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.

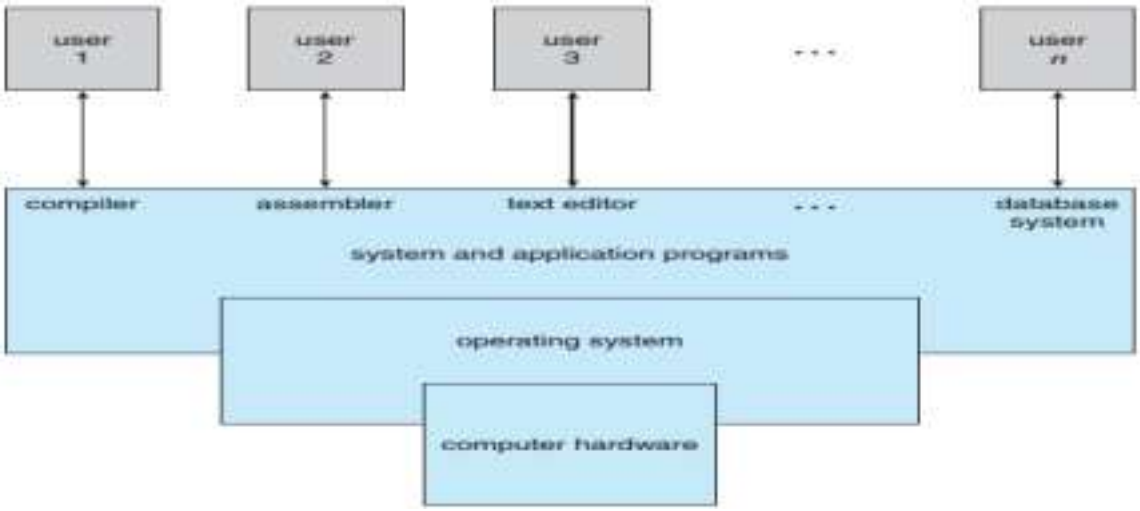
· Mainframe operating systems are designed primarily to optimize utilization of hardware. · Personal computer (PC) operating systems support complex games, business applications and everything in between.

· Mobile computer operating systems provides 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 Central processing unit (CPU), Memory and the Input/Output (I/O) devices provides 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 system provides 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 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 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 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 which are not associated with the operation 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), EEPROM are considered as firmware.

COMPUTER-SYSTEM ORGANIZATION

Computer system organization has several parts:

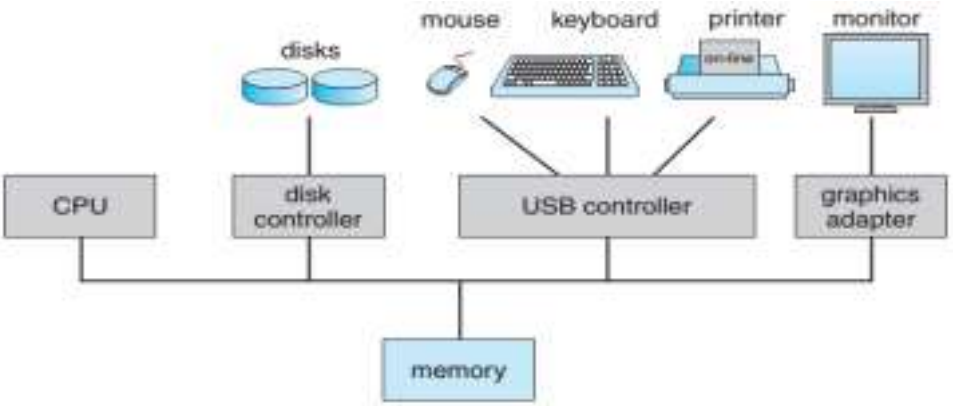
1. Computer system operation
2. Storage structure
3. I/O structure

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 device 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 **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 executing, it can start providing 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 call**.

· When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location. The fixed location usually the starting address where the service routine for the interrupt is located.

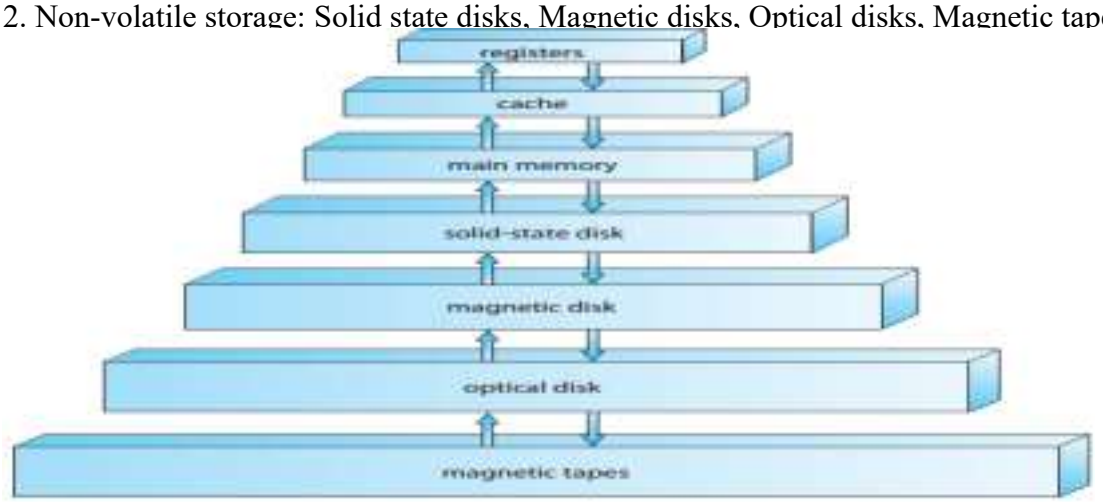
· The interrupt service routine executes and after completion the interrupt, the CPU resumes the interrupted computation.

· Interrupt vector is an array which 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 storages:

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



Volatile storage

Volatile storage devices are fast and expensive but they loses its 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 main memory to an internal register within the CPU. Store instruction moves the content of a register to main memory. · **Cache:** It is faster storage system used to store the information temporarily and the size of cache is very small. It is located between CPU and Main Memory.

· **Main Memory:** It is also called as 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. Main memory is too small to store all needed programs and data permanently.

Non-volatile storage or Secondary storage devices

These are extension of main memory. These devices stores large quantities of data permanently.

Read-Only Memory (ROM) stores static programs such as bootstrap program. 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 contains 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 RAM to the magnetic disk. When external power is restored, the controller copies the data back 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)**, in robots and general-purpose computers. Flash memory is slower than DRAM but needs no power to retain its contents.

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

I/O Structure

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

· Each device controller is in charge of a specific type of devices. 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.

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 runs 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 bus, 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 increases.

ii. **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 system if the processor fails the entire system fails, but in multi-processor system if processor fails then the entire system will not fail, other processor will pick up and shares the work of failed processor.

Multi-processor systems are of two types:

1. Asymmetric multiprocessing
2. Symmetric multiprocessing

Asymmetric Multi-Processing:

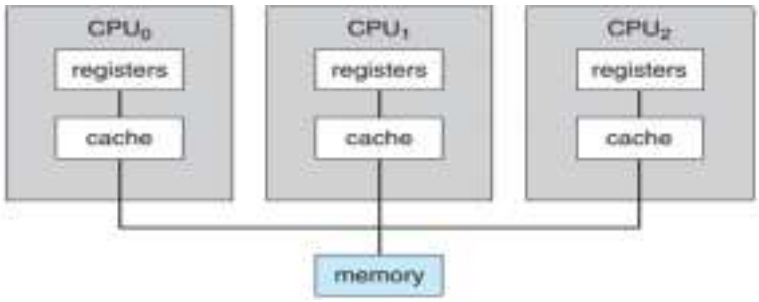
· Each processor is assigned a specific task in asymmetric multiprocessing and it uses boss–worker relationship.

· The Boss processor controls the system by scheduling and assigning the tasks to worker processor. The worker processor has to complete the task assigned by 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.

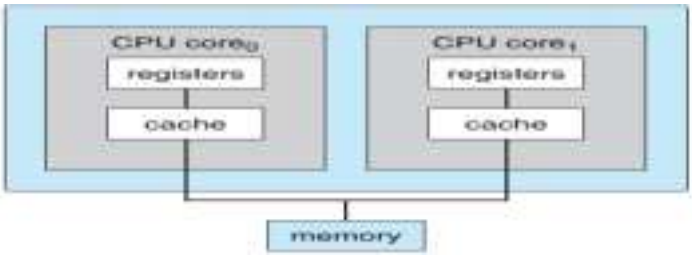
· In SMP if there are N CPU'S then N processes can run without causing performance degradation.

· It is the most common used system. All modern operating systems including Windows, Mac OS X and Linux provide support for SMP.

Multicore processors

· Multicore processors are special type 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 dual-core design with two cores on the same chip



Blade processors

· In Blade servers multiple processor boards, I/O boards and networking boards are placed in the same chassis.

· The difference between Blade processors and traditional multiprocessor systems is that each blade-processor board boots independently and runs its own operating system. **Memory Access models**

· Multiprocessing can cause a system to change its memory access model from uniform memory access (**UMA**) to non-uniform memory access (**NUMA**).

· UMA is defined as the situation in which access to any RAM from any CPU takes the same amount of time.

· In NUMA, some parts of memory may take longer to access than other parts, NUMA creates a performance issues.

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 allow 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. Operating system needs two modes of operations (Dual Mode):

i. User mode

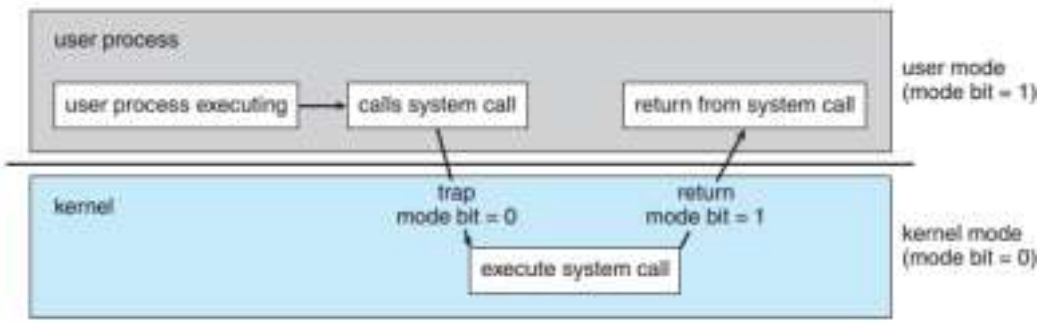
ii. Kernel mode or Supervisor mode or System mode or Privileged mode. · Mode bit is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1).

· 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 **Interrupt 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 that 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 to 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 Windows7, 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, 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 completes.

· 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 process executes system code whereas user processes executes 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 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 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 program 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, Disk scheduling. **Caching**

Cache is faster storage system used to store the information temporarily and the size of 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 cache then we use the information directly from the cache. · If information is not cache, 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 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

· Drivers 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 on 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 still system 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.

COMPUTING ENVIRONMENTS

There are several computing environment are built for different computing purposes: 1. Traditional computing
2. Mobile computing
3. Distributed computing
4. Client server computing
5. Peer-to- Peer computing
6. Virtualization
7. Cloud computing
8. Real time embedded system

Traditional computing

Traditional computing environment consisted of PC's connected to a network, with servers providing file and print services.

· In traditional computing, for a period of time, systems were either batch systems or interactive systems.

· Batch systems processed jobs in bulk, with predetermined input from files or other data sources.

· Interactive systems waited for input from users.

· To optimize the use of the computing resources, multiple users shared time on these systems.

· Time-sharing systems used a timer and scheduling algorithms to cycle processes rapidly through the CPU by giving each user a share of the resources.

· Web technologies and increasing Wide Area Network (i.e. Internet) bandwidth are stretching the boundaries of traditional computing.

· Companies establish **portals**, which provide Web accessibility to their internal servers. · **Network computers** are essentially terminals that understand web-based computing are used in place of traditional workstations where more security or easier maintenance is desired.

· Mobile computers can also connect to **wireless networks** and cellular data networks to use the company's Web portal.

· At home most of the users once had a single computer with a slow modem connection to the office, the Internet, or both.

Mobile Computing

Mobile computing refers to computing on handheld smartphones and tablet computers. · These devices share the distinguishing physical features of being portable and lightweight.

· As compared with desktop and laptop computers, mobile systems gave up screen size,

memory capacity and overall functionality in return for handheld mobile access to services such as e-mail and web browsing.

· The memory capacity and processing speed of mobile devices are more limited than those of PCs.

· In mobile devices the power consumption is less and use processors that are smaller and slower and offer fewer processing cores.

· **Apple iOS** and **Google Android** operating systems are dominating mobile computing. · Mobile systems are used for e-mail and web browsing, playing music and video, reading digital books, taking photos, recording high-definition video, Global positioning system (GPS) chips, accelerometers and gyroscopes.

Distributed Systems

It is a collection of physically separate, possibly heterogeneous, computer systems that are networked to provide users with access to the various resources that the system maintains. · Access to a shared resource increases computation speed, functionality, data availability and reliability.

· A **network** is a communication path between two or more systems. ·
Distributed systems depend on networking for their functionality.

· **TCP/IP** is the most common network protocol of the Internet.

Network Operating System is an operating system that provides features such as file sharing across the network, along with a communication scheme that allows different processes on different computers to exchange messages.

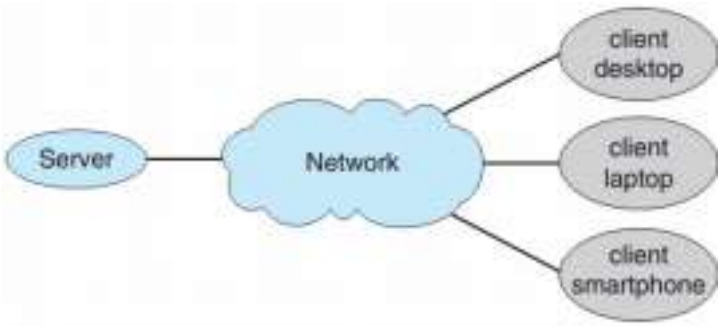
Distributed Operating System provides an autonomous environment. The different computers communicate closely enough to provide the illusion that only a single operating system controls the network.

Client–Server Computing

Client –Server computing is a Centralized system in which Server systems are used to satisfy requests generated by **client systems**. PC's and mobile devices are connected to the Centralized system.

Server systems can be categorized into 2 types: Compute servers and File servers: · **Compute-Server System** provides an interface to which a client can send a request to perform an action (for example, read data). In response, the server executes the action and sends the results to the client. A server running a database that responds to client requests for data is an example of such a system.

· **File-Server System** provides a file-system interface where clients can create, update, read and delete files. **Ex:** A web server that delivers files to clients running web browsers.



Peer-to-Peer Computing

The peer-to-peer (P2P) system model is a distributed system. In this model, clients and servers are not distinguished from one another.

· In Peer-to-Peer computing all nodes within the system are considered peers and each may act as either a client or a server depending on whether it is requesting or providing a service.

· Peer-to-peer systems offer an advantage over traditional client-server systems. In a client server system, the server is a bottleneck; but in a peer-to-peer system, services can be provided by several nodes distributed throughout the network.

· To participate in a peer-to-peer system, a node must first join the network of peers. · Once a node has joined the network, it can begin providing services to other nodes in the network and requesting services from other nodes in the network.

Services in Peer-to-Peer computing:

1. **Centralized Lookup Service:**
   When a node joins a network, it registers its service with a centralized lookup service on the network. Any node desiring a specific service first contacts this centralized lookup service to determine which node provides the service. The remainder of the communication takes place between the client and the service provider.
2. **Non-Centralized Lookup Service:** It uses no centralized lookup service. A peer acting as a client must discover which node provides a desired service by broadcasting a request for the service to all other nodes in the network. The node providing that service responds to the peer making the request.
   To support this approach, a **discovery protocol** must be provided that allows peers to discover services provided by other peers in the network.

Example: Skype uses a hybrid peer-to-peer approach used to make voice and video calls and also sends text messages by using Voice over IP (VoIP) technology.
It includes a centralized login server, but it also incorporates decentralized peers and allows two peers to communicate.

Virtualization

Virtualization is a technology that allows operating systems to run as applications within other operating systems.

· Virtualization allows an entire operating system written for one platform to run on another platform.

· With **virtualization** an operating system that is natively compiled for a particular CPU architecture runs within another operating system also native to that CPU. · Virtualization first came about on IBM mainframes as a method for multiple users to run tasks concurrently.

· Running multiple virtual machines allows many users to run tasks on a single user system.

Example: An Apple laptop running Mac OS X on the x86 CPU can run a Windows guest to allow execution of Windows applications.

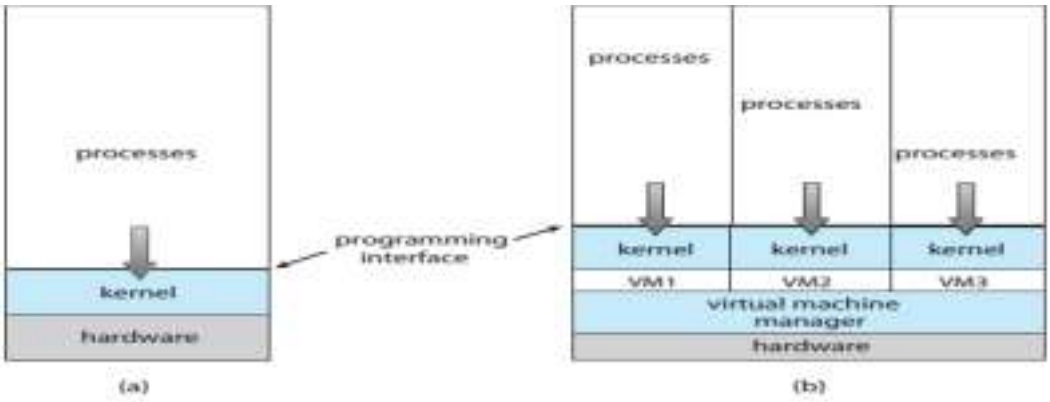


Figure 1.20 VMware.

VMware is a virtualization software application. If VMware is installed on Windows OS, then Windows is the **host** operating system and the VMware application was the virtual machine manager (VMM).

· If VMware application runs one or more guest copies of other operating systems such as Linux, then LINUX will be the guest operating system.

· The VMM runs the guest operating systems, manages their resource use and protects each guest from the others.

Cloud Computing

Cloud computing is a type of computing that delivers computing, storage and even applications as a service across a network.

· It is a logical extension of virtualization, because it uses virtualization as a base for its functionality.

· The Amazon Elastic Compute Cloud **(EC2)** facility has thousands of servers, millions of virtual machines and petabytes of storage available for use by anyone on the Internet.

· Based on how much of those resources the users use, they have to pay money on monthly basis.

The cloud is categorized into 3 ways: **Public cloud, Private cloud, Hybrid cloud**. · **Public** cloud is a cloud available via Internet to anyone willing to pay for the services. · **Private** cloud is a cloud run by a company for that company's own use. · **Hybrid** cloud is a cloud that includes both public and private cloud components. Cloud services can be categorized into 3 ways: **SaaS, PaaS, IaaS**.

· Software as a service **(SaaS)**—one or more applications available via the Internet. Example: word processors or spreadsheets

· Platform as a service **(PaaS)**—a software stack ready for application use via the Internet Example: A database server.

· Infrastructure as a service **(IaaS)**—servers or storage available over the Internet. Example: Storage available for making backup copies of production data. The cloud services and the cloud user interface are protected by a firewall.

Real-Time Embedded Systems

Embedded computers are the most prevalent form of computers in existence. · These devices are found everywhere from car engines and manufacturing robots to DVDs and microwave ovens.

· They tend to have very specific tasks. They have little or no user interface preferring to spend their time monitoring and managing hardware devices such as automobile engines and robotic arms.

· Embedded systems always run **Real-Time Operating Systems**.

· Real-time system is used as a control device in a dedicated application. · Sensors bring data to the computer. The computer must analyze the data and possibly adjust controls to modify the sensor inputs.

· A real-time system has well-defined, fixed time constraints. Processing must be done within the defined constraints, or the system will fail.

Example: Systems that control scientific experiments, home-appliance controllers, medical imaging systems, industrial control systems, automobile-engine fuel-injection systems and weapon systems are considered as real-time systems.

OPERATING-SYSTEM SERVICES

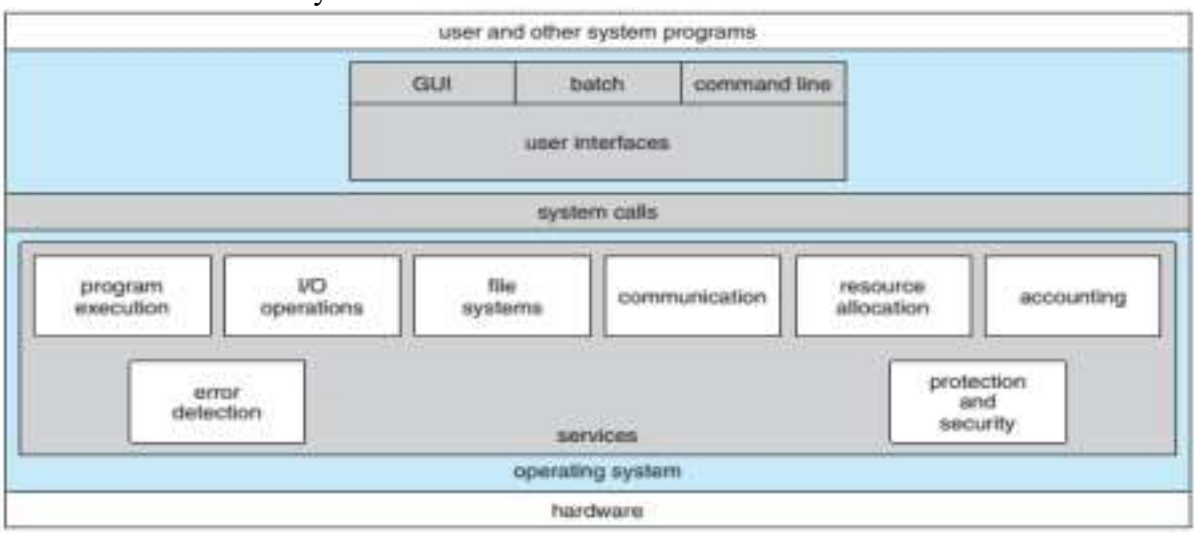
An Operating system provides an environment for the execution of programs. OS provides certain services to programs and to the users of those programs.

Operating system services are provided for the convenience of the programmer and to make the programming task easier.

The list of Operating system services that are helpful to the user:

1. User interface
2. Program Execution
3. I/O Operation
4. File system manipulation
5. Communication
6. Error Detection

7. Resource allocation
8. Accounting
9. Protection and Security



User Interface

Almost all operating systems have a **User Interface (UI)**. An 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 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. Example 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 permissions management to allow or deny access to files or directories based on file ownership.

Communications

· One process in its life time needs to communicate with another process to exchange information.

· Communication occurs between processes that are executing 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 code. I/O devices may have much more general request and release code.

· 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.

Accounting

· Operating system keeps track of different kind 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 are 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 username and password to gain access to system resources.

USER AND OPERATING-SYSTEM INTERFACE

There are two different types of interfaces are available:

1. Command Line Interface (CLI) or Command interpreter.
2. Graphical user interface (GUI)

Command Line Interface (CLI) or Command Interpreter

Command Line Interface allows users to directly enter commands to be performed by the operating system.

· Windows and UNIX treat the command interpreter as a special program that is running when a job is initiated. Command interpreters in UNIX or Linux are called as Shells. · Other operating systems include the command interpreter in the kernel. · UNIX and Linux systems uses different shells such as **Bourne** shell, **C** shell, **Bourne Again** shell, **Korn** shell etc.

· The main function of the command interpreter is to get and execute the next user specified command mostly to manipulate files: create, delete, list, print, copy, execute etc. · Programmers can add new commands to the system easily by creating new files with the proper names.

· Command-interpreter program does not have to be changed for new commands to be added.

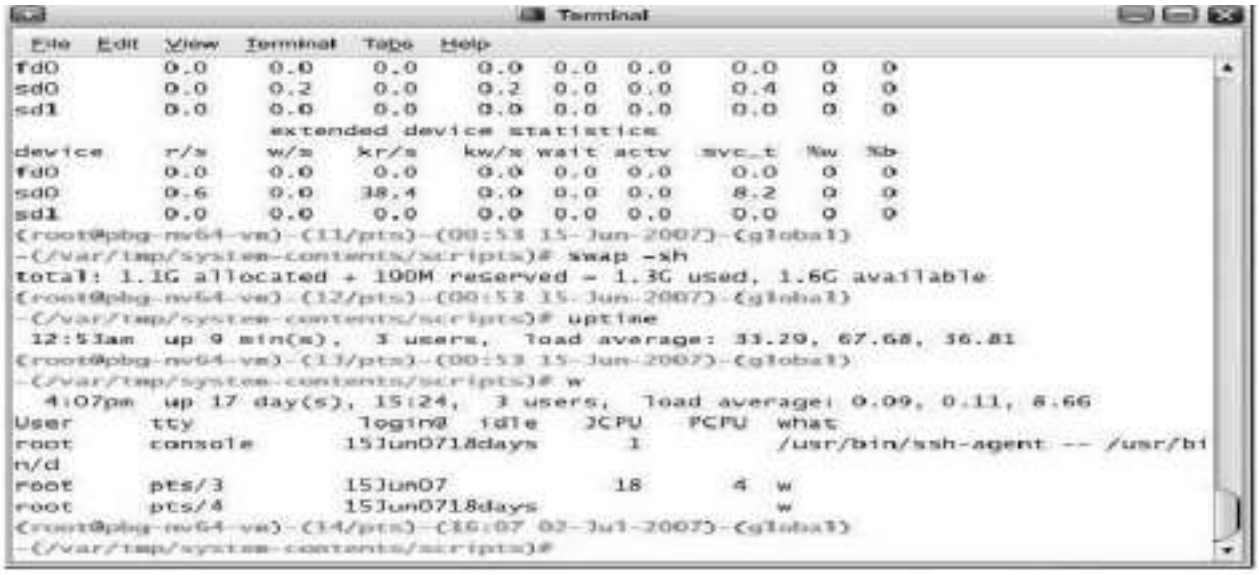


Figure 2.2 The Bourne shell command interpreter in Solrais 10.

These commands can be implemented in two ways:

1. The command interpreter itself contains the code to execute the command. A command to delete a file may cause the command interpreter to jump to a section of its code that sets up the parameters and makes the appropriate system call.
2. The command interpreter does not understand the command and then it merely uses the command to identify a file to be loaded into memory and executed.
   For example to delete a file in UNIX we use the following command: **rm file.txt** The command would search for a file called rm and load the file into memory and then execute it with the parameter **file.txt**.

The function associated with the **rm** command would be defined completely by the code in the file **rm**.

Graphical User Interfaces

Graphical user interface (GUI) is a user friendly interfacing with the operating system. · In GUI 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 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.

· The first Graphical user interfaces appeared on the Xerox Alto computer in 1973. · Graphical interfaces became more widespread with the advent of Apple Macintosh computers in the 1980s.

· Microsoft's first version of Windows Version 1.0 was based on the addition of a GUI interface to the MS-DOS operating system.

· Smartphones and handheld tablet computers uses a touchscreen interface. Users interact with touchscreen by pressing and swiping fingers across the screen.

· **K Desktop Environment** (**KDE**) and the **GNOME** desktop by the GNU project are GUI designs which run on Linux OS.

Figure 2.3 The iPad touchscreen.

SYSTEM CALLS

System calls provide an interface to the services made available by an operating system. · System calls are generally available as routines written in C and C++ and also written using assembly-language instructions.

· Each operating system has its own name for each system call.

· Application developers design programs according to an **Application Programming Interface (API)**.

· The API specifies a set of functions that are available to an application programmer including the parameters that are passed to each function and the return values the programmer can expect.

· The functions that make up an API typically invoke the actual system calls on behalf of the application programmer.

Example: The Windows function CreateProcess( ) which is used to create a new process actually invokes the NTCreateProcess( ) system call in the Windows kernel.

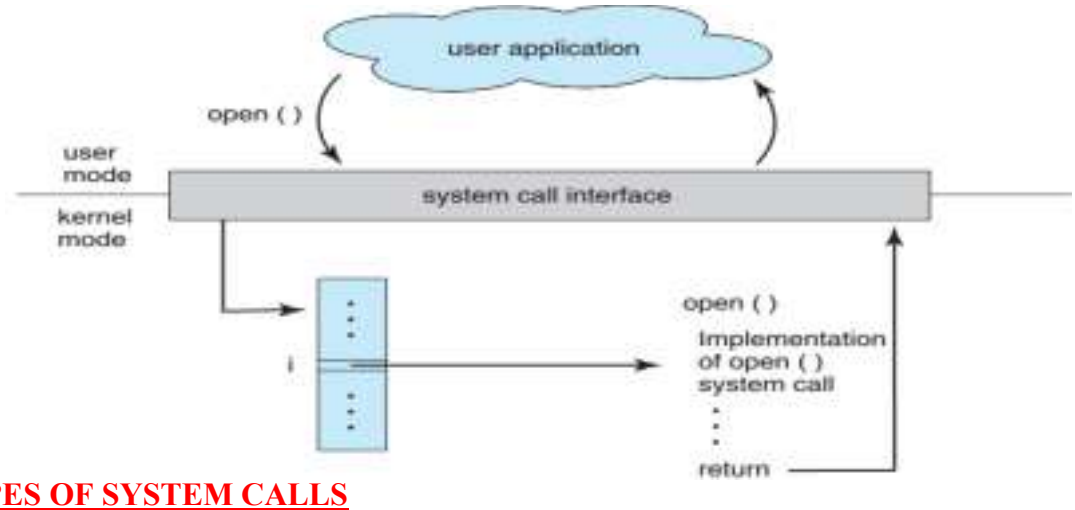
For most programming languages, the run-time support system (a set of functions built into libraries included with a compiler) provides a **System Call Interface** that serves as the link to system calls made available by the operating system.

· The system-call interface intercepts function calls in the API and invokes the necessary system calls within the operating system.

· A number is associated with each system call and the system-call interface maintains a table indexed according to these numbers.

· The system call interface then invokes the intended system call in the operating-system kernel and returns the status of the system call and any return values.

The below figure shows the relationship between an API, the system-call interface and the operating system for open( ) system call.



TYPES OF SYSTEM CALLS

System calls can be grouped into six major categories:

1. Process Control
2. File Manipulation
3. Device Manipulation
4. Information Maintenance
5. Communications
6. Protection

Process Control System Calls

The list of process control system calls are:

· **create process( )** is used to create a new process.

· **terminate process( )** is used to terminate the process that have already created. · **load( ) and execute( ):** A process or job executing one program may want to load( ) and execute( ) another program.

· **end( )**: It is used when a running program needs to be able to halt its execution normally. · **abort( )**: It is used when a running program needs to be able to halt its execution abnormally.

· **wait event( )** After creating new processes, it may need to wait for a certain amount of time to pass to finish its execution.

· **signal event( ):** The jobs or processes signals when an event has occurred.

· **get process attributes( )** and **set process attributes( )** are used to control execution of a process.

· **acquire lock( ) or release lock( ):** Acquire lock ( ) is used to lock the process and release lock( ) is used to release the lock on process.

If the program runs into a problem or terminates abnormally and causes an error trap, a dump of memory is taken and an error message generated. The dump is written to disk and may be examined by a **debugger.** A debugger is a system program designed to aid the programmer in finding and correcting errors, or **bugs** to determine the cause of the problem.

File Management

· create( ) is used to create a new file and delete( ) is used to delete an existing file. · Every file have some attributes includes file name, file type, protection codes, accounting information and so on. get file attributes( ) and set file attributes( ) are the system calls used to retrieve and get values of attributes.

· open( ) is used to open already created file.

· After opening a file, it may be read( ), write( ), or reposition( ).

· Close( ): After all the operations are performed with the file, the file needs to be close.

Device Management

· A process may need several resources to execute such as main memory, disk drives, access to files and so on.

· If the resources are available, they can be granted and control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available. · The various resources controlled by the operating system such as disk drives, files etc. · A system with multiple users may require us to first request( ) a device to use it. · Once the device has been requested, we can read( ), write( ) and reposition( ) the device. · After we are finished with the device, we release( ) it.

Information Maintenance

Many system calls exist simply for the purpose of transferring information between the user program and the operating system.

· Most systems have a system call to return the current time( ) and date( ). ·

dump( ) system calls is helpful in debugging a program.

· Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space and so on.

Communication

There are two common models of inter-process communication: Message passing model and the Shared-memory model.

In **Message-passing model**, the communicating processes exchange messages with one another to transfer information by establishing an internet connection.

· Messages can be exchanged between the processes either directly or indirectly through a common mailbox.

· Each computer in a network has a **host name and** each process has a **process name.**

· OS can refer to the process by translating Process name into process identifier. · **gethostid( )** and **getprocessid( )** system calls used to retrieve host name and process id. · open_connection( ) and close_connection( ) system calls used to opening and closing the connection.

· accept_connection( ) is used to accept the connection by the receiptent. In S**hared-memory** model, processes use shared memory create( ) and shared memory attach( ) system calls to create and gain access to regions of memory owned by other processes.

· Operating system tries to prevent one process from accessing another processes memory. · Shared memory requires that two or more processes agree to remove this restriction. · They can exchange information by reading and writing data in the shared areas.

Protection

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

· System calls providing protection include set_permission( ) and get_permission( ), which manipulate the permission settings of resources such as files and disks. · The allow_user( ) and deny_user( ) system calls specify whether particular users can or cannot be allowed access to certain resources.

SYSTEM PROGRAMS

System programs are also known as **system utilities** provide a convenient environment for program development and execution.

System programs can be divided into following categories:

· **File management**. These programs create, delete, copy, rename, print, dump, list and generally manipulate files and directories.

· **Status information** System programs ask the system for the date, time and amount of available memory or disk space, number of users, detailed performance, logging and debugging information.
Typically, these programs format and print the output to the terminal or other output devices or files or display it in a window of the GUI.

· **File modification**. Several text editors are available to create, modify and search the content of files stored on disk or other storage devices.

· **Programming-language support**: Compilers, assemblers, debuggers and interpreters for common programming languages such as C, C++ and Java are often provided with the operating system.

· **Program loading and Execution**: Once a program is assembled or compiled, it must be loaded into memory to be executed. The system provides loaders, linkage editors and overlay loaders.

· **Communication**: These programs provide the mechanism for creating virtual connections among processes, users and computer systems.
They allow users to send messages to one another's screens, to browse Web pages, to send e-mail messages, to log in remotely or to transfer files from one machine to another.

· **Background services**. All general-purpose systems have methods for launching certain system-program processes at boot time. Some of these processes terminate after completing their tasks, while others continue to run until the system is halted. Constantly running system-program processes are known as **services**, **subsystems** or **daemons.**

OPERATING-SYSTEM DESIGN AND IMPLEMENTATION

Design Goals

1. **Choice of Hardware:** The design of the system will be affected by the choice of hardware and the type of system such as batch, time sharing, single user, multiuser, distributed, real time or general purpose.

2. **Requirements Specification:** The requirements may be much harder to specify. · The requirements can be divided into two groups: **User goals** and **System goals**. · User's wants system should be convenient to use, easy to learn, reliable, safe and fast. · The system should be easy to design, implement and maintain. System should be flexible, reliable, error free and

efficient.

Mechanisms and Policies

Mechanisms determine ***how*** to do something; policies determine ***what*** will be done. Example: The timer construct is a mechanism for ensuring CPU protection, but deciding how long the timer is to be set for a particular user is a policy decision.

· The separation of policy and mechanism is important for flexibility. · Policies are likely to change across places or over time. In the worst case, each change in policy would require a change in the underlying mechanism.

· A change in policy would require redefinition of certain parameters of the system. Example: Consider a mechanism for giving priority to certain types of programs over others. If the mechanism is properly separated from policy, it can be used either to support a policy decision that I/O-intensive programs should have priority over CPU-intensive programs.

Implementation

Operating systems are collections of many programs, written by many people over a long period of time. Once an operating system is designed, it must be implemented. · Operating systems can be written in assembly languages and Higher level languages such as C, C++, Python, PERL, Shell scripts.

· An operating system can be written in more than one language.

· The lowest levels of the kernel might be assembly language. Higher-level routines can be written in C and system programs might be in C or C++, in interpreted scripting languages like PERL or Python, or in shell scripts.

Example: A Linux distribution probably includes programs written in all of those languages. · Advantages of using a higher-level language for implementing operating systems are the code can be written faster and is easier to understand and debug.

· Only disadvantage of implementing an operating system in a higher-level language are reduced speed and increased storage requirements. But compared to today's processor speed it is a minor issue.

OPERATING-SYSTEM STRUCTURE

Multi-Programming System

· The most important aspects of operating systems is the ability to multi-program. · A single program cannot keep either the CPU or the I/O devices busy at all times. Single users frequently have multiple programs running.

· Multiprogramming increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute.

· The operating system keeps several jobs in memory simultaneously. Since main memory is too small to accommodate all jobs, the jobs are kept initially on the disk in the **job pool**. This job pool consists of all processes residing on disk awaiting allocation of main memory.

· The set of jobs in memory can be a subset of the jobs kept in the job pool. The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete.

· In a non-multi programmed system, the CPU would sit idle.

· In a multi-programmed system, the operating system simply switches to, and executes, another job. When *that* job needs to wait, the CPU switches to *another* job, and so on. Eventually, the first job finishes waiting and gets the CPU back. As long as at least one job needs to execute, the CPU is never idle.

· Multi-programmed systems provide an environment in which the various system resources such as CPU, memory, and peripheral devices are utilized effectively, but they do not provide for user interaction with the computer system.

Time-Sharing or Multi-tasking System

· Time sharing is a logical extension of multiprogramming. In time-sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.

· Time sharing requires an **interactive** computer system, which provides direct communication between the user and the system. The user gives instructions to the operating system using a input device such as a keyboard, mouse, touch pad, or touch screen, and waits for immediate results on an output device. Accordingly, the **response time** should be less than one second.

· A time-shared operating system allows many users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user.

· As the system switches rapidly from one user to the next, each user is given the impression that the entire computer system is dedicated to his use, even though it is being shared among many users.

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 few people and started as small, simple and limited systems and then grew beyond their 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.

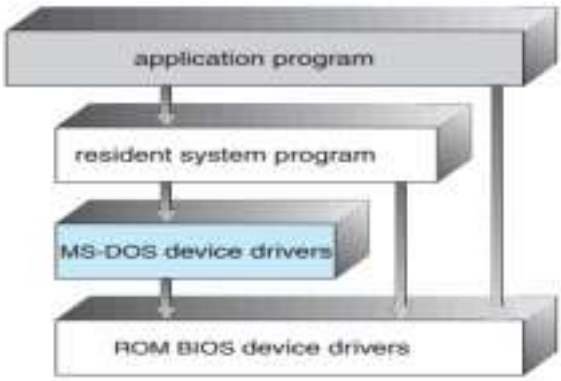


Figure 2.11 MS-DOS layer structure.

Original UNIX operating system is also have limited structuring, it was limited by its hardware functionality.

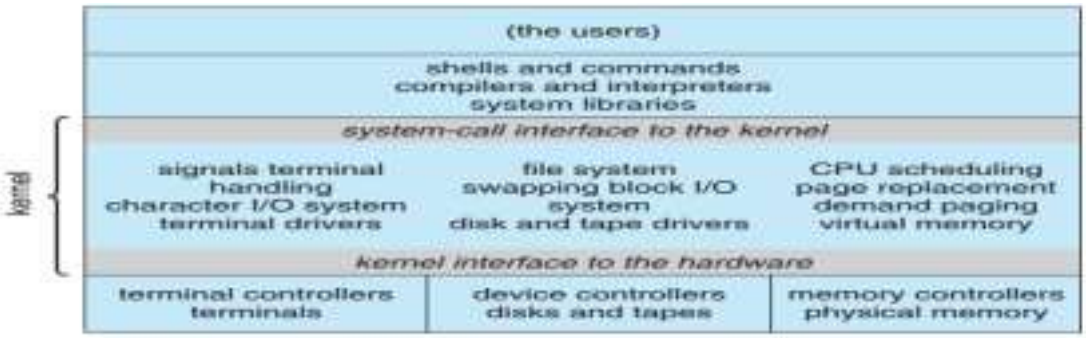


Figure 2.12 Traditional UNIX system structure.

· It consists of two separable parts: **Kernel** and **System programs**.

· The kernel is 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 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 to be combined into one level.

· This monolithic structure was difficult to implement and maintain.

Layered Approach

· In 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 User interface. · A 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 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.

Other advantage of 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 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. Other 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 to 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 is 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 a smaller kernel (i.e.) Microkernel.

· There is little consensus regarding which services should remain in the kernel and which service should be implemented in user space.

· Microkernels provide minimal process and memory management and also communication facility.

· 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 user space.

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 is 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 when 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.

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:

Hybrid Systems

In practice, most of the 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 microkernel. Windows systems also provide support for dynamically loadable kernel modules. Operating Systems that supports 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 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 micro kernel 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 development of device drivers and dynamically loadable

modules.
· The BSD application environment can make use of BSD facilities directly.

iOS

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

· 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.