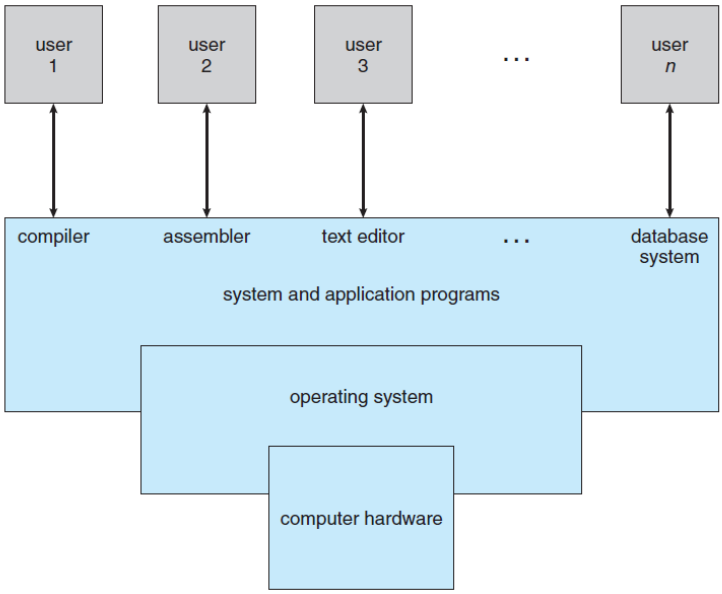
UNIT - I

Operating System Introduction

System components

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

- **Hardware**: The hardware the central processing unit (CPU), the memory, and the input/output (I/O) devices—provide the basic computing resources for the system.
- **Application programs**: **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.
- **Operating system**: 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. The operating system provides the means for proper use of these resources in the operation of the computer system. An operating system is similar to a government. Like a government, it performs no useful function by itself. It simply provides an ***environment*** within which other programs can do useful work.

Operating System Definition

An **operating system** is a program that manages a computer's hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware.

or

An Operating system is one of the programs running at all times on the computer, usually called kernel.

Structures

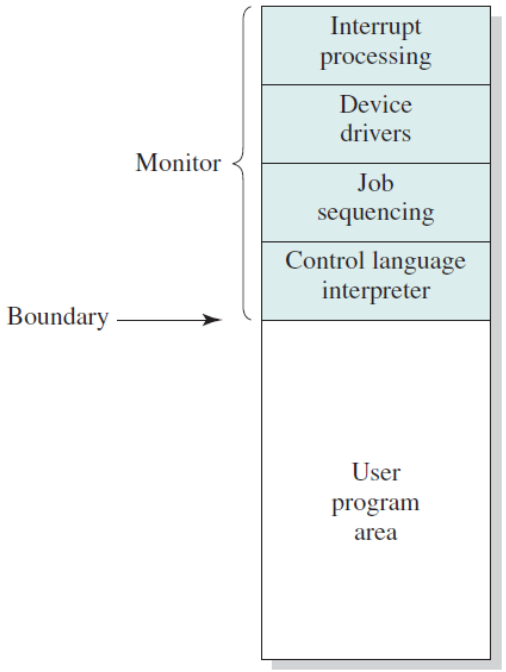
Serial Processing

- With the earliest computers, from the late 1940s to the mid-1950s, the programmer interacted directly with the computer hardware; there was no OS.

- These computers were run from a console consisting of display lights, toggle switches, some form of input device, and a printer.
- The users have access to computer in series.
- If the programmer wish to execute a program, they need to follow certain steps,
  - Programs in machine code were loaded via the input device (e.g., a card reader).
  - If an error halted the program, the error condition was indicated by the lights.
  - If the program proceeded to a normal completion, the output appeared on the printer.
- **Two main problems:**
  - **Scheduling:** A user could sign up for a block of time in multiples of a half hour or so. A user might sign up for an hour and finish in 45 minutes; this would result in wasted computer processing time. On the other hand, the user might run into problems, not finish in the allotted time, and be forced to stop before resolving the problem.
  - **Setup time:** A single program, called a **job**, could involve loading the compiler plus the high-level language program (source program) into memory, saving the compiled program (object program), and then loading and linking together the object program and common functions. Each of these steps could involve mounting or dismounting tapes or setting up card decks. If an error occurred, the hapless user typically had to go back to the beginning of the setup sequence. Thus, a considerable amount of time was spent just in setting up the program to run.

Simple Batch

- Early computers were very expensive, and wasted time, so the concept of batch OS was developed
- **Monitor**
  - The central idea behind the simple batch-processing scheme is the use of a piece of software known as the **monitor**. With this type of OS, the user no longer has direct access to the processor.
  - Instead, the user submits the job on cards or tape to a computer operator, who batches the jobs together sequentially and places the entire batch on an input device, for use by the monitor.
  - Each program is constructed to branch back to the monitor when it completes processing, at which point the monitor automatically begins loading the next program.
- **Two view points**
  - **Monitor point of view:** The monitor controls the sequence of events. For this to be so, much of the monitor must always be in main memory and available for execution. That portion is referred to as the **resident monitor.** The rest of the monitor consists of utilities and common functions that are loaded as subroutines to the user program at the beginning of any job that requires them. The monitor reads in jobs one at a time from the input device (typically a card reader or magnetic tape drive). As it is read in, the current job is placed in the user program area, and control is passed to this job. When the job is completed, it returns control to the monitor, which immediately reads in the next job. The results of each job are sent to an output device, such as a printer, for delivery to the user.



- **Processor point of view:** At a certain point, the processor is executing instructions from the portion of main memory containing the monitor. These instructions cause the next job to be read into another portion of main memory. Once a job has been read in, the processor will encounter a branch instruction in the monitor that instructs the processor to continue execution at the start of the user program. The processor will then execute the instructions in the user program until it encounters an ending or error condition. Either event causes the processor to fetch its next instruction from the monitor program.

- The monitor performs a scheduling function: A batch of jobs is queued up, and jobs are executed as rapidly as possible, with no intervening idle time. The monitor improves job setup time using **job control language (JCL)**.

Multi-programmed

- A single program cannot, in general, keep either the CPU or the I/O devices busy at all times. Single users frequently have multiple programs running.
- **Multiprogramming** increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute.
- The idea is as follows: The operating system keeps several jobs in memory simultaneously. Since, in general, main memory is too small to accommodate all jobs; the jobs are kept initially on the disk in the **job pool**. This pool consists of all processes residing on disk awaiting allocation of main memory.
- The set of jobs in memory can be a subset of the jobs kept in the job pool. The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete.
- In a non-multiprogrammed system, the CPU would sit idle. In a multiprogrammed system, the operating system simply switches to, and executes, another job. When ***that*** job needs to wait, the CPU switches to ***another*** job, and so on.
- Eventually, the first job finishes waiting and gets the CPU back. As long as at least one job needs to execute, the CPU is never idle.

- This idea is common in other life situations. A lawyer does not work for only one client at a time, for example. While one case is waiting to go to trial or have papers typed, the lawyer can work on another case. If he has enough clients, the lawyer will never be idle for lack of work.

Time-shared

- **Time sharing** (or **multitasking**) is a logical extension of multiprogramming. In time-sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.
- Time sharing requires an **interactive** computer system, which provides direct communication between the user and the system. The user gives instructions to the operating system or to a program directly, using an input device such as a keyboard, mouse, touch pad, or touch screen, and waits for immediate results on an output device. Accordingly, the **response time** should be short—typically less than one second.
- A time-shared operating system allows many users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user. As the system switches rapidly from one user to the next, each user is given the impression that the entire computer system is dedicated to his use, even though it is being shared among many users.
- A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer.

Personal Computer

- A Personal computer is a small relatively in expensive computer designed for an individual user. These are based on microprocessor technology that enables manufactures to put an entire CPU on one chip.
- The goal is to maximize the work (or play) that the user is performing with some attention paid to performance and none paid to **resource utilization.**
- . Personal computers are used for word processing, accounting, running spreadsheets, database management applications, games etc.,

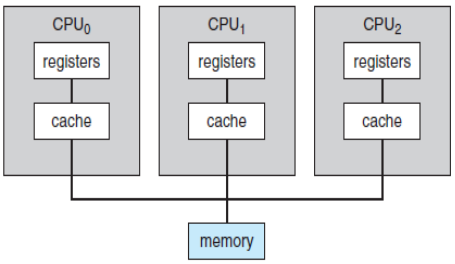
Parallel Systems

- Also known as **parallel systems** or **multicore systems.**
- **Definition:** Systems having **two** or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.
- **Advantages**
  - **Increased throughput.** By increasing the number of processors, we expect to get more work done in less time. When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead, plus contention for shared resources, lowers the expected gain from additional processors.
  - **Economy of scale**. Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and

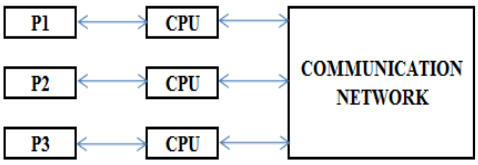
power supplies.

- **Increased reliability**. If functions can be distributed properly among several processors, then the failure of one processor will not halt them system, only slow it down. If we have ten processors and one fails, then each of the remaining nine processors can pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether. Increased reliability of a computer system is crucial in many applications. The ability to continue providing service proportional to the level of surviving hardware is called **graceful degradation**. Some systems go beyond graceful degradation and are called **fault tolerant**, because they can suffer a failure of any single component and still continue operation.

- **Two Types Multiple-Processor Systems**
  - **Asymmetric multiprocessing:** each processor is assigned a specific task. A ***boss*** processor controls the system; the other processors either look to the boss for instruction or have predefined tasks. This scheme defines a boss–worker relationship. The boss processor schedules and allocates work to the worker processors.
  - **Symmetric multiprocessing (SMP)**: each processor performs all tasks within the operating system. SMP means that all processors are peers; no boss–worker relationship exists between processors.



Symmetric multiprocessing architecture.



Asymmetric Multi Processing

Distributed Systems

- A distributed system is a collection of physically separate, possibly heterogeneous, computer systems that are networked to provide users with access to the various resources that the system maintains.
- Access to a shared resource increases computation speed, functionality, data availability, and reliability.
- Distributed systems depend on networking for their functionality.
- Networks vary by the
  - Protocols used : Most OS support TCP/IP
  - Distances between the nodes: LAN, WAN, MAN, PAN
  - Transport media: Copper wires, fiber stands & wireless transmissions
- A **network operating system** is an operating system that provides features such as file sharing across the network, along with a communication scheme that allows different

processes on different computers to exchange messages.

- A distributed operating system provides a less autonomous environment. A distributed operating system provides a less autonomous environment. The different computers communicate closely enough to provide the illusion that only a single operating system controls the network.
- **Reasons for building distributed systems**
  There are four major reasons for building distributed systems,
  - **Resource Sharing :** If a number of different sites (with different capabilities) are connected to one another, then a user at one site may be able to use the resources available at another. For example, a user at site A may be using a laser printer located at site B. Meanwhile, a user at B may access a file that resides at A.
  - **Computation Speedup:** If a particular computation can be partitioned into sub computations that can run concurrently, then a distributed system allows us to distribute the sub computations among the various sites. The sub computations can be run concurrently and thus provide **computation speedup**.
  - **Reliability:** If one site fails in a distributed system, the remaining sites can continue operating, giving the system better reliability.
  - **Communication:** When several sites are connected to one another by a communication network, users at the various sites have the opportunity to exchange information.

Real-Time Systems

- A real time system is a computer system that requires not only that the computing results be correct but also they are produced within specified deadline
- A real-time system is used when rigid time requirements have been placed on the operation of a processor or the flow of data; thus, it is often used as a control device in a dedicated application.
- Sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs.
- Systems that control scientific experiments, medical imaging systems, industrial control systems, and certain display systems are real time systems. Some automobile-engine fuel-injection systems, home-appliance controllers, and weapon systems are also real-time systems.
- A real-time system has well-defined, fixed time constraints. Processing ***must*** be done within the defined constraints, or the system will fail. For instance, it would not do for a robot arm to be instructed to halt ***after*** it had smashed into the car it was building.
- **Types**
  It is of 2 types
  - **Hard real time systems**: Has the most strict requirements, guaranteeing that critical real time tasks be completed within their deadlines. Safety critical systems are typically hard real time systems
  - **Soft real time systems:** Less restrictive simple provides a critical real time task will receive priority over other task and that it will retain that priority until it completes.

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

Operating System Services Those Are Helpful to the User

1. **User interface**.

Almost all operating systems have a **user interface (UI)**. This interface can take several forms.

- One is a **command-line interface (CLI)**, which uses text commands and a method for entering them (say, a keyboard for typing in commands in a specific format with specific options).
- 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/O, choose from menus, and make selections and a keyboard to enter text.
- Some systems provide two or all three of these variations.

2. **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).

3. **I/O operations**.
   - A running program may require I/O, which may involve a file or an I/O 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/O devices directly.
   - Therefore, the operating system must provide a means to do I/O.

4. **File-system manipulation**
   - Programs need to read and write files and directories.
   - Many operating systems provide a variety of file systems, sometimes to allow personal choice and sometimes to provide specific features or performance characteristics.

5. **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 different computer systems using Shared memory or Message passing techniques.

6. **Error detection**.
   - The operating system needs to be detecting and correcting errors constantly.
   - Errors may occur in

- ❖ CPU and memory hardware (such as a memory error or a power failure),
- ❖ I/O devices (such as a parity error on disk, a connection failure on a network, or lack of paper in the printer),
- ❖ 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.

Operating System for Efficient Operation of the System Itself

1. **Resource allocation**.
   - When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.
   - The operating system manages many different types of resources such as CPU cycles, main memory, and file storage.

2. **Accounting**.
   - We want to keep track of 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.

3. **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 starts with requiring each user to authenticate himself or herself to the system, usually by means of a password, to gain access to system resources. It extends to defending external I/O devices.

System Calls

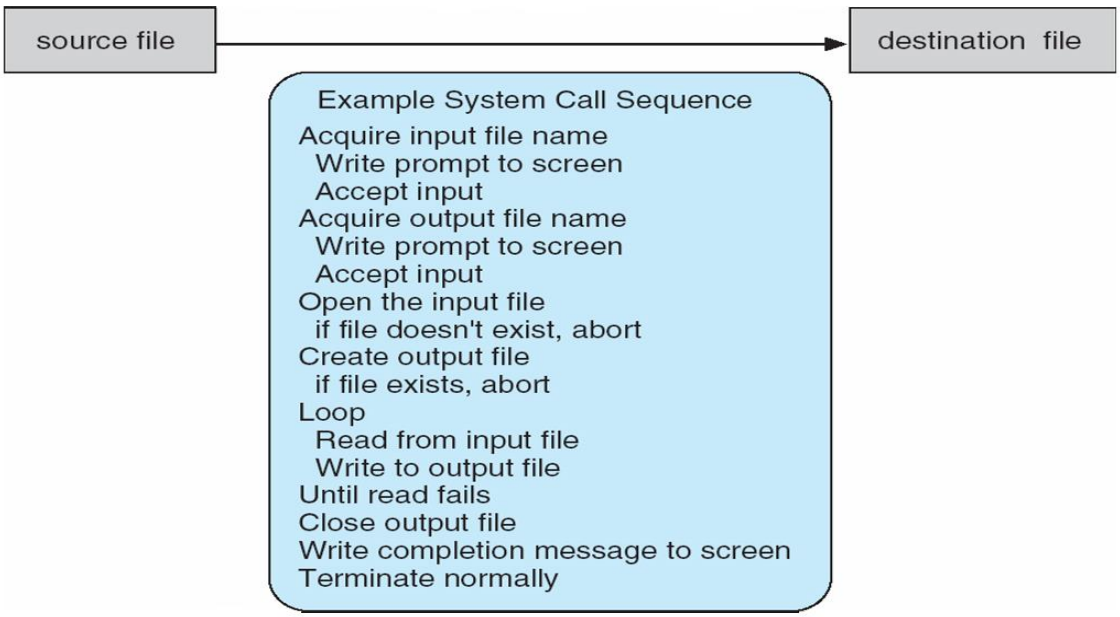
Definition: System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++, although certain low-level tasks (for example, tasks where hardware must be accessed directly) may have to be written using assembly-language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

Example To Illustrate How System Calls Are Used: Writing A Simple Program To Read Data From One File And Copy Them To Another File.

- The first input that the program will need is the names of the two files: the input file and the output file.
- These names can be specified in many ways, depending on the operating-system design. One approach is for the program to ask the user for the names.
- In an interactive system, this approach will require a sequence of system calls, first to write a prompting message on the screen and then to read from the keyboard the

characters that define the two files. On mouse-based and icon-based systems, a menu of file names is usually displayed in a window.

- The user can then use the mouse to select the source name, and a window can be opened for the destination name to be specified. This sequence requires many I/O system calls.
- Once the two file names have been obtained, the program must open the input file and create the output file. Each of these operations requires another system call.
- Possible error conditions for each operation can require additional system calls. When the program tries to open the input file, for example, 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 (yet 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.
- When 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 (for example, no more disk space).
- 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).

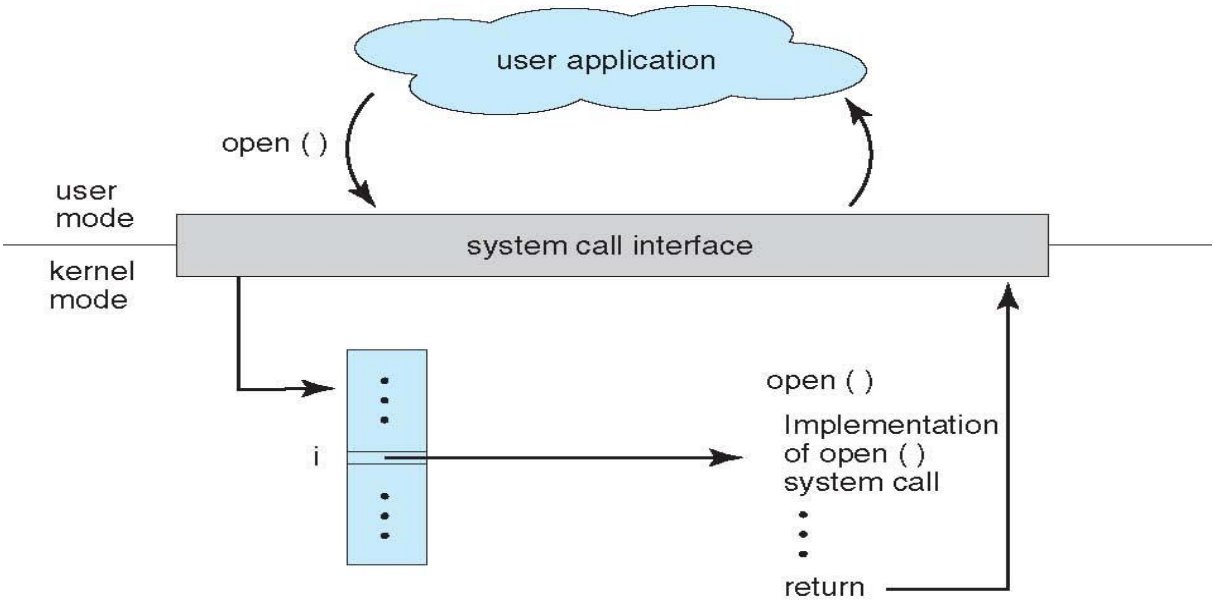


Application Programming Interface (API)

- 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 the Windows API for Windows systems, the POSIX API for POSIX-based systems (which include virtually all versions of UNIX, Linux, and Mac OSX), and the Java API for programs that run on the Java virtual machine.
- The functions that make up an API typically invoke the actual system calls on behalf of the application programmer.

System call Interface

- For most programming languages, the run-time support system (a set of functions built into libraries included with a compiler) 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 caller need know nothing about how the system call is implemented or what it does during execution. Rather, the caller need only obey the API and understand what the operating system will do as a result of the execution of that system call.
- Thus, 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.



Types of System Calls

System calls can be grouped roughly into six major categories:

1. **Process control**

a. end, abort

- A running program needs to be able to halt its execution either normally (end ()) or abnormally (abort ()).
- When a running program terminates abnormally, a dump is written to the disk and examined by- a system program designed to aid the programmer in finding and correcting errors, or **bugs**—to determine the cause of the problem.
- In Command line interfaces, it is 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. Some systems use Control cards which determine severity of errors and determine which action to be taken.

b. load, execute

- 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.

c. create process, terminate process

- An interesting question is where to return control when the loaded program terminates. This question is related to whether the existing program is lost, saved, or allowed to continue execution concurrently with the new program.
- If control returns to the existing program when the new program terminates, we must save the memory image of the existing program; thus, we have effectively created a mechanism for one program to call another program.
- If both programs continue concurrently, we have created a new job or process to be multiprogrammed. The system call for this purpose is create process or submit job.
- We may also want to terminate a job or process that we created (terminate process) if we find that it is incorrect or is no longer needed.

d. get process attributes, set process attributes

- If we create a new job or process, or perhaps even a set of jobs or processes, we should be able to control its execution.
- This control requires the ability to determine and reset the attributes of a job or process, including the job's priority, its maximum allowable execution time, and so on using get process attributes() and set process attributes().

e. wait for time

f. wait event, signal event

- Having created new jobs or processes, we may need to wait for them to finish their execution. We may want to wait for a certain amount of time to pass (wait time ()). More probably, we will want to wait for a specific event to occur (wait event ()).
- The jobs or processes should then signal when that event has occurred (signal event ()).

g. allocate and free memory

- Most OS allocate memory for programs and releases it after completion.

2. File management

a. create file, delete file

b. open, close

c. read, write, reposition

d. get file attributes, set file attributes

- We 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, we need to open () it and to use it.
- We may also read (), write (), or reposition () (rewind or skip to the end of the file, for example).
- Finally, we need to close () the file, indicating that we are no longer using it.
- 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 attributes () and set file attributes (), are required for this function.
- Some operating systems provide many more calls, such as calls for file move () and copy().

3. Device management

a. request device, release device

b. read, write, reposition

c. get device attributes, set device attributes

d. logically attach or detach devices

- 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 us to first request () a 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 to us), we can read(), write(), and (possibly) reposition() the device,
- Many OS including UNIX merge file and devices structures into one and use same set of system calls

4. Information maintenance

a. get time or date, set time or date

b. get system data, set system data

c. get process, file, or device attributes

d. set process, file, or device attributes

- 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.
- 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 ()).

5. **Communications**
   a. **create, delete communication connection**
   b. **send, receive messages**
   c. **transfer status information**
   d. **attach or detach remote devices**

There are two common models of inter process communication:

a) The message passing model

b) The shared-memory model.

a) The Message Passing Model

- In the **message-passing model**, the communicating 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.
- 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 get hostid () and get processid () system calls do this translation. The identifiers are then passed to the general purpose open () and close () calls provided by the file system or to specific open connection () and close connection () system calls, depending on the system's model of communication.
- The recipient process usually must give its permission for communication to take place with an accept connection () call. Most processes that will be receiving connections are special-purpose **daemons**, which are system programs provided for that purpose.
- They execute a wait for connection () call and are awakened when a connection is made.
- 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. The close connection () call terminates the communication.

b) Shared-Memory Model

- 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.

- 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.

Comparison of both the models

- Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided. It is also easier to implement than is shared memory for inter computer communication.
- Shared memory allows maximum speed and convenience of communication, since it can be done at memory transfer speeds when it takes place within a computer. Problems exist, however, in the areas of protection and synchronization between the processes sharing memory.

6. Protection

a. set permission, get permission

b. allow user, deny user

- 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.