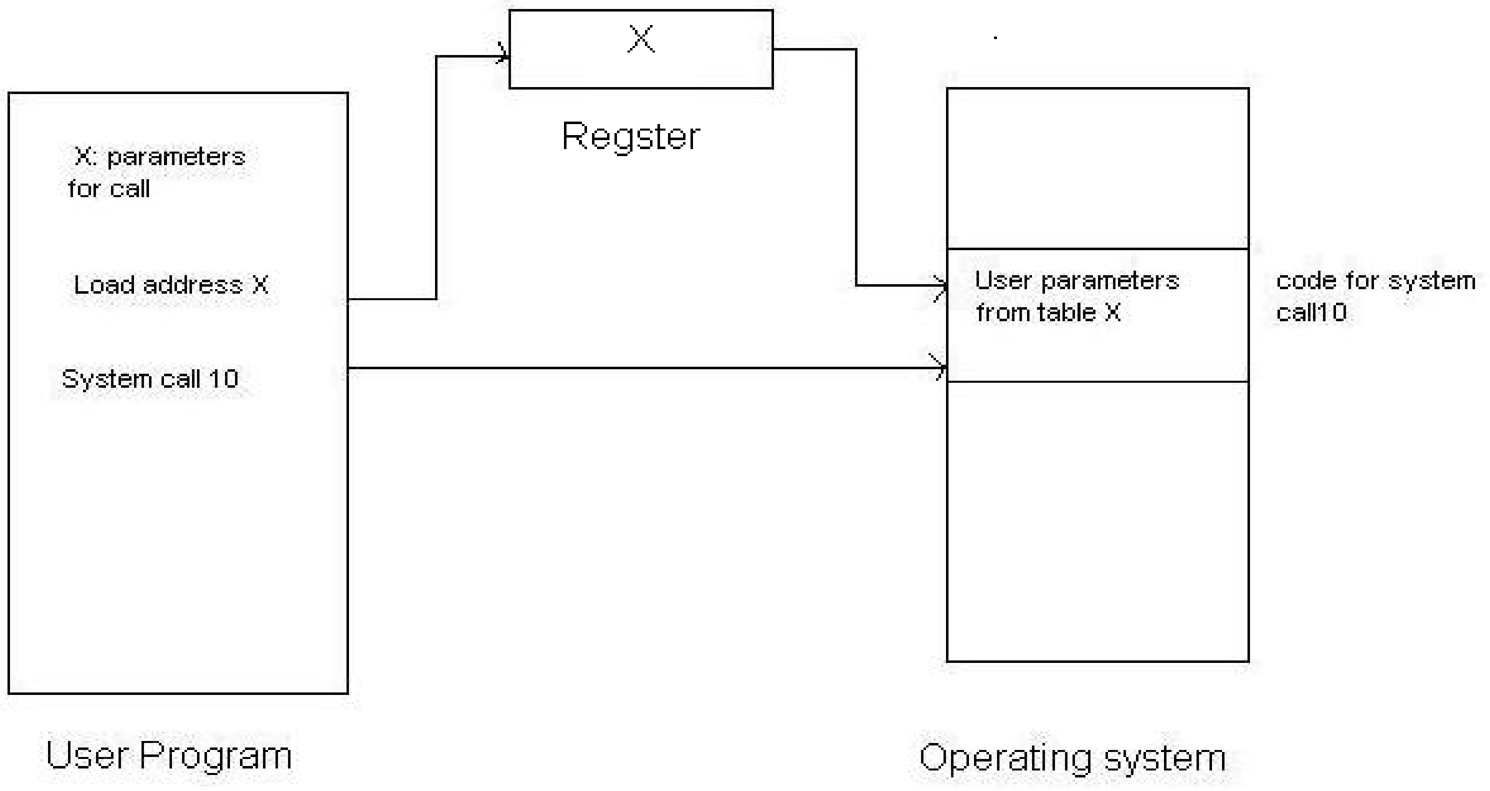
- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

UNIX System Structure



System calls

- System calls provide the interface between a process and the Operating system.



- 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
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?

(Note that the system-call names used throughout this text are generic)

- System calls can be grouped roughly in to 5 categories:

1. Process control
2. File management/Manipulation
3. Device management
4. Information maintenance
5. Communication

1. Process control :

- End, abort
- Load, execute
- Create process, terminate process
- Get process attribute, set process attribute
- Wait for time
- Wait event, signal event
- Allocation and free memory

2. File management:

- create file, delete file.
- Open, close
- read, write
- Get file attributes, set file attributes

3. Device management:

- Request device, release device
- Read, write repository
- Get device attribute, set device attribute
- Logically attach or detach device

4. Information maintenance:

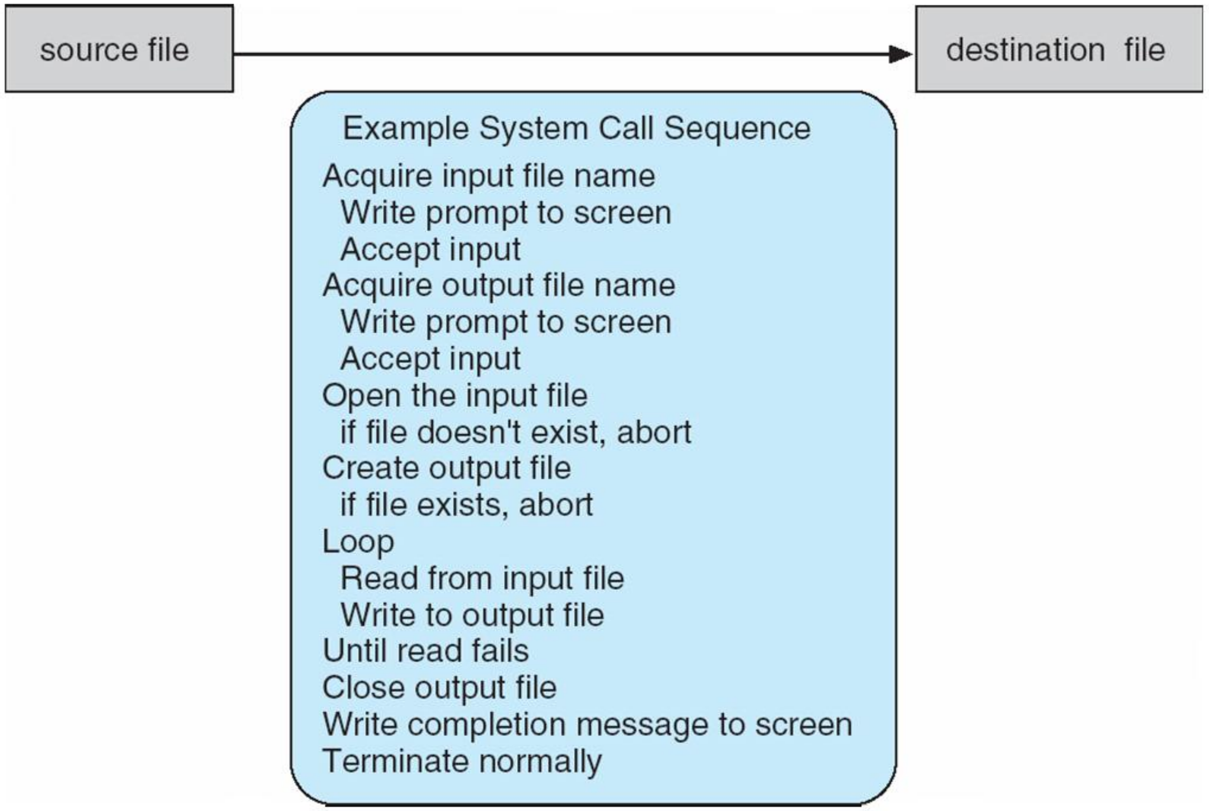
- Get time/date, set time/date (Ex: Mobile time setting)
- Get system data, set system data
- Get process, file/device attributes
- Set process, file/ device attributes

5. Communication

- Create, delete communication connection
- Send, receive messages
- Transfer status information
- Attach/detach remote devices

Example of System Calls

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

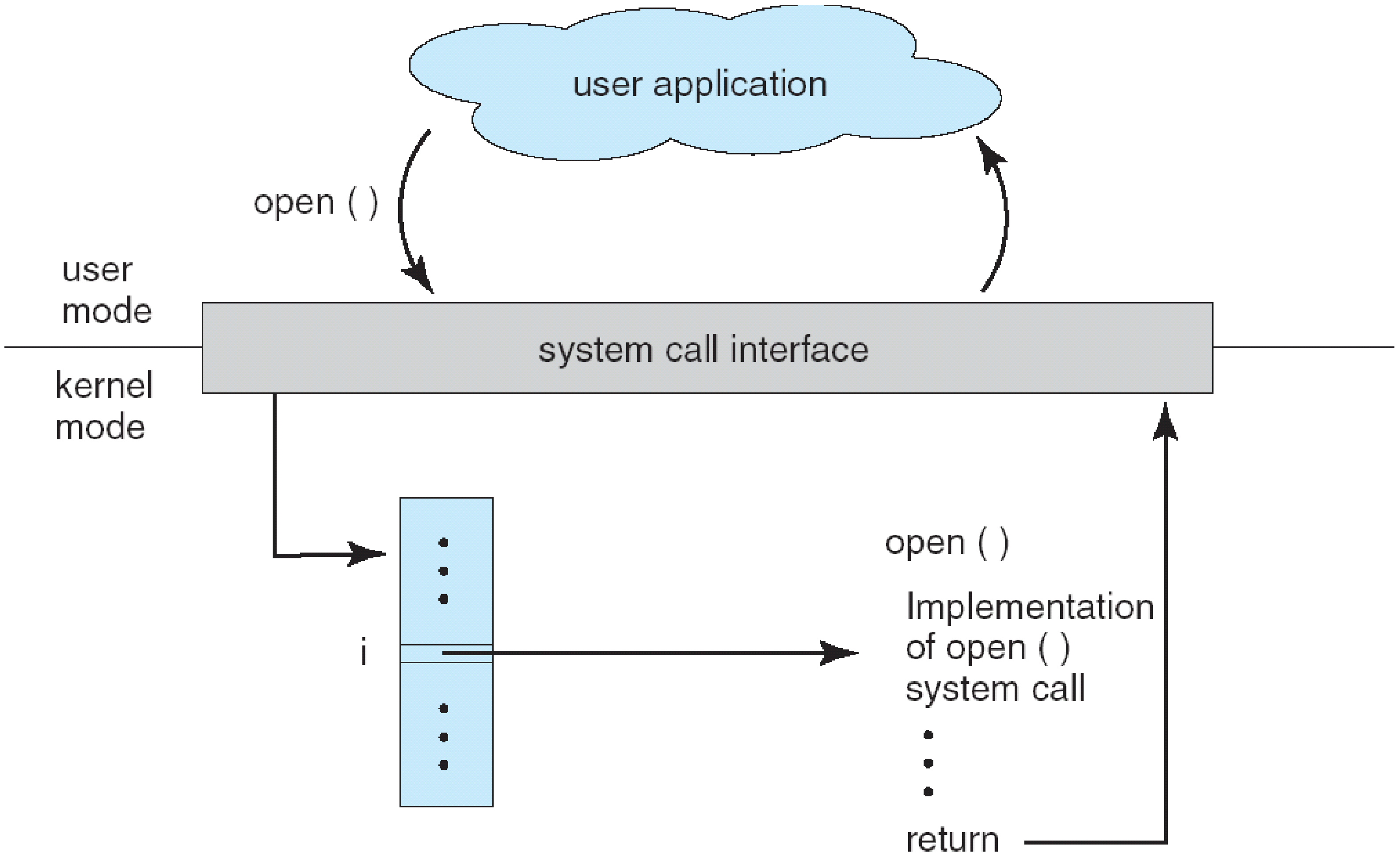


```
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

System Call Implementation

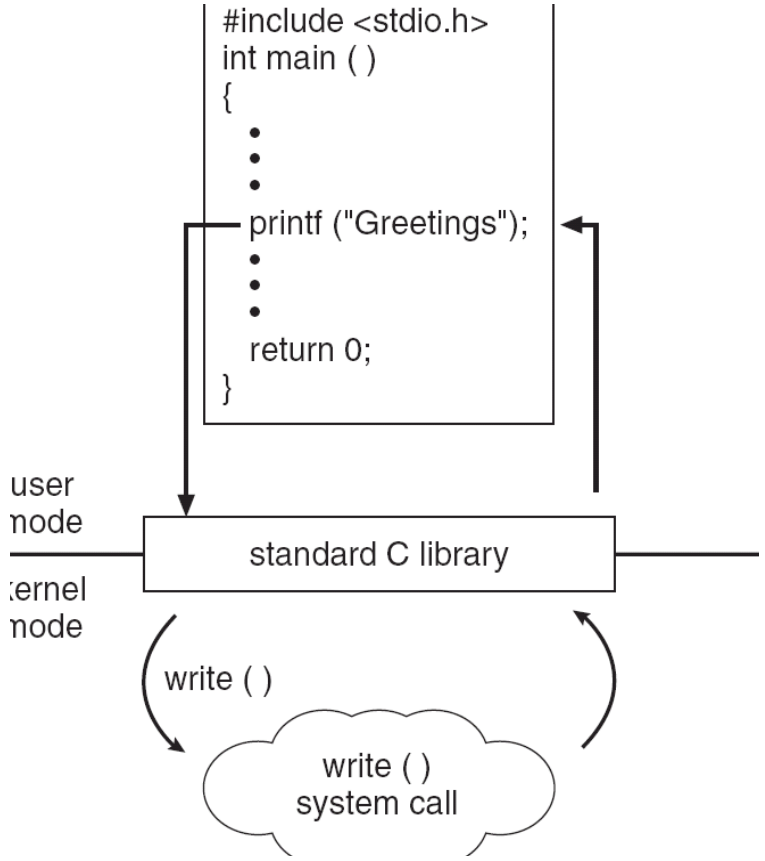
- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

API – System Call – OS Relationship



Standard C Library Example

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

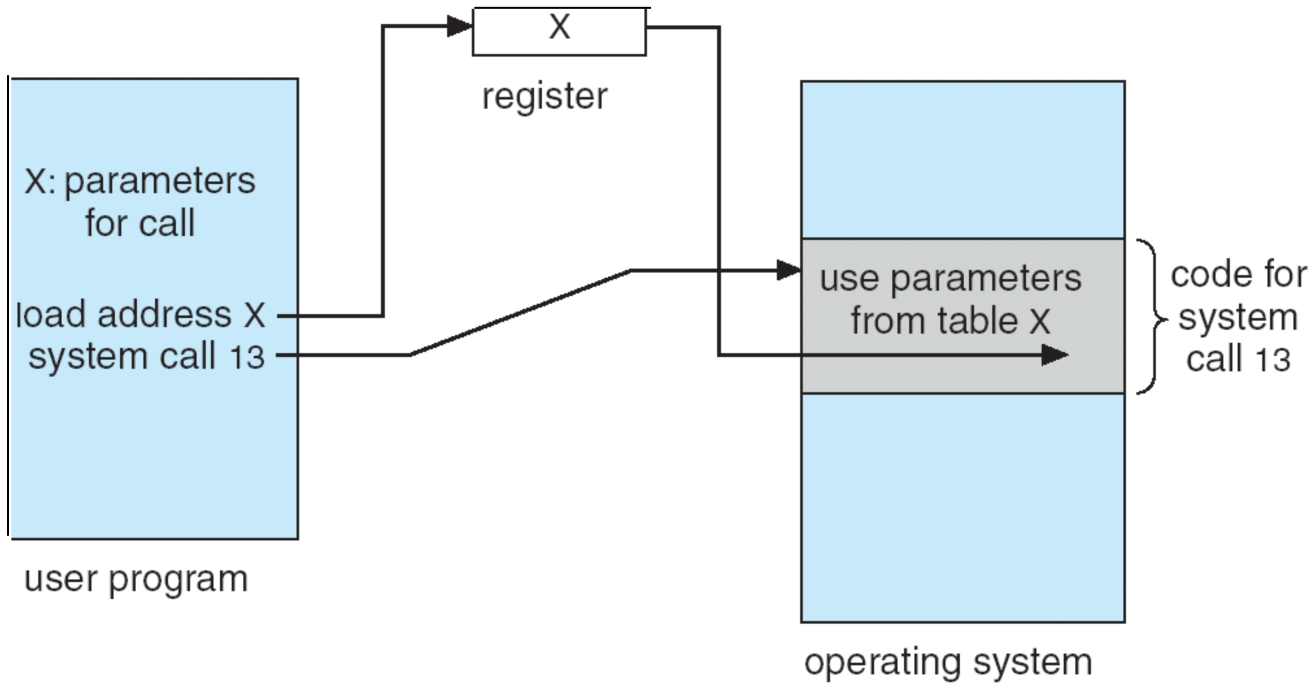


```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- 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 passed as a parameter in a register
    - This approach taken by Linux and 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

Parameter Passing via Table



Operating Systems Generation

- It is possible to design, code, and implement an OS specifically for one machine at one site (Embedded System). Commonly OS are designed to run on any class of machines at a variety of sites with a variety of peripheral configuration. The system must be configured for each specific computer site called as System generation (SYSGEN).
- SYSGEN program reads from a file/disk or ask operation for information concerning the specific configuration hardware system.

1. What CPU is to be busy? What options are installed? For multiple CPU systems each CPU must be described.
2. How much memory is available?
3. What devices are available? To know device characteristics.
4. What OS options are desired, what parameter values are to be used? I.e. How many buffers of which sizes should be used, what CPU Scheduling algorithm is desired, maximum no. of processors to be supported.

Once the information is defined, it can be used in several ways. The OS is completely compiled. Selection allows the library to contain the device drivers for all supported I/O devices, but only those needed are linked to the OS.