MODULE 1

CHAPTER 2 – COMPUTER SYSTEM STRUCTURES

SYSTEM STRUCTURE

- A modern, general-purpose computer system consists of a CPU and a number of device controllers that are 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, and video displays).
- The CPU and the device controllers can execute concurrently, competing for memory cycles.
- To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory.

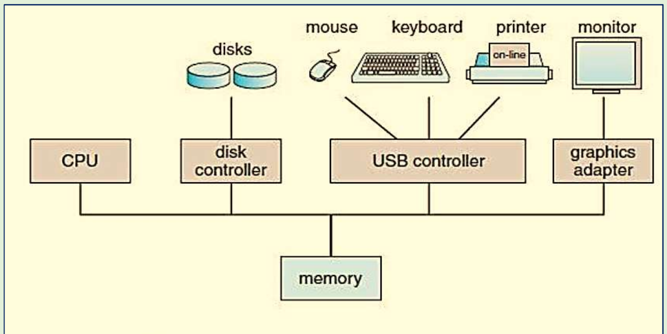


Figure: A modern computer system.

- When a computer is powered up or rebooted, it needs to have an initial program to run. This initial program, or bootstrap program, tends to be simple. Typically, it is stored in read-only memory (ROM) such as firmware or EEPROM within the computer hardware. It initializes all aspects of the system, from CPU registers to device controllers to memory contents.

- The bootstrap program locate and load into memory the operating-system kernel.
- The operating system then starts executing the first process, such as "init“, and waits for some event to occur.
- The occurrence of an event is usually signaled by an interrupt from either the hardware or the software.
- Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus.
- Software may trigger an interrupt by executing a special operation called a system call (also called a monitor call).

- Modern operating systems are interrupt driven. If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen.
- A trap (or an exception) is a software-generated interrupt caused either by an error (for example, division by zero or invalid memory access) or by a specific request
- For each type of interrupt, separate segments of code in the operating system determine what action should be taken.
- An Interrupt Service Routine(ISR) is provided that is responsible for dealing with the interrupt.

- When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location.
- The fixed location usually contains the starting address where the service routine for the interrupt is located.
- The interrupt service routine executes; on completion, the CPU resumes the interrupted computation.
- Each computer design has its own interrupt mechanism, but several functions are common.
- The interrupt must transfer control to the appropriate interrupt service routine.

- The straightforward method for handling this transfer would be to invoke a generic routine to examine the interrupt information; the routine, in turn, would call the interrupt-specific handler.
- However, interrupts must be handled quickly, and, given that only a predefined number of interrupts is possible, a table of pointers to interrupt routines can be used instead.
- The interrupt routine is then called indirectly through the table, with no intermediate routine needed.
- Generally, the table of pointers is stored in low memory (the first 100 or so locations).
- These locations hold the addresses of the interrupt service routines for the various devices.

- This array, or interrupt vector, of addresses is then indexed by a unique device number, given with the interrupt request, to provide the address of the interrupt service routine for the interrupting device.
- After the interrupt is serviced, the saved return address is loaded into the program counter, and the interrupted computation resumes as though the interrupt had not occurred.

I/O INTERRUPTS

- To start an I/O operation, the CPU 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. Once the transfer of data is complete, the device controller informs the CPU that it has finished its operation. It accomplishes this communication by triggering an interrupt.
- Once the I/O is started, two courses of action are possible. In the simplest case, the I/O is started; then, at I/O completion, control is returned to the user process. This case is known as **synchronous I/O.**

- The other possibility, called **asynchronous I/O**, returns control to the user program without waiting for the I/O to complete. The I/O then can continue while other system operations occur.

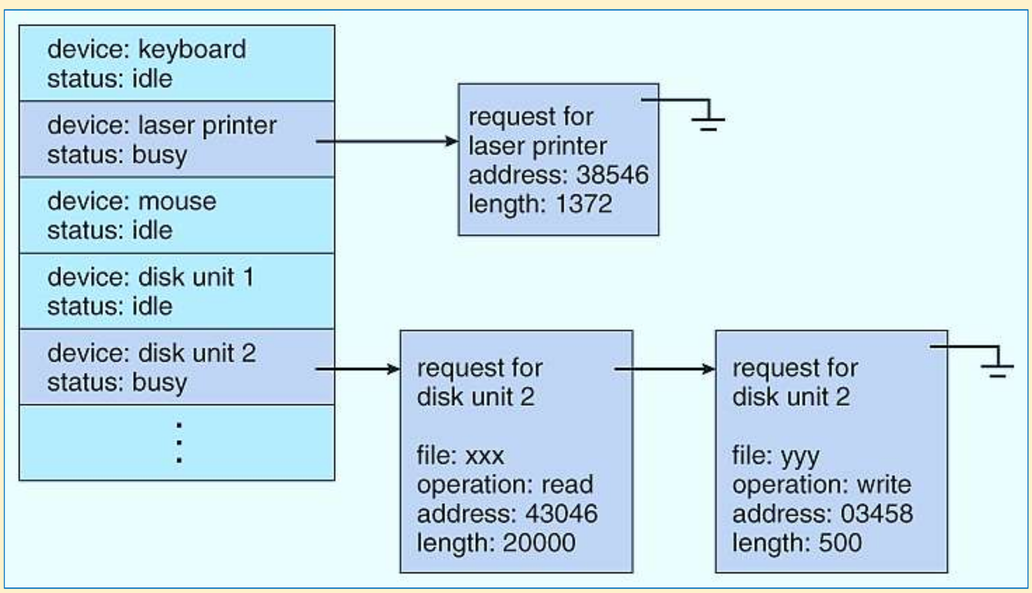
➢ Waiting for I/O completion may be accomplished in one of two ways.

- Special wait instruction that idles the CPU until the next interrupt.
- Wait loop

- If no user programs are ready to run, and the operating system has no other work to do, we still require the wait instruction or idle loop.

- We also need to be able to keep track of many I/O requests at the same time.
  - For this purpose, the operating system uses a table containing an entry for each I/O device called the **device-status table** .
  - Each table entry indicates the device's type, address, and state (not functioning, idle, or busy).
  - If the device is busy with a request, the type of request and other parameters will be stored in the table entry for that device.
  - Since it is possible for other processes to issue requests to the same device, the operating system will also maintain a **wait queue** — a list of waiting requests — for each I/O device.

Device-status table



- An I/O device interrupts when it needs service.
- When an interrupt occurs, the operating system first determines which I/O device caused the interrupt.
- It then indexes into the I/O device table to determine the status of that device, and modifies the table entry to reflect the occurrence of the interrupt.
- The main advantage of asynchronous I/O is increased system efficiency.

HARDWARE PROTECTION

- In early computers, programmers had complete control over the system. As operating systems developed, this control was given to the operating system.
- To improve system utilization, the operating system began to share system resources among several programs simultaneously.
- Multiprogramming put several programs in memory at the same time.
- Errors can occur in a multiprogramming system, where one erroneous program might modify the program or data of another program, or even the resident monitor itself.

- Many programming errors are detected by the hardware. These errors are normally handled by the operating system.
- If a user program fails in some way — such as by making an attempt either to execute an illegal instruction, or to access memory that is not in the users address space — then the hardware will trap to the operating system.
- The trap transfers control through the interrupt vector to the operating system, just like an interrupt.
- Whenever a program error occurs, the operating system must abnormally terminate the program.

❖Dual-Mode Operation

- To ensure proper operation, we must protect the operating system and all other programs and their data from any malfunctioning program. Protection is needed for any shared resource.

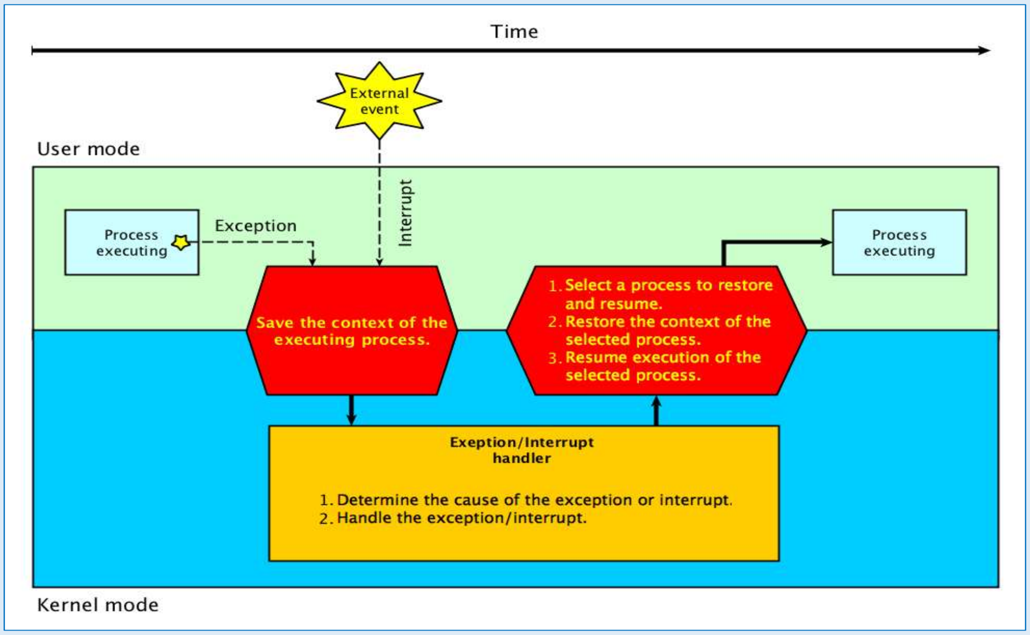
➢The two separate modes of operations are:

- **User mode**
- **Monitor mode** (also called Kernel mode, 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: **monitor (0)** or **user (1).**

- At system boot time, the hardware starts in monitor mode. The operating system is then loaded, and starts user processes in user mode.
- Whenever a trap or interrupt occurs, the hardware switches from user mode to monitor 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 monitor 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.
- We accomplish this protection by designating some of the machine instructions that may cause harm as privileged instructions.
- The hardware allows privileged instructions to be executed only in monitor mode.
- If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction, but rather treats the instruction as illegal and traps it to the operating system.



I/O PROTECTION

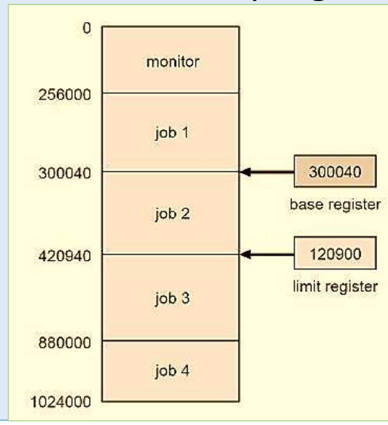
- A user program may disrupt the normal operation of the system by issuing illegal I/O instructions, by accessing memory locations within the operating system itself, or by refusing to relinquish the CPU.
- We can use various mechanisms to ensure that such disruptions cannot take place in the system.
- To prevent users from performing illegal I/O, we define all I/O instructions to be privileged instructions. Thus, users cannot issue I/O instructions directly; they must do it through the operating system. For I/O protection to be complete, we must be sure that a user program can never gain control of the computer in monitor mode.

MEMORY PROTECTION

- To ensure correct operation, we must protect the interrupt vector from modification by a user program.
- In addition, we must also protect the Interrupt Service Routines (ISR) in the operating system from modification.
- Even if the user did not gain unauthorized control of the computer, modifying the interrupt service routines would probably disrupt the proper operation of the computer system and of its spooling and buffering
- We must provide memory protection at least for the interrupt vector and the interrupt-service routines of the operating system.

- To separate each program's memory space, we need the ability to determine the range of legal addresses that the program may access.
- We can provide this protection by using two registers, usually a **base** and a **limit**.
- **Base register holds** the smallest legal physical memory address; the **limit register** contains the size of the range.
- For example, if the base register holds 300040 and limit register is 120900, then the program can legally access all addresses from 300040 through 420940 inclusive.
- The base and limit registers can be loaded by only the operating system, which uses a special privileged instruction.

- Since privileged instructions can be executed in only monitor mode, and since only the operating system executes in monitor mode, only the operating system can load the base and limit registers. This scheme allows the monitor to change the value of the registers, but prevents user programs from changing the registers' contents.



CPU PROTECTION

- We must prevent a user program from getting stuck in an infinite loop or not calling system services, and never returning control to the operating system.
- To accomplish this goal, we 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.
- Before turning over control to the user, the operating system ensures that the timer is set to interrupt. Thus, we can use the timer to prevent a user program from running too long.
- A more common use of a timer is to implement time sharing.
- In the most straightforward case, the timer could be set to interrupt every N millisecond, where N is the time slice that each user is allowed to execute before the next user gets control of the CPU.
- Another use of the timer is to compute the current time.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 59

OPERATING SYSTEMS COMPONENTS

1. Process Management

- **process - program in execution.**
- Eg: compiler , word-processing program , A system task, such as sending output to a printer
- 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.
- a program is a passive entity, whereas a process is an active entity, with a program counter specifying the next instruction to execute.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 60

- The execution of a process must be sequential. The CPU executes one instruction of the process after another; until the process completes .A process is the unit of work in a system.

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

✓ Creating and deleting both user and system processes

✓ Suspending and resuming processes

✓ Providing mechanisms for process synchronization

✓ Providing mechanisms for process communication

✓ Providing mechanisms for deadlock handling

2. Main-Memory Management

- Main memory is a large array of words or bytes. Main memory is a repository of quickly accessible data shared by the CPU and I/O devices.
- The central processor reads instructions from main memory during the instruction-fetch cycle, and it both reads and writes data from main memory during the data-fetch cycle.
- The I/O operations implemented via DMA also read and write data in main memory.
- For the CPU to process data from disk, those data must first be transferred to main memory by CPU-generated I/O calls.

- To improve both the utilization of the CPU and the speed of the computer's response to its users, we must keep several programs in memory.

➢ 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 are to be loaded into memory when memory space becomes available

✓ Allocating and deallocating memory space as needed

3. File Management

- File management is one of the most visible components of an operating system.
- For convenient use of the computer system, 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 called the file.
- The operating system maps files onto physical media, and accesses these files via the storage devices.

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

✓ Creating and deleting files

✓ Creating and deleting directories

✓ Supporting primitives for manipulating files and directories

✓ Mapping files onto secondary storage

✓ Backing up files on stable (nonvolatile) storage media

4. I/O System Management

- One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user.

➢ The I/O subsystem consists of

✓ A memory-management component that includes buffering, caching, and spooling

✓ A general device-driver interface

✓ Drivers for specific hardware devices

- Only the device driver knows the peculiarities of the specific device to which it is assigned.

5. Secondary-Storage Management

- Because 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.

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

✓ Free-space management

✓ Storage allocation

✓ Disk scheduling

6. Networking

- A distributed system is a collection of processors that do not share memory, peripheral devices, or a clock.
- Instead, each processor has its own local memory and clock, and the processors communicate with one another through various communication lines, such as high-speed buses or networks.
- The processors in a distributed system vary in size and function. They may include small micro- processors, workstations, minicomputers, and large, general-purpose computer systems.
- The processors in the system are connected through a communication net- work, which can be configured in a number of different ways

7. Protection System

- If a computer system has multiple users and allows the concurrent execution of multiple processes, then the various processes must be protected from one another's activities.
- For that purpose, mechanisms ensure that the 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 is any mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.

USER-SYSTEM INTERFACE

➢ There are several ways for users to interact with the operating system. The two fundamental approaches are

❖ **Command interpreter**

❖ **Graphical User Interface (GUI)**

- **Menu based interface**
- **Iconic interface**

➢ **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.
- For example, on UNIX and Linux systems, a user may choose among several different shells, including the Bourne shell, Cshell, Bourne-Again shell, Korn shell, and others.
- 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.

❖**Implementation of commands**

1. In one approach, the command interpreter itself contains the code to execute the command.

- Here the number of commands is fixed and it is predefined.

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

- Commands are not fixed and it is expandable.
- New commands can be added.

➢Graphical User Interfaces (GUI)

- 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.

OPERATING SYSTEM SERVICES

➢An operating system provides an environment for the execution of programs.

- **Program execution:** The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).
- **I/O operations:** A running program may require I/O. This I/O may involve a file or an I/O device. For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the operating system must provide a means to do I/O.

- **File-system manipulation:** Obviously, programs need to read and write files. Programs also need to create and delete files by name.
- **Communications:** one process needs to exchange information with another process. Communications may be implemented via shared memory, or by the technique of message passing, in which packets of information are moved between processes by the operating system.
- **Error detection:** Errors may occur in the CPU and memory hardware (memory error or a power failure), in I/O devices (connection failure on a network, or lack of paper in the printer), and in the user program (an arithmetic overflow, an attempt to access an illegal memory location). For each type of error, the operating system should take the appropriate action

- **Resource allocation:** When multiple users are logged on the system or multiple jobs are running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system.
- **Accounting:** We want to keep track of which users use how much and what kind of computer resources. This record keeping may be used for accounting or simply for accumulating usage statistics.
- **User interface:** One is a command-line interface (CLI), which uses text commands and a method for entering them. 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.

- **Protection and security:** When several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the operating system itself. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important.

SYSTEM CALLS

- System calls provide the **interface between a process and the operating system.**
- The purpose of system call is to request the operating system to perform some activity.
- The execution of a system call requires the user process to save its current state, let the operating system take control of the CPU and perform some function. Then OS should save its state and give control of the CPU back to the user process
- System calls are generally available as assembly language instructions.

- But assembly language programming is considered difficult, therefore, high level languages such as C, C++, PERL etc, allow these system calls to be made directly.

➢ System calls can be grouped roughly into **five** major categories:

1. 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

2. File management

- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

3. Device management

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

4. 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

5. Communications

- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 81

- The following Figure , which illustrates how the operating system handles a user application invoking the open() system call.

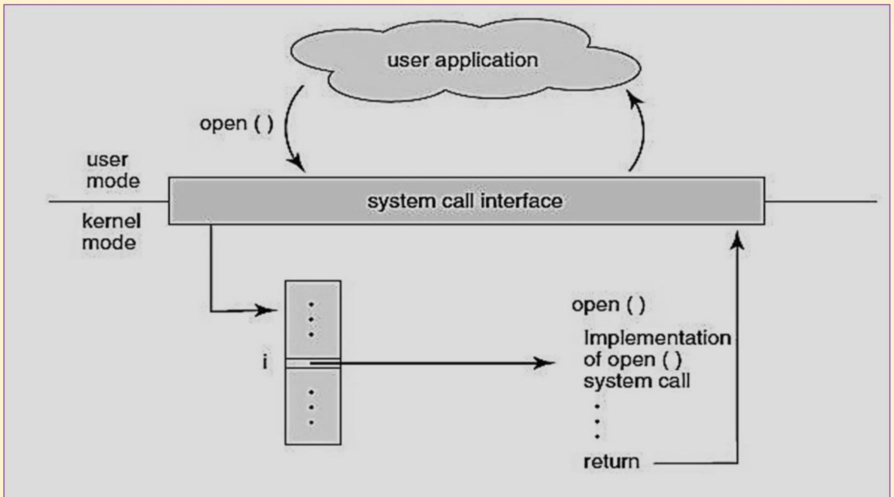


Figure: The handling of a user application invoking the open() system call.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 82

- Three general methods are used to pass parameters to the operating system.
- The simplest approach is to pass the parameters in registers.
- In some cases, however, there may be more parameters than registers. In these cases, the parameters are generally stored in a block, or table, in memory, and the address of the block is passed as a parameter in a register. 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.

OPERATING SYSTEM STRUCTURES

1. Simple Structure

- Here, no modular approach is done.
- Each component has no specific function.
- It was written to provide the most functionality in least space, because of the limited hardware on which it run.
- So it was not divided in to modules carefully.
- **MS-DOS** is an example of such a system.

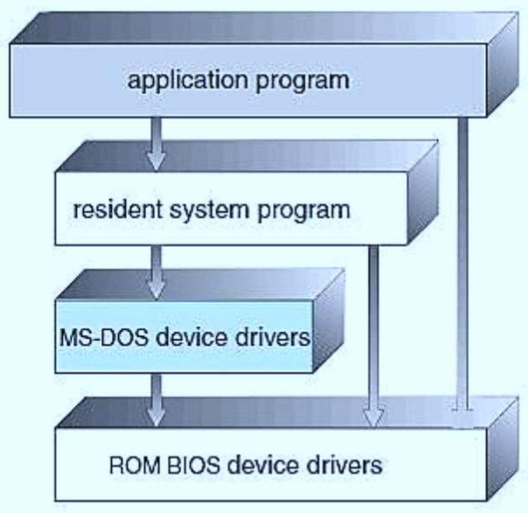


Figure: MS-DOS layer structure.

❖Disadvantages

- There is no hardware protection and I/O protection
- No system call concept
- Direct interaction causes errors or malfunctions.

➢UNIX initially was limited by hardware functionality. It consists of two separable parts: the kernel and the system programs. The kernel is further separated into a series of interfaces and device drivers, which have been added and expanded over the years as UNIX has evolved. The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls.

2. Layered Approach

- Layered approach introduces the modularization of the system.
- Here the operating system is broken up into a number of layers (or levels), each built on top of lower layers.
- The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.
- Each layer has a specific function.
- Each layer has its own Data structures and operations.
- Highest level has high functionality, i.e., the services for highest level are provided by the lowest level.

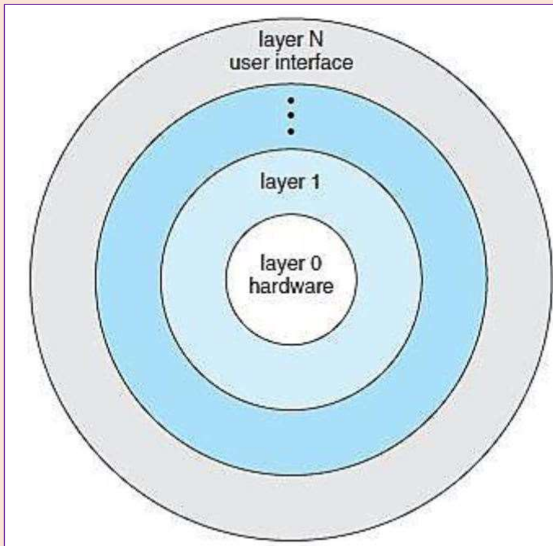


Figure: A layered operating system.

❖Advantages

- The main advantage of the layered approach is modularity.
- Each layers are independent, so debugging is simple
- Independent error checking is possible
- Design and implementation are simplified when the system is broken down into layers.
- Each layer hides the existence of certain data structures, operations, and hardware from higher-level layers.

❖Disadvantages

- It is a difficult task to define the functions of each layer. Layer 1 can access only the lowest layer (Layer 0)
- Less efficient because when user give a request, that process is passes through each layer. So more time is consumed.

3. Microkernel Approach

- In this method, the structure of the OS is changed by removing all nonessential components from the kernel, and implementing them as system- and user-level programs.
- The result is a smaller kernel. The removed components are implemented using user programs. The kernel only gives communication support.

❖Advantages

- Ease of extending the operating system without modifying the kernel.
- The OS is portable.
- Provides more security and reliability.

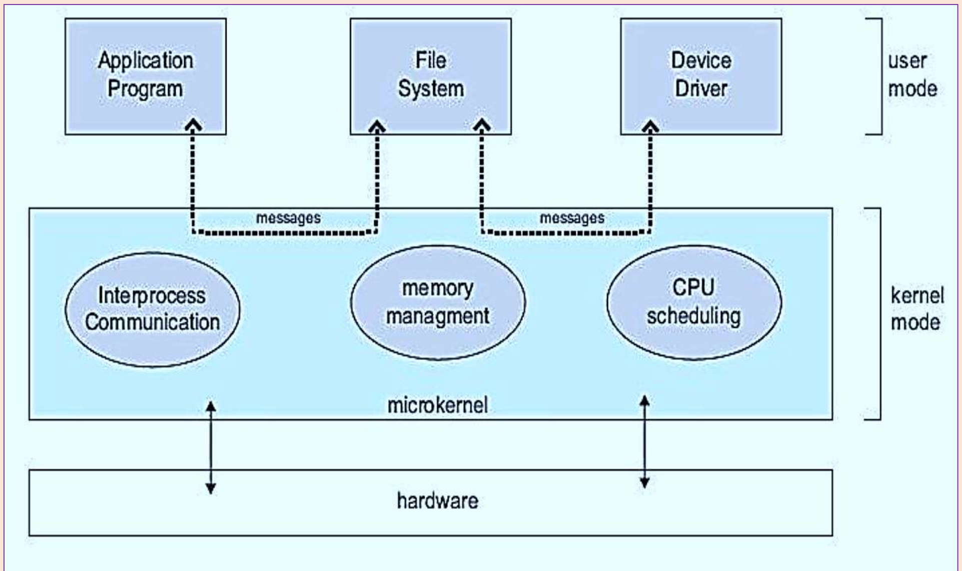


Figure: Architecture of a typical microkernel.

4. Modular Approach

- The best current methodology for operating-system design involves using loadable kernel modules.
- Here, the kernel has a set of core components and links in additional services via modules, either at boot time or during runtime.
- This type of design is common in modern implementations of UNIX, such as Solaris, Linux, and Mac OS X, as well as Windows.
- Object oriented Programming (OOP) technique is used to create a modular kernel.

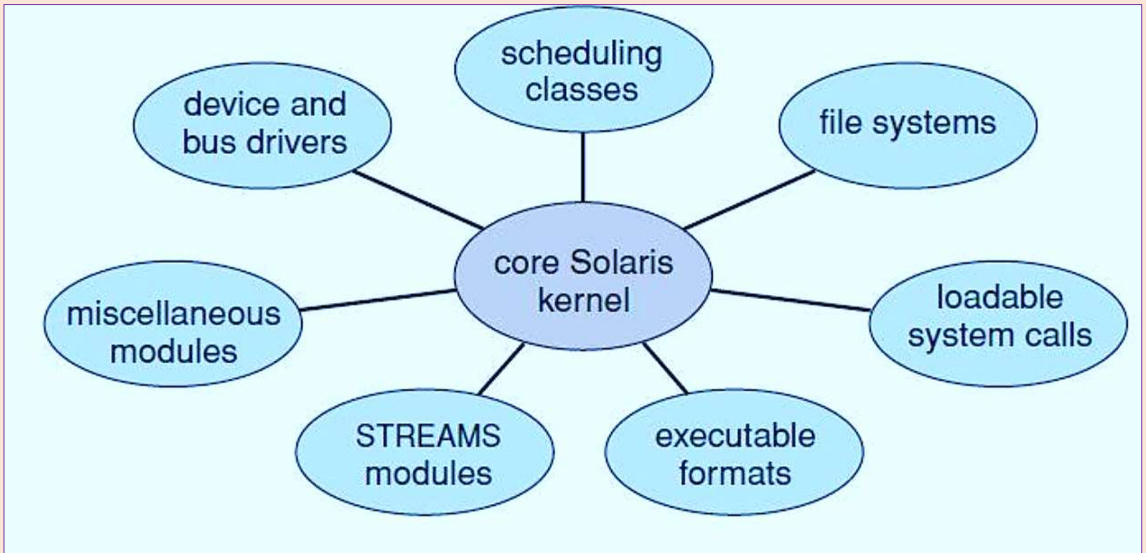


Figure: Solaris loadable modules

SYSTEM BOOT PROCESS

- The procedure of starting a computer by loading the kernel is known as booting the system.
- On most computer systems, a small piece of code known as the bootstrap program or bootstrap loader locates the kernel, loads it into main memory, and starts its execution.
- Some computer systems, such as PCs, use a two-step process in which a simple bootstrap loader fetches a more complex boot program from disk, which in turn loads the kernel.

- When a CPU is powered up or rebooted—the instruction register is loaded with a predefined memory location, and execution starts there.
- At that location is the initial bootstrap program.
- This program is in the form of read-only memory (ROM), because the RAM is in an unknown state at system startup.
- ROM is convenient because it needs no initialization and cannot easily be infected by a computer virus.
- The bootstrap program can perform a variety of tasks. Usually, one task is to run diagnostics to determine the state of the machine.
- If the diagnostics pass, the program can continue with the booting steps.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 95

- It can also initialize all aspects of the system, from CPU registers to device controllers and the contents of main memory.
- Sooner or later, it starts the operating system. GRUB is an example of an open-source bootstrap program for Linux systems.
- Modern computer system is to store the small loader for bootstrap program in ROM.
- All the rest of this program is written to a dedicated area on the hard disk. This area of the disk is known as boot sector.
- If a disk has a boot partition, it is called boot disk or system disk. Then to start computer, the loader in ROM is started.

Prepared By Mr.EBIN PM, AP, IESCE EDULINE 96

- This loader finds the boot sector on the disk, loads bootstrap program from the boot sector in to memory and then transfers the control to the bootstrap program (now loaded in memory), which in turn does the initialization job. This procedure is called Booting from the Disk.