SHRI VISHNU ENGINEERING COLLEGE FOR WOMEN :: BHIMAVARAM
(Autonomous)

OPERATING SYSTEMS

Unit – I :: Introduction

1.1.What is Operating Systems

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

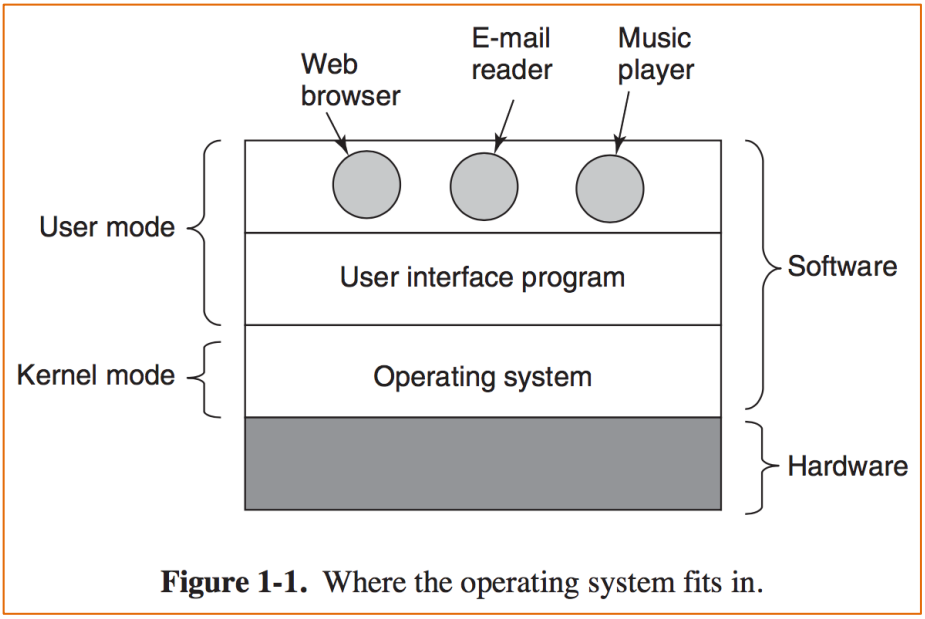


Figure 1-1. Where the operating system fits in.

Every general-purpose computer consists of the hardware, operating system, system programs, and application programs. The hardware consists of memory, CPU, ALU, and I/O devices, peripheral devices, and storage devices. System program consists of compilers, loaders, editors, OS, etc. The application program consists of business programs, database programs.

Every computer must have an operating system to run other programs. The operating system coordinates the use of the hardware among the various system programs and application programs for various users. It simply provides an environment within which other programs can do useful work.

Kernel mode has complete access to all the hardware and can execute any instruction the machine is capable of executing. The rest of the software runs in user mode, in which only a subset of the machine instructions is available.

Operating System providing application programmers a clean abstract set of resources instead of messy hardware ones and managing these hardware resources.

The abstraction is the key to managing all this complexity.

Operating Systems Objectives and Functions

The objectives of the operating system are:

- ✓ To make the computer system convenient to use in an efficient manner.
- ✓ To hide the details of the hardware resources from the users.
- ✓ To provide users a convenient interface to use the computer system.
- ✓ To act as an intermediary between the hardware and its users, making it easier for the users to access and use other resources.
- ✓ To manage the resources of a computer system.
- ✓ To keep track of who is using which resource, granting resource requests, and mediating conflicting requests from different programs and users.
- ✓ To provide efficient and fair sharing of resources among users and programs.

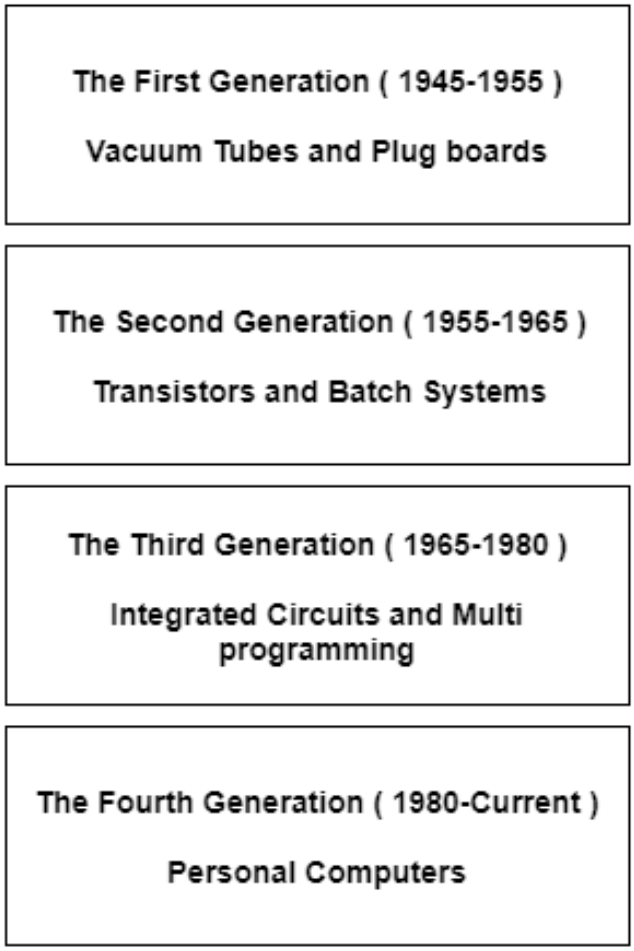
The Functions of the operating system are:

- ✓ **Resource Management:** When parallel accessing happens in the OS means when multiple users are accessing the system the OS works as Resource Manager, Its responsibility is to provide hardware to the user. It decreases the load in the system.
- ✓ **Process Management:** It includes various tasks like scheduling, termination of the process. OS manages various tasks at a time. Here CPU Scheduling happens means all the tasks would be done by the many algorithms that use for scheduling.
- ✓ **Storage Management:** The file system mechanism used for the management of the storage. NIFS, CFS, CIFS, NFS, etc. are some file systems. All the data stores in various tracks of Hard disks that all managed by the storage manager. It included Hard Disk.
- ✓ **Memory Management:** Refers to the management of primary memory. The operating system has to keep track, how much memory has been used and by whom. It has to decide which process needs memory space and how much. OS also has to allocate and deallocate the memory space.
- ✓ **Security / Privacy Management:** Privacy is also provided by the Operating system by means of passwords so that unauthorized applications can't access programs or data. For example, Windows uses Kerberos authentication to prevent unauthorized access to data.

Generations of Operating systems

Operating Systems have evolved over the years. So, their evolution through the years can be mapped using generations of operating systems. There are four generations of operating systems.

These can be described as follows:



OPERATING SYSTEM GENERATIONS

The First Generation (1945 - 1955): Vacuum Tubes and Plugboards

Digital computers were not constructed until the second world war. Calculating engines with mechanical relays were built at that time. However, the mechanical relays were very slow and were later replaced with vacuum tubes. These machines were enormous but were still very slow.

Programming languages were unknown and ***there were no operating systems*** so all the programming was done in machine language. All the problems were simple numerical calculations.

By the 1950's punch cards were introduced and this improved the computer system. Instead of using plugboards, programs were written on cards and read into the system.

The Second Generation (1955 - 1965): Transistors and Batch Systems

Transistors led to the development of the computer systems that could be manufactured and sold to paying customers. These machines were known as ***mainframes*** and were locked in air-conditioned computer rooms with staff to operate them.

The ***Batch System*** was introduced to reduce the wasted time in the computer. A tray full of jobs was collected in the input room and read into the magnetic tape. After that, the tape was rewound and mounted on a tape drive. Then the batch operating system was loaded in which read the first job from the tape and ran it. The output was written on the second tape. After the whole batch was done, the input and output tapes were removed and the output tape was printed.

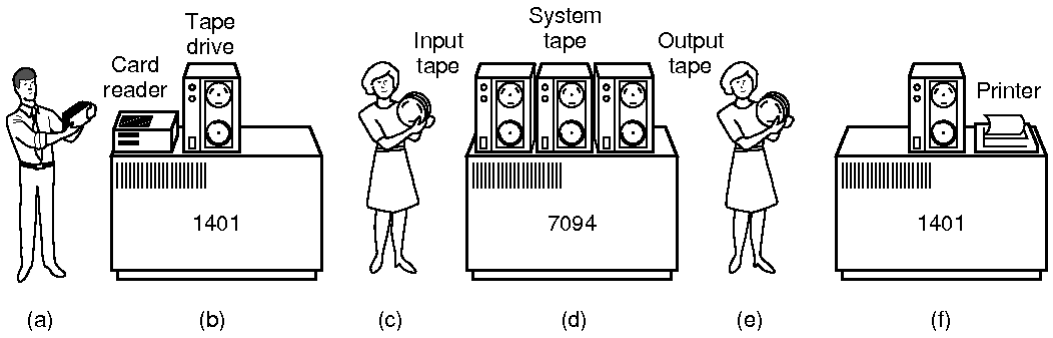


Figure. An early batch system. (a) Programmers bring cards to 1401. (b) 1401 reads batch of jobs onto tape. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output

The Third Generation (1965 - 1980): Integrated Circuits and Multiprogramming

Until the 1960's, there were two types of computer systems i.e the ***scientific and the commercial computers***. These were combined by IBM in the System/360. This used integrated circuits and provided a major price and performance advantage over the second generation systems.

The third generation operating systems also introduced ***multiprogramming***. This meant that the processor was not idle while a job was completing its I/O operation. Another job was scheduled on the processor so that its time would not be wasted.

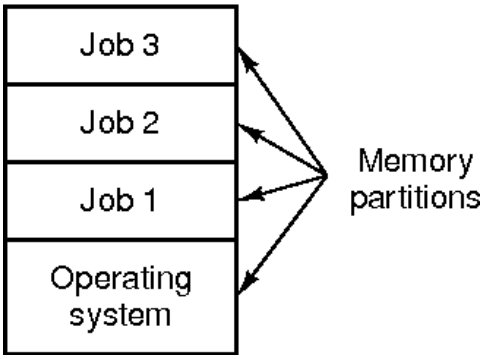


Figure. A multiprogramming system with three jobs in memory.

The Fourth Generation (1980 - Present): Personal Computers

Personal Computers were easy to create with the development of large-scale integrated circuits. These were chips containing thousands of transistors on a square centimeter of silicon. Because of these, microcomputers were much cheaper than minicomputers and that made it possible for a single individual to own one of them.

The advent of personal computers also led to the growth of networks. This created ***network operating systems and distributed operating systems.*** The users were aware of a network while using a network operating system and could log in to remote machines and copy files from one machine to another.

1.2.Types of Operating Systems

An operating system is a well-organized collection of programs that manages the computer hardware. It is a type of system software that is responsible for the smooth functioning of the computer system.

a. Mainframe System
b. Multi-processer System
c. Distributed System
d. Clustered Systems
e. Real Time System
f. Handheld Systems
g. Server Operating Systems
h. Personal Computer Systems
i. Embedded Systems
j. Sensor Node Systems

a. Mainframe System

At the high end are the operating systems for the mainframes, those room sized computers still found in major corporate data centers. These computers differ from personal computers in terms of their I/O capacity.

A mainframe with 1000 disks and millions of gigabytes of data is not unusual. Mainframes are also making something of a comeback as high-end Web servers, servers for large-scale electronic commerce sites, and servers for business-to-business transactions.

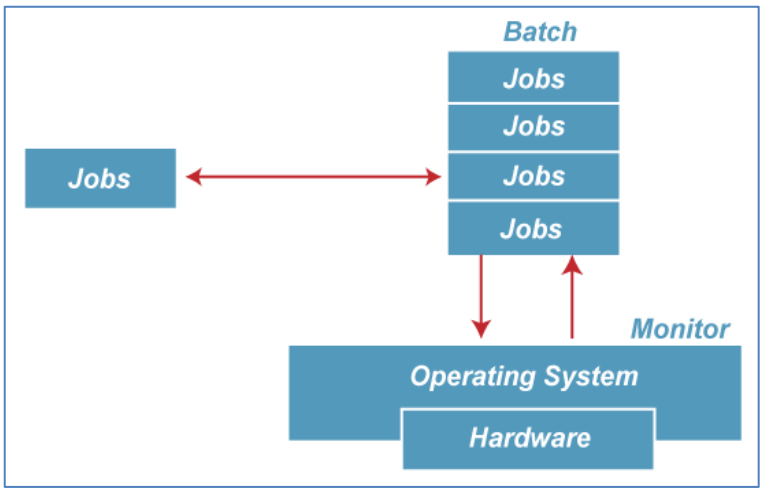
The operating systems for mainframes are heavily oriented toward processing many jobs at once, most of which need prodigious amounts of I/O.

They typically offer three kinds of services: ***batch, transaction processing, and timesharing.***

A batch system is one that processes routine jobs without any interactive user present. Claims processing in an insurance company or sales reporting for a chain of stores is typically done in batch mode.

In the 1970s, Batch processing was very popular. In this technique, similar types of jobs were batched together and executed in time. People were used to having a single computer which was called a mainframe.

The system put all of the jobs in a queue on the basis of first come first serve and then executes the jobs one by one. The users collect their respective output when all the jobs get executed.



The purpose of this operating system was mainly to transfer control from one job to another as soon as the job was completed. It contained a small set of programs called the resident monitor that always resided in one part of the main memory. The remaining part is used for servicing jobs.

Transaction processing systems handle large numbers of small requests, for example, check processing at a bank or airline reservations. Each unit of work is small, but the system must handle hundreds or thousands per second.

Timesharing systems allow multiple remote users to run jobs on the computer at once, such as querying a big database. These functions are closely related; mainframe operating systems often perform all of them.

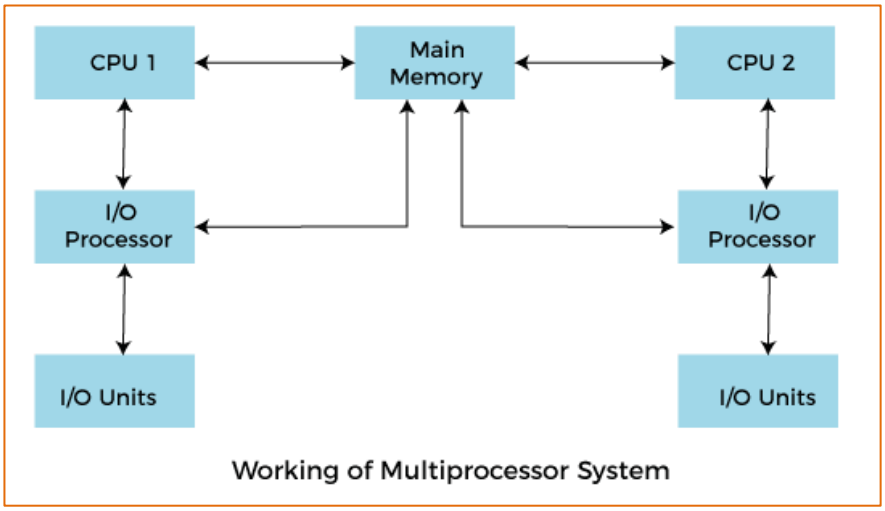
An example mainframe operating system is OS/390, a descendant of OS/360. However, mainframe operating systems are gradually being replaced by UNIX variants such as Linux.

b. Multi-processer System

An increasingly common way to get major-league computing power is to connect multiple CPUs into a single system. Depending on precisely how they are connected and what is shared, these systems are called parallel computers, multicomputers, or multiprocessors. They need special operating systems, but often these are variations on the server operating systems, with special features for communication, connectivity, and consistency.

With the recent advent of multicore chips for personal computers, even conventional desktop and notebook operating systems are starting to deal with at least small-scale multiprocessors and the number of cores is likely to grow over time.

Many popular operating systems, including Windows and Linux, run on multiprocessors.



Working of Multiprocessor System

In Multiprocessing, Parallel computing is achieved. More than one processor present in the system can execute more than one process simultaneously, which will increase the throughput of the system.

Advantages of Multiprocessing operating system:

- ✓ ***Increased reliability:*** Due to the multiprocessing system, processing tasks can be distributed among several processors. This increases reliability as if one processor fails, the task can be given to another processor for completion.

- ✓ ***Increased throughout:*** As several processors increase, more work can be done in less.

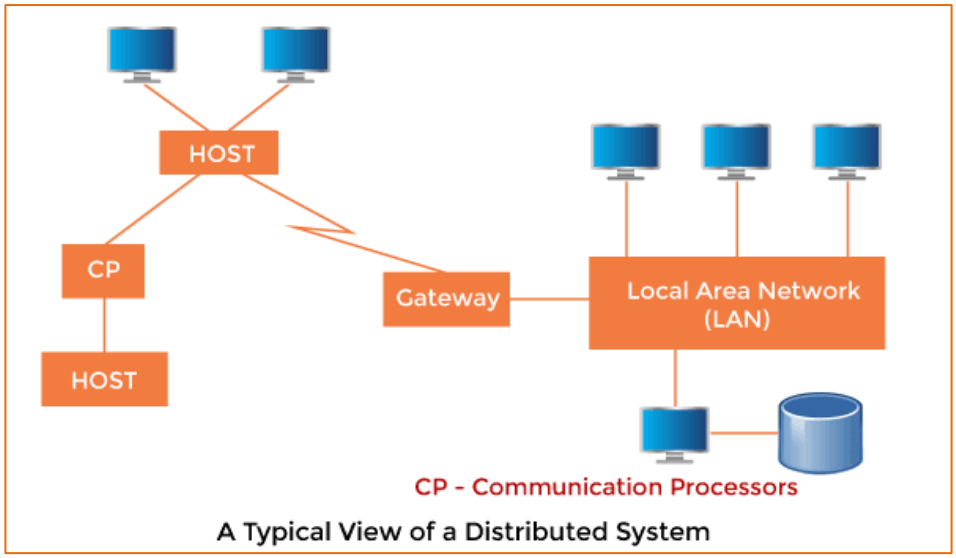
Disadvantages of Multiprocessing operating System:

Multiprocessing operating system is more complex and sophisticated as it takes care of multiple CPUs simultaneously.

c. Distributed Operating System

The Distributed Operating system is not installed on a single machine, it is divided into parts, and these parts are loaded on different machines. A part of the distributed Operating system is installed on each machine to make their communication possible. Distributed Operating systems are much more complex, large, and sophisticated than Network operating systems because they also have to take care of varying networking protocols.

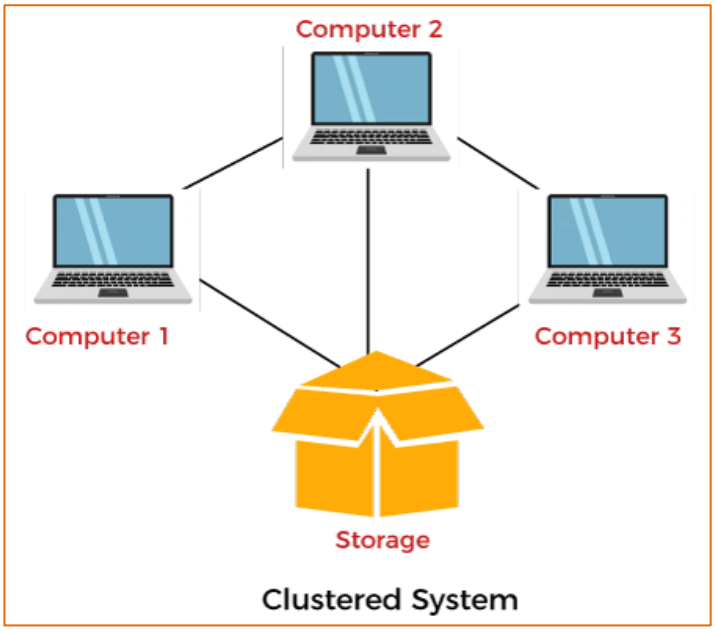
Example: **Telephone and cellular networks**



A Typical View of a Distributed System

d. Clustered Systems

Cluster systems are similar to parallel systems because both systems use multiple CPUs. The primary difference is that clustered systems are made up of two or more independent systems linked together. They have independent computer systems and a shared storage media, and all systems work together to complete all tasks. All cluster nodes use two different approaches to interact with one another, like message passing interface (MPI) and parallel virtual machine (PVM).



Clustered System

Cluster operating systems are a combination of software and hardware clusters. Hardware clusters aid in the sharing of high-performance disks among all computer systems, while software clusters give a better environment for all systems to operate.

A cluster system consists of various nodes, each of which contains its cluster software. The cluster software is installed on each node in the clustered system, and it

monitors the cluster system and ensures that it is operating properly. If one of the clustered system's nodes fails, the other nodes take over its storage and resources and try to restart.

Cluster components are generally linked via fast area networks, and each node executing its instance of an operating system. In most cases, all nodes share the same hardware and operating system, while different hardware or different operating systems could be used in other cases. The primary purpose of using a cluster system is to assist with weather forecasting, scientific computing, and supercomputing systems.

e. Real Time Operating System

In Real-Time Systems, each job carries a certain deadline within which the job is supposed to be completed, otherwise, the huge loss will be there, or even if the result is produced, it will be completely useless.

These systems are characterized by having time as a key parameter.

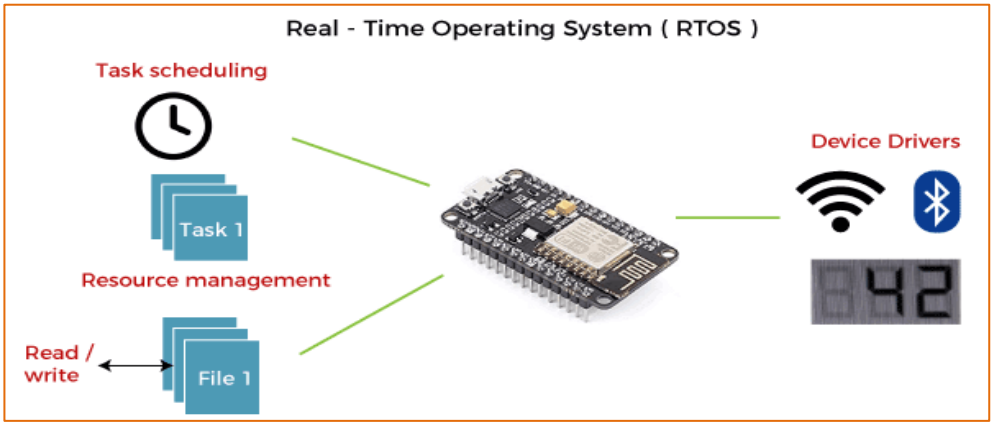
For example, in industrial process control systems, real-time computers have to collect data about the production process and use it to control machines in the factory. Often there are hard deadlines that must be met.

For example, if a car is moving down an assembly line, certain actions must take place at certain instants of time. If a welding robot welds too early or too late, the car will be ruined. If the action absolutely must occur at a certain moment (or within a certain range), we have a hard real-time system.

Many of these are found in industrial process control, avionics, military, and similar application areas. These systems must provide absolute guarantees that a certain action will occur by a certain time.

Another kind of real-time system is a soft real-time system, in which missing an occasional deadline, while not desirable, is acceptable and does not cause any permanent damage. Digital audio or multimedia systems fall in this category.

Digital telephones are also soft real-time systems. Since meeting strict deadlines is crucial in real-time systems, sometimes the operating system is simply a library linked in with the application programs, with everything tightly coupled and no protection between parts of the system.



The Application of a Real-Time system exists in the case of military applications, if you want to drop a missile, then the missile is supposed to be dropped with a certain precision.

f. Handheld Computer Operating Systems

Continuing on down to smaller and smaller systems, we come to handheld computers. A handheld computer or PDA (Personal Digital Assistant) is a small computer that fits in a shirt pocket and performs a small number of functions, such as an electronic address book and memo pad. Furthermore, many mobile phones are hardly any different from PDAs except for the keyboard and screen.

In effect, PDAs and mobile phones have essentially merged, differing mostly in size, weight, and user interface. Almost all of them are based on 32-bit CPUs with protected mode and run a sophisticated operating system.

The operating systems that run on these handhelds are increasingly sophisticated, with the ability to handle telephony, digital photography, and other functions.

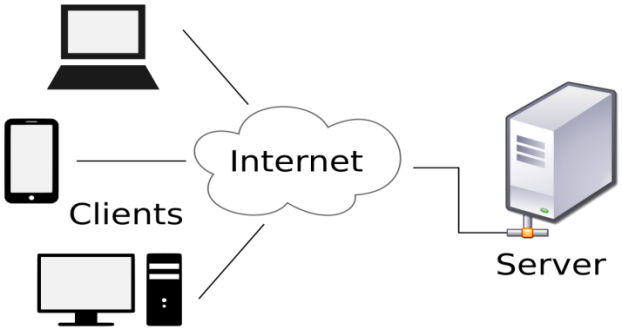
Many of them also run third-party applications. In fact, some of them are beginning to resemble the personal computer operating systems of a decade ago.

One major difference between handhelds and PCs is that the former do not have multigigabyte hard disks, which changes a lot. Two of the most popular operating systems for handhelds are Symbian OS and Palm OS.

g. Server Operating Systems

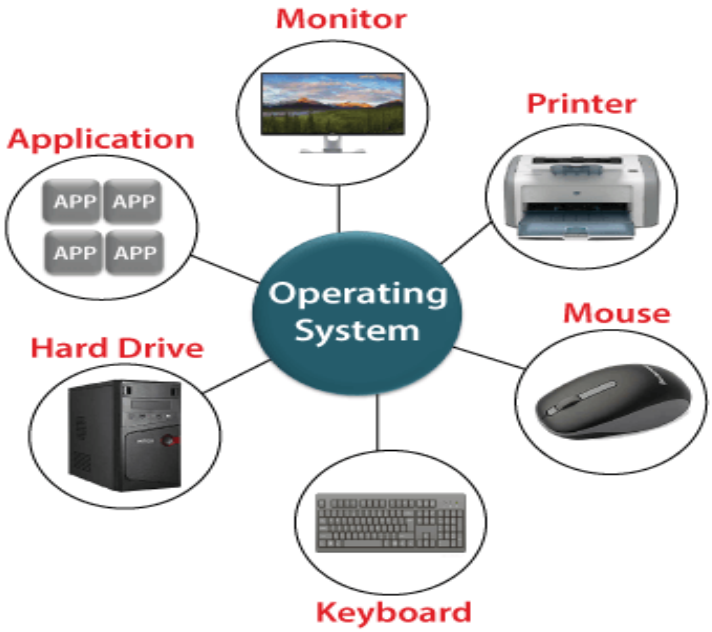
They run on servers, which are either very large personal computers, workstations, or even mainframes. They serve multiple users at once over a network and allow the users to share hardware and software resources. Servers can provide print service, file service, or Web service.

Internet providers run many server machines to support their customers and Websites use servers to store the Web pages and handle the incoming requests. Typical server operating systems are Solaris, FreeBSD, Linux and Windows Server 200x.



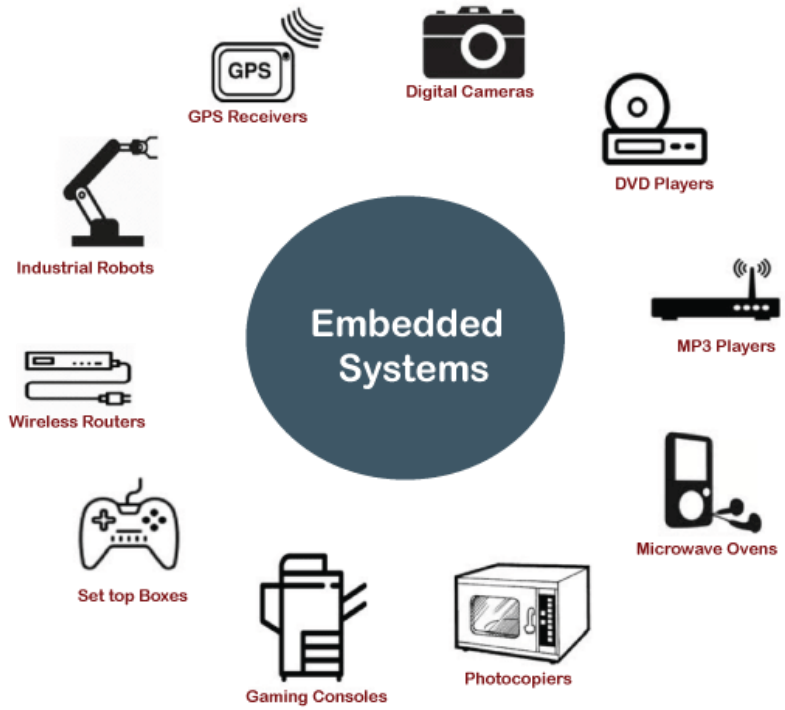
h. Personal Computer Operating Systems

The next category is the personal computer operating system. Modem ones all support multiprogramming, often with dozens of programs started up at boot time. Their job is to provide good support to a single user. They are widely used for word processing, spreadsheets, and Internet access. Common examples are Linux, FreeBSD, Windows 10,11, and the Macintosh operating system. Personal computer operating systems are so widely known that probably little introduction is needed.



i. Embedded Operating Systems

Embedded systems run on the computers that control devices that are not generally thought of as computers and which do not accept user-installed software. Typical examples are microwave ovens, TV sets, cars, DVD recorders, cell phones, MP3 players. The main property which distinguishes embedded systems from handhelds is the certainty that no untrusted software will ever run on it. You cannot download new applications to your microwave oven—all the software is in ROM. This means that there is no need for protection between applications, leading to some simplification. Systems such as QNX and VxWorks are popular in this domain.



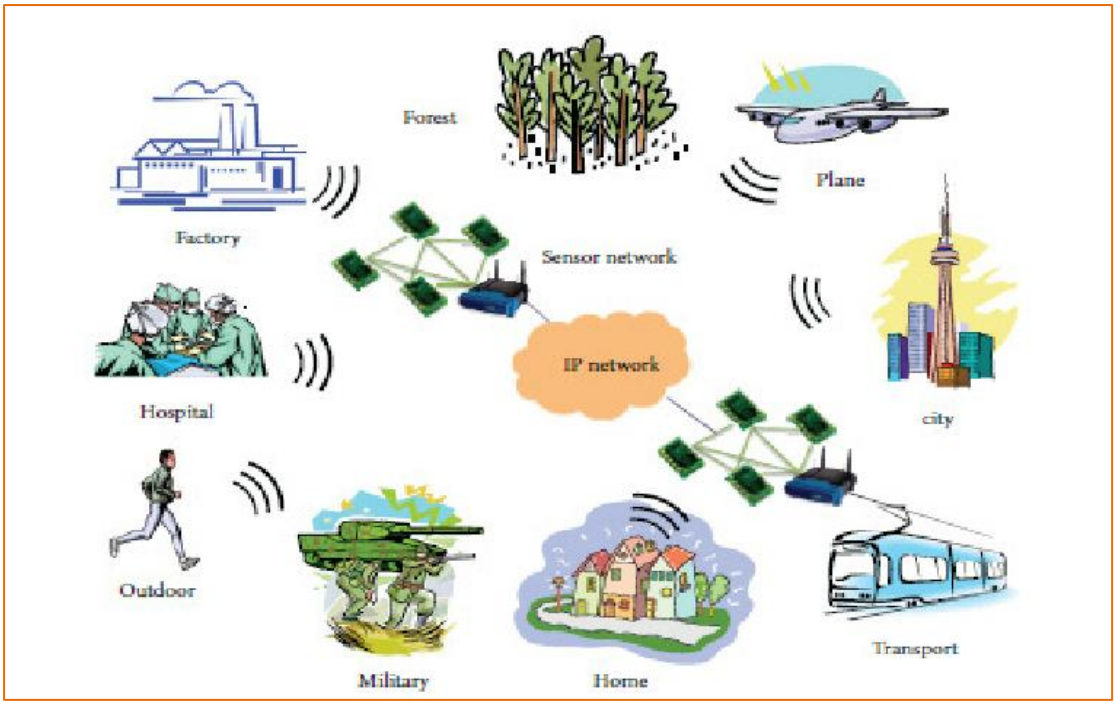
j. Sensor Node Operating Systems

Networks of tiny sensor nodes are being deployed for numerous purposes. These nodes are tiny computers that communicate with each other and with a base station using wireless communication. These sensor networks are used to protect the perimeters of buildings, guard national borders, detect fires in forests, measure temperature and precipitation for weather forecasting, glean information about enemy movements on battlefields, and much more.

The sensors are small battery-powered computers with built-in radios. They have limited power and must work for long periods of time unattended outdoors, frequently in environmentally harsh conditions. The network must be robust enough to tolerate failures of individual nodes, which happen with ever increasing frequency as the batteries begin to run down.

Each sensor node is a real computer, with a CPU, RAM, ROM, and one or more environmental sensors. It runs a small,, but real operating system, usually one that is event driven, responding to external events or making measurements periodically based on an internal clock. The operating system has to be small and simple because the nodes have little RAM and battery lifetime is a major issue.

Also, as with embedded systems, all the programs are loaded in advance; users do not suddenly start programs they downloaded from the Internet, which makes the design much simpler. TinyOS is a well-known operating system for a sensor node.



1.3. Computer System Operations:

A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory (Figure). Each device controller is in charge of a specific type of device (for example, disk drives, audio devices, and video displays).

The CPU and the device controllers can execute concurrently, competing for memory cycles. To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory.

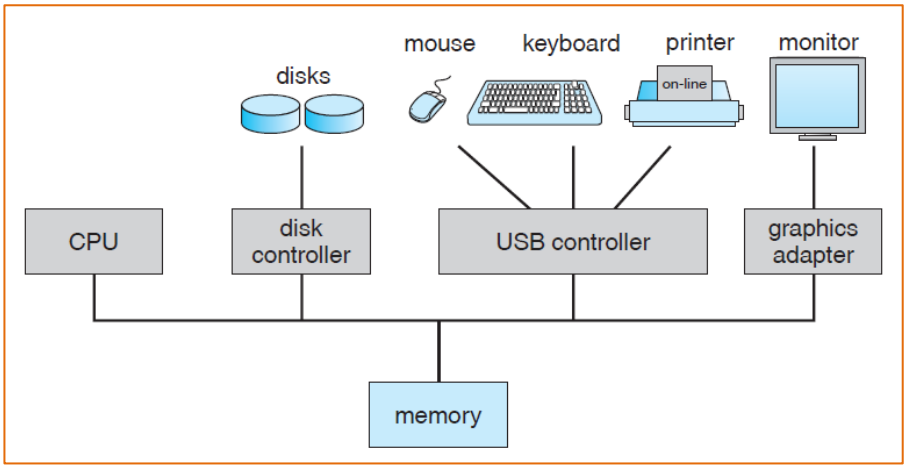


Figure : A Modern Computer System

For a computer to start running—for instance, when it is powered up or rebooted—it needs to have an initial program to run. This initial program, or bootstrap program, tends to be simple. Typically, it is stored in read-only memory (ROM) or electrically erasable programmable read-only memory (EEPROM), known by the general term

firmware, within the computer hardware. It initializes all aspects of the system, from CPU registers to device controllers to memory contents. The bootstrap program must know how to load the operating system and how to start executing that system. To accomplish this goal, the bootstrap program must locate and load into memory the operating system kernel. The operating system then starts executing the first process, such as "init," and waits for some event to occur.

The occurrence of an event is usually signaled by an interrupt from either the hardware or the software. Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus. Software may trigger an interrupt by executing a special operation called a system call (also called a monitor call).

When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location. The fixed location usually contains the starting address where the service routine for the interrupt is located.

The interrupt service routine executes; on completion, the CPU resumes the interrupted computation. A time line of this operation is shown in Figure.

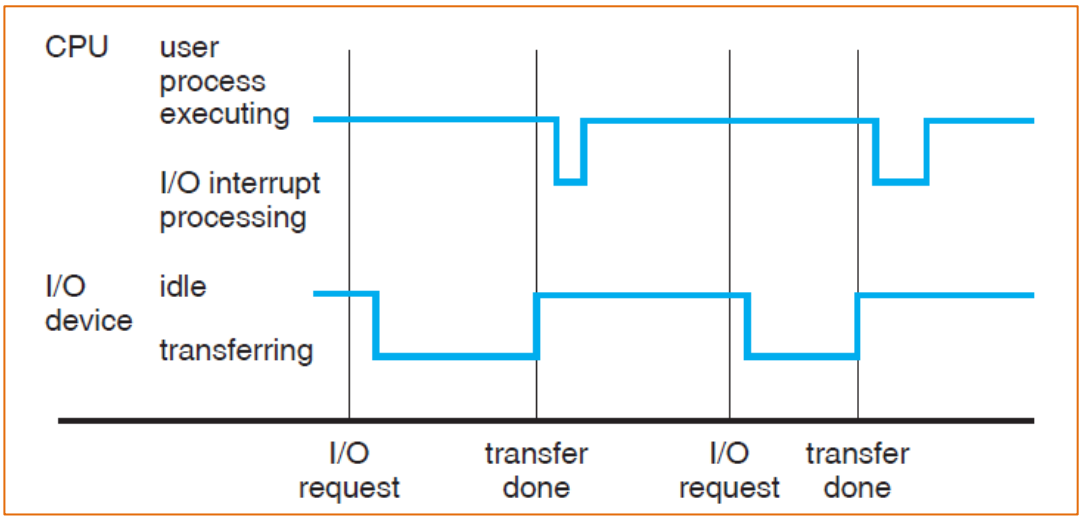


Figure : Interrupt time line for a single process doing output

Interrupts are an important part of a computer architecture. Each computer design has its own interrupt mechanism, but several functions are common.

However, interrupts must be handled quickly. Since only a predefined number of interrupts is possible, a table of pointers to interrupt routines can be used instead to provide the necessary speed. The interrupt routine is called indirectly through the table, with no intermediate routine needed.

The interrupt architecture must also save the address of the interrupted instruction. More recent architectures store the return address on the system stack. If the interrupt routine needs to modify the processor state—for instance, by modifying register values—it must explicitly save the current state and then restore that state before returning. After the interrupt is serviced, the saved return address is loaded into the

program counter, and the interrupted computation resumes as though the interrupt had not occurred.

1.4. Storage Structure

The CPU can load instructions only from memory, so any programs to run must be stored there. General-purpose computers run most of their programs from rewriteable memory, called main memory (also called random-access memory or RAM). Main memory commonly is implemented in a semiconductor technology called dynamic random-access memory (DRAM). Computers use other forms of memory as well. EEPROM cannot be changed frequently and so contains mostly static programs. For example, smartphones have EEPROM to store their factory-installed programs.

All forms of memory provide an array of words. Each word has its own address. Interaction is achieved through a sequence of load or store instructions to specific memory addresses.

A typical instruction–execution cycle, as executed on a system with a von Neumann architecture, first fetches an instruction from memory and stores that instruction in the instruction register. The instruction is then decoded and may cause operands to be fetched from memory and stored in some internal register. After the instruction on the operands has been executed, the result may be stored back in memory. Accordingly, we can ignore how a memory address is generated by a program. We are interested only in the sequence of memory addresses generated by the running program. Ideally, we want the programs and data to reside in main memory permanently. This arrangement usually is not possible for the following two reasons:

- ✓ Main memory is usually too small to store all needed programs and data permanently.
- ✓ Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost.

Thus, most computer systems provide secondary storage as an extension of main memory. The main requirement for secondary storage is that it be able to hold large quantities of data permanently.

The most common secondary-storage device is a magnetic disk, which provides storage for both programs and data. Most programs (system and application) are stored on a disk until they are loaded into memory. Many programs then use the disk as both the source and the destination of their processing.

In a larger sense, however, the storage structure that we have described— consisting of registers, main memory, and magnetic disks—is only one of many possible storage systems. Others include cache memory, CD-ROM, magnetic tapes, and so on. The main differences among the various storage systems lie in speed, cost, size, and volatility.

The wide variety of storage systems in a computer system can be organized in a hierarchy (Figure) according to speed and cost. The higher levels are expensive, but

they are fast. As we move down the hierarchy, the cost per bit generally decreases, whereas the access time generally increases. This trade-off is reasonable; if a given storage system were both faster and less expensive than another—other properties being the same—then there would be no reason to use the slower, more expensive memory. In fact, many early storage devices, including paper tape and core memories, are relegated to museums now that magnetic tape and semiconductor memory have become faster and cheaper. The top four levels of memory in Figure may be constructed using semiconductor memory.

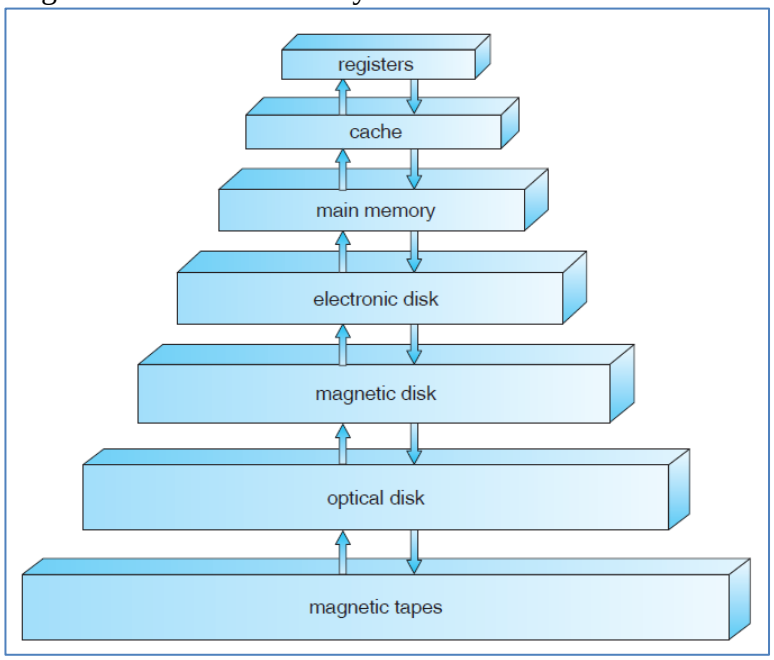


Figure : Storage-Device Hierarchy

In addition to differing in speed and cost, the various storage systems are either volatile or nonvolatile.

The design of a complete memory system must balance all the factors: it must use only as much expensive memory as necessary while providing as much inexpensive, nonvolatile memory as possible. Caches can be installed to improve performance where a large access-time or transfer-rate disparity exists between two components.

1.5.IO Structure

Storage is only one of many types of I/O devices within a computer. A large portion of operating-system code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system and because of the varying nature of the devices.

A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device. Depending on the controller, more than one device may be attached. For instance, seven or more devices can be attached to the small computer-systems interface (SCSI) controller. A device controller maintains some local buffer storage and a set of special-purpose registers. The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage. Typically, operating systems have a device driver for each device controller.

This device driver understands the device controller and presents a uniform interface to the device to the rest of the operating system.

To start an I/O operation, the device driver loads the appropriate registers within the device controller. The device controller, in turn, examines the contents of these registers to determine what action to take (such as “read a character from the keyboard”). The controller starts the transfer of data from the device to its local buffer. Once the transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation. The device driver then returns control to the operating system, possibly returning the data or a pointer to the data if the operation was a read.

For other operations, the device driver returns status information. This form of interrupt-driven I/O is fine for moving small amounts of data but can produce high overhead when used for bulk data movement such as disk I/O. To solve this problem, direct memory access (DMA) is used. After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU. Only one interrupt is generated per block, to tell the device driver that the operation has completed, rather than the one interrupt per byte generated for low-speed devices. While the device controller is performing these operations, the CPU is available to accomplish other work.

Some high-end systems use switch rather than bus architecture. On these systems, multiple components can talk to other components concurrently, rather than competing for cycles on a shared bus. In this case, DMA is even more effective. Figure shows the interplay of all components of a computer system.

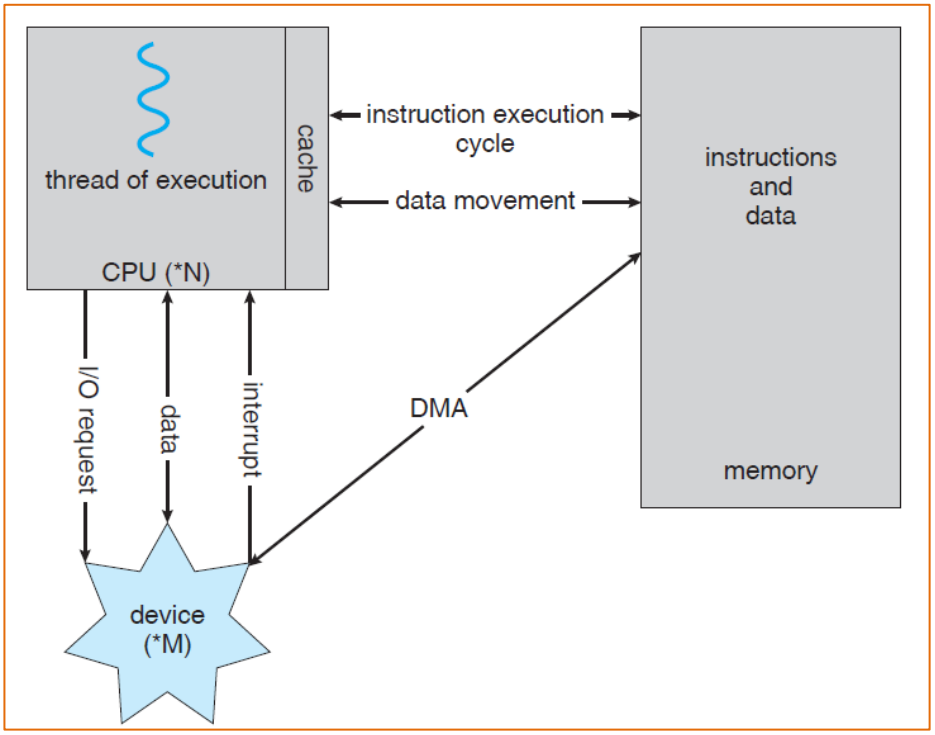


Figure : How a Modern Computer System Works

1.6. OS Services

Operating System provides environments in which programs run, and services for the users of the system, including:

- ✓ **User Interfaces** - Means by which users can issue commands to the system. Depending on the system these may be a command-line interface (e.g. sh, csh, ksh, tcsh, etc.), a GUI interface (e.g. Windows, X-Windows, KDE, Gnome, etc.), or a batch command systems. The latter are generally older systems using punch cards of job-control language, JCL, but may still be used today for specialty systems designed for a single purpose.

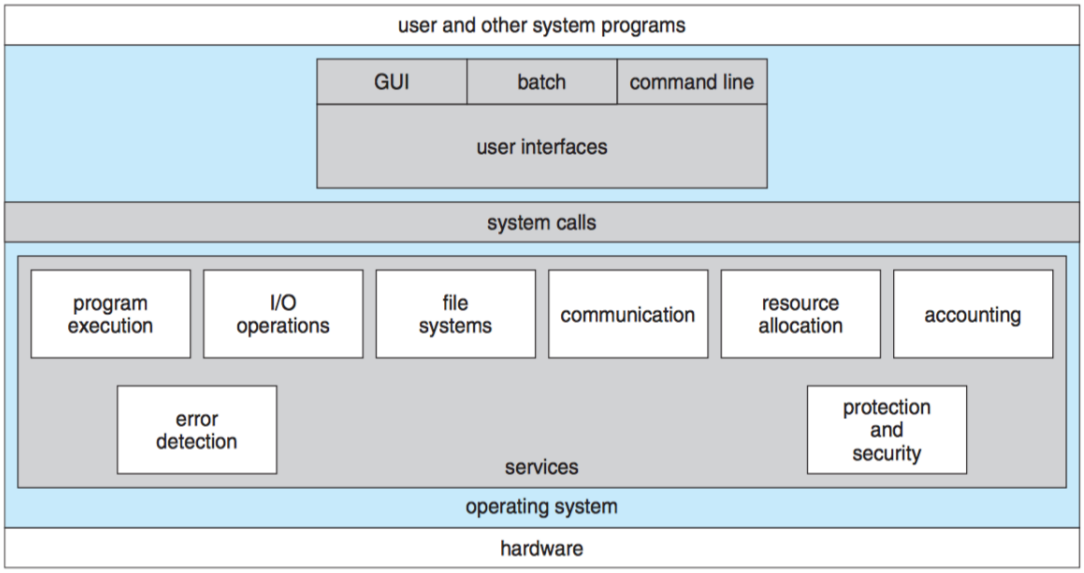
- ✓ **Program Execution** - The OS must be able to load a program into RAM, run the program, and terminate the program, either normally or abnormally.

- ✓ **I/O Operations** - The OS is responsible for transferring data to and from I/O devices, including keyboards, terminals, printers, and storage devices.

- ✓ **File-System Manipulation** - In addition to raw data storage, the OS is also responsible for maintaining directory and subdirectory structures, mapping file names to specific blocks of data storage, and providing tools for navigating and utilizing the file system.

- ✓ **Communications** - Inter-process communications, IPC, either between processes running on the same processor, or between processes running on separate processors or separate machines. May be implemented as either shared memory or message passing, (or some systems may offer both.)

- ✓ **Error Detection** - Both hardware and software errors must be detected and handled appropriately, with a minimum of harmful repercussions. Some systems may include complex error avoidance or recovery systems, including backups, RAID drives, and other redundant systems. Debugging and diagnostic tools aid users and administrators in tracing down the cause of problems.



Other systems aid in the efficient operation of the OS itself:

- ✓ **Resource Allocation** - E.g. CPU cycles, main memory, storage space, and peripheral devices. Some resources are managed with generic systems and others with very carefully designed and specially tuned systems, customized for a particular resource and operating environment.

- ✓ **Accounting** - Keeping track of system activity and resource usage, either for billing purposes or for statistical record keeping that can be used to optimize future performance.

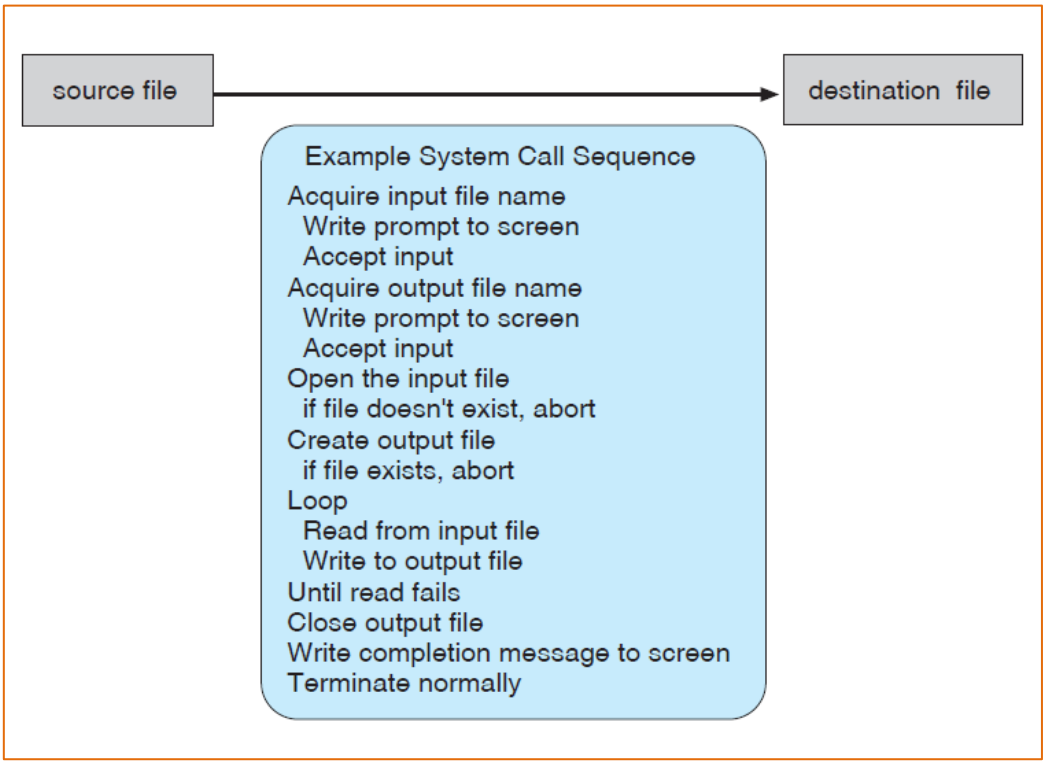
- ✓ **Protection and Security** - Preventing harm to the system and to resources, either through wayward internal processes or malicious outsiders. Authentication, ownership, and restricted access are obvious parts of this system. Highly secure systems may log all process activity down to excruciating detail, and security regulation dictate the storage of those records on permanent non-erasable medium for extended times in secure (off-site) facilities.

1.7.System Calls

We have seen that operating systems have two main functions: (1) Providing abstractions to user programs and (2) Managing the computer's resources.

System calls provide an interface to the servicesmade available by an operating system. These calls are generally available as routines written in C and C++, although certain low-level tasks (for example, tasks where hardware must be accessed directly) may need to be written using assembly-language instructions.

The example to illustrate how system calls are used: writing a simple program to read data from one file and copy them to another file.

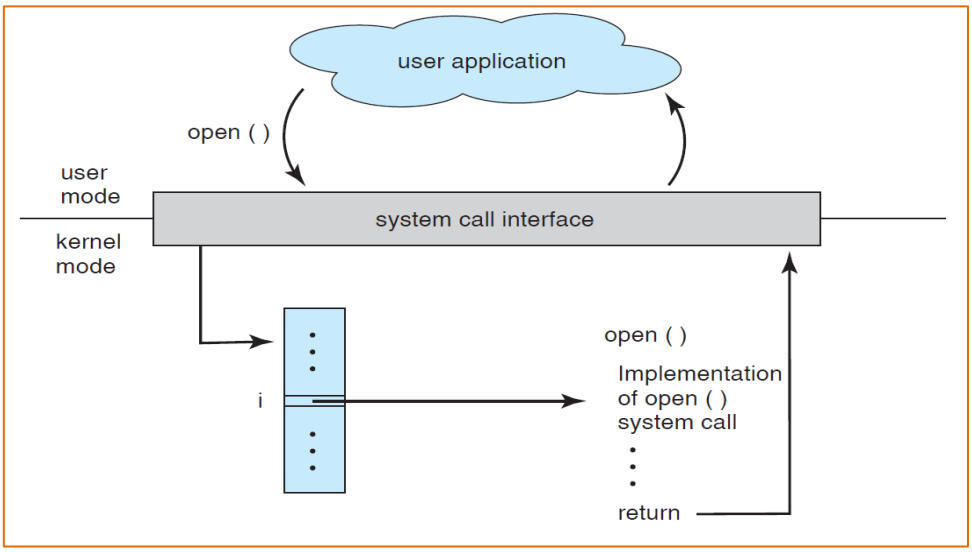


Most programmers never see this level of detail, however. Typically, application developers design programs according to an application programming interface (API). The API specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect.

Three of the most common APIs available to application programmers are theWin32 API forWindows systems, the POSIX API for POSIX-based systems (which include virtually all versions of UNIX, Linux, and Mac OS X), and the Java API for designing programs that run on the Java virtual machine.

Each operating system has its own name for each system call. The system-call interface intercepts function calls in the API and invokes the necessary system calls within the operating system. Typically, a number is associated with each system call, and the system-call interface maintains a table indexed according to these numbers. The system call interface then invokes the intended system call in the operating-system kernel and returns the status of the system call and any return values.

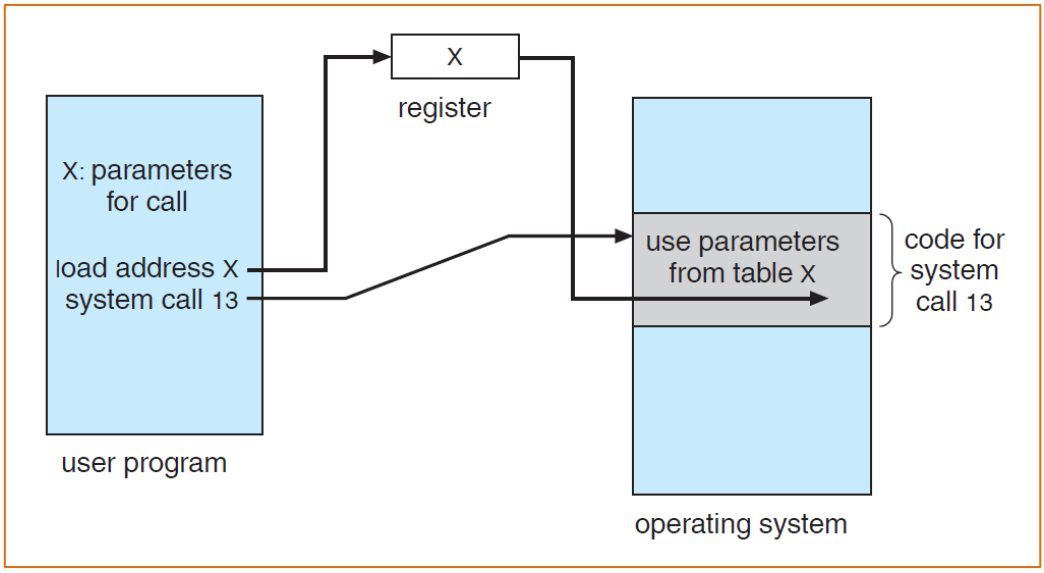
Most of the details of the operating-system interface are hidden from the programmer by the API and are managed by the run-time support library. The relationship between an API, the system-call interface, and the operating system is shown in Figure, which illustrates how the operating system handles a user application invoking the open() system call.



System calls occur in different ways, depending on the computer in use. Often, more information is required than simply the identity of the desired system call. The exact type and amount of information vary according to the particular operating system and call.

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, however, there may be more parameters than registers. In these cases, the parameters are generally stored in a ***block***, or ***table***, in memory, and the address of the block is passed as a

parameter in a register (Figure). This is the approach taken by Linux and Solaris. Parameters also can be placed, or pushed, onto the ***stack*** by the program and popped off the stack by the operating system. Some operating systems prefer the block or stack method because those approaches do not limit the number or length of parameters being passed.



Types of System Calls:

System calls can be grouped roughly into six major categories: Process control, File manipulation, Device manipulation, Information maintenance, Communications, and Protection. Figure summarizes the types of system calls normally provided by an operating system.

- 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

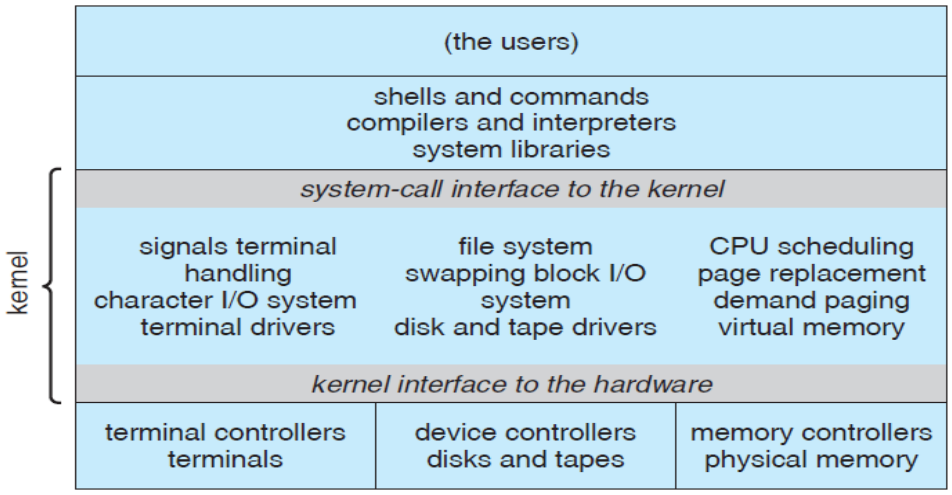
1.8. System Structure

1.9.Structure of an OS – Layered

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 and UNIX systems. The operating system can then retain much greater control over the computer and over the applications that make use of that computer. Under a top-down approach,

the overall functionality and features are determined and are 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 into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface. This layering structure is depicted in Figure.



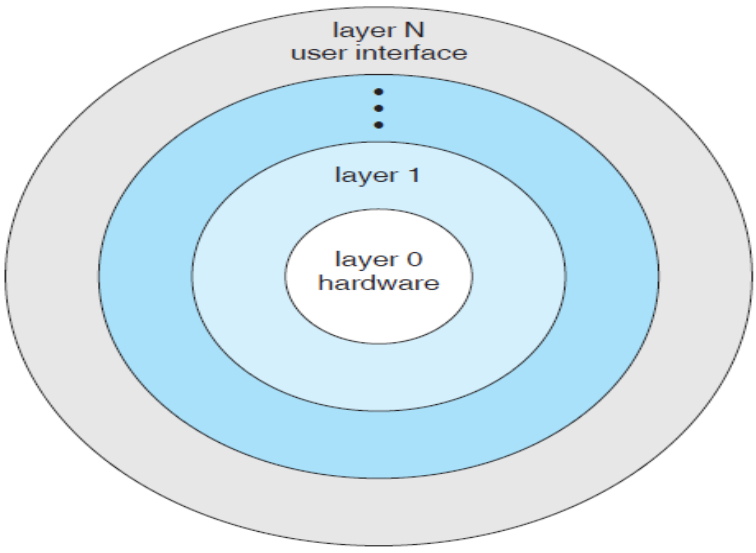


Figure 2.14 A layered operating system.

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—says, 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 are 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. For example, the device driver for the backing store (disk space used by virtual-memory algorithms) must be at a lower level than the memory-management routines, because memory management requires the ability to use the backing store.

Other requirements may not be so obvious. 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.

1.10. Virtual Machines

The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer.

By using CPU scheduling and virtual-memory techniques, an operating system host can create the illusion that a process has its own processor with its own (virtual) memory. The virtual machine provides an interface that is identical to the underlying bare hardware. Each guest process is provided with a (virtual) copy of the underlying

computer. Usually, the guest process is in fact an operating system, and that is how a single physical machine can run multiple operating systems concurrently, each in its own virtual machine.

Virtual machines first appeared commercially on IBM mainframes via the VM operating system in 1972. VM has evolved and is still available, and many of the original concepts are found in other systems, making this facility worth exploring.

IBM VM370 divided a mainframe into multiple virtual machines, each running its own operating system. A major difficulty with the VM virtual machine approach involved disk systems. Suppose that the physical machine had three disk drives but wanted to support seven virtual machines. Clearly, it could not allocate a disk drive to each virtual machine, because the virtual machine software itself needed substantial disk space to provide virtual memory and spooling. The solution was to provide virtual disks—termed minidisks in IBM's VMoperating system—that are identical in all respects except size. The system implemented each minidisk by allocating as many tracks on the physical disks as the minidisk needed.

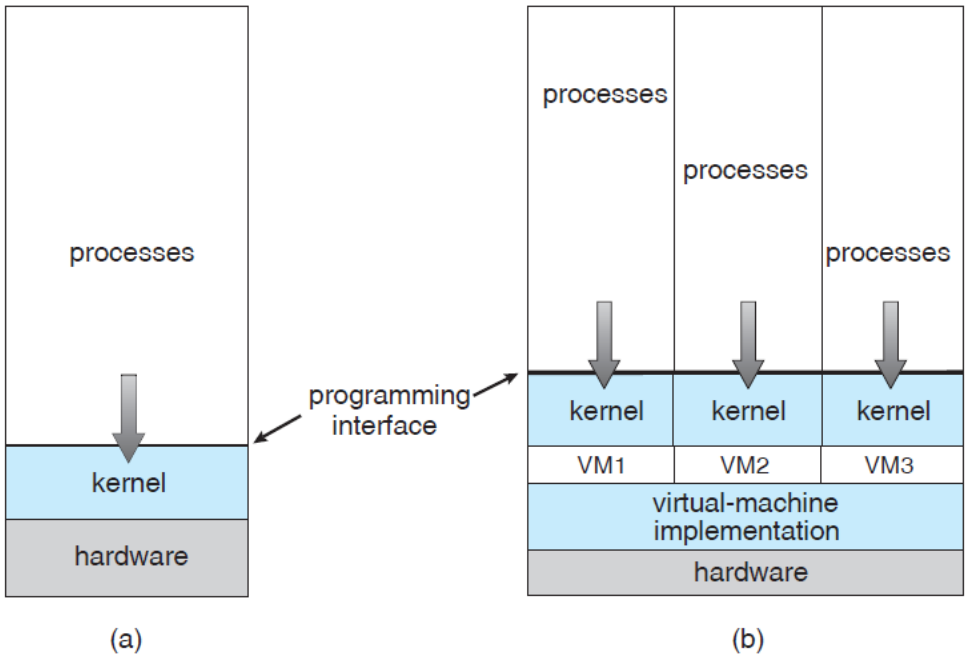


Figure : System Models (a) Nonvirtual Machine (b) Virtual Machine

Once these virtual machines were created, users could run any of the operating systems or software packages that were available on the underlying machine.

There are several reasons for creating a virtual machine. Most of them are fundamentally related to being able to share the same hardware yet run several different execution environments (that is, different operating systems) concurrently.

Advantages:

- ✓ The host system is protected from the virtual machines, just as the virtual machines are protected from each other. A virus inside a guest operating system might damage that OS but is unlikely to affect the host or the other guests.

- ✓ Two approaches to provide sharing have been implemented. First, it is possible to ***share a file-system volume and thus to share files***. Second, it is possible to define ***a network of virtual machines***, each of which can send information over the virtual communications network.

- ✓ A virtual-machine system is a perfect vehicle for operating-systems research and development. Normally, changing an operating system is a difficult task. Operating systems are large and complex programs, and it is difficult to be sure that a change in one part will not cause obscure bugs to appear in some other part.

- ✓ The operating system, however, runs on and controls the entire machine. Therefore, the current system must be stopped and taken out of use while changes are made and tested. A virtual-machine system can eliminate much of this problem. System programmers are given their own virtual machine, and system development is done on the virtual machine instead of on a physical machine.

- ✓ Another advantage of virtual machines for developers is that multiple operating systems can be running on the developer's workstation concurrently.

- ✓ This virtualized workstation allows for rapid porting and testing of programs in varying environments. Similarly, quality-assurance engineers can test their applications in multiple environments without buying, powering, and maintaining a computer for each environment.

- ✓ A major advantage of virtual machines in production data-center use is system consolidation, which involves taking two or more separate systems and running them in virtual machines on one system. Such physical-to-virtual conversions result in resource optimization, as many lightly used systems can be combined to create one more heavily used system.

1.11. Case study on UNIX and WINDOWS Operating System

Windows and UNIX differ in a fundamental way in their respective programming models.

A UNIX program consists of code that does something or other, making system calls to have certain services performed.

In contrast, a Windows program is normally event driven. The main program waits for some event to happen, then calls a procedure to handle it. Typical events are keys being struck, the mouse being moved, a mouse button being pushed, or a CD-ROM inserted. Handlers are then called to process the event, update the screen and update the internal program state.

All in all, this leads to a somewhat different style of programming than with UNIX, but since the focus of this book is on operating system function and structure, these different programming models will not concern us much more.

Of course, Windows also has system calls. With UNIX, there is almost a one to one relationship between the system calls (e.g., read) and the library procedures (e.g., read) used to invoke the system calls. In other words, for each system call, there is

roughly one library procedure that is called to invoke it, as indicated in Fig. Furthermore, POSIX has only about 100 procedure calls.

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

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

With Windows, the situation is radically different. To start with, the library calls and the actual system calls are highly decoupled. Microsoft has defined a set of procedures called the Win32 API (Application Program Interface) that programmers are expected to use to get operating system services.

The number of Win32 API calls is extremely large, numbering in the thousands. Furthermore, while many of them do invoke system calls, a substantial number are carried out entirely in user space. As a consequence, with Windows it is impossible to see what is a system call (i.e., performed by the kernel) and what is simply a user-space library call. In fact, what is a system call in one version of Windows may be done in user space in a different version, and vice versa.