Operating System

MODULE-1: Introduction

Compiled by: Prof. Snehal Andhare

snehal.andhare@vit.edu.in

Vidyalankar Institute of Technology

Vidyalankar Institute of Technology
Wadala (E), Mumbai
www.vit.edu.in

Certificate

This is to certify that the e-book titled "OPERATING SYSTEM" comprises all elementary learning tools for a better understating of the relevant concepts. This e-book is comprehensively compiled as per the predefined eight parameters and guidelines.

Signature

Date: 12-10-2015

Prof. Snehal Andhare

Assistant Professor

Department of Computer Engineering

MODULE -1

INTRODUCTION

➢ Introduction to Operating System:

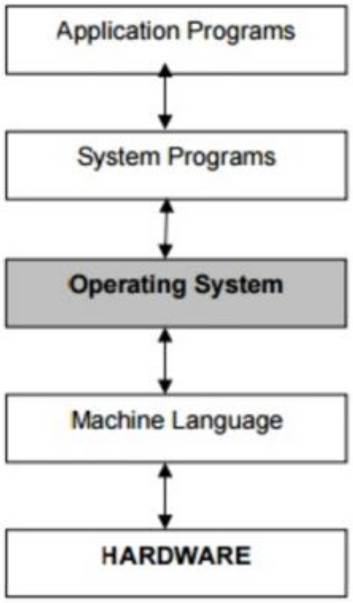
An OS is a program which acts as an interface between computer system users and the computer hardware. It provides a user-friendly environment in which a user may easily develop and execute programs. Otherwise, hardware knowledge would be mandatory for computer programming. So, it can be said that an OS hides the complexity of hardware from uninterested users. In general, a computer system has some resources which may be utilized to solve a problem.

They are :
- Memory
- Processor(s)
- I/O
- File System etc.

The OS manages these resources and allocates them to specific programs and users. With the management of the OS, a programmer is rid of difficult hardware considerations.

An OS provides services for :
- Processor Management
- Memory Management
- File Management
- Device Management
- Concurrency Control

Another aspect for the usage of OS is that; it is used as a predefined library for hardware/software interaction and this is why, system programs apply to the installed OS since they cannot reach hardware directly.



Since we have an already written library, namely the OS, to add two numbers we simply write the following line to our program:

```
c = a + b ;
```

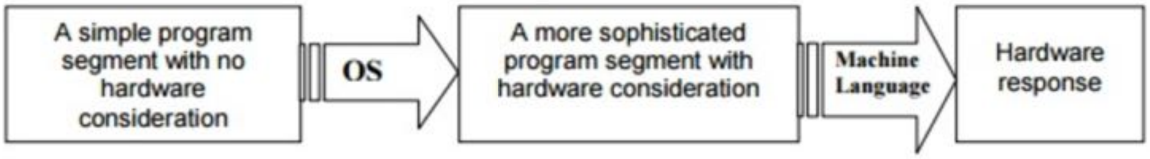
whereas in a system where there is no OS installed, we should consider some hardware work as:

(Assuming an MC 6800 computer hardware)

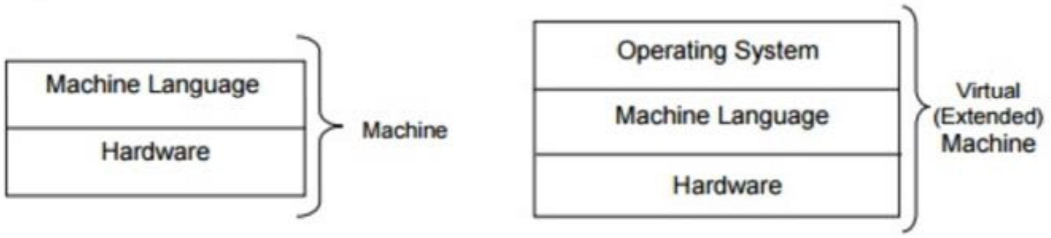
```
LDAA $80 → Loading the number at memory location 80
LDAB $81 → Loading the number at memory location 81
ADDB     → Adding these two numbers
STAA $55 → Storing the sum to memory location 55
```

As seen, we considered memory locations and used our hardware knowledge of the system.

In an OS installed machine, since we have an intermediate layer, our programs obtain *some advantage of mobility* by not dealing with hardware. For example, the above program segment would not work for an 8086 machine, where as the "c = a + b ;" syntax will be suitable for both.


A simple program segment with no hardware consideration
OS
A more sophisticated program segment with hardware consideration
Machine Language
Hardware response

With the advantage of easier programming provided by the OS, the hardware, its machine language and the OS constitutes a new combination called as a **virtual (extended) machine**.



In a more simplistic approach, in fact, OS itself is a program. But it has a priority which application programs don't have. OS uses the **kernel mode** of the microprocessor, whereas other programs use the **user mode**. The difference between two is that; all hardware instructions are valid in kernel mode, where some of them cannot be used in the user mode.

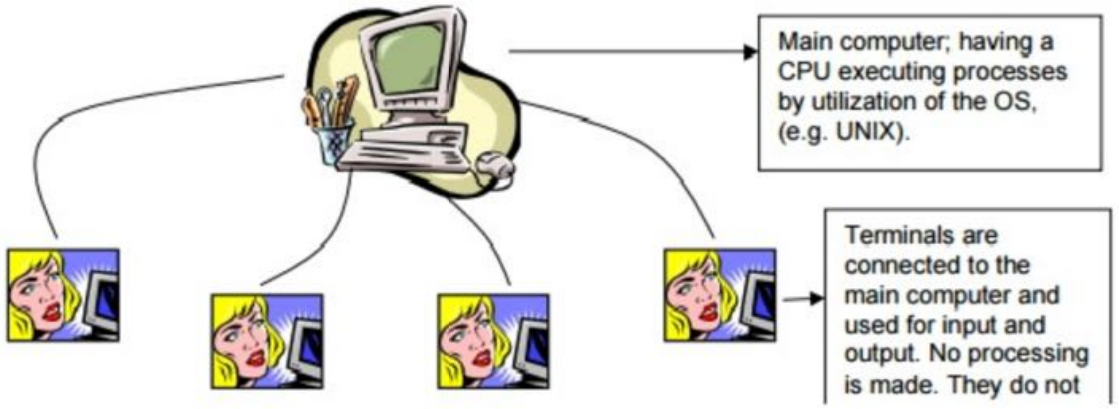
1.2 History of Operating Systems

It all started with computer hardware in about 1945s. Computers were using vacuum tube technology. Programs were loaded into memory manually using switches, punched cards, or paper tapes.

As time went on, card readers, printers, and magnetic tape units were developed as additional hardware elements. Assemblers, loaders and simple utility libraries were developed as software tools. Later, off-line spooling and channel program methods were developed sequentially.

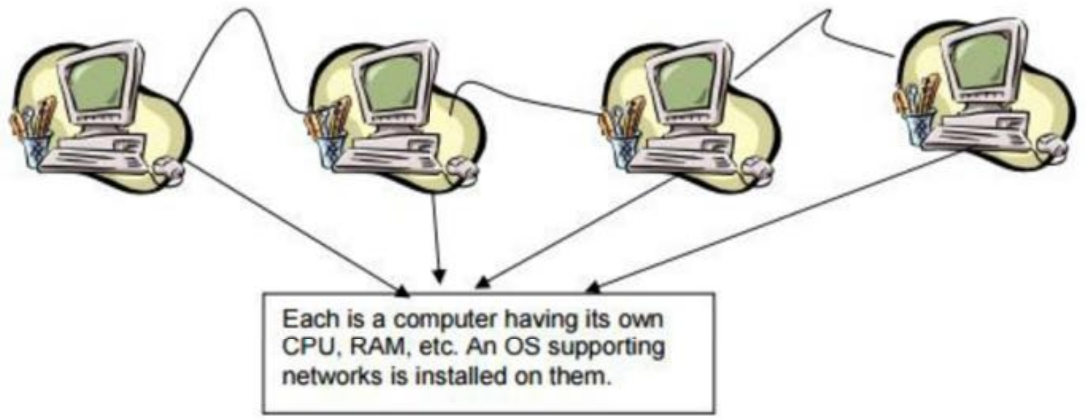
Finally, the idea of **multiprogramming** came. Multiprogramming means sharing of resources between more than one processes. Now the CPU time was not wasted. Because, while one process moves on some I/O work, the OS picks another process to execute till the current one passes to I/O operation.

With the development of interactive computation in 1970s, **time-sharing systems** emerged. In these systems, multiple users have *terminals* (not computers) connected to a *main computer* and execute her task in the main computer.


Main computer; having a CPU executing processes by utilization of the OS, (e.g. UNIX).
Terminals are connected to the main computer and used for input and output. No processing is made. They do not

Another computer system is the **multiprocessor system** having multiple processors sharing memory and peripheral devices. With this configuration, they have greater computing power and higher reliability. Multiprocessor systems are classified into two as tightly-coupled and loosely-coupled (distributed). In the former one, each processor is assigned a specific duty but processors work in close association, possibly sharing the memory. In the latter one, each processor has its own memory and copy of the OS.

Use of the networks required OSs appropriate for them. In **network systems**, each process runs in its own machine. The OS can access to other machines. By this way, file sharing, messaging, etc. became possible.In networks, users are aware of the fact that s/he is working in a network and when information is exchanged. The user explicitly handles the transfer of information.


Each is a computer having its own CPU, RAM, etc. An OS supporting networks is installed on them.

Distributed systems are similar to networks. However in such systems, there is no need to exchange information explicitly, it is handled by the OS itself whenever necessary.

With continuing innovations, new architectures and compatible OSs are developed. But their details are not in the scope of this text since the objective here is to give only a general view about developments in OS concept.

➢ Objectives and Functions of O.S.

✓ Objectives:

- Convenience: An OS makes a computer more convenient to use.
- Efficiency: An OS allows the computer system resources to be used in an efficient manner.
- Ability to evolve: An OS should be constructed in a way to permit the efficient development, testing and introduction of new system functions without interfacing with service.

✓ Functions Of Operating system:

Operating systems perform the following important functions:
i) Processor Management: It means assigning processor to different tasks which has to be performed by the computer system.

ii) Memory Management: It means allocation of main memory and secondary storage areas to the system programmes, as well as user programmes and data.
iii) Input and Output Management: It means co-ordination and assignment of the different output and input devices while one or more programmes are being executed.
iv) File System Management: Operating system is also responsible for maintenance of a file system, in which the users are allowed to create, delete and move files.
v) Establishment and Enforcement of a Priority System: It means the operating system determines and maintains the order in which jobs are to be executed in the computer system.
vi) Assignment of system resources, both software and hardware to the various users of the system.

An operating system has three main responsibilities:

1. Perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk and controlling peripheral devices such as disk drives and printers.
2. Ensure that different programs and users running at the same time do not interfere with each other.
3. Provide a software platform on top of which other programs (i.e., application software) can run.

The first two responsibilities address the need for managing the computer hardware and the application programs that use the hardware. The third responsibility focuses on providing an interface between application software and hardware so that application software can be efficiently developed. Since the operating system is already responsible for managing the hardware, it should provide a programming interface for application developers.

<u>Services provided by OS:</u>

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

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

Following are few common services provided by operating systems.

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

Program execution: Operating system handles 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: I/O subsystem comprised of I/O devices and their corresponding driver software. Drivers hides the peculiarities of specific hardware devices from the user as the device driver knows the peculiarities of the specific device. Operating System manages the communication between user and device drivers.

Following are the major activities of an operating system with respect to I/O Operation.

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

File system manipulation: A file represents a collection of related information. Computer can store files on the disk (secondary storage), for long term storage purpose. Few examples of storage media are 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, operating system manages communications between processes. Multiple processes with one another through communication lines in the network. 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.
- The both processes can be on the one computer or on different computer but are connected through computer network.
- Communication may be implemented by two methods either by Shared Memory or by Message Passing.

Error handling: Error can occur anytime and anywhere. 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.

- OS constantly remains aware of possible errors.

• OS takes the 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.

• OS manages all kind of resources using schedulers.

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

Protection: Considering computer systems having multiple users the concurrent execution of multiple processes, then the various processes must be protected from each another's activities. Protection refers to mechanism or a way to control the access of programs, processes, or users to the resources defined by computer systems.
Following are the major activities of an operating system with respect to protection.

• OS ensures that all access to system resources is controlled.

• OS ensures that external I/O devices are protected from invalid access attempts.

• OS provides authentication feature for each user by means of a password.

<u>Types of Operating System:</u>

Operating systems are there from the very first computer generation. Operating systems keep evolving over the period of time.
Following are few of the important types of operating system which are most commonly used.

➢ **Batch operating system:** The users of batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. Thus, the programmers left their programs with the operator. The operator then sorts programs into batches with similar requirements.

The problems with Batch Systems are following.

- Lack of interaction between the user and job.
- CPU is often idle, because the speeds of the mechanical I/O devices are slower than CPU.
- Difficult to provide the desired priority.

➢ **Time-sharing operating systems:** Time sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing. The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of multiprogrammed batch systems, objective is to maximize processor use, whereas in Time-Sharing Systems objective is to minimize response time. Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, processor execute each user program in a short burst or quantum of computation. That is if n users are present, each user can get time quantum. When the user submits the command, the response time is in few seconds at most. Operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are following:

- Provide advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

Disadvantages of Timesharing operating systems are following.

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

➢ **Distributed operating System**: Distributed systems use multiple central processors to serve multiple real time application and multiple users. Data processing jobs are distributed among the processors

accordingly to which one can perform each job most efficiently. The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as loosely coupled systems or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, and computers and so on.
The advantages of distributed systems are following.

- With resource sharing facility user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

➢ **<u>Network operating System</u>:** Network Operating System runs on a server and and provides server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks. Examples of network operating systems are Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.
The advantages of network operating systems are following.

- Centralized servers are highly stable.
- Security is server managed.
- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are following.

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

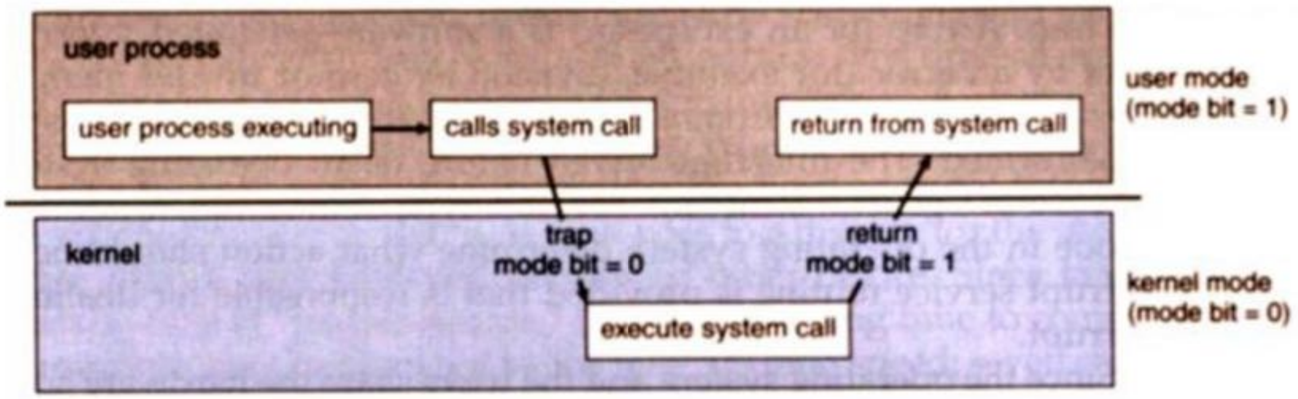
- **Real Time operating System:** Real time system is defines as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. Real time processing is always on line whereas on line system need not be real time. The time taken by the system to respond to an input and display of required updated information is termed as response time. So in this method response time is very less as compared to the online processing. Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. Real-time operating system has well-defined, fixed time constraints otherwise system will fail. For example Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, and home-appliance controllers, Air traffic control system etc.

  There are two types of real-time operating systems.

  Hard real-time systems: Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems secondary storage is limited or missing with data stored in ROM. In these systems virtual memory is almost never found.

  Soft real-time systems: Soft real time systems are less restrictive. Critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard realtime systems. For example, Multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers etc.

System call:



The system call provides an interface to the operating system services.

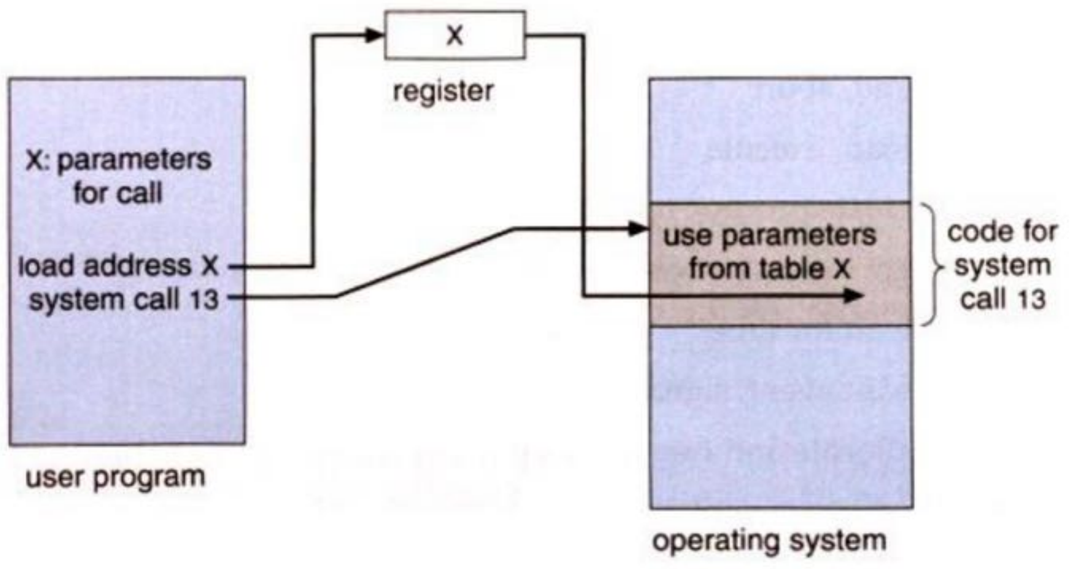
Application developers often do not have direct access to the system calls, but can access them through an application programming interface (API). The functions that are included in the API invoke the actual system calls. By using the API, certain benefits can be gained:

- Portability: as long a system supports an API, any program using that API can compile and run.
- Ease of Use: using the API can be significantly easier then using the actual system call.

System Call Parameters:

Three general methods exist for passing parameters to the OS:

1. Parameters can be passed in registers.
2. When there are more parameters than registers, parameters can be stored in a block and the block address can be passed as a parameter to a register.
3. Parameters can also be pushed on or popped off the stack by the operating system.



Types Of System Call:

There are 5 different categories of system calls:

- process control, file manipulation, device manipulation, information maintenance and communication.

✓ **Process Control**

A running program needs to be able to stop execution either normally or abnormally. When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger.

✓ **File Management**

Some common system calls are *create*, *delete*, *read*, *write*, *reposition*, or *close*. Also, there is a need to determine the file attributes – *get* and *set* file attribute. Many times the OS provides an API to make these system calls.

✓ **Device Management**

Process usually require several resources to execute, if these resources are available, they will be granted and control returned to the user process. These resources are also thought of as devices. Some are physical, such as a video card, and others are abstract, such as a file.

User programs *request* the device, and when finished they *release* the device. Similar to files, we can *read*, *write*, and *reposition* the device.

✓ **Information Management**

Some system calls exist purely for transferring information between the user program and the operating system. An example of this is *time*, or *date*.

The OS also keeps information about all its processes and provides system calls to report this information.

✓ **Communication**

There are two models of interprocess communication, the message-passing model and the shared memory model.

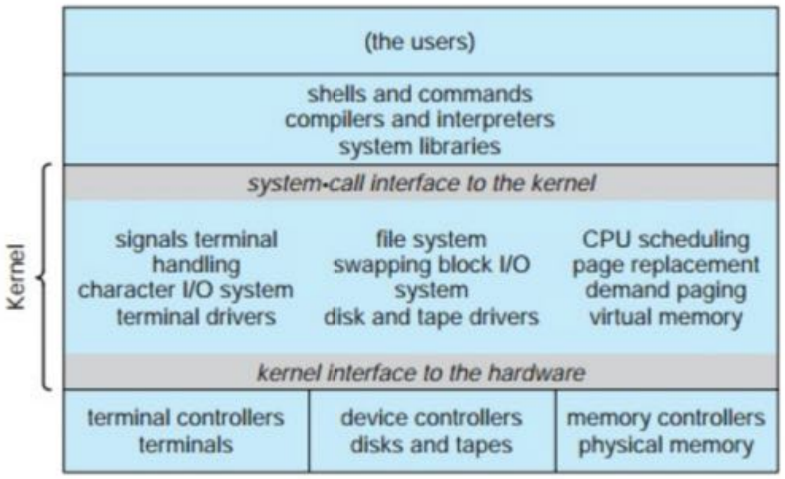
- Message-passing uses a common mailbox to pass messages between processes.
- Shared memory use certain system calls to create and gain access to create and gain access to regions of memory owned by other processes. The two processes exchange information by reading and writing in the shared data.

Operating System Structure:

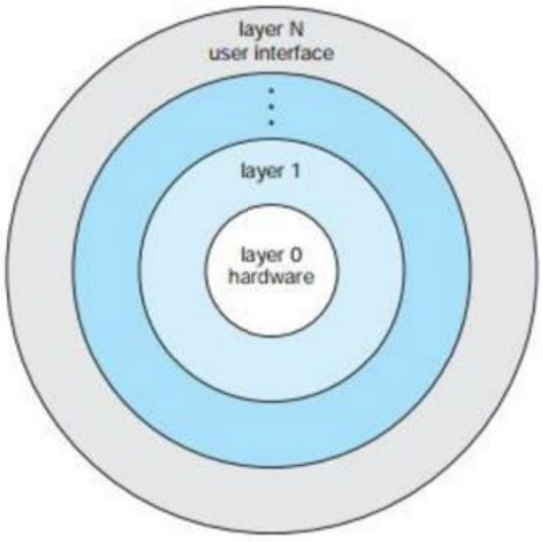
A system as large and complex as a modern operating system must be engineered carefully if it is to function properly and be modified easily. A common approach is to partition the task into small components rather than have one monolithic system. Each of these modules should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions.

(1) **Simple Structure:** Many commercial systems do not have well-defined structures. Frequently, such operating systems started as small, simple, and limited systems and then grew beyond their original scope. MS-DOS is an example of such a system. It was originally designed and implemented by a few people who had no idea that it would become so popular. It was written to provide the most functionality in the least space.

(2) **Layered Approach:** With proper hardware support, operating systems can be broken into pieces that are smaller and more appropriate than those allowed by the original MS-DOS or UNIX systems. The operating system can then retain much greater control over the computer and over the applications that make use of that computer. Implementers have more freedom in changing the inner workings of the system and in creating modular operating systems. Under the topdown approach, the overall functionality and features are determined and are 60 Chapter 2 Operating-System Structures Kernel (the users) shells and commands compilers and interpreters system libraries system-call interface to the kernel signals terminal handling character I/O system terminal drivers file system swapping block I/O system disk and tape drivers CPU scheduling page replacement demand paging virtual memory kernel interface to the hardware terminal controllers terminals device controllers disks and tapes memory controllers physical memory separated into components.



Information hiding is also important, because it leaves programmers free to implement the low-level routines as they see fit, provided that the external interface of the routine stays unchanged and that the routine itself performs the advertised task. A system can be made modular in many ways. One method is the layered approach, in which the operating system is broken up into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface. An operating-system layer is an implementation of an abstract object made up of data and the operations that can manipulate those data. A typical operating-system layer—say, layer M—consists of data structures and a set of routines that can be invoked by higher-level layers. Layer M, in turn, can invoke operations on lower-level layers. The main advantage of the layered approach is simplicity of construction and debugging. The layers are selected so that each uses functions (operations) and services of only lower-level layers. This approach simplifies debugging and system verification.



The first layer can be debugged without any concern for the rest of the system, because, by definition, it uses only the basic hardware (which is assumed correct) to implement its functions. Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged, and so on. If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged. Thus, the design and implementation of the system is simplified. Each layer is implemented with only those operations provided by lower-level layers. A layer does not need to know how these operations are implemented; it needs to know only what these operations do. Hence, each layer hides the existence of certain data structures, operations, and hardware from higher-level layers. The major difficulty with the layered approach involves appropriately defining the various layers. Because a layer can use only lower-level layers, careful planning is necessary. The backing-store driver would normally be above the CPU scheduler, because the driver may need to wait for I/O and the CPU can be rescheduled during this time. However, on a large system, the CPU scheduler may have more information about all the active processes than can fit in memory. Therefore, this information may need to be swapped in and out of memory, requiring the backing-store driver routine to be below the CPU scheduler. A final problem with layered implementations is that they tend to be less efficient than other types. For instance, when a user program executes an

I/O operation, it executes a system call that is trapped to the I/O layer, which calls the memory-management layer, which in turn calls the CPU-scheduling layer, which is then passed to the hardware. At each layer, the parameters may be modified, data may need to be passed, and so on. Each layer adds overhead to the system call; the net result is a system call that takes longer than does one on a non-layered system. These limitations have caused a small backlash against layering in recent years. Fewer layers with more functionality are being designed, providing most of the advantages of modularized code while avoiding the difficult problems of layer definition and interaction.

(3) Microkernels: We have already seen that as UNIX expanded, the kernel became large and difficult to manage. In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called Mach that modularized the kernel using the microkernel approach. This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs. The result is a smaller kernel. There is little consensus regarding which services should remain in the kernel and which should be implemented in user space. Typically, however, microkernels provide minimal process and memory management, in addition to a communication facility. The main function of the microkernel is to provide a communication facility between the client program and the various services that are also running in user space. For example, if the client program wishes to access a file, it must interact with the file server. The client program and service never interact directly. Rather, they communicate indirectly by exchanging messages with the microkernel. One benefit of the microkernel approach is ease of extending the operating system. All new services are added to user space and consequently do not require modification of the kernel. When the kernel does have to be modified, the changes tend to be fewer, because the microkernel is a smaller kernel. The resulting operating system is easier to port from one hardware design to another. The microkernel also provides more security and reliability, since most services are running as user—rather than kernel—processes. If a service fails, the rest of the operating system remains untouched. Several contemporary operating systems have used the microkernel approach. Tru64 UNIX (formerly Digital UNIX) provides a UNIX interface to the user, but it is implemented with a Mach kernel. The Mach kernel maps UNIX system calls into messages to the appropriate user-level services. Another example is QNX. QNX is a real-time operating system that is also based on the microkernel design. The QNX microkernel provides services for message passing and process scheduling. It also handles low-level network communication and hardware interrupts. All other services in QNX are provided by standard processes that run outside the kernel in user mode. Unfortunately, microkernels can suffer from performance decreases due to increased system function overhead. Consider the history of Windows NT. The first release had a layered microkernel organization. However, this version delivered low performance compared with that of Windows 95. Windows NT 4.0 partially redressed the performance problem by moving layers from user space to kernel space and integrating them more closely. By the time Windows XP was designed, its architecture was more monolithic than microkernel.