UNIT 1

SYLLABUS

Operating System Principles: what Operating Systems do, Computer System organization, computer system architecture, Operating System structure, Computing environments, Operating System Services, User - Operating System interface, System calls and system programs, Operating System structure.

1.1 What is an Operating System?

- An Operating System (OS) acts as an intermediary between the user of a computer and the computer hardware
- An operating system is a software that manages the computer hardware.
- OS allows the user to execute programs in a convenient and efficient manner.
- An Operating System provides the environment within which programs are executed.

Operating system goals:

- ✓ Make the computer system convenient to use. It hides the difficulty in managing the hardware.
- ✓ Use the computer hardware in an efficient manner.
- ✓ Provide the environment in which user can easily interface with computer.
- ✓ It acts as a **resource allocator.**

Components of Computer System (Computer System Structure)

- Computer system mainly consists of four components as shown in figure 1:
  1. ***Hardware****:* Provides the basic computing resources for the system. It includes CPU, memory, input/output (I/O) devices.
  2. ***Operating System****:* Controls and coordinates the use of hardware among the various applications and users
  3. ***Application Programs****:* Define the ways in which the system resources are used to solve the computing problems of the users. It includes word processors, spreadsheets, compilers, web browsers etc.
  4. ***Users****:* Can be people, machines, other components

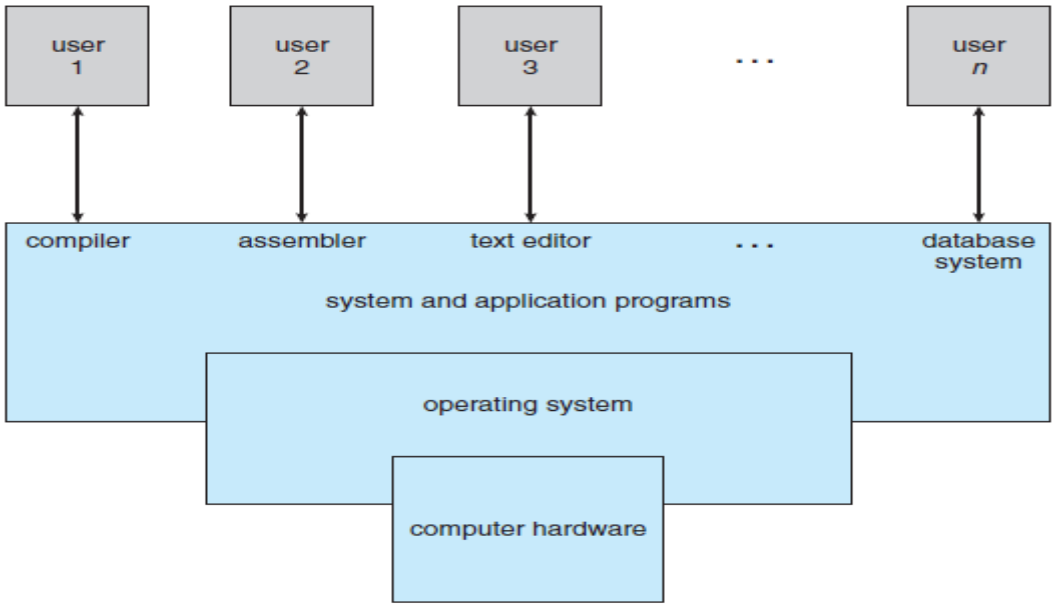

Figure 1: Abstract view of the components of a computer system

- The basic hardware components comprise of CPU, memory, I/O devices. The application program uses these components. The OS controls and co-ordinates the use of hardware, among various application programs (like compiler, word processor etc.) for various users.
- The OS allocates the resources among the programs such that the hardware is efficiently used.
- The operating system is the program running at all the times on the computer. It is usually called as the ***kernel***.

Views of OS

Operating System can be viewed from two viewpoints: *User view* and *System view*

1 **User View:** The user's view of the OS depends on the type of the user

- ✓ If the user is using **standalone** system, then OS is designed for *ease of use* and *high performances*. Here resource utilization is not given importance.
- ✓ If the users are at different **terminals** connected to a mainframe or minicomputers, by sharing information and resources, then the OS is designed to *maximize resource utilization*. OS is designed such that the CPU time, memory and I/O are used efficiently and no single user takes more than the resource allotted to them
- ✓ If the users are in **workstations**, connected to networks and servers, then the user have a dedicated resources but also share resources such as networking and servers-files, print servers. Here the OS is designed for both *ease of use* and *resource availability (files)*.

- ✓ Users of **hand held** systems, expects the OS to be designed for *ease of use* and *performance* per amount of battery life.
- ✓ Other systems like embedded systems used in home devices (like washing m/c) and automobiles do not have any user interaction. They are connected to network either directly by wire or through wireless modems and networking.

2 **System View:** OS can be viewed as a **resource allocator** and **control program.**

- ✓ **Resource allocator:** The OS acts as a manager of hardware and software resources.(CPU time, memory space, file-storage space, I/O devices, shared files etc. are the different resources required during execution of a program) There can be conflicting request for these resources by different programs running in same system. The OS assigns the resources to the requesting program depending on the priority. *Resource allocation is important when many users access the same computer.*
- ✓ **Control Program:** The OS is a control program and manage the execution of user program to prevent errors and improper use of the computer. It is concerned with the operation and control of I/O devices.

1.2 Operating System Organization

Computer System Operation

- 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.
- Each device controller is in charge of a specific type of device (for example, disk drives, audio devices, or video displays). Figure 2

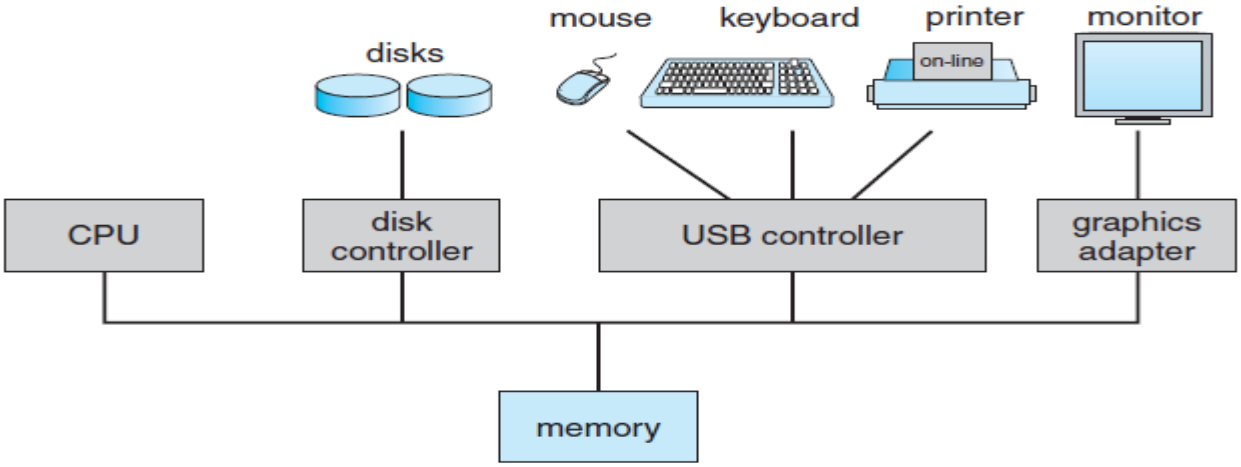

Figure 2: Computer Components

- The CPU and the device controllers can execute in parallel, competing for memory cycles. To ensure orderly access to the shared memory, a memory controller synchronizes access to the memory.

- 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 within the computer hardware in read-only memory (**ROM**) or electrically erasable programmable read-only memory (**EEPROM**).
- The bootstrap program must know how to load the operating system and how to start executing that system.
- On UNIX, the first system process is **"init,"** and it starts many other daemons. Once this phase is complete, the system is fully booted, and the system 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.**
- 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.

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 rewritable 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)**.
- Ideally, programs and data are to be resided in main memory permanently. This arrangement usually is not possible for the following two reasons:
  1. Main memory is usually too small to store all needed programs and data permanently.
  2. 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.
- The wide variety of storage systems can be organized in a hierarchy (Figure 1.3) 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.
- The top four levels of memory in Figure 3 may be constructed using semiconductor memory.

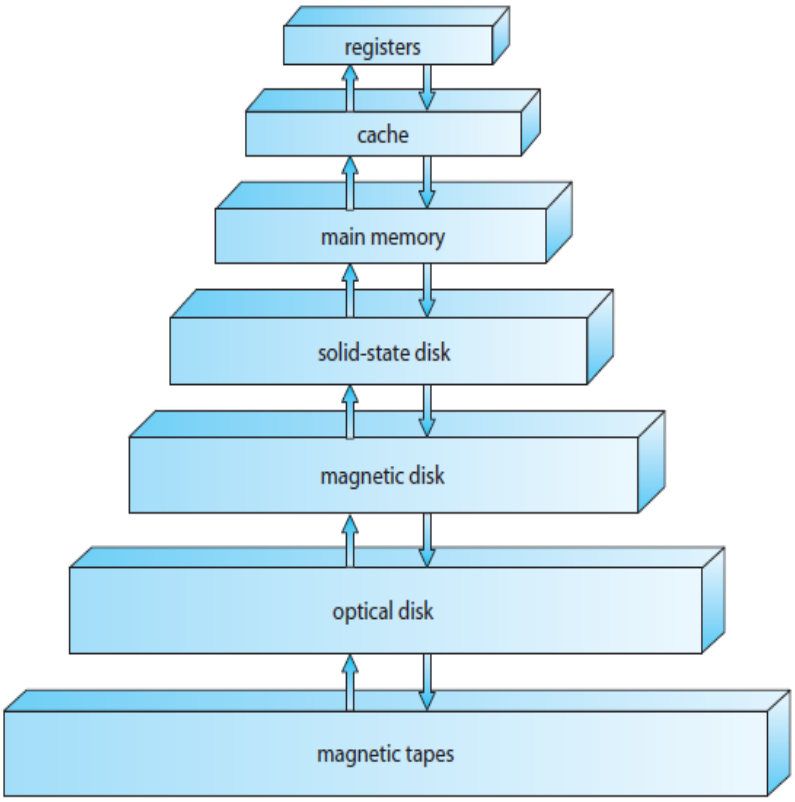

Figure 3: Storage Device Hierarchy

- **Volatile storage** loses its contents when the power to the device is removed. In the absence of expensive battery and generator backup systems, data must be written to **non-volatile storage** for safekeeping.
- In the hierarchy shown in Figure 1.3, the storage systems above the solid-state disk are volatile, whereas those including the solid-state disk and below are non-volatile.

I/O 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.
- Depending on the controller, more than one device may be attached.
- 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.
- 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.

1.3 Computer System Architecture

- A computer system can be organized in a number of different ways, which we can categorize roughly according to the number of general-purpose processors used.

Single Processor Systems

- On a single processor system, there is one main CPU capable of executing a general-purpose instruction set, including instructions from user processes.
- Almost all single processor systems have other special-purpose processors as well. They may come in the form of device-specific processors, such as disk, keyboard, and graphics controllers; or, on mainframes, they may come in the form of more general-purpose processors, such as I/O processors that move data rapidly among the components of the system.
- All of these special-purpose processors run a limited instruction set and do not run user processes. Sometimes, they are managed by the operating system, in that the operating system sends them information about their next task and monitors their status.

Multiprocessor Systems(Parallel systems or Tightly coupled systems)

- These systems have two or more *processors* in close communication, sharing the computer bus, the clock, memory, and peripheral devices.
- Multiprocessor systems have three main advantages:
  - *Increased Throughput*: Throughput is defined as number of processes computed per unit time. By increasing the number of processors, the more work can be done in less time. In multiprocessor system, execution of different programs take place simultaneously. Even if the number of processors is increased the performance cannot be simultaneously increased. This is due to the overhead incurred in keeping all the parts working correctly. The speed-up ratio with *N* processors is not *N,* rather, it is less than *N*. Thus the speed of the system is not has expected.
  - *Economy of Scale:* Multiprocessor systems can cost less than equivalent multiple single-processor systems. As the multiprocessor systems share peripherals, mass storage, and power supplies, the cost of implementing this system is economical. If several processes are working on the same data, the data can also be shared among them.
  - *Increased reliability:* In multiprocessor systems, functions are shared among several processors. If one processor fails, the system is not halted, it only slows

down. The job of the failed processor is taken up, by other processors. This ability to continue to operate in spite of failure makes the system *fault tolerant.* Fault tolerance requires a mechanism to allow the failure to be detected, diagnosed, and, if possible, corrected.

There are two types of multiprocessors systems

i **Asymmetric Multiprocessing (Master-Slave Architecture)** : Here each processor is assigned a specific task, by the master processor. A master processor controls the other processors in the system. It schedules and allocates work to the slave processors. This scheme defines *master-slave relationship.*
Ex- Sun's Operating system: SUNOS version 4

ii **Symmetric multiprocessing (SMP):** The processors are considered as peers; no master-slave relationship exists between processors. Each processor performs all tasks within the OS. All the processors have its own registers and CPU, only memory is shared as shown in figure 4

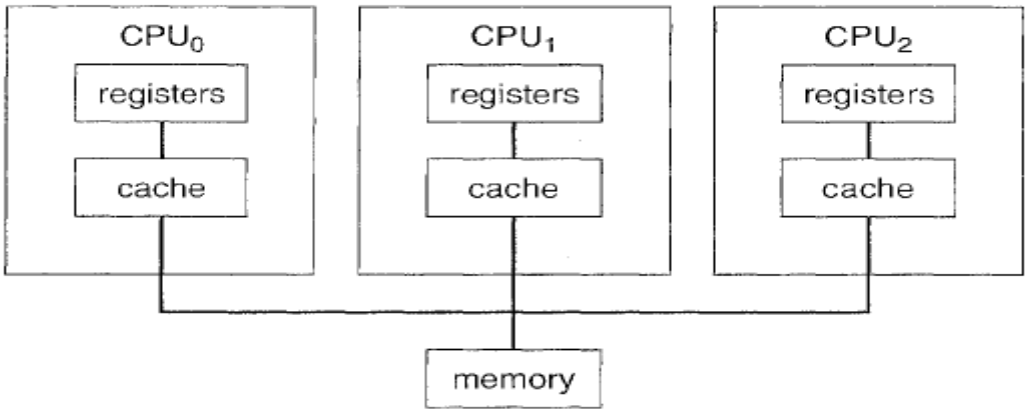

Figure 4: Symmetric Multiprocessing Architecture

The benefit of this model is that many processes can run simultaneously. *N* processes can run if there are *N* CPUs—without causing a significant deterioration of performance. Operating systems like Windows, Windows XP, Mac OS X, and Linux—now provide support for SMP.

Clustered Systems

- Another type of multiprocessor system is a **clustered system**, which gathers together multiple CPUs.
- Clustered systems differ from the multiprocessor systems in that they are composed of two or more individual systems—or nodes—joined together. Such systems are considered **loosely coupled**.
- Each node may be a single processor system or a multicore system.
- Clustering is usually used to provide **high-availability** service—that is, service will continue even if one or more systems in the cluster fail.
- Clustering can be structured asymmetrically or symmetrically.

- In **asymmetric clustering**, one machine is in **hot-standby mode** while the other is running the applications. The hot-standby host machine does nothing but monitor the active server. If that server fails, the hot-standby host becomes the active server.
- In **symmetric clustering**, two or more hosts are running applications and are monitoring each other. This structure is obviously more efficient, as it uses all of the available hardware.
- Figure 5 depicts the general structure of a clustered system.

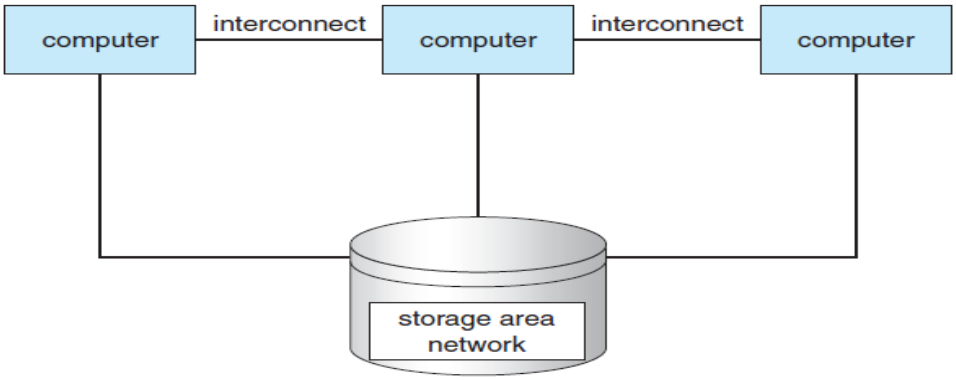

Figure 5: General structure of a clustered system

1.4 Operating System Structure

✓ OS structure must be carefully designed. The task of OS is divided into small components and then interfaced to work together.

1) Simple Structure

- Many operating systems do not have well-defined structures. They started as small, simple, and limited systems and then grew beyond their original scope. Eg: MS-DOS.
- In MS-DOS, the interfaces and levels of functionality are not well separated as shown in Figure 6.
- Application programs can access basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS in bad state and the entire system can crash down when user programs fail.
- *UNIX OS consists of two separable parts*: the *kernel* and the *system programs*. The kernel is further separated into a series of interfaces and device drivers.

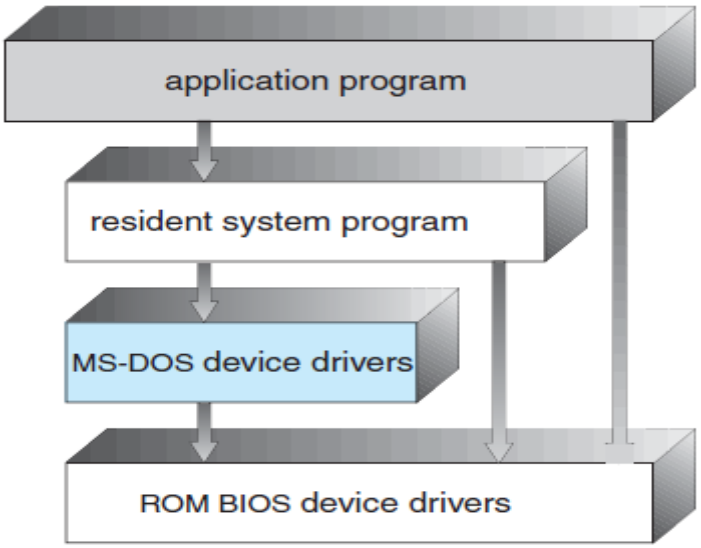

Figure 6 : MS-DOS Layer Structure

- But the traditional UNIX operating System is layered as shown in Figure 7.
- Everything below the system-call interface and above the physical hardware is the kernel.
- The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls.

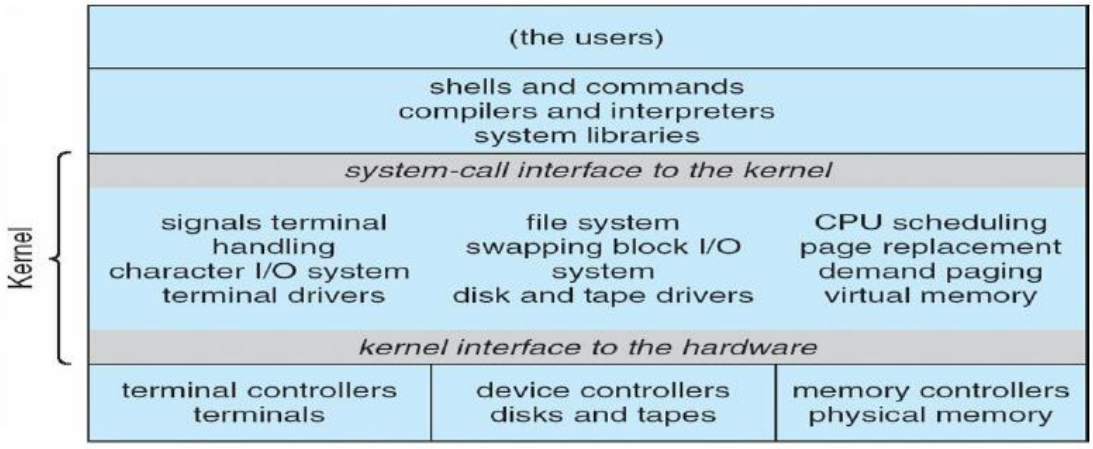

Figure 7: Traditional UNIX system structure

2) Layered Approach

- ✓ The OS is broken into number of layers (levels). Each layer rests on the layer below it, and relies on the services provided by the next lower layer.
- ✓ Bottom layer(layer 0) is the hardware and the topmost layer (layer N) is the user interface as shown in Figure 8.
- ✓ A typical layer, consists of data structure and routines that can be invoked by higher-level layer.

- ✓ Under a top-down approach, the overall functionality and features are determined and are separated into components.
- ✓ A OS layer is an implementation of an abstract object made up of *data* and the *operations* that can manipulate those data

- ***Advantages:***
  - ➢ Simplicity of construction and debugging
  - ➢ The layers are selected such that each uses functions and services of only lower-level layers. Hence simplifies debugging and system verification i.e., the first layer can be debugged without concerning the rest of the system. Once the first layer is debugged, its correct functioning is assumed while the 2nd layer is debugged & 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 & implementation of the system are simplified when the system is broken down into layers.
  - ➢ A layer doesn‘t need to know how these operations are implemented; it only needs to know what these operations do.

- ***Disadvantages:***
  - ➢ The various layers must be appropriately defined, as a layer can use only lower level layers.
  - ➢ Less efficient than other types, because any interaction with layer 0 required from top layer. The system call should pass through all the layers and finally to layer 0. This is an overhead.

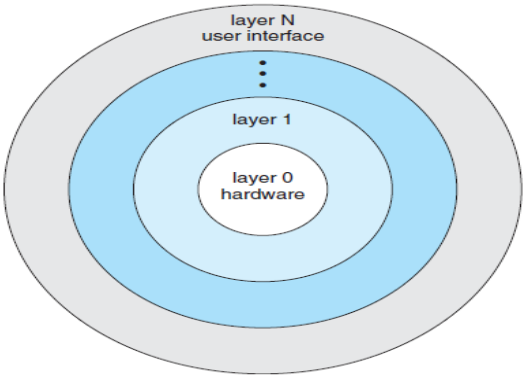

Figure 8: A Layered OS

3) Microkernels

- The basic idea behind micro kernels is to *remove all non-essential services* from the kernel, thus making the kernel as small and efficient as possible.

- The removed services are implemented as system applications(user space)
- Most microkernels provide basic process and memory management, and message passing between other services.
- Figure 9 illustrates the architecture of a typical microkernel.
- ***Benefit***: To provide *communication facility* between the client program and the various services that are running in user space. Communication is provided by *message passing.*
- Mach was the first and most widely known microkernel, and now forms a major component of Mac OS X.

- ***Advantages:***
  - System expansion can be easier, because it only involves adding more system applications, not rebuilding a new kernel.
  - When the kernel does not have to be modified, the changes are fewer, because the microkernel is a smaller kernel.
  - Hence microkernel is smaller kernel.
  - The microkernel OS is easier to port from one hardware design to another.
  - Provides more security and reliability, since most services are running as user programs rather than kernel processes.

- ***Disadvantages:***
  - Performance is reduced due to increased system function overhead.

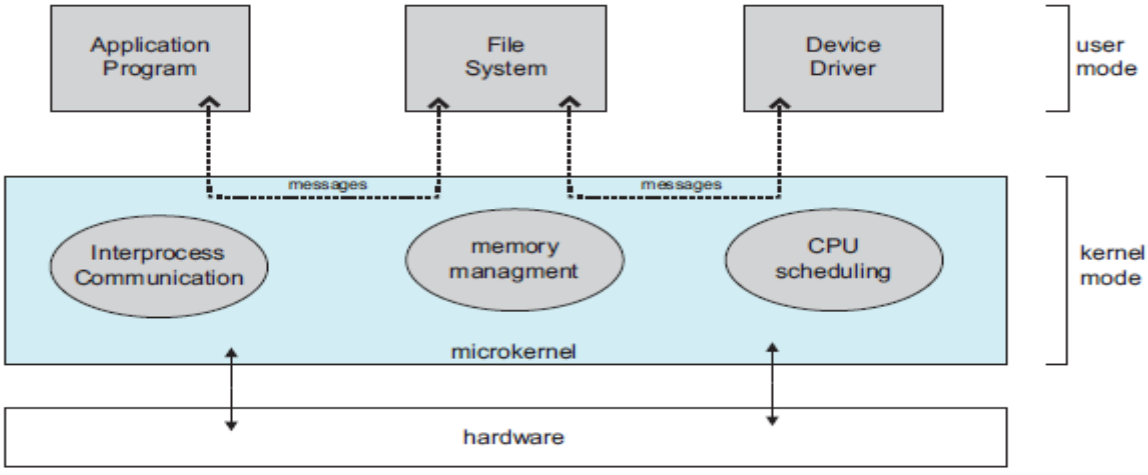

Figure 9: Architecture of a typical microkernel.

4) Modules

- ✓ Modern OS development is *object-oriented*, with a relatively small core kernel and a set of ***modules*** which can be linked in dynamically.
- ✓ For Example: The Solaris OS structure shown in figure 10 is organized around a core kernel with 7 types of loadable kernel modules:
  - Scheduling Classes

- File Systems
- Loadable System calls
- Executable formats
- STREAMS modules
- Miscellaneous
- Device and bus drivers

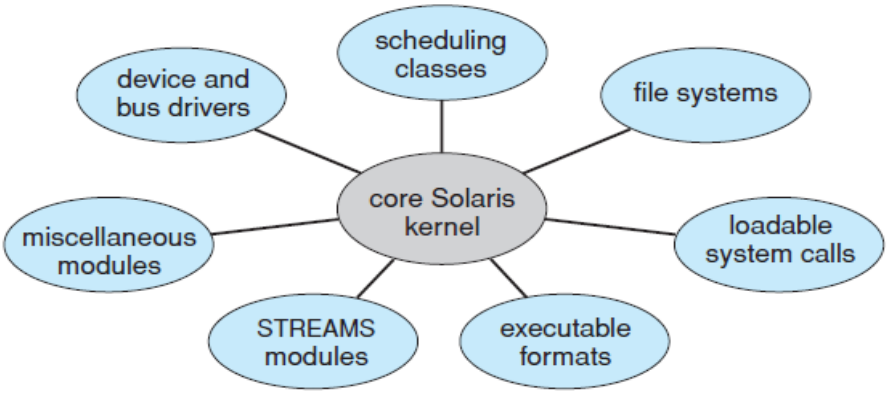

Figure 10: Solaris Loadable Modules

- ✓ Modules are similar to *Layered systems* in that each subsystem has clearly defined tasks and interfaces, but any module is free to contact any other module, eliminating the problems of going through multiple intermediary layers
- ✓ The kernel is relatively small in this architecture, similar to *microkernels*, but the kernel does not have to implement message passing since modules are free to contact each other directly. Eg: Solaris, Linux and Mac OS X
- ✓ The Apple Mac OS X Operating system uses a hybrid structure. It is a layered system in which one layer consists of the Mach microkernel.

1.5 Computing Environments

Traditional Computing

- Consider the "typical office environment." Just a few years ago, this environment consisted of PCs connected to a network, with servers providing file and print services.
- Terminals attached to mainframes were prevalent at many companies as well, with even fewer remote access and portability options.
- Web technologies and increasing WAN bandwidth are stretching the boundaries of traditional computing. Companies establish **portals**, which provide Web accessibility to their internal servers. **Network computers** (or **thin clients**)—which are essentially terminals that understand web-based computing—are used in place of traditional workstations where more security or easier maintenance is desired.
- At home, most users once had a single computer with a slow modem connection to the office, the Internet, or both.
- Many homes use **firewalls** to protect their networks from security breaches.

Mobile Computing

- **Mobile computing** refers to computing on handheld smartphones and tablet computers. These devices share the distinguishing physical features of being portable and lightweight.
- Today, mobile systems are used not only for e-mail and web browsing but also for playing music and video, reading digital books, taking photos, and recording high-definition video.
- Many developers are now designing applications that take advantage of the unique features of mobile devices, such as global positioning system (GPS) chips, accelerometers, and gyroscopes.
- An embedded GPS chip allows a mobile device to use satellites to determine its precise location on earth.
- An accelerometer allows a mobile device to detect its orientation with respect to the ground and to detect certain other forces, such as tilting and shaking. In several computer games that employ accelerometers, players interface with the system not by using a mouse or a keyboard but rather by tilting, rotating, and shaking the mobile device.
- To provide access to on-line services, mobile devices typically use either IEEE standard 802.11 wireless or cellular data networks.
- The memory capacity and processing speed of mobile devices, however, are more limited than those of PCs.
- Two operating systems currently dominate mobile computing: **Apple iOS** and **Google Android**. iOS was designed to run on Apple iPhone and iPad mobile devices.

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.
- A **network**, in the simplest terms, is a communication path between two or more systems. Distributed systems depend on networking for their functionality. Networks vary by the protocols used, the distances between nodes, and the transport media. **TCP/IP** is the most common network protocol, and it provides the fundamental architecture of the Internet.
- To an operating system, a network protocol simply needs an interface device—a network adapter, for example—with a device driver to manage it.
- Networks are characterized based on the distances between their nodes.
- A **local-area network (LAN)** connects computers within a room, a building, or a campus.
- A **wide-area network (WAN)** usually links buildings, cities, or countries.
- A **metropolitan-area network (MAN)** could link buildings within a city.
- Bluetooth and 802.11 devices use wireless technology to communicate over a distance of several feet, in essence creating a **personal-area network (PAN)** between a phone and a headset or a smartphone and a desktop computer.

- 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.

Client-Server Computing

- As PCs have become faster, more powerful, and cheaper, designers have shifted away from centralized system architecture.
- Figure 11 depicts the General structure of a client–server system.
- Many of today's systems act as **server systems** to satisfy requests generated by **client systems.**
- Server systems can be broadly categorized as compute servers and file servers:
- The **compute-server system** provides an interface to which a client can send a request to perform an action (for example, read data). In response, the server executes the action and sends the results to the client.
- The **file-server system** provides a file-system interface where clients can create, update, read, and delete files.

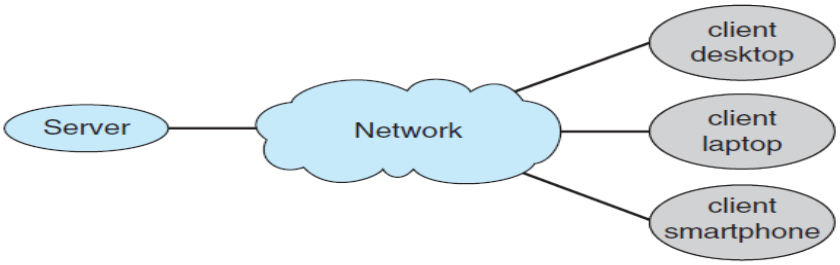

Figure 11: General structure of a client–server system

Peer-to-Peer Computing

- Another structure for a distributed system is the peer-to-peer (P2P) system model.
- In this model, clients and servers are not distinguished from one another.
- Instead, all nodes within the system are considered peers, and each may act as either a client or a server, depending on whether it is requesting or providing a service.
- Peer-to-peer systems offer an advantage over traditional client-server systems.
- To participate in a peer-to-peer system, a node must first join the network of peers. Once a node has joined the network, it can begin providing services to—and requesting services from—other nodes in the network.
- Determining what services are available is accomplished in one of two general ways:
  - ✓ When a node joins a network, it registers its service with a centralized lookup service on the network. Any node desiring a specific service first contacts this centralized lookup service to determine which node provides the service. The remainder of the communication takes place between the client and the service provider.
  - ✓ An alternative scheme uses no centralized lookup service. Instead, a peer acting as a client must discover what node provides a desired service by broadcasting a request for the service to all other nodes in the network as shown in figure 12.

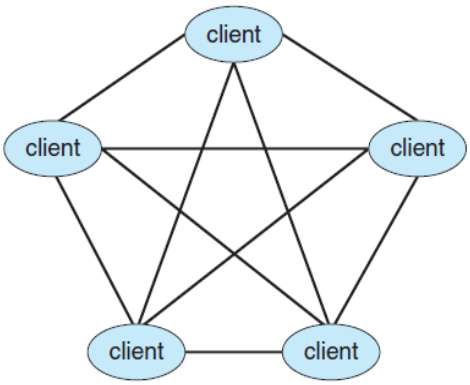

Figure 12: Peer-to-peer system with no centralized service

Virtualization

- Virtualization is a technology that allows operating systems to run as applications within other operating systems.
- Virtualization is one member of a class of software that also includes emulation. **Emulation** is used when the source CPU type is different from the target CPU type.
- Every machine-level instruction that runs natively on the source system must be translated to the equivalent function on the target system, frequently resulting in several target instructions.
- Running multiple virtual machines allowed (and still allows) many users to run tasks on a system designed for a single user.
- With Windows being the **host** operating system, and the VMware application was the virtual machine manager VMM. The VMM runs the guest operating systems, manages their resource use, and protects each guest from the others.
- Even though modern operating systems are fully capable of running multiple applications reliably, the use of virtualization continues to grow. On laptops and desktops, a VMM allows the user to install multiple operating systems for exploration or to run applications written for operating systems other than the native host. For example, an Apple laptop running Mac OS X on the x86 CPU can run a Windows guest to allow execution of Windows applications.

Cloud Computing

- **Cloud computing** is a type of computing that delivers computing, storage, and even applications as a service across a network. In some ways, it's a logical extension of virtualization, because it uses virtualization as a base for its functionality.
- There are three types of cloud computing:
  - ✓ **Public cloud**—a cloud available via the Internet to anyone willing to pay for the services.
  - ✓ **Private cloud**—a cloud run by a company for that company's own use.
  - ✓ **Hybrid cloud**—a cloud that includes both public and private cloud components.
  - ✓ Software as a service **(SaaS)**—one or more applications (such as word processors or spreadsheets) available via the Internet
  - ✓ Platform as a service **(PaaS)**—a software stack ready for application use via the Internet (for example, a database server)

- ✓ Infrastructure as a service **(IaaS)**—servers or storage available over the Internet (for example, storage available for making backup copies of production data)

Real-Time Embedded Systems

- Embedded computers are the most prevalent form of computers in existence.
- These devices are found everywhere, from car engines and manufacturing robots to DVDs and microwave ovens. They tend to have very specific tasks.
- The systems they run on are usually primitive, and so the operating systems provide limited features.
- Usually, they have little or no user interface, preferring to spend their time monitoring and managing hardware devices, such as automobile engines and robotic arms.
- Embedded systems almost always run **real-time operating systems**. 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 analyse the data and possibly adjust controls to modify the sensor inputs.
- A real-time system has well-defined, fixed time constraints. Processing ***must*** be done within the defined constraints, or the system will fail.

1.6 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.
- The specific services provided, differ from one OS to another.
- The figure 13 shows one view of the OS service.

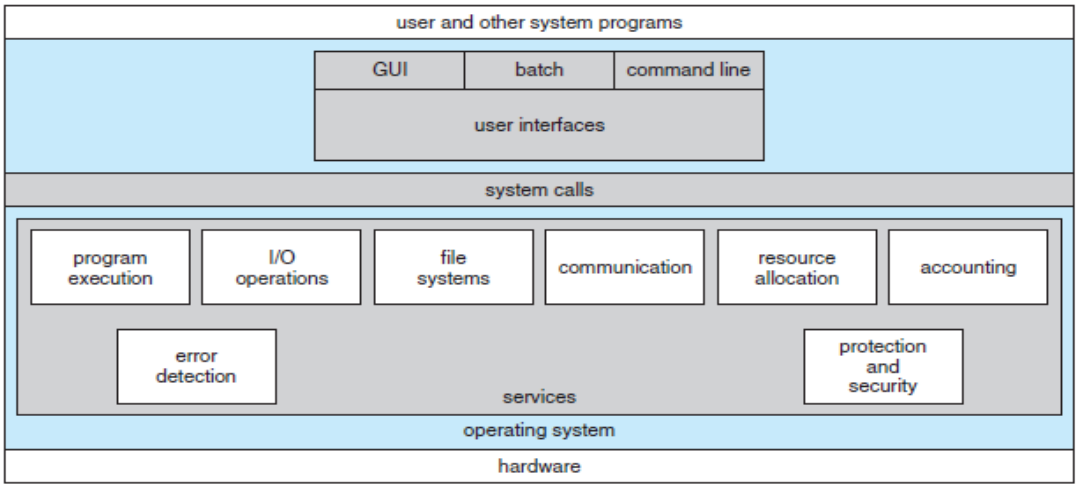

Figure 13: A view of OS service

✓ OS provide services (functions) that are helpful to the user, which includes:

- **User Interface**: Means by which users can issue commands to the system. Depending on the operating system these may be a *command-line interface (CLI)*, a *Graphical User Interface (GUI)* or a *Batch Command Interface*.
  In Command Line Interface(CLI)- commands are given to the system. In Batch interface-commands and directives to control these commands are entered in a file and then the file is executed. In GUI systems- windows with pointing device to get inputs and keyboard to enter the text.
- **Program Execution:** The OS must be able to load a program into memory, run the program, and terminate the program, either normally or abnormally.
- **I/O Operations:** The OS is responsible for transferring the data to and from I/O devices, including keyboards, terminals, printers, and files. For specific devices, special functions (device drivers) are provided by OS.
- **File-System Manipulation:** Programs need to read and write files or directories. The services required to create or delete files, search for a file, list the contents of a file and change the file permissions are provided by OS.
- **Communications:** Communication may occur when one process needs to exchange information with another process. Such communication between the processes may takes place on the same computer or between processes that are executing on different computer systems. Communication may be implemented by OS via *shared memory* or through *message passing.*
- **Error Detection:** Both hardware and software errors must be detected and handled appropriately by the OS. Errors may occur in the *CPU and memory hardware* (such as power failure and memory error), in *I/O 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).

✓ OS also provide services (functions) for the efficient operation of the system, which includes:

- **Resource Allocation:** Resources like CPU cycles, main memory, storage space, and I/O devices must be allocated to multiple users and multiple jobs at the same time. The OS provide CPU-scheduling routines that take into account the speed of the CPU, the jobs that must be executed, the number of registers available etc.
- **Accounting:** There are services in OS to keep 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:** The owners of information (file) in multiuser or networked computer system may want to control the use of that information. When several separate processes execute concurrently, one process should not interfere with other or with OS.

Protection involves ensuring that all access to system resources is controlled. *Security* of the system from outsiders must also be done, by means of a password in order to authenticate the user.

1.7 System Calls

- System calls provide an interface to access the services of the operating system.
- Generally written in C or C++, although some are written in assembly-language for optimal performance.
- The below figure 14 illustrates the sequence of system calls required to copy a file content from one file(input file) to another file (output file).

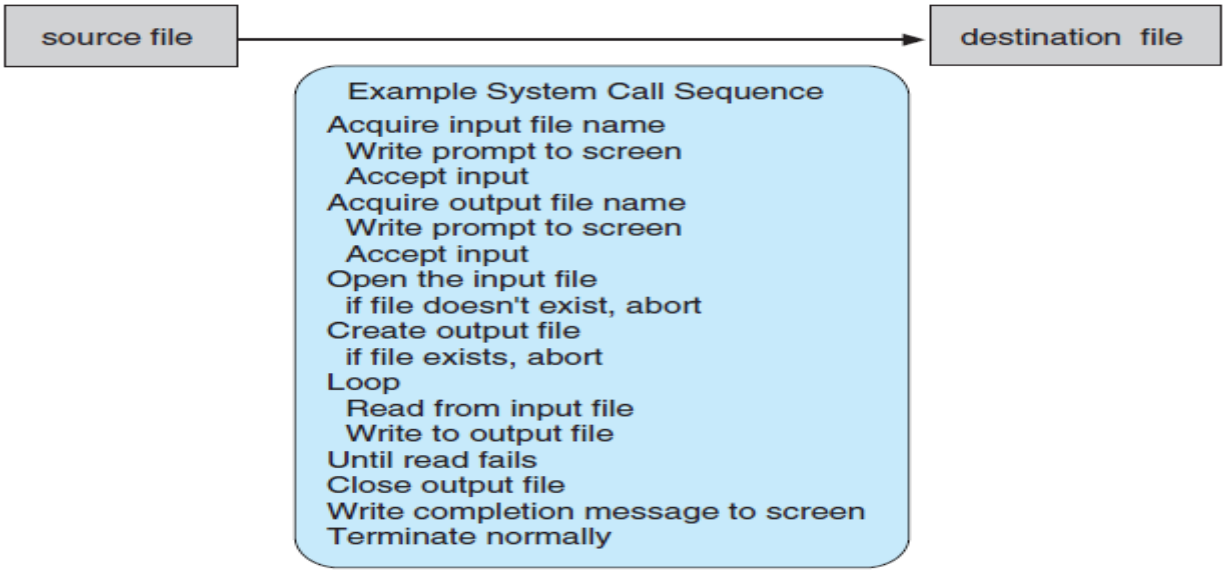

Figure 14: Example of how system calls are used

- For this, the program need names of two files: input file and output file. These names can be provided in many ways.
- **One approach**: In *interactive system*, a sequence of system calls are required. First, to write a message on the screen and then to accept the input filename from keyboard
- **Second approach**: In *mouse-based and icon-based systems*, a menu of filenames is displayed on the screen. Then the user can use the mouse to select the source name and the destination name
- 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 system call) and then terminate abnormally (another system call).
- If input file exists, then create a new output file (another system call). In this situation, we may find that there is already an output file with the same name,

which cause the program to abort (a system call), or may delete the existing file (another system call) and create a new one (another system call)

- Now that both the files are opened, we enter a *loop* that reads from the input file(another system call) and writes to output file (another system call).
- Finally, after the entire file is copied, the program may close both files (another system call), write a message to the console or window(system call), and finally terminate normally (final system call).
- Most programmers do not use the low-level system calls directly, but instead use 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.
- The APIs instead of direct system calls provides for greater program *portability* between different systems. The API then makes the appropriate system calls through the system call interface, using a system call table to access specific numbered system calls, as shown in Figure 15.
- 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 table (consisting of system call number and address of the particular service) invokes a particular service routine for a specific system call.
- The caller need know nothing about how the system call is implemented or what it does during execution.

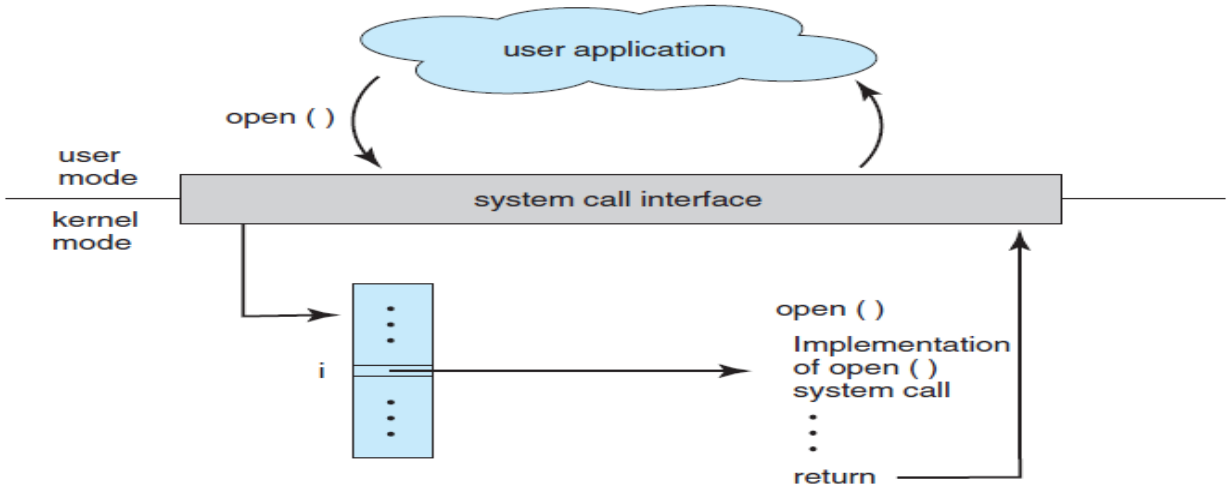

Figure 15: The handling of a user application invoking the open( ) system call

✓ Three general methods used to pass parameters to OS are –

- To pass parameters in *registers*
- Parameters are stored in a blocks, or table, in memory and the address of block is passed as a parameter in a register as shown in Figure 16. (Ex: Linux & Solaris).

- Parameters can be pushed onto the stack by program and popped off the stack by OS.

✓ Some OS prefer the block or stack method because those approaches do not limit the number or length of parameters being used.

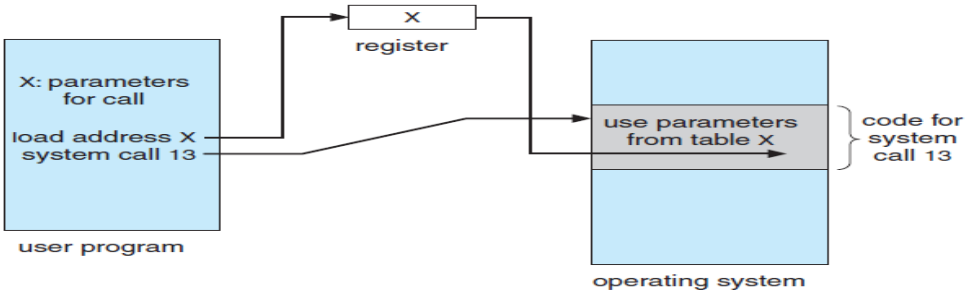

Figure 16: Passing of parameters as a table.

1.8 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.**

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.
- A process or job executing one program may want to **load()** and **execute()** another program.
- If both programs continue concurrently, we have created a new job or process to be multiprogrammed. Often, there is a system call specifically for this purpose (**create_process()** or **submit_job()**).
- 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 (**get_process _attributes()** and **set_process_attributes()**).
- 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()).**

File Management

- 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.
- 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.

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.
- 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, just as we can with files.

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.
- 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 interprocess communication: the message passing model and the shared-memory 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.
- 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.

Protection

- Protection provides a mechanism for controlling access to the resources provided by a computer system.
- 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.

- 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 17: Types of System Calls

1.9 System Programs

- A collection of programs that provide a convenient environment for program development and execution (other than OS) are called ***system programs*** or ***system utilities***.
- It is not a part of Kernel or command interpreters.
- System programs are divided into following categories:

  - ✓ **File Management:** These programs create, delete, copy, rename, print & manipulate files and directories
  - ✓ **Status Information:** Utilities to check on the date, time, number of users, processes running, data logging, etc. System **registries** are used to store and recall configuration information for particular applications.
  - ✓ **File Modification:** Text editors are available to create and modify the contents of file stored on disk. Special commands can also be used to search contents of files or perform text manipulation on files
  - ✓ **Programming-Language Support:** Compliers, assemblers & interpreters are provided to the user with the OS
  - ✓ **Program loading and Execution:** Once a program is assembled or compiled, it must be loaded into memory to be executed. The OS provide absolute loaders, relocatable loaders, linkage editors, overlay loaders etc .
  - ✓ **Communications:** Programs provide the mechanism for creating virtual connections among processors, users, and computer systems. It also includes sending electronic mail messages, to browse web page, remote logins, file transfers, and remote command execution.
  - ✓ **Application Programs:** Most OS provide programs that are useful to solve common problems or to perform common operations. Ex: web browsers, word processors & text formatters etc.

1.10 User and Operating-System Interface

- There are several ways for users to interface with the operating system.
- One provides a command-line interface, or **command interpreter**, 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 Interpreters

- Some operating systems include the command interpreter in the kernel. Others, such as Windows and UNIX, treat the command interpreter as a special program that is running when a job is initiated or when a user first logs on (on interactive systems).
- On systems with multiple command interpreters to choose from, the interpreters are known as **shells.**

- 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.
  - ✓ An alternative approach—used by UNIX, among other operating systems—implements most commands through system programs.

Graphical User Interfaces

- A second strategy for interfacing with the operating system is through a user-friendly graphical user interface, or GUI.
- 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 **icons**, 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.
- Graphical user interfaces first appeared due in part to research taking place in the early 1970s at Xerox PARC research facility. The first GUI appeared on the Xerox Alto computer in 1973.
- Because a mouse is impractical for most mobile systems, smartphones and handheld tablet computers typically use a touchscreen interface. Here, users interact by making **gestures** on the touchscreen—for example, pressing and swiping fingers across the screen.