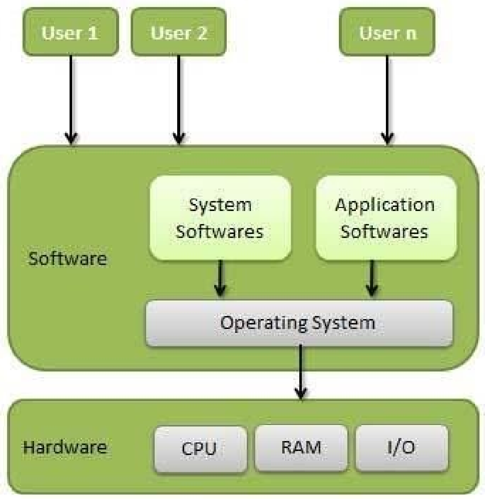
Operating System

- An Operating System (OS) is an interface between a computer user and computer hardware.
- An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.
- Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Definition
An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



Following are some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

Memory Management
Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management −

- Keeps tracks 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 to do so.
- 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 **process scheduling**. An Operating System does the following activities for processor management −

- Keeps tracks of processor and status of process. 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.

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 directions.

An Operating System does the following activities for file management −

- Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.

Other Important Activities

Following are some of the important activities that an Operating System performs −

- **Security** − By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performance** − Recording delays between request for a service and response from the system.
- **Job 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 systems.

Operating Systems Operations

Dual Mode Operations in Operating System

- The dual-mode operations in the operating system protect the operating system from illegal users.
- To ensure proper operating system execution, we must differentiate between machine code execution and user-defined code.
- . We have two modes of the operating system: ***user mode*** and ***kernel mode***.
- Mode bit is required to identify in which particular mode the current instruction is executing.
- If the mode bit is 1, it operates user mode, and if the mode bit is 0, it operates in kernel mode.

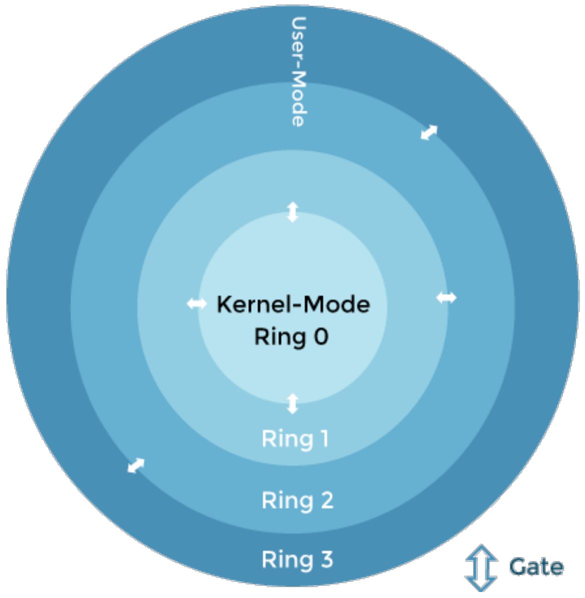
NOTE: At the booting time of the system, it always starts with the kernel mode.

Types of Dual Mode in Operating System

The operating system has two modes of operation to ensure it works correctly: user mode and kernel mode.

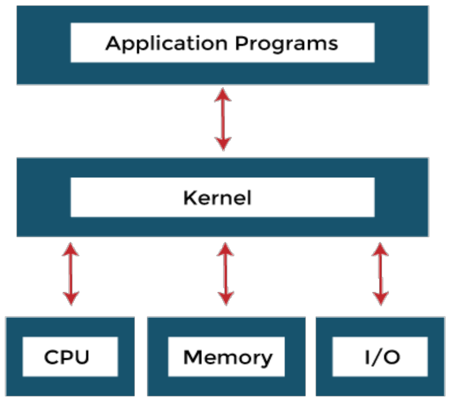
1. User Mode

- When the computer system runs user applications like file creation or any other application program in the User Mode, this mode does not have direct access to the computer's hardware.
- For performing hardware related tasks, like when the user application requests for a service from the operating system or some interrupt occurs,
- in these cases, the system must switch to the Kernel Mode.
- The mode bit of the user mode is 1. This means that if the mode bit of the system's processor is 1, then the system will be in the User Mode.



2. Kernel Mode

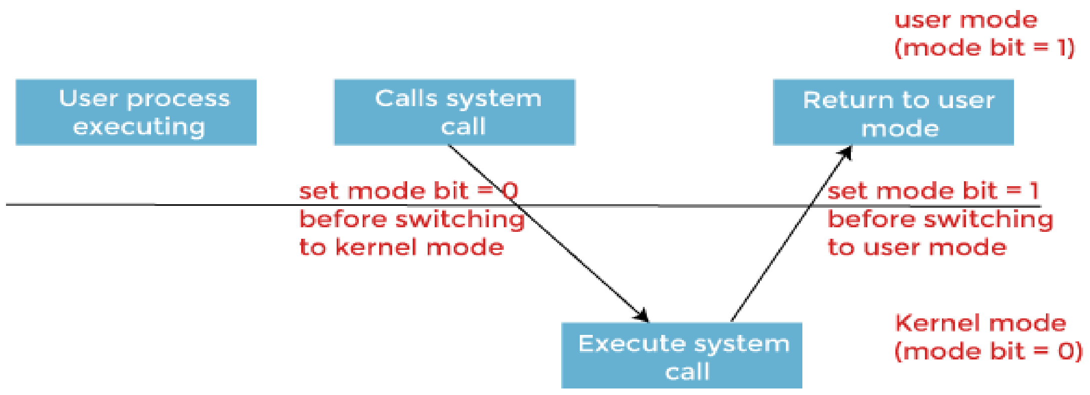
- All the bottom level tasks of the Operating system are performed in the Kernel Mode
- As the Kernel space has direct access to the hardware of the system, so the kernel-mode handles all the processes which require hardware support.
- Apart from this, the main functionality of the Kernel Mode is to execute privileged instructions.



- These privileged instructions are not provided with user access, and that's why these instructions cannot be processed in the User mode.
- So, all the processes and instructions that the user is restricted to interfere with are executed in the Kernel Mode of the Operating System.
- The mode bit for the Kernel Mode is 0. So, for the system to function in the Kernel Mode, the Mode bit of the processor must be equal to 0.

Example

With the mode bit, we can distinguish between a task executed on behalf of the operating system and one executed on behalf of the user.



- When the computer system executes 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 can say, this architectural enhancement is useful for many other aspects of system operation.
- 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***, changing the mode bit's state 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.

Need for Dual-Mode Operations

- Certain types of processes are to be made hidden from the user, and certain tasks that do not require any type of hardware support.
- Using the ***dual mode*** of the OS, these tasks can be deal with separately.
- Also, the Operating System needs to function in the ***dual mode*** because the Kernel Level programs perform all the bottom level functions of the OS like process management, Memory management, etc.
- If the user alters these, then this can cause an entire system failure. So, for specifying the access to the users only to the tasks of their use, Dual Mode is necessary for an Operating system.
- So, whenever the system works on the user applications, it is in the User mode.
- Whenever the user requests some hardware services, a transition from User mode to Kernel mode occurs, and this is done by changing the mode bit from 1 to 0.
- And for returning back into the User mode, the mode bit is again changed to 1.

User Mode and Kernel Mode Switching

- In its life span, a process executes in user mode and kernel mode.
- The user mode is a normal mode where the process has limited access.
- However, the kernel-mode is the privileged mode where the process has unrestricted access to system resources like hardware, memory, etc.
- A process can access services like hardware I/O by executing accessing kernel data in kernel mode. Anything related to process management, I/O hardware management, and memory management requires a process to execute in Kernel mode.
- This is important to know that a process in Kernel mode get power to access any device and memory, and same time any crash in kernel mode brings down the whole system. But any crash in user mode brings down the faulty process only.

Need for Switching

There are two main reasons behind the switching between User mode and kernel mode, such as:

1. If everything were to run in a single-mode, we would end up with Microsoft's issue in the earlier versions of Windows. If a process were able to exploit a vulnerability, that process then could control the system.
2. Certain conditions are known as a trap, an exception or a system fault typically caused by an exceptional condition such as division by zero, invalid memory access, etc. If the process is running in kernel mode, such a trap situation can crash the entire operating system. A process in user mode that encounters a trap situation only crashes the user-mode process.

So, the overhead of switching is acceptable to ensure a more stable, secure system.

Difference between User Mode and Kernel Mode

- A computer operates either in user mode or kernel mode.
- The difference between User Mode and Kernel Mode is that user mode is the restricted mode in which the applications are running, and kernel-mode is the privileged mode the computer enters when accessing hardware resources.
- The computer is switching between these two modes.
- Frequent context switching can slow down the speed, but it is impossible to execute all processes in the kernel mode.
- That is because; if one process fails, the whole operating system might fail.
- Below are some more differences between User mode and kernel mode, such as:

Terms	User Mode	Kernel Mode
Definition	User Mode is a restricted mode, which the application programs are executing and starts.	Kernel Mode is the privileged mode, which the computer enters when accessing hardware resources.
Modes	User Mode is considered as the slave mode or the restricted mode.	Kernel mode is the system mode, master mode or the privileged mode.
Address Space	In User mode, a process gets its own address space.	In Kernel Mode, processes get a single address space.
Interruptions	In User Mode, if an interrupt occurs, only one process fails.	In Kernel Mode, if an interrupt occurs, the whole operating system might fail.
Restrictions	In user mode, there are restrictions to access kernel programs. Cannot access them directly.	In kernel mode, both user programs and kernel programs can access.

COMPUTING ENVIRONMENTS :

Computing environments include computer machinery, data storage devices, workstations, software applications, and networks that support the processing and exchange of electronic information demanded by software solutions.

Types of Computing Environments:

A brief discussion of how operating systems are used in various types of computing environments are as follows:

i. Traditional Computing: As computing has matured, the line separating many of the traditional computing environments have blurred. Consider the 'typical office environment.' Just a few years ago, it considered PCs connected to a network, with servers providing file and print services. Remote access was awarded and portability was achieved by the use of laptop computers. Terminals attached to the mainframe were prevalent at many companies as well, with even fewer remote access and portability options.

ii. Mobile Computing:

- Mobile computing refers to computing on handheld smartphones and tablet computers.
- These device distinguishing physical features of being portable and lightweight.
- Historically, compared with desktop and laptop computers, mobile systems gave up screen size, memory capacity, and overall functionality in return for handheld mobile access to services such as email and web browsing.
- Over the past few years, however, features of mobile devices have been so rich that the distinction in functionality between, say, a consumer laptop and a tablet computer may be difficult to discern.
- In fact, we might argue that the features of a contemporary mobile device allow it to provide functionality that is easier unavailable or impractical on a desktop or a laptop computer.
- Today, mobile systems are used not only for e-mail and web browsing but also for playing music and video, reading digital books, taking photos, and recording high-definition video.
- Two operating systems currently dominate mobile computing: Apple iOS and Google Android.
- iOS was designed to run on Apple iPhone and iPad mobile devices. Android powers smartphones and tablets computers available from many manufactures.

iii. Distributed Systems:

- A distributed system is a collection of physically separate, possibly heterogeneous, computer systems that are networked to provide users with access to the various resources that the systems maintain.
- Access to a shared resource increases computation speed, functionality, data availability, and reliability.
- Some operating systems generalize network access as a form of file access, with the details of networking contained in the network interfaces device driver.
- Others make users specifically invoke network functions. Generally, systems contain a mix of two modes – for example, FTP and NFS.

- A network operating system is an operating system that provides features such as file sharing across the network, along with a communication scheme that allows a different process on different computers to exchange messages.
- A computer running a network operating system acts autonomously from all other computers on the network, although it is aware of the network and is able to communicate with other networked computers.
- A distributed operating system provides a less autonomous environment. The different computers communicate closely enough to provide the illusion that only a single operating system controls the network.

iv. Client-Server Computing:

- As PCs have become faster, more powerful, and cheaper, designers have shifted away from centralized system architecture.
- Terminals connected to centralized systems are now being supplanted by PCs and mobile devices. Server systems can be broadly categorized as compute servers and file servers:

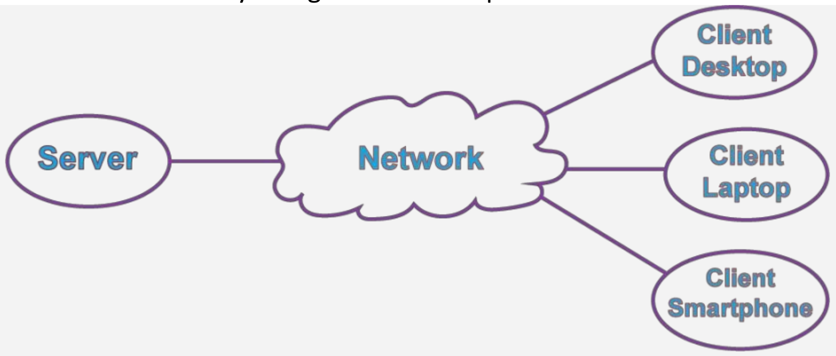


Figure: General structure of a client server system

- **The computer-server** system provides an interface to which a client can send a request to perform an action (for example, read data). In response, the server executes the action and sends the results to the client. A server running a database that responds to client requests for data is an example of such a system.
- **The file-server** system provides a file-system interface where clients can create, update, read, and delete files. An example of such a system is a web server that delivers files to clients running web browsers.

v. Peer-to-Peer Computing:

Another structure for a distributed system is the peer-to-peer (P2P) system, model. In this method, clients and servers are not distinguished from one another

Peer-to-peer systems offer an advantage over traditional client-server systems and services can be provided by several nodes distributed throughout the network.

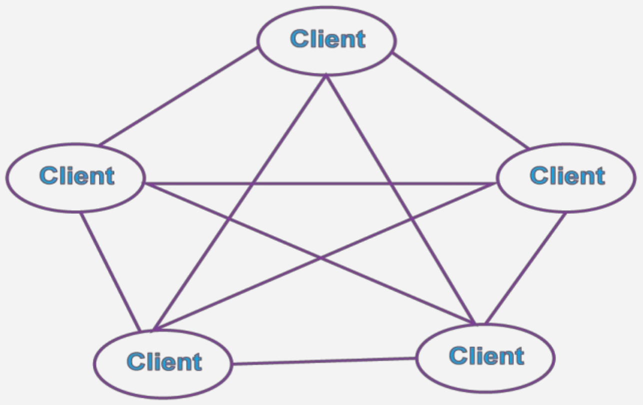


Figure: Peer-to-peer system with no centralized service

To participate in a peer-to-peer system, a node must first join the network of peers. Once a node has joined the network, it can begin providing services to – and requesting services from – other nodes in the network. Determining what services are available is accomplished is one of two general ways:

- When a node joins a network, it registers its services with a centralized lookup service on the network. The remainder of the communication takes place between the client and the server provider.
- An alternative scheme uses no centralized lookup service. Instead, a peer acting as a client must discover what node provides the desired service by broadcasting a request for the service to all other nodes in the network. The node (or nodes) providing that service responds to the peer making the request. A **discovery protocol** must be provided that allows peers to discover services provided by other peers in the network. Skype is an example of peer-to-peer computing. It allows clients to make voice calls and video calls and to send text messages over the internet using a technology known as voice over IP (**VoIP**). It includes a centralized login server, but it also incorporates decentralized peers and allows two peers to communicate.

vi. Virtualization:

Virtualization is a technology that allows operating systems to run as applications within other operating systems. The virtualization industry is vast and growing, which is a testament to its utility and importance.

Broadly speaking, virtualization is one member of a class of software that also includes emulation. Emulation is used when the source CPU type is different from the target CPU type. A common example of emulation occurs when a computer language does not comply with native code but instead is either executed in its high-level form or translated to an intermediate form.

vii. Cloud Computing:

Cloud Computing is a type of computing that delivers computing storage and even applications as a service across a network. There are actually many types of cloud computing, including the following:

- **Public Cloud –** a cloud available via the internet to anyone willing to pay for the services.
- **Private Cloud –** a cloud run by a company for that company's own use.
- **Hybrid Cloud –** a cloud that includes both public and private cloud components.
- **Software as a service (SaaS) –** one or more applications (such as word processors or spreadsheets) available via the internet.
- **Platform as a service (PaaS) –** a software stack ready for application use via the internet (for example, a database server).
- **Infrastructure as a service (IaaS) –**servers or storage available over the internet (for example, storage available for making backup copies of production data).

viii. Real-Time Embedded Systems:

- Embedded computers are the most prevalent form of computers in existence.
- These devices are found everywhere, from car engines and manufacturing robots DVDs and microwave ovens. They tend to have very specific tasks.
- The systems they run on are usually primitive, and so the operating systems provide limited features. Usually, they have little or no user interface, preferring to spend their time monitoring and managing hardware devices, such as automobile engines and robotic arms.
- The use of embedded systems continues to expand. It almost runs a real-time operating system.
- A real-time system is used when rigid time requirements have been placed on the operation of the processor or the flow of data, it is often used as a control device in a dedicated application.

- A real-time system functions correctly only if it returns the correct result within its time constraints.

Operating System - Services

An Operating System provides services to both the users and to the programs.

- It provides programs an environment to execute.
- It provides users the services to execute the programs in a convenient manner.

Following are a few common services provided by an operating system −

- Program execution
- I/O operations
- File System manipulation
- Communication
- Error Detection
- Resource Allocation
- Protection

Program execution

Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process.

A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use). Following are the major activities of an operating system with respect to program management −

- Loads a program into memory.
- Executes the program.
- Handles program's execution.
- Provides a mechanism for process synchronization.
- Provides a mechanism for process communication.
- Provides a mechanism for deadlock handling.

I/O Operation

An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users.

An Operating System manages the communication between user and device drivers.

- I/O operation means read or write operation with any file or any specific I/O device.
- Operating system provides the access to the required I/O device when required.

File system manipulation

A file represents a collection of related information. Computers can store files on the disk (secondary storage), for long-term storage purpose. Examples of storage media include magnetic tape, magnetic disk and optical disk drives like CD, DVD. Each of these media has its own properties like speed, capacity, data transfer rate and data access methods.

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management −

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

Communication

In case of distributed systems which are a collection of processors that do not share memory, peripheral devices, or a clock, the operating system manages communications between all the processes. Multiple processes communicate with one another through communication lines in the network.

The OS handles routing and connection strategies, and the problems of contention and security. Following are the major activities of an operating system with respect to communication −

- Two processes often require data to be transferred between them
- Both the processes can be on one computer or on different computers, but are connected through a computer network.
- Communication may be implemented by two methods, either by Shared Memory or by Message Passing.

Error handling

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling −

- The OS constantly checks for possible errors.
- The OS takes an appropriate action to ensure correct and consistent computing.

Resource Management

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management −

- The OS manages all kinds of resources using schedulers.

- CPU scheduling algorithms are used for better utilization of CPU.

Protection

Considering a computer system having multiple users and concurrent execution of multiple processes, the various processes must be protected from each other's activities.

Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection −

- The OS ensures that all access to system resources is controlled.
- The OS ensures that external I/O devices are protected from invalid access attempts.
- The OS provides authentication features for each user by means of passwords.

User and Operating System Interface

- The user and operating system are connected with each other with the help of interface, so interface is used to connect the user and OS.
- In computers there are different types of interface that can be used for connection with computers to users and their connection is responsible for data transfer.
- Also, in computers there are different interfaces. These interfaces are not necessarily used but can be used in computers whenever it is needed. So, different types of tasks can be performed by the help of different interfaces.

Command line interface

- The command-line interface is an interface whenever the user needs to have different commands regarding the input and output and then a task is performed so this is called the command-line argument and it is used to execute the output and create, delete, print, copy, paste, etc.
- All these operations are performed with the help of the command-line interface.
- The interface is always connected to the OS so that the command given by the user directly works by the OS and a number of operations can be performed with the help of the command line interface because multiple commands can be interrupted at same time and execute only one.
- The command line interface is necessary because all the basic operations in the computer are performed with the help of the OS and it is responsible for memory management.
- By using this we can divide the memory and we can use the memory.

Command Line Interface advantages –

- Controls OS or application
- faster management
- ability to store scripts which helps in automating regular tasks.
- Troubleshoot network connection issues.

Command Line Interface disadvantages –

- The steeper learning curve is associated with memorizing commands and a complex syntax.
- Different commands are used in different shells.

Graphical user interface

- The graphical user interface is used for playing games, watching videos, etc. these are done with the help of GUI because all these applications require graphics.
- The GUI is one of the necessary interfaces because only by using the user can clearly see the picture, play videos.
- So we need GUI for computers and this can be done only with the help of an operating system.
- When a task is performed in the computer then the OS checks the task and defines the interface which is necessary for the task. So, we need GUI in the OS.

The basic components of GUIs are –

- Start menu with program groups
- Taskbar which showing running programs
- Desktop screen
- Different icons and shortcuts.

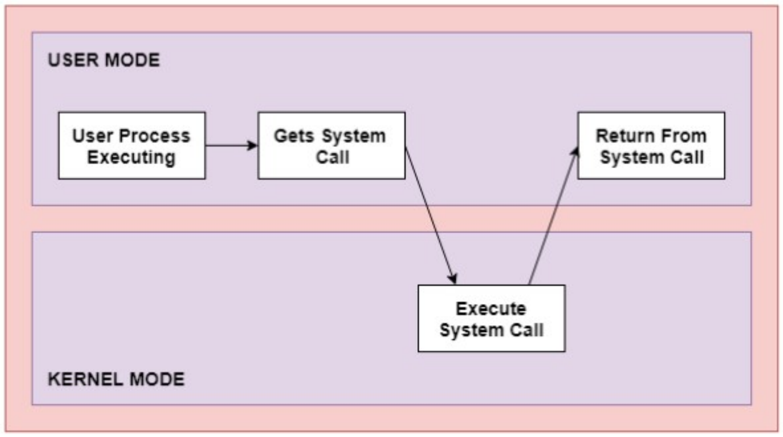
Choice of interface

- The interface that is used with the help of OS for a particular task and that task can be performed with minimum possible time and the output is shown on the screen in that case we use the choice of interface.
- The choice of interface means the OS checks the task and finds out which interface can be suitable for a particular task. So that type of interface is called the choice of interface and this can be done with the help of an OS.

System Calls and Types of System Calls

The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers. System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

A figure representing the execution of the system call is given as follows –



As can be seen from this diagram, the processes execute normally in the user mode until a system call interrupts this. Then the system call is executed on a priority basis in the kernel mode. After the execution of the system call, the control returns to the user mode and execution of user processes can be resumed.

In general, system calls are required in the following situations −

- If a file system requires the creation or deletion of files. Reading and writing from files also require a system call.
- Creation and management of new processes.
- Network connections also require system calls. This includes sending and receiving packets.
- Access to a hardware devices such as a printer, scanner etc. requires a system call.

Types of System Calls

There are mainly five types of system calls. These are explained in detail as follows –

1. Process Control
2. File Management
3. Device Management
4. Information Maintaince
5. Communication

Process Control

These system calls deal with processes such as process creation, process termination etc.

File Management

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

Device Management

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

Information Maintenance

These system calls handle information and its transfer between the operating system and the user program.

Communication

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

Some of the examples of all the above types of system calls in Windows and Unix are given as follows −

Types of System Calls	**Windows**	**Linux**

Process Control	CreateProcess()
	ExitProcess()
	WaitForSingleObject()
		fork()
		exit()
		wait()

File Management	CreateFile()
	ReadFile()
	WriteFile()
	CloseHandle()
		open()
		read()
		write()
		close()

Device Management	SetConsoleMode()
	ReadConsole()
	WriteConsole()
		ioctl()
		read()
		write()

Information Maintenance	GetCurrentProcessID()
	SetTimer()
	Sleep()
		getpid()
		alarm()
		sleep()

Communication	CreatePipe()
	CreateFileMapping()
	MapViewOfFile()
		pipe()
		shmget()
		mmap()

There are many different system calls as shown above. Details of some of those system calls are as follows –

open()

- The open() system call is used to provide access to a file in a file system.
- This system call allocates resources to the file and provides a handle that the process uses to refer to the file.
- A file can be opened by multiple processes at the same time or be restricted to one process. It all depends on the file organisation and file system.

read()

- The read() system call is used to access data from a file that is stored in the file system.
- The file to read can be identified by its file descriptor and it should be opened using open() before it can be read.
- In general, the read() system calls takes three arguments i.e. the file descriptor, buffer which stores read data and number of bytes to be read from the file.

write()

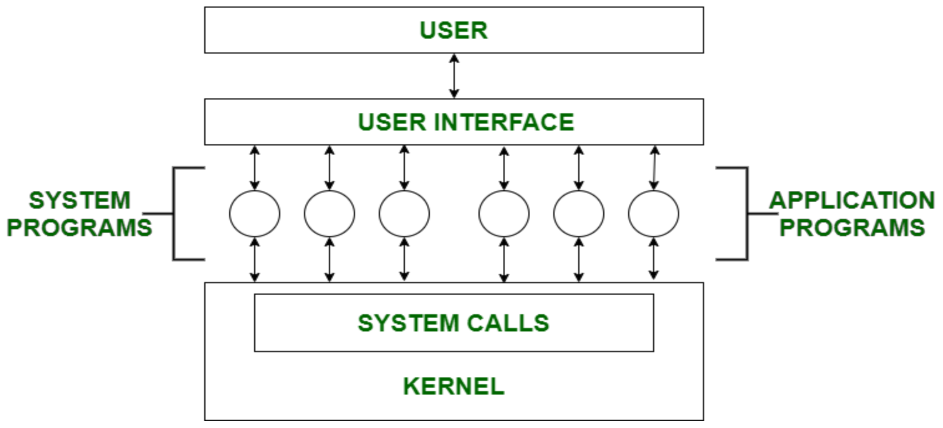
- The write() system calls writes the data from a user buffer into a device such as a file.
- This system call is one of the ways to output data from a program.
- In general, the write system calls takes three arguments i.e. file descriptor, pointer to the buffer where data is stored and number of bytes to write from the buffer.

close()

- The close() system call is used to terminate access to a file system.
- Using this system call means that the file is no longer required by the program and so the buffers are flushed, the file metadata is updated and the file resources are de-allocated.

System Programs

- **System Programming** can be defined as the act of building Systems Software using System Programming Languages.
- According to Computer Hierarchy, one which comes at last is Hardware.
- Then it is Operating System, System Programs, and finally Application Programs.
- Program Development and Execution can be done conveniently in System Programs.
- Some of the System Programs are simply user interfaces, others are complex. It traditionally lies between the user interface and system calls.



So here, the user can only view up-to-the System Programs he can't see System Calls.
System Programs can be divided into these categories :

1. **File Management –**
   A file is a collection of specific information stored in the memory of a computer system. File management is defined as the process of manipulating files in the computer system, its management includes the process of creating, modifying and deleting files.
   - It helps to create new files in the computer system and placing them at specific locations.
   - It helps in easily and quickly locating these files in the computer system.
   - It makes the process of sharing files among different users very easy and user-friendly.
   - It helps to store files in separate folders known as directories.
   - These directories help users to search files quickly or to manage files according to their types of uses.
   - It helps users to modify the data of files or to modify the name of files in directories.

2. **Status Information –**
   Information like date, time amount of available memory, or disk space is asked by some users. Others providing detailed performance, logging, and debugging information which is more complex.

All this information is formatted and displayed on output devices or printed. Terminal or other output devices or files or a window of GUI is used for showing the output of programs.

3. **File Modification –**
   For modifying the contents of files we use this. For Files stored on disks or other storage devices, we used different types of editors. For searching contents of files or perform transformations of files we use special commands.

4. **Programming-Language support –**
   For common programming languages, we use Compilers, Assemblers, Debuggers, and interpreters which are already provided to users. It provides all support to users. We can run any programming language. All languages of importance are already provided.

5. **Program Loading and Execution –**
   When the program is ready after Assembling and compilation, it must be loaded into memory for execution. A loader is part of an operating system that is responsible for loading programs and libraries. It is one of the essential stages for starting a program. Loaders, relocatable loaders, linkage editors, and Overlay loaders are provided by the system.

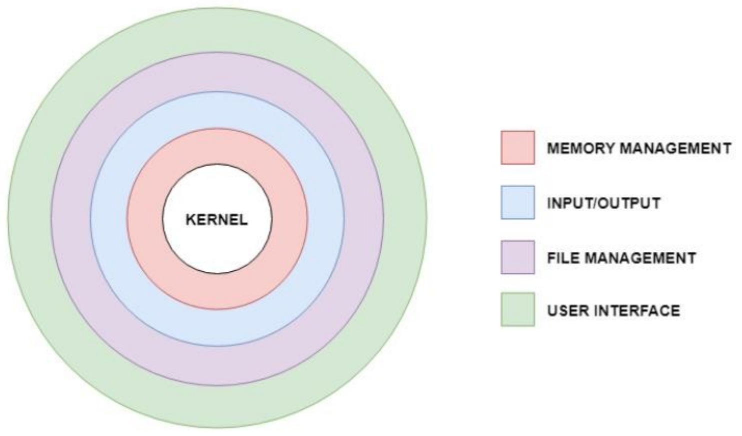
6. **Communications –**
   Virtual connections among processes, users, and computer systems are provided by programs. Users can send messages to another user on their screen, User can send e-mail, browsing on web pages, remote login, the transformation of files from one user to another.

Some examples of system program in O.S. are –

- Windows 10
- Mac OS X
- Ubuntu
- Linux
- Unix
- Android
- Anti-virus
- Disk formatting
- Computer language translators

Operating System Design and Implementation

- An operating system is a construct that allows the user application programs to interact with the system hardware.
- Operating system by itself does not provide any function but it provides an atmosphere in which different applications and programs can do useful work.
- There are many problems that can occur while designing and implementing an operating system. These are covered in operating system design and implementation.



Layered Operating System Design

Operating System Design Goals

It is quite complicated to define all the goals and specifications of the operating system while designing it.The design changes depending on the type of the operating system i.e if it is batch system, time shared system, single user system, multi user system, distributed system etc.

There are basically two types of goals while designing an operating system. These are −

User Goals

The operating system should be convenient, easy to use, reliable, safe and fast according to the users. However, these specifications are not very useful as there is no set method to achieve these goals.

System Goals

The operating system should be easy to design, implement and maintain. These are specifications required by those who create, maintain and operate the operating system. But there is not specific method to achieve these goals as well.

Operating System Mechanisms and Policies

- There is no specific way to design an operating system as it is a highly creative task. However, there are general software principles that are applicable to all operating systems.
- A subtle difference between mechanism and policy is that mechanism shows how to do something and policy shows what to do.
- Policies may change over time and this would lead to changes in mechanism.
- So, it is better to have a general mechanism that would require few changes even when a policy change occurs.
- For example - If the mechanism and policy are independent, then few changes are required in mechanism if policy changes.
- If a policy favours I/O intensive processes over CPU intensive processes, then a policy change to preference of CPU intensive processes will not change the mechanism.

Operating System Implementation

- The operating system needs to be implemented after it is designed.

- Earlier they were written in assembly language but now higher level languages are used.
- The first system not written in assembly language was the Master Control Program (MCP) for Burroughs Computers.

Advantages of Higher Level Language

- There are multiple advantages to implementing an operating system using a higher level language such as: the code is written more fast,
- it is compact and also easier to debug and understand.
- Also, the operating system can be easily moved from one hardware to another if it is written in a high level language.

Disadvantages of Higher Level Language

- Using high level language for implementing an operating system leads to a loss in speed and increase in storage requirements.
- However in modern systems only a small amount of code is needed for high performance, such as the CPU scheduler and memory manager.
- Also, the bottleneck routines in the system can be replaced by assembly language equivalents if required.

Operating-System Structure

For efficient performance and implementation an OS should be partitioned into separate subsystems, each with carefully defined tasks, inputs, outputs, and performance characteristics.

These subsystems can then be arranged in various architectural configurations:

Simple Structure

When DOS was originally written its developers had no idea how big and important it would eventually become.
It was written by a few programmers in a relatively short amount of time, without the benefit of modern software engineering techniques, and then gradually grew over time to exceed its original expectations.
It does not break the system into subsystems, and has no distinction between user and kernel modes, allowing all programs direct access to the underlying hardware.

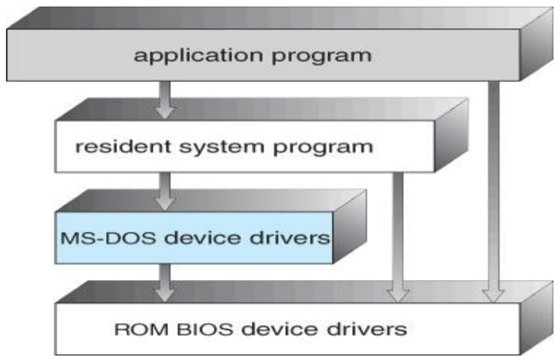


Figure 2.11 - MS-DOS layer structure

The original UNIX OS used a simple layered approach, but almost all the OS was in one big layer, not really breaking the OS down into layered subsystems:

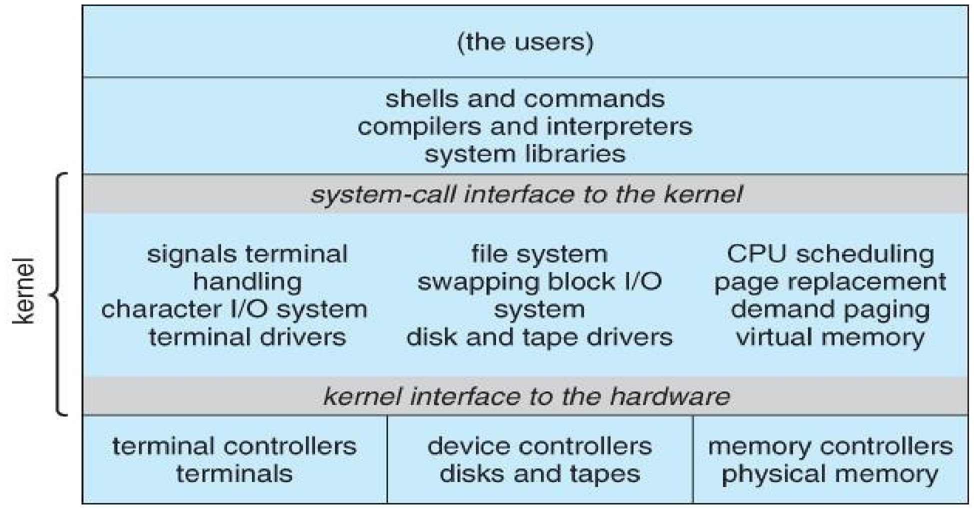
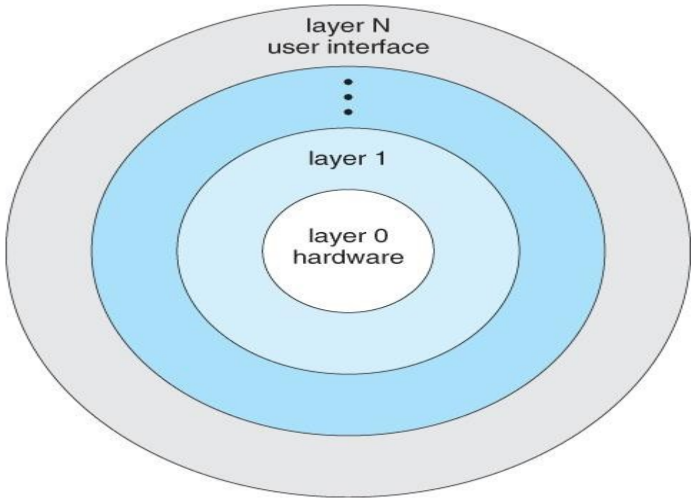


Figure 2.12 - Traditional UNIX system structure

Layered Approach

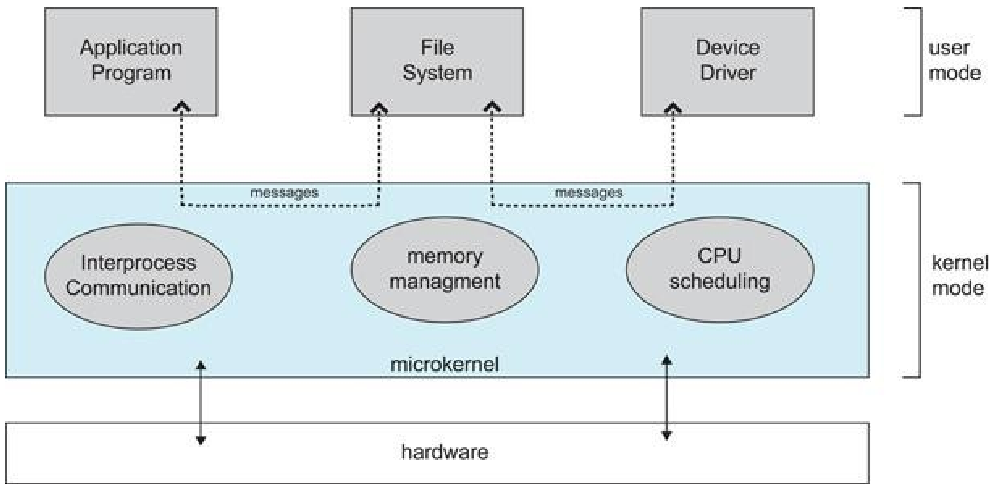
- Another approach is to break the OS into a number of smaller layers, each of which rests on the layer below it, and relies solely on the services provided by the next lower layer.
- This approach allows each layer to be developed and debugged independently, with the assumption that all lower layers have already been debugged and are trusted to deliver proper services.
- The problem is deciding what order in which to place the layers, as no layer can call upon the services of any higher layer, and so many chicken-and-egg situations may arise.
- Layered approaches can also be less efficient, as a request for service from a higher layer has to filter through all lower layers before it reaches the HW, possibly with significant processing at each step.



- A layered operating system

Microkernels

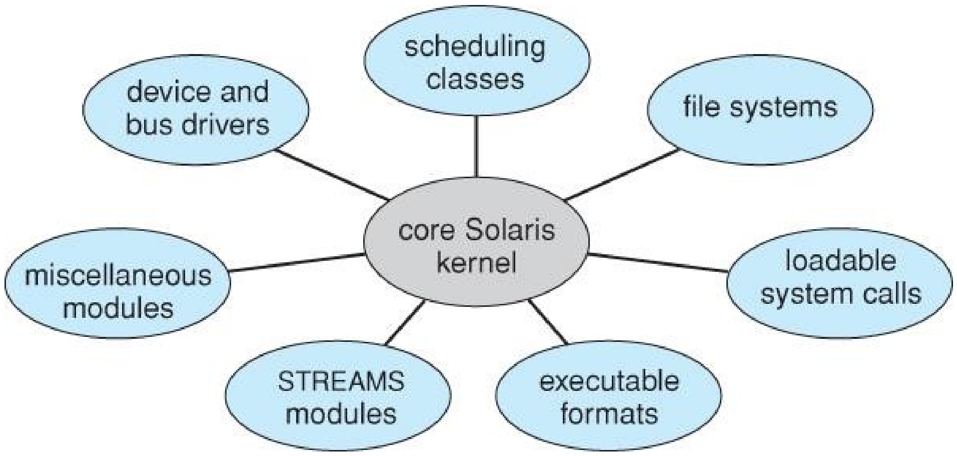
- The basic idea behind micro kernels is to remove all non-essential services from the kernel, and implement them as system applications instead, thereby making the kernel as small and efficient as possible.
- Most microkernels provide basic process and memory management, and message passing between other services, and not much more.
- 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.



- Architecture of a typical microkernel

Modules

- Modern OS development is object-oriented, with a relatively small core kernel and a set of ***modules*** which can be linked in dynamically. See for example the Solaris structure, as shown in Figure 2.13 below.
- Modules are similar to layers in that each subsystem has clearly defined tasks and interfaces, but any module is free to contact any other module, eliminating the problems of going through multiple intermediary layers, as well as the chicken-and-egg problems.
- The kernel is relatively small in this architecture, similar to microkernels, but the kernel does not have to implement message passing since modules are free to contact each other directly.



- Solaris loadable modules

Hybrid Systems

- Most OSes today do not strictly adhere to one architecture, but are hybrids of several.

Mac OS X

- The Max 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:

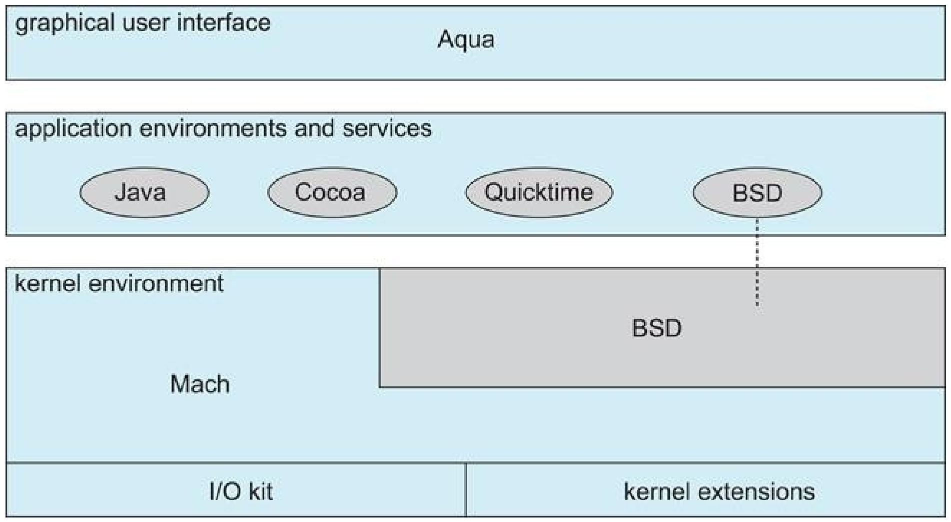
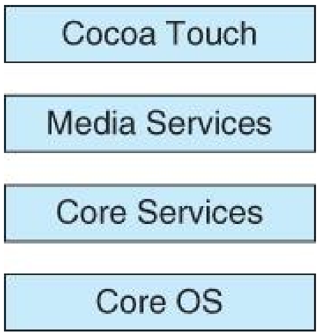


Figure 2.16 - The Mac OS X structure

iOS

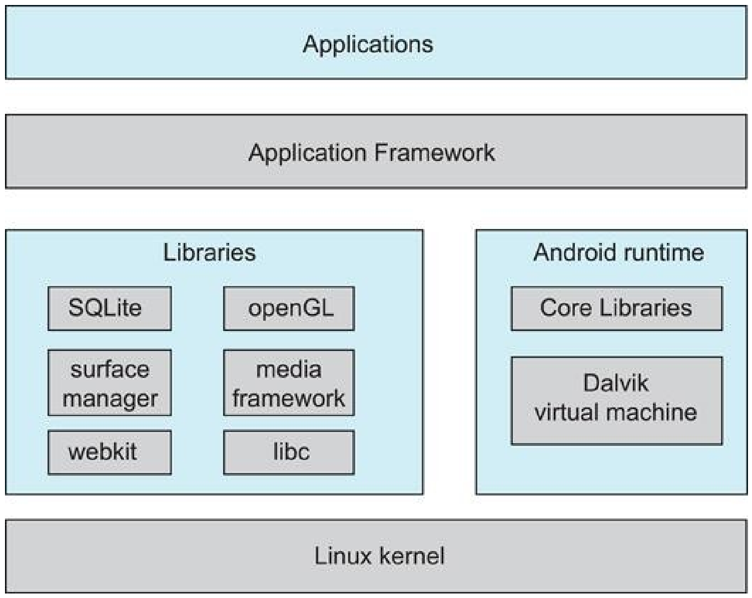
- The **iOS** operating system was developed by Apple for iPhones and iPads. It runs with less memory and computing power needs than Max OS X, and supports touchscreen interface and graphics for small screens:



- Architecture of Apple's iOS.

Android

- The Android OS was developed for Android smartphones and tablets by the Open Handset Alliance, primarily Google.
- Android is an open-source OS, as opposed to iOS, which has lead to its popularity.
- Android includes versions of Linux and a Java virtual machine both optimized for small platforms.
- Android apps are developed using a special Java-for-Android development environment.



- Architecture of Google's Android

Operating-System Debugging

Kernighan's Law

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

- Debugging here includes both error discovery and elimination and performance tuning.

Failure Analysis

- Debuggers allow processes to be executed stepwise, and provide for the examination of variables and expressions as the execution progresses.
- Profilers can document program execution, to produce statistics on how much time was spent on different sections or even lines of code.
- If an ordinary process crashes, a memory dump of the state of that process's memory at the time of the crash can be saved to a disk file for later analysis.
    - The program must be specially compiled to include debugging information, which may slow down its performance.
- These approaches don't really work well for OS code, for several reasons:
    - The performance hit caused by adding the debugging (tracing) code would be unacceptable. (Particularly if one tried to "single-step" the OS while people were trying to use it to get work done!)
    - Many parts of the OS run in kernel mode, and make direct access to the hardware.
    - If an error occurred during one of the kernel's file-access or direct disk-access routines, for example, then it would not be practical to try to write a crash dump into an ordinary file on the filesystem.
        - Instead the kernel crash dump might be saved to a special unallocated portion of the disk reserved for that purpose.

Performance Tuning

- Performance tuning (debottlenecking) requires monitoring system performance.
- One approach is for the system to record important events into log files, which can then be analyzed by other tools. These traces can also be used to evaluate how a proposed new system would perform under the same workload.
- Another approach is to provide utilities that will report system status upon demand, such as the unix "top" command. (w, uptime, ps, etc.)
- System utilities may provide monitoring support.

DTrace

- DTrace is a special facility for tracing a running OS, developed for Solaris 10.
- DTrace adds "probes" directly into the OS code, which can be queried by "probe consumers".
- Probes are removed when not in use, so the DTrace facility has zero impact on the system when not being used, and a proportional impact in use.

System Boot

The general approach when most computers boot up goes something like this:

- When the system powers up, an interrupt is generated which loads a memory address into the program counter, and the system begins executing instructions found at that address. This address points to the "bootstrap" program located in ROM chips (or EPROM chips) on the motherboard.
- The ROM bootstrap program first runs hardware checks, determining what physical resources are present and doing power-on self tests (POST) of all HW for which this is applicable. Some devices, such as controller cards may have their own on-board diagnostics, which are called by the ROM bootstrap program.
- The user generally has the option of pressing a special key during the POST process, which will launch the ROM BIOS configuration utility if pressed. This utility allows the user to specify and configure certain hardware parameters as where to look for an OS and whether or not to restrict access to the utility with a password.
    - Some hardware may also provide access to additional configuration setup programs, such as for a RAID disk controller or some special graphics or networking cards.
- Assuming the utility has not been invoked, the bootstrap program then looks for a non-volatile storage device containing an OS. Depending on configuration, it may look for a floppy drive, CD ROM drive, or primary or secondary hard drives, in the order specified by the HW configuration utility.
- Assuming it goes to a hard drive, it will find the first sector on the hard drive and load up the fdisk table, which contains information about how the physical hard drive is divided up into logical partitions, where each partition starts and ends, and which partition is the "active" partition used for booting the system.
- There is also a very small amount of system code in the portion of the first disk block not occupied by the fdisk table. This bootstrap code is the first step that is not built into the hardware, i.e. the first part which might be in any way OS-specific. Generally this code knows just enough to access the hard drive, and to load and execute a (slightly) larger boot program.
- For a single-boot system, the boot program loaded off of the hard disk will then proceed to locate the kernel on the hard drive, load the kernel into memory, and then transfer control over to the kernel. There may be some opportunity to specify a particular kernel to be loaded at this stage, which may be useful if a new kernel has just been generated and doesn't work, or if the system has multiple kernels available with different configurations for different purposes. (Some systems may boot different configurations automatically, depending on what hardware has been found in earlier steps.)
- For dual-boot or multiple-boot systems, the boot program will give the user an opportunity to specify a particular OS to load, with a default choice if the user does not pick a particular OS within a given time frame. The boot program then finds the boot loader for the chosen single-boot OS, and runs that program as described in the previous bullet point.
- Once the kernel is running, it may give the user the opportunity to enter into single-user mode, also known as maintenance mode. This mode launches very few if any system services, and does not enable any logins other than the primary log in on the console. This mode is used primarily for system maintenance and diagnostics.
- When the system enters full multi-user multi-tasking mode, it examines configuration files to determine which system services are to be started, and launches each of them in turn. It then

spawns login programs (gettys) on each of the login devices which have been configured to enable user logins.

  - (The getty program initializes terminal I/O, issues the login prompt, accepts login names and passwords, and authenticates the user. If the user's password is authenticated, then the getty looks in system files to determine what shell is assigned to the user, and then "execs" (becomes) the user's shell. The shell program will look in system and user configuration files to initialize itself, and then issue prompts for user commands. Whenever the shell dies, either through logout or other means, then the system will issue a new getty for that terminal device.)

- An operating system is a control program that manages the execution of user programs