OPERATING SYSTEMS

II B .TECH,II-SEM CSE

UNIT-I

Operating Systems Overview: Operating system functions, Operating system structure, operating systems Operations, protection and security, Computing Environments, Open- Source Operating Systems

System Structures: Operating System Services, User and Operating-System Interface, systems calls, Types of System Calls, system programs, operating system structure, operating system debugging, System Boot.

Operating Systems Overview :

Computer software can be divided into two main categories:

1.Application Software
2. System Software

1. Application Software

Application software consists of the programs for performing tasks particular to the machine's utilization. This software is designed to solve a particular problem for users.

Examples of application software include spreadsheets , database systems, desktop publishing systems , program development software and games.

2. System Software

On the other hand **system software** is more transparent and less noticed by the typical computer user. This software provides a general programming environment in which programmers can create specific applications to suit their needs. This environment provides new functions that are not available at the hardware level and performs tasks related to executing the application programs. System software acts as an interface between the hardware of the computer and the application software that users need to run on the computer.

The most important type of system software is the Operating System.

Operating System : An Operating System is a program or system software that acts as an intermediary between a user of a computer and the computer hardware .

An **Operating System (OS)** is a collection of programs that acts as an interface between a user of a computer and the computer hardware.

- The purpose of an operating system is to provide an environment in which a user may execute their application programs.
- Operating Systems are viewed as resource managers. The main resource is the computer hardware in the form of processors, storage devices, input/output devices, communication devices and data.
- A program that acts as an intermediary between a user of a computer and the computer hardware

- Provides an interface between the computer hardware and the programmer that simplifies and makes feasible the creation ,coding ,debugging and maintenance of application programs .
- Computer system consisting of hardware software and data . The Operating System provides the means for proper use of these resources in the operation of the computer system .
- An Operating System is similar to government . Like a government it performs no useful function by itself . It simply provides an environment within which other programs can do useful work

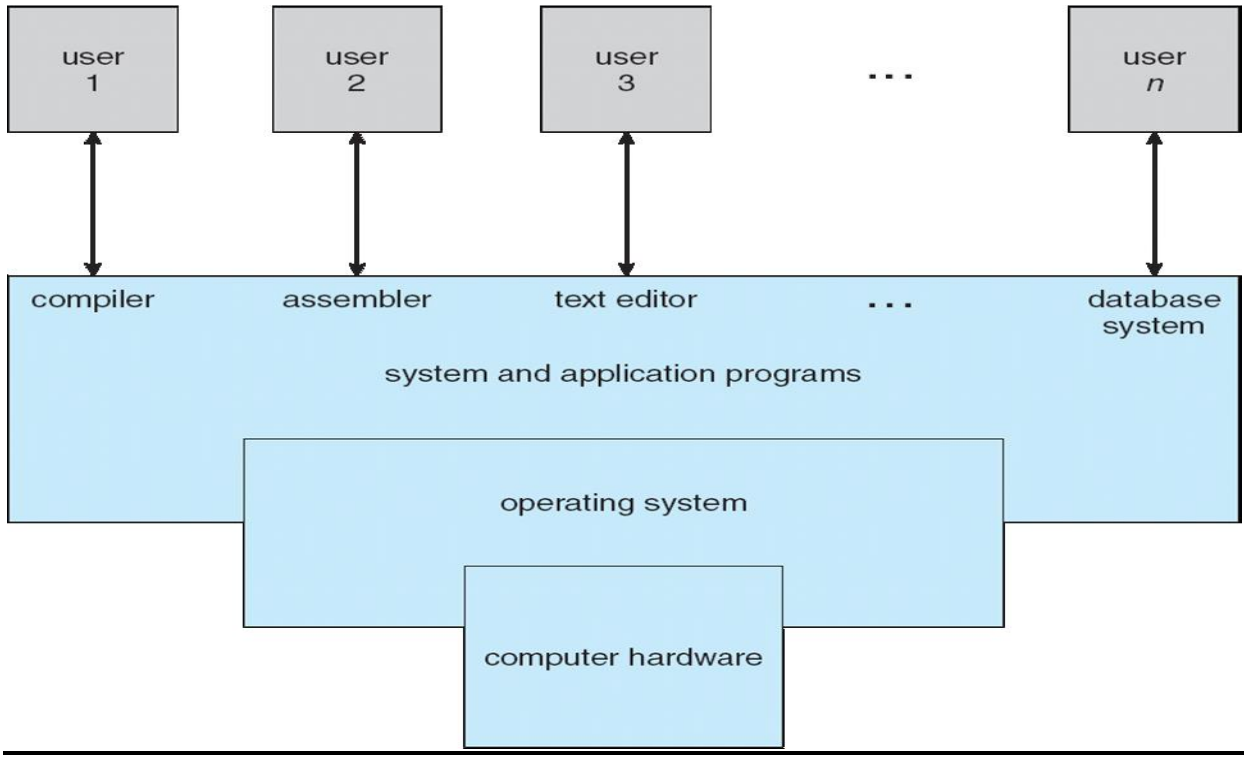
The purpose of an Operating System is to provide an environment in which users can execute their programs in a convenient and efficient manner.

Computer System Structure

Computer system can be divided into four components

- **Hardware** – provides basic computing resources CPU, memory, I/O devices
- **Operating system**-Controls and coordinates use of hardware among various applications and users
- **Application programs** – define the ways in which the system resources are used to solve the computing problems of the users-Word processors, compilers, web browsers, database systems, video games
- **Users**- People, machines, other computers

Four Components of a Computer System or Layered Architecture of Computer System



Two Views of Operating System

The Role of Operating System can be viewed in two ways

1. **User's View**
2. **System View**

User's View :

The user view of the computer depends on the interface being used

i) Some users may use PC's : Users sit in front of a PC Such systems are designed for one user to monopolize its resources .In these cases the operating system is designed mostly for ease of use (convenience) with some attention paid to performance and none paid to resource utilization.

ii) Some users may use a terminal connected to a mainframe or minicomputers : The multiple users share resources and may exchange information . In these cases the operating system is designed to maximize resource utilization-so that all available CPU time memory & I/O are used efficiently and no user takes more than his share.

iii)Some users sit at PC connected to networks of other workstations and servers: These users have dedicated resources at their disposal but they also share resources such as networking and servers-file compute and print servers. Therefore their operating system is designed to compromise between individual usability and resource utilization.

iv) Recently many varieties of handheld computers have come into fashion. Most of these devices are standalone units for individual users. Some are connected to networks either directly by wire or (more often) through wireless modems and networking. Because of power speed and interface limitations they perform relatively few remote operations. Their operating systems are designed mostly for individual usability but performance per unit of battery life is important as well.

v) Some computers have little or no user view. For example embedded computers in home devices and automobiles may have numeric keypads and may turn indicator lights on or off to show status but they and their operating systems are designed primarily to run without user intervention

System View

- From the computer's point of view operating system viewed as a **resource allocator.**

A computer system has many resources (hardware and software) that may be required to solve a problem: CPU time memory space files storage space input/output devices etc. The operating system acts as the manager of these resources and allocates them to specific programs and users as necessary for their tasks.

- From the computer's point of view operating system viewed as a **control program.**

The OS controls and co-ordinates the execution of user programs to prevent errors and improper use of the computer . It is especially concerned with the operation and control of I/O devices.

Operating System goals or Objectives :

There are two main goals of an Operating System :-

1. Convenience:- The primary goal of OS is to make computer system easier for user i.e. OS makes interaction b/w user and hardware.

2. Efficiency:- The secondary goal of OS is to allocate the system resources to various application program as efficient as possible.

The primary goal of some operating system is convenience for the user.

This exists because they are supposed to make it easier to compute them without them. This view is particularly clear when you look at Operating Systems for small PCs.

While the primary goal of some other operating system is efficient operation of the computer system. These are used for large shared multi user systems. These systems are expensive so it is desirable to make them as efficient as possible.

These two goals- convenience and efficiency are sometimes contradictory In past the efficiency was often more important than convenience. Thus much of the Operating System theory concentrates on optimal use of the computing resources.

1.1 Operating System Functions :

Following are some of important functions of an operating System

- Providing User Interface
- Memory Management
- Processor Management
- Process Management
- Device Management
- File Management

- Protection and Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

Providing User Interface

OS provides an interface to the user to interact with a computer as

- **Graphical user interface (GUI)**

  Most common interface

  Windows OS X, Gnome KDE

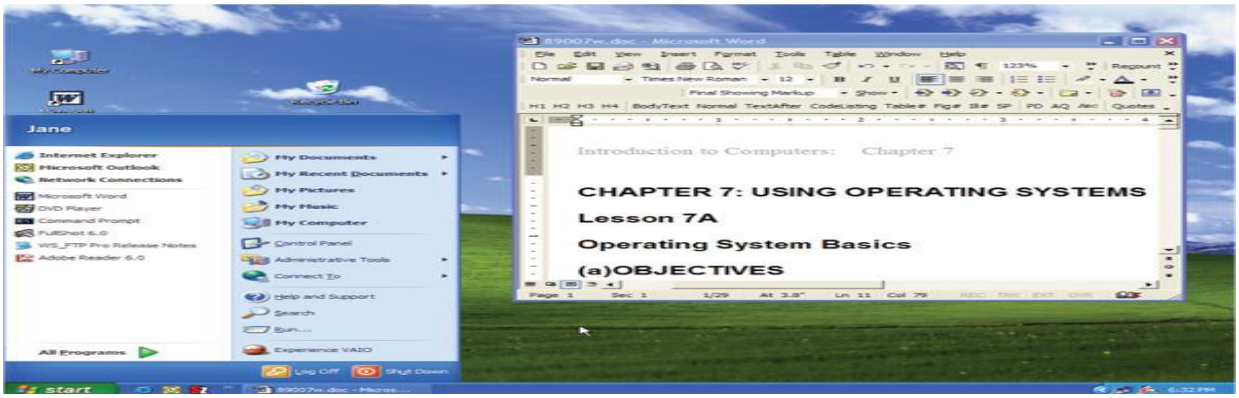
  Uses a mouse to control objects

  Uses a desktop

  Shortcuts to open programs or documents

  Task switching

  Dialog boxes allow directed input



- **Command line interfaces**

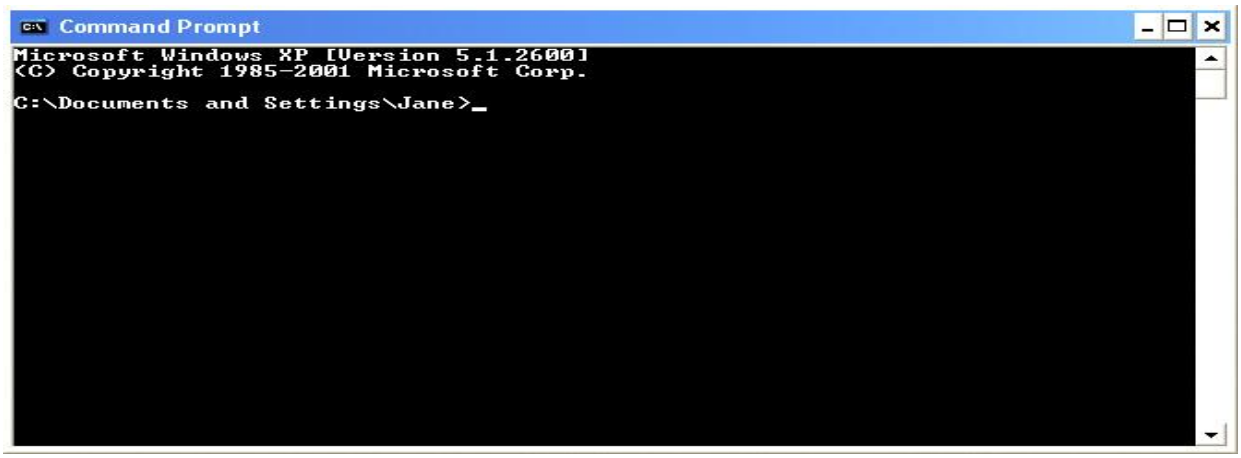
  Older interface

  DOS, Linux, UNIX

  User types commands at a prompt

  User must remember all commands

  Included in all GUIs



Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed it must be in the main memory.

An Operating System does the following activities for memory management :

- Keeps track of primary memory i.e. what part of it are in use by whom what part are not in use.
- In multiprogramming the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it .
- De-allocates the memory when a process no longer needs it or has been terminated.

Processor Management

In multiprogramming environment the OS decides which process gets the processor when and for how much time. This function is called processor or CPU Scheduling.

An Operating System does the following activities for processor management −

- Keeps tracks of processor and status of processor. The program responsible for this task is known as traffic controller.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.

Process Management

A process is a program in execution.
The OS is responsible for the following activities of the process management

• Creating & destroying of the user & system process .
• Allocating H/w resources among the processes.
• Controlling the progress of the process.
• Provides mechanisms for process synchronization and

communication.

• Provides mechanism for deadlock handling.

Device Management

An Operating System manages device communication via their respective drivers.

It does the following activities for device management:

- Keeps tracks of all devices. Program responsible for this task is known as the I/O controller.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

File Management

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directories .

The OS is responsible for the following activities of the **File Management**

- It helps to create new files in computer system and placing them at the specific locations.
- It helps in easily and quickly locating these files in computer system.
- It makes the process of sharing of the files among different users very easily and user friendly.
- It helps to stores the files in separate folders known as directories. These directories help users to search file quickly or to manage the files according to their types or uses.
- It helps the user to modify the data of files or to modify the name of the file in the directories.

Protection and Security

Security : It involves guarding of a user's data and programs against interference by external entities, **e.g**. unauthorized persons.

ex: Restricting the access of system with firewalls and authentication

Protection : It involves guarding a user's data and programs against interference by other authorized users of the system.

ex: Restricting the access of information with permissions

Control over system performance : Recording delays between request for a service and response from the system.

Job and Resource accounting − Keeping track of time and resources used by various jobs and users.

Error detecting aids − Production of dumps, traces, error messages, and other debugging and error detecting aids.

Coordination between other softwares and users − Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer system

Handling network communications : Providing communication b/w two computers

1.2 Operating System Structure

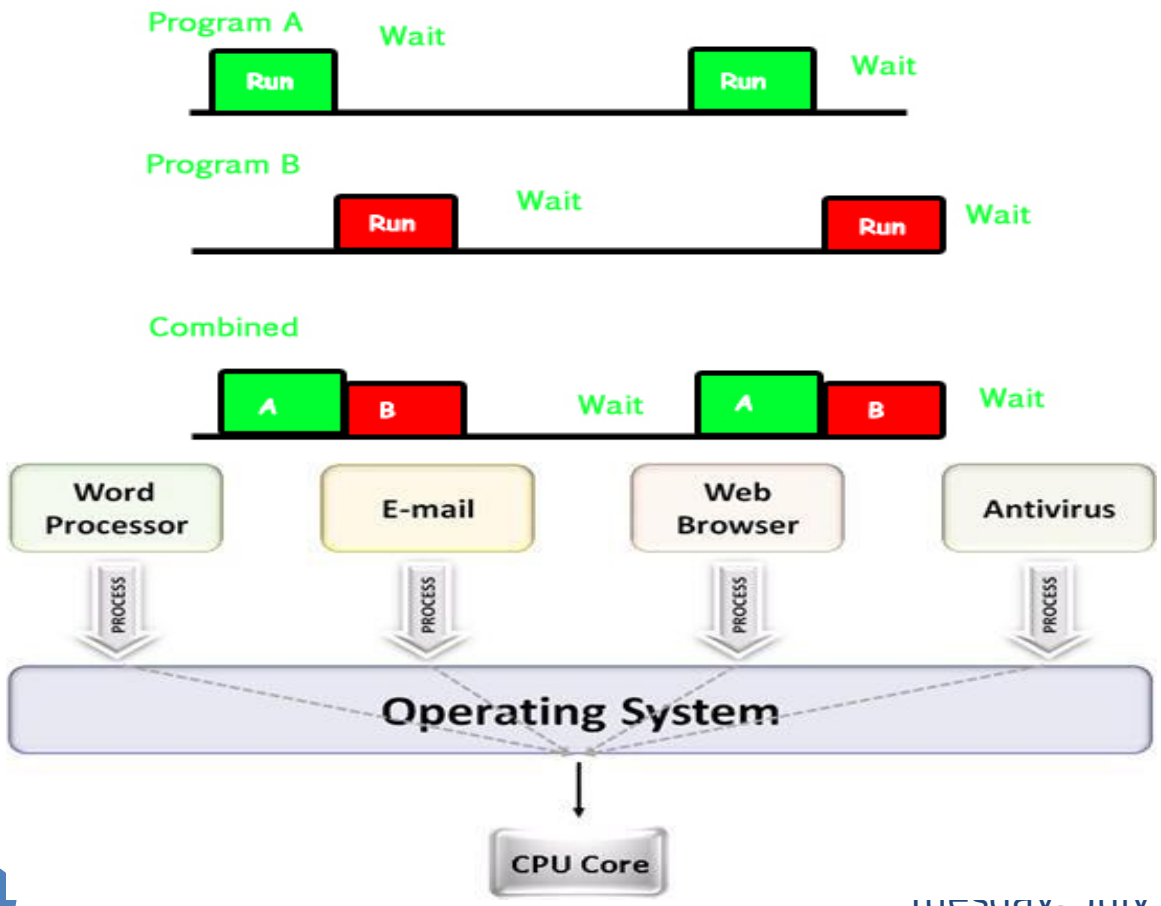
Non-multi programmed system's working –

- In a non multi programmed system, As soon as one job leaves the CPU and goes for some other task (say I/O), the CPU becomes idle. The CPU keeps waiting and waiting until this job (which was executing earlier) comes back and resumes its execution with the CPU. So CPU remains free for all this while.
- Now it has a drawback that the CPU remains idle for a very long period of time. Also, other jobs which are waiting to be executed might not get a chance to execute because the CPU is still allocated to the earlier job.
  This poses a very serious problem that even though other jobs are ready to execute, CPU is not allocated to them as the CPU is allocated to a job which is not even utilizing it (as it is busy in I/O tasks).
- It cannot happen that one job is using the CPU for say 1 hour while the others have been waiting in the queue for 5 hours. To avoid situations like this and come up with efficient utilization of CPU, the concept of multi programming came up.

The main idea of multi programming is to utilize the CPU efficiently .

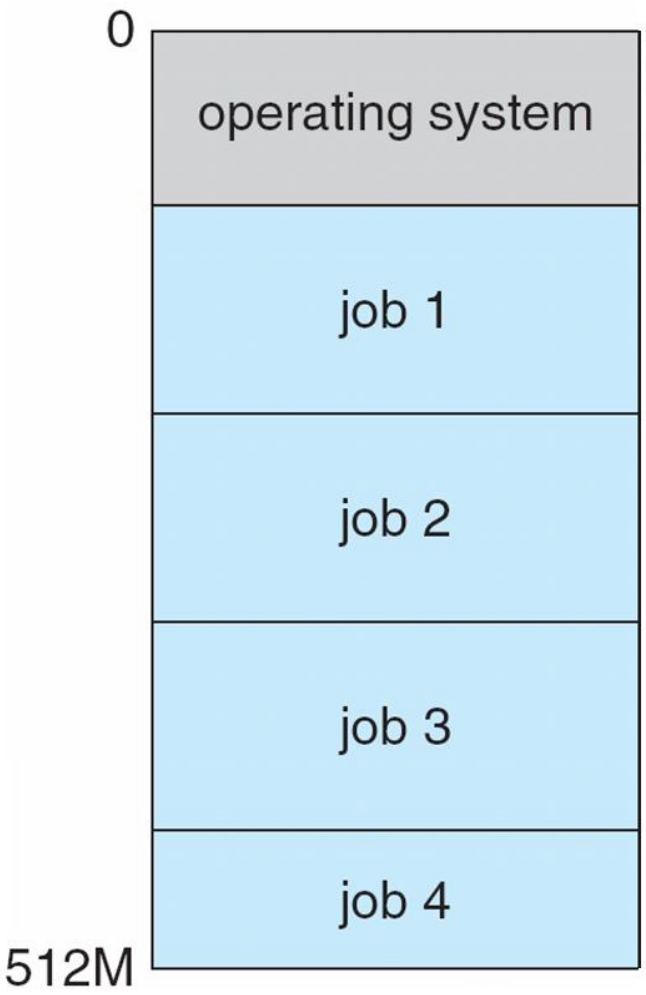
Multi programmed system's working –

- In a multi-programmed system, as soon as one job goes for an I/O task, the Operating System interrupts that job, chooses another job from the job pool (waiting queue), gives CPU to this new job and starts its execution.
- The previous job keeps doing its I/O operation while this new job does CPU bound tasks. Now say the second job also goes for an I/O task, the CPU chooses a third job and starts executing it. As soon as a job completes its I/O operation and comes back for CPU tasks, the CPU is allocated to it.
- In this way, no CPU time is wasted by the system waiting for the I/O task to be completed. Therefore, the ultimate goal of multi programming is to keep the CPU busy as long as there are processes ready to execute. This way, multiple programs can be executed on a single processor by executing a part of a program at one time, a part of another program after this, then a part of another program and so on, hence executing multiple programs. Hence, the CPU never remains idle.



31 Tuesday, July

Memory Layout for Multi programmed System



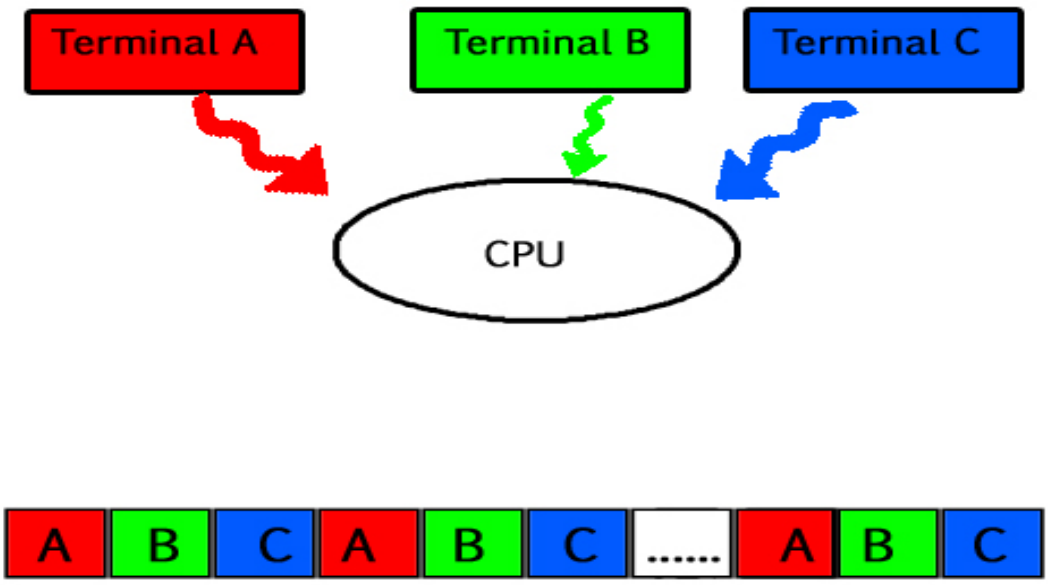
Multi tasking system's working –

- In a time sharing system, each process is assigned some specific quantum of time for which a process is meant to execute. Say there are 4 processes P1, P2, P3, P4 ready to execute. So each of them are assigned some time quantum for which they will execute e.g time quantum of 5 nanoseconds (5 ns). As one process begins execution (say P2), it executes for that quantum of time (5 ns). After 5 ns the CPU starts the execution of the other process (say P3) for the specified quantum of time.

- Thus the CPU makes the processes to share time slices between them and execute accordingly. As soon as time quantum of one process expires, another process begins its execution.

- Here also basically a context switch is occurring but it is occurring so fast that the user is able to interact with each program separately while it is running.

- This way, the user is given the illusion that multiple processes/ tasks are executing simultaneously. But actually only one process/ task is executing at a particular instant of time.



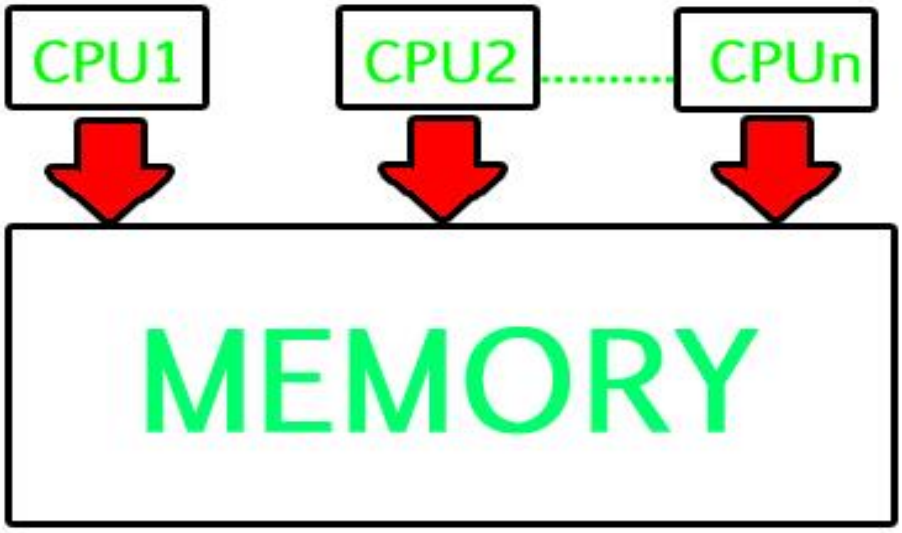
Multiprocessing

In a uni-processor system, only one process executes at a time. Multiprocessing is the use of two or more CPUs (processors) within a single Computer system. The term also refers to the ability of a system to support more than one processor within a single computer system.

Multi processing system's working

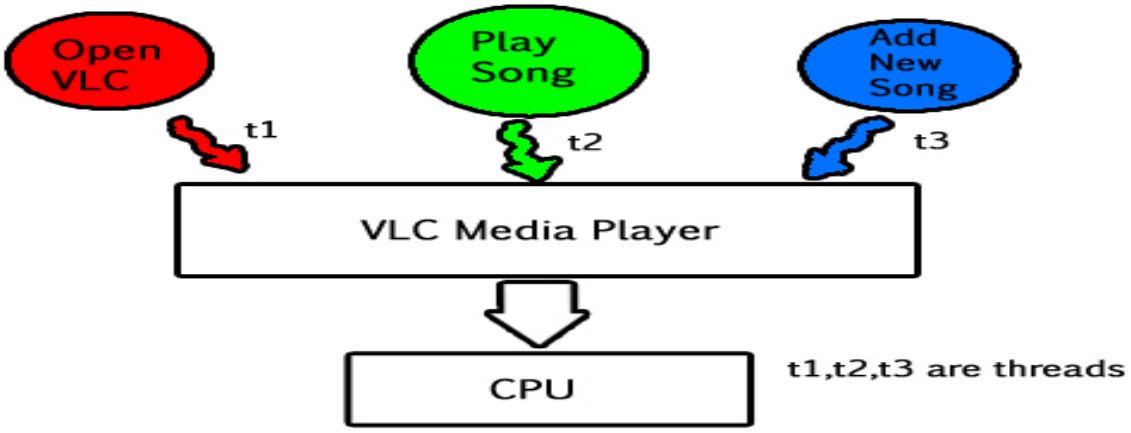
- With the help of multiprocessing, many processes can be executed simultaneously. Say processes P1, P2, P3 and P4 are waiting for execution. Now in a single processor system, firstly one process will execute, then the other, then the other and so on.
- But with multiprocessing, each process can be assigned to a different processor for its execution. If its a dual-core processor (2 processors), two processes can be executed simultaneously and thus will be two times faster, similarly a quad core processor will be four times as fast as a single processor

Multiprocessing refers to the hardware (i.e., the CPU units) rather than the software (i.e., running processes).



Multi threading –

A thread is a basic unit of CPU utilization. Multi threading is an execution model that allows a single process to have multiple code segments (i.e., threads) running concurrently within the "context"of that process. e.g. VLC media player, where one thread is used for opening the VLC media player, one thread for playing a particular song and another thread for adding new songs to the playlist



Operating System Structure

Multiprogramming environment

- Multiprogramming idea is as follows:

1. The operating system keeps several jobs in memory simultaneously .
2. One job **selected** and run via job scheduling.
3. When it has to **wait** (for I/O for example), OS switches to another job
4. Eventually, the first job finishes waiting and gets the CPU back.
5. As long as at least one job needs to execute. the CPU is never idle.

Time sharing (or multitasking) system:

- Multiple jobs are executed by switching the CPU between them. frequently that the users can interact with each program while it is running.
- In this, the CPU time is shared by different processes, so it is called as "Time sharing Systems".
- Time slice is defined by the OS, for sharing CPU time between processes.
- CPU is taken away from a running process when the allotted time slice expires.

ex: Unix, etc.

1.3 Operating-System Operations

- ✓ Modern operating systems are interrupt driven that means If there are no processes to execute, no I/O devices to service, and no users to whom to respond,
- ✓ An operating system will sit quietly, waiting for something to happen.

Events are almost always signaled by the occurrence of an **interrupt** or a **trap**.

what is trap ?

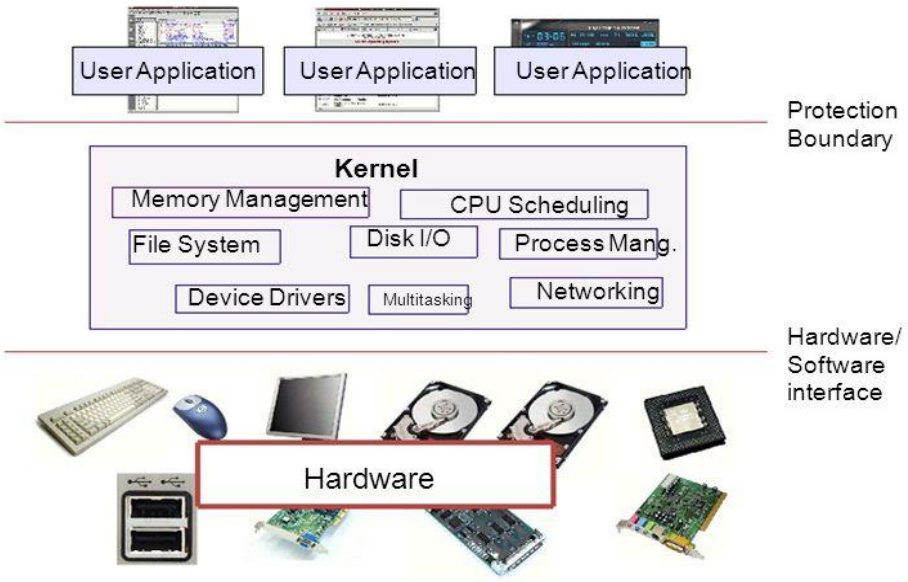
A trap (or an exception) is a software-generated interrupt caused either by an error (for example, division by zero or invalid memory access) or by a specific request from a user program to the operating-system to get the services of the OS.

The interrupt-driven nature of an operating system defines that system's general structure. For each type of interrupt, separate segments of code in the operating system determine what action should be taken. An interrupt service routine is provided that is responsible for dealing with the interrupt.

Since the operating system and the users share the hardware and software resources of the computer system, we need to make sure that an error in a user program could cause problems only for the one program that was running. With sharing, many processes could be adversely affected by a bug in one program. For example, if a process gets stuck in an infinite loop, this loop could prevent the correct operation of many other processes.

A properly designed operating system must ensure that an incorrect (or malicious) program cannot cause other programs to execute incorrectly.

OS Operations a snapshot



Dual-Mode Operation

In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user defined code.
The approach taken by most computer systems is to provide hardware support that allows us to differentiate among various modes of execution.

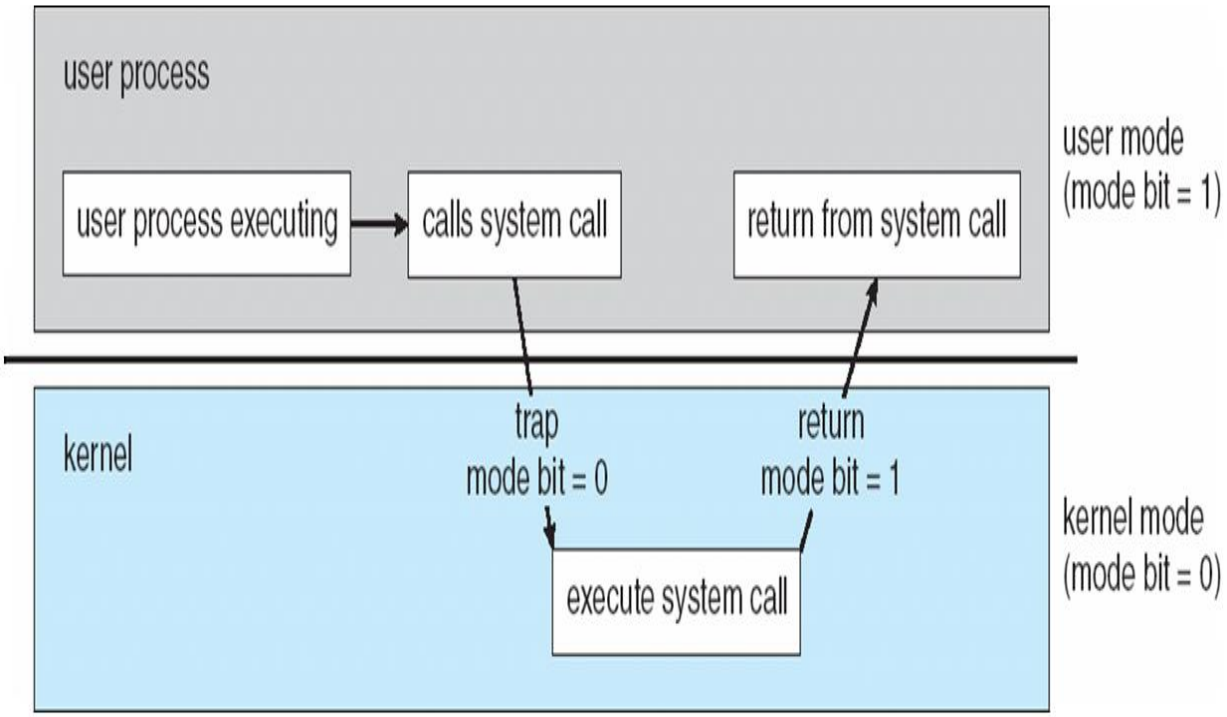
At the very least, we need two separate modes of operation:
user mode and **kernel mode** (also called supervisor mode, system mode, or privileged mode).

A bit, called the mode bit, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1). With the mode bit, we are able to distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user

- When the computer system is executing on behalf of a user application, **the system is in user mode**.
- However, when a user application requests a service from the operating system (via a system call), it must **transition from user to kernel mode** to fulfill the request. As we shall see, this architectural enhancement is useful for many other aspects of system operation as well.
- At system boot time, the hardware starts in **kernel mode.**

- The operating system is then loaded and starts user applications in **user mode**.
- Whenever a trap or interrupt occurs, the hardware switches from **user mode to kernel mode** (that is, changes the state of the mode bit to 0).
- Thus, whenever the operating system gains control of the computer, it is in **kernel mode**.
- The system always **switches to user mode** (by setting the mode bit to 1) before passing control to a user program.

Transition from User to Kernel Mode



- ✓ The dual mode of operation provides us with the means for protecting the operating system from errant users.

- ✓ We accomplish this protection by designating some of the machine instructions that may cause harm as privileged instructions. The hardware allows privileged instructions to be executed 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 it as illegal and traps it to the operating system

Timer

We must ensure that the operating system maintains control over the CPU. We ust prevent a user program from getting stuck in an infinite loop .To accomplish this goal, we can use a timer. A timer can be set to interrupt the computer after a specified period.
The period may be fixed (for example, 1/60 second) or variable (for example, from 1 millisecond to 1 second).

The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.
As long as the counter is positive, control is returned to the user program.
When the counter becomes negative, the operating system terminates the program for exceeding the assigned time limit.

1.4 Protection and Security

- Interference in resource utilization is a very serious threat in an OS
- The nature of the threat depends on the nature of a resource and the manner in which it is used.
- The most common threats are to information stored in files because they are the most common.

OS use two sets of techniques to counter threats to information namely:

1. Protection

2. Security

- **Protection** : It involves guarding a user's data and programs against interference by other authorized users of the system.

ex: Restricting the access of information with permissions

- **Security :** It involves guarding of a user's data and programs against interference by external entities, e.g. unauthorized persons.

ex: Restricting the access of system with firewalls and authentication

1.5 Computing Environments

Computing Environment is a collection of computers which are used to process and exchange the information to solve various types of computing problems.
The following are the various types of computing environments

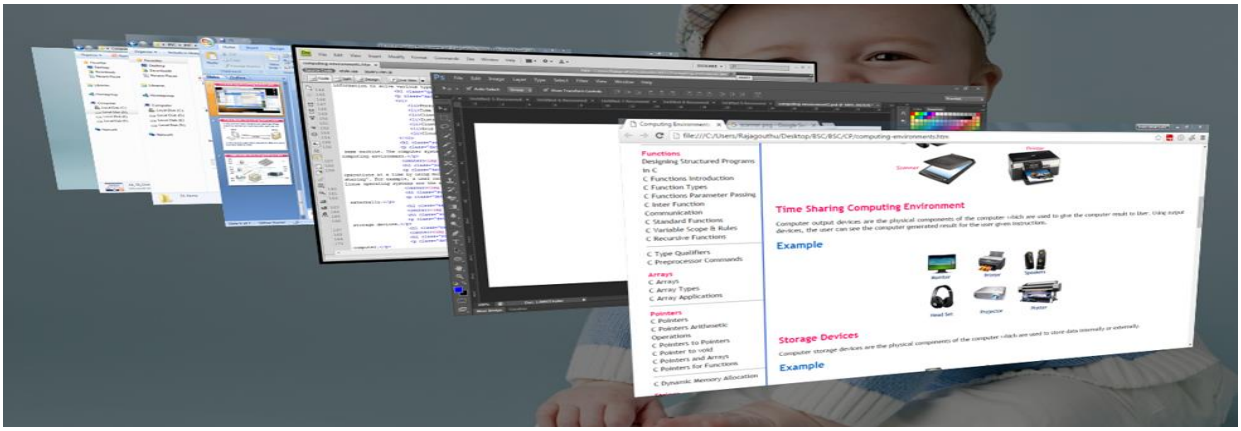
- Personal Computing Environment
- Time Sharing Computing Environment
- Client Server Computing Environment
- Distributed Computing Environment
- Peer-to-Peer Computing
- Grid Computing Environment
- Cluster Computing Environment
- Web-Based Computing

Personal Computing Environment

Personal computing is a stand alone machine. In personal computing environment, the complete program resides on stand alone machine and executed from the same machine. Laptops, mobile devices, printers, scanners and the computer systems we use at home, office are the examples for personal computing environment

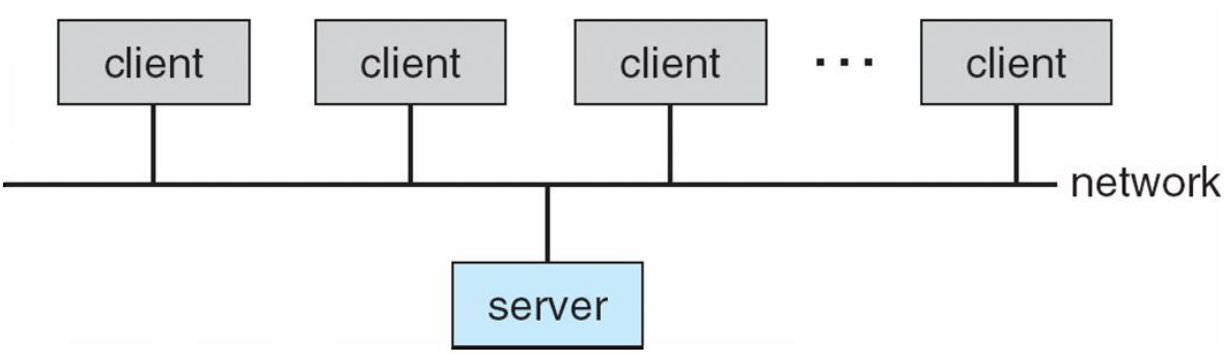
Time Sharing Computing Environment

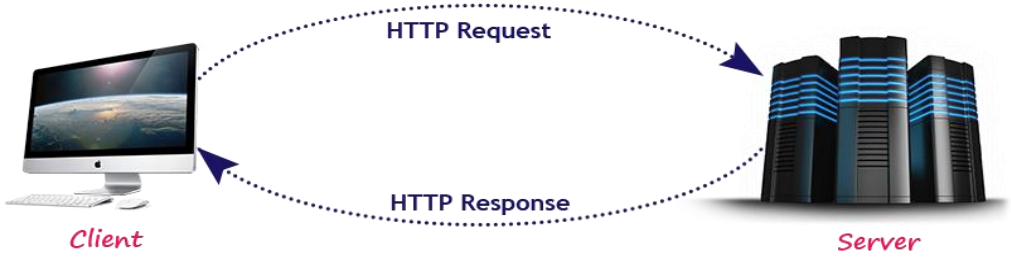
Time sharing computing environment is stand alone computer in which a single user can perform multiple operations at a time by using multitasking operating system. Here the processor time is divided among different tasks and this is called “Time sharing”. For example, a user can listen to music while writing something in a text editor. Windows 95 and later versions of windows OS, iOS and Linux operating systems are the examples for this computing environment.



Client Server Computing Environment

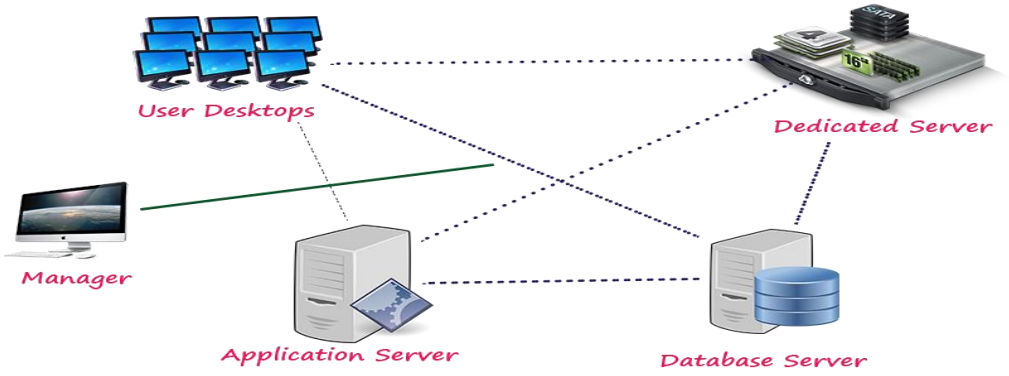
The client server environment contains two machines (Client machine and Server machine). These both machines will exchange the information through an application. Here Client is a normal computer like PC, Tablet, Mobile, etc., and Server is a powerful computer which stores huge data and manages huge amount of file and emails, etc., In this environment, client requests for data and server provides data to the client.





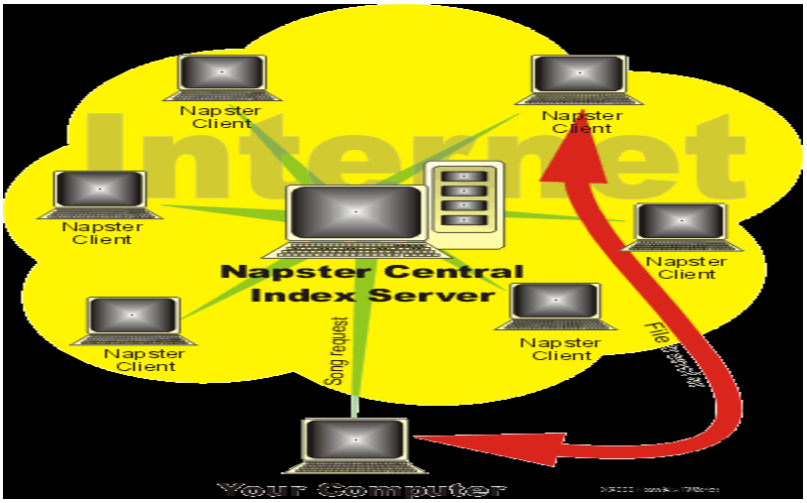
Distributed Computing Environment

In the distributed computing environment, the complete functionality of a software is not on single computer but is distributed among multiple computers. Here we use a method of computer processing in which different programs of an application run simultaneously on two or more computers. These computers communicate with each other over a network to perform the complete task. In distributed computing environment, the data is distributed among different systems and that data is logically related to each other.

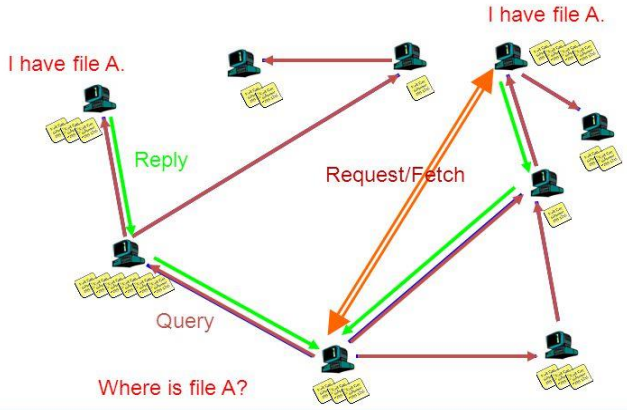


Peer-to-Peer Computing

- Another model of distributed system
- P2P does not distinguish clients and servers
  - Instead all nodes are considered peers
  - May each act as client server or both
  - Advantage over traditional client-server system is that the server is a bottleneck. But in a P2P system services can be provided by several nodes.



Gnutella: Search



Q: Compare with Napster (publishing, searching, anything else)

Marina Papatriantafilou – Overlays and peer-to-peer applications 9

Grid Computing Environment

Grid computing serves computing resources such as network, server, applications to the individual users. Grid involves the loosely coupled systems in which jobs are managed and scheduled in a distributed way. It divides a massive job in smaller chunks and processes those chunks separately. Grid computing is a combination of non-centralized computing resources where each geographically separate, independent site has its own administrative control over it.

Cloud Computing Environment

Cloud computing is a modern computing paradigm which provides scalable and flexible IT infrastructure and essential services to the users through the internet.

The difference between a cloud and a grid can be expressed as below:

Resource distribution: Cloud computing is a centralized model whereas grid computing is a decentralized model where the computation could occur over many administrative domains.

Ownership: A grid is a collection of computers which is owned by multiple parties in multiple locations and connected together so that users can share the combined power of resources. Whereas a cloud is a collection of computers usually owned by a single party.

Examples of Clouds: Amazon Web Services (AWS), Google App Engine.

Examples of Grids: FutureGrid.

Examples of cloud computing services: Dropbox, Gmail, Facebook, Youtube, RapidShare.

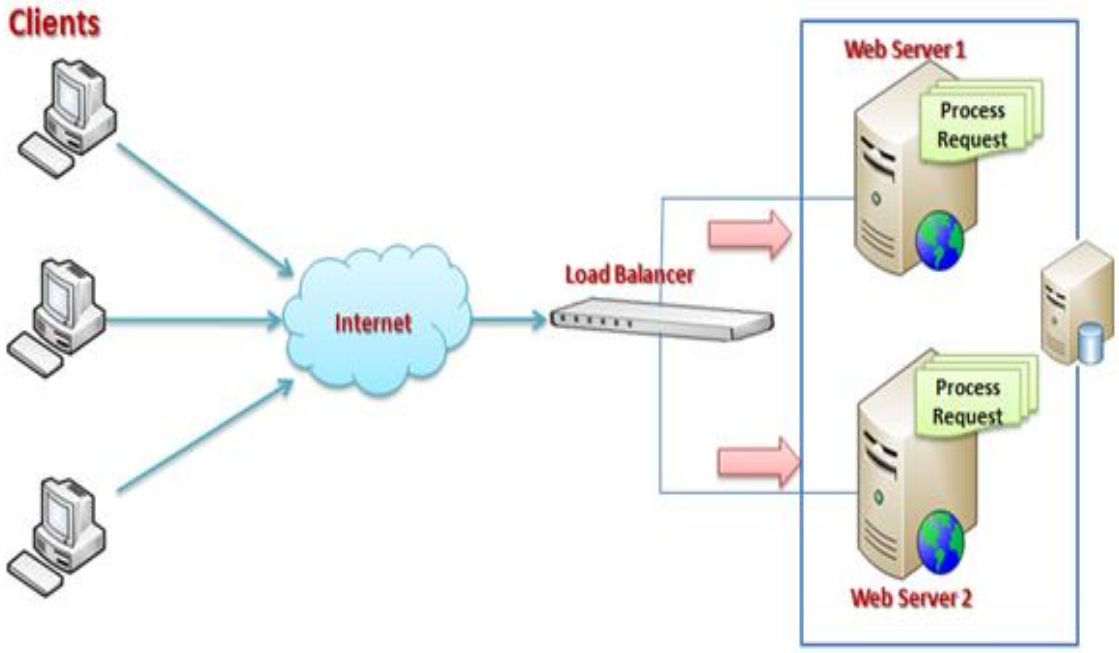
Cluster Computing Environment

Cluster computing is a collection of inter connected computers connected by LAN . These computers work together to solve a single problem. In cluster computing environment, a collection of systems work together as a single system. Cluster is tightly coupled . clusters are made up of machines with similar hardware

Web-Based Computing

Web has become ubiquitous(being present everywhere at once)

- Now **load balancers** are used to manage web traffic among similar servers
- Use of operating systems like Windows 95 client-side have evolved into Linux and Windows XP which can be clients and servers



1.6 Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary closed-source
- Started by Free Software Foundation (FSF) which has “copyleft” GNU Public License (GPL)
- Copyrights exist in order to protect authors of documentation or software from unauthorized copying or selling of their work. A copyright infers that only with the author's permission may such activities take place.
- Copyleft overrides copyright and promotes the concept that materials be freely used, copied and modified by others. Copyleft also requires all modified and extended versions of the material also be freely accessible, used and modified by others.
- Examples include GNU/Linux BSD UNIX (including core of Mac OS X) and Sun OpenSolaris , Ubuntu , ReactOS , Haiku OS

Ubuntu

- The most widely used open source data base is Ubuntu. It is a Linux based operating system and is distributed free along with the source code. Its desktop looks somewhat similar to that of Windows, with window controls and icons. There is reasonable large software support available on Ubuntu; the common applications include Mozilla Firefox web browser, LibreOffice office application suite, GIMP image editor and so on.

- Ubuntu is distributed under GNU and GPL license. It has UNIX shell called as Terminal that can be used to interact with the network and install third-party applications.

OpenSolaris

- OpenSolaris is a computer operating system developed by Sun Microsystems. It runs well on desktops, laptops, servers and data centers. OpenSolaris is GUI based like Ubuntu and has rich graphical desktop and windows for easy navigation.

FreeBSD

- FreeBSD is an advanced operating system for x86 compatible (including Pentium and Athlon), AMD64 compatible. It is popular among network developers, as FreeBSD offers advanced networking, performance, security and compatibility features. Most software that runs on Linux can run on FreeBSD without the need for any compatibility layer.

ReactOS

- It is a free Windows compatible OS that offers the benefits of running Windows apps natively. Apart from being an open-source software (its main highlight), the tool has one really cool feature that Windows failed to offer – An application manager very much similar to Linux package manager.

Haiku OS

- What most users like about this OS is its uniformity and cohesiveness.
- To start up the operating system, one simply inserts the thumb drive into a USB port and reboot. This should not pose a problem as most modern computers these days can be set to boot from the USB key. The OS is fast and responsive. Moreover, it is equipped with a number of applications and demos pre-installed.

System Structures:

1.7 Operating System Services

Operating systems provide an environment for execution of user programs and provides services to programs and users

1. One set of operating-system services provides functions that are helpful to the user:

- **User interface** - Almost all operating systems have a user interface (UI).
  - Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch

  1945–1968: Batch interface

  1969–present: Command-line user interface

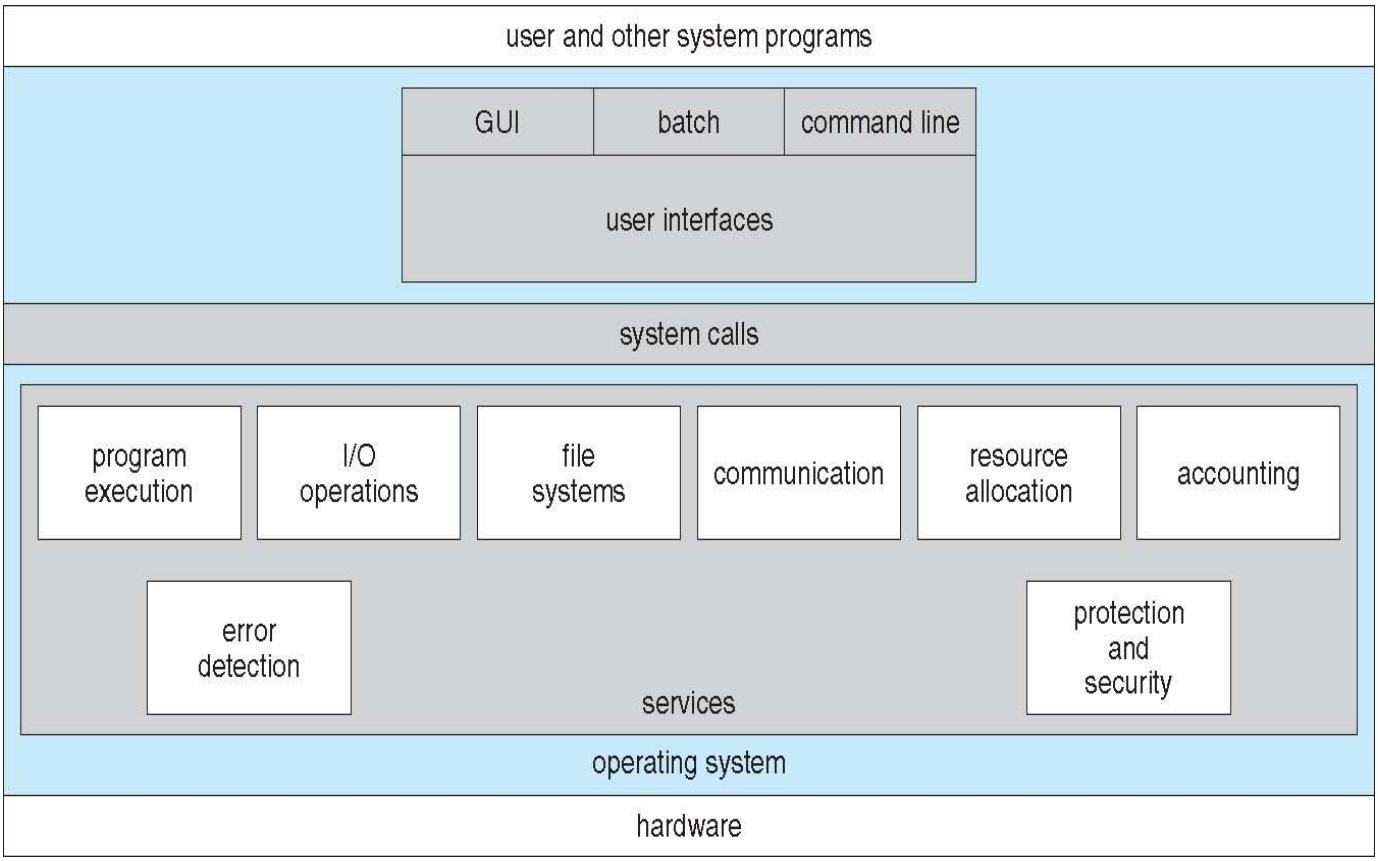
  1968–present: Graphical User Interface
- **Program execution** – The operating system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
- **I/O operations** - A running program may require I/O, which may involve a file or an I/O device
- **File-system manipulation** - The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

- **Communications** – Processes may exchange information, on the same computer or between computers over a network
  - Communications may be via shared memory or through message passing (packets moved by the OS)
- **Error detection** – OS needs to be constantly aware of possible errors
  May occur in the CPU and memory hardware, in I/O devices, in user program
  For each type of error, OS should take the appropriate action to ensure correct and consistent computing
- **Debugging** facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

2. Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing

- **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
  - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
- **Accounting -** To keep track of which users use how much and what kinds of computer resources
- **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
  - **Protection** involves ensuring that all access to system resources is controlled
  - **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts

A View of Operating System Services



1.8 User Operating System Interface

User Operating System Interface – CLI

CLI or command interpreter allows direct command entry

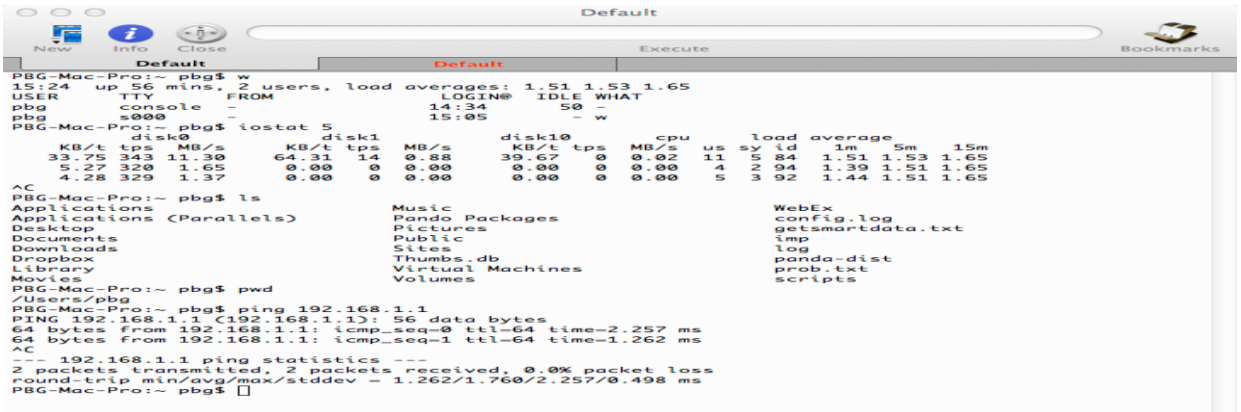
- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – shells

Bourne shell (sh),C shell (csh),TC shell (tcsh)

Korn shell (ksh),Bourne Again shell (bash)

- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs

Bourne Shell Command Interpreter



User Operating System Interface – GUI

- User-friendly desktop metaphor interface (an interface metaphor is a set of user interface visuals, actions and procedures that exploit specific knowledge that users already have of other domain)
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a folder)
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI "command" shell
  - Apple Mac OS X is "Aqua" GUI interface with UNIX kernel underneath and shells available
  - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

User Operating System Interface –Touch screen Interfaces

Touch screen devices require new interfaces

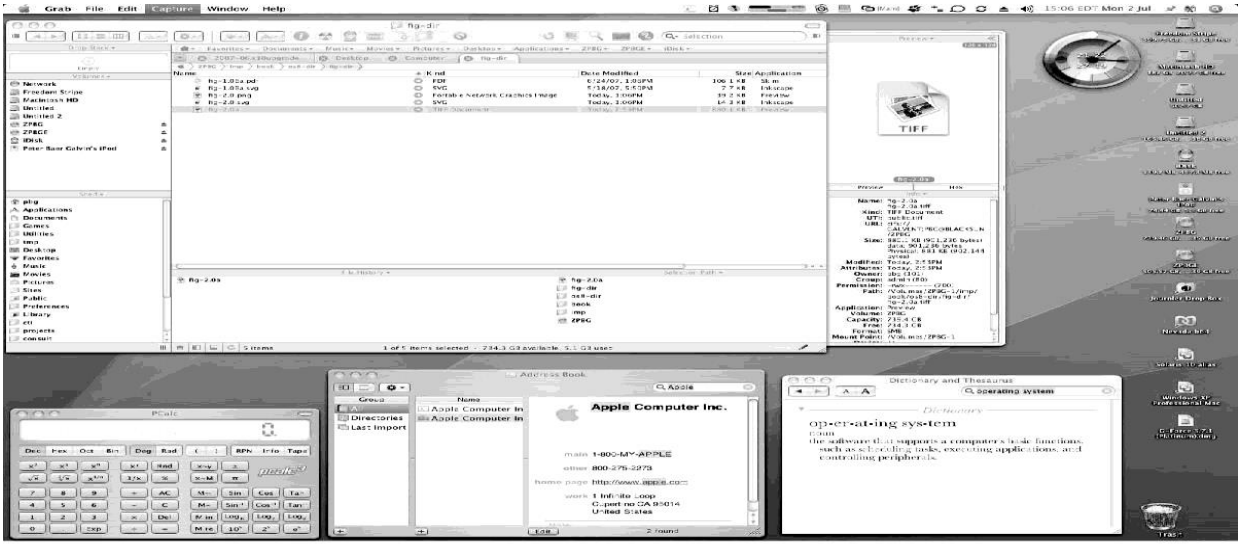
Mouse not possible or not desired

Actions and selection based on touches or gestures

Virtual keyboard for text entry

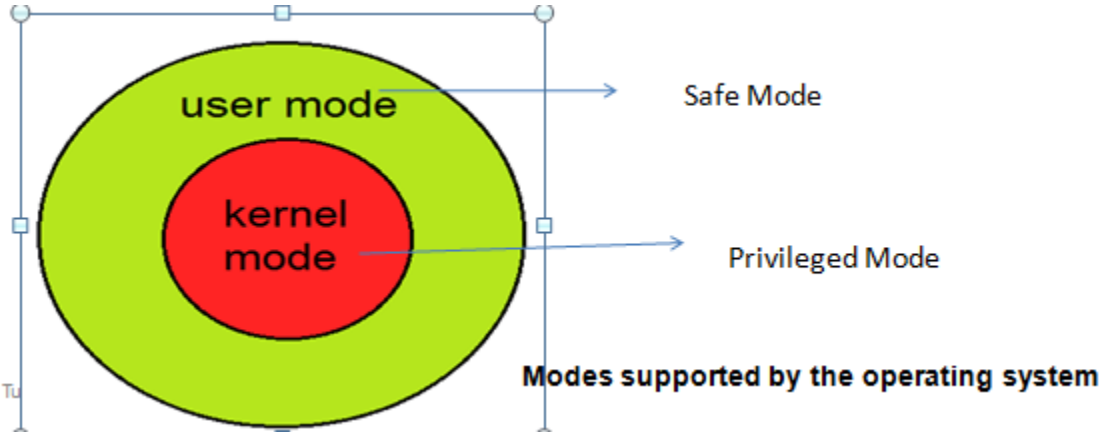
Voice commands.

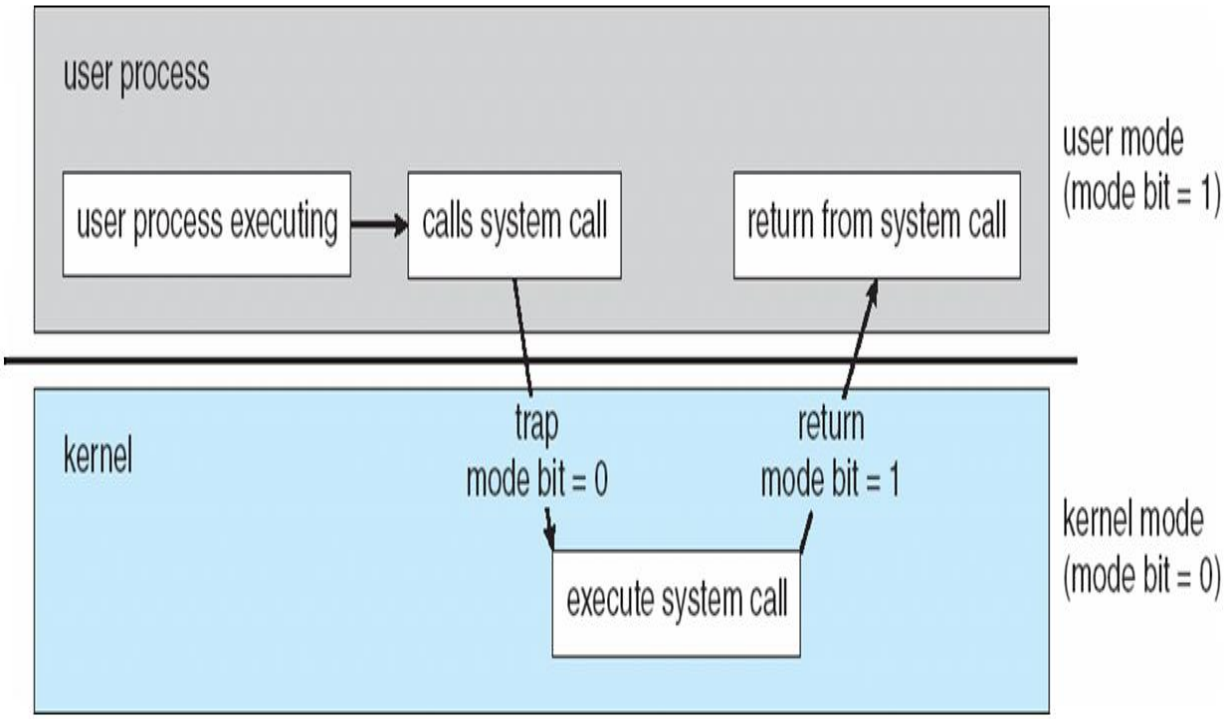
The Mac OS X GUI



1.9 System Calls

- System calls provide an interface to the services made available by an operating system
- To understand system calls, first one needs to understand the difference between kernel mode and user mode of a CPU. Every modern operating system supports these two modes.





User Mode

- When CPU is in **user mode**, the programs don't have direct access to memory and hardware resources.
- In user mode, if any program crashes, only that particular program is halted.
- That means the system will be in a safe state even if a program in user mode crashes.
- Hence, most programs in an OS run in user mode.

Kernel Mode

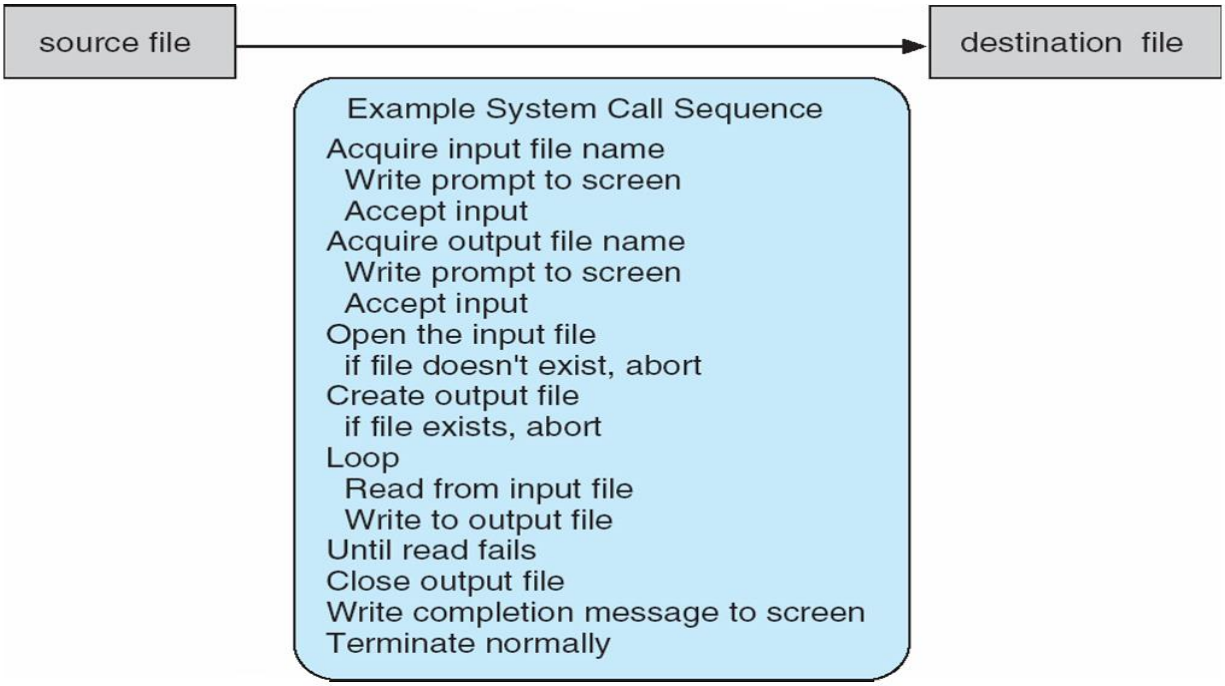
- When CPU is in **kernel mode**, the code being executed can access any memory address and any hardware resource.
- Hence kernel mode is a very privileged and powerful mode.
- If a program crashes in kernel mode, the entire system will be halted.

System Calls

- 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 of CPU is switched from user mode to kernel mode. This is called a **context switch**.
- Then the kernel provides the resource which the program requested. After that, another **context switch** happens which results in change of mode from kernel mode back to user 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.
- So System call is the programmatic way in which a computer program requests a service from the kernel of the Operating System .

- Programming interface to the services provided by the OS
- These calls are generally available as routines written in C or C++
- 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)

Example of System Calls



Example of Standard API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

return value — function name — parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

System Call Implementation

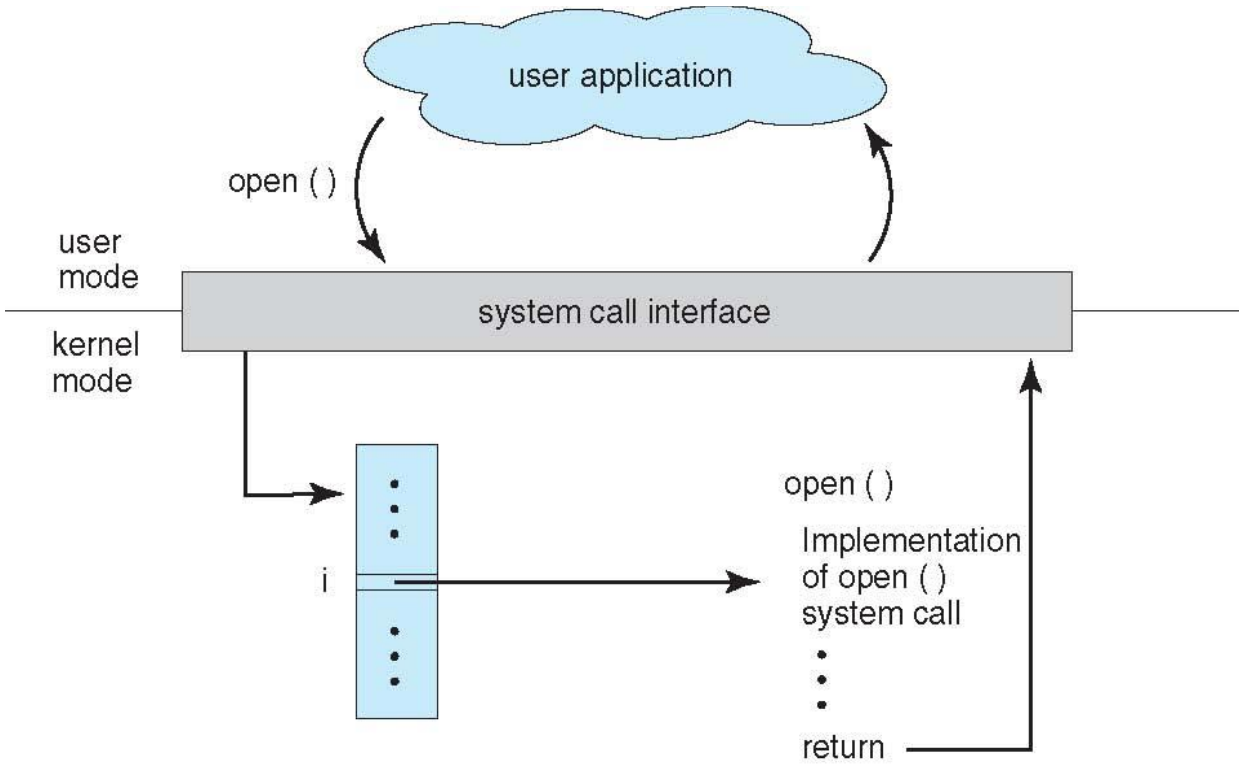
Typically, a number associated with each system call

- System-call interface maintains a table indexed according to these numbers

- The system call interface invokes the 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

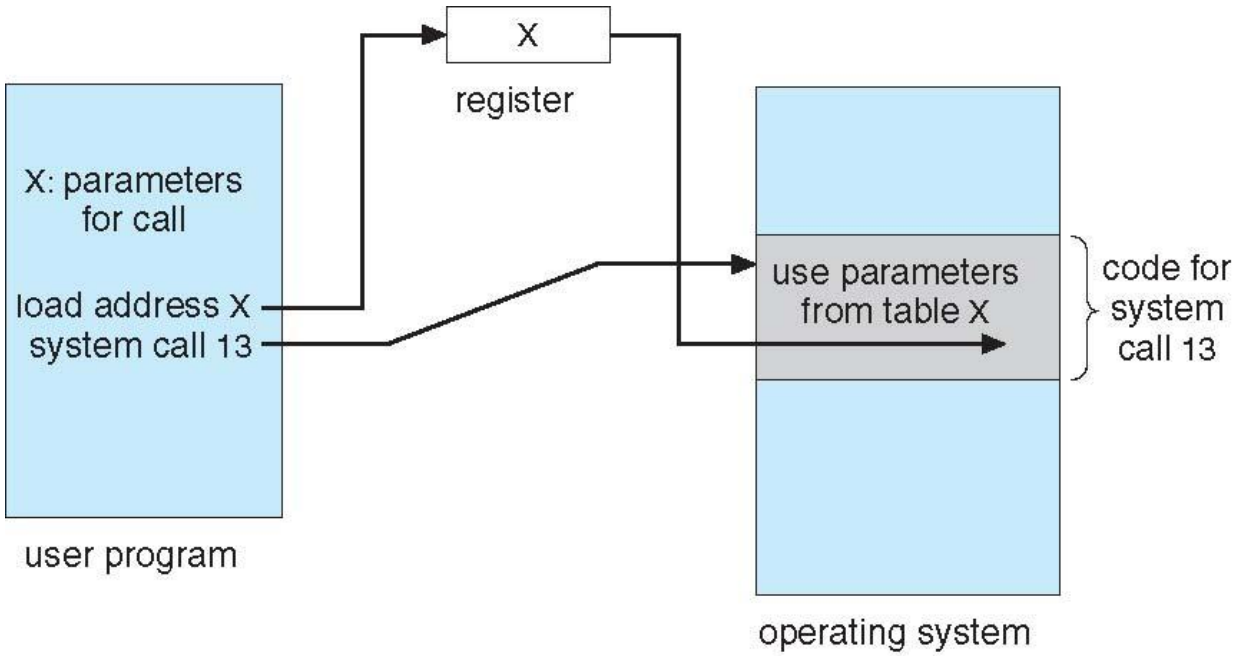


System Call Parameter Passing

- 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



1.10 Types of System Calls

Six major categories system calls are as follows:

1. Process control

- create process, terminate process
- end, abort
- load, execute
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory
- Dump memory if error
- Debugger for determining bugs, single step execution
- Locks for managing access to shared data between processes

2. File management

- create file, delete file
- open, close file

- read, write, reposition
- get and set file attributes

3. Device management

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

4. Information maintenance

- get time or date, set time or date
- get system data, set system data
- get and set process, file, or device attributes

5. Communications

- create, delete communication connection
- send, receive messages if message passing model to host name or process name
    - From client to server
- Shared-memory model create and gain access to memory regions
- transfer status information
- attach and detach remote devices

6. Protection

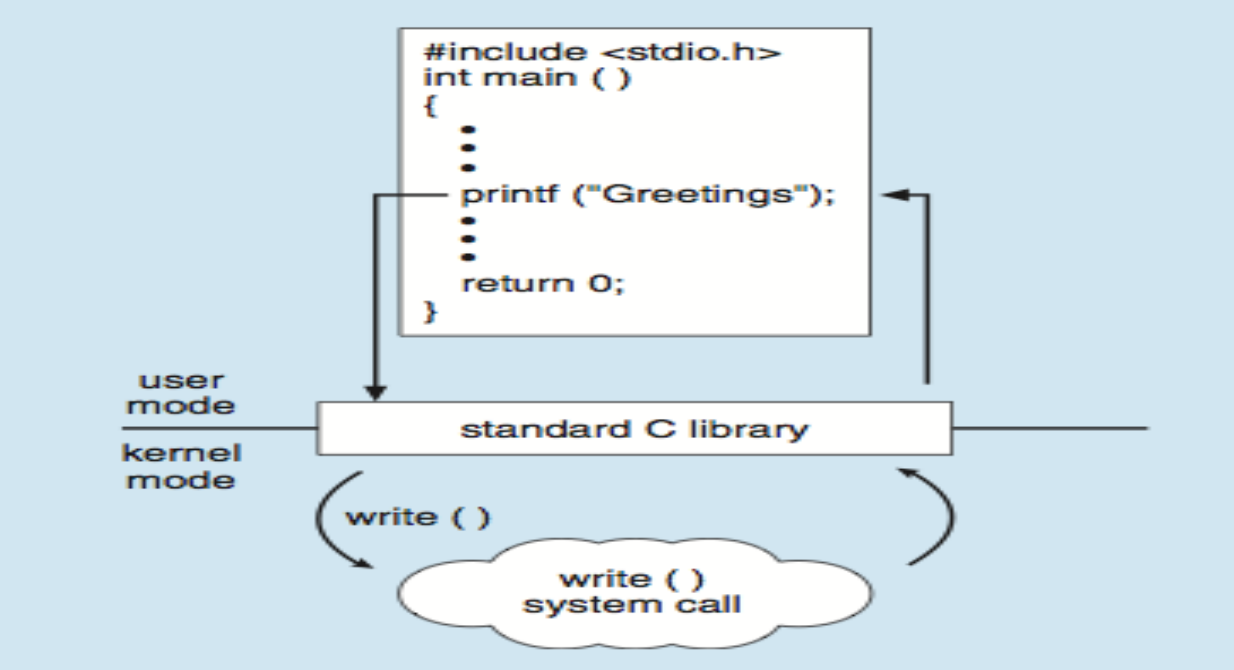
- Control access to resources
- Get and set permissions
- Allow and deny user access

Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

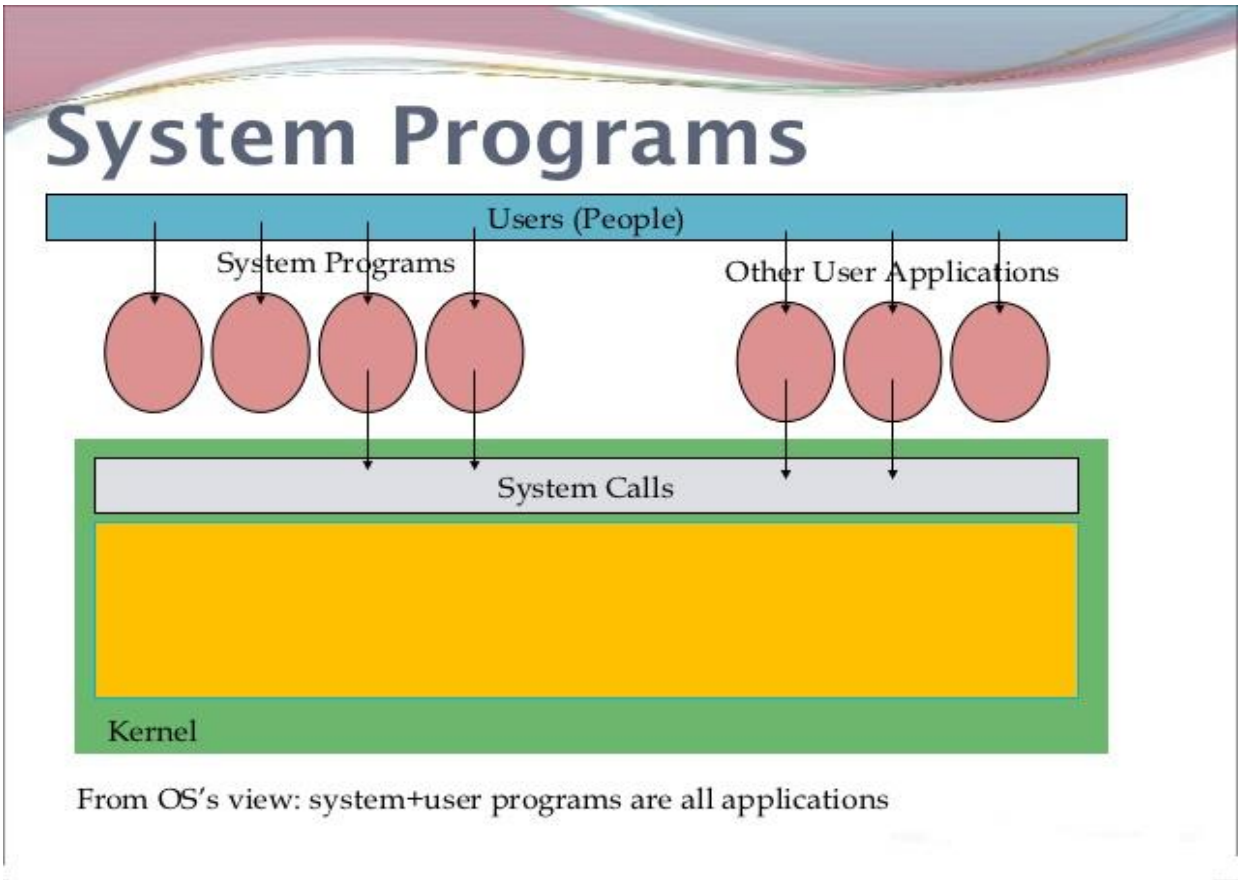
Standard C Library Example

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



1.11 System Programs

- System programs also known as system utilities provide a convenient environment for program development and execution. Some of them are simply user interfaces to system calls ; others are considerably more complex. They can be divided into:
  - File manipulation :
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs
- Most users' view of the operating system is defined by system programs, not the actual system calls



- Provide a convenient environment for program development and execution

- Some of them are simply user interfaces to system calls; others are considerably more complex

- **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 sometimes provided

- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and 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**
  - Launch at boot time
    - Some for system startup, then terminate
    - Some from system boot to shutdown
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as services, subsystems, daemons

- **Application programs**
  - Don't pertain to system

- Run by users
- Not typically considered part of OS
- Launched by command line, mouse click, finger poke

Operating-System Design and Implementation

- **Design 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
- **Mechanisms and Policies :**
  - Policies determine what is to be done.
  - Mechanisms determine how it is to be implemented.
- **Implementation**
  - Traditionally OSes were written in assembly language

Most modern OSes are written in C, or more recently, C++. Critical sections of code are still written in assembly language, (or written in C, compiled to assembly

1.12 Operating System Structure

Requirements for operating systems can vary greatly depending on the planned scope and usage of the system. (Single user / multi-user, specialized system / general purpose, high/low security, performance needs, operating environment, etc.). That means affected by choice of hardware, type of system

- So the Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications

User goals and System goals

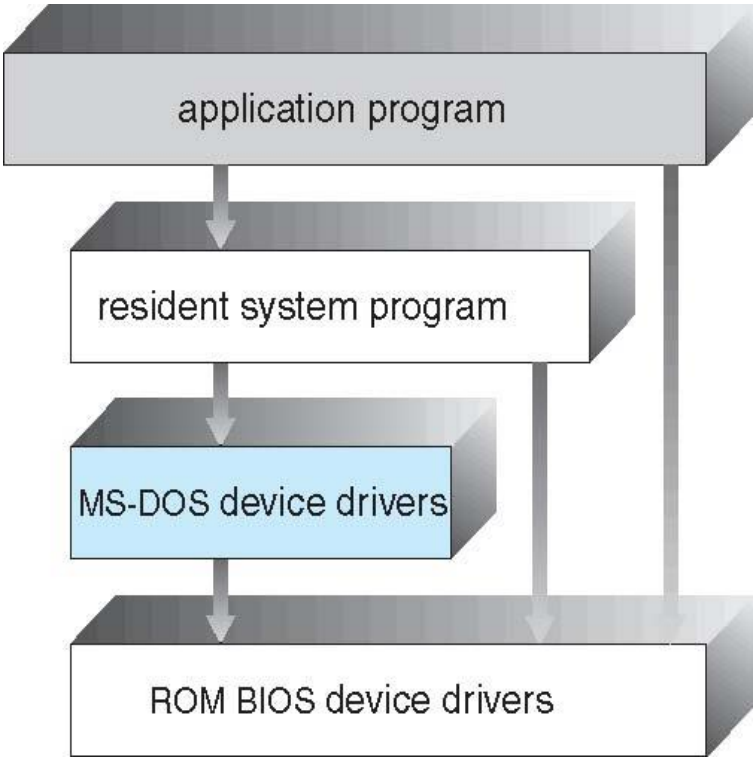
- **User goals** – operating system should be convenient to use, easy to learn, reliable, safe, and fast
- **System goals** – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

Structure/Organization/Layout of Operating Systems:

OS components and its organization and its interactions is called **Operating System structure**. The following are the some of the **OS** structures**.**

1.Simple Structure -- MS-DOS

- 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

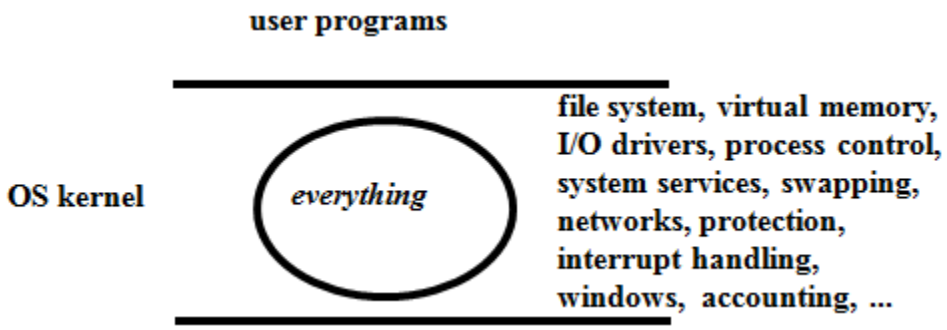


2.Non Simple Structure or Monolithic -- UNIX

Traditionally, systems such as Unix were built as a monolithic kernel:

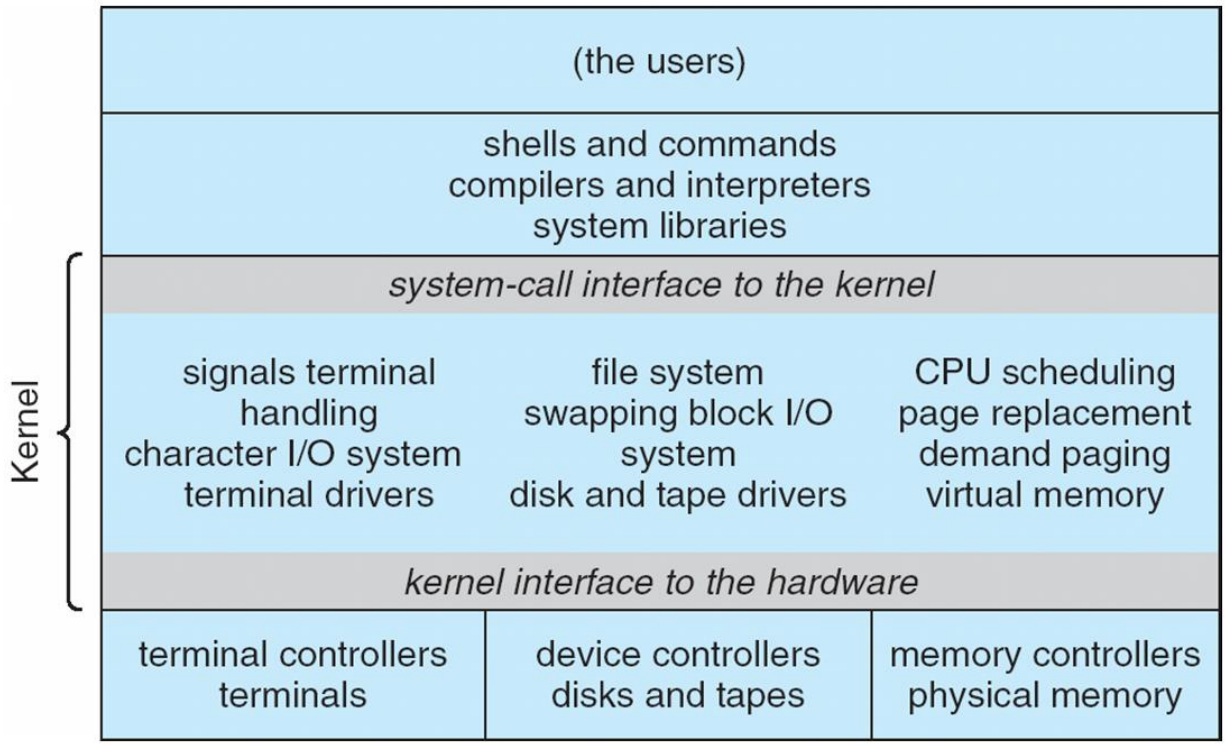
The UNIX OS consists of two separable parts

- 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



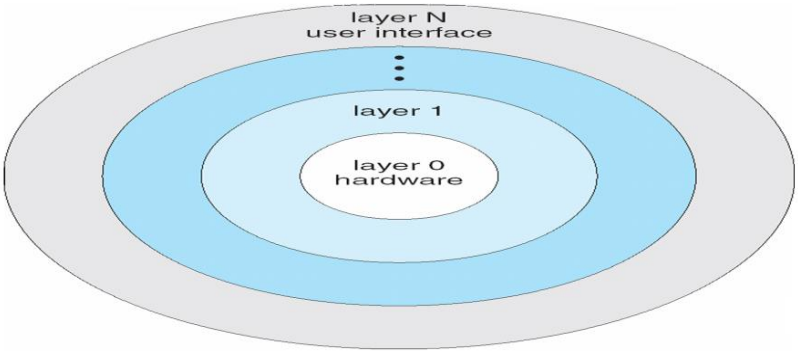
Traditional UNIX System Structure

- Beyond simple but not fully layered

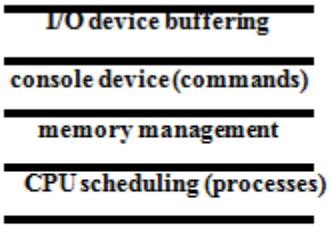


3.Layered Approach

- 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.



The first description of this approach was Dijkstra's system.

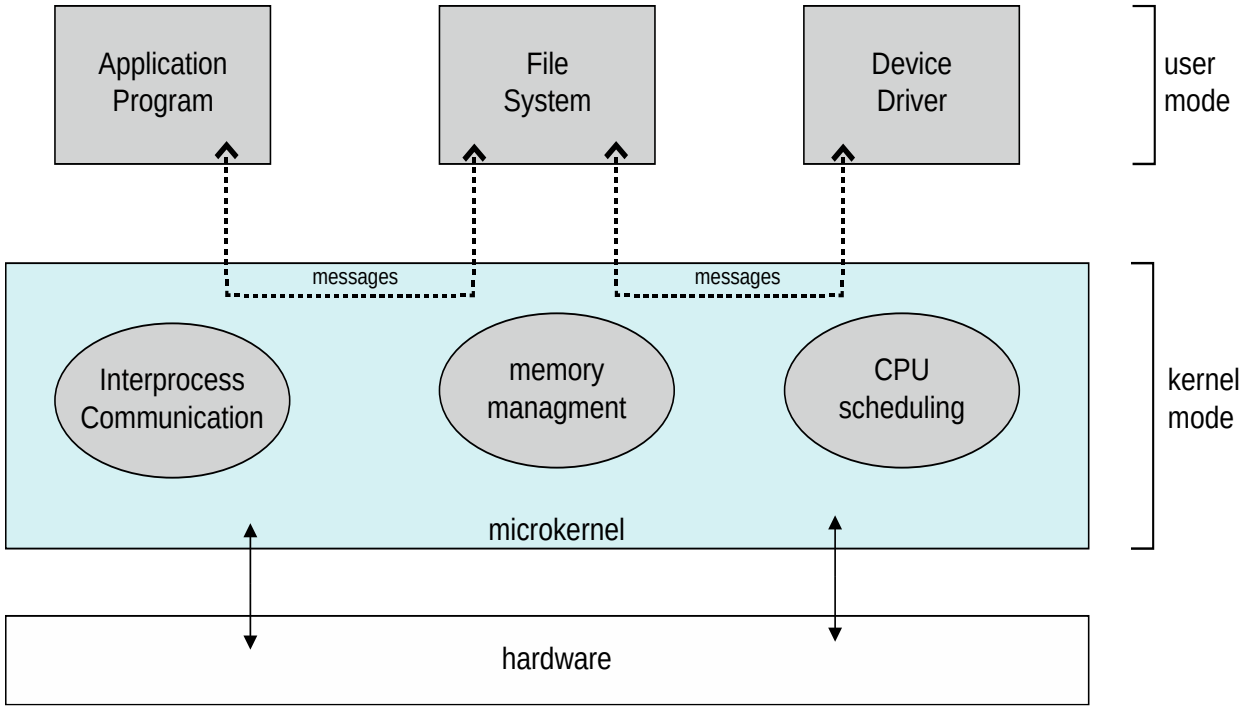


Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

4.Microkernel System Structure

- The basic idea behind micro kernels is to remove all non-essential services from the kernel, and implement them as system applications instead, thereby making the kernel as small and efficient as possible.
- Most microkernels provide basic process and memory management, and message passing between other services, and not much more.
- Communication takes place between user modules using message passing
- Security and protection can be enhanced, as most services are performed in user mode, not kernel mode.
- System expansion can also be easier, because it only involves adding more system applications, not rebuilding a new kernel.
- Mach was the first and most widely known microkernel, and now forms a major component of Mac OSX.
- Windows NT was originally microkernel, but suffered from performance problems relative to Windows 95. NT 4.0 improved performance by moving more services into the kernel, and now XP is back to being more monolithic.
- Another microkernel example is QNX, a real-time OS for embedded systems.
- 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

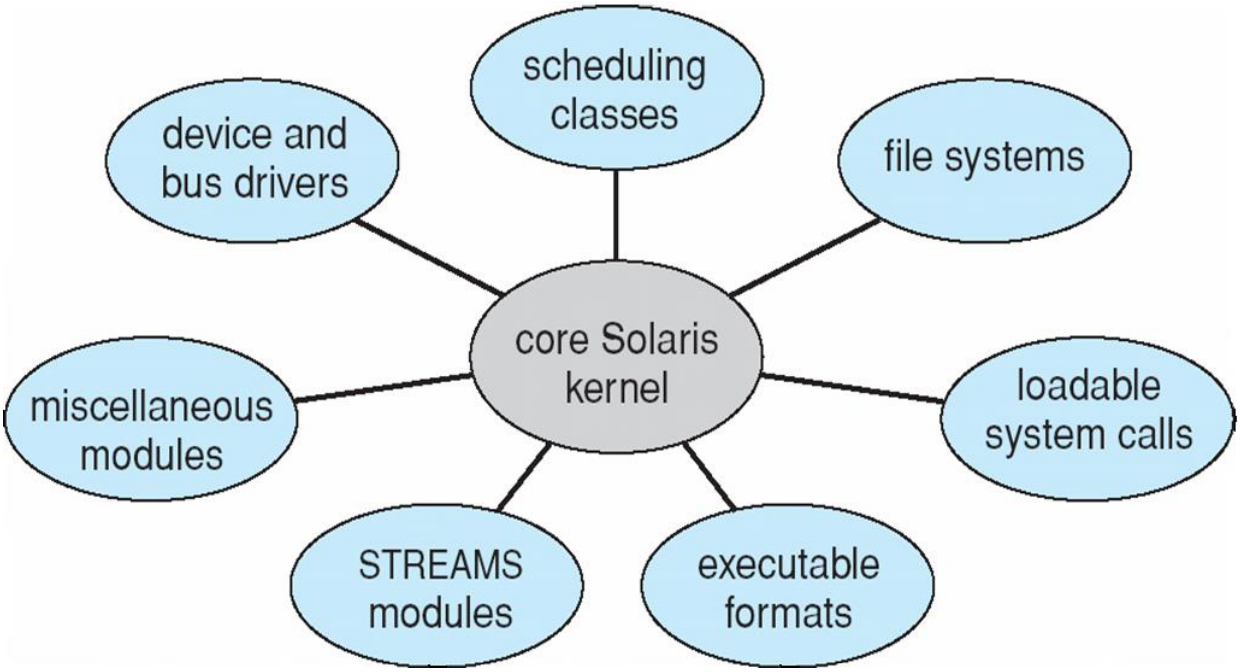
Microkernel System Structure



5.Modular kernel

- Many modern operating systems implement loadable 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
  - Linux, Solaris, etc

Solaris Modular Approach



Hybrid Systems

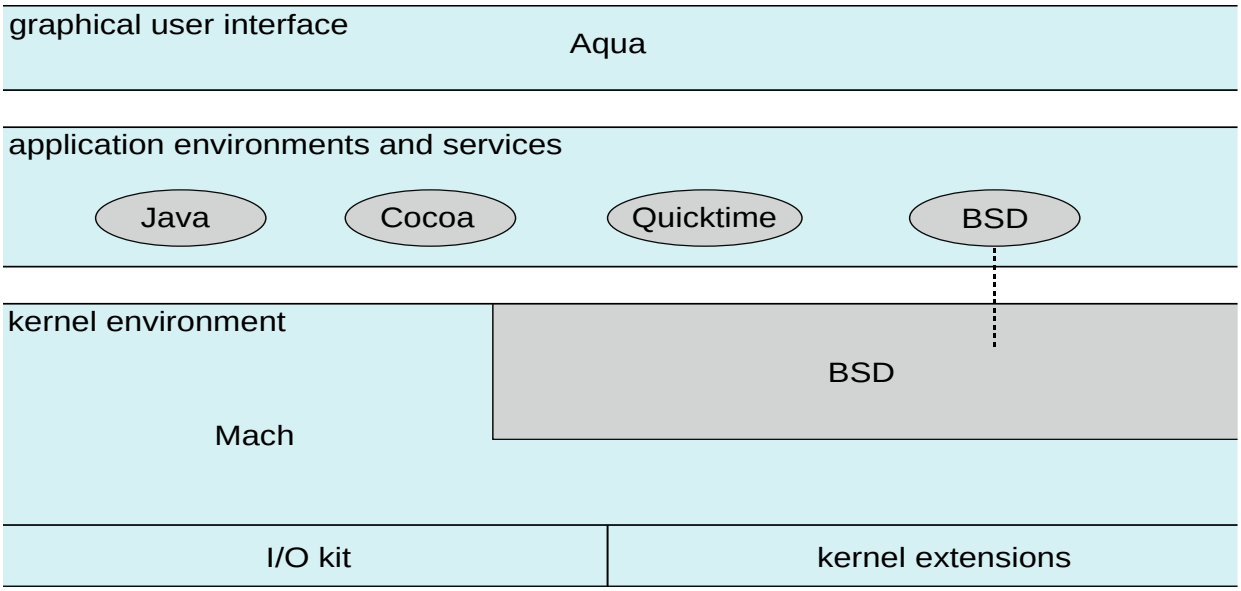
- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem *personalities*

ex: Mac OS X Structure

- Apple Mac OS X hybrid, layered, Aqua UI plus Cocoa programming environment
- Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called kernel extensions)
- The Mac OSX architecture relies on the Mach microkernel for basic system management services, and the BSD kernel for additional services.

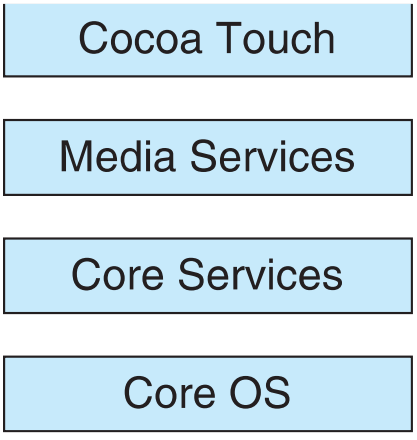
Application services and dynamically loadable modules (kernel extensions) provide the rest of the OS functionality:

Mac OS X Structure



iOS

- Apple mobile OS for *iPhone*, *iPad*
  - Cocoa Touch Objective-C API for developing apps
  - Media services layer for graphics, audio, video
  - Core services provides cloud computing, databases
  - Core operating system, based on Mac OS X kernel

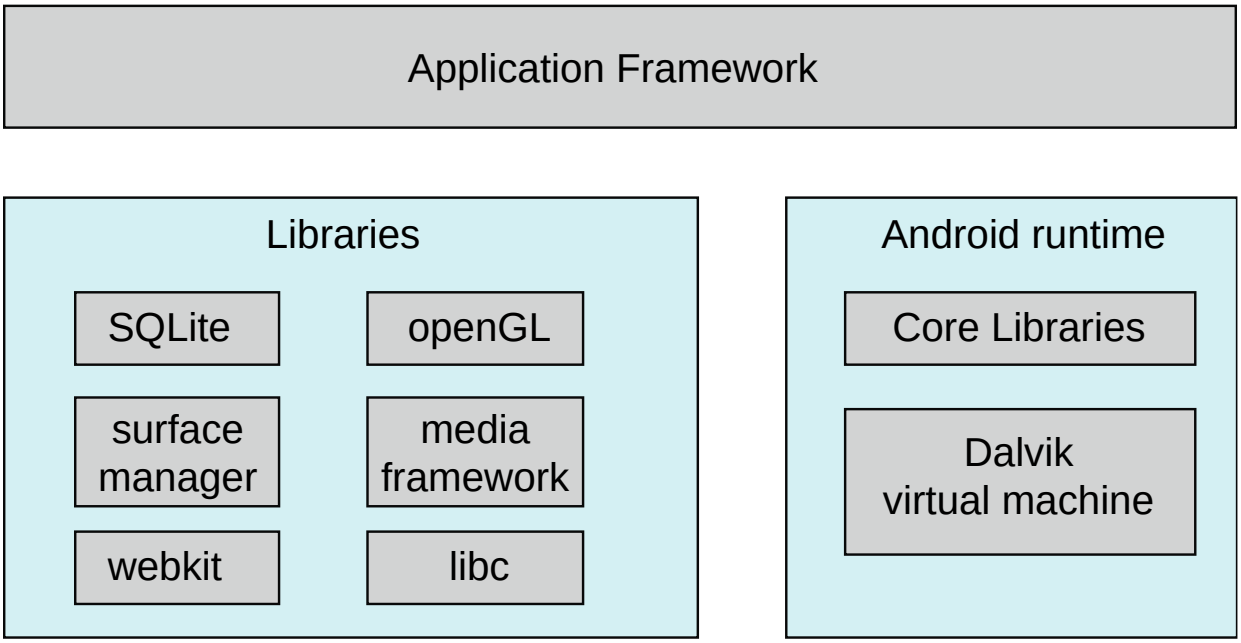


Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source

- Similar stack to IOS
- Based on Linux kernel but modified
  - – Provides process, memory, device-driver management
  - – Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - – Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Android Architecture



1.13 Operating-System Debugging

Debugging is the process of finding the problems in a computer system and solving them. There are many different ways in which operating systems perform debugging. Some of these are:

Log Files

The log files record all the events that occur in an operating system. This is done by writing all the messages into a log file. There are different types of log files. Some of these are given as follows:

Event Logs

These stores the records of all the events that occur in the execution of a system. This is done so that the activities of all the events can be understood to diagnose problems.

Transaction Logs

The transaction logs store the changes to the data so that the system can recover from crashes and other errors. These logs are readable by a human.

Message Logs

These logs store both the public and private messages between the users. They are mostly plain text files, but in some cases they may be HTML files.

Core Dump Files

The core dump files contain the memory address space of a process that terminates unexpectedly. The creation of the core dump is triggered in response to program crashes by the kernel. The core dump files are used by the developers to find the program's state at the time of its termination so that they can find out why the termination occurred.

The automatic creation of the core dump files can be disabled by the users. This may be done to improve performance, clear disk space or increase security.

Crash Dump Files

In the event of a total system failure, the information about the state of the operating system is captured in crash dump files. There are three types of dump that can be captured when a system crashes. These are:

Complete Memory Dump

The whole contents of the physical memory at the time of the system crash are captured in the complete memory dump. This is the default setting on the Windows Server System.

Kernel Memory Dump

Only the kernel mode read and write pages that are present in the main memory at the time of the system crash are stored in the kernel memory dump.

Small Memory Dump

This memory dump contains the list of device drivers, stop code, process and thread information, kernel stack etc.

Trace Listings (DTrace)

The trace listing record information about a program execution using logging. This information is used by programmers for debugging. System administrators and technical personnel can use the trace listings to find the common problems with software using software monitoring tools.

DTrace tool in Solaris, FreeBSD, Mac OS X allows live instrumentation on production systems

Use an operating system probe to monitor system resources, CPU usage, disk activity, physical memory use, and network traffic. Probes fire when code is executed within a provider, capturing state data and sending it to consumers of those probes

Example of following XEventsQueued system call move from libc library to kernel and back

---

DTrace code to record amount of time each process with UserID 101 is in running mode (on CPU) in nanoseconds

```
# dtrace -s sched.d
dtrace: script 'sched.d' matched 6 probes
^C
    gnome-settings-d              142354
    gnome-vfs-daemon              158243
    dsdm                          189804
    wnck-applet                   200030
    gnome-panel                   277864
    clock-applet                  374916
    mapping-daemon                385475
    xscreensaver                  514177
    metacity                      539281
    Xorg                         2579646
    gnome-terminal               5007269
    mixer_applet2                7388447
    java                        10769137
```

Figure 2.21 Output of the D code.

Profiling

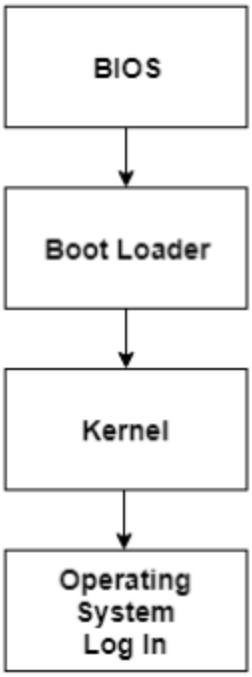
This is a type of program analysis that measures various parameters in a program such as space and time complexity, frequency and duration of function calls, usage of specific instructions etc. Profiling is done by monitoring the source code of the required system program using a code profiler.

1.14 System Boot

The BIOS, operating system and hardware components of a computer system should all be working correctly for it to boot. If any of these elements fail, it leads to a failed boot sequence.

System Boot Process

The following diagram demonstrates the steps involved in a system boot process:



Here are the steps:

- The CPU initializes itself after the power in the computer is first turned on. This is done by triggering a series of clock ticks that are generated by the system clock.
- After this, the CPU looks for the system's ROM BIOS to obtain the first instruction in the start-up program. This first instruction is stored in the ROM BIOS and it instructs the system to run POST (Power On Self Test) in a memory address that is predetermined.
- POST first checks the BIOS chip and then the CMOS RAM. If there is no battery failure detected by POST, then it continues to initialize the CPU.
- POST also checks the hardware devices, secondary storage devices such as hard drives, ports etc. And other hardware devices such as the mouse and keyboard. This is done to make sure they are working properly.
- After POST makes sure that all the components are working properly, then the BIOS finds an operating system to load.
- In most computer system's, the operating system loads from the C drive onto the hard drive. The CMOS chip typically tells the BIOS where the operating system is found.
- The order of the different drives that CMOS looks at while finding the operating system is known as the boot sequence. This sequence can be changed by changing the CMOS setup.

- After finding the appropriate boot drive, the BIOS first finds the boot record which tells it to find the beginning of the operating system.
- After the initialization of the operating system, the BIOS copies the files into the memory. Then the operating system controls the boot process.
- In the end, the operating system does a final inventory of the system memory and loads the device drivers needed to control the peripheral devices.
- The users can access the system applications to perform various tasks.

Without the system boot process, the computer users would have to download all the software components, including the ones not frequently required. With the system boot, only those software components need to be downloaded that are legitimately required and all extraneous components are not required. This process frees up a lot of space in the memory and consequently saves a lot of time.