UNIT-I

INTRODUCTION

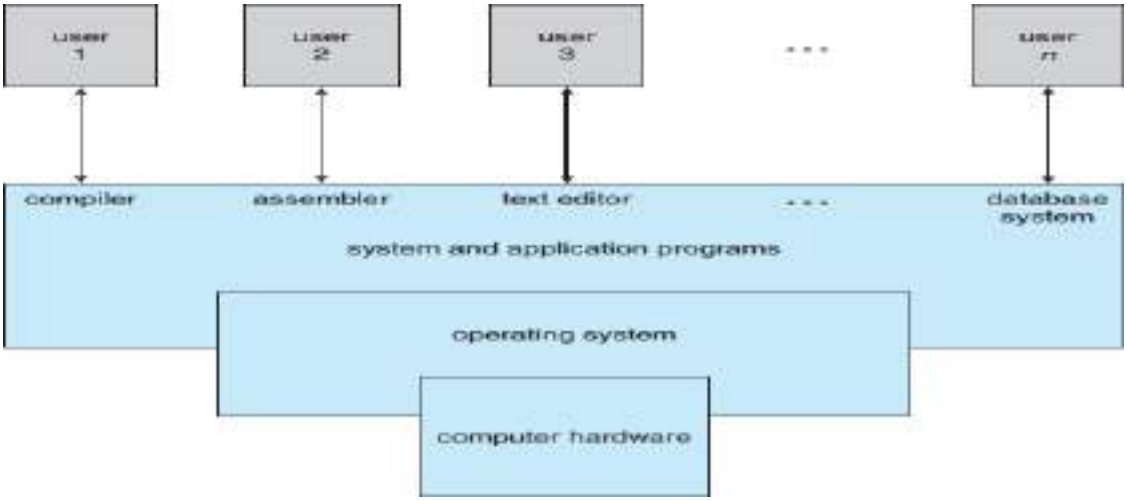
Operating System acts as an intermediary between the user of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.

- Mainframe operating systems are designed primarily to optimize the utilization of hardware.
- Personal computer (PC) operating systems support complex games, business applications, and everything in between.
- Mobile computer operating systems provide an environment in which a user can easily interface with the computer to execute programs.

Components of Computer System

A computer system can be divided roughly into four components:

1. Hardware
2. Operating system
3. Application programs
4. Users



- Hardware such as the Central processing unit (CPU), Memory and Input/Output (I/O) devices provide the basic computing resources for the system.
- 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 controls the hardware and coordinates its use among the various application programs for the various users. Operating systems provide an **environment** within which other programs can do useful work.

SYSTEM VIEW OF OPERATING SYSTEM

From the computer's point of view an operating system is viewed as a **Resource Allocator** and a **Control Program**.

- The operating system acts as the manager of the resources such as CPU time, memory space, file-storage space, I/O devices and so on. Resource allocation is important where many users access the same mainframe or minicomputer. An operating system is a **control program** that manages the execution of user programs to prevent errors and improper use of the computer. It is especially concerned with the

operation and control of I/O devices.

Note: Resource utilization means how various hardware and software resources are shared.

USER VIEW OF OPERATING SYSTEM

The user's view of the computer varies according to the **Interface** being used.

- **Personal Computer:** Most computer users sit in front of a PC consisting of a monitor, keyboard, mouse, and system unit. Such a system is designed for one user to monopolize its resources. The goal is to maximize the work that the user is performing. These operating systems are designed mostly for **ease of use** without considering resource utilization because these systems are single-user systems.
- **Main frame:** A user sits at a terminal connected to a mainframe or a minicomputer. Other users are accessing the same computer through other terminals. These users share resources and may exchange information. The operating system in such cases is designed to maximize resource utilization— to assure that all available CPU time, memory, and I/O are used efficiently and that no individual user takes more than her fair share.
- **Workstations:** Users sit at workstations connected to networks of other workstations and servers. These users have dedicated resources at their disposal, but they also share resources such as networking and servers, including file, compute, and print servers. Therefore the operating system is designed to compromise between individual usability and resource utilization.
- **Mobile Computers:** The user interface for mobile computers generally features a **touch screen**, where the user interacts with the system by pressing and swiping fingers across the screen rather than using a physical keyboard and mouse.
- **Embedded Computing**: Some computers may not have a user view. Embedded computers in-home devices and automobiles may have numeric keypads and may turn indicator lights on or off to show status, but their operating systems are designed primarily to run without user intervention. Ex: Printers, GPS, Microwave Ovens, TV, Digital Camera, etc.

Defining Operating Systems

Universally there is no accepted definition for an operating system:

- Operating system is software that is used for controlling and allocating resources. The fundamental goal of computer systems is to execute user programs and to make solving user problems easier. Computer hardware alone is not easy to use, application programs are developed. These programs require certain common operations such as controlling the I/O devices.
- The operating system is also defined as the program that is running at all times on the computer is called the **Kernel**.

Terms related to operating system

- **System programs**: These are associated with the operating system but are not necessarily part of the kernel.
- **Application programs**: Programs that are not associated with the operating system.
- **Middleware**: It is a set of software frameworks that provide additional services to application developers. **Ex**: Mobile operating system of Apple's iOS and Google's Android features a core kernel along with middleware that supports databases, multimedia, and graphics.
- **Firmware**: Read Only memory (ROM), and EEPROM are considered as firmware.

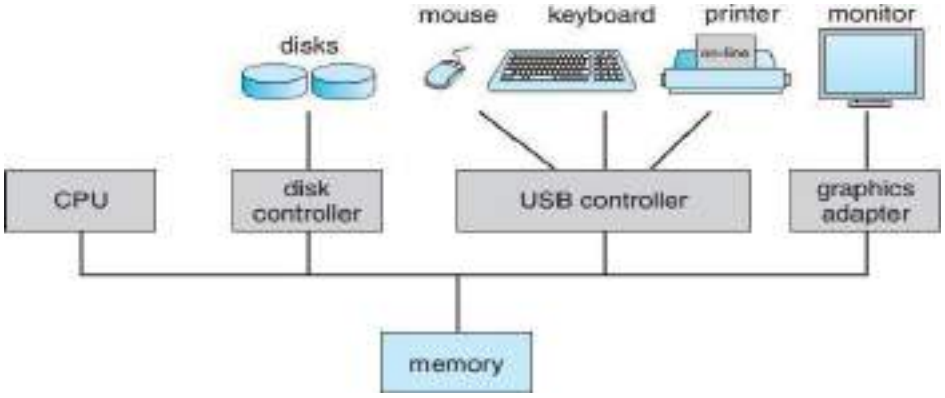
COMPUTER-SYSTEM ORGANIZATION

Computer system organization has several parts:

1. Computer system operation
2. Storage structure
3. I/O structure

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 devices such as disk drives, audio devices, or video displays. 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, it needs to run a bootstrap **program**. This bootstrap program is stored



within Read-Only Memory (**ROM**).

- It initializes all aspects of the system, from CPU registers to device controllers to memory contents.
- The bootstrap program must know how to load the operating system and how to start executing that system.
- To accomplish this goal, the bootstrap program must locate the operating-system kernel and load it into memory.
- Once the kernel is loaded and executed, it can provide services to the system and its users.
- Some services are provided outside of the kernel by system programs that are loaded into memory at boot time to become **System processes** or **System daemons** that run the entire time the kernel is running.
  Ex: On UNIX, the first system process is **—init** and it starts many other daemons.

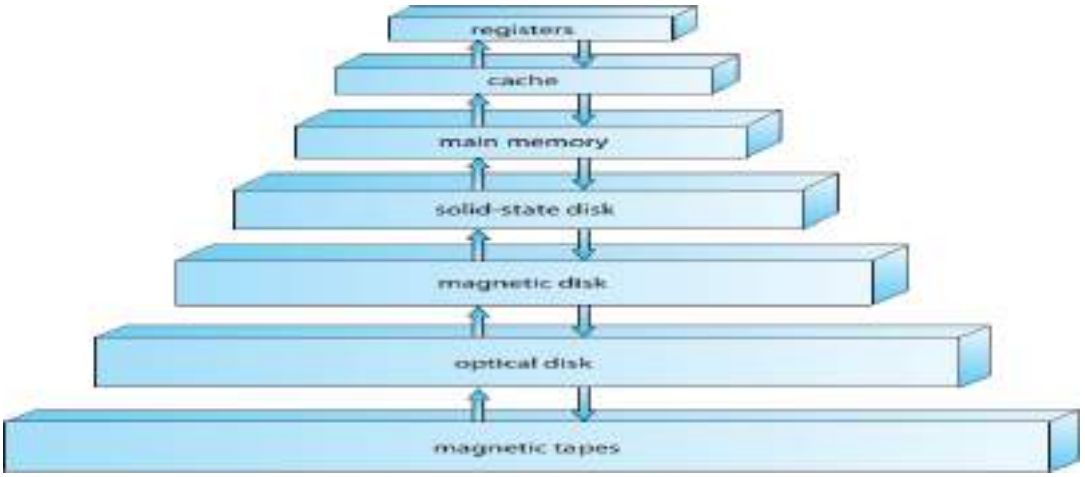
Hardware and Software Interrupts

- 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 by way of the system bus.
- Software may trigger an interrupt by executing a special operation called a **System call** or M**onitor calls**.
- When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location. The fixed location is usually the starting address where the service routine for the interrupt is located.
- The interrupt service routine executes and after completion of the interrupt, the CPU resumes the interrupted computation.
- Interrupt vector is an array that stores the addresses of the interrupt service routines for the various devices.

Storage Structure

The CPU can load instructions only from memory, so any programs to run must be stored in memory. There are two types of storage:

1. Volatile storage: Registers, Cache, and Main memory.
2. Non-volatile storage: Solid state disks, Magnetic disks, Optical disks, Magnetic tapes.



Volatile storage

devices are fast and expensive but lose their contents when power is turned off.

- **Register:** Each byte of memory has its own address. Interaction is achieved through a sequence of load or store instructions to specific memory addresses.
  Load instruction moves a byte or word from the main memory to an internal register within the CPU. Store instruction moves the content of a register to the main memory.
- **Cache:** It is a faster storage system used to store information temporarily and the size of the cache is very small. It is located between the CPU and Main Memory.
  - **Main Memory:** It is also called Random Access memory (RAM) is a rewritable memory. Computers run most of their programs from Main memory. Main memory is implemented in Dynamic RAM (DRAM) technology. The main memory is too small to store all the needed programs and data permanently.
  - Non-volatile storage or Secondary storage devices

  These are extensions of main memory. These devices store large quantities of data permanently.
  Read-Only Memory (ROM) stores static programs such as bootstrap programs. Electrically Erasable Programmable Read-Only Memory (EEPROM) can be changed but cannot be changed frequently because it contains mostly static programs. Smartphones have EEPROM to store their factory-installed programs.
- **Solid-State Disks:** Solid-state disk stores data in a large DRAM array during normal operation but also contain a hidden magnetic hard disk and a battery for backup power. If external power is interrupted, this solid-state disk's controller copies the data from the RAM to the magnetic disk. When external power is restored, the controller copies the data into RAM. These are faster than magnetic disks.
- **Magnetic disk**: It provides storage for both programs and data. Most of the system programs and application programs are stored on a disk until they are loaded into memory.
- **Optical Disk**: It uses Optical Storage techniques. Ex: CD, DVD.
- **Magnetic Tape**: It uses a long strip of narrow plastic film with tapes of thin magnetizable coating. This is used to record video and audio data.
- **Flash memory:** These are popular in cameras, **Personal Digital Assistants (PDAs)**, robots, and general-

purpose computers. Flash memory is slower than DRAM but needs no power to retain its contents.

I/O Structure

- A Computer system consists of CPUs and multiple device controllers that are connected through a common bus.

Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

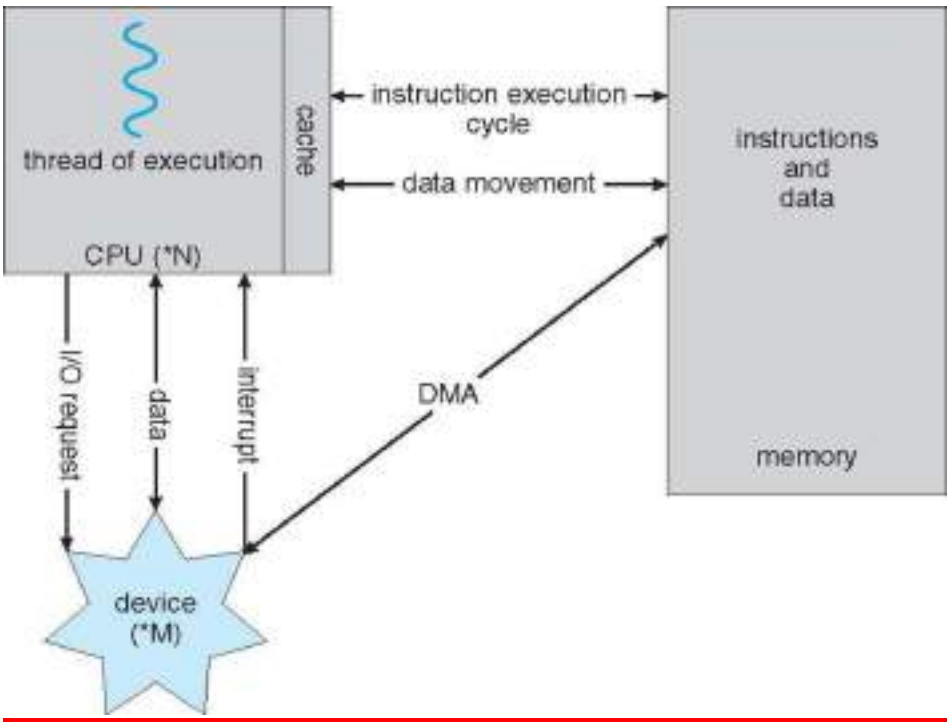
- Each device controller is in charge of a specific type of device. A device controller maintains some local buffer storage and a set of special-purpose registers.
- The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage.
- Operating systems have a **device driver** for each device controller. This device driver understands the device controller and provides the rest of the operating system with a uniform interface to the device.
- To start an I/O operation, the device driver loads the appropriate registers within the device controller. The device controller examines the contents of these registers to determine what action to be taken such as read or write etc.
- 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 status information.

Direct Memory Access Structure

Used for high-speed I/O devices able to transmit information at close to memory speeds

Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention

Only one interrupt is generated per block, rather than the one interrupt per byte



COMPUTER-SYSTEM ARCHITECTURE

A computer system can be categorized according to the number of general-purpose processors used.

1. Single-Processor Systems
2. Multi-Processor Systems
3. Clustered systems

Single Processor system

- Single processor system has one main CPU capable of executing a general-purpose instruction set from system processes and user processes.
- All single-processor systems have other special-purpose processors also. They come in the form of device-specific processors such as disk, keyboard and graphics controllers, I/O processors, etc.
- These special-purpose processors run a limited instruction set and do not run user processes. They are managed by the operating system.
- Operating system sends information about their next task to these processors and monitors their status.
- **Example:** A disk-controller microprocessor receives a sequence of requests from the main CPU and implements its own disk queue and scheduling algorithm. This arrangement relieves the main CPU of the overhead of disk scheduling.

  Note: The use of special-purpose microprocessors does not turn a single-processor system into a multiprocessor.

Multi-Processor system

Multi-processor systems have two or more processors that share the resources such as computer buses, memory, clock, and peripheral devices.

Multiprocessor systems have three main advantages:

i. **Increases the throughput:** By increasing the number of processors more work will be done in less time (i.e.) the speed of the system will increase.

ii. **The economy of scale**: Multiprocessor systems share peripherals, mass storage, and power supplies; hence the cost is less as compared to multiple single-processor systems.

iii. **Increased reliability:** In single-processor systems, if the processor fails the entire system fails, but in

multi-processor systems if a processor fails then the entire system will not fail, other processors will pick up and share the work of the failed processor.

Multi-processor systems are of two types:

1. Asymmetric multiprocessing
2. Symmetric multiprocessing

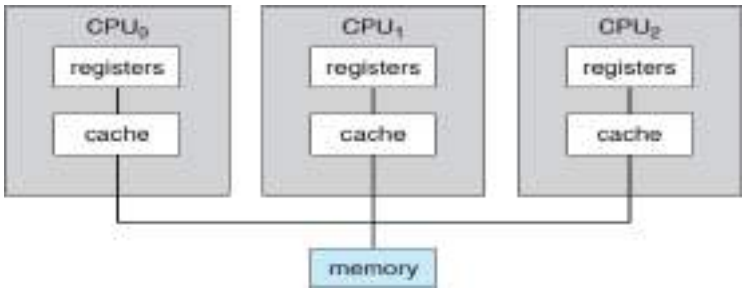
Asymmetric Multi-Processing:

- Each processor is assigned a specific task in asymmetric multiprocessing and it uses a boss–worker relationship.
- The Boss processor controls the system by scheduling and assigning tasks to the worker processor. The worker processor has to complete the task assigned by the boss processor.

Symmetric Multi-Processing (SMP)

Each processor performs all tasks within the operating system and no boss–worker relationship exists between processors.

The below figure shows the SMP architecture:

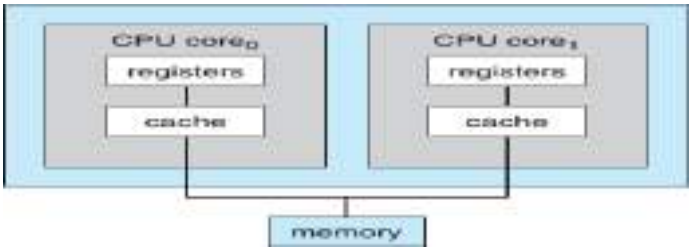


- Each processor has its own set of registers and cache memory and all processors share physical memory.
- In SMP if there are N CPUs then N processes can run without causing performance degradation.
- It is the most commonly used system. All modern operating systems including Windows, Mac OS X, and Linux provide support for SMP.

Multicore processors

- Multicore processors are special types of multiprocessors in which multiple computing cores reside on a single chip. Each core has its own register set and its own local cache.
- These are more efficient than multiple chips with single cores because on-chip communication is faster and one chip with multiple cores uses less power than multiple single-core chips.

The below figure shows a dual-core design with two cores on the same chip



Clustered Systems

- **Clustered system** is also a type of multiprocessor system. Clustered systems are composed of two or more individual systems or nodes joined together.
- Each node may be a single processor system or a multicore system.

- Clustered computers share storage and are closely linked via a local-area network LAN.
- Clustering is usually used to provide **High-availability** service (i.e.) 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 only monitors the active server. If the active 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.
  Parallel clusters: parallel clustering allows multiple hosts to access the same data on shared storage. The clustering is done over Wide Area Networks (WAN's).

OPERATING SYSTEM OPERATIONS

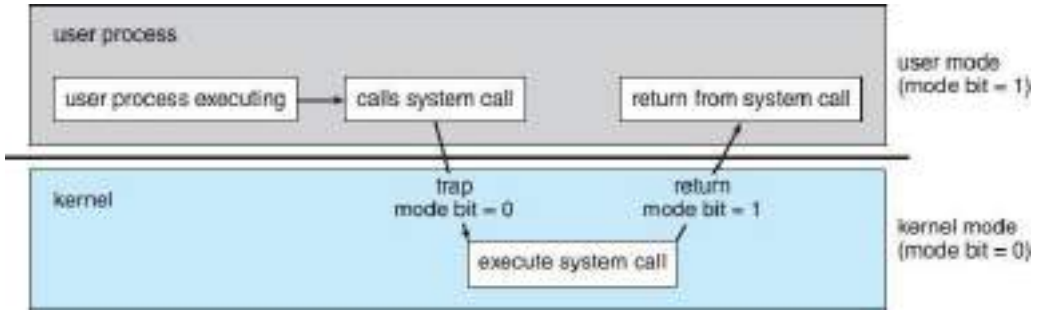
1. Dual mode or multimode operation
2. Timers

Dual mode or Multimode operation

In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating-system code and user-defined code.
The operating system needs two modes of operation (Dual Mode):

i. User mode
ii. Kernel mode Supervisor mode or System mode or Privileged mode.

- A mode bit is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1).
- Mode bit can distinguish between a task that is executed on behalf of either the operating system or the user.
- The system is in user mode when the computer system is executing on behalf of a user application.
- When a user application requests a service from the operating system via a system call, the system must transition from user to kernel mode to fulfill the request.
  Modern operating systems are **Interrupted driven**. A **trap** or an **Exception** is a software-generated interrupt caused either by an error or by a specific request from a user program for an operating-system service to be performed.
  Consider the figure that shows the transition from user to kernel mode.



- At system boot time, the hardware starts in kernel mode.

- The operating system is then loaded and starts user applications in user mode.
- Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode by changing the state of the mode bit to 0.
- (i.e.) Whenever the operating system gains control of the computer, it is in kernel mode.
- The system always switches to user mode by setting the mode bit to 1 before passing control to a user program.
  The dual mode of operation protects the operating system from errant users and errant users from one another.
- 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 kernel mode.
- If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it in the operating system.
- The instruction to switch to kernel mode is an example of a privileged instruction.
- Examples of Privileged instructions include I/O control, timer management and interrupt management.

Dual-mode operation is implemented in Microsoft Windows 7, Unix, and Linux OS.

> **Sysem calls:** System calls provide the means for a user program to ask the operating system to perform tasks reserved for the operating system on the user program's behalf. When a system call is executed, it is typically treated by the hardware as software interrupt. The kernel examines the interrupting instruction to determine what system call has occurred.

Timer

- **Timer**s are implemented in operating systems to maintain control over the CPU.
- A timer can be set to interrupt the computer after a specified period of time to avoid getting stuck into an infinite loop.
- The period may be fixed or varied from 1 millisecond to 1 second
- A **variable timer** is 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.
- If the timer interrupts, the control transfers automatically to the operating system, which may treat the interrupt as a fatal error or may give the program more time.
- Instructions that modify the content of the timer are privileged.

PROCESS MANAGEMENT

A process is a program in execution. A program does nothing unless its instructions are executed by a CPU.

Example:

1. A time-shared user program such as a compiler is a process.
2. A word-processing program being run by an individual user on a PC is a process.
3. A system task such as sending output to a printer is a process.

- A process needs certain resources including CPU time, memory, files, and I/O devices to accomplish its task.
- These resources are either given to the process when it is created or allocated to it while it is running.
- A program is a ***passive*** entity, like the contents of a file stored on disk, whereas a process is an ***active*** entity.
- The CPU executes one instruction of the process after another until all instructions in that process complete.
- At any time, at most only one instruction is executed on behalf of the process.
- A process is the unit of work in a system. A system consists of a collection of processes such as operating-system processes and user processes.
- Operating system processes execute system code whereas user processes execute user code.

The operating system is responsible for the following activities of process management:

1. Creating and deleting both user and system processes.
2. Scheduling processes and threads on the CPUs.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization and process communication.

MEMORY MANAGEMENT

- Main memory is a large array of bytes. Each byte has its own address. Main memory is a repository of quickly accessible data shared by the CPU and I/O devices.
- A program to be executed, it must be mapped to absolute addresses and loaded into memory.
- As the program executes, it accesses program instructions and data from the main memory by generating these absolute addresses.
- The program terminates, its memory space is declared available and the next program can be loaded and executed.

 Memory management is needed to improve both CPU utilization and computer speed. Operating systems is responsible for the following activities of memory management:

1. Allocating and deallocating memory space as needed.
2. Deciding which processes and data to move into the memory and move out of memory.
3. Keeping track of which parts of memory are currently being used and who is using them.

STORAGE MANAGEMENT

Storage management can be divided into 4 different categories:

1. File system management
2. Mass storage management
3. Cache
4. I/O devices

File system management

A file is a collection of related information that represents programs and data. Examples: Data files, text files, etc.

OS is responsible for the following activities in connection with file management:

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

Mass-Storage Management

Secondary storage disks are permanent storage devices.

- Most programs including compilers, assemblers, word processors, editors, and formatters are stored on a disk until loaded into memory.
- Magnetic tape drives and tapes and CD and DVD drives and platters are typical **tertiary storage** devices.
- The media tapes and optical platters vary between **WORM** (write-once, read-many-times) and **RW** (read–write) formats.

The operating system is responsible for the following activities in connection with disk management: Free-space management, Storage allocation, and Disk scheduling.

Caching

Cache is a faster storage system used to store information temporarily and the size of the cache is very small.

- Information is normally kept in storage systems such as main memory. As it is used, it is copied into the cache on a temporary basis.
- When we need a particular piece of information, we first check whether it is in the cache.
- If information is in the cache then we use the information directly from the cache.
- If information is not cached, we use the information from the source, putting a copy in the cache.

I/O Systems

One of the purposes of an operating system is to hide details of specific hardware devices from the user. Only the device driver knows the details of the specific device.

Example: In UNIX, the peculiarities of I/O devices are hidden from the bulk of the operating system itself by the **I/O subsystem**.

The I/O subsystem consists of several components:

- A memory-management component that includes buffering, caching and spooling
- A general device-driver interface
- Driver for specific hardware devices

PROTECTION AND SECURITY

Protection is a mechanism for controlling the access of processes or users to the resources defined by a computer system.

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

Example:

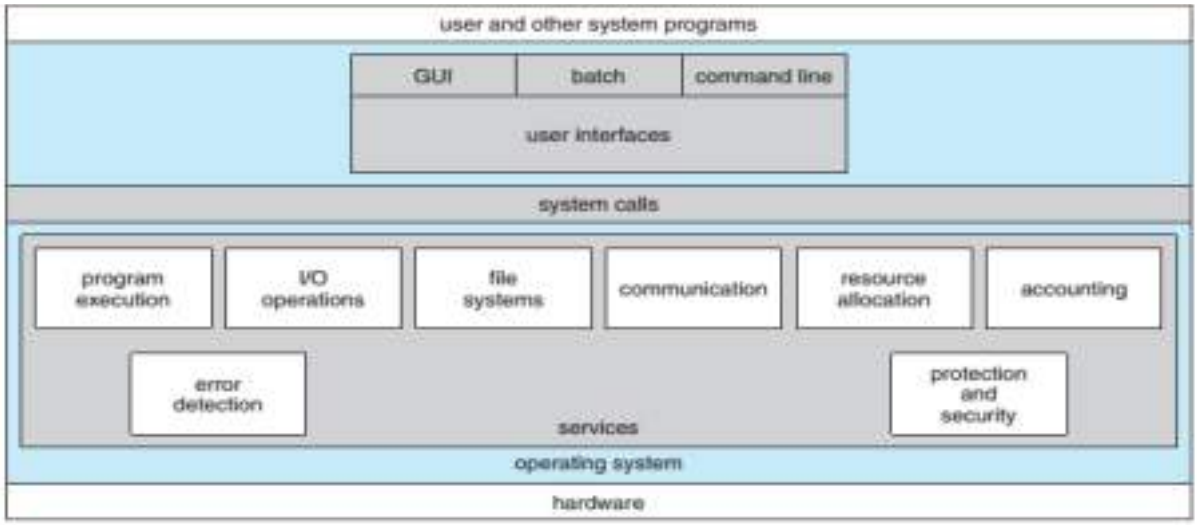
1. Memory-addressing hardware ensures that a process can execute only within its own address space.
2. The timer ensures that no process can gain control of the CPU without eventually relinquishing control.
3. Device-control registers are not accessible to users, so the integrity of the various peripheral devices is protected.

Security is a mechanism to defend a system from external and internal attacks.

- Such attacks include viruses and worms, denial-of-service attacks, identity theft, and theft of service such as unauthorized use of a system.
- Even though the system provides Protection mechanisms, the system still needs security mechanisms. Example: A user whose authentication information such as user id and password is stolen. User data could be copied or deleted even though file and memory protection are working.

OPERATING-SYSTEM SERVICES

1. An Operating system provides an environment for the execution of programs. OS provides certain services to programs and to the users of those programs.
2. Operating system services are provided for the convenience of the programmer and to make the programming task easier.
3. The list of Operating system services that are helpful to the user:
4. 1. User interface
5. 2. Program Execution
6. 3. I/O Operation
7. 4. File system manipulation
8. 5. Communication
9. 6. Error Detection
10. 7. Resource allocation
11. 8. Accounting
12. 9. Protection and Security



User Interface

Almost all operating systems have a **User Interface (UI)**. A UI is generally of 3 types: · **Command-Line Interface (CLI)** uses text commands and a method for entering them. · **Batch Interface:** The commands and directives to control those commands are entered into files and those files are executed.

Graphical User Interface (GUI) is a window system with a pointing device (i.e. mouse) to direct I/O, choose from menus, and make selections and a keyboard to enter text. **Program Execution**

The system must be able to load a program into the main memory and to run that program. The program must be able to end its execution, either normally or abnormally. **I/O operations**

A running program may require I/O, which may involve a file or an I/O device. Examples are blanking a display screen, recording to a CD or DVD drive or.

File-system manipulation

Programs need to read and write files and directories.

· They need to create and delete them by name, search for a file and list file information. · Operating systems include permission management to allow or deny access to files or directories based on file ownership.

Communications

One process in its lifetime needs to communicate with another process to exchange information.

Communication occurs between processes that are executed on the computer. · Communications may be implemented via **Shared Memory or Message Passing**. · In **Shared memory** communication two or more processes read and write to a shared section of memory.

In **Message passing** communication the packets of information in predefined formats are moved between processes by the operating system.

Error Detection

The operating system needs to be detecting and correcting errors constantly. The different types of errors occurred are

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

· User Program errors such as an arithmetic overflow, an attempt to access an illegal memory location.

For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

Resource Allocation

When there are multiple users or multiple jobs running at the same time, resources must be allocated to

each of them.

Operating system resources are CPU cycles, main memory, file storage, I/O devices, etc. ·The operating system manages many of these resources.

CPU cycles, main memory, and file storage may have special allocation codes. I/O devices may have much more general requests and release codes.

In determining how best to use the CPU, operating systems have CPU-scheduling routines that take into account the speed of the CPU, the jobs that must be executed, the number of registers available, and other factors.

There may also be routines to allocate printers, USB storage drives, and other peripheral devices.

The Accounting

Operating system keeps track of different kinds of resources used by all users. · This record keeping may be used for accounting so that users can be billed or simply for accumulating usage statistics.

Usage statistics is a valuable tool for researchers who wish to reconfigure the system to improve computing services.

Protection and security

Protection ensures that all access to system resources is controlled because 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.

Security of the system defends the system from outsider's attacks such as invalid access of system resources such as I/O devices etc. Security starts with user authentication by providing a username and password to gain access to system resources.

OPERATING-SYSTEM STRUCTURES

TYPES OF OS STRUCTURES

1. Simple Structure
2. Layered Approach
3. Microkernels
4. Modules
5. Hybrid Systems

Simple Structure

MS-DOS operating systems do not have well-defined structures.

- It was designed and implemented by a few people and started as small, simple, and limited system and then grew beyond its original scope that the implementers didn't imagine it would be so popular.
- In MS-DOS, the interfaces and levels of functionality are not well separated.
- In MS-DOS, application programs are able to access the basic I/O routines to write directly to the display and disk drives.
- Such freedom leaves MS-DOS vulnerable to errant or malicious programs, causing entire system crashes when user programs fail.

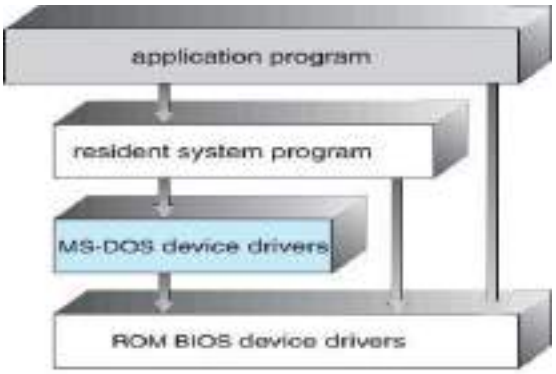


Figure 2.11 MS-DOS layer structure.

The original UNIX operating system also had limited structuring; it was limited by its hardware functionality.

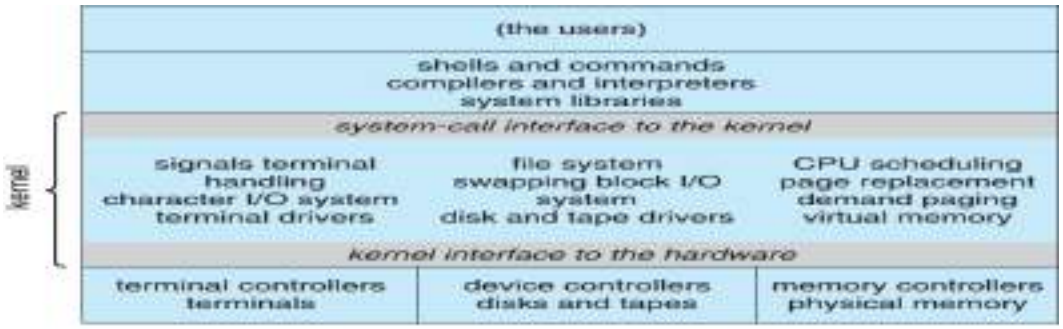


Figure 2.12 Traditional UNIX system structure.

- It consists of two separable parts: **Kernel** and **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 traditional UNIX operating system is layered and structured to some extent.

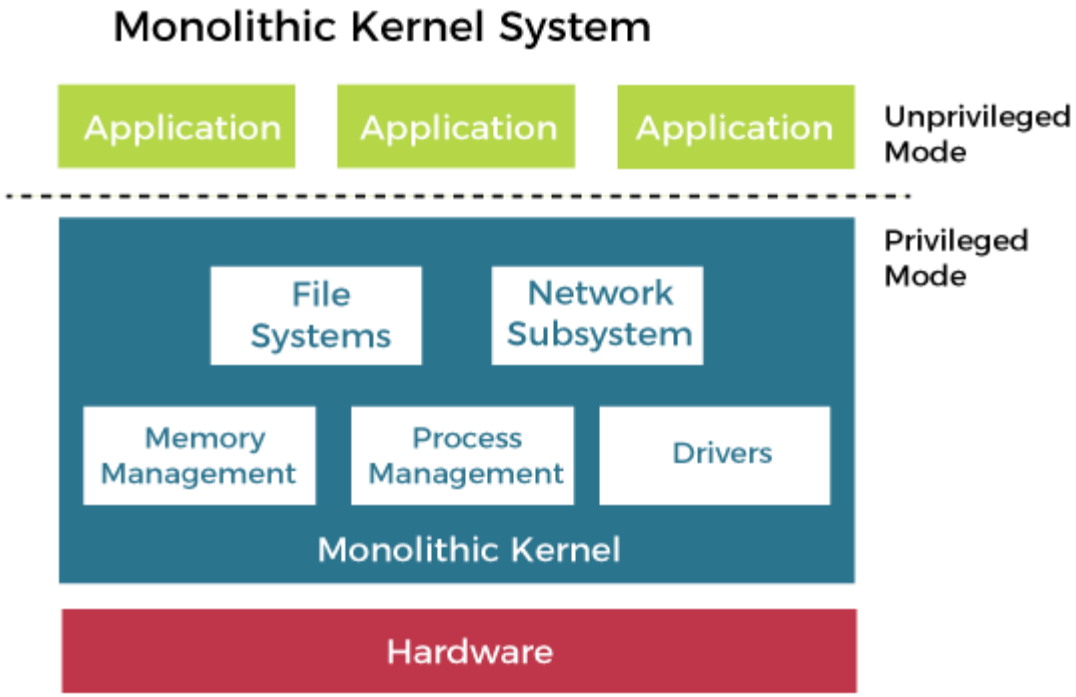
- Everything below the system-call interface and above the physical hardware is the kernel.
- The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls and all these functionalities are combined into one level.
- This monolithic structure was difficult to implement and maintain.

Monolithic Structure

The monolithic operating system is a very basic operating system in which file management, memory management, device management, and process management are directly controlled within the kernel. The kernel can access all the resources present in the system. In monolithic systems, each component of the operating system is contained within the kernel. Operating systems that use monolithic architecture were first used in the 1970s.

The monolithic operating system is also known as the monolithic kernel. This is an old operating system used to perform small tasks like batch processing and time-sharing tasks in banks. The monolithic kernel acts as a virtual machine that controls all hardware parts.
It is different from a microkernel, which has limited tasks. A microkernel is divided into two parts, kernel space, and user space. Both parts communicate with each other through IPC (Inter-process communication). Microkernel's advantage is that if one server fails, then the other server takes control of it.

Monolithic kernel
A monolithic kernel is an operating system architecture where the entire operating system is working in kernel space. The monolithic model differs from other operating system architectures, such as the microkernel architecture, in that it alone defines a high-level virtual interface over computer hardware.



Advantages of Monolithic Architecture:

Monolithic architecture has the following advantages, such as:

- Simple and easy to implement structure.
- Faster execution due to direct access to all the services

Disadvantages of Monolithic Architecture:

Here are some disadvantages of monolithic architecture:

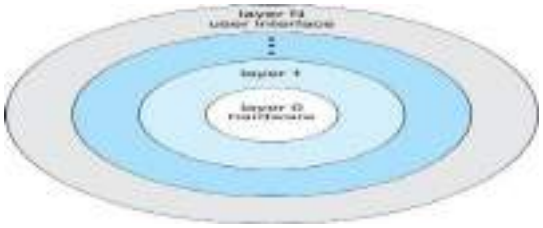
- The addition of new features or removal of obsolete features is very difficult.
- Security issues are always there because there is no isolation among various servers present in the kernel.

Layered Approach

- In the Layered approach the operating system is broken into a number of layers (levels).
- The **bottom layer 0** is the Hardware and the **highest layer *N*** is the User interface.
- An operating-system layer consists of data structures and a set of routines that can be invoked by higher-

level layers and can also invoke operations on lower-level layers.

- An operating-system layer is an implementation of an abstract object made up of data and the operations that can manipulate those data.



Advantages of Layered Approach

The main advantage of the layered approach is the simplicity of construction and debugging.

- Each layer uses functions (operations) and services of only lower-level layers.
- This approach simplifies debugging and system verification.
- The first layer can be debugged without any concern for the rest of the system because it uses only the basic hardware to implement its functions.
- Once the first layer is debugged and functions correctly while the second layer is debugged and so on.
- If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged.

Another advantage of a Layered structure is Data Hiding:

- Each layer is implemented only with operations provided by lower-level layers.
- A layer does not need to know how these operations are implemented instead a layer just knows what these operations do.
- Hence, each layer hides the existence of certain data structures, operations and hardware from higher-level layers.

Problems with Layered Approach

Major difficulty with the layered approach involves appropriately defining the various layers.

- A layer can use only lower-level layers hence we have to plan carefully which layer to be kept in which place.
- **Example:** The device driver for the backing store (i.e. disk space used by virtual-memory algorithms) must be at a lower level than the memory-management routines because memory management requires the ability to use the backing store.
- The backing-store driver would normally be above the CPU scheduler because the driver may need to wait for I/O and the CPU can be rescheduled during this time.

Another problem with layered implementations is that they tend to be less efficient.

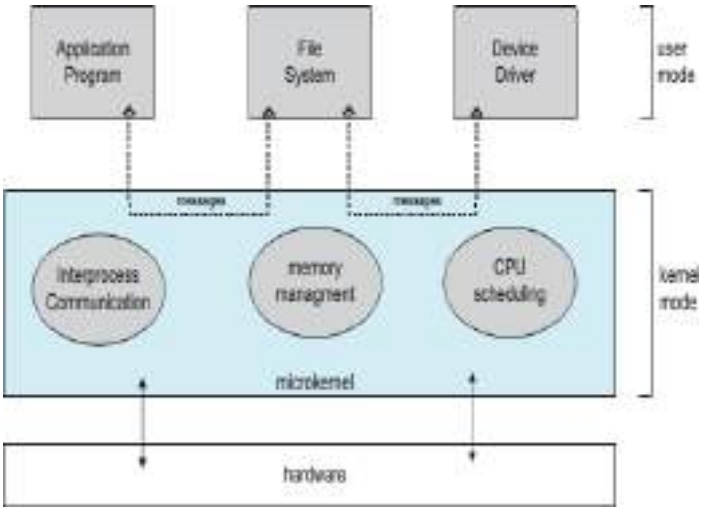
- When a user program executes an I/O operation, it executes a system call that is trapped in the I/O layer, which calls the memory-management layer, which in turn calls the CPU- scheduling layer, which is then passed to the hardware.
- At each layer, the parameters may be modified, data may need to be passed, and so on. Each layer adds overhead to the system call.

- The net result is a system call that takes longer time than a non-layered system.

Microkernels

As the UNIX OS was expanded, the kernel also became large and difficult to manage.

- In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called **Mach** that modularized the kernel using the **Microkernel** approach.
- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs resulting in a smaller kernel (i.e.) Microkernel.
- There is little consensus regarding which services should remain in the kernel and which services should be implemented in user space.
- Microkernels provide minimal process and memory management and also communication facilities.
- The main function of the microkernel is to provide **Message Passing Communication** between the client program and the various services that are also running in the user space.



For example: if the client program wishes to access a file, it must interact with the file server. The client program and service never interact directly. Rather, they communicate indirectly by exchanging messages with the microkernel.

Advantages of Microkernel

Microkernel approach makes extending the operating system easier.

- All new services are added to user space and consequently do not require modification of the kernel.
- Even-though though the kernel must have to be modified, the changes in the kernel will be very few, because the microkernel is a smaller kernel.
- The resulting operating system is easier to port from one hardware design to another.
- The microkernel also provides more security and reliability, since most services are running as user processes rather than kernel processes.
- If a service fails, the rest of the operating system remains untouched.

The disadvantage of Microkernel

- The performance of microkernels can suffer due to increased system-function overhead.

Modules

Loadable Kernel Modules are the current best methodology for operating-system design.

- The kernel has a set of core components and links in additional services via modules, either at boot time or during run time.
- This type of design is common in modern implementations of UNIX such as Solaris, Linux, and Mac OS X as well as Windows.
- The idea of the design is for the kernel to provide core services while other services are implemented dynamically as the kernel is running.
- Linking services dynamically is preferable to adding new features directly to the kernel, which would require recompiling the kernel every time a change was made.
- Example: We might build CPU scheduling and memory management algorithms directly into the kernel and then add support for different file systems by way of loadable modules.
  The below figure shows the Solaris operating system structure is organized around a core kernel with seven types of loadable kernel modules:

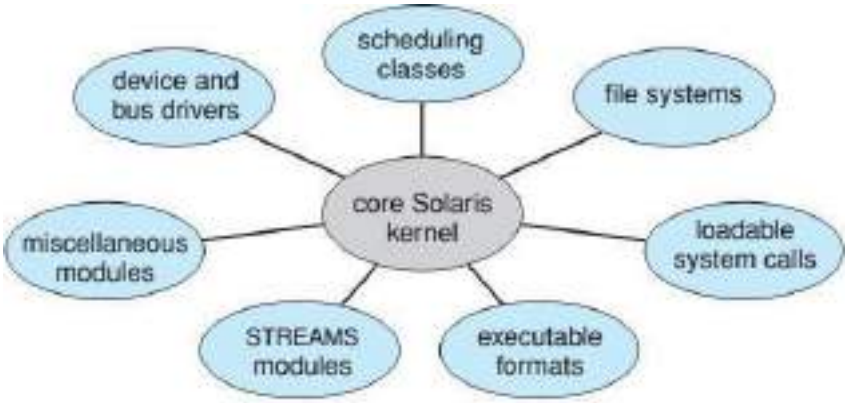


Figure 2.15 Solaris loadable modules.

Hybrid Systems

In practice, most operating systems combine different structures, resulting in hybrid systems that address performance, security, and usability issues.
Example:

1. Linux and Solaris are monolithic and modular.
2. Windows is largely monolithic and some part of it behaves as a microkernel. Windows systems also provide support for dynamically loadable kernel modules.

Operating Systems that support the structure of Hybrid systems are:

- Mac OS X
- iOS
- Android

Mac OS X

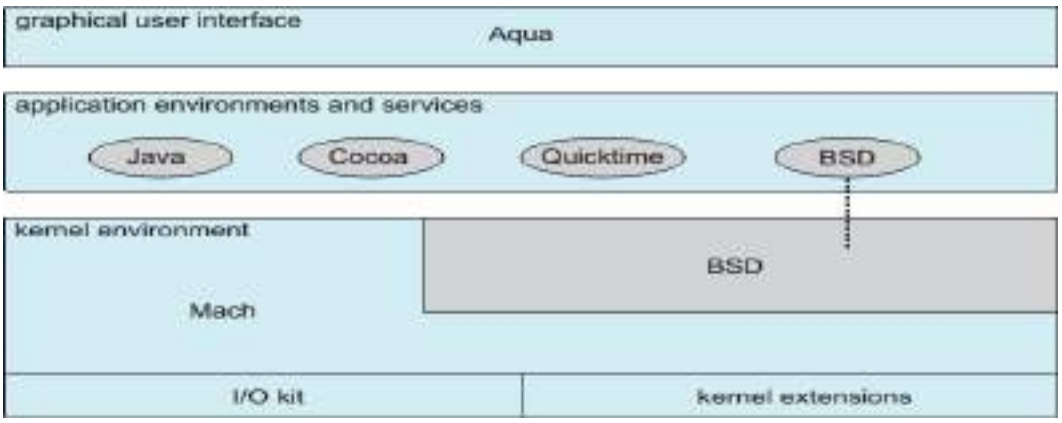
The Apple Mac OS X operating system uses a hybrid structure and it is a layered system.

- The top layers include the ***Aqua*** user interface and a set of application environments and services.
- The **Cocoa** environment specifies an API for the Objective-C programming language, which is used for writing Mac OS X applications.
- Below these layers is the **Kernel Environment**, which consists primarily of the Mach microkernel and the BSD UNIX kernel.
- Mach microkernel provides memory management, support for remote procedure calls (RPCs), and inter-process communication (IPC) facilities including message passing, and thread scheduling.
- The BSD component provides a BSD command-line interface, support for networking and file systems, and an implementation of POSIX APIs including Pthreads.
- The kernel environment provides an I/O kit for the development of device drivers and dynamically loadable modules.
- The BSD application environment can make use of BSD facilities directly.

iOS

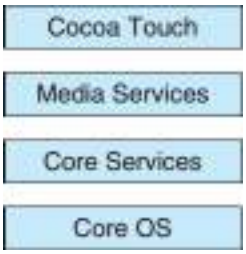
is a mobile operating system designed by Apple to run its **iPhone** and **iPad**.

- **Cocoa Touch** is an API for Objective-C that provides several frameworks for developing applications that run on iOS devices.



- The fundamental difference between Cocoa and Cocoa Touch is that Cocoa Touch provides support for hardware features unique to mobile devices such as touch screens. The **media services** layer provides services for graphics, audio, and video.
- The **Core services** layer provides a variety of features, including support for cloud computing and databases.

- The bottom layer represents the core operating system, which is based on the kernel environment.



Android

The Android operating system was developed for Android smartphones and tablet computers. It was designed by the Open Handset Alliance led by Google.

- Android is a layered stack of software that provides a rich set of frameworks for developing mobile applications.
- At the bottom of this software stack is the Linux kernel. Linux is used primarily for process, memory, and device-driver support for hardware and has been expanded to include power management.

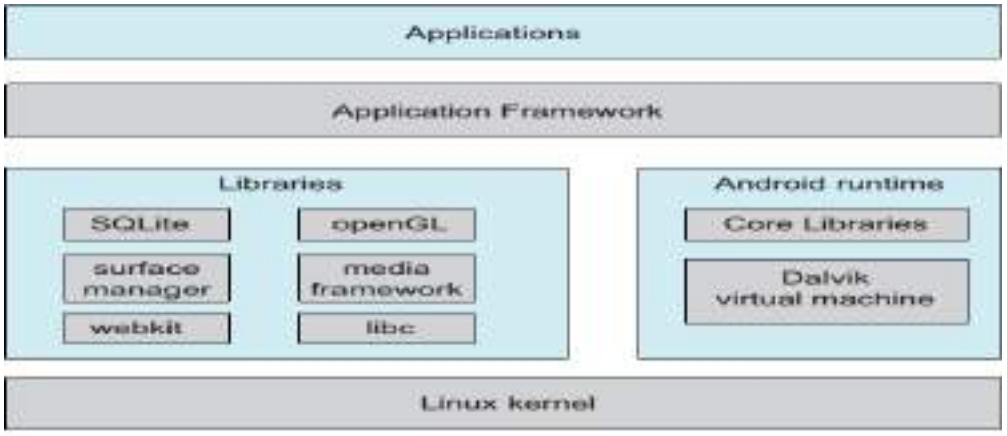


Figure 2.18 Architecture of Google's Android.

- The Android runtime environment includes a core set of libraries as well as the Dalvik virtual machine.
- Software designers for Android devices develop applications in the Java language and Google has designed a separate Android API for Java development.
- The Java class files are first compiled to Java byte-code and then translated into an executable file that runs on the Dalvik virtual machine.
- The Dalvik virtual machine was designed for Android and is optimized for mobile devices with limited memory and CPU processing capabilities.
- The set of libraries available for Android applications includes frameworks for developing web browsers (webkit), database support (SQLite), and multimedia.
- The libc library is similar to the standard C library but is much smaller and has been designed for the slower CPUs that characterize mobile devices.

Microkernel vs Monolithic Kernel :

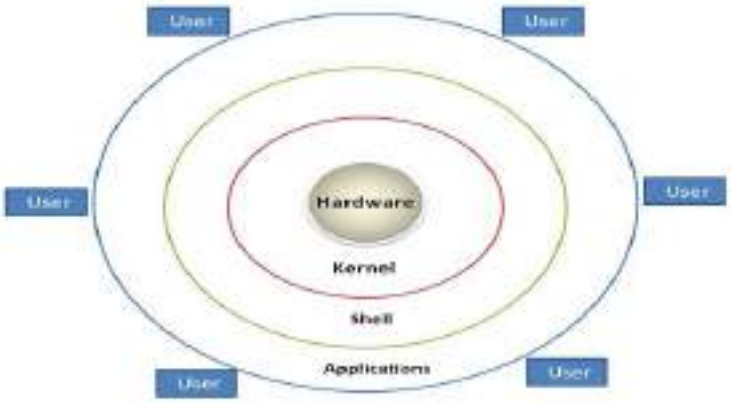
Microkernel	**Monolithic kernel**
In microkernel user services and kernel, services are kept in separate address spaces.	In monolithic kernels, both user services and kernel services are kept in the same address space.
The OS is complex to design.	The OS is easy to design and implement.
Microkernels are smaller in size.	A monolithic kernel is larger than a microkernel.
Easier to add new functionalities.	Difficult to add new functionalities.
To design a microkernel, more code is required.	Less code when compared to microkernel

Failure of one component does not affect the working of the microkernel.	Failure of one component in a monolithic kernel leads to failure of the entire system.
The execution speed is low.	The execution speed is high.
It is easy to extend the Microkernel.	It is not easy to extend a monolithic kernel.
Example: Mac OS X.	**Example:** Microsoft Windows 95.

Linux Architecture

Linux is a free and open-source operating system that was developed in the early 1990s by Linus Torvalds. It is based on the Unix operating system and has become a popular choice for both personal and enterprise use due to its stability, security, and flexibility.

Linux is built on a modular architecture, which means that it is made up of a number of different components that work together to form a complete operating system. These components are organized into layers, each of which serves a specific purpose and interacts with the other layers to provide the functionality that users expect from an operating system.



Linux System Architecture

The following is a high-level overview of the main layers of the Linux architecture:

- **Hardware layer**: This is the bottommost layer of the Linux architecture and represents the physical hardware components of the computer, such as the processor, memory, and storage. The hardware layer is responsible for interacting with the various hardware devices and providing access to them for the rest of the operating system.

- **Kernel layer**: The kernel is the core of the operating system and is responsible for managing the resources of the computer, such as the CPU, memory, and I/O devices. It also provides services to the other components of the operating system and acts as the intermediary between the hardware and the software layers.

The kernel is the core part of the operating system, which is responsible for all the major activities of the LINUX operating system. This operating system consists of different modules and interacts directly with the underlying hardware. The kernel offers the required abstraction to hide application programs or low-level hardware details to the system.

The types of Kernels are as follows:

- Monolithic Kernel
- Microkernels
- Exo kernels
- Hybrid kernels

SHELL: Shell is an environment in which we can run our commands, programs, and shell scripts. It is a user interface for access to an operating system's services

1. Shell is the interface between the user and kernel.
2. It is the outer part of the operating system.

3. A shell is a user interface for access to an operating system's services Shell is an environment in which we can run our commands, programs, software, and shell scripts.
4. Computers do not have any inherent capability of translating commands into actions, it is done by Shell.
5. There can be many shells in action - one shell for each user who logged in.

How Shell Works

When we enter commands through the keyboard, it gathers input from you and executes programs based on that input. When a program finish executing, it displays that program's output. OR the shell thoroughly examines the keyboard input for special characters. If it finds any, it rebuilds a simplified command line and finally communicates with the Kernel to see that the command is executed. To know the running shell, use the echo $SHELL command in the terminal.

Application programs/software

An application, or application program, is a software program that runs on your computer. It is exciting for users. Some inbuilt application programs in Linux are terminal, Firefox browser, and Libre office

System calls

There are over a thousand commands available in the Linux operating system. These all commands use a function to communicate with the kernel - and it is called a system call. A system call is the interface between a process and an operating system or System calls are the only entry points into the kernel system

Features

The main **features of the Linux operating system** are

Portable: Linux operating system can work on different types of hardware as well as Linux kernel supports the installation of any kind of hardware platform.

Open Source: The source code of the LINUX operating system is freely available and, to enhance the ability of the LINUX operating system, many teams work in collaboration.

Multiuser: Linux operating system is a multiuser system, which means, multiple users can access the system resources like RAM, Memory, or Application programs at the same time.

Multiprogramming: Linux operating system is a multiprogramming system, which means multiple applications can run at the same time.

Hierarchical File System: Linux operating system affords a standard file structure in which system files or user files are arranged.

Shell: Linux operating system offers a special interpreter program, that can be used to execute commands

of the OS. It can be used to do several types of operations like call application programs, and so on.

Security: Linux operating system offers user security systems using authentication features like encryption of data or password protection or control access to particular files.

How Does Linux Different from Other OS?

There are several features of the Linux OS that demonstrate that it is superior as compared to other OS. But some other OS can be more helpful than Linux. The main major advantages of the Linux system include the following and that will decide why it is superior as compared to other operating systems.

- Open Source
- Heavily Documented for beginners
- Security
- Multiple Desktop Support
- Multitasking

  Free
- Installation
- Lightweight

What is vi?

The default editor that comes with the UNIX operating system is called vi (visual editor). [Alternate editors for UNIX environments include Pico and emacs, a product of GNU.]

The UNIX vi editor is a full-screen editor and has two modes of operation:

1. *Command mode* commands which cause action to be taken on the file, and
2. *Insert mode* in which entered text is inserted into the file.

In the command mode, every character typed is a command that does something to the text file being edited; a character typed in the command mode may even cause the vi editor to enter the insert mode. In the insert mode, every character typed is added to the text in the file; pressing the <Esc> (*Escape*) key turns off the Insert mode.

While there are a number of vi commands, just a handful of these is usually sufficient for beginning vi users. To assist such users, this Web page contains a sampling of basic vi commands. The most basic and useful commands are marked with an asterisk (* or star) in the tables below. With practice, these commands should become automatic.

NOTE: Both UNIX and vi are case-sensitive. Be sure not to use a capital letter in place of a lowercase letter; the results will not be what you expect.

To Get into and Out Of vi

To Start vi

To use vi on a file, type in vi filename. If the file named filename exists, then the first page (or screen) of the file will be displayed; if the file does not exist, then an empty file and screen are created into which you may enter text.

* vi filename *edit filename starting at line 1*

  vi -r filename *recover filename that was being edited when system crashed*

To Exit vi

Usually, the new or modified file is saved when you leave vi. However, it is also possible to quit vi without saving the file.

Note: The cursor moves to the bottom of the screen whenever a colon (:) is typed. This type of command is completed by hitting the <Return> (or <Enter>) key.

* :x<Return> *quit vi, writing out modified file to file named in original invocation*

  :wq<Return> *quit vi, writing out modified file to file named in original invocation*

  :q<Return> *quit (or exit) vi*

* :q!<Return> *quit vi even though latest changes have not been saved for this vi call*

To save and quit

You can save and quit vi editor from command mode. Before writing save or quit command you have to press the colon (:). Colon allows you to give instructions to vi.

exit vi table:

Commands	Action
:wq	Save and quit
:w	Save
:q	Quit
:w fname	Save as fname

ZZ	Save and quit
:q!	Quit discarding changes made
:w!	Save (and write to a non-writable file)

To exit from vi, first ensure that you are in command mode. Now, type:wq and press enter. It will save and quit vi. Type :wq to save and exit the file.

Linux Commands:

A shell is a command-line interpreter that reads user input and executes commands. The user input to a shell is normally from the terminal (an interactive shell) or sometimes from a file (called a shell script).

Below is a list of few commonly used Linux commands.

1. **man**: manual display for any command

   man mv **[** press q to close the manual page]

2. **pwd:** present working directory

   pwd **[** displays the current directory path]

3. **date:** display date, time, time zone etc

   date **[** displays the current directory path]

4. **who:** get information about currently logged in user on to system

who **[** Login name of the users, Terminal line numbers, Login time of the users in to system, Remote host name of the user]

Directory Handling Commands:

1. **mkdir:** used to create one or more directories

   mkdir book [Creates the directory named book]

   mkdirdbs doc dmc [Creates three directories]

2. **cd:** used to change the directory

   cd book [changes the access to the directory named book]

3. **rmdir**: remove directories [Removes only empty directories]

   Eg: rmdir book [removes directory named book if it is empty]

   rmdirdbs doc dmc [Removes 3 directories]

File Handling Commands:

1. **cp:** Copying Files - To create an exact copy of a file

   cp [-option] source destination

   Eg: cp file1 file2 [Here file1 is copied to file2.]
   cp file1 file2 dir [File1 file2 are copied to dir.]
   cp -i file1 file2 [turns to interactive when –i option is used]
   overwrite file2 (yes/no)?
   Y at this prompt overwrites the file.

2. **mv**: Moving and Renaming Files - Used to rename the files/directories.

   Eg: mv test sample [Here test is renamed as sample.]

3. **rm**: Deleting Files and Directories

   Eg: rm mylist.txt [will delete “mylist.txt”]

With –i option removes the files interactively.

rm –i file1

With –r option recursively removes directories.

rm –r dir1

4. **touch:** creates more than one empty files

   Eg: touch a.txt b.txt [will delete "mylist.txt"]

5. **ls:** list files and directories

   ls [-option] [path]

   Eg:

   ls [Displays the contents of the current working directory]

   ls –a[Displays the hidden files and directories of the current working directory]

   ls –l[long listing one per line – display complete information about file]

   ls –i [Display the file index number]

   ls –d [display only subdirectories]

   ls –r [display the list in reverse order

   ls –R [display the contents of the subdirectories also]

6. **Cat displays** the content of a file, copy content from one file to another, concatenate the contents of multiple files etc

   Eg : cat myfile [Display the contents of the file named myfile]

   cat>newfile [Will create a file named newfile – type required text and press Ctrl+D to quit]

   cat file1 > file2 [copy content of file1 to file2]

   cat [file1 file2 so on] > target [merge contents of multiple files and copy to new file]

Few More Commands:

1. **uname :** displays the information about the system

   uname [OPTION]

Eg: uname –a [display Kernel name, network node, hostname, kernel release date, kernel version, machine hardware name, hardware platform, operating system]

uname –s [display Kernel name]

uname –n [display Hostname of the network node]

uname –r [display the kernel release data]

uname –v [display version of the current kernel]

uname –p [display the type of processor]

2. **passwd:** Change user password for user account

passwd [options] [LOGIN]

A normal user may only change the password for his/her own account, while the super-user may change the password for any account.

passwd -S <username> **[** displays the status of user account password settings.]

passwd -d <username> **[**delete a password for an account.]

passwd -e <username> [expire a password for an account. The user will be forced tochange the password during the next login attempt]

passwd -n <no of days><username>

[This sets the number of days before a password can be changed. By default, a value of zero is set, which indicates that the user may change their password at any time]

3. **script :** is used to make typescript or record all the terminal activities

After executing the *script* command, it starts recording everything printed on the screen including the inputs and outputs until exit. By default, all the terminal information is saved in the file *typescript* , if no argument is given.

script [options] [file]

script **[** automatically create a file *typescript* in home directory]

Note: We need to execute *exit* command and script will stop the capturing process.

script screenrecord.txt **[**save recording in a file named *screenrecord.txt*]

4. **printf:** used to display the given string, number or any other format specifier on the terminal window. *printf can have format specifiers, escape sequences or ordinary characters.*

printf [-v var] format [arguments]

Eg: printf "Hello, World!“

printf "%s\n" "Hello, World!“

5. **echo :**used to display line of text/string that are passed as an argument

echo [option] [string]

Eg: echo "Hello, World!“

echo “%s\n” “ Hello World”

echo -e “ \b Hello \b World!“ [-b removes blank spaces between the words]

6. **tar:** Archive file – file composed of one or more files and metadata

a. Easily portable and storage
b. Simple to compress
c. Less Storage
d. tar is used to archive as well as extract the archive

tar [options] [archive-name] [file or folder to be archived]

Options: -c :Create Archive

-x :Extract Archive

-v : Display verbose information

-f : Create archive with given filename

tar –cvfmytar [directory-name]

7. **Gzip:** Compresses Files

Each single file is compressed into a single file. The compressed file consists of a GNU zip header and deflated data. gzip compresses the file, adds a “.gz” suffix, and deletes the original file.

gzip [Options] [filenames]

Eg: gzip mydoc.txt [create a compressed file of mydoc.txt named as mydoc.txt.gz and delete the original file]

Gzip command is used as a combination with tar to compress the size of the tar file)

- Tar is an archiver, meaning it would archive multiple files into a single file but without compression.
- Gzip which handles the .gz extension is the compression tool that is used to reduce the disk space used by the file.

tar –zcf filename.tar.gz directory/filename

To view the size of the files use **ls –lh** command.

- **PATH command:**
- PATH is an environment variable that instructs a Linux system in which directories to search for executables. The PATH variable enables the user to run a command without specifying a path.
- When a user invokes a command in the terminal, the system executes a program. Therefore, Linux has to be able to locate the correct executable. PATH specifies program directories and instructs the system where to search for a program to run. To print all the configured directories in the system's PATH variable, run the echo command:
- echo $PATH
- 

- The output shows directories configured in PATH by default. The printenv command delivers the same output:
- printenv PATH
- 

- Furthermore, running on a certain command shows where its executable is. For instance, execute which with whoami:
- which whoami
- 

- The output shows that the executable for whoami is located in the */usr/bin/* directory.