UNIT 2. OPERATING SYSTEM STRUCTURES

2.1 INTRODUCTION
An OS provides the environment within which the programs are executed. OS varies in their design. Before designing OS, the goals of the OS must be decided. The type of system desired is the basis for choices among various algorithms and strategies.

An OS can be viewed from different angles:
- By examining the services provided by it.
- By looking at the interface that it makes available to users and programmers.
- By disassembling the system into its components and their interconnections.

Here, we will discuss all these aspects of OS, showing the viewpoints of users, programmers, and OS designers. We consider what services an OS provides, how they are provided, and what the various methodologies are for designing such systems.

2.2 SYSTEM COMPONENTS
A large and complex OS can be created by designing small pieces. Each piece should be a well defined portion of the system, with carefully defined inputs, outputs, and functions. Obviously, not all systems have the same structure. However, many modern systems share the goal of supporting the system components as discussed in the following sections.

2.2.1 Process Management
In simple terms, a process can be thought of as a job or time-shared program in execution. A notepad being run, a system task sending output to the printer, a compiler running a program etc. are examples of a process.

A process requires some resources like CPU time, memory, files, I/O devices etc to accomplish its task. These resources are either given to the process when it is created, or allocated to it while it is running. Along with these physical and logical resources, various inputs may be passed when a process is running. For example, consider a process whose function is to display the status of a file on the screen of a terminal. At the beginning, name of the file can be given as an input to the process. It will then execute the appropriate instructions and system calls to obtain and display the information on the terminal. When the process terminates, the OS will claim back the resources.

Program is a passive entity and a process is an active entity. So, a simple program is not a process; whereas a program with a program counter specifying the next instruction to execute is a process. The execution of a process must be sequential and only one instruction is executed at a time. Thus two processes associated with the same program are not treated as two separate execution sequences. A program may generate many processes. A system consists of a collection of processes:
- operating-system processes – those that execute system code
- user processes – those that execute user code

All these processes can potentially execute concurrently, by multiplexing the CPU among them.

The operating system is responsible for the following activities of with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Process management techniques are discussed later.

2.2.2 Main-memory Management

The main memory is central to the operation of a modern computer system. Main memory is a large array of words or bytes. Each word or byte has its own address. Main memory is a repository of quickly accessible data shared by the CPU and I/O devices. The central processor reads instructions from main memory during the instruction-fetch cycle, and it reads/writes data from main memory during the data-fetch cycle. The I/O operations implemented via DMA also read/write data in main memory. The main memory is the only large storage device that the CPU is able to address and access directly. For example, the data stored in the disk must be transferred to main memory for processing that data. Similarly, instructions must be in memory for the CPU to execute them.

For a program to be executed, it must be mapped to absolute addresses and loaded into memory. As the program executes, it accesses program instructions and data from memory by generating these absolute addresses. Eventually, the program terminates, its memory space is declared available, and the next program can be loaded and executed.

To improve the utilization of CPU and the speed of computer's response, several programs must be kept in the memory. Different memory management schemes are available and the effectiveness of them depends on situation. Selection of a memory management scheme for a specific system depends on many factors; especially on the hardware design of the system.

The OS is responsible for the following activities in connection with memory management:

- Keeping track of which parts of memory are currently being used and by whom
- Deciding which processes are to be loaded into memory when memory space becomes available
- Allocating and deallocating memory space as needed

2.2.3 File Management

File management is one of the most visible components of an OS. Computers can store information on different types of physical media like magnetic tape, magnetic disk and optical disk etc. Each of such media has its own characteristics and physical organization. And every medium is controlled by a device, such as a disk drive or tape drive. These devices also have unique characteristics like access speed, capacity, data-transfer rate, and access method (sequential or random).

For convenient use of the computer system, the OS provides a uniform logical view of information storage. The OS uses the concepts of the physical properties of its storage devices to define a logical storage unit, the file. OS then maps files onto physical media, and accesses these files via the storage devices.

A file is a collection of related information. Commonly, files represent programs (may be source code or object code) and data. Data files may be numeric, alphabetic, or alphanumeric. Files may be of free-form like text files or they may be formatted rigidly like the files containing fixed fields. A file consists of a sequence of bits, bytes, lines, or records whose meanings are defined by their creators.

The OS implements the abstract concept of a file by managing mass storage media, such as disks and tapes, and the devices that control them. Also, files are normally organized into directories to ease their use. Finally, when multiple users have access to files, we may want to decide by whom (user) and how (read, write, append) files may be accessed.

The OS is responsible for the following activities in connection with file management:

- Creating and deleting files
- Creating and deleting directories
- Supporting primitives for manipulating files and directories
- Mapping files onto secondary storage
- Backing up files on stable (non-volatile) storage media

2.2.4 I/O System Management

One of the purposes of an OS is to hide the peculiarities of specific hardware devices from the user. For example, in UNIX, the peculiarities of I/O devices are hidden from the bulk of the operating system itself by the I/O subsystem. The I/O subsystem consists of

- A memory-management component that includes buffering, caching, and spooling
- A general device-driver interface
- Drivers for specific hardware devices

Only the device driver knows the peculiarities of the specific device to which it is assigned. How the I/O subsystem interfaces to the other system components, manages devices, transfers data, and detects I/O completion etc are discussed later in detail.

2.2.5 Secondary Storage Management

The main purpose of a computer system is to execute programs. These programs along with the data they access must be in main memory, or primary storage, during execution. As main memory is not sufficient to store all data and programs; and data may get lost when the power goes off, the computer system must provide secondary storage to back up main memory.

Most modern computer systems use disks as the principal on-line storage medium, for both programs and data. Most programs including compilers, assemblers, sort routines, editors, and formatters-are stored on a disk until loaded into memory, and then use the disk as both the source and destination of their processing. Hence, the proper management of disk storage is of central importance to a computer system.

The operating system is responsible for the following activities of disk management:

- Free-space management
- Storage allocation
- Disk scheduling

Because secondary storage is used frequently, it must be used efficiently. The entire speed of operation of a computer may attract the speeds of the disk subsystem and of the algorithms that manipulate that subsystem.

2.2.6 Networking

A distributed system is a collection of processors that do not share memory, peripheral devices, or a clock. Instead, each processor has its own local memory and clock, and the processors communicate with each other through various communication lines, such as high-speed buses or networks. The processors in a distributed system vary in size and function. They may include small microprocessors, workstations, minicomputers, and large, general-purpose computer systems.

The communication network may be fully or partially connected. Its design must focus on message routing, connection strategies, problems of contention and security. A distributed system collects physically separate and heterogeneous systems into a single coherent system, providing the user with access to the various resources that the system maintains.

Access to a shared resource allows computation speedup, increased functionality, increased data availability, and enhanced reliability. OS generalizes network access as a form of file access and the details of networking are contained in the device driver of network interface.

The protocols of a distributed system influence the utility and popularity of that system. The aim of World Wide Web was to create a new access method for information sharing. It improved file-transfer protocol (FTP) and network file-system (NFS) protocol by removing the need for a user to log in for accessing remote resource. It defined a new protocol, hypertext transfer protocol (HTTP), for use in communication between a web server and a web browser. A web browser then needs to send a request for information to a remote machine's web server, and the information is returned. Hence, the HTTP and web usage got increased.

2.2.7 Protection System

If a computer system has multiple users and concurrent execution of multiple processes is allowed, then these processes must be protected from each other's activities. For that purpose, OS must ensure that the files, memory segments, CPU, and other resources are be operated by only authorized processes. For example, memory-addressing hardware ensures that a process can execute only within its own address space. The timer ensures that no process can gain control of the CPU without eventually relinquishing control.

Protection is any mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. This mechanism must provide the rules for how and what type of controls must be enforced. Protection can improve reliability by detecting hidden errors at the interfaces between component subsystems. Early detection of interface errors can prevent contamination of a healthy subsystem by a malfunctioning subsystem. An unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user. A protection-oriented system provides a means to distinguish between authorized and unauthorized usage.

2.2.8 Command Interpreter System

Command interpreter is the interface between the user and the OS. It is one of the most important system programs for an OS. Some OS include the command interpreter in the kernel. Few OS like MS-DOS and UNIX treat the command interpreter as a special program that is running when a job is initiated, or when a user first logs on.

Many commands are given to the OS by control statements. When a new job is started in a batch system, or when a user logs on to a time-shared system, a program that reads and interprets control statements is executed automatically. This program is sometimes called the ***control-card interpreter*** or the ***command-line interpreter***, and is often known as the ***shell***. The job of shell is to get the next command statement and to execute it.

Normally, shell is differentiated from user-friendly command interpreter. For this purpose, mouse-based window and menu systems are used in OS like Microsoft Windows and Macintosh. Here, a mouse pointer and clicks will do the job of navigation and selection.

In a complex shell like MS-DOS and UNIX, commands are typed using a keyboard and displayed on a screen. Then enter key has to be pressed to indicate end of the command and then the task is executed.

The command statements themselves deal with process creation and management, I/O handling, secondary-storage management, main-memory management, file-system access, protection, and networking.

2.3 OPERATING SYSTEM SERVICES

An OS provides an environment for the execution of programs. Also, it provides certain services to the programs and its users. Though these services differ from one OS to the other, following are some general services provided by any OS.

- **Program execution:** The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).

- **I/O operations:** A running program may require I/O. This I/O may involve a file or an I/O device. For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the OS must provide a means to do I/O.
- **File-system manipulation:** The OS must facilitate the programs to read and write the files. And also, programs must be allowed to create and delete files by name.
- **Communications:** Processes may need to exchange information with each other. These processes may be running on same computer or on different computers. Communications may be implemented via shared memory, or by the technique of message passing, in which packets of information are moved between processes by the OS.
- **Error detection:** The OS constantly needs to be aware of possible errors. Errors may occur in any of CPU, memory hardware, I/O devices, user program etc. For each type of error, the OS should take the appropriate action to ensure correct and consistent computing.

OS also has another set of functionalities to help the proper functioning of itself.

- **Resource allocation:** When multiple users are logged on the system or multiple jobs are running at the same time, resources must be allocated to each of them. OS has to manage many resources like CPU cycles, main memory, and file storage etc. OS uses CPU scheduling routines for effective usage of CPU. These routines manage speed of the CPU, the jobs that must be executed, the number of registers available, and such other factors.
- **Accounting:** We want to keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics. Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.
- **Protection:** Information on a multi-user computer system must be secured. When multiple processes are executing at a time, one process should not interfere with the others. Protection involves ensuring that all access to system resources is controlled. System must be protected from outsiders as well. This may be achieved by authenticating the users by means of a password. It also involves defending external I/O devices, modems, network adapters etc. from invalid access.

2.4 SYSTEM CALLS

System calls provide the interface between a process and OS. These calls are normally assembly-language instructions and used by the assembly-language programmers. Some systems allow system calls to be made directly from a higher level language program. In such cases, the calls resemble predefined function or subroutine calls. They may generate a call to a special run-time routine that makes the system call or the system call may be generated directly in-line.

Several languages-such as C, C++, and Perl have replaced assembly language for systems programming. These languages allow system calls to be made directly. For example, UNIX system calls may be invoked directly from a C or C++ program. System calls for modern Microsoft Windows platforms are part of the Win32 application programmer interface (API), which is available for use by all the compilers written for Microsoft Windows.

To understand how system calls are used, consider an example of writing a program to read data from one file and to copy them to another file. The program needs names of two files as an input. These names may be asked from the user. Now, this will initiate a sequence of system calls: to write a prompting message on the screen, and to read the characters from keyboard which defines names of files. Then, the program opens an input file and creates the output file. This operation requires another system call that may possibly face error conditions. Each of the error conditions (like input file doesn't exist, no permission to read, output file can't be created, no write permission etc) require different system calls. When everything is proper, we enter a loop to read from input file and write into the output file. Both of these operations are system calls. Every read/write must return status information for any possible error or end-of-file. Again, this requires system call. Once the task is done, both the files must be closed using system calls. Finally, the program will be terminated using final system call. Thus, we can make out that a very simple program also uses OS heavily. However, the users never see such details. Because, the set of built-in functions provided by the compiler of programming languages provides very simpler interface.

Occurrence of system calls differ from computer to computer. Apart from the identity of the system call, some more information may also be required. The type and nature of information vary according to OS and call. For example, to get input, we may need to specify the file or device to use as the source, and the address and length of the memory buffer into which the input should be read.

Three general methods are used to pass parameters to the operating system:

- The simplest approach is to pass the parameters in registers.
- In some cases, there may be more parameters than registers. Then, the parameters are stored in a block or table in memory. And the address of the block is passed as a parameter in a register as shown in Figure 2.1. This is the approach taken by Linux. Parameters can also be placed, or pushed, onto the stack by the program, and popped of the stack by the operating system.
- Some OS prefer the block or stack methods, because those approaches do not limit the number or length of parameters being passed.

System calls can be grouped roughly into five major categories: process control, file management, device management, information maintenance, and communications. These are discussed in the following sections. Table 2.1 summarizes the types of system calls provided by OS.

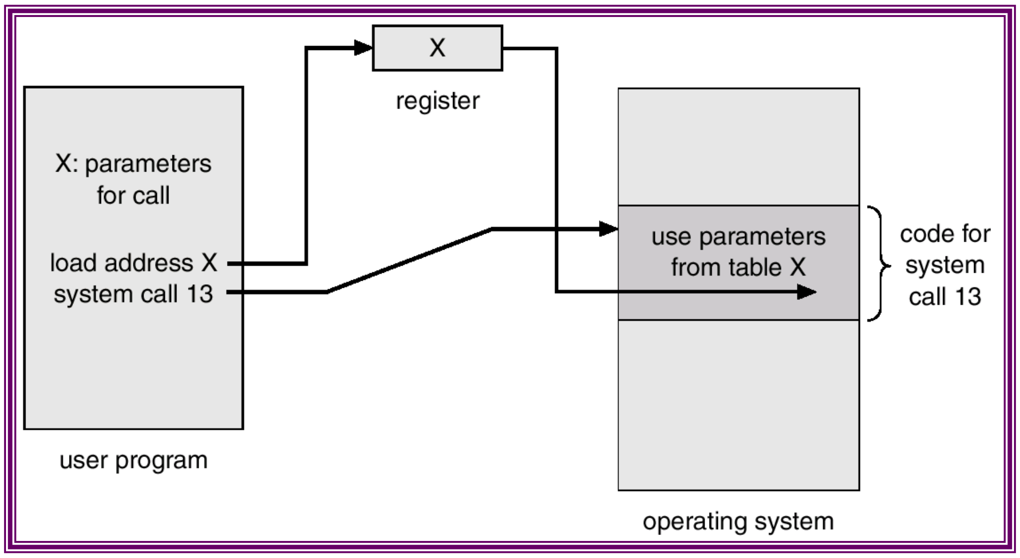


Figure 2.1 Passing of parameters as a table

Table 2.1 Types of System Calls

Category	System Calls
Process Control	• end, abort • load, execute • create process, terminate process • get process attributes, set process attributes • wait for time • wait event, signal event • allocate and free memory
File Management	• create file, delete file • open, close • read, write, reposition • get file attributes, set file attributes
Device Management	• request device, release device • read, write, reposition • get device attributes, set device attributes • logically attach or detach devices
Information Maintenance	• get time or date, set time or date • get system data, set system data • get process, file, or device attributes • set process, file, or device attributes
Communications	• create, delete communication connection • send, receive messages • transfer status information • attach or detach remote devices

2.4.1 Process Control

Various system calls for process controls may be listed as below:

- A running program should halt its execution either normally (*end*) or abnormally (*abort*).

- A process or job executing one program may want to *load* and *execute* another program.
- To create a new job or process, a system call *create process* is used.
- We may also want to terminate a job or process that we created (*terminate process*) if we find that it is incorrect or is no longer needed.
- We should be able to control the execution of a process. This control requires the ability to determine and reset the attributes of a job or process, including the job's priority, its maximum allowable execution time, and so on (*get process attributes* and *set process attributes*).
- We may need to wait for a specific amount time (*wait time)* so that the process is completed or, we may need to wait for a particular event to happen (*wait event*).
- The process may need to signal when the event has occurred (*signal event*).
- Memory has to be allocated to the process (*allocate*) when it needs to be executed and the memory has to be deallocated (*free*) when the process is completed.

2.4.2 File Management
Whenever we are dealing with file handling, we should be able to *create* and *delete* files using respective system calls. The files have to be *opened* for using it. One must be able to *read* and *write* the data into/from files. After finishing the job, the file has to be *closed*.

The same set of system calls is required for directories. On both files and directories, few attributes like file name, file type, protection codes (access rights) etc have to be set or retrieved. For this purpose, we may use the system calls *get file attribute* and *set file attribute.*

2.4.3 Device Management
A running program may need additional resources like more memory, tape drives, access to files etc. to proceed. If the resources are available, they can be granted, and control can be returned to the user program; otherwise, the program will have to wait until sufficient resources are available.

As, files can be thought of as abstract or virtual devices, many system calls used for files are also needed for devices. If the system has multiple users, we must first *request* the device. After we are finished with the device, we must *release* it. These functions are similar to the open and close system calls for files. Once the device has been requested (and allocated to us), we can *read*, *write*, and reposition the device, just as we can with ordinary files.

2.4.4 Information Maintenance
Many system calls exist simply for the purpose of transferring information between the user program and the OS. For example, return the *current time* and *date,* the number of current users, the version number of the OS, the amount of free memory or disk space etc.

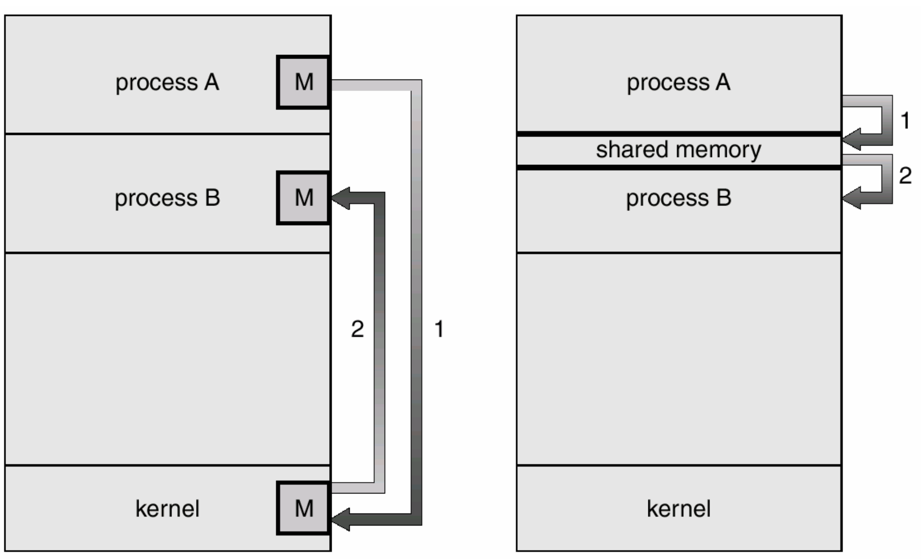
In addition, there are also system calls to reset the process information (get process attributes and set process attributes).

2.4.5 Communication

There are two common models of communication viz.

- Message passing model
- Shared memory model

In the message-passing model, information is exchanged through an inter-process communication facility provided by the OS. Before communication can take place, a connection must be *opened*. The clients and the servers will communicate using *send* and *receive* system calls. In the shared-memory model, processes use *map memory* system calls to gain access to regions of memory owned by other processes. The two communications models are depicted in Figure 2.2.



(a) Message Passing (b) Shared memory

Figure 2.2 Two communication modesl

2.5 SYSTEM PROGRAMS

System programs provide a convenient environment for program development and execution. Some of them are just user interfaces to system calls; others are considerably more complex. They can be divided into following categories:

- **File Management:** These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
- **Status Information:** Information like date, time, amount of available memory or disk space, number of users etc. is formatted, and is printed to the terminal or other output device or file.
- **File Modification:** Several text editors may be available to create and modify the content of files stored on disk or tape.
- **Programming-language Support:** Compilers, assemblers, and interpreters for common programming languages (such as C, C++, Java etc) are often provided to the user with the OS.

- **Program loading and execution:** Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either higher-level languages or machine language are required.
- **Communications**: These programs provide the mechanism for creating virtual connections among processes, users, and different computer systems. They allow users to send messages to one another's screens, to browse web pages, to send electronic-mail messages, to log in remotely, or to transfer files from one machine to another.

Most operating systems will supply the programs for solving common problems, or to perform common operations. For example, web browsers, word processors and text formatters, spreadsheets, database systems, compilers, plotting and statistical-analysis packages, and games. These programs are known as **system utilities** or **application programs.**

2.6 SYSTEM STRUCTURE

The modern OS are very large and complex. To design such an OS, proper care has to be taken. So, it is advised to partition the task into smaller components. Each of such components should be a well-defined unit with properly designed input, output and other functions. We have discussed important components in Section 2.2. Here, we will discuss how these components are interconnected and integrated into kernel.

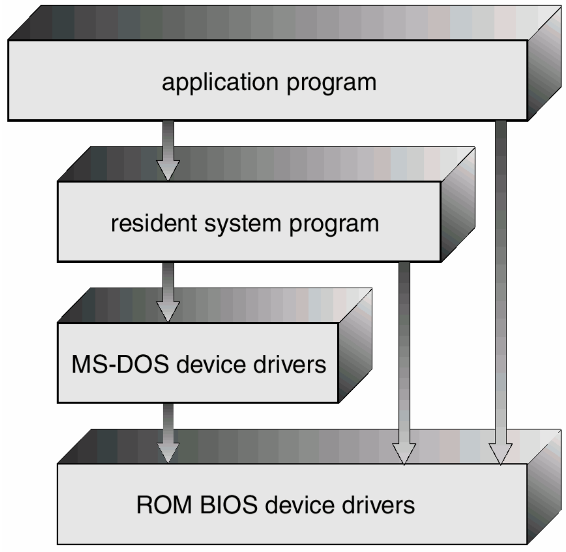
2.6.1 Simple Structure

Many commercial systems do not have a well-defined structure. Frequently, such operating systems started as small, simple, and limited systems, and then grew into wider range. MS-DOS is an example for one such OS. It was written to provide the most functionality in the least space, so it was not divided into modules carefully. Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated as shown in Figure 2.3.

UNIX is another system that was initially limited by hardware functionality. The UNIX OS consists of two separable parts:

- **Systems programs**
- **The kernel**
    - o Consists of everything below the system-call interface and above the physical hardware
    - o Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

The traditional UNIX OS can be layered as shown in Figure 2.4.



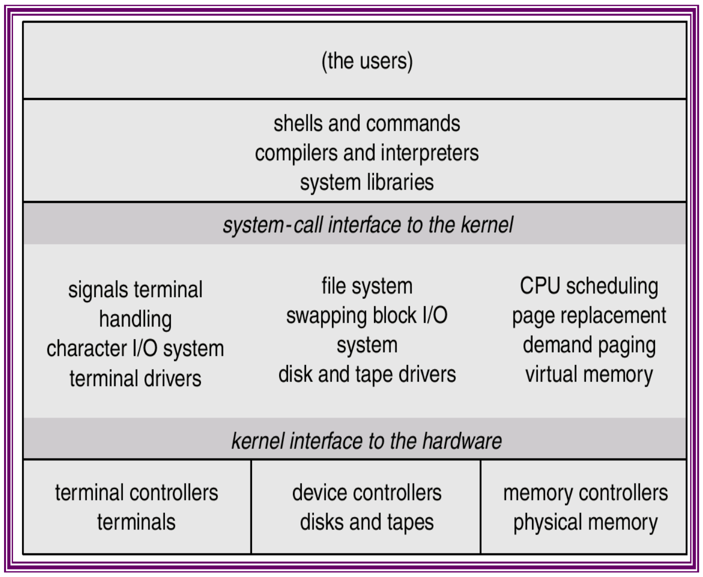


Figure 2.3 MS DOS Layer Structure Figure 2.4 UNIX System Structure

2.6.2 Layered Approach

In the layered approach, the OS is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface. A typical OS with layered approach is shown in Figure 2.5.

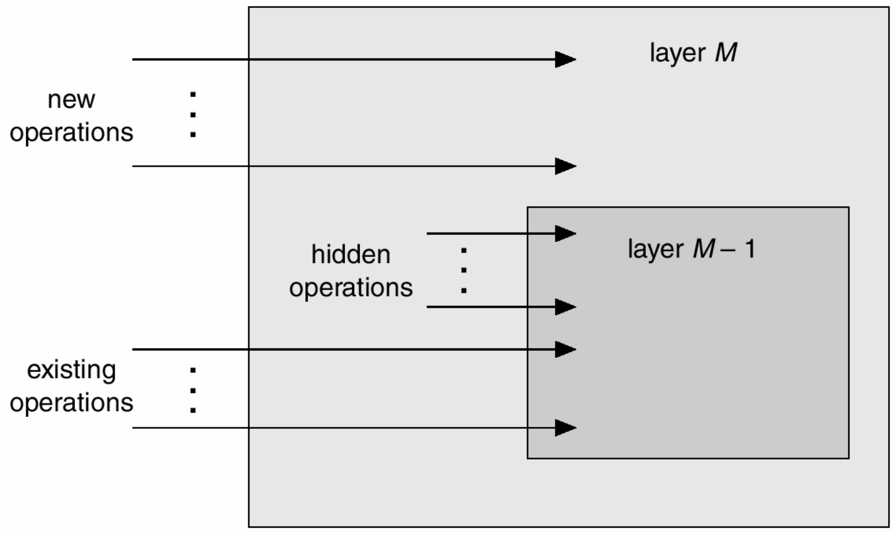


Figure 2.5 A layered approach of OS

The main advantage of layered approach is modularity. The layers are selected such that each uses functions (operations) and services of only lower-level layers. This approach simplifies debugging and system verification.

The major difficulty with the layered approach involves the careful definition of the layers, because a layer can use only those layers below it.

2.6.3 Microkernels

In this approach, all non-essential components of earlier UNIX kernel have been removed and only system and user level programs have been implemented. Hence, it became a smaller kernel. Generally, microkernels provide minimal process and memory management, in addition to a communication facility. Here, the communication takes place between user modules using message passing. The major benefits of using microkernels are:

- easier to extend a microkernel
- easier to port the operating system to new architectures
- more reliable (less code is running in kernel mode)
- more secure

2.7 VIRTUAL MACHINES

A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware. A virtual machine provides an interface *identical* to the underlying bare hardware. The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.

The resources of the physical computer are shared to create the virtual machines. That is,

- CPU scheduling can create the appearance that users have their own processor.
- Spooling and a file system can provide virtual card readers and virtual line printers.
- A normal user time-sharing terminal serves as the virtual machine operator's console.

The difference between non-virtual and virtual machines can be understood by referring Figure 2.6.

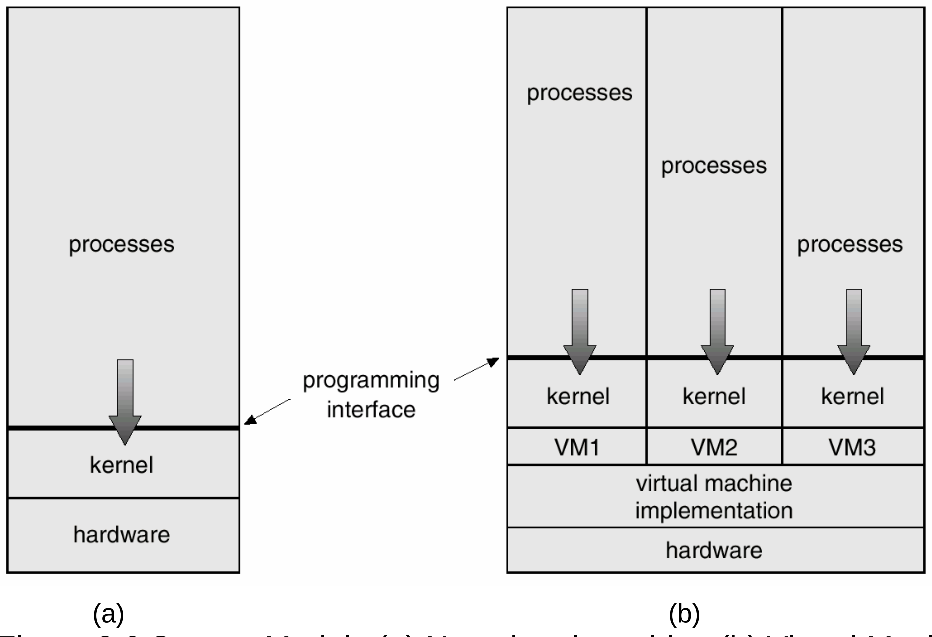


Figure 2.6 System Models (a) Non-virtual machine (b) Virtual Machine

2.7.1 Implementation
Though the virtual-machine (VM) concept is useful, it is difficult to implement due to the effort required to provide an exact duplicate of the underlying machine. Compared to actual I/O, the virtual I/O may take considerably more time, as it is interpreted. Also, since CPU is multiprogrammed among many virtual machines, the VM will slow down unpredictably.

2.7.2 Benefits
The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources. A virtual-machine system is a perfect means for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.

2.7.3 Java
Java is a popular object oriented programming language which provides specifications for Java Virtual machine (JVM). The java compiler produces a platform independent bytecode (.class) file for every class in the program. This bytecode can run on any JVM. The JVM consists of a class loader, a class verifier and a java interpreter. The just-in-time (JIT) compiler converts the bytecode into native code for a particular computer. Thus, the JVM helps to develop platform independent and portable programs. The JVM can be seen in Figure 2.7.

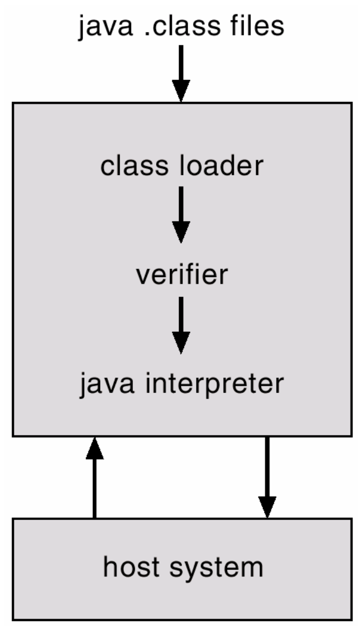


Figure 2.7 Java Virtual Machine

2.8 SYSTEM DESIGN AND IMPLEMENTATION
Here, we will discuss the various problems in designing and implementing a system. No concrete solution exists for the problem, but some approaches are useful.

2.8.1 Design Goals

The very first challenge in a system design is specifying the goals of the system. At the highest level, the design of the system will be affected by the choice of hardware and type of system: batch, time shared, single user, multiuser, distributed, real time, or general purpose. Beyond this highest level, the requirements can be divided into two types:

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

2.8.2 Mechanisms and Policies

The specification and design of an OS is a highly creative task. There are some general software engineering principles which are applicable to OS. One important principle is the separation of policy from mechanism. Mechanisms determine how to do something; policies determine what will be done.

The separation of policy and mechanism is important for flexibility. Policies are likely to change across places or over time. Each change in policy may require a change in the underlying mechanism. Policy decisions must be made for all resource-allocation and scheduling problems.

2.8.3 Implementation

After designing an OS, the next task would be implementation. Earlier, OS would have been written in assembly language. Now, they can be written in higher languages like C, C++ etc. Code written in higher languages has certain advantages:

- Can be written faster
- Is more compact
- Is easier to understand and debug
- Easier to port (move to some other hardware)

Some may say that implementing OS in higher languages may reduce the speed and performance. But, the modern processors have deep pipelining and multiple functional units. So, they can handle many complexities and perform better. Moreover, the performance of OS depends on the selection of data structures and algorithms rather than the assembly-language code.

2.9 SYSTEM GENERATION

We can design, code, and implement an operating system specifically for one machine at one site. The system must then be configured or generated for each specific computer site, a process sometimes known as system generation (SYSGEN).

The SYSGEN program reads from a given file, or asks the operator of the system for information concerning the specific configuration of the hardware system, or probes the hardware directly to determine what components are there. The following kinds of information must be determined.

- What CPU will be used? For multiple-CPU systems, each CPU must be described.
- How much memory is available?
- What devices are available?
- What operating-system options are desired, or what parameter values are to be used?

Using this information the OS can be compiled for a specific requirement. After an OS is generated, it must be made available for use by the hardware. But how does the hardware know where the kernel is, or how to load that kernel? The procedure of starting a computer by loading the kernel is known as ***booting the system***. Most computer systems have a small piece of code, stored in ROM, known as the ***bootstrap program*** or bootstrap loader. This code is able to locate the kernel, load it into main memory, and start its execution.