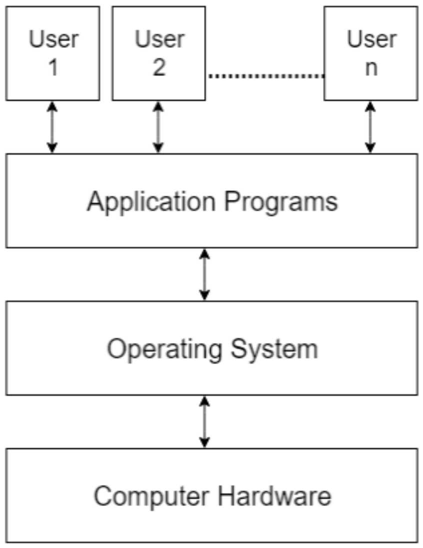
Unit-1

Introduction of Operating System

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.

The place of an operating system in a computer can be demonstrated aptly using the following diagram:



As can be seen from the above diagram, the computer system is divided into four components namely, hardware, operating system, application programs and users.

The operating system coordinates between the hardware and the application programs. It makes sure that adequate hardware resources are distributed evenly among various applications so they can all complete their execution.

➢ *Goals of Operating System*

In general, an operating system provides mainly two goals. These are:

- ***Convenience***

An operating system should make a computer easy and convenient to use. The interface should make the interaction with a computer system simple and hassle free.

- ***Efficiency***

The operating system should provide hardware resources to the different application programs in such a way to maximize efficiency.

➢ *Operation of an OS*

The primary concerns of an OS during its operation are execution of programs, use of resources, and prevention of interference with programs and resources. Accordingly, its three principal functions are:

- ***Program management:*** The OS initiates programs, arranges their execution on the CPU, and terminates them when they complete their execution. Since many programs exist in the system at any time, the OS performs a function called scheduling to select a program for execution.
- ***Resource management:*** The OS allocates resources like memory and I/O devices when a program needs them. When the program terminates, it deallocates these resources and allocates them to other programs that need them.
- ***Security and protection:*** The OS implements noninterference in users' activities through joint actions of the security and protection functions.

❖ Types of Operating systems

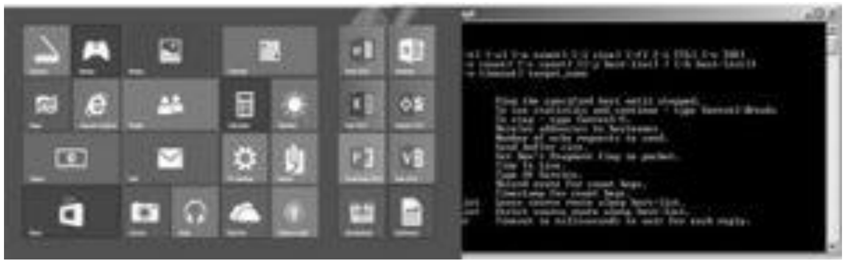
On a general basis, the computer operating systems are essentially categorized into two types:

1. Normal Operating System
2. Real-Time Operating System

1. *Normal Operating System*

The normal operating system is further classified into two types:

- Character User Interface Operating System
- Graphical User Interface Operating System



GUI and CUI

• *Character User Interface Operating System (CUI)*

The CUI operating system is a text-based operating system, which is used for interacting with the software or files by typing commands to perform specific tasks. The command-line operating system uses only the keyboard to enter commands. The command-line operating systems include DOS and UNIX. The advanced command-line operating system is faster than the advanced GUI operating system.

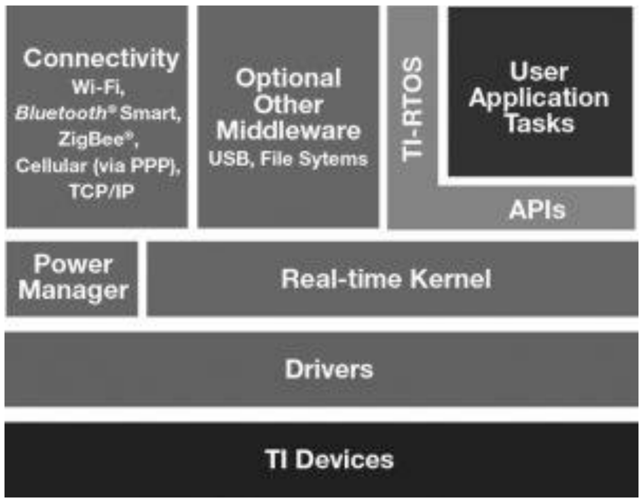
- ***Graphical User Interface Operating System (GUI)***

The graphical mode interface operating system is a mouse-based operating system (Windows Operating System, LINUX), wherein a user performs the tasks or operations without typing the commands from the keyboard. The files or icons can be opened or closed by clicking them with a mouse button.

In addition to this, the mouse and keyboard are used to control the GUI operating systems for several purposes. Most of the embedded-based projects are developed on this operating system. The advanced GUI operating system is slower than the command line operating system.

2. *Real-time Operating System*

Real-time operating systems are also known as multitasking operating systems. The normal operating system is responsible for managing the hardware resources of a computer. The RTOS performs these tasks, but it is specially designed to run applications at a scheduled or precise time with high reliability.



RTOS

A real-time operating system is designed for real-time applications, such as embedded systems, industrial robots, scientific research equipment, and others. There are different types of operating systems in real-time, such as soft real-time operating systems and hard real-time operating systems.

Examples of RTOS

- Linux
- VxWorks
- TRON
- Windows CE

- ***Hard Real-time System***

The hard real-time system is a purely time constant system. For a hard real-time operating system, finishing the tasks within a deadline is very important for efficient system performance.

For example, for a given input, if a user expects the output after 10seconds, then the system should process the input data and give the output exactly after 10seconds. Here, the deadline is 10 seconds, and therefore, the system should not give the output after 11th sec or 9th sec. Therefore, hard real-time systems are used in the army and defense.

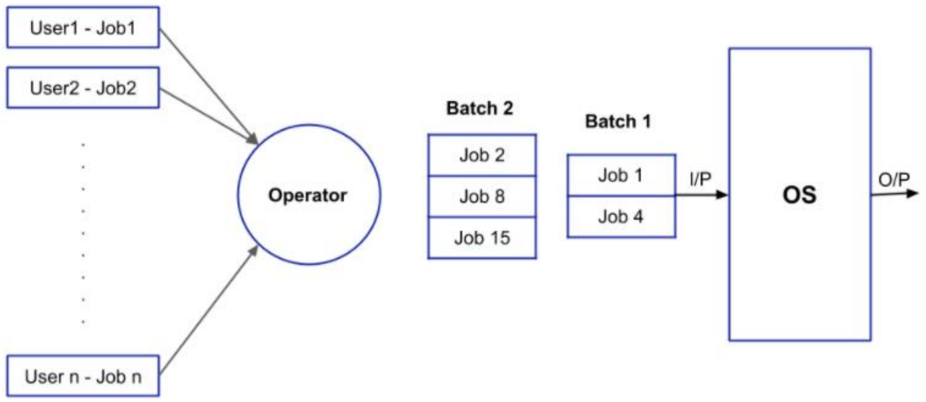
- ***Soft Real-time System***

For a soft real-time system, meeting the deadline is not mandatory for every task. Therefore, a soft real-time system can miss the deadline by one or two seconds. However, if the system misses deadlines every time, this will degrade the system performance. Computers, audio, and video systems are examples of soft real-time systems. Nowadays, Androids are being widely used for applications like automatic gate openers.

In addition, there are many other different types of the operating systems on the computer along with their advantages and disadvantages. A few of the types can be explained as follows:

1. Batch Operating System:

This type of operating system does not interact with the computer directly. There is an operator which takes similar jobs having the same requirement and groups them into batches. It is the responsibility of the operator to sort jobs with similar needs.



Advantages:

- It is very difficult to guess or know the time required for any job to complete. Processors of the batch systems know how long the job would be when it is in queue

- Multiple users can share the batch systems
- The idle time for the batch system is very less
- It is easy to manage large work repeatedly in batch systems

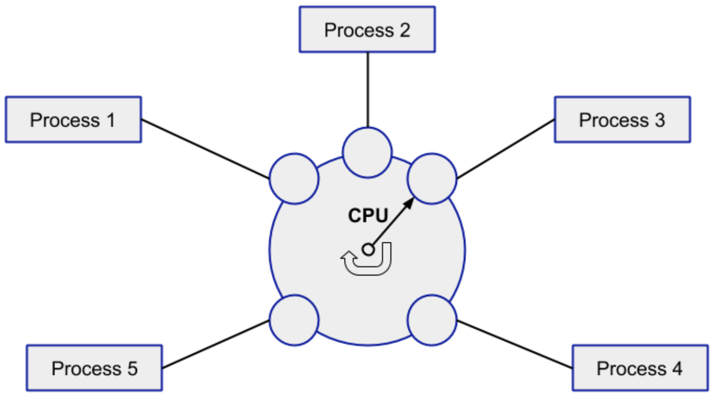
Disadvantages:

- The computer operators should be well known with batch systems
- Batch systems are hard to debug
- It is sometimes costly
- The other jobs will have to wait for an unknown time if any job fails

Examples: Payroll System, Bank Statements, etc.

2. *Time-Sharing Operating Systems:*

Each task is given some time to execute so that all the tasks work smoothly. Each user gets the time of CPU as they use a single system. These systems are also known as Multitasking Systems. The task can be from a single user or different users also. The time that each task gets to execute is called quantum. After this time interval is over OS switches over to the next task.



Advantages:

- Each task gets an equal opportunity
- Fewer chances of duplication of software
- CPU idle time can be reduced

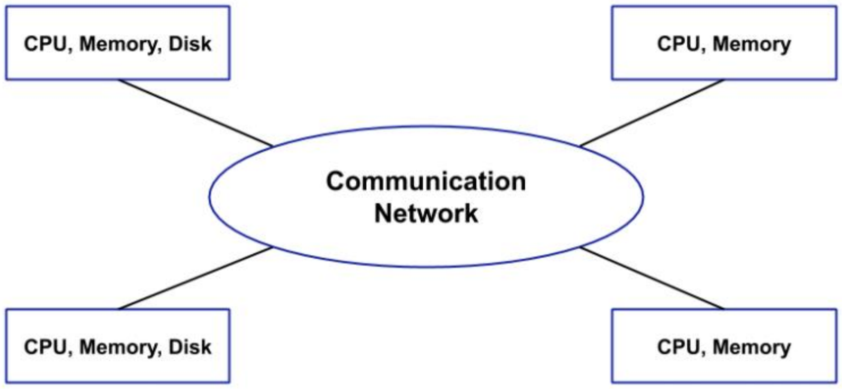
Disadvantages:

- Reliability problem
- One must have to take care of the security and integrity of user programs and data
- Data communication problem

Examples: Multics, Unix, etc.

3. *Distributed Operating System*

In a Distributed Operating System, we have various systems and all these systems have their own CPU, main memory, secondary memory, and resources. These systems are connected to each other using a shared communication network. Here, each system can perform its task individually. The best part about these Distributed Operating System is remote access i.e. one user can access the data of the other system and can work accordingly. So, remote access is possible in these distributed Operating Systems.



Advantages:

- Since the systems are connected with each other so, the failure of one system can't stop the execution of processes because other systems can do the execution.
- Resources are shared between each other.
- The load on the host computer gets distributed and this, in turn, increases the efficiency.

Disadvantages:

- Since the data is shared among all the computers, so to make the data secure and accessible to few computers, you need to put some extra efforts.
- If there is a problem in the communication network then the whole communication will be broken.

Examples: LOCUS, etc.

4. *Embedded Operating System*

An Embedded Operating System is designed to perform a specific task for a particular device which is not a computer. For example, the software used in elevators is dedicated to the working of elevators only and nothing else. So, this can be an example of Embedded Operating System. The Embedded Operating System allows the access of device hardware to the software that is running on the top of the Operating System.

Advantages:

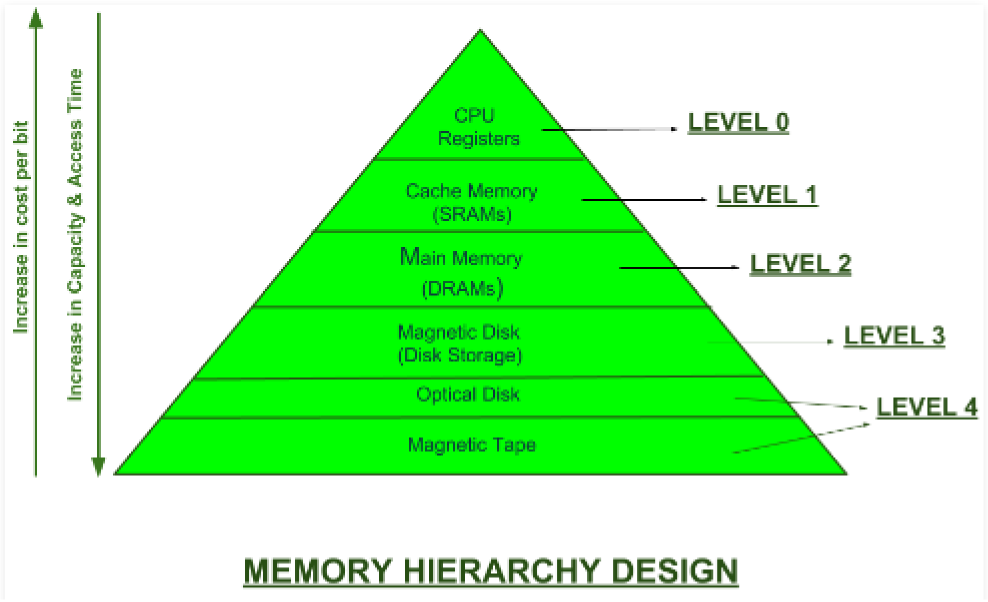
- Since it is dedicated to a particular job, so it is fast.
- Low cost.
- These consume less memory and other resources.

Disadvantages:

- Only one job can be performed.
- It is difficult to upgrade or is nearly scalable.

❖ Memory Hierarchy

In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references. The figure below clearly demonstrates the different levels of memory hierarchy:



This Memory Hierarchy Design is divided into 2 main types:

1. **External Memory or Secondary Memory:** Comprising of Magnetic Disk, Optical Disk, and Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.
2. **Internal Memory or Primary Memory:** Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

Memory Hierarchy Design:

- **Level-0:**
  - At level-0, registers are present which are contained inside the CPU.
  - Since they are present inside the CPU, they have least access time.
  - They are most expensive and therefore smallest in size (in KB).
  - Registers are implemented using Flip-Flops.
- **Level-1:**
  - At level-1, Cache Memory is present.
  - It stores the segments of program that are frequently accessed by the processor.
  - It is expensive and therefore smaller in size (in MB).
  - Cache memory is implemented using static RAM.
- **Level-2:**
  - At level-2, main memory is present.
  - It can communicate directly with the CPU and with auxiliary memory devices through an I/O processor.
  - It is less expensive than cache memory and therefore larger in size (in few GB).
  - Main memory is implemented using dynamic RAM.
- **Level-3:**
  - At level-3, secondary storage devices like Magnetic Disk are present.
  - They are used as back up storage.
  - They are cheaper than main memory and therefore much larger in size (in few TB).
- **Level-4:**
  - At level-4, tertiary storage devices like magnetic tape are present.
  - They are used to store removable files.
  - They are cheapest and largest in size (1-20 TB).

- ***Goals of Memory Hierarchy:***

The goals of memory hierarchy are-

- To obtain the highest possible average access speed
- To minimize the total cost of the entire memory system

❖ Interrupts

Interrupts are signals sent to the CPU by external devices, normally I/O devices. They tell the CPU to stop its current activities and execute the appropriate part of the operating system.

There are three types of interrupts:

1. **Hardware Interrupts** are generated by hardware devices to signal that they need some attention from the OS. They may have just received some data (e.g., keystrokes on the keyboard or a data on the Ethernet card); or they have just completed a task which the operating system previous requested, such as transferring data between the hard drive and memory.

2. **Software Interrupts** are generated by programs when they want to request a system call to be performed by the operating system.

3. **Traps** are generated by the CPU itself to indicate that some error or condition occurred for which assistance from the operating system is needed.

Interrupts are important because they give the user better control over the computer. Without interrupts, a user may have to wait for a given application to have a higher priority over the CPU to be run. This ensures that the CPU will deal with the process immediately.

❖ Resource Allocation Techniques for Processes

The Operating System allocates resources when a program needs them. When the program terminates, the resources are de-allocated, and allocated to other programs that need them. Now the question is, what strategy does the operating system use to allocate these resources to user programs?

There are two Resource allocation techniques:

1. Resource partitioning approach:

In this approach, the operating system decides beforehand, that what resources should be allocated to which user program. It divides the resources in the system to many *resource partitions*, where each partition may include various resources – for example, 1 MB memory, disk blocks, and a printer.

Then, it allocates one resource partition to each user program before the program's initiation. A resource table records the resource partition and its current allocation status (Allocated or Free).

Advantages:

- Easy to Implement
- Less Overhead

Disadvantages:

- **Lacks flexibility –** if a resource partition contains more resources than what a particular process requires, the additional resources are wasted.
- If a program needs more resources than a single resource partition, it cannot execute (Though free resources are present in other partitions).

2. *Pool based approach:*

In this approach, there is a *common pool of resources*. The operating System checks the allocation status in the resource table whenever a program makes a request for a resource. If the resource is free, it allocates the resource to the program.

Advantages:

- Allocated resources are not wasted.
- Any resource requirement can be fulfilled if the resource is free (unlike partitioning approach).

Disadvantages:

- Overhead of allocating and de-allocating the resources on every request and release.

❖ Structure of Operating Systems

During the lifetime of an operating system, we can expect several changes to take place in computer systems and computing environments. To adapt an operating system to these changes, it should be easy to implement the OS on a new computer system, and to add new functionalities to it. These requirements are called portability and extensibility of an operating system, respectively.

Porting is the act of adapting software for use in a new computer system. Portability refers to the ease with which a software program can be ported—it is inversely proportional to the porting effort. Extensibility refers to the ease with which new functionalities can be added to a software system.

The structure of an operating system concerns the nature of the OS core and other parts of the operating system, and their interactions with one another.

➢ Operating Systems with Monolithic Structure

Early operating systems had a monolithic structure, whereby the OS formed a single software layer between the user and the bare machine, i.e., the computer system's hardware. The user interface was provided by a command interpreter. The command interpreter organized creation of user processes. Both the command interpreter and user processes invoked OS functionalities and services through system calls.

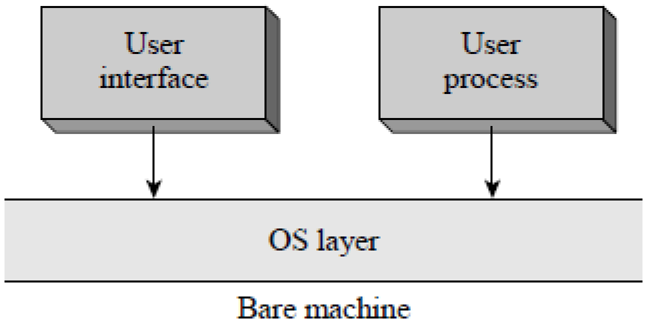


Figure: *Monolithic OS.*

Two kinds of problems with the monolithic structure were realized over a period of time. The OS layer had an interface with the bare machine. Hence architecture-dependent code was spread throughout the OS, and so there was poor portability. It also made testing and debugging difficult, leading to high costs of maintenance and enhancement. These problems led to the search for alternative ways to structure an OS. In the following three methods of structuring an OS that have been implemented as solutions to these problems.

Layered structure: The layered structure attacks the complexity and cost of developing and maintaining an OS by structuring it into a number of layers.

Kernel-based structure: The kernel-based structure confines architecture dependence to a small section of the OS code that constitutes the kernel, so that portability is increased. The Unix OS has a kernel-based structure.

Microkernel-based OS structure: The microkernel provides a minimal set of facilities and services for implementing an OS. Its use provides portability. It also provides extensibility because changes can be made to the OS without requiring changes in the microkernel.

➢ Layered Design of Operating Systems

The monolithic OS structure suffered from the problem that all OS components had to be able to work with the bare machine. This feature increased the cost and effort in developing an OS because of the large semantic gap between the operating system and the bare machine.

Semantic Gap: The mismatch between the nature of operations needed in the application and the nature of operations provided in the machine.

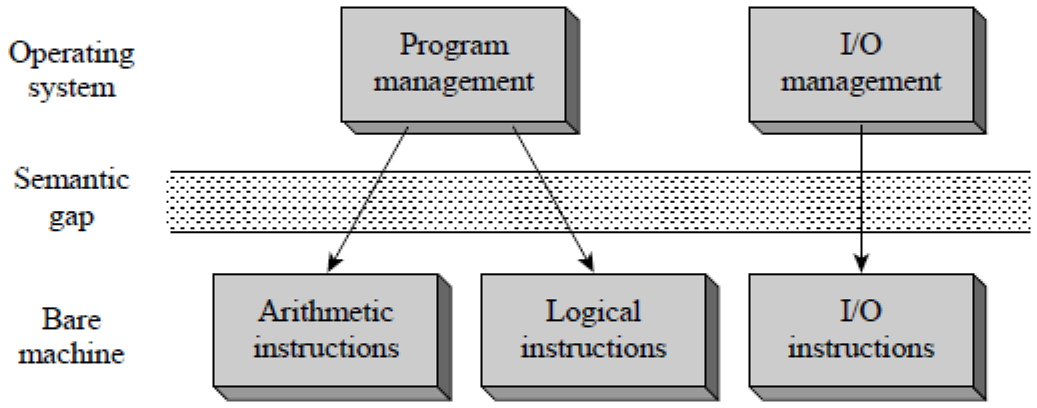


Figure: *Semantic gap.*

An OS module may contain an algorithm, say, that uses OS-level primitive operations like saving the context of a process and initiating an I/O operation. These operations are more complex than the machine-level primitive operations. This difference leads to a large semantic gap, which has to be bridged through programming and it leads to high programming costs.

The semantic gap between an OS and the machine on which it operates can be reduced by either using a more capable machine—a machine that provides instructions to perform some (or all) operations that operating systems have to perform—or by simulating a more capable machine in the software.

The simulator, which is a program, executes on the bare machine and mimics a more powerful machine that has many features desired by the OS. This new "machine" is called an extended machine, and its simulator is called the extended machine software. Now the OS interfaces with the extended machine rather than with the bare machine; the extended machine software forms a layer between the OS and the bare machine.

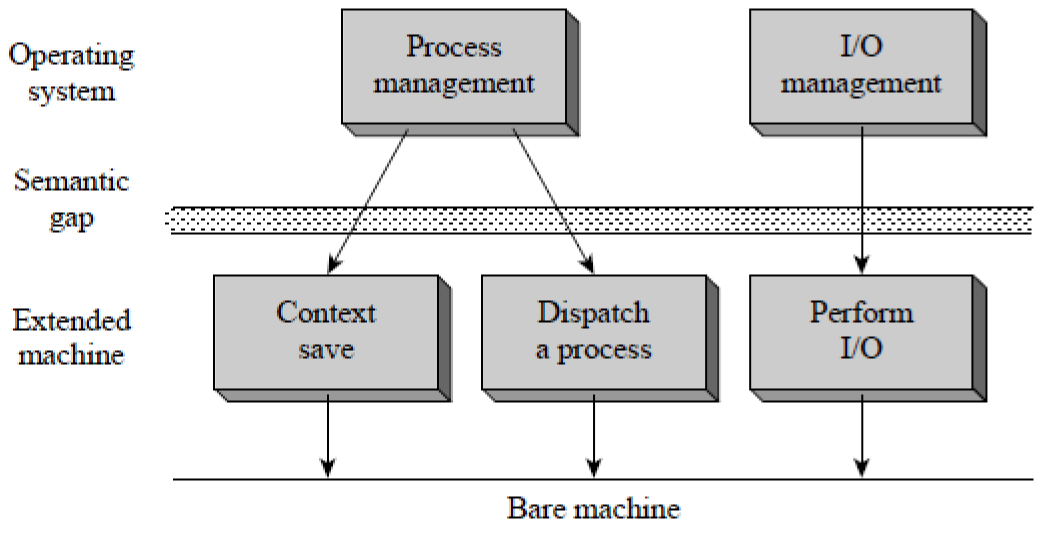


Figure: *Layered OS design.*

The above figure illustrates a two-layered OS. The extended machine provides operations like context save, dispatching, swapping, and I/O initiation. The operating system layer is located on top of the extended machine layer. This arrangement considerably simplifies the coding and testing of OS modules by separating the algorithm of a function from the implementation of its primitive operations. It is now easier to test, debug, and modify an OS module than in a monolithic OS.

➢ Kernel-Based Operating Systems

The kernel is the core of the OS; it provides a set of functions and services to support various OS functionalities. The rest of the OS is organized as a set of nonkernel routines, which implement operations on processes and resources that are of interest to users, and a user interface.

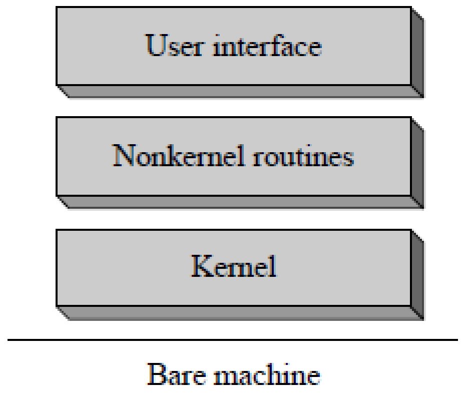


Figure: *Structure of a kernel-based OS.*

The operation of the kernel is interrupt-driven. The kernel gets control when an interrupt such as a timer interrupt or an I/O completion interrupt notifies occurrence of an event to it, or when the software-interrupt instruction is executed to make a system call. When the interrupt occurs, an interrupt servicing routine performs the context save function and invokes an appropriate event handler, which is a nonkernel routine of the OS.

A system call may be made by the user interface to implement a user command, by a process to invoke a service in the kernel, or by a nonkernel routine to invoke a function of the kernel. For example, when a user issues a command to execute the program stored in some file, say file alpha, the user interface makes a system call, and the interrupt servicing routine invokes a nonkernel routine to set up execution of the program. The nonkernel routine would make system calls to allocate memory for the program's execution, open file alpha, and load its contents into the allocated memory area, followed by another system call to initiate operation of the process that represents execution of the program.

The motivations for the kernel-based OS structure were portability of the OS and convenience in the design and coding of nonkernel routines. Portability of the OS is achieved by putting architecture-dependent parts of OS code in the kernel and keeping architecture-independent parts of code outside it, so that the porting effort is limited only to porting of the kernel. The kernel is typically monolithic to ensure efficiency; the nonkernel part of an OS may be monolithic, or it may be further structured into layers.

Kernel-based operating systems have poor extensibility because addition of a new functionality to the OS may require changes in the functions and services offered by the kernel.

➢ Microkernel-Based Operating Systems

Putting all architecture-dependent code of the OS into the kernel provides good portability. However, in practice, kernels also include some architecture independent code. This feature leads to several problems. It leads to a large kernel size, which detracts from the goal of portability. It may also necessitate kernel modification to incorporate new features, which causes low extensibility. A large kernel supports a large number of system calls. Some of these calls may be used rarely, and so their implementations across different versions of the kernel may not be tested thoroughly. This compromises reliability of the OS.

The microkernel was developed in the early 1990s to overcome the problems concerning portability, extensibility, and reliability of kernels. A microkernel is an essential core of OS code, thus it contains only a subset of the mechanisms typically included in a kernel and supports only a small number of system calls, which are heavily tested and used. This feature enhances portability and reliability of the microkernel.

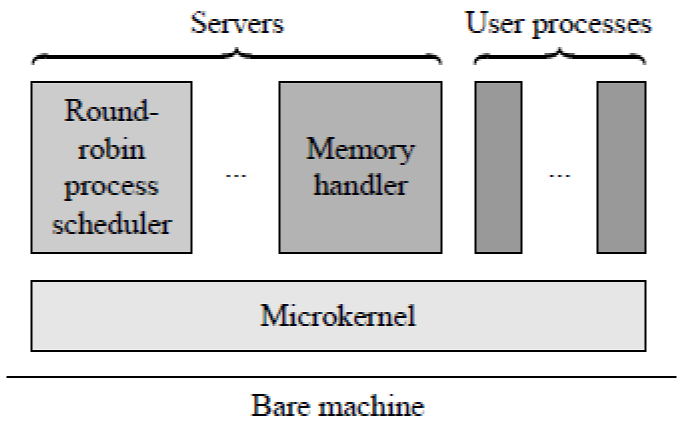


Figure: *Structure of microkernel-based operating systems.*

The above figure illustrates the structure of a microkernel-based OS. The microkernel includes mechanisms for process scheduling and memory management, etc., but does not include a scheduler or memory handler. These functions are implemented as servers, which are simply processes that never terminate. The servers and user processes operate on top of the microkernel, which merely performs interrupt handling and provides communication between the servers and user processes.

The small size and extensibility of microkernels are valuable properties for the embedded systems environment, because operating systems need to be both small and fine-tuned to the requirements of an embedded application.

❖ Virtual Machine Operating Systems

The Virtual Machine Operating System (VM OS) technique helps different user to use different operating system.

The concept of a virtual machine was introduced around 1960. It is the evolution of the time-sharing technique. In the time-sharing method, each program has full access to all the computer resources but at a time, only one program will be executed. The system switch between programs in time slices while saving and restoring program states each time. With the use of the time-sharing method, multiple users can use the computer system concurrently. IBM research centers evolved the time-sharing method as Virtual Machines. CP-67 was the first available virtual machine architecture.

A VM (virtual machine) is an emulation of a computer system, where these machines use computer architectures to provide the functionality of a physical computer. The physical device on which virtual machines work is known as Host, whereas the virtual machines are known as Guest. A single host can have multiple numbers of guests.

The VM OS creates several virtual machines. Each virtual machine is allocated to one user, who can use any OS of his own choice on the virtual machine and run his programs under this OS. This way users of the computer system can use different operating systems at the same time.

➢ Virtual Machine Apps

There are several different virtual machine programs are:

VirtualBox: (Windows, Linux, Mac OS X): VirtualBox is very popular because it's open-source and completely free. There's no paid version of VirtualBox, so you don't have to deal with the usual "upgrade to get more features" upsells and nags. VirtualBox works very well, particularly on Windows and Linux where there's less competition, making it a good place to start with VMs.

VMware Player: (Windows, Linux): VMware has their own line of virtual machine programs. You can use VMware Player on Windows or Linux as a free, basic virtual machine tool. More advanced features—many of which are found in VirtualBox for free—require upgrading to the paid VMware Workstation program. We recommend starting out with VirtualBox, but if it doesn't work properly you may want to try VMware Player.

VMware Fusion: (Mac OS X): Mac users must buy VMware Fusion to use a VMware product, since the free VMware Player isn't available on a Mac. However, VMware Fusion is more polished.

Parallels Desktop: (Mac OS X): Macs also have Parallels Desktop available. Both Parallels Desktop and VMware Fusion for Mac are more polished than the virtual machine programs on other platforms, since they're marketed to average Mac users who might want to run Windows software.