UNIT – I Introduction

An operating system is a program that manages the computer hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware. An amazing aspect of operating systems is how varied they are in accomplishing these tasks. Mainframe operating systems are designed primarily to optimize utilization of hardware. Personal computer (PC) operating systems support complex games, business applications, and everything in between. Operating systems for handheld computers are designed to provide an environment in which a user can easily interface with the computer to execute programs. Thus, some operating systems are designed to be *convenient*, others to be *efficient*, and others some combination of the two.

Operating System (OS) Definition

An operating system (OS) is the program that, after being initially loaded into the computer by a boot program, manages all of the other application programs in a computer. The application programs make use of the operating system by making requests for services through a defined application program interface (API). In addition, users can interact directly with the operating system through a user interface, such as a command-line interface (CLI) or a graphical UI (GUI).

The operating system manages a computer's software hardware resources, including:

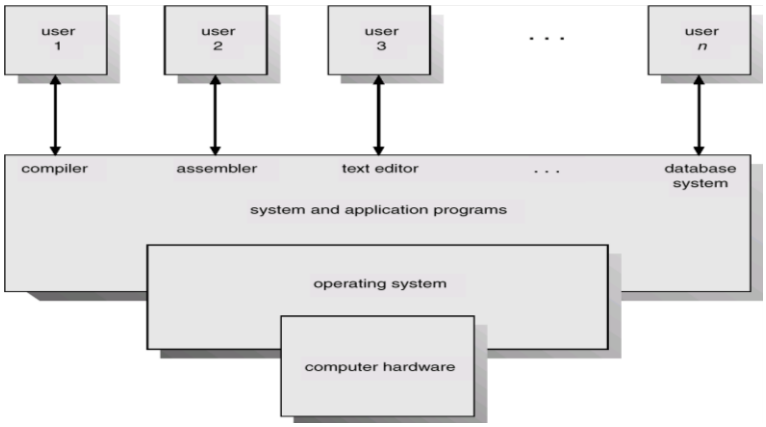
- Input devices such as a keyboard and mouse.
- Output devices such as display monitors and printers.
- Network devices such as modems, routers and network connections.
- Storage devices such as internal and external drives.

The OS also provides services to facilitate the efficient execution and management of, and memory allocations for, any additional installed software application programs.

If several programs are running at the same time (such as an Internet browser, firewall, and antivirus), the OS will allocate the computer's resources (memory, CPU, and storage) to make sure that each one of them receives what is needed to function

Role of Operating System

A computer system can be divided roughly into four components: the *hardware*, the *operating system*, the *application programs*, and the *users.*



(Abstract view of the components of a computer system.)

The hardware - the *Central Processing Unit* (CPU) the *memory*, and the *Input /Output devices* - provides the basic computing resources for the system. The *application programs* - such as word processors/ spreadsheets/ compilers, and Web browsers - define the ways in which these resources are used to solve users' computing problems. The operating system controls the hardware and coordinates its use among the various application programs for the various users.

We can also view a computer system as consisting of hardware, software, and data. Operating system provides the means for proper use of these resources in the operation of the computer system. Operating systems have two viewpoints: that of the *user* and that of the *system*.

User View

The user's view of the computer varies according to the interface being used. Most computer users sit in front of a PC, consisting of a monitor, keyboard, mouse, and system unit. Such a system is designed for one user to monopolize its resources. The goal is to maximize the work (or play) that the user is performing. In this case, the operating system is designed mostly for ease of use, with some attention paid to performance and none paid to resource utilization.

In other cases, a user sits at a terminal connected to a mainframe or a minicomputer. Other users are accessing the same computer through other terminals. These users share resources and may exchange information. The operating system in such cases is designed to maximize resource utilization - to assure that all available CPU time, memory, and I/0 are used efficiently and that no individual user takes more than the fair share.

System View

Operating system is the program most intimately involved with the hardware and the operating system's role as a resource allocator. A computer system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/0 devices, and so on. The operating system acts as the manager of these resources. Facing numerous and possibly conflicting requests for resources, the operating system must decide how to allocate them to specific programs and users so that it can operate the computer system efficiently and fairly. Resource allocation is especially important where many users access the same mainframe or minicomputer.

A slightly different view of an operating system emphasizes the need to control the various I/0 devices and user programs. An operating system is a control program. A control program manages the execution of user programs to prevent errors and improper use of the computer. It is especially concerned with the operation and control of I/O devices.

Functions of Operation System

An Operating System acts as a communication bridge (interface) between the user and computer hardware. The purpose of an operating system is to provide a platform on which a user can execute programs in a convenient and efficient manner. Functions are specified below:

(1) Processor Management

A program does nothing unless its instructions are executed by a CPU. A program in execution is a process. A time-shared user program such as a compiler is a process. A word-processing program

being run by an individual user on a PC is a process. A system task, such as sending output to a printer, can also be a process.

A process needs certain resources - including CPU time, memory, files, and I/O devices to accomplish its task. These resources are either given to the process when it is created or allocated to it while it is running. In addition to the various physical and logical resources that a process obtains when it is created, various initialization data (input) may be passed along

A process is the unit of work in a system. Such a system consists of a collection of processes, some of which are operating-system processes (those that execute system code) and the rest of which are user processes (those that execute user code). All these processes can potentially execute concurrently, for example, by multiplexing on a single CPU.

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

- Scheduling processes and threads on the CPUs
- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication

(2) 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, ranging in size from hundreds of thousands to billions. 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 both reads and writes data from main memory during the data-fetch. The main memory is generally the only large storage device that the CPU is able to address and access directly. For example, for the CPU to process data from disk, those data must first be transferred to main memory by CPU-generated I/O calls. In the same way, 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 both the utilization of the CPU and the speed of the computer's response to its users, general-purpose computers must keep several programs in memory, creating a need for memory management.

The operating system 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 (or parts thereof) and data to move into and out of memory
- Allocating and de-allocating memory space as needed

(3) Storage Management

To make the computer system convenient for users, the operating system provides a uniform, logical view of information storage. The operating system abstracts from the physical properties of

its storage devices to define a logical storage unit that is the file. The operating system maps files onto physical media and accesses these files via the storage devices.

3.1 File-System Management

File management is one of the most visible components of an operating system. Computers can store information on several different types of physical media. Magnetic disk, optical disk, and magnetic tape are the most common. Each of these media has its own characteristics and physical organization. Each medium is controlled by a device, such as a disk drive or tape drive, that also has its own unique characteristics. These properties include access speed, capacity, data-transfer rate, and access method (sequential or random).

A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary. Files may be free-form (for example, text files), or they may be formatted rigidly (for example, fixed fields). Clearly, the concept of a file is an extremely general one. Files are normally organized into directories to make them easier to use. When multiple users have access to files, it may be desirable to control by whom and in what ways (for example, read, write, append) files may be accessed.

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

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

3.2 Mass-Storage Management

The main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lost, 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, word processors, 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 in connection with disk management:

- Free-space management
- Storage allocation
- Disk scheduling

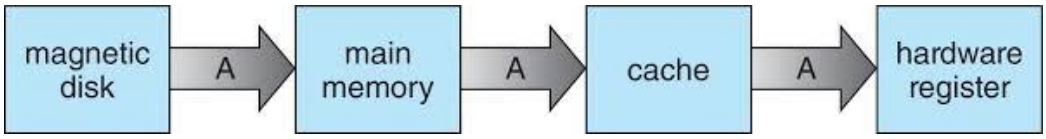
3.3 Caching

Cache is a high-speed memory with a limited size which locates in between the position of RAM and CPU registers. Internal programmable registers, such as index registers, provide a high-speed cache for main memory. The programmer (or compiler) implements the register-allocation and register-replacement algorithms to decide which information to keep in registers and which to keep in main memory. There are also caches that are implemented totally in hardware. For instance, most

systems have an instruction cache to hold the instructions expected to be executed next. Without this cache, the CPU would have to wait several cycles while an instruction was fetched from main memory. For similar reasons, most systems have one or more high-speed data caches in the memory hierarchy.

Main memory can be viewed as a fast cache for secondary storage, since data in secondary storage must be copied into main memory for use, and data must be in main memory before being moved to secondary storage for safekeeping. The file-system data, which resides permanently on secondary storage, may appear on several levels in the storage hierarchy. At the highest level, the operating system may maintain a cache of file-system data in main memory.

In a hierarchical storage structure, the same data may appear in different levels of the storage system. For example, suppose that an integer *A* that is to be incremented by 1 is located in file B, and file B resides on magnetic disk. The increment operation proceeds by first issuing an I/O operation to copy the disk block on which *A* resides to main memory. This operation is followed by copying *A* to the cache and to an internal register. Thus, the copy of *A* appears in several places: on the magnetic disk, in main memory, in the cache, and in an internal register. Once the increment takes place in the internal register, the value of *A* differs in the various storage systems. The value of *A* becomes the same only after the new value of *A* is written from the internal register back to the magnetic disk.



Migration of integer A from disk to register.

In a computing environment where only one process executes at a time, this arrangement poses no difficulties, since an access to integer *A* will always be to the copy at the highest level of the hierarchy. However, in a multitasking environment, where the CPU is switched back and forth among various processes, extreme care must be taken to ensure that, if several processes wish to access A, then each of these processes will obtain the most recently updated value of *A*.

(4) Protection and Security

If a computer system has multiple users and allows the concurrent execution of multiple processes, then access to data must be regulated. For that purpose, mechanisms ensure that files, memory segments, CPU, and other resources can be operated on by only those processes that have gained proper authorization from the operating system.

Protection, then, is any mechanism for controlling the access of processes or users to the resources defined by a computer system. This mechanism must provide means to specify the controls to be imposed and means to enforce the controls. Protection can improve reliability by detecting latent errors at the interfaces between component subsystems. Early detection of interface errors can often prevent contamination of a healthy subsystem by another subsystem that is malfunctioning. Furthermore, an unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user.

A system can have adequate protection but still be prone to failure and allow inappropriate access. Consider a user whose authentication information is stolen. This data could be copied or deleted, even though file and memory protection are working. It is the job of security to defend a system from external and internal attacks. Such attacks spread across a huge range and include viruses and worms, denial-of-service attacks (which use all of a system's resources and so keep legitimate users out of the system), identity theft, and theft of service (unauthorized use of a system).

Protection and security require the system to be able to distinguish among all its users. Most operating systems maintain a list of user names and associated user identifiers (user IDs). In Windows Vista parlance, this is a security ID (SID). These numerical IDs are unique, one per user. When a user logs in system, the authentication stage determines the appropriate user ID for the user. That user ID is associated with all of the user's processes and threads. When an ID needs to be user readable, it is translated back to the user name via the user name list.

(5) Device Management

An Operating System manages device communication via their respective drivers. It performs the following activities for device management.

- Keeps tracks of all devices connected to system.
- Designates a program responsible for every device known as the Input/output controller.
- Decides which process gets access to a certain device and for how long.
- Allocates devices in an effective and efficient way.
- De-allocates devices when they are no longer required.

(6) Control over system performance

- Monitors overall system health to help improve performance.
- Records the response time between service requests and system response.
- Improves the performance by providing important information needed to troubleshoot problems.

(7) Coordination between other software and users

Operating systems also coordinate and assign interpreters, compilers, assemblers and other software to the various users of the computer systems.

Evolution of OS

The evolution of operating systems is directly dependent on the development of computer systems and how users use them. The following specifications are the computing systems through the past 75 years in the timeline.

Early Evolution

- 1945: ENIAC, Moore School of Engineering, University of Pennsylvania.
- 1949: EDSAC and EDVAC
- 1949: BINAC - a successor to the ENIAC

- 1951: UNIVAC by Remington
- 1952: IBM 701
- 1956: The interrupt
- 1954-1957: FORTRAN was developed

Operating Systems - Late 1950s

By the late 1950s Operating systems were well improved and started supporting following usages:

- It was able to perform Single stream batch processing.
- It could use Common, standardized, input/output routines for device access.
- Program transition capabilities to reduce the overhead of starting a new job were added.
- Error recovery to clean up after a job terminated abnormally was added.
- Job control languages that allowed users to specify the job definition and resource requirements were made possible.

Operating Systems - In 1960s

- 1961: The dawn of minicomputers
- 1962: Compatible Time-Sharing System (CTSS) from MIT
- 1963: Burroughs Master Control Program (MCP) for the B5000 system
- 1964: IBM System/360
- 1960s: Disks became mainstream
- 1966: Minicomputers got cheaper, more powerful, and really useful.
- 1967-1968: Mouse was invented.
- 1964 and onward: Multics
- 1969: The UNIX Time-Sharing System from Bell Telephone Laboratories.

Supported OS Features by 1970s

- Multi User and Multi tasking was introduced.
- Dynamic address translation hardware and Virtual machines came into picture.
- Modular architectures came into existence.
- Personal, interactive systems came into existence.

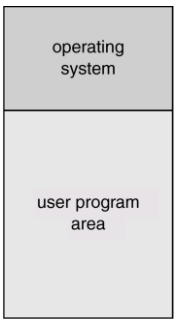
Accomplishments after 1970

- 1971: Intel announces the microprocessor
- 1972: IBM comes out with VM: the Virtual Machine Operating System
- 1973: UNIX 4th Edition is published
- 1973: Ethernet
- 1974 The Personal Computer Age begins
- 1974: Gates and Allen wrote BASIC for the Altair
- 1976: Apple II
- August 12, 1981: IBM introduces the IBM PC

- 1983 Microsoft begins work on MS-Windows
- 1984 Apple Macintosh comes out
- 1990 Microsoft Windows 3.0 comes out
- 1991 GNU/Linux
- 1992 The first Windows virus comes out
- 1993 Windows NT
- 2007: iOS
- 2008: Android OS

The Batch System

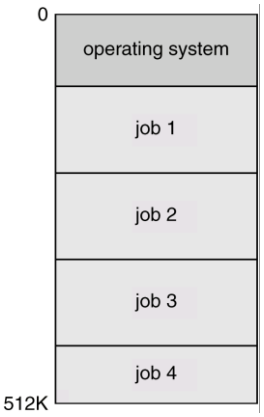
It is used by improving the utilization and application of computers. Jobs were scheduled and submitted on cards and tapes. Then it was sequentially executed on the monitors by using Job Control Language. The first computers are used in the process of the batch operating process made the computer batch of jobs without any pause or stop. The program is written in the punch cards and then copied to the processing unit of the tape. When the computer completed a single job, it instantly begins the next task on the tape. Professional operators are trained to communicate with the machine where the users dropped the jobs and fetched back to pick the results after the job is executed.



Though it is uncomfortable for the users it is made to keep the expensive computer as busy up to the extent by running a leveraged stream of jobs. The protection of memory doesn't allow the memory area comprises the monitor to alter and the timer protects the job from monopolizing the system. The processor sustains as idle when the input and output devices are in use by the bad utilization of CPU time.

Multi-programmed Batch System

It is used to have several jobs to execute which should be held in main memory. Job scheduling is made up of the processor to decide which program to execute.



I/O routine is supplied by the system. The system must allocate the memory to several jobs. The system must choose among several jobs ready to run as CPU scheduling and allocation of devices.

Time-Shared Operating System

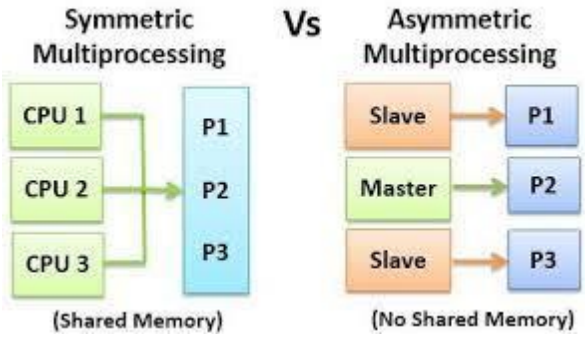
It is used to develop the substitute batch systems. The user communicated directly with the computer by printing ports like an electric teletype. Few users shared the computer instantaneously and spent a fraction of a second on every job before starting with the next one. The fast server can act on many users' processes instantly by creating the iteration when they were receiving its full attention. The Timesharing systems are used by multiple programs to apply to the computer system by sharing the system interactively.

The multi-programming is used to manage multiple communicative jobs as interactive computing. The time of the processor is shared among multiple users and many users can simultaneously access the system via terminals. Printing ports needed that programs with the command-line user interface, where the user has written responses to prompt or written commands.

Parallel System

This is a multiprocessor system with more than one CPU in close communication. Communication usually takes place through the shared memory that is *tightly coupled system*–processors share memory and a clock. Main advantages are increased *throughput*, economical and increased reliability- i.e., graceful degradation and fail-soft systems.

Parallel system works in two ways- Symmetric multiprocessing (SMP) And Asymmetric multiprocessing.



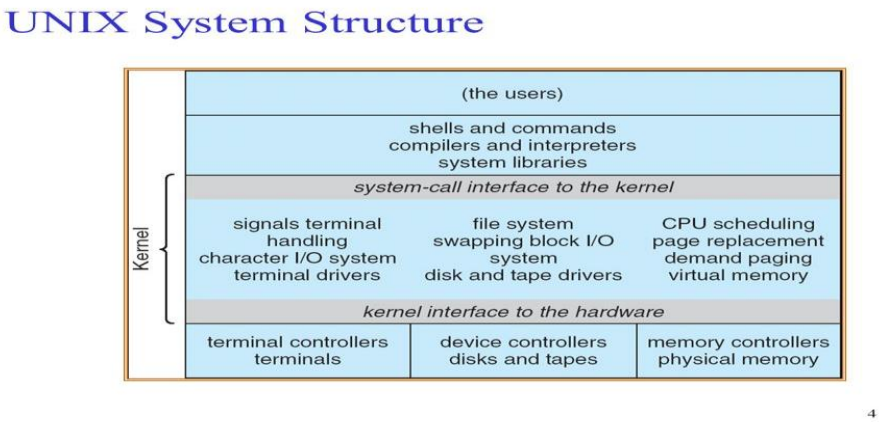
Operating System Structure

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. These components are interconnected and melded into a kernel.

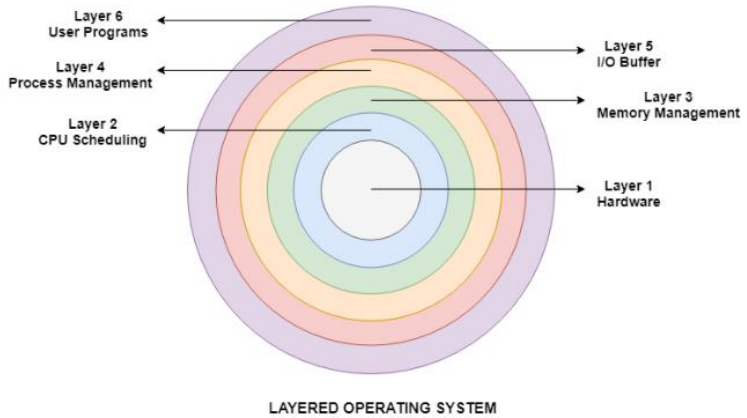
Layered Approach

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 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. Implementers have more freedom in changing the inner workings of the system and in creating modular operating systems. 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.



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 the following figure. 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.



LAYERED OPERATING SYSTEM

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. 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.

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/0 operation, it executes a system call that is

trapped to the I/0 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

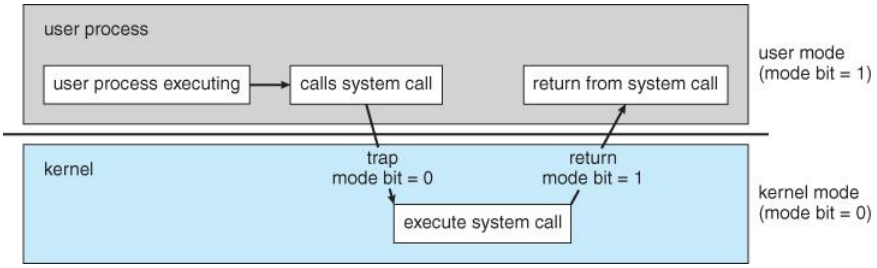
Operating System Operations

Modern operating systems are interrupt driven. If there are no processes to execute, no I/0 devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen. Events are almost always signalled by the occurrence of an interrupt or a trap. (or an exception) is a software-generated interrupt caused either by an error (for division by zero or invalid memory access) or by a specific request from a user program that an operating system service be performed. For each type of interrupt, separate segments of code in the operating system determine what action should be taken.

Dual-Mode Operation

The proper execution of the operating system must be able to distinguish between the execution of operating-system code and user defined code. The approach taken by most computer systems is to provide hardware support that allows us to differentiate among various modes of execution.

At the very least, we need two separate mode of operation: **user mode** and **kernel mode** (also called **supervisor mode, system mode or privileged mode**). A bit, called the mode bit, is added to the hardware of the computer to indicate the current mode: **kernel(0)** or **user(1)**. With the mode bit we are to distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of user. When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), a transmission occurred from user to kernel mode to fulfil the request.



At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode. The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.

The dual mode of operation provides us with the means for protecting the operating system from errant users. It is accomplish this protection by designating some of the machine instructions that may cause harm as privileged instructions. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system. The instruction to switch to kernel mode is an example of a privileged instruction. Some other examples include I/0 control timer management and interrupt management. Initial control resides in the operating system, where instructions are executed in kernel mode.

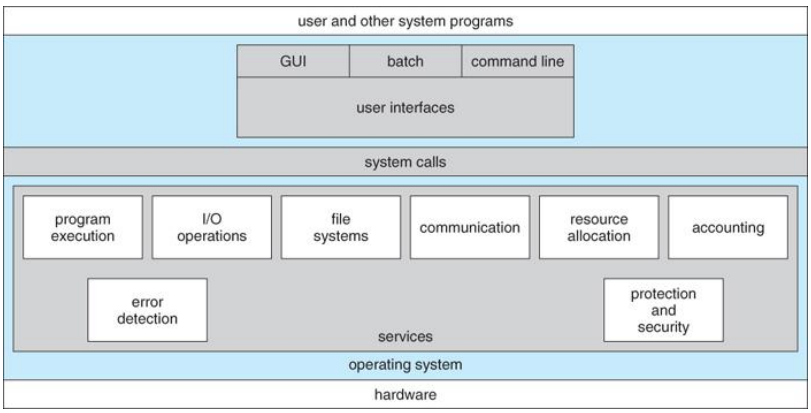
Timer

Operating system maintains control over the CPU. It cannot allow a user program to get stuck in an infinite loop or to fail to call system services and never return control to the operating system. To accomplish this goal, it can use a timer. A timer can be set to interrupt the computer after a specified period. The period may be fixed (for example, 1/60 second) or variable (for example, from 1 millisecond to 1 second). A variable timer is generally implemented by a fixed-rate clock and a counter. The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs. For instance, a 10-bit counter with a 1-millisecond clock allows interrupts at intervals from 1 millisecond to 1,024 milliseconds, in steps of 1 millisecond.

Before turning over control to the user, the operating system ensures that the timer is set to interrupt. If the timer interrupts, control transfers automatically to operating system which may treat the interrupt as a fatal error or may give the program more time. Clearly, instructions that that modify the content of the timer are privileged. Thus, it can use the timer to prevent a user program from running too long.

Operating System Services

An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs. These operating-system services are provided for the convenience of the programmer, to make the programming task easier. The following figure shows one view of the various operating-system services and how they interrelate.



One set of operating-system services provides functions that are helpful to the user.

- **User interface**

Almost all operating systems have a user interface (UI). This interface can take several forms. One is Command-line interface (CLI), which uses text commands and a method for entering them (say, a program to allow entering and editing of commands). Another is a batch interface in which commands and directives to control those commands are entered into files, and those files are executed. Most commonly, a Graphical User Interface (GUI) is used. Here, the interface is a window system with a pointing device to direct I/0, choose from menus, and make selections and a keyboard to enter text. Some systems provide two or all three of these variations.

- **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/0, which may involve a file or an I/0 device. For specific devices, special functions may be desired (such as recording to a CD or DVD drive or blanking a display screen). For efficiency and protection, users usually cannot control I/0 devices directly. Therefore, the operating system must provide a means to do I/0.

- **File-system manipulation**

The file system is of particular interest. Obviously, programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information. Finally, some programs include permissions management to allow or deny access to files or directories based on file ownership. Many operating systems provide a variety of file systems, sometimes to allow personal choice, and sometimes to provide specific features or performance characteristics.

- **Communications**

There are many circumstances in which one process needs to exchange information with another process. Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network. Communications may be implemented via *shared memory* or through *message passing*, in which packets of information are moved between processes by the operating system.

- **Error detection**

The operating system needs to be constantly aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/0 devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing. Of course, there is variation in how operating systems react to and correct errors. Debugging facilities can greatly enhance the user's and programmer’s abilities to use the system efficiently.

Another set of operating-system functions exists not for helping the user but rather for ensuring the efficient operation of the system itself.

- **Resource allocation**

When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system. Some (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/0 devices) may have much more general request and release code. For instance, in determining how best to use the CPU, operating systems have CPU-scheduling routines that take into account the speed of the CPU, the jobs that must be executed, the number of registers available, and other factors. There may also be routines to allocate printers, modems, USB storage drives, and other peripheral devices.

- **Accounting**

It is to recognize that which users use how much and what 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 and security**

The owners of information stored in a multiuser or networked computer system may want to control use of that information. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important. Such security starts with requiring each user to authenticate the person to the system, usually by means of a password, to gain access to system resources. It extends to defending external I/0 devices, including modems and network adapters, from invalid access attempts and to recording all such connections for detection of break-ins.

User Operating System Interface

There are several ways for users to interface with the operating system. From this two fundamental approaches that is, one provides a **command-line interface**, or that allows users to directly enter commands to be performed by the operating system. The other allows users to interface with the operating system via a **Graphical User Interface**, or GUI.

Command Interpreter

The main function of the command interpreter is to get and execute the next user-specified command. Many of the commands given at this level manipulate files: create, delete, list, print, copy, execute, and so on. The MS-DOS and UNIX shells operate in this way. These commands can be implemented in two general ways.

In one approach, the command interpreter itself contains the code to execute the command. For example, a command to delete a file may cause the command interpreter to jump to a section of its code that sets up the parameters and makes the appropriate system call. In this case, the number of commands that can be given determines the size of the command interpreter, since each command requires its own implementing code.

An alternative approach - used by UNIX, among other operating systems implements most commands through system programs. In this case, the command interpreter does not understand the command in any way; it merely uses the command to identify a file to be loaded into memory and executed. Thus, the UNIX command to delete a file

rm file.txt

would search for a file called **rm**, load the file into memory, and execute it with the parameter **file.txt**. The function associated with the **rm** command would be defined completely by the code in the file **rm**. In this way, programmers can add new commands to the system easily by creating new files with the proper names. Example for this type of OS interface is MS-DOS and Unix.

Graphical User Interfaces

A second strategy for interfacing with the operating system is through a user friendly graphical user interface, or CUI. Here, rather than entering commands directly via a command-line interface, users employ a mouse-based window-and-menu system characterized by a desktop metaphor. The user moves the mouse to position its pointer on images, or on the screen (the desktop) that represent programs, files, directories, and system functions. Depending on the mouse pointer's location,

clicking a button on the mouse can invoke a program, select a file or directory known as a folder or pull down a menu that contains commands. Example for this type of OS interface is Windows and Linux.

System Calls

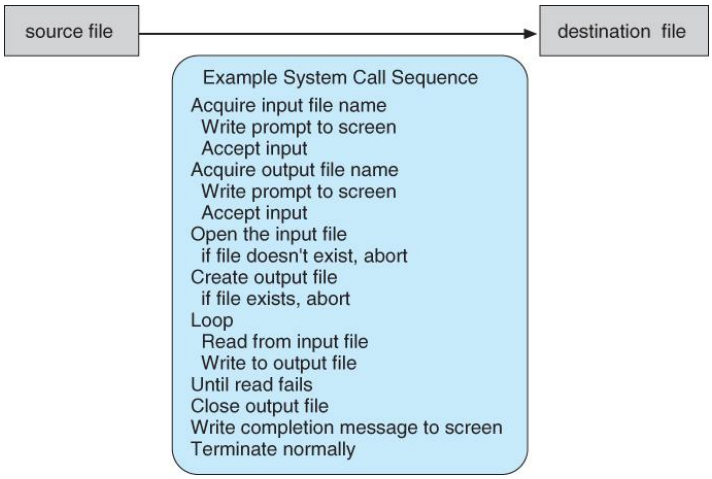
System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++, although certain low-level tasks (for example, tasks where hardware must be accessed directly), may need to be written using assembly-language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

Basic Concept of system Calls

Once the two file names are obtained, the program must open the input file and create the output file. Each of these operations requires system call. There are also possible error conditions for each operation. When the program tries to open the input file, it may find that there is no file of that name or that the file is protected against access. In these cases, the program should print a message on the console (another sequence of system calls) and then terminate abnormally (another system call). If the input file exists, then we must create a new output file. We may find that there is already an output file with the same name. This situation may cause the program to abort (a system call), or we may delete the existing file (another system call) and create a new one (another system call). Another option, in an interactive system, is to ask the user (via a sequence of system calls to output the prompting message and to read the response from the terminal) whether to replace the existing file or to abort the program.

Now that both files are set up, we enter a loop that reads from the input file (a system call) and writes to the output file (another system call). Each read and write must return status information regarding various possible error conditions. On input, the program may find that the end of the file has been reached or that there was a hardware failure in the read (such as a parity error). The write operation may encounter various errors, depending on the output device (no more disk space, printer out of paper, and so on).

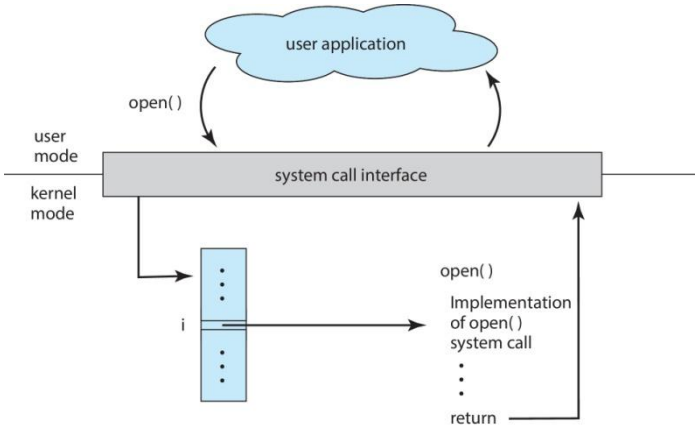
Finally, after the entire file is copied, the program may close both files (another system call), write a message to the console or window (more system calls), and finally terminate normally (the final system call). Frequently/ systems execute thousands of system calls per second.



System Call Interface

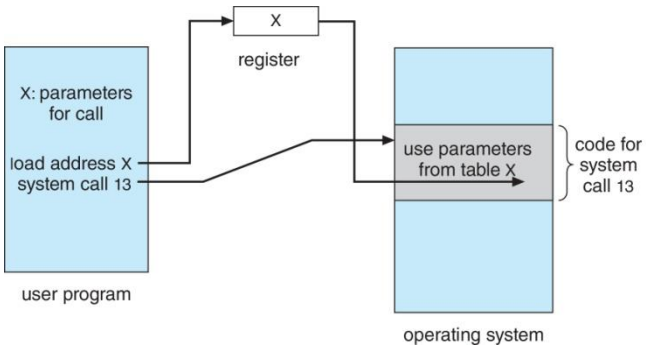
The run-time support system (a set of functions built into libraries included with a compiler) for most programming languages provides a **system-call interface** that serves as the link to system calls made available by the operating system. The system-call interface intercepts function calls in the 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.

The relationship between an API, the system-call interface, and the operating system is shown in the following figure, which illustrates how the operating system handles a user application invoking the open() system call.



(The handling of a user application invoking the open() system call)

Passing Parameters to OS



(The handling of a user application invoking the open () system call)

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. 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**, **communication**, **device manipulation**, **information maintenance**, **communications**, and **protection**. The following entries are 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

Figure 2.8 Types of system 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()`
	`InitlializeSecurityDescriptor()`	`umask()`
	`SetSecurityDescriptorGroup()`	`chown()`

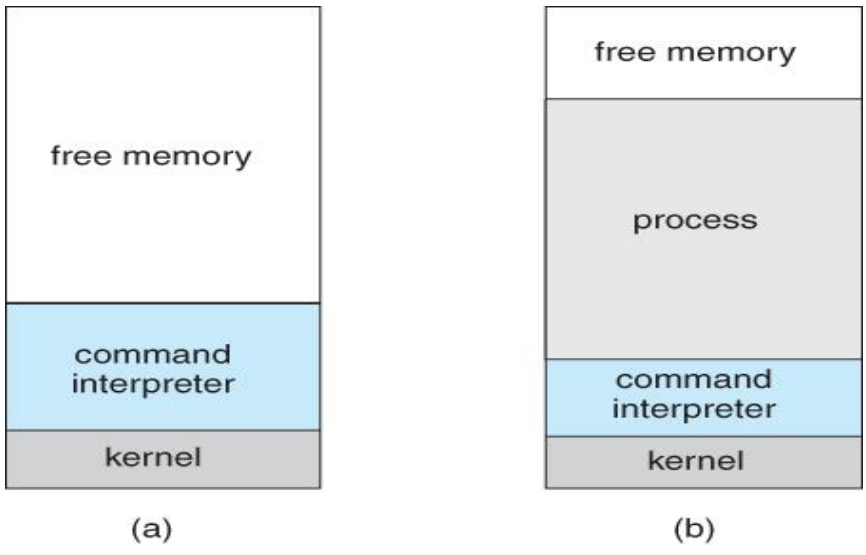
Process Control

A running program needs to be able to halt its execution either normally (end) or abnormally (abort). If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated. Under either normal or abnormal circumstances, the operating system must transfer control to the invoking command interpreter. The command interpreter then reads the next command.

In an interactive system, the command interpreter simply continues with the next command; it is assumed that the user will issue an appropriate command to respond to any error. In a GUI system, a pop-up window might alert the user to the error and ask for guidance. In a batch system, the command interpreter usually terminates the entire job and continues with the next job.

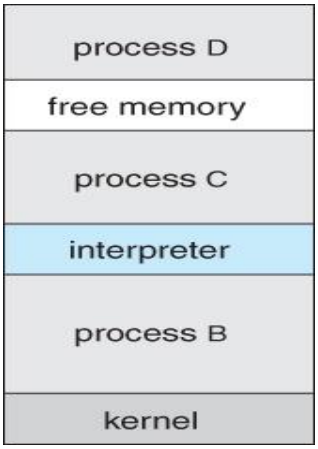
A process or job executing one program may want to load and execute another program. This feature allows the command interpreter to execute a program as directed by, for example, a user command, the click of a mouse, or a batch command. This is related to the problem of whether the existing program is lost, saved, or allowed to continue execution concurrently with the new program. There are so many facets of and variations in process and job control that shows two examples. One is involving a **single-tasking** system (MS-DOS) and the other a **multitasking** system (UNIX).

MS-DOS execution



(At system start-up) (Running a program)

FreeBSD (derived from Berkeley UNIX) is an example of a multitasking system. When a user logs on to the system, the shell of the user's-choice is run. It accepts commands and executes programs that the user requests. However, since FreeBSD is a multitasking system, the command interpreter may continue running while another program is executed.



(FreeBSD running multiple programs)

To start a new process, the shell executes a fork() system call. Then, the selected program is loaded into memory via an exec() system call, and the program is executed. Depending on the way the command was issued, the shell then either waits for the process to finish or runs the process "in the background." In the latter case, the shell immediately requests another command. When a process is running in the background, it cannot receive input directly from the keyboard, because the shell is using this resource. When the process is done, it executes an exit () system call to terminate, returning to the invoking process a status code of 0 or a nonzero error code.

File Management

Now identify several common system calls dealing with files. First, need to be able to create and delete files. Either system call requires the name of the file and perhaps some of the file's attributes. Once the file is created, it needs to open it and to use it. It may also read, write, or reposition (rewinding or skipping to the end of the file, for example). Finally, it needs to close the file, indicating that this is no longer using it.

We have a directory structure for organizing files in the file system. In addition, for either files or directories, we need to be able to determine the values of various attributes and perhaps to reset them if necessary. File attributes include the file name, file type, protection codes, accounting information, and so on. At least two system calls, get file attribute and set file attribute, are required for this function. Some operating systems provide many more calls, such as calls for file move and copy. Others might provide an API that performs those operations using code and other system calls, and others might just provide system programs to perform those tasks.

Device Management

A process may need several resources to execute - main memory, disk drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available.

The various resources controlled by the operating system can be thought of as devices. Some of these devices are physical devices (for example, disk drives), while others can be thought of as abstract or virtual devices (for example, files). A system with multiple users may require to first request the device, to ensure exclusive use of it. After we are finished with the device, we release it. These functions are similar to the open and close system calls for files. Once the device has been requested (and allocated), it can read, write, and (possibly) reposition the device.

Information Maintenance

Many system calls exist simply for the purpose of transferring information between the user program and the operating system. For example, most systems have a system call to return the current time and date. Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on.

Many operating systems provide a time profile of a program to indicate the amount of time that the program executes at a particular location or set of locations. A time profile requires either a tracing or regular timer interrupts. At every occurrence of the timer interrupt, the value of the program counter is recorded.

In addition, the operating system keeps information about all its processes, and system calls are used to access this information. Generally, calls are also used to reset the process information (get process attributes and set process attributes).

Communication

There are two common models of inter-process communication: the **message passing model** and the **shared memory model**. In the message passing model, the communication processes exchange messages with one another to transfer information. Messages can be exchanged between the processes either directly or indirectly through a common mailbox. Before communication can take place, a connection must be opened. The name of the other communicator must be known, be it another process on the same system or a process on another computer connected by a communications network. Each computer in a network has a host name by which it is commonly known. A host also has a network identifier, such as an IP address. Similarly, each process has a process name, and this name is translated into an identifier by which the operating system can refer to the process. The source of the communication, known as the *client*, and the receiving daemon, known as a *server*, then exchange messages by using read message and write message system calls.

In the **shared memory model,** processes use *shared memory create* and *shared memory attach* system calls to create and gain access to regions of memory owned by other processes. Recall that, normally, the operating system tries to prevent one process from accessing another process's memory. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas. The form of the data is determined by the processes and is not under the operating system's control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

Protection

Protection provides a mechanism for controlling access to the resources provided by a computer system. Historically, protection was a concern only on multi-programmed computer systems with several users. However, with the advent of networking and the Internet, all computer systems, from servers to PDAs, must be concerned with protection.

Typically, system calls providing protection include *set permission* and *get permission*, which manipulate the permission settings of resources such as files and disks. The *allow user* and *deny user* system calls specify whether particular users can - or cannot - be allowed access to certain resources.