|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | 13CS2203 | - | OPERATING SYSTEMS | | | | | | | | |
|  |  | |  | | | | |
| Hours / Week | : | 4 | |  | Sessional Marks | : | 40 |
| Credits | : | 4 | |  | End Examination Marks | : | 60 |

|  |
| --- |
| **UNIT - I** |
| Introduction, Definition, views, OS structure, operations.  **OS Concepts**: Process, Memory and Storage Management, Protection & Security, Computing Environments.  **System Structures**: OS services, interfaces, system calls & types, OS design & Implementation, OS structures. |
|  |
| **UNIT – II** |
| **Process Concepts**: Process states, PCB, Process Scheduling, Operations, Interprocess communication.  **Multithreaded Programming**: Multithreading models, Thread libraries, Threading issues, Examples.  **CPU Scheduling**: Basic Concepts, Scheduling Criteria, Scheduling Algorithms, Disk Scheduling algorithms. |
|  |
| **UNIT – III** |
| **Process Synchronization**: The Critical-Section Problem, Semaphores, Monitors, Message Passing, Classical IPC problems (Readers-Writers, Dining philosophers and producer & consumer problems).  **Deadlocks**: Resources, Conditions for resource deadlocks, deadlock avoidance, deadlock prevention. Deadlock detection and recovery. |
|  |
| **UNIT – IV** |
| **Memory Management Techniques**: Introduction, swapping, Contiguous Memory Allocation, Paging, Structure of page table, Segmentation, Examples.  **Virtual Memory Management**: Introduction, Demand Paging, Copy on write, page replacement, Frame allocation, Thrashing, Memory Mapped Files, Kernel Memory allocation, Examples. |
|  |
| **UNIT – V** |
| **File-System Implementation**: File-System Structure, File-System Implementation Directory Implementation.  **I/O Systems**: Overview, I/O hardware, Kernel I/O subsystem  **Case Studies**: Linux, Windows XP. |
|  |
|  |
| TEXT BOOKS |
| 1. Silberschatz A, Galvin P B , Gagne G, Operating System Principles, 7th Edition, Wiley-India 2004 |
|  |
| REFERENCE BOOKS |
| 1. Tanenbaum AS, Modern Operating Systems, 3rd Edition, Pearson Education 2008.( for Interprocess Communication, Deadlocks, File Systems and Case studies) 2. Deitel HM, Deitel PJ and Choffnes DR, Operating Systems, 3rd Edition, Pearson Education 2004. 3. Stallings W, Operating Systems – Internals and Design Principles, 5th Edition, Prentice Hall of India 2005. |

## UNIT-1

## Introduction:

**1.1 What is an OperatingSystem?**

* An operating system is a program that manages the computerhardware.
* It also provides a basis for application programs and acts as anintermediary between a user of a computer and the computerhardware.
* The purpose of an operating system is to provide an environment in which a user can executeprograms.

## Goals of an OperatingSystem

* The primary goal of an operating system is thus to make thecomputer system convenient touse.
* The secondary goal is to use the computer hardware in an efficientmanner.

## Components of a ComputerSystem

* An operating system is an important part of almost every computersystem.
* A computer system can be divided roughly into fourcomponents.
  1. Hardware
  2. Operatingsystem
  3. The applicationprograms
  4. Users

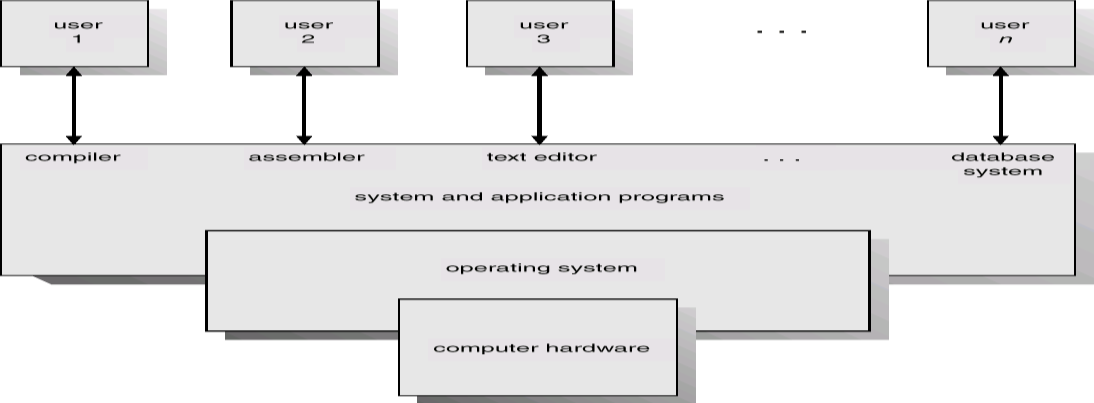


FIG1.1 : Abstract view of a components of a computer system

* Thehardware-thecentralprocessingunit**(CPU),**thememory,andthe Input/output **(I/O*)*** devices-provides the basic computingresources.
* The application programs- such as word processors, spreadsheets,compilers, andwebbrowsers-definethewaysinwhichtheseresourcesareusedto solve the computing problems of theusers.
* An operating system is similar to a *government. The* OS simply providesan environment within which other programs can do usefulwork.

## Abstract view of the components of a computersystem.

* Operating system can be viewed as a resourceallocator.
* TheOSactsasthemanageroftheresources(suchasCPUtime,memory space, file storage space, I/O devices) and allocates them tospecific programs and users as necessary fortasks.
* Anoperatingsystemisacontrolprogram.Itcontrolstheexecutionofuser programs to prevent errors and improper use ofcomputer.

**1.2 Views:**

Operating System is designed both by taking **user view** and **system view** into consideration.

Below is what the users and system thinks about Operating System.

### User View

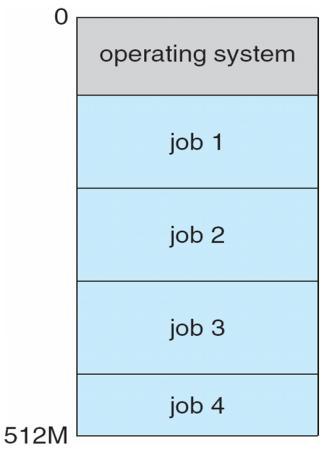
The user’s view of the computer varies according to the interface being used.

1. The goal of the Operating System is to maximize the work and minimize the effort of the user.
2. Most of the systems are to be operated by single user, however in some systems multiple users can share resources, memory. In these cases Operating System is designed to handle available resources among multiple users and CPU efficiently.
3. Operating System must be designed by taking both usability and efficient resource utilization into view.
4. In embedded systems (Automated systems) user view is not present.
5. Operating System gives an effect to the user as if the processor is dealing only with the current task, but in background processor is dealing with several processes.

### System View

1. From the system point of view Operating System is a program involved with the hardware.
2. Operating System is allocator, which allocate memory, resources among various processes. It controls the sharing of resources among programs.
3. It prevents improper usage, error and handles deadlock conditions.
4. It is a program that runs all the time in the system in the form of Kernel.
5. It controls application programs that are not part of Kernel
   1. **OS structure:**
6. 

* **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 userscan 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
* **Memory Layout for Multiprogrammed System**



**Fig: Memory Layout for Multiprogrammed System**

* 1. **Operating system operations**

Interrupt-driven nature of modern OSes requires that erroneous processes not be able to disturb anything else.

**1.4.1 Dual-Mode and Multimode Operation**

* User mode when executing harmless code in user applications
* Kernel mode ( a.k.a. system mode, supervisor mode, privileged mode ) when executing potentially dangerous code in the system kernel.
* Certain machine instructions ( privileged instructions ) can only be executed in kernel mode.
* Kernel mode can only be entered by making system calls. User code cannot flip the mode switch.
* Modern computers support dual-mode operation in hardware, and therefore most modern OSes support dual-mode operation.

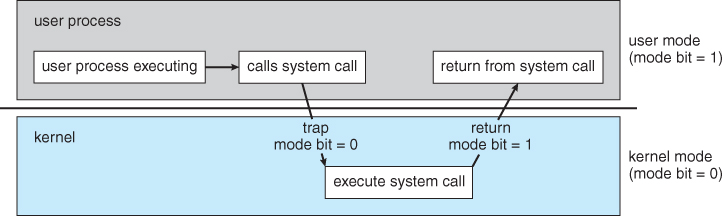


Figure 1.6 - Transition from user to kernel mode

* The concept of modes can be extended beyond two, requiring more than a single mode bit
* CPUs that support virtualization use one of these extra bits to indicate when the virtual machine manager, VMM, is in control of the system. The VMM has more privileges than ordinary user programs, but not so many as the full kernel.
* System calls are typically implemented in the form of software interrupts, which causes the hardware's interrupt handler to transfer control over to an appropriate interrupt handler, which is part of the operating system, switching the mode bit to kernel mode in the process. The interrupt handler checks exactly which interrupt was generated, checks additional parameters ( generally passed through registers ) if appropriate, and then calls the appropriate kernel service routine to handle the service requested by the system call.
* User programs' attempts to execute illegal instructions ( privileged or non-existent instructions ), or to access forbidden memory areas, also generate software interrupts, which are trapped by the interrupt handler and control is transferred to the OS, which issues an appropriate error message, possibly dumps data to a log ( core ) file for later analysis, and then terminates the offending program.

**1.4.2 Timer**

* Before the kernel begins executing user code, a timer is set to generate an interrupt.
* The timer interrupt handler reverts control back to the kernel.
* This assures that no user process can take over the system.
* Timer control is a privileged instruction, ( requiring kernel mode. )

**OS Concepts:**

**1.5 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



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 user, some operating system running concurrently on one or more CPUs



Concurrency by multiplexing the CPUs among the processes / threads

An OS is responsible for the following tasks with regards to process management:

* Creating and deleting both user and system processes
* Ensuring that each process receives its necessary resources, without interfering with other processes.
* Suspending and resuming processes
* Process synchronization and communication
* Deadlock handling

### Memory Management

An OS is responsible for the following tasks with regards to memory management:

* Keeping track of which blocks of memory are currently in use, and by which processes.
* Determining which blocks of code and data to move into and out of memory, and when.
* Allocating and deallocating memory as needed. ( E.g. new, malloc )

### 1.7 Storage Management

#### 1.7.1 File-System Management

An OS is responsible for the following tasks with regards to filesystem management:

* Creating and deleting files and directories
* Supporting primitives for manipulating files and directories. ( open, flush, etc. )
* Mapping files onto secondary storage.
* Backing up files onto stable permanent storage media.

#### 1.7.2 Mass-Storage Management

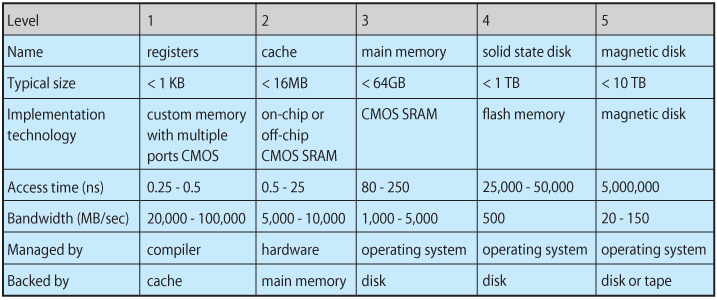
An OS is responsible for the following tasks with regards to mass-storage management:

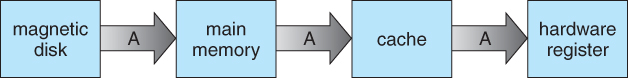
* Free disk space management
* Storage allocation
* Disk scheduling

Note the trade-offs regarding size, speed, longevity, security, and re-writability between different mass storage devices, including floppy disks, hard disks, tape drives, CDs, DVDs, etc.

#### 1.7.3 Caching

* There are many cases in which a smaller higher-speed storage space serves as a cache, or temporary storage, for some of the most frequently needed portions of larger slower storage areas.
* The hierarchy of memory storage ranges from CPU registers to hard drives and external storage. ( See table below. )
* The OS is responsible for determining what information to store in what level of cache, and when to transfer data from one level to another.
* The proper choice of cache management can have a profound impact on system performance.
* Data read in from disk follows a migration path from the hard drive to main memory, then to the CPU cache, and finally to the registers before it can be used, while data being written follows the reverse path. Each step ( other than the registers ) will typically fetch more data than is immediately needed, and cache the excess in order to satisfy future requests faster. For writing, small amounts of data are frequently buffered until there is enough to fill an entire "block" on the next output device in the chain.
* The issues get more complicated when multiple processes ( or worse multiple computers ) access common data, as it is important to ensure that every access reaches the most up-to-date copy of the cached data ( amongst several copies in different cache levels. )

  
**Figure 1.11 - Performance of various levels of storage**

  
**Figure 1.7 - Migration of integer A from disk to register**

#### 1.7.4 I/O Systems

The I/O subsystem consists of several components:

* A memory-management component that includes buffering, caching, and spooling.
* A general device-driver interface.
* Drivers for specific hardware devices.
* ( UNIX implements multiple device interfaces for many types of devices, one for accessing the device character by character and one for accessing the device block by block. These can be seen by doing a long listing of /dev, and looking for a "c" or "b" in the first position. You will also note that the "size" field contains two numbers, known as the major and minor device numbers, instead of the normal one. The major number signifies which device driver handles I/O for this device, and the minor number is a parameter passed to the driver to let it know which specific device is being accessed. Where a device can be accessed as either a block or character device, the minor numbers for the two options usually differ by a single bit. )

### 1.8Protection and Security

* ***Protection*** involves ensuring that no process access or interfere with resources to which they are not entitled, either by design or by accident. ( E.g. "protection faults" when pointer variables are misused. )
* ***Security*** involves protecting the system from deliberate attacks, either from legitimate users of the system attempting to gain unauthorized access and privileges, or external attackers attempting to access or damage the system.

### 1.9 Computing Environments

#### 1.9.1 Traditional Computing

#### 1.9.2 Mobile Computing

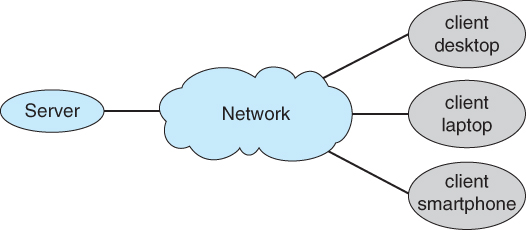
* Computing on small handheld devices such as smart phones or tablets. ( As opposed to laptops, which still fall under traditional computing. )
* May take advantage of additional built-in sensors, such as GPS, tilt, compass, and inertial movement.
* Typically connect to the Internet using wireless networking ( IEEE 802.11 ) or cellular telephone technology.
* Limited in storage capacity, memory capacity, and computing power relative to a PC.
* Generally uses slower processors, that consume less battery power and produce less heat.
* The two dominant OSes today are Google Android and Apple iOS.

#### 1.9.3 Distributed Systems

* Distributed Systems consist of multiple, possibly heterogeneous, computers connected together via a network and cooperating in some way, form, or fashion.
* Networks may range from small tight LANs to broad reaching WANs.
  + WAN = Wide Area Network, such as an international corporation
  + MAN =Metropolitan Area Network, covering a region the size of a city for example.
  + LAN =Local Area Network, typical of a home, business, single-site corporation, or university campus.
  + PAN = Personal Area Network, such as the bluetooth connection between your PC, phone, headset, car, etc.
* Network access speeds, throughputs, reliabilities, are all important issues.
* OS view of the network may range from just a special form of file access to complex well-coordinated network operating systems.
* Shared resources may include files, CPU cycles, RAM, printers, and other resources.

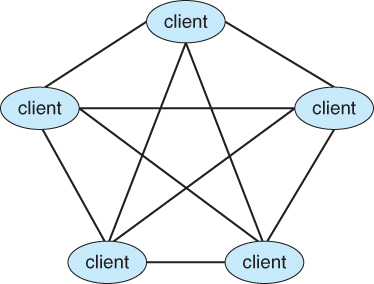
#### 1.9.4 Client-Server Computing

* A defined server provides services ( HW or SW ) to other systems which serve as clients. ( Technically clients and servers are processes, not HW, and may co-exist on the same physical computer. )
* A process may act as both client and server of either the same or different resources.
* Served resources may include disk space, CPU cycles, time of day, IP name information, graphical displays ( X Servers ), or other resources.

  
**Figure 1.8 - General structure of a client-server system**

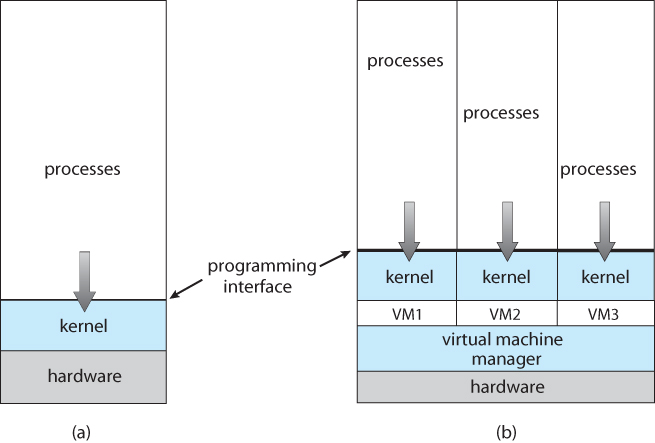
#### 1.9.5 Peer-to-Peer Computing

* Any computer or process on the network may provide services to any other which requests it. There is no clear "leader" or overall organization.
* May employ a central "directory" server for looking up the location of resources, or may use peer-to-peer searching to find resources.
* E.g. Skype uses a central server to locate a desired peer, and then further communication is peer to peer.

  
**Figure 1.9 - Peer-to-peer system with no centralized service**

#### 1.9.6 Virtualization

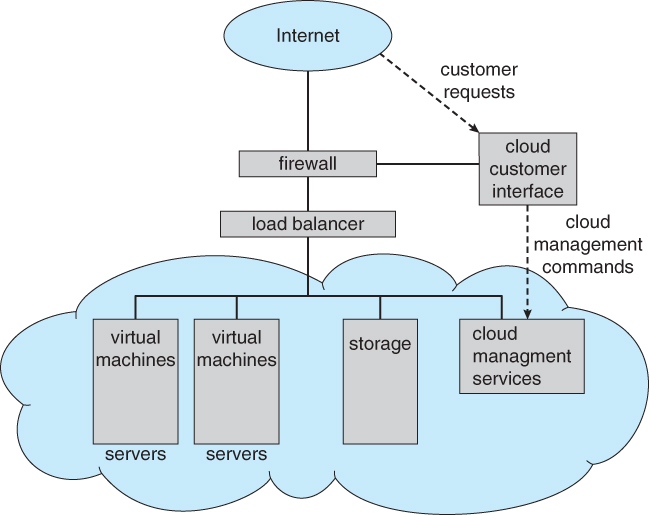
* Allows one or more "guest" operating systems to run on virtual machines hosted by a single physical machine and the virtual machine manager.
* Useful for cross-platform development and support.
* For example, a student could run UNIX on a virtual machine, hosted by a virtual machine manager on a Windows based personal computer. The student would have full root access to the virtual machine, and if it crashed, the underlying Windows machine should be unaffected.
* System calls have to be caught by the VMM and translated into ( different ) system calls made to the real underlying OS.
* Virtualization can slow down program that have to run through the VMM, but can also speed up some things if virtual hardware can be accessed through a cache instead of a physical device.
* Depending on the implementation, programs can also run simultaneously on the native OS, bypassing the virtual machines.

  
**Figure 1.10 - VMWare**

* To run Linux on a Windows system using VMware, follow these steps:
  1. Download the free "VMware Player" tool from <http://www.vmware.com/download/player> and install it on your system
  2. Choose a Linux version from among hundreds of virtual machine images at <http://www.vmware.com/appliances>
  3. Boot the virtual machine within VMware Player.

#### 1.9.7 Cloud Computing

* Delivers computing, storage, and applications as a service over a network.
* Types of cloud computing:
  + Public cloud - Available to anyone willing to pay for the service.
  + Private cloud - Run by a company for internal use only.
  + Hybrid cloud - A cloud with both public and private components.
  + Software as a Service - SaaS - Applications such as word processors available via the Internet
  + Platform as a Service - PaaS - A software stack available for application use, such as a database server
  + Infrastructure as a Service - IaaS - Servers or storage available on the Internet, such as backup servers, photo storage, or file storage.
  + Service providers may provide more than one type of service
* Clouds may contain thousands of physical computers, millions of virtual ones, and petabytes of total storage.
* Web hosting services may offer ( one or more ) virtual machine(s) to each of their clients.

  
**Figure 1.11 - Cloud computing**

#### 1.9.8 Real-Time Embedded Systems

* Embedded into devices such as automobiles, climate control systems, process control, and even toasters and refrigerators.
* May involve specialized chips, or generic CPUs applied to a particular task. ( Consider the current price of 80286 or even 8086 or 8088 chips, which are still plenty powerful enough for simple electronic devices such as kids toys. )
* Process control devices require real-time ( interrupt driven ) OSes. Response time can be critical for many such devices.

**System Structures**:

**1.10 Operating System Services**

**Program execution** – system capability to load a program into memory and to run it.

**I/O operations** – since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.

**File-system manipulation** – program capability to read, write, create, and delete files.

**Communications** – exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via shared memory or message passing.

**Error detection** – ensure correct computing by detecting errors in the CPU and memory hardware, in I/O

devices, or in user programs.

Additional Operating System Functions

Additional functions exist not for helping the user, but rather for ensuring efficient system operations.

1**. Resource allocation** – allocating resources to multiple users or multiple jobs running at the same time.

2**. Accounting** – keep track of and record which users use how much and what kinds of computer

resources for account billing or for accumulating usage statistics.

3. **Protection** – ensuring that all access to system resources is controlled.

**1.11 Interfaces:**

User interface is used to interact with the computer to performs various tasks. User gives commands to computer and enters the data into computer. These are examples of user interfacing. The operating system plays the main role for interfacing between user and computer. The input devices like keyboard and mouse are commonly used for giving commands to computer.

**Types of Operating Systems**

Based on the user interface, there are two types of operating systems.

**Graphical User Interface Operating System**

**Command Line Operating System**

**1- Graphical User Interface Operating System**

Graphical User Interface (GUI) operating system presents commands in graphical form. For example, application programs, commands, disk drives, files etc. are presented in the form of icons. Usually a command is given to the computer by clicking with mouse on the icon. GUI also provides menus, buttons and other graphical objects to the user to perform different tasks. GUI is very easy to interact with the computer.

**Example**

Examples of GUI operating systems are Windows, Linux, and Solaris. Today Windows is commonly used in PCs. In Windows, mouse is used as input device.



**Features of Graphical user interface OS:**

**Interfacing**

It provides commands in graphical form on the computer screen. The user gives commands to computer by checking with mouse on the icon. The users have not to memorize commands. Usually mouse is used for interfacing with computer.

**Control**

Although a GUI offers a better control of a file system and computer resources but often users have to use command line to complete a specific tasks.

***Ease***

It is easy to learn and use.

**Multitasking**

GUI provides facility to open multiple programs each in a separate window. So it enables a user to view, and to manipulate multiple things at a time on computer screen.

**Speed**

A GUI is easier to use. However, it is slower to perform different tasks.

**Scripting**

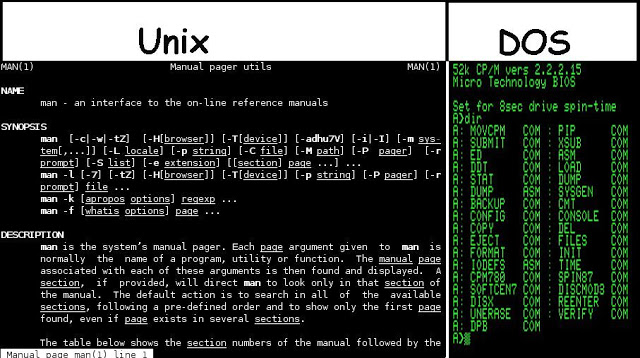
Although a GUI enables a user to create shortcuts, or other similar actions to complete a task. However, GUI does not Features of Graphical user interface OS:

**2- Command Line Operating System**

A command line operating system provides a command prompt on the computer screen. The commands are given to the computer by typing on the keyboard. The commands are typed according to the predefined format. The users have to memorize commands and rules of writing these commands. It is not an easy way to interface with the computer.

Example

Examples of Command line operating systems are DOS (Disk Operating System), and Unix etc.



**Features of Command Line OS:**

**Interfacing**

It provides a commands prompt on the computer screen. The user gives commands to computer by typing on the keyboard. The users have to memorize commands and rules of writing these commands. Usually keyboard is used for interfacing with computer.

**Control**

It provides full access to computer resources.

**Ease**

It is difficult to learn and use.

**Multitasking**

Although many command line operating systems allow multitasking, but it is difficult in these operating systems to view multiple things at a time on computer screen.

**Speed**

The command line interface is faster than GUI to perform different tasks.

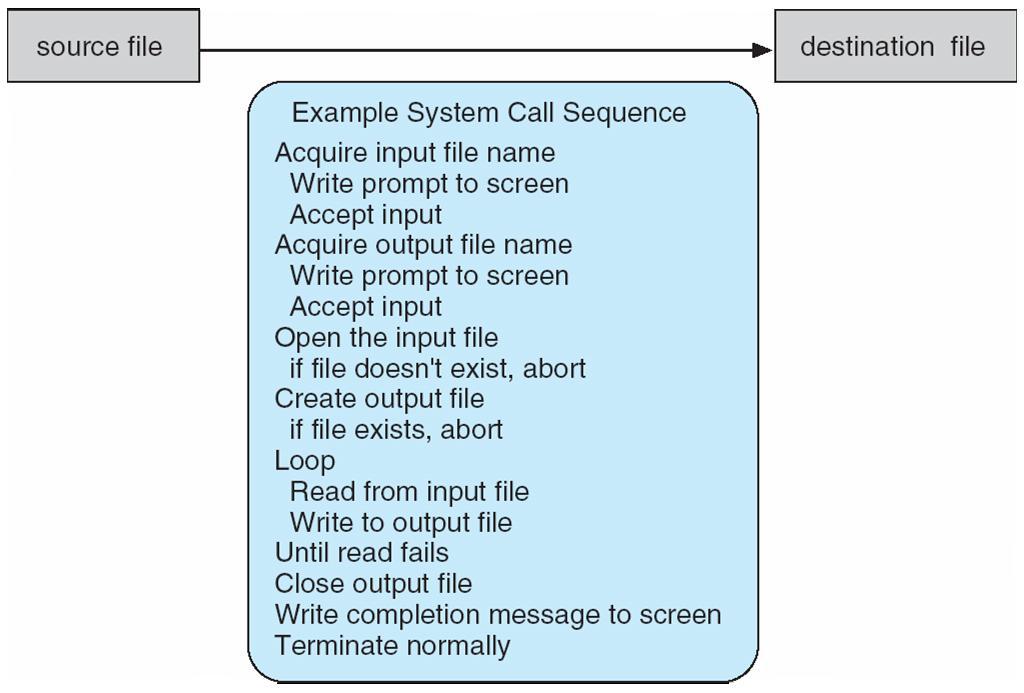
**Scripting**

A command line interface enables a user to easily script a sequence of commands to perform a tasks.

**1.12 system calls & types**

**System Calls**

* System calls provide the interface between a process and the operating system.
* These calls are generally available as assembly-language instructions.

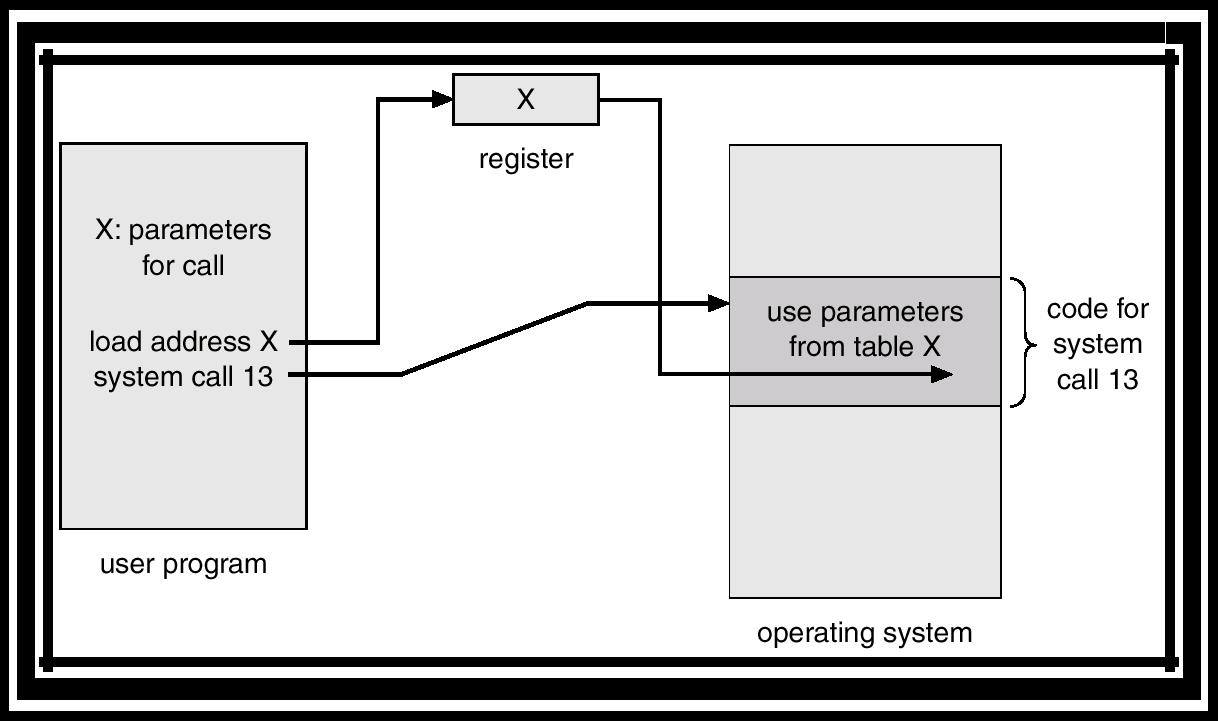


**Passing parameters to OS**

Three general approaches are used to pass parameters to OS.

1. Pass parameters in registers.
2. In cases, where there may be more parameters than registers, the parameters are generally stored in a block or table in memory, and address of block is passed as parameter in register.
3. Parameters can also be placed, or *pushed,* onto the *stack* by the program, and *popped o f*the stack by the operating system.

**Passing of parameters as a table.**



**System Calls**



Programming interface to the services provided by the OS Typically written in a high-level language (C or C++)

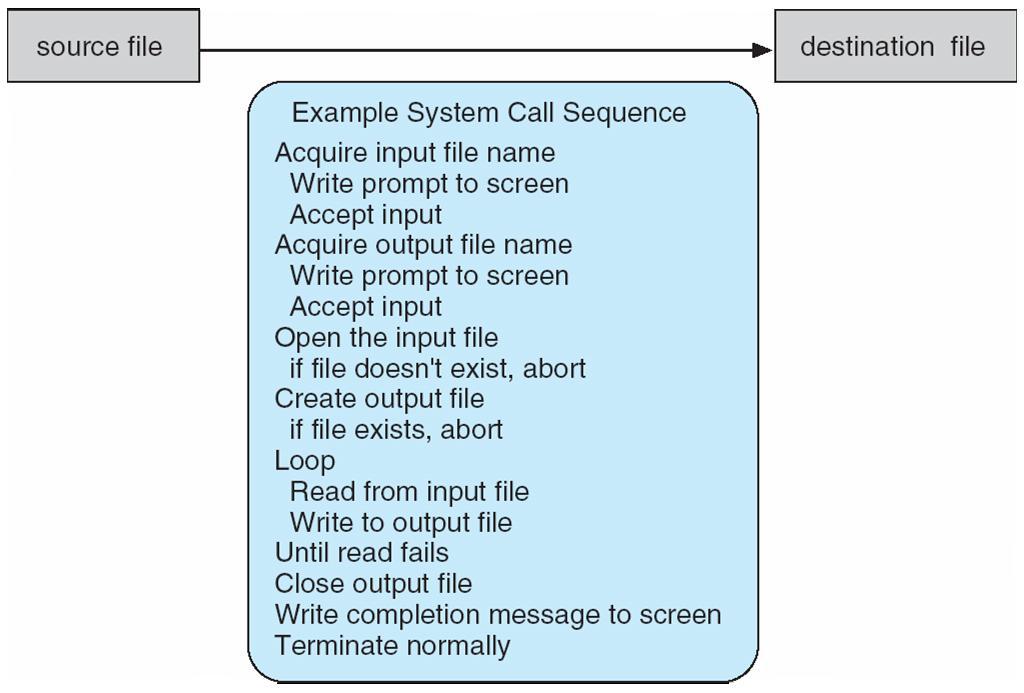
Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call usenThree most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)



Why use APIs rather than system calls?(Note that the system-call names used throughout this text are generic)

**Example of System Calls**





**System Call Implementation**



Typically, a number associated with each system call System-call interface maintains a table indexed according to these Numbers

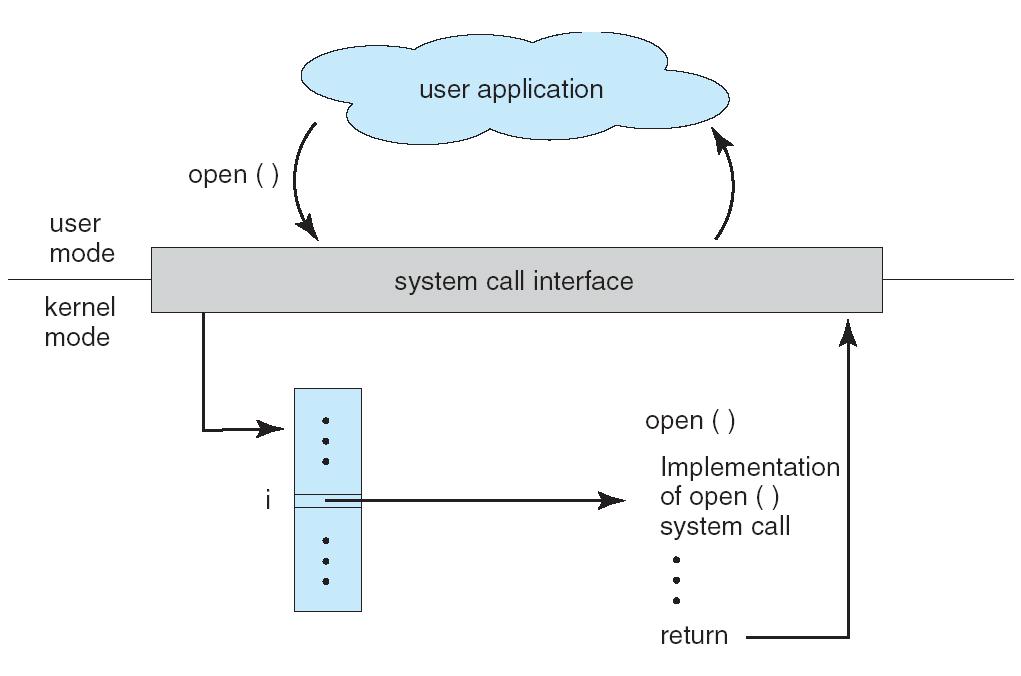
The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values

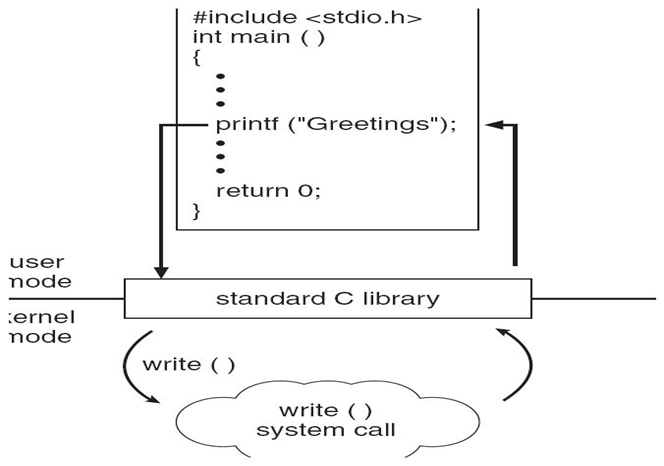
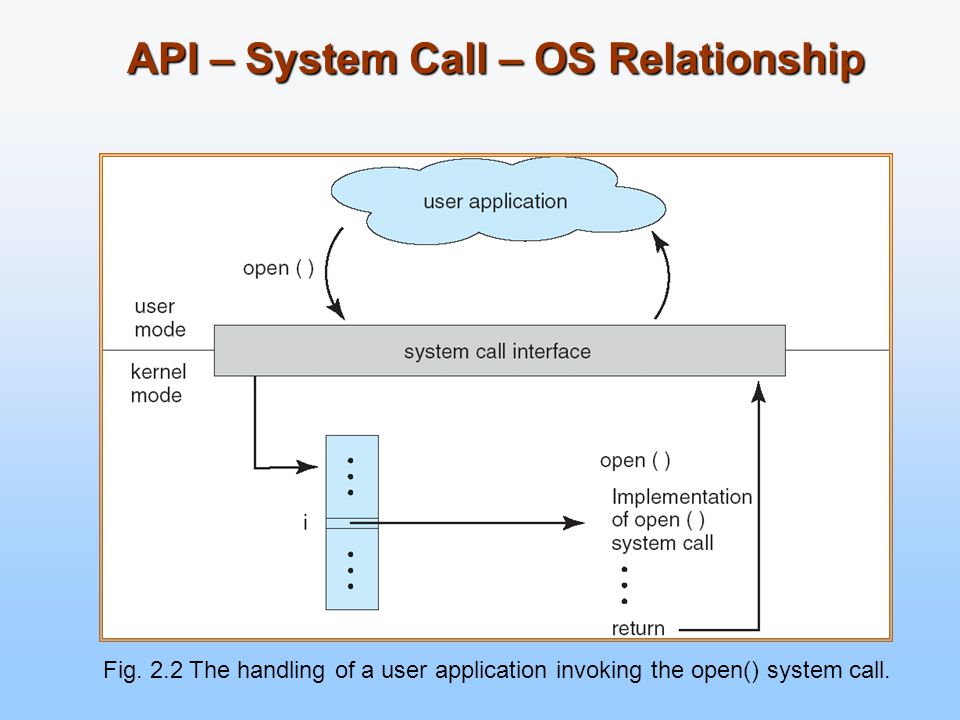


The caller need know nothing about how the system call is implemented Just needs to obey API and understand what OS will do as a result call Most details of OS interface hidden from programmer by API

Managed by run-time support library (set of functions built into libraries included with compiler) **API – System Call – OS Relationship**





******

**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

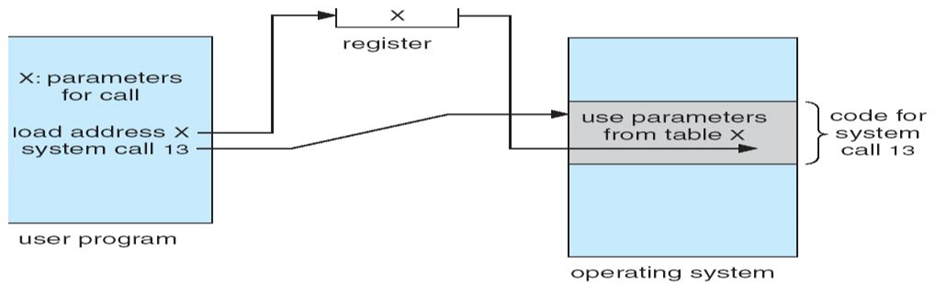


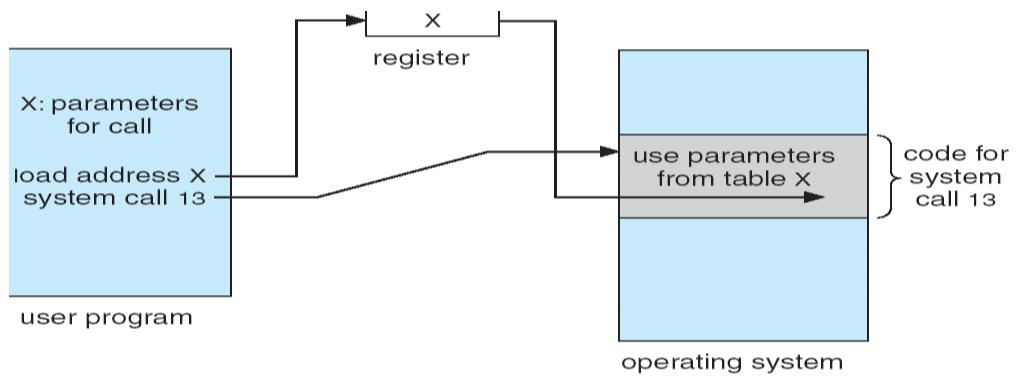
Parameters placed, or *pushed,* onto the *stack* by the program and *popped* off the stack by the operating system



Block and stack methods do not limit the number or length of parameters being passed

**Parameter Passing via Table**

******



***TYPES OF SYSTEM CALLS:***

* System calls can be grouped roughly into five major categories:
  1. Process control
  2. file management
  3. device management
  4. information maintenance
  5. communications

1. **Process Control**
   * end,abort
   * load, execute
   * Create process and terminate process
   * get process attributes and 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 attribtues, 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

**1.13 OS design & Implementation**



Design and Implementation of OS not “solvable”, but some approaches have proven successful



Internal structure of different Operating Systems can vary widely

Start by defining goals and specifications

Affected by choice of hardware, type of system

*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 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 is a very important principle, it allows maximum flexibility if policy decisions are to be changed later

**1.14 OS structures**



**Simple Structure**

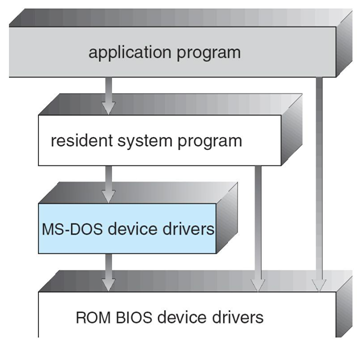
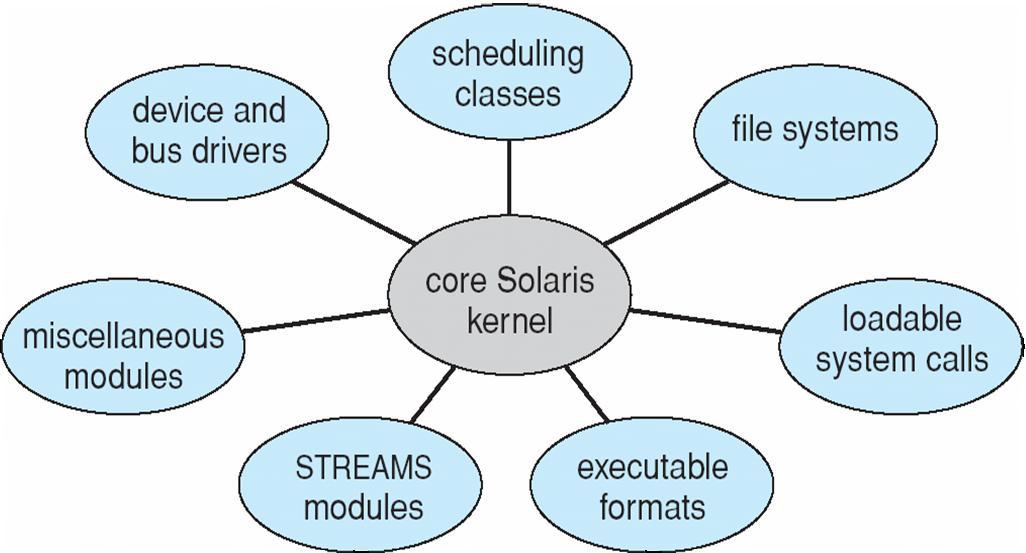
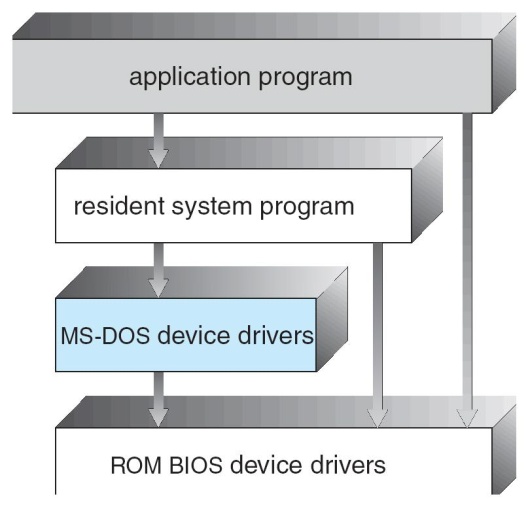
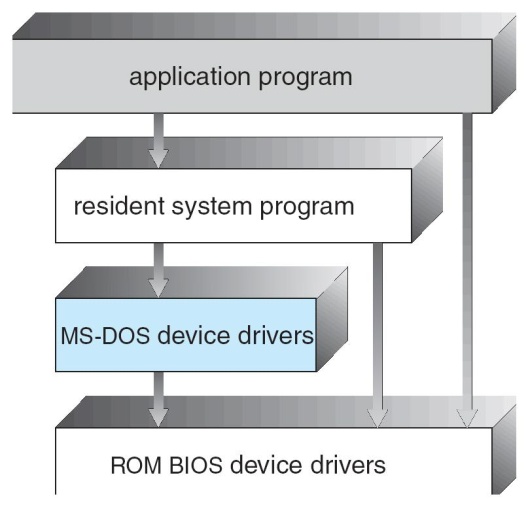


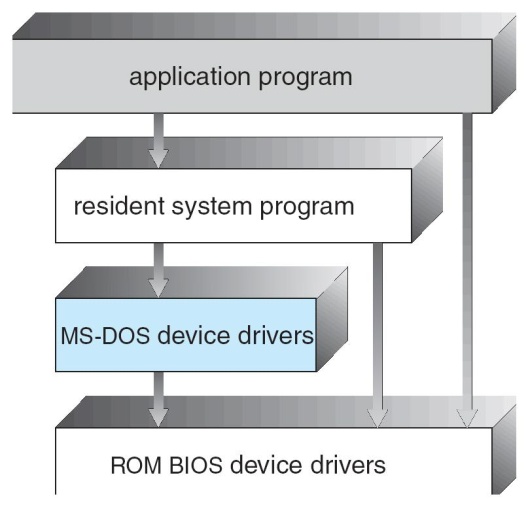
MS-DOS – written to provide the most functionality in the least space Not divided into modules



Although MS-DOS has some structure, its interfaces and levels of Functionality are not well separated

**MS-DOS Layer Structure**





**Layered Approach**

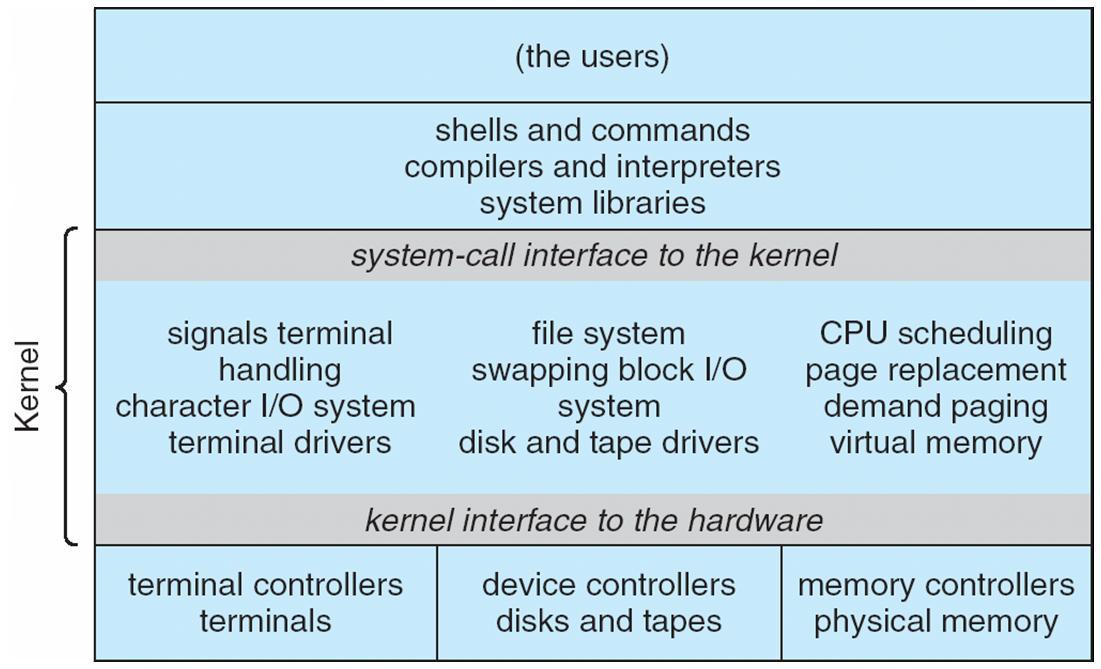


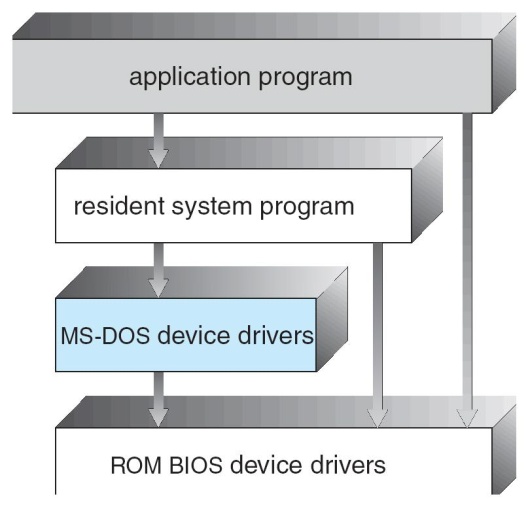
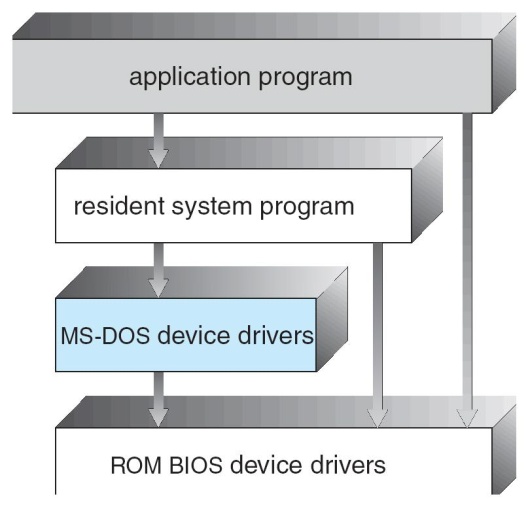
The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.



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

**Traditional UNIX System Structure**





**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

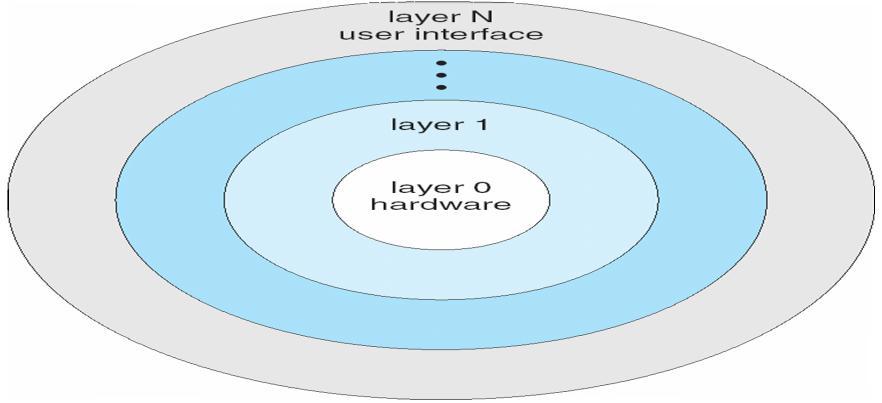


Consists of everything below the system-call interface and above the physical hardware Provides the file system, CPU scheduling, memory management, and other operating-system



functions; a large number of functions for one level

**Layered Operating System**



**Micro kernel System Structure**



Moves as much from the kernel into “*user*” space

Communication takes place between user modules using message passing



Benefits:

Easier to extend a microkernel

Easier to port the operating system to new architectures

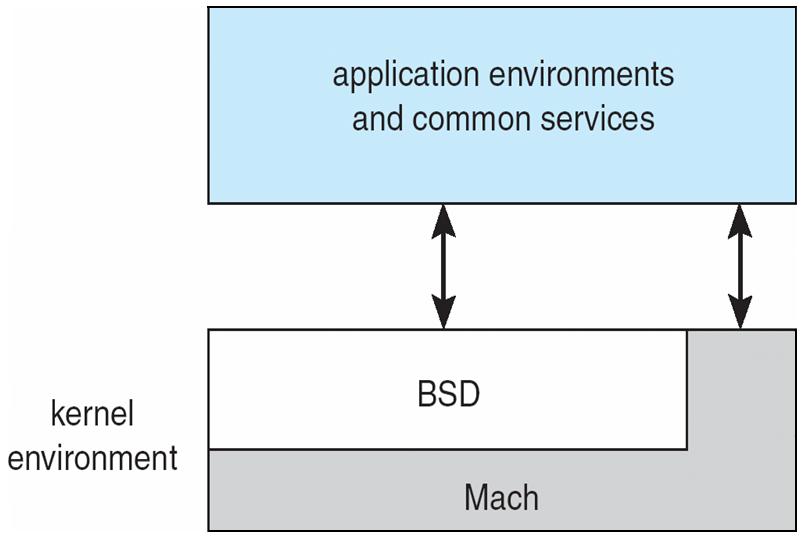
More reliable (less code is running in kernel mode

More secure

Detriments:

Performance overhead of user space to kernel space communication

**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**

